КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

До захисту допущено

Завідувач кафедри ІСТ

_____ Олександр КУЧАНСЬКИЙ

(підпис) (ім'я, ПРІЗВИЩЕ)

"___"_____2022р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

спеціальності 126 «Інформаційні системи та технології»

освітньої програми «Програмні технології інтернет речей»

на тему: «IoT система мінерального живлення рослин»

Виконав (-ла): студент (-ка) 4 курсу, групи IP-41

(шифр групи)

Максим ЯМКОВЕНКО _____

(Ім'я, ПРІЗВИЩЕ) (підпис)

Керівник к.ф.-м.н., доцент Іван ЧИЧКАНЬ _____

(посада, науковий ступінь, вчене звання, Ім'я, ПРІЗВИЩЕ) (підпис)

Консультант нормо контроль к.т.н. Ростислав ЛІСНЕВСЬКИЙ _____

(назва розділу) (посада, вчене звання, науковий ступінь, Ім'я, ПРІЗВИЩЕ) (підпис)

Рецензент к.ф-м.н., доцент Світлана СПАСІТЄЛЄВА _____

(посада, науковий ступінь, вчене звання, науковий ступінь, Ім'я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у пояснювальна записка немає запозичень з праць інших авторів без відповідних посилань.

Здобувач освіти _____

(підпис)

Київ – 2022 року

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра Інформаційні системи та технології
Освітній рівень Бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітня програма Програмні технології інтернет речей

ЗАТВЕРДЖУЮ
Завідувач кафедри ІСТ,
д.т.н., доцент
Олександр КУЧАНСЬКИЙ
_____
«__» _____ 2022 року

ЗАВДАННЯ
НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Здобувач освіти: Максим ЯМКОВЕНКО

Група: ІР-41

1. **Тема кваліфікаційної роботи бакалавра**: «IoT система мінерального живлення рослин».

Затверджена протоколом засідання кафедри ІСТ №05/21_22 від 03.12.2021 року.
2. **Строк подання студентом готової роботи** – «22» червня 2022 р.

3. **Вихідні дані до роботи**: дослідження в області моніторингу та контролю за мінеральним живленням рослин. Програмні рішення для IoT системи мінерального живлення рослин. Дані з показниками температури та вологості повітря, освітленості у приміщенні, кислотності й електропровідності розчину, отримані з датчиків, аналіз та візуалізація отриманих показників.

4. **Зміст роботи:** РОЗДІЛ 1. АНАЛІЗ ВИМОГ ДО СИСТЕМИ, ЩО ДОСЛІДЖУЄТЬСЯ (аналіз сфери гідропоніки, умови вирощування полуниці гідропонікою, огляд існуючих рішень), РОЗДІЛ 2. РОЗРОБКА ПРОЄКТУ (аналіз і підбір обладнання для створення проєкту, створення логічної і фізичної бази даних для системи, опис засобів реалізації), РОЗДІЛ 3. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ(інтерфейс програми користувача: дашборд,

приміщення теплиці, каталог сортів, прилади; інтерфейс програми адміністратора, налаштування електронних листів IoT теплиці, візуалізація отриманих даних, реагування системи на недопустимі значення показників).

5. **Перелік графічного матеріалу**: Складові компоненти Raspberry Pi 4, концептуальна модель архітектури системи мінерального живлення рослин, логічна модель бази даних, фізична модель бази даних, архітектура MVT Django, загальна структура картки Bootstrap, графічний інтерфейс веб-застосунку, алгоритм надсилання електронних листів.

6. **Календарний план виконання роботи:**

Етапи виконання кваліфікаційної роботи бакалавра	**Термін виконання**	**Результат виконання**
1. Вибір тематики кваліфікаційної роботи бакалавра	01.09.2021-01.10.2021	виконано
2. Наказ про затвердження тем кваліфікаційної роботи бакалавра та призначення керівників	03.12.2021	виконано
3. Розробка плану кваліфікаційної роботи бакалавра і його погодження з керівником	25.12.2021	виконано
4. Написання I розділу кваліфікаційної роботи	19.03.2022	виконано
5. Написання II розділу кваліфікаційної роботи	25.04.2022	виконано
6. Написання III розділу кваліфікаційної роботи	29.04.2022	виконано
8. Підготовка висновків і пропозицій	30.04.2022	виконано
9. Попередній захист кваліфікаційної роботи	12.05.2022	виконано
10. Перевірка на плагіат	13.05.2022-15.06.2022	виконано
11. Нормо контроль	02.06.2022-06.06.2022	виконано
12. Рецензування кваліфікаційної роботи бакалавра і представлення роботи на кафедру в друкованому вигляді	15.06.2022	виконано
13. Захист кваліфікаційної роботи бакалавра	23.06.2021-24.06.2021	

Дата видачі завдання «___» _____2022 р.

Керівник роботи: к.ф.-м.н., доц. Іван ЧИЧКАНЬ _____ (підпис)

Завдання прийняв до виконання:

Здобувач освіти на освітньому рівні «бакалавр» 4-го курсу групи ІР-41

Максим ЯМКОВЕНКО _____

(Власне Ім’я, ПРІЗВИЩЕ) (підпис)

АНОТАЦІЯ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра Інформаційних систем та технологій

Освітня програма «Програмні технології інтернет речей»

Кваліфікаційна робота бакалавра Максима ЯМКОВЕНКА

Тема роботи: «IoT система мінерального живлення рослин».

Мета кваліфікаційної роботи бакалавра – створення IoT системи мінерального живлення рослин, дослідження обладнання та програмних засобів реалізації, які використовуються при створенні гідропонної системи, а також створення додатку для моніторингу та контролю за робою системи.

Об’єкт дослідження – IoT система мінерального живлення рослин.

Предмет дослідження – обробка, аналіз та візуалізація даних отриманих з датчиків температури та вологості повітря, освітленості у приміщенні, кислотності й електропровідності розчину.

Кваліфікаційна робота бакалавра складається зі змісту, вступу, основної частини, яка включає три розділи, висновків та списку використаних джерел. Всього 72 сторінок.

КЛЮЧОВІ СЛОВА: IoT, гідропоніка, система мінерального живлення рослин, ООП, Django, Python, SQLite, Raspberry Pi.

ABSTRACT

TARAS SHEVCHENKO NATIONAL UNIVERSITY OF KYIV

Faculty of Information Technologies

Department of Information Systems and Technologies

Educational Program "Software Technologies of the Internet of Things"

Qualification work of master Maksym YAMKOVENKO.

Work topic: "IoT system of mineral plant nutrition".

The purpose of the bachelor's qualification work is to create an IoT plant mineral nutrition system, research equipment and software used in the creation of a hydroponic system, as well as create an application for monitoring and control of the system.

The object of study is IoT system of mineral nutrition of plants.

The subject of research is processing, analysis and visualization of data obtained from sensors of temperature and humidity, indoor lighting, acidity and electrical conductivity of the solution.

The bachelor's qualification work consists of the content, introduction, main part, which includes three sections, conclusions and a list of sources used. Total 72 pages.

KEY WORDS: IoT, hydroponics, plant mineral nutrition system, OOP, Django, Python, SQLite, Raspberry Pi.

ЗМІСТ

ВСТУП 7
РОЗДІЛ 1. АНАЛІЗ ВИМОГ ДО СИСТЕМИ, ЩО ДОСЛІДЖУЄТЬСЯ 9
1.1. Аналіз сфери гідропоніки 9
1.2. Умови вирощування полуниці гідропонікою 12
1.3. Огляд існуючих рішень 13
1.4. Висновок до першого розділу 16
РОЗДІЛ 2. РОЗРОБКА ПРОЄКТУ 17
2.1. Аналіз і підбір обладнання для створення проєкту 17
2.2. Створення логічної і фізичної бази даних для системи 23
2.3. Опис програмних засобів реалізації 29
2.4. Висновок до другого розділу 34
РОЗДІЛ 3. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 35
3.1. Інтерфейс програми користувача 35
3.1.1. Дашборд 35
3.1.2. Приміщення теплиці 38
3.1.3. Каталог сортів 47
3.1.4. Прилади 48
3.2. Інтерфейс програми адміністратора 49
3.3. Налаштування електронних листів IoT теплиці 56
3.4. Візуалізація отриманих даних 60
3.5. Реагування системи на недопустимі значення показників 63
3.6. Висновок до третього розділу 66
ВИСНОВОК 67
ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ 68
ДОДАТОК А 73
ДОДАТОК Б 82

ВСТУП

У традиційному методі вирощування рослин на ґрунті, фермерам необхідний ґрунт високої якості з природними мінеральними властивостями. Це також вимагає витрат на видалення бур'янів, боротьбу із шкідниками та догляд за землею. Більше того, зміна клімату є серйозною проблемою у вирощуванні рослин, адже це явище може спричинити сезонну неврожайність. Нинішнє землеробство залежить від багатьох неконтрольованих факторів, саме тому необхідне запровадження IoT системи мінерального живлення рослин. Гідропоніка – метод вирощування рослин у приміщенні без використання ґрунту. Замість ґрунту використовується вода, що надходить безпосередньо до коренів рослини. Вода змішується з мінеральними поживними речовинами. Знаючи необхідні мікроелементи для вирощування рослин, можна отримати врожай від 3 до 10 разів більший з тієї ж самої площі. Цей метод вирощування рослин контролюється завдяки показникам кислотності та електропровідності рідини. Вирощуючи рослини у приміщенні, можна підібрати ідеальні мікрокліматичні умови, тим самим уникнути проблем з непередбачуваними погодними умовами. Інтегрувавши Інтернет речей (IoT) в систему гідропоніки, фермерам відкриваються нові можливості в постійному моніторингу та контролю за показниками температури та вологості повітря, освітленості у приміщенні, кислотності й електропровідності розчину. Таким чином заощаджуються водні ресурси, контролюються мікроклімат та показники, що відповідають за мінеральне живлення рослин.

Об'єкт дослідження: IoT система мінерального живлення рослин.

Метою цієї роботи є: створення IoT системи мінерального живлення рослин, а також дослідження обладнання та програмних засобів реалізації, які використовуються при створенні гідропонної системи, а також створення додатку для моніторингу та контролю за робою системи.

Для досягнення мети роботи необхідно:

- провести аналіз сфери гідропоніки (переваги, недоліки) та дослідити умови вирощування полуниці гідропонікою;

- дослідити на проаналізувати існуючі системи мінерального живлення рослин;
- обрати обладнання для реалізації апаратної частини;
- розробити програмне забезпечення IoT системи мінерального живлення рослин.

Результати: спроєктовано IoT систему мінерального живлення рослин.

Практичне значення одержаних результатів: реалізовано веб-додаток, який складається з двох частин: інтерфейсу користувача та адміністратора. Інтерфейс користувача дозволяє переглядати дані отримані з датчиків і реле та інформацію про висаджені рослини на ділянках. За допомогою інтерфейсу адміністратора виконується управління базою даних. Також виконується аналіз отриманих даних з датчиків та у разі виникнення позаштатної ситуації, користувачеві надсилаються електронні листи, а також реагування системи на недопустимі значення показників.

РОЗДІЛ 1. АНАЛІЗ ВИМОГ ДО СИСТЕМИ, ЩО ДОСЛІДЖУЄТЬСЯ

1.1. Аналіз сфери гідропоніки

Гідропоніка – це метод вирощування рослин у приміщенні без використання ґрунту. Замість того, щоб витягувати з землі мінеральні поживні речовини, необхідні для росту, рослини отримують все своє харчування через живильний розчин, який надходить до їх коренів [1]. Розчини можуть містити азот, фосфор, калій, кальцій та багато інших мікроелементів, залежно від сорту вирощуваних рослин.

Насамперед рослини ростуть завдяки наступним елементам: вода, поживні речовини та сонячне світло. Система гідропоніки усуває потребу в ґрунті, забезпечуючи живильним водним розчином безпосередньо коріння, що забезпечує живлення та зволоження рослини.

У порівнянні з традиційним вирощуванням рослин на ґрунті гідропоніка має наступні переваги:

- Продовжений вегетаційний період. За допомогою гідропоніки рослини можна вирощувати цілий рік, оскільки фермер має контроль над кліматичними умовами та поживними речовинами, що надходять до рослини.
- Збільшення врожайності. Обсяги виробництва збільшуються від 3 до 10 разів на тій самій площі [2].
- Менше споживання води. Гідропонні системи використовують від 80% до 90% менше води, ніж рослини, вирощені в землі. У традиційному садівництві велику кількість води вносять у ґрунт, щоб забезпечити достатню кількість вологи до кореневої зони. При переміщенні води через ґрунт, вона випаровується і лише її частина досягає коріння або взагалі не може до них досягти. У гідропоніці вода відразу досягає коріння, мало втрачаючи при випаровуванні. В таких системах поживний розчин

циркулює кілька разів, перш ніж він стане непридатним, що ще більше збільшує ефективність використання води.

- Менше проблем зі шкідниками. Оскільки гідропонні системи знаходяться в закритих приміщеннях, шкідникам важче проникнути в систему. Крім того, це означає майже відсутню потребу в пестицидах.
- Внутрішнє землеробство в клімат-контрольованому середовищі означає, що ферми можуть існувати в місцях, де погодні та ґрунтові умови не є сприятливими для традиційного вирощування рослин.
- Легший збір рослин. Рослини, вирощені в гідропонних системах, зазвичай вирощують на підставках, столах тощо, що ставить їх на висоту талії для більшості фермерів. На такій висоті плоди легше збирати, оскільки не потрібно нахилятися або ставати на коліна, щоб дістатися до рослин [1].

Гідропоніка може стати більш актуальною через притаманні їй рішення проблем з водою, землею, нестачею робочої сили, водночас безпечною для навколишнього середовища та технологій. Попри всі свої переваги, гідропоніка має ряд недоліків:

- Вартісне встановлення та налаштування. Вартість варіюється в залежності від типу та розмірів системи, а також чи вона є збірною, чи створена під індивідуальне замовлення.
- Чутливість до відключення електроенергії. Система залежить від електрики для живлення різних компонентів, таких як освітлювальні прилади, водяні насоси, датчики тощо [1].
- Хвороби, що передаються через воду. Оскільки вода безперервно циркулює по системі, інфекції можуть швидко поширюватися, вражаючи всі рослин.
- Не маючи ґрунту як буфера, рослини, вирощені в гідропонних системах, набагато швидше реагують на такі проблеми, як дефіцит поживних речовин і хвороби.

Незважаючи на свої недоліки, гідропонні ферми пропонують шлях, який надає пріоритет здоров’ю їжі та навколишньому середовищу без інтенсивного

використання хімікатів. Крім того, ферми можуть існувати в місцях, де кліматичні та ґрунтові умови не є сприятливими для традиційного вирощування рослин.

Для успішного росту рослин методом гідропонного вирощування мають контролюватись наступні показники: кислотність та електропровідність розчину, температура та вологість повітря й освітленість у приміщенні.

Для контролю та моніторингу даних умов для вирощування можна використати концепцію Інтернету речей.

Інтернет речей (Internet of Things, IoT) – це мережа фізичних об’єктів, оснащених датчиками, програмним забезпеченням та іншими технологіями. Підключені до Інтернету, ці «речі» можуть обмінюватися даними в реальному часі з іншими підключеними пристроями та системами через мережі. Підключені пристрої поєднуються з автоматизованими системами для збору даних IoT, які можна проаналізувати, щоб допомогти у виконанні завдань або дізнатися, як покращити процес [4].

Використання концепції Internet of Things у вирощуванні рослин має наступні переваги:

- Сільське господарство з підтримкою IoT дозволяє фермерам контролювати свою продукцію та умови вирощування рослин в режимі реального часу. Фермери можуть візуалізувати рівень виробництва, вологість ґрунту, інтенсивність сонячного світла тощо в реальному часі та віддалено, щоб прискорити процес прийняття рішень [5].
- Рішення IoT у сільському господарстві впроваджують автоматизацію, наприклад, клімат-контроль або внесення добрив.
- Охорона води – прогноз погоди та датчики вологості ґрунту дозволяють використовувати воду лише тоді й там, де це необхідно.
- Оптимізація використання ресурсів – води, енергії.
- Сільське господарство стає більш екологічним, значно зменшується використання пестицидів. Також зменшення використання води та

збільшення виробництва на одиницю землі, позитивно впливають на екологічний слід [6].

Поєднуючи переваги гідропоніки та IoT можна створити IoT систему мінерального живлення рослин, яка б виконувала моніторинг та контроль за умовами вирощування рослин, тим самим реалізуючи автоматизовану концепцію гідропоніки.

1.2. Умови вирощування полуниці гідропонікою

Правильне розуміння фізіології полуниці та використання гідропоніки, може значно збільшує врожайність. Фермери можуть вирощувати плоди полуниці правильного кольору, розміру та форми. Однак не менш важливим показником якості полуниці – смак. Він залежить від відносно невеликої маси сухої речовини. У середньому лише 8% сухої речовини полуниці впливають на її смак [7]. Солодкість полуниці залежить від розчиненої глюкози та фруктози. Невеликі відмінності в концентрації цих цукрів можуть повністю змінити смак ягоди. Солодкість фруктів вимірюється в одиницях, які називаються Brix. Градуси Брікса (°Bx) вимірюють загальну кількість цукру у фруктовому соку порівняно із вмістом води у фрукті. Показник розраховується за формулою(1.1):

$$Bx = \frac{Ds*100}{Ds+w}, \qquad (1.1)$$

де Ds – вага розчинених твердих речовин (фруктоза, глюкоза та сахароза);

w – маса води.

Для полуниці °Bx 12 – нормально, °Bx 14 – добре, а °Bx – 16 – чудово [6].

Для збільшення солодкість полуниці, фермерам необхідно розташувати джерела світла так, щоб ягода повністю була на світлі. Таке розташування полуниці дозволяє ягоді виробляти більше природних цукрів.

Невеликі зміни в гідропонному середовищі можуть суттєво змінити смак полуниці. Для максимізації солодкості плоду, необхідно забезпечувати потреби ягід у різних поживних речовин на різних етапах розвитку.

На ранніх стадіях зростання полуниця потребує мінеральне середовище з більшим вмістом азоту. Для поглинання корінням рослини більшої кількості азоту, середовище вирощування повинне мати нижчу ЕС. ЕС – електропровідність, вимірюється в $mScm^{-1}$, міліСіменс на сантиметр. ЕС є мірою кількості поживних речовин у живильному розчині. ЕС-датчик вимірює потенціал електричного струму, який переноситься через воду. Азот легше поглинається з мінерального розчину, який має нижчий ЕС, зазвичай близько 1.2. Щоб максимізувати солодкість полуниці на етапі коли вона зав’язує плоди, необхідно додати більше калію до гідропонного розчину. Щоб рослина засвоїла оптимальну кількість калію, показник ЕС має бути від 2 до 3.2. Якщо рослини отримують максимальне сонячне світло, то значення ЕС може бути ближче до 2, а якщо коли рослини знаходяться в тіні, то ближче до 3.

Показники ЕС мають значення лише тоді, коли мінеральне середовище має постійний pH. pH є мірою того, наскільки кислим або лужним є розчин, який контролюється активністю катіонів водню. Водневий показник розчину не повинен перевищувати 5.8 – 6.2.

Також необхідно забезпечити правильне освітлення, вологість та температурні умови. Температура в приміщенні має бути від +16 до +20 °C. Полуниця потребує постійного світла. З точки зору освітлення, полуниця має отримувати 8 – 12 годин світла на день. Необхідний рівень освітленість становить від 10000 до 20000 Люкс. Не менш важливою є вологість повітря. Вона має бути 60 – 70%.

1.3. Огляд існуючих рішень

iFarm Growtune – Control Centre of your vertical farm

SaaS-платформа iFarm Growtune (рис. 1.1) – це система автоматизованого управління вертикальними фермами, яка дозволяє отримувати якісний урожай у прогнозовані терміни та максимально знижувати собівартість продукції.

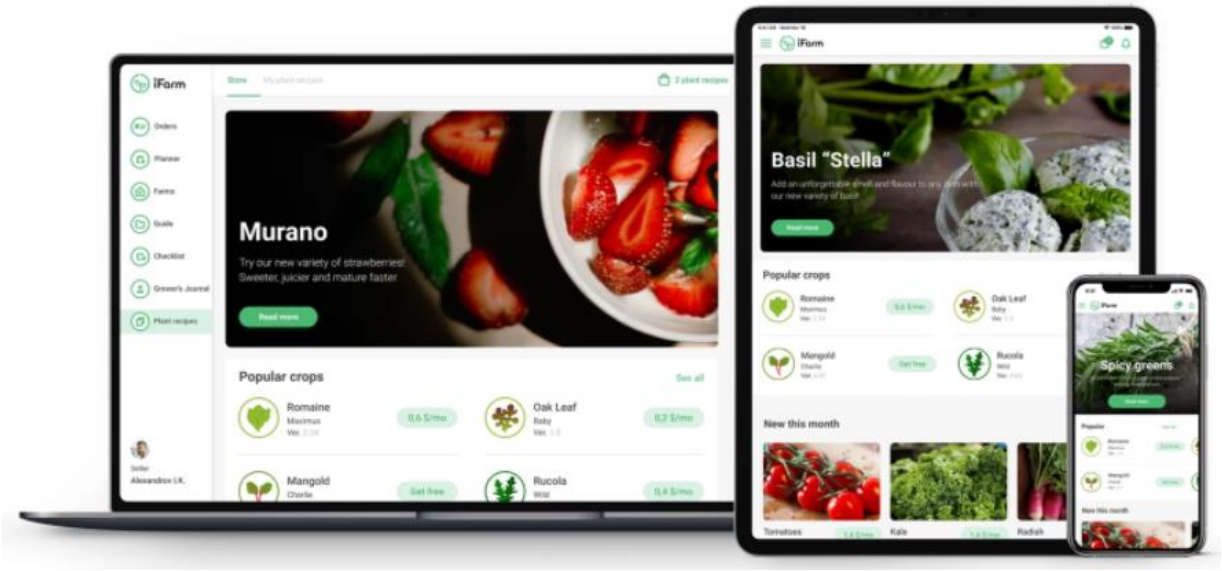

Рисунок 1.1. Інтерфейс користувача додатку iFarm Growtune [8]

Хмарне рішення безпосередньо взаємодіє з усіма датчиками та контролерами, управляє параметрами температури, вологості, CO2, складом розчину та графіком поливу, розкладом включення та відключення світла. Також додатком надається контроль стану мікроклімату приміщень із сервісом екстрених повідомлень та автоматично згенеровані завдання з чітким переліком дій з догляду за посадками та технічним обслуговуванням ферм [8].

Agcetra – Urban Farming for everyone

Agcetra призначене для задоволення потреб сучасних міських методів сільського господарства, таких як гідропоніка, аквапоніка, аеропоніка та інші.

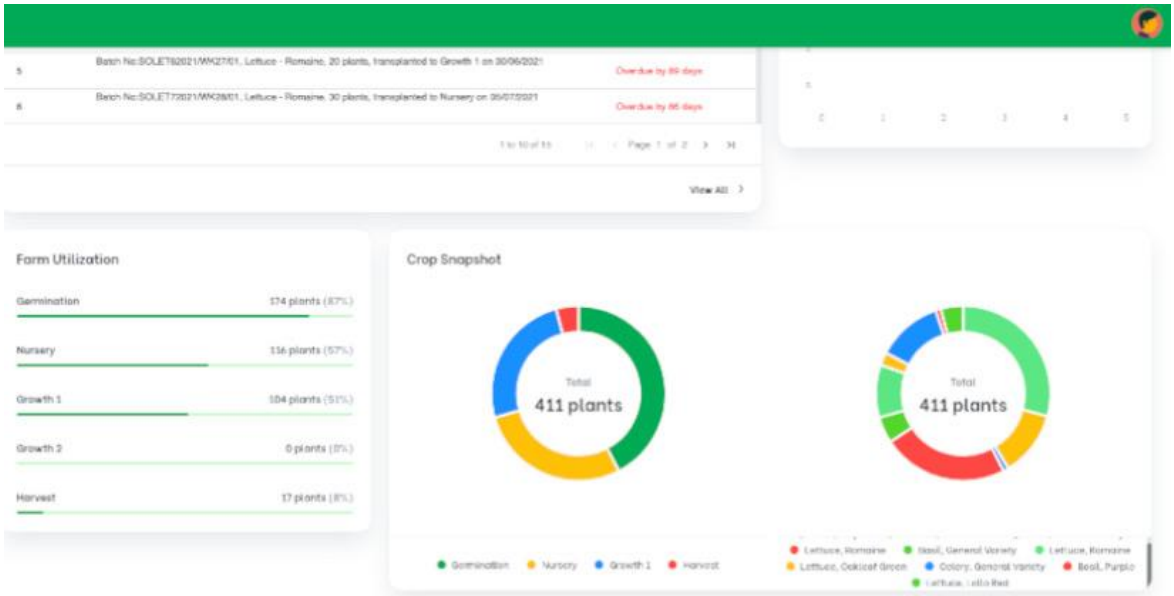

Рисунок 1.2. Інтерфейс користувача додатку Agcetra [9]

Дане програмне забезпечення надає користувачеві наступні функції для використання (рис. 1.2): відстеження циклу росту висадженої рослини, моніторинг у режимі реального часу та управління поживними речовинами. Agcetra має управління та планування циклу врожаю [9].

Agrowtek

Функції віддаленого доступу Agrowtek полегшують ведення сільського господарства, дозволяючи налаштовувати все у своїй фермі з телефону або комп’ютера.

Використовуючи Agrowtek можна переглянути показники в режимі реального часу або швидко змінити налаштування клімату, рівень поживних речовин або графік освітлення. Функції моніторингу (рис. 1.3) дозволяють озирнутися на минулі кліматичні умови, допомагаючи забезпечити оптимальну роботу ферми.

Рисунок 1.3. Інтерфейс користувача додатку Agrowtek [10]

Температура повітря, вологість, рівень поживних речовин і pH води регулюються автоматично, щоб ферма завжди підтримувала ідеальні умови вирощування [9].

1.4. Висновок до першого розділу

Отже, в даному розділі здійснено аналітичний огляд концепції гідропоніки та реалізованих систем IoT в теплицях.

- Проаналізовано умови вирощування полуниці за допомогою гідропоніки; досліджено переваги та недоліки використання гідропоніки.
- Виконано порівняння існуючих проектів в даній сфері – програмне забезпечення для моніторингу та контролю за показниками теплиці.
- Зроблено висновки щодо необхідності створення IoT системи мінерального живлення рослин, яка б виконувала моніторинг та контроль за умовами вирощування рослин, тим самим реалізуючи автоматизовану концепцію гідропоніки.

В результаті проведених досліджень виявлено, що основними причинами впровадження гідропоніки є збільшення врожайності, менше використання води та контроль над вегетаційним періодом рослини.

РОЗДІЛ 2. РОЗРОБКА ПРОЄКТУ

2.1. Аналіз і підбір обладнання для створення проєкту

В основі даної автоматизованої системи мінерального живлення рослин лежить одноплатний комп'ютер Raspberry Pi 4. Пристрій невеликого розміру, однак має всі компоненти повноцінного комп'ютера (рис. 2.1) такі як: процесор, оперативна пам'ять, відео/аудіо вихід, USB, WiFi, Bluetooth, Ethernet та micro SD.

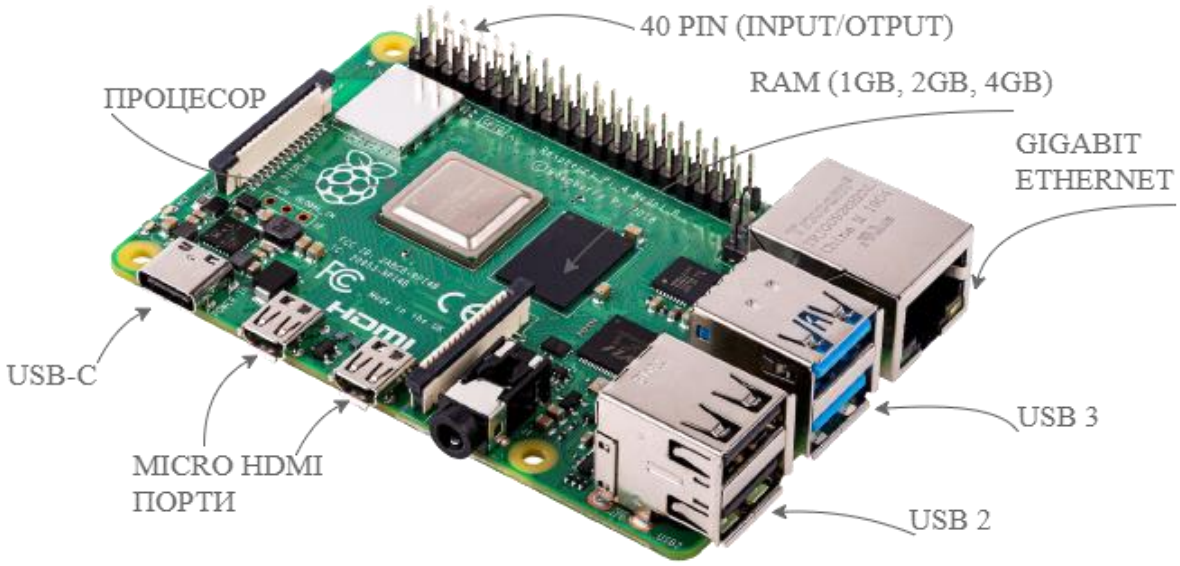

Рисунок 2.1. Складові компоненти Raspberry Pi 4

Основні характеристики Raspberry Pi 4 зображено в наступній табл. 2.1:

Таблиця 2.1 – **Основні характеристики Raspberry Pi 4 [10]**

Характеристика	**Опис**
1	**2**
Система на кристалі (SoC)	Broadcom BCM2711 (CPU + GPU)
Процесор	64-бітний чотириядерний ARMv8 Cortex-A72 процесор із тактовою частотою 1.5 ГГц
Графічний процесор	VideoCore VI GPU підтримує стандарти OpenGL ES 3.0; кодування H.264 до 1080p30, декодування 4Kp60 відео (H.265 до 4Kp60, H.264 до 1080p60)

Продовження табл 2.1

Характеристика	Опис
1	**2**
ОЗУ	½/4/8 ГБ SDRAM LPDDR4
Сховище	слот для картки пам’яті MicroSD
Ethernet	10/100/1000 Мбіт Gigabit Ethernet (контролер Broadcom BCM54213PE)
WiFi/Bluetooth	2.4 ГГц та 5 ГГц IEEE 802.11.b/g/n/ac WI-FI та Bluetooth 5.0 Low Energy (BLE), що забезпечуються мікросхемою Cypress CYW43455
Відео вхід	1 x CSI-2 для підключення камери за інтерфейсом MIPI
Відео вихід	2x micro-HDMI 1 x DSI (Display Serial Interface) для підключення штатного дисплея; 1x композитний відеовихід (CVBS відео, PAL та NTSC) 3.5 мм роз’єм
Аудіо вхід	Ні, але можна додати USB-мікрофон або звукову карту
Аудіо вихід	гніздо 3.5 мм, 2 порти micro-HDMI
USB-порти	2 порти USB 2.0 та 2 порти USB 3.0 через VLI VL805
Периферія	40 портів введення-виведення загального призначення (GPIO), UART (Serial), I²C/TWI, SPI з селектором між двома пристроями; піни живлення: 3,3 В, 5 В та земля.
Живлення	5, 3.0 А через порт USB type C або GPIO; Power over Ethernet (PoE) через окремий PoE HAT (окремі 4 піна)
Модуль керування живленням (PMIC)	MxL7704

Продовження табл 2.1

Характеристика	Опис
1	2
Розміри	85.6 мм x 56.5 мм x 17 мм
Вага	45 г
ОС	Ubuntu, Debian, Fedora, Arch Linux, Gentoo, RISC OS, Android, Firefox OS, NetBSD, FreeBSD, Slackware, Tiny Core Linux, Windows 10 IOT

Raspberry Pi 4 має роз'єм GPIO з 40 контактами. GPIO – інтерфейс введення/виведення загального призначення, тобто інтерфейс для зв'язку між компонентами комп'ютерної системи, наприклад, мікропроцесором та різними периферійними пристроями. Контакти GPIO можуть діяти як входи та виходи, і це можна налаштовувати [11]. В даній системі GPIO будуть використовуватись для підключення необхідних датчиків.

Для реалізації системи мінерального живлення рослин можуть використовуватись наступні датчики (табл 2.2):

Таблиця 2.2 – Список датчиків для реалізації системи

Назва	Фото	Застосування
1	2	3
Датчик температури DS18B20		Waterproof DS18B20 Digital Temperature Sensor – захищений від вологи цифровий датчик температури на основі DS18B20 у захищеному корпусі (металева гільза). Необхідний для вимірювання температури у вологому середовищі, у важкодоступних місцях, у будівлях та механізмах, а також в

Продовження табл 2.2

Назва	Фото	Застосування
1	2	3
		управлінні технологічними процесами. Сумісність з напругою живлення та рівнями сигналів: 3, 5.5 В, точність ±0.5°C
Датчик вологості DHT22		Цифровий датчик вологості з підвищеною точністю. Особливість DHT22 заводське калібрування та низьке енергоспоживання. Діапазон вимірювання вологості: 0 – 100%. Робоча напруга: 3.3, 5 В [13].
Датчик освітленості TSL25911FN		Цей модуль являє собою датчик зовнішнього освітлення з ядром TSL25911. Він може вимірювати інтенсивність навколишнього освітлення. Вихідні дані використовуються для обчислення освітленості в Люксах (інтенсивність навколишнього світла), щоб приблизно наблизити реакцію людського ока. Ефективний діапазон: 0 – 88000 Люкс. Робоча напруга: 3.3, 5 В [14].
Датчик кислотності рідини pH		Датчик pH використовується для визначення рівня кислотності рідини. Датчик допоможе контролювати комфортне середовище для вирощування рослин. Перед початком вимірювання електрод потребує калібрування стандартним буферним розчином з

Продовження табл 2.2

Назва	**Фото**	**Застосування**
1	**2**	**3**
		відомим значенням pH. Робоча напруга: 5 В [15].
Датчик електропровідності рідини EC		Датчик електричної провідності (EC датчик) вимірює електропровідність у розчині. Цей комплект включає EC-зонд і плату драйвера, яка підтримує робочу напругу 3.3, 5 В. Діапазон EC: 0 – 2000 мкс/см [16].

Дана система потребує керування пристроями високої напруги, 220В. На приклад: осушувач/зволожувач повітря, система опалення, фітолампи і тд. Raspberry Pi не може управляти елементами з вимогами до напруги. Для вирішення цієї проблеми необхідно включити інтерфейс керування, тобто реле, яке знаходиться між Raspberry Pi та потужним пристроєм [17].

Реле – це вимикач з електричним приводом. Реле замикає та розмикає ланцюги, коли приводиться в дію електронним сигналом малої потужності [17]. Для реалізації системи можна використати 8-канальну плату розширення реле для Raspberry Pi (рис. 2.2).

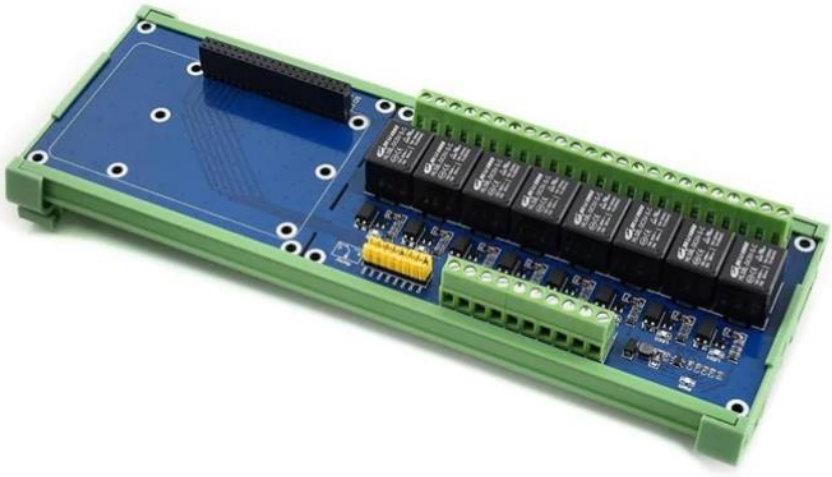

Рисунок 2.2. 8-канальна плата розширення реле для Raspberry Pi

Плата розширення з реле, розроблена спеціально для Raspberry Pi. Вбудовані світлодіоди вказують стан кожного реле для візуальної індикації та зручного налагодження. Його навантаження до 5A 250V AC або 5A 30V DC [18].

Нижче наведено концептуальну модель архітектури системи мінерального живлення рослин (рис 2.3).

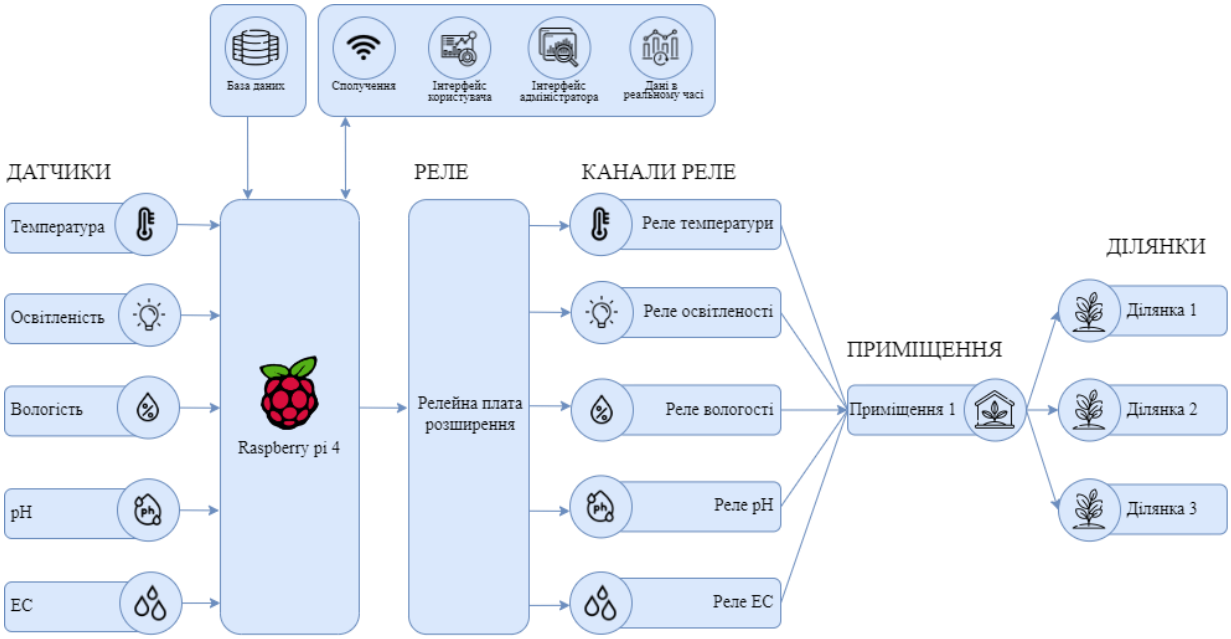

Рисунок 2.3. Концептуальна модель архітектури системи мінерального живлення рослин

Raspberry Pi поставляється з різними попередньо встановленим програмним забезпеченням, включаючи веб-браузер, офісний пакет та термінал. Raspberry Pi підтримує використання будь-яких мов програмування, компіляторів або інтерпретаторів, які існують для Linux. Однак Python за замовчуванням встановлено на Raspberry Pi. Для створення системи мінерального живлення рослин на Raspberry Pi необхідно встановлення веб-фреймворку Django на основі Python. Будучи відкритим і безкоштовним, фреймворк Django є відмінним вибором для створення веб-додатку за допомогою мови Python.

Оскільки впровадження системи мінерального живлення рослин є вартісним рішенням, для імітації роботи датчиків та інших пристроїв створено методи, які генерують дані, тим самим демонструється робота системи.

2.2. Створення логічної і фізичної бази даних для системи

Логічна модель даних описує дані якомога детальніше, незалежно від того, як вони будуть фізично реалізовані в базі даних. Логічна модель бази даних демонструє первинні та зовнішні ключі, взаємозв'язки між таблицями та атрибути всіх сутностей [19]. Побудовано наступну логічну модель (рис. 2.4):

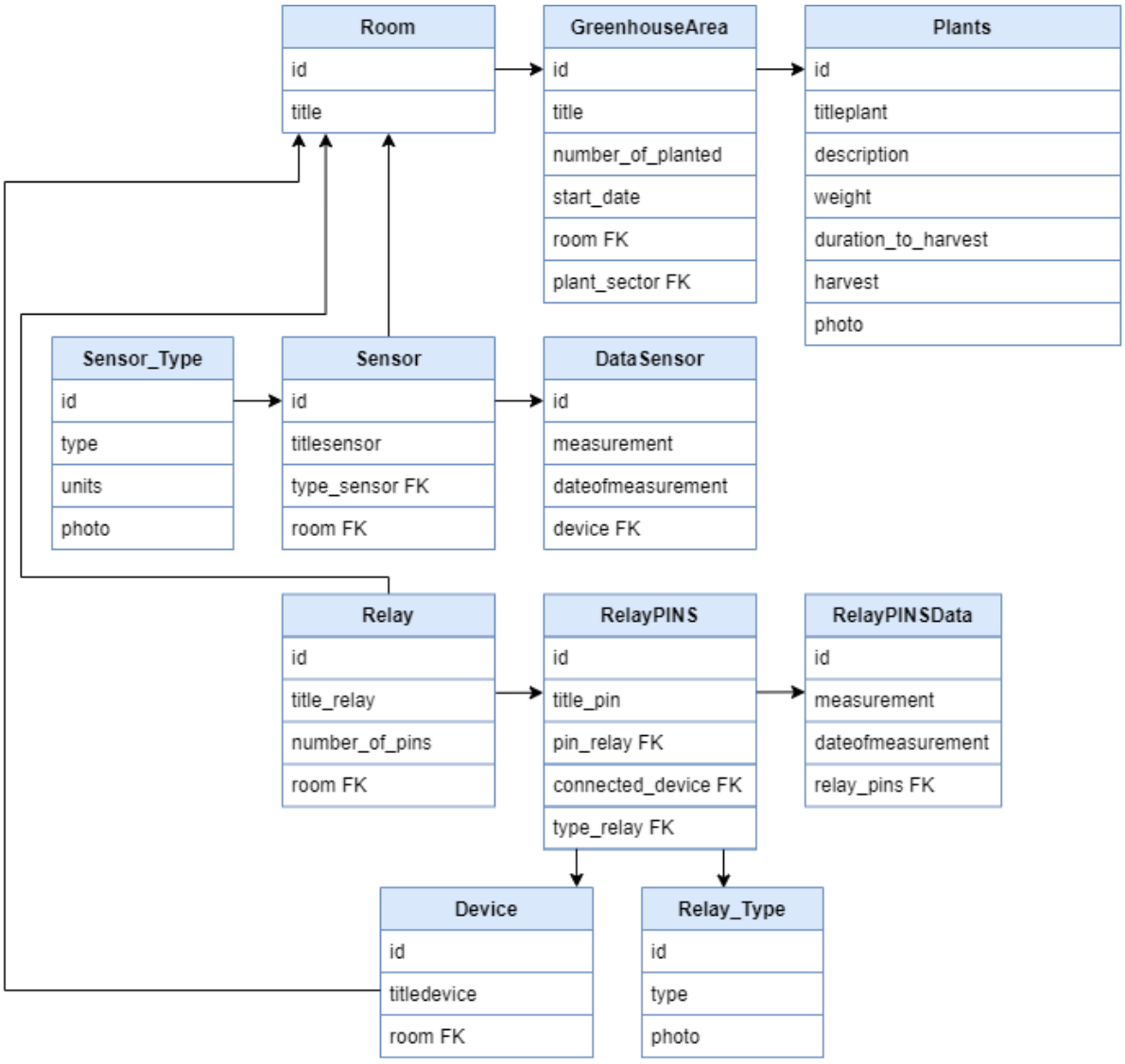

Рисунок 2.4. Логічна модель бази даних

Фізична модель даних представляє, як модель буде побудована в базі даних. Вона показує всі структури таблиць, включаючи ім'я стовпця, тип даних стовпця, обмеження стовпців, первинний ключ, зовнішній ключ і зв'язки між таблицями [20]. В результаті побудовано наступну фізичну модель (рис 2.5):

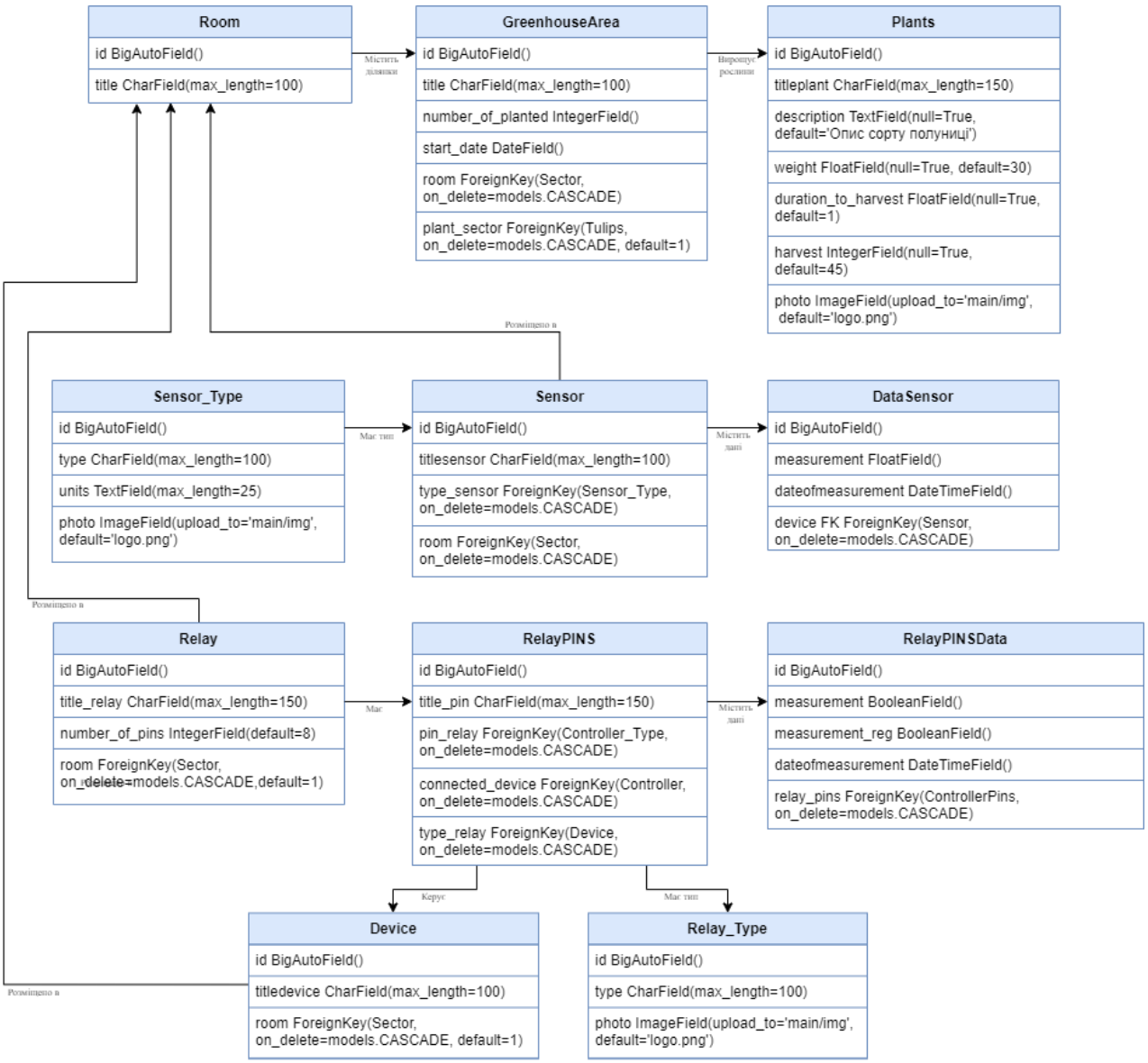

Рисунок 2.5. Фізична модель бази даних

Таблиця Рослини (Plants) призначена для внесення інформації про сорт полуниці, що вирощуються в теплиці. Вона (табл. 2.3) містить наступні поля: унікальний ідентифікатор, назву сорту полуниці, відомості про сорт, середню вагу однієї ягоди, середню вагу ягід в кг, зібраних з одного куща, кількість днів для достигання ягоди та зображення сорту.

Таблиця 2.3 – **Поля таблиці Рослини**

Назва поля моделі	**Назва типу поля**
1	**2**
id	BigAutoField()
titleplant	CharField()

Продовження табл 2.3

Назва поля моделі	Назва типу поля
1	**2**
description	TextField()
weight	FloatField()
harvest	FloatField()
duration_to_harvest	IntegerField()
photo	ImageField()

Таблиця Приміщення(Room) використовується для створення нового приміщення в теплиці. Вона (табл. 2.4) містить наступні поля: унікальний ідентифікатор та назву приміщення теплиці.

Таблиця 2.4 – **Поля таблиці Приміщення**

Назва поля моделі	Назва типу поля
1	**2**
id	BigAutoField()
title	CharField()

Таблиця Ділянки приміщення (GreenhouseArea) використовується для створення нової ділянки в приміщенні теплиці. Одне приміщення теплиці може містити багато ділянок, для вирощування рослин. На одній ділянці може рости лише один сорт полуниці. Вона (табл. 2.5) містить наступні поля: унікальний ідентифікатор, назву ділянки, кількість саджанців, що висаджено на ділянці, дату висадки саджанців, приміщення в якому розміщена ділянка та відповідний сорт полуниці.

Таблиця 2.5 – **Поля таблиці Ділянки приміщення**

Назва поля моделі	Назва типу поля
1	**2**
id	BigAutoField()
title	CharField()

Продовження табл 2.5

Назва поля моделі	Назва типу поля
1	**2**
start_date	DateField()
finish_date	DateField()
room	ForeignKey(Room)
plant_sector	ForeignKey(Plant)

Таблиця Типи датчиків (Sensor_Types) призначена для створення будь-якого типу датчика, що буде встановлено в теплиці. Вона (табл. 2.6) містить наступні поля: унікальний ідентифікатор, тип датчика, одиниці вимірювання та ілюстрація типу датчика.

Таблиця 2.6 – **Поля таблиці Типи датчиків**

Назва поля моделі	Назва типу поля
1	**2**
id	BigAutoField()
type	CharField()
units	CharField()
photo	ImageField()

Таблиця Датчики (Sensors) призначена для створення датчику та присвоєння йому відповідного типу датчика. Датчик може містити лише один тип датчику. Також вказується в якому приміщенні встановлено даний датчик. Вона (табл. 2.7) містить наступні поля: унікальний ідентифікатор, назву датчика, тип датчика та місце розташування датчика в теплиці.

Таблиця 2.7 – **Поля таблиці Датчики**

Назва поля моделі	Назва типу поля
1	**2**
id	BigAutoField()
titlesensor	CharField()

Продовження табл 2.7.

Назва поля моделі	Назва типу поля
1	**2**
type_sensors	ForeignKey(Sensor_Type)
room	ForeignKey(Room)

Таблиця Дані з датчиків (DataSensors) призначена для того, щоб зберігати дані, що надійшли з датчика. Вона (табл. 2.8) містить наступні поля: унікальний ідентифікатор, показники з датчика, дата та час коли отримано дані з датчика та датчик, з якого отримано дані.

Таблиця 2.8 – **Поля таблиці Дані з датчиків**

Назва поля моделі	Назва типу поля
1	**2**
id	BigAutoField()
measurement	IntegerField()
dateofmeasurement	DateTimeField()
device	ForeignKey(Sensor)

Таблиця Типи реле (Relay_Type) призначена для створення будь-якого типу каналу реле. Вона (табл. 2.9) містить наступні поля: унікальний ідентифікатор, тип реле, та ілюстрацію типу каналу реле.

Таблиця 2.9 – **Поля таблиці Типи реле**

Назва поля моделі	Назва типу поля
1	**2**
id	BigAutoField()
type	CharField()
photo	ImageField()

Таблиця Реле (Relay) призначена для створення плати розширення реле, що буде встановлено в теплицю. Вказується в якому приміщенні встановлено дане реле. Вона (табл. 2.10) містить наступні поля: унікальний ідентифікатор, назву реле, кількість каналів реле та місце розташування реле в теплиці.

Таблиця 2.10 – **Поля таблиці Реле**

Назва поля моделі	**Назва типу поля**
1	**2**
id	BigAutoField()
titlerelay	CharField()
number_of_pins	IntegerField()
room	ForeignKey(Room)

Таблиця Пристрої (Device) призначена для створення пристрою, що буде встановлено в теплицю, яким буде керувати канал реле. В таблиці вказується в якому приміщенні встановлено даний пристрій. Вона (табл. 2.11) містить наступні поля: унікальний ідентифікатор, назву пристрою та місце розташування реле в теплиці.

Таблиця 2.11 – **Поля таблиці Пристрої**

Назва поля моделі	**Назва типу поля**
1	**2**
id	BigAutoField()
titledevice	CharField()
room	ForeignKey(Room)

Таблиця Канали реле (RelayPINS) призначена для створення каналів реле, що розташовані на платі розширення. Плата розширення реле може містити багато каналів реле. Також вказується тип каналу реле та пристрій з яким взаємодіє реле. Вона (табл. 2.12) містить наступні поля: унікальний ідентифікатор, назву каналу реле, тип каналу, на якій платі розташовані канали та пристрій з яким взаємодіє канал реле.

Таблиця 2.12 – **Поля таблиці Канали реле**

Назва поля моделі	**Назва типу поля**
1	**2**
id	BigAutoField()
title_pin	CharField()

Продовження табл. 2.12

Назва поля моделі	Назва типу поля
1	**2**
pin_relay	ForeignKey(RelayPINS)
connected_device	ForeignKey(Device)
type_relay	ForeignKey(Relay_Type)

Таблиця Дані з каналу реле (RelayPINSData) призначена для того, щоб зберігати дані, що надійшли з каналів реле. Вона (табл. 2.13) містить наступні поля: унікальний ідентифікатор, показники з каналу реле, дата та час коли отримано дані та канал реле, з якого отримано дані.

Таблиця 2.13 – **Поля таблиці Дані з каналу реле**

Назва поля моделі	Назва типу поля
1	**2**
id	BigAutoField()
measurement	BooleanField()
dateofmeasurement	DateTimeField()
relay_pins	ForeignKey(RelayPINS)

2.3. Опис програмних засобів реалізації

Для реалізації системи обрано фреймворк для створення веб-застосунків за допомогою мови програмування Python – Django. Цей фреймворк використовується для розробки великої кількості проєктів, у тому числі таких, як Pinterest, PBS, Instagram, BitBucket, Washington Times, Mozilla та багатьох інших [21].

Фреймворк Django є безкоштовним. Він розвивається як Open Source, його вихідний код відкритий, його репозиторій можна знайти на githube.

Фреймворк Django реалізує архітектурний патерн Model-View-Template або скорочено MVT [22]. Схематичний вигляд архітектури MVT Django має наступний вигляд (рис. 2.6):

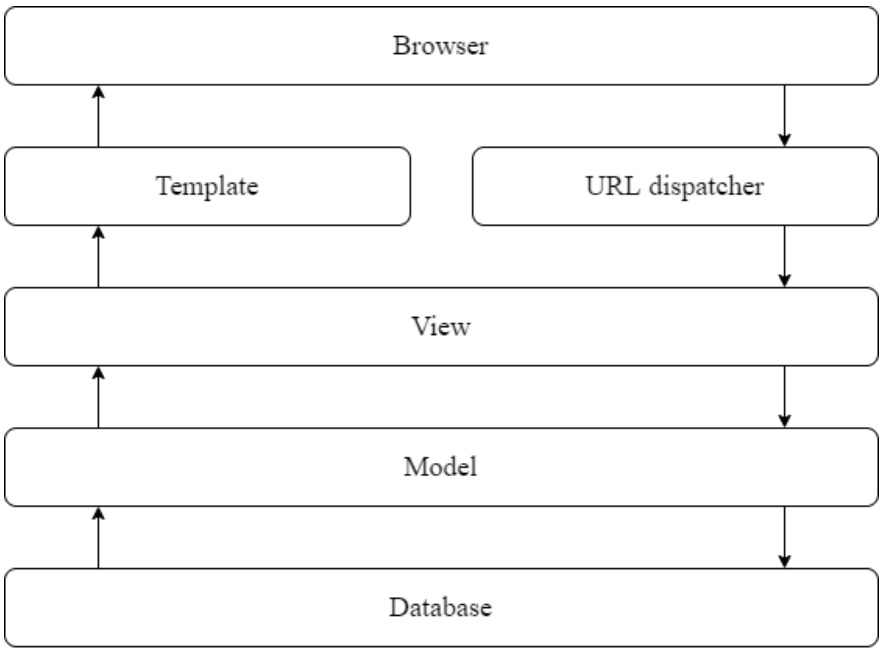

Рисунок 2.6. Архітектура MVT Django

Складовими патерну є наступні елементи:

- URL dispatcher: при отриманні запиту на підставі запитаної URL-адреси визначає, який ресурс повинен обробляти даний запит.
- View: отримує запит, обробляє його та відправляє у відповідь користувачеві певну відповідь. Якщо для обробки запиту необхідне звернення до моделі та бази даних, то View взаємодіє з нею. Для створення відповіді може використовуватися Template.
- Model: описує дані, що використовуються програмою. Окремі класи відповідають таблицям бази даних.
- Template: зображує логіку представлення згенерованої розмітки html [22].

Коли до програми надходить запит, URL dispatcher визначає, з яким ресурсом зіставляється даний запит і передає цей запит обраному ресурсу. Ресурс фактично представляє функцію або View, який отримує запит та певним чином обробляє його. У процесі обробки View може звертатися до моделей та бази даних, отримувати з неї дані або зберігати у ній дані.

Результат обробки запиту відправляється назад, і цей результат користувач бачить у своєму браузері [22].

Даний фреймворк має наступні переваги, завдяки яким його обрано для створення системи:

- Вбудований інтерфейс адміністратора
- Диспетчер URL на основі регулярних виразів
- Система кешування
- Система фільтрів для побудови додаткових обробників запитів
- Бібліотека для роботи з формами
- Вбудована автоматична документація до тегів шаблонів [23]

Django ORM (Object Relational Mapping) є однією з найпотужніших особливостей Django. ORM – технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних» [24].

Веб-додатки Django отримують доступ і керують даними через об'єкти Python, які називаються моделями. Моделі визначають структуру даних, що зберігаються. Моделі містять типи полів, їх максимальний розмір, значення за замовчуванням, параметри списку вибору, текст довідки для документації, текст міток для форм і так далі. Після того, як обрано базу даних, в яку зберігаються дані, немає необхідності безпосередньо працювати з нею, оскільки Django конвертує моделі в SQL код [25].

Оскільки модель – це по суті, макет бази даних, з додатковими метаданими, то будуть використовуватись наступні типи даних полів (табл. 2.14).

Таблиця 2.14 – **Типи полів моделей**

Назва типу поля	**Опис**
1	**2**
BigAutoField()	зберігає 64-бітове ціле число
BooleanField()	зберігає значення True або False(0 або 1)
FloatField()	зберігає число з плаваючою точкою
DateField()	зберігає дату
DateTimeField()	зберігає дату та час

Продовження табл. 2.14

Назва типу поля	Опис
1	**2**
IntegerField()	зберігає значення типу Number
CharField(max_length=N)	зберігає рядок довжиною не більше N
TextField()	зберігає рядок будь-якої довжини
ImageField()	зберігає рядок, що представляє собою зображення

В файлі models.py для полів моделей можна вказувати параметри:

- max_length: встановлює максимальну можливу кількість символів при заповненні поля;
- on_delete=models.CASCADE: при видаленні запису з однієї таблиці, автоматично буде видалено усі записи з інших таблиць, що були зв’язані;
- default: значення, яким автоматично буде заповнюватись поле при заповненні таблиці;
- ulpoad_to: вказує на директорію, в яку необхідно завантажувати файл.

Реалізувавши всі моделі в models.py, необхідно виконати та застосувати міграції. Вони слугують способом розпізнання змін, що внесено в моделі (додавання поля або видалення моделі і так далі) для того, щоб внести їх в таблиці бази даних. Для того, щоб виконати міграцію, необхідно в консоль ввести наступні команди:

- python manage.py makemigrations: відповідає за створення нових міграцій на основі змін, що внесено в моделі;
- python manage.py migrate: відповідає за застосування міграції.

Слід розглядати міграції, як систему контролю версій бази даних. Файли міграцій зберігаються з каталозі migrations. Весь створений SQLite код знаходиться в файлі db.sqlite3.

Кожен проєкт написаний на Django має стандартну структуру, що містить наступні папки та файли:

- Папка migrations: зберігає інформацію, що дозволяє співставити базу даних SQLite та визначення моделей;
- __init__.py: вказує інтерпретатору python, що поточний каталог буде розглядатись в якості пакета;
- admin.py: призначений для адміністративних функцій, саме тут відбувається реєстрація моделей, що використовуються в інтерфейсі адміністратора;
- apps.py: визначає конфігурацію додатка;
- models.py: зберігає визначення моделей – джерело інформації про таблиці. Модель – це по суті, макет бази даних, з додатковими метаданими. Модель відповідає за бізнес-логіку, методи, властивості та інші елементи пов'язані з маніпуляцією даних. Також моделі дозволяють розробникам створювати, читати, оновлювати та видаляти об'єкти в базі даних;
- views.py: визначає функції, які отримують запити користувачів, обробляють їх та повертають відповідь;
- Settings.py: файл, що містить конфігурації додатку [26].

Django підтримує наступні бази: MySQL, PostgreSQL, Oracle та SQLite. В даній роботі використовується SQLite, який включено в Python за замовчуванням.

В якості середовища розробки додатку використано PyCharm – інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічну відладку, інструмент для запуску юніт-тестів і підтримує веб-розробку на Django [27].

Для створення клієнтського інтерфейсу використано Bootstrap – вільний набір інструментів для створення сайтів та веб-додатків. Він включає в себе HTML та CSS-шаблони оформлення веб-форм, кнопок, блоків навігації та інших компонентів веб-інтерфейсу [28].

Для візуалізації даних у вигляді графіків застосовано бібліотеку Python Plotly Library та Chart.js. Plotly – це бібліотека з відкритим вихідним кодом, яку

можна використовувати для візуалізації та розуміння даних просто й легко. Plotly підтримує різноманітні типи графіків, як-от лінійні діаграми, діаграми розсіювання, гістограми, тощо [29]. Chart.js дозволяє користувачеві будувати графіки різних типів. В бібліотеці також реалізовано готові анімації, які можна застосувати при зміні даних і оновленні кольорів.

Для побудови логічної та фізичної моделі бази даних використано draw.io – безкоштовне кросплатформне програмне забезпечення для малювання графіків із відкритим вихідним кодом. Його інтерфейс можна використовувати для створення діаграм, побудови блок-схем та відношень сутностей [30].

2.4. Висновок до другого розділу

Отже, в даному розділі було підібрано обладнання, яке підійшло би для створення системи мінерального живлення рослин.

- Обрано одноплатний комп'ютер Raspberry Pi 4, який лежить основі системи мінерального живлення рослин.
- Підібрано датчики температури та вологості повітря, освітленості у приміщені, кислотності та електропровідності рідини.
- Обрано плату розширення реле для Raspberry Pi для того щоб керувати пристроями високої напруги, 220В.
- Створено базу даних для системи мінерального живлення рослин, розроблено логічну та фізичну моделі бази даних. Для побудови моделей бази даних використано draw.io – безкоштовне кросплатформне програмне забезпечення.
- Виконано опис засобів реалізації, а саме: обрано фреймворк для створення веб-застосунків за допомогою мови програмування Python – Django, описано процес створення моделей класів, які Django конвертує в SQL код та підібрано бібліотеки для візуалізації отриманих значень з датчиків – Python Plotly Library та Chart.js.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Інтерфейс програми користувача

Для інтерфейсу користувача передбачено наступні сторінки: дашборд, приміщення теплиці, каталог сортів, прилади та адміністрування, до який можна потрапити з навігаційного меню.

3.1.1. Дашборд

Дашборд – це інструмент керування інформацією, який використовується для відстеження, аналізу та відображення ключових показників. Він є динамічним інструментом, в якому дані постійно оновлюються – автоматично або вручну.

Інформаційна панель IoT (дашборд) – програмний інструмент, що дозволяє організовувати, відображати та керувати зібраними даними, які передаються пристроями в екосистемі. Тобто дашборд зв’язується з таблицями, що містять дані, аналізує та відображає показники в реальному часі та в зручному для сприйняття форматі [31]. Інформаційна панель IoT може бути корисною з наступних причин:

- Перегляд та аналіз вхідної інформації, отриманої з датчиків в режимі реального часу.
- Аналіз та систематизація, отриманих значень.
- Візуальне представлення показників, на приклад, за допомогою діаграм та графіків [31].

Для відображення показників обрано картки Bootstrap. Картка – це гнучкий і розширюваний контейнер. Картка може містити заголовки, підзаголовки, текст, списки, зображення, посилання, тощо. Вона також включає параметри верхніх та нижніх колонтитулів, внутрішніх та зовнішніх відступів, кольори фону за контекстом та параметри відображення [32]. В загальному випадку структура карти виглядає наступним чином (рис. 3.1):

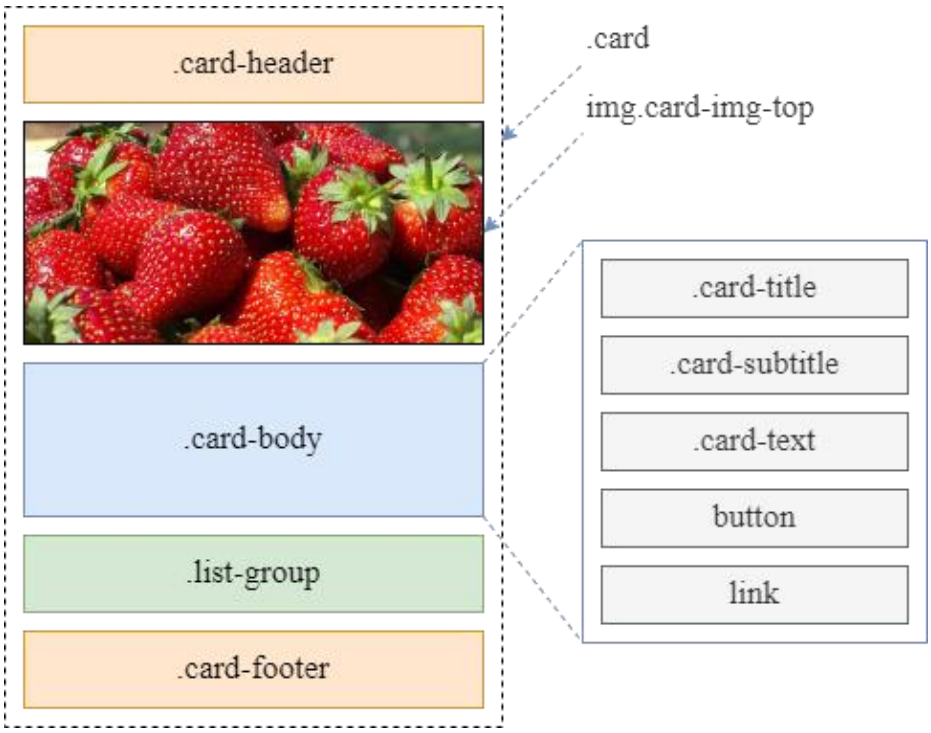

Рисунок 3.1. Загальна структура картки Bootstrap

Основна інформація картки зберігається в блоці .card-body. Даний блок може містити заголовки, підзаголовки, текст, кнопки, посилання та інше. Компоненти як .card-header, img.card-img, .card-body, .list-group, .card-footer не є обов'язковими. Компоненти .card-header, .card-footer, img.card-img можуть з'явитись один або жодного разу.

У випадку з дашбордом структура картки Bootstrap виглядає наступним чином (рис. 3.2). В блоці .card-body міститься заголовок та підзаголовок.

```
<div class="col my-2 ">
  <div class="card my-1 ">
    <div class="card-body ">
          <center>
              <h4>{{count}}</h4>
              <h5>Сортів рослин</h5>
          </center>
      </div>
  </div>
</div>
```

Рисунок 3.2. Картка Bootstrap для відображення показників

Заголовок містить показник розрахований в методі index(). Цей показник представляється об'єктами, які можна логічно розділити на дві групи: показники

отримані за допомогою функції count() та показники розраховані за допомогою агрегованої функції з відповідними параметрами Min, Avg, Max, Sum (рис. 3.3).

```
count = Tulips.objects.all().count()
count_sen = Sensor.objects.all().count()
count_con = Controller.objects.all().count()
count_plant = Planting_Sector.objects.all().count()

count_plant_min = Planting_Sector.objects.aggregate(a=Min('number_of_planted'))
count_plant_avg = Planting_Sector.objects.aggregate(a=Avg('number_of_planted', ))
count_plant_max = Planting_Sector.objects.aggregate(a=Max('number_of_planted', ))
count_plant_sum = Planting_Sector.objects.aggregate(a=Sum('number_of_planted', ))
```

Рисунок 3.3. Розрахунок показників для картки Bootstrap

Для відображення карток використовується Bootstrap Grid System – система сіток [33], а також Bootstrap Spacing для задання зовнішніх та внутрішніх відступів [34].

В результаті готовий дашборд виглядає наступним чином (рис. 3.4):

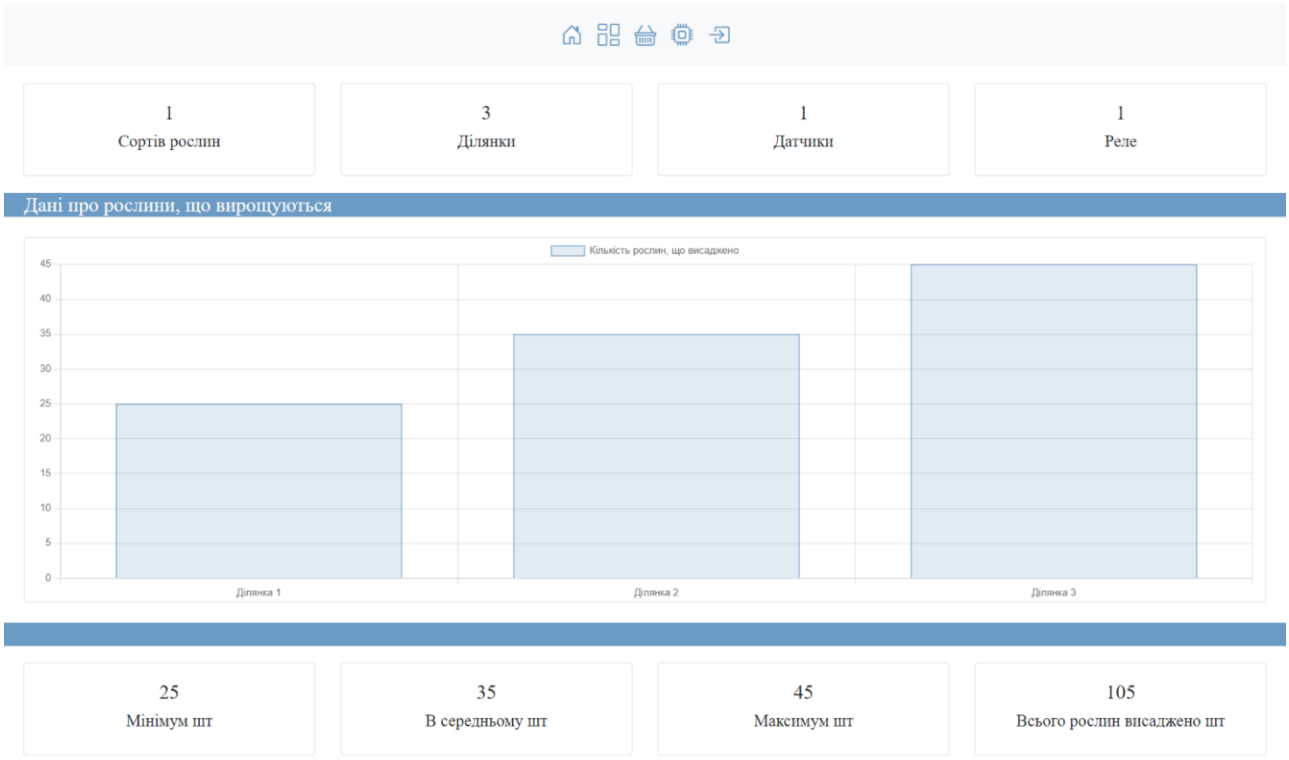

Рисунок 3.4. Дашборд

Даний дашборд відображає інформацію про кількість сортів рослин, що вирощуються, кількість ділянок, кількість встановлених датчиків та реле, а також мінімум, в середньому, максимум та суму висаджених рослин. Крім того в

дашборді зображено гістограму з даними про рослини, що вирощуються на ділянках. Гістограма містить назву ділянки та кількість висаджених на ній рослин. На сторінку дашборду можна потрапити натиснувши на кнопку Дашборд в навігаційному меню (рис. 3.5).

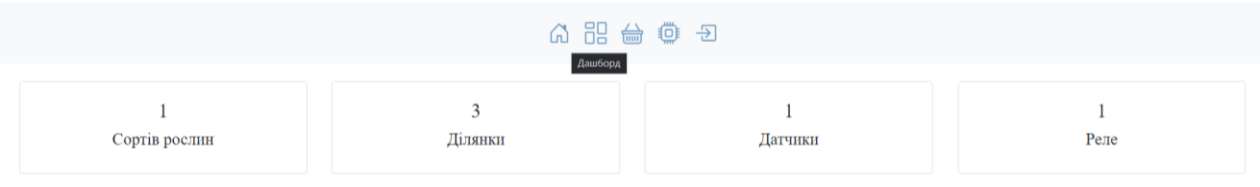

Рисунок 3.5. Посилання на дашборд в навігаційному меню

3.1.2. Приміщення теплиці

Натиснувши на кнопку Приміщення теплиці в навігаційному меню, користувач зможе переглянути список всіх приміщень теплиці, що додано до бази даних. Для цього в методі sector() створено об’єкт sectors, що містить всі об’єкти класу Sector (рис. 3.6).

```
47  def sector(request):
48      sectors = Sector.objects.all()
49      return render(request, 'main/sector.html', {'sectors':sectors})
```

Рисунок 3.6. Метод sector()

Клас Sector містить об’єкт, що задає назву приміщенню. Даний клас представляє модель, яка визначає структуру даних, що зберігається (рис. 3.7).

```
15  class Sector(models.Model):
16      title = models.CharField('Назва кімнати', max_length=100)
17      def __str__(self):
18          return self.title
19
20      class Meta:
21          verbose_name = 'Кімната'
22          verbose_name_plural = 'Кімнати'
```

Рисунок 3.7. Клас Sector

Після цього на сторінці приміщення теплиці створено нову картку Bootstrap (рис. 3.8), яка автоматично заповнюється даними, використовуючи шаблонізатор Jinja – набір інструкцій, який допомагає автоматизувати створення

html шаблонів [35]. Для цього було створено цикл, який заповнює карту Bootstrap, даними, що зберігається в об’єкті sectors. Також до картки додано три кнопки з посиланнями на сторінки Ділянки, Датчики та Реле.

```
<div class="row row-cols-1 row-cols-md-3 m-2">
  {% for el in sectors %}
  <div class="col my-2 ">
    <div class="card my-1 ">
      <div class="card-body ">
        {% load static %}<img class ="greenimg1 pb-3" src="{% static 'main/img/smart-farming.png' %}" alt="Card image cap">
        <h4 class="card-title">{{el.title}}</h4>
        <a href="{% url 'sector-details' el.id %}" class="btn btn-info">Сектори для вирощування</a>
        <a href="{% url 'sensor-details' el.id %}" class="btn btn-info">Датчики</a>
        <a href="{% url 'controller-details' el.id %}" class="btn btn-info">Контролери</a>
      </div>
    </div>
  </div>
  {% endfor %}
</div>
```

Рисунок 3.8. Картка Bootstrap для відображення показників

Зовнішній вигляд сторінки, зображено на (рис. 3.9).

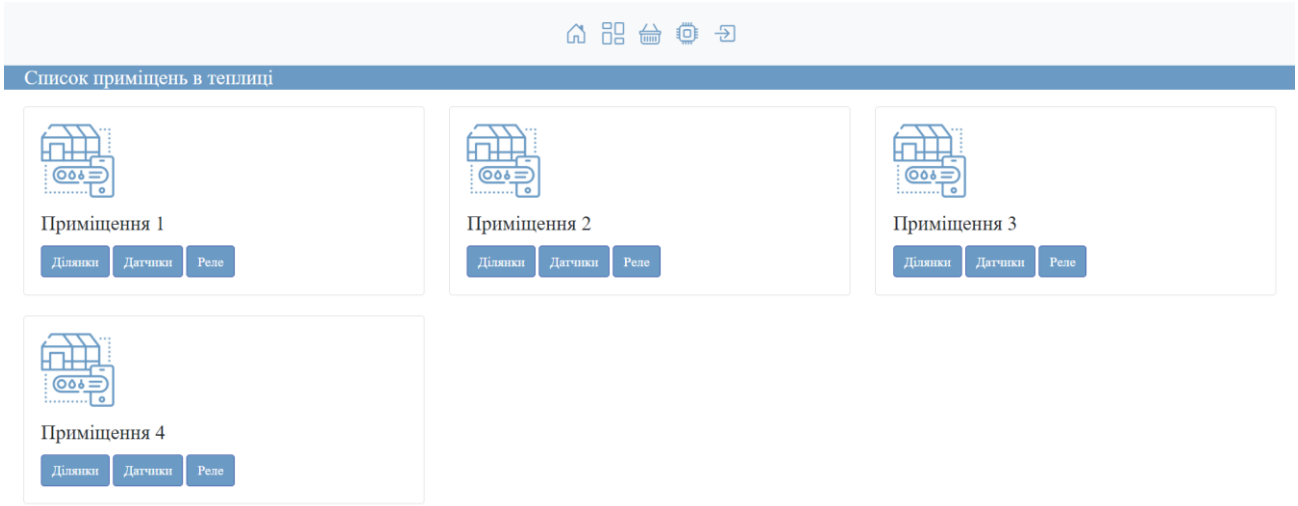

Рисунок 3.9. Сторінка списку приміщень теплиці

Натиснувши на кнопку Ділянки, що розміщена карточці, відриється перелік ділянок, що належать відповідному приміщенню. Інформація про ділянку виводиться за допомогою метода planting_sector_detail (рис. 3.10), який приймає параметр id.

```
69  def planting_sector_detail(request, id):
70      sector = Sector.objects.get(id=id)
71      planting_sectors = Planting_Sector.objects.filter(sector=id)
72      return render(request, 'main/planting_sector.html', {'sector': sector, 'planting_sectors': planting_sectors})
```

Рисунок 3.10. Метод planting_sector_detail()

Для того, щоб виконати фільтрацію об’єктів всередині об’єкта planting_sectors, що містить об’єкти класу Planting_Sector, де sector=id. Клас

Planting_Sector (рис. 3.11) має наступні об’єкти: title – назва ділянки, number_of_planted – кількість висаджених саджанців, start_date – дата висадки. Оскільки клас Planting_Sector представляє собою модель, у якій кожен атрибут моделі представляє поле бази даних, то для зв’язку Planting_Sector з іншими моделями додано два зовнішніх ключі. Sector – для зв’язку Planting_Sector з Sector та straw_sector для зв’язку з Strawberry. Таким чином, на приклад, Кімната 1 матиме лише ті ділянки, що належать їй.

```
class Planting_Sector(models.Model):
    title = models.CharField('Назва ділянки', max_length=100)
    number_of_planted=models.IntegerField('Кількість висаджених саджанців')
    start_date = models.DateField('Дата висадки')
    sector = models.ForeignKey(Sector, on_delete=models.CASCADE)
    tulip_sector = models.ForeignKey(Tulips, on_delete=models.CASCADE, default=1)
    def __str__(self):

        return str(self.title) +str(self.start_date)

    class Meta:
        verbose_name = 'Ділянка'
        verbose_name_plural = 'Ділянки'
```

Рисунок 3.11. Клас Planting_Sector

Інформація про ділянку виводиться у вигляді картки Bootstrap. На ній вказано: назву ділянки, кількість висаджених рослин, назву сорту рослини, кількість кілограм полуниці, що можна зібрати з ділянки, а також з дати висадження саджанцю вираховується період вегетації, цвітіння та збір ягід полуниці. Назва сорту полуниці має гіперпосилання для того, щоб переглянути інформацію про рослину, що висаджено на даній ділянці.

Також на цій сторінці (рис. 3.12) побудовано гістограму, що відображає ділянку та кількісті кілограм полуниці, що можна зібрати.

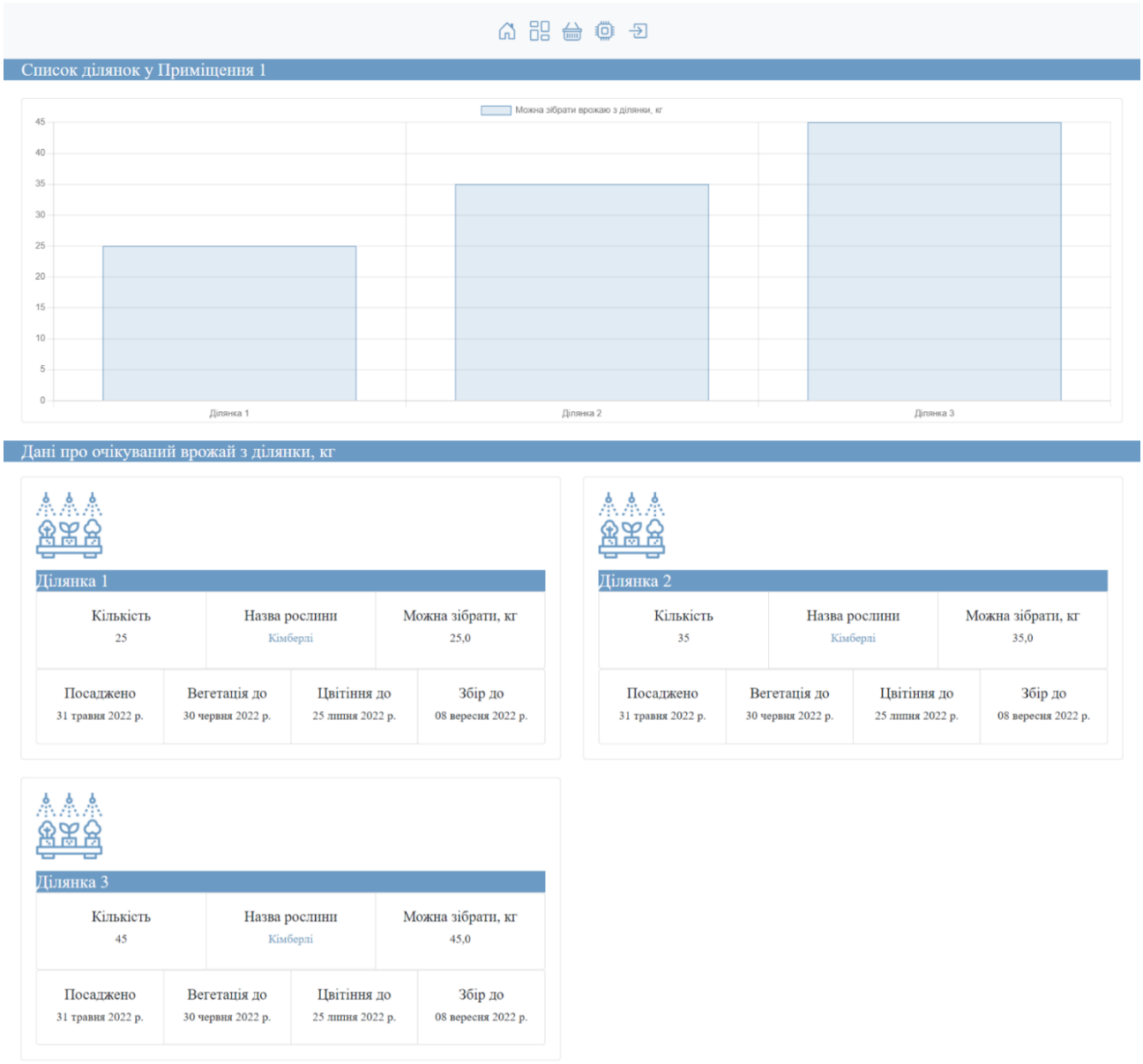

Рисунок 3.12. Сторінка списку ділянок приміщення теплиці

Натиснувши на гіперпосилання з назвою сорту, користувач потрапить на сторінку з карточкою (рис. 3.13), що містить інформацію про рослину. Карточка містить зображення, назву сорту полуниці, середню вагу однієї ягоди, вагу середнього врожаю з одного куща, кількість днів за скільки достигає ягода, а також нотатки до даного сорту.

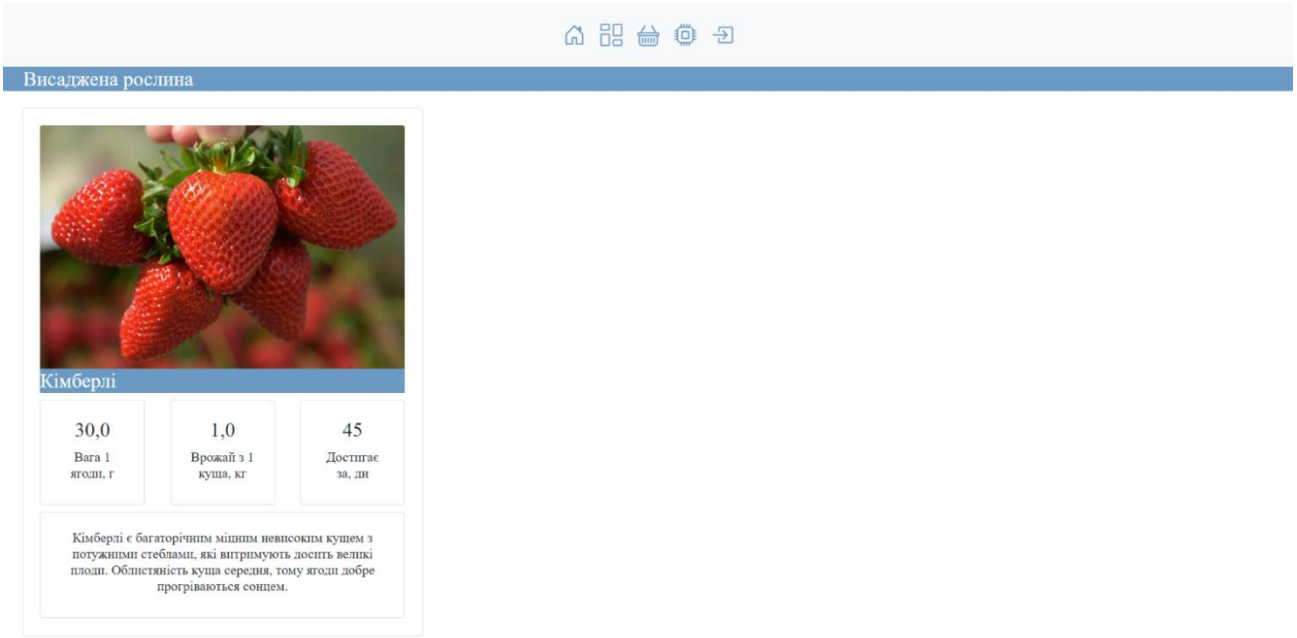

Рисунок 3.13. Сторінка для перегляду інформації про висаджений сорт полуниці

Натиснувши на кнопку Датчики на карточці, користувач побачить перелік встановлений датчиків у даному приміщенні.

Інформація про датчик, що встановлений в приміщенні відображається за допомогою метода sensore_detail (рис. 3.14), який приймає параметр id.

```
79  def sensore_detail(request,id):
80      sector = Sector.objects.get(id=id)
81      sensors = Sensor.objects.filter(sector=id)
82      phototype = Sensor_Type.objects.filter(id=id)
83      return render(request, 'main/sensor.html', {'sector': sector, 'sensors': sensors, 'phototype':phototype})
```

Рисунок 3.14. Метод sensore_detail()

Для виконання фільтрації об’єктів всередині об’єкта sensors, який містить об’єкти класу Sensor, де sector=id. Клас Sensor (рис. 3.15) має наступний об’єкти: title sensor – назва датчика Для зв’язку класу Sensor з іншими моделями додано наступні зовнішніх ключі. Sector – для зв’язку Sensor з Sector та type_sensors для зв’язку з Sensor_Type. Таким чином, на приклад, Кімната 1 матиме лише ті датчики, що в ній встановлено з відповідним типом датчика.

```python
class Sensor(models.Model):
    titlesensor = models.CharField('Назва датчика',max_length=100)
    type_sensors = models.ForeignKey(Sensor_Type, on_delete=models.CASCADE)
    sector = models.ForeignKey(Sector, on_delete=models.CASCADE)
    def __str__(self):
        return self.titlesensor

    class Meta:
        verbose_name = 'Датчик'
        verbose_name_plural = 'Датчики'
```

Рисунок 3.15. Клас Sensor

Інформація про датчик для виводиться у вигляді картки Bootstrap (рис. 3.16). На карточці датчика вказано назву датчика, його розташування, а також тип встановленого датчика.

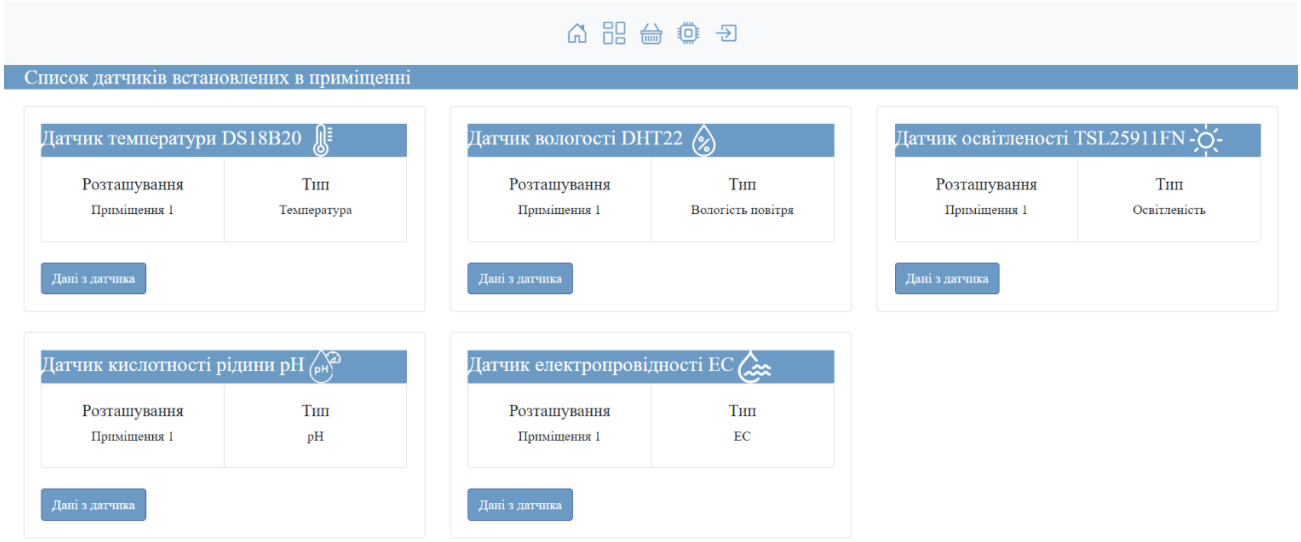

Рисунок 3.16. Сторінка списку датчиків встановлених в приміщенні

Для перегляду даних, що отримав датчик необхідно натиснути на кнопку Дані з датчика. Показники датчику зображено у вигляді лінійного графіку, що відображає отримані показники за останню годину, а також індикатор, що показує останній отриманий показник даного графіка (рис. 3.17). Графіки було реалізовано за допомогою ChartJs та Plotly.

Рисунок 3.17. Сторінка перегляду даних датчика

Кожний показник, отриманий датчиком, перевіряється в методі datafromsensors() для відображення інформації про отримані дані. Для кожного типу датчика створена відповідна перевірка на відхилення від нормальних показників. В результаті користувач побачить чи показники в нормі, чи вони більше або менше за неї.

Для відображення встановлених реле у приміщенні, необхідно натиснути на кнопку Реле на відповідній карточці приміщення. Інформація про реле, встановлене в приміщенні відображається за допомогою метода controller_detail (рис. 3.18), що приймає параметр id.

```
220    def controller_detail(request,id):
221        sector = Sector.objects.get(id=id)
222        controllers = Controller.objects.filter(sector_con=id)
223        phototype = Controller_Type.objects.filter(id=id)
224        return render(request, 'main/controller.html', {'sector': sector, 'controllers': controllers,
225                                                        'phototype':phototype})
```

Рисунок 3.18. Метод controller_detail()

Для виконання фільтрації об’єктів всередині об’єкта controllers, який містить об’єкти класу Controller, де sector_con=id. Клас Controller (рис. 3.19) має наступний об’єкти: titlecontroller_con – назва реле для зв’язку класу Controller з іншими моделями додано наступні зовнішніх ключі. Sector_con – для зв’язку Controller з Sector.

```python
class Controller(models.Model):
    titlecontroller_con = models.CharField('Назва контроллера', max_length=150)
    type_controller_con = models.ForeignKey(Controller_Type, on_delete=models.CASCADE)
    sector_con = models.ForeignKey(Sector, on_delete=models.CASCADE, default=1)
    def __str__(self):
        return self.titlecontroller_con

    class Meta:
        verbose_name = 'Контроллер'
        verbose_name_plural = 'Контроллери'
```

Рисунок 3.19. Клас Controller

Карточка реле включає назву реле, розташування, а також кнопку Канали реле (рис. 3.20).

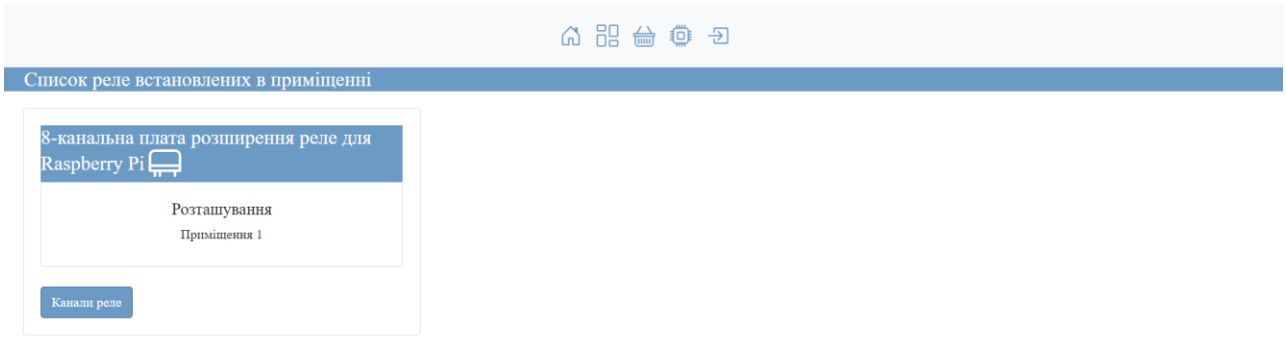

Рисунок 3.20. Сторінка списку реле встановлених в приміщенні

Для відображення каналів реле, необхідно натиснути на кнопку Канали реле на відповідній карточці реле. Інформація про реле, встановлене в приміщенні відображається за допомогою метода controller_pins_detail (рис. 3.21), що приймає параметр id.

```python
def controller_pins_detail(request, id):
    controllers = Controller.objects.get(id=id)
    controller_pins = ControllerPins.objects.filter(controllername_con=id)
    phototype = Controller_Type.objects.filter(id=id)
    return render(request, 'main/controller_pins.html', {'controllers': controllers,
                                                         'controller_pins':controller_pins, 'phototype':phototype})
```

Рисунок 3.21. Метод controller_pins_detail()

Для виконання фільтрації об’єктів всередині об’єкта controllers, який містить об’єкти класу Controller, де id=id. Клас ControllerPins має наступний об’єкти: titlecontroller_con – назва каналу реле, type_controller_con. Для зв’язку класу ControllerPins з іншими моделями додано наступні зовнішніх ключі.

type_controller_con – для зв’язку ControllerPins з Controller_Type, відповідно кожний канал реле матиме свій тип. controllername_con – для зв’язку Controller з ControllerPins, кожному реле належить лише його канали реле. device – для зв’язку ControllerPins з Device, відповідний канал реле керуватиме вказаним пристроєм.

Карточка реле включає назву каналу реле, пристрій, яким керує канал реле та тип реле, а також кнопку Дані з каналу реле (рис. 3.22).

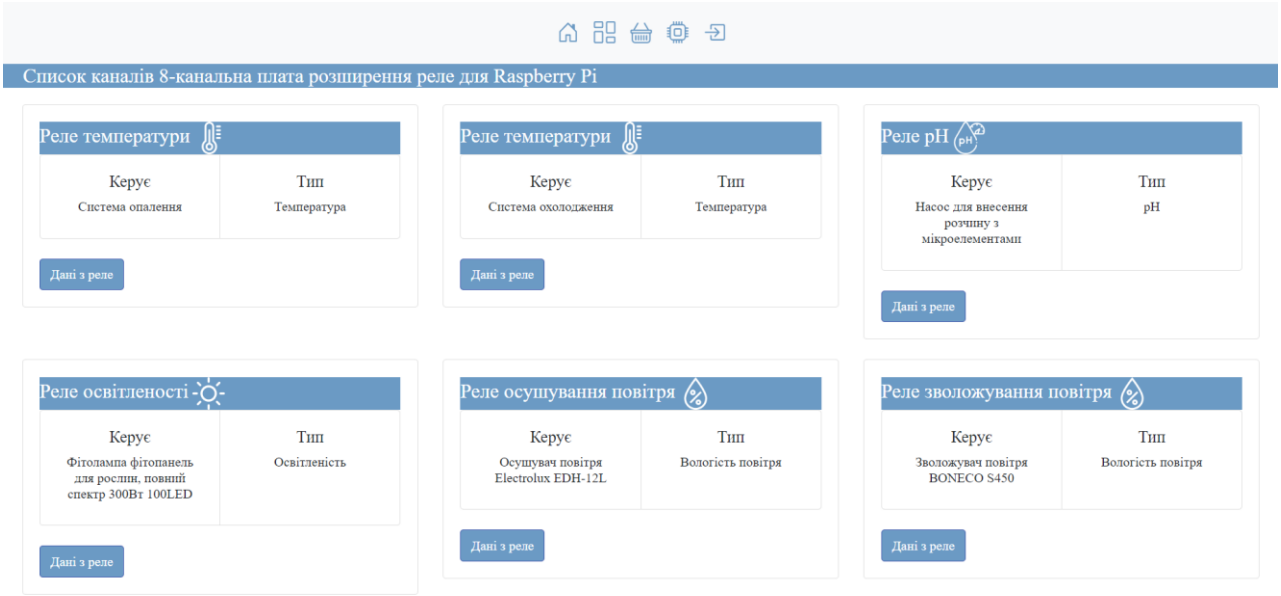

Рисунок 3.22. Сторінка списку каналів реле

Щоб переглянути відповідні дані з каналу реле, необхідно натиснути на кнопку Дані з каналу реле. Після цього користувач потрапить на сторінку Дані з каналу реле

Для відображення даних каналу реле створено метод datafromcontroller() (рис. 3.23), який має об’єкти controllername та datafromcon.

```
228 def datafromcontroller(request, id):
229     controllername = Controller.objects.get(id=id)
230     datafromcon = DataController.objects.filter(controllername_con=id).order_by('-id')
231     return render(request, 'main/datacon.html',{'controllername': controllername, 'datafromcon':datafromcon})
```

Рисунок 3.23. Метод datafromcontroller()

Controllername – містить всі об’єкти класу Controller та приймає метод get з параметром id=id.

Datafromcon – містить всі об’єкти класу DataController (рис. 3.24), які були відфільтровані за допомогою метода filter() з параметрами controllername_con =id, а також відсортовані в порядку спаду. Клас DataController має наступні об'єкти measurement_con – стане реле, measurement_reg – режим реле dateofmeasurement_con – дата та час отримання даних та зовнішній ключ controllername_con для зв’язку з класом ControllerPins.

```
class DataController(models.Model):
    measurement_con = models.BooleanField('Стан реле')
    measurement_reg = models.BooleanField('Режим реле',default=1)
    dateofmeasurement_con = models.DateTimeField('Дата та час')
    controllername_con = models.ForeignKey(ControllerPins, on_delete=models.CASCADE)

    def __str__(self):
        return 'Пристрій: ' + str(self.controllername_con) + ', показники реле: ' + str(self.measurement_con)


    class Meta:
        verbose_name = 'Дані з реле'
        verbose_name_plural = 'Дані з реле'
```

Рисунок 3.24. Клас DataController

Стан каналу реле відображено в таблиці, що містить дані про стан з реле, режим реле, а також дату та час їх отримання (рис. 3.25).

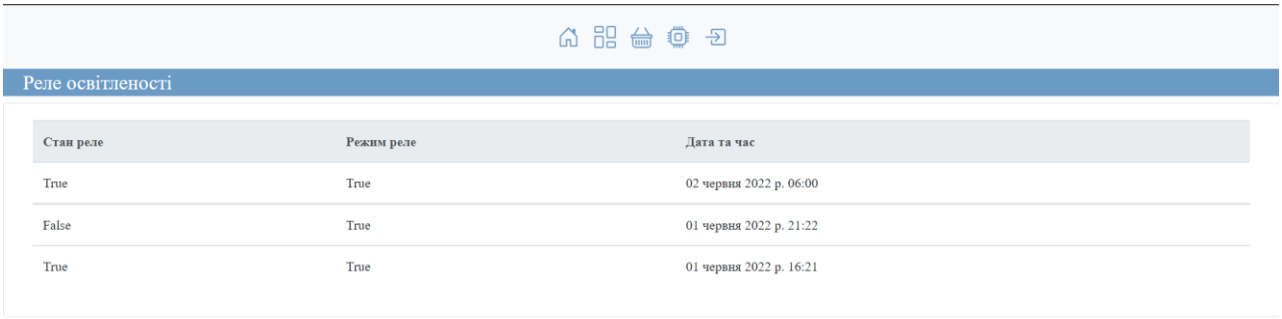

Рисунок 3.25. Сторінка перегляду даних реле

3.1.3. Каталог сортів

Натиснувши на кнопку Каталог сортів в навігаційному меню, користувач зможе переглянути список всіх сортів полуниці, що додано до бази даних. Створену сторінку каталогу сортів, зображено на (рис. 3.26).

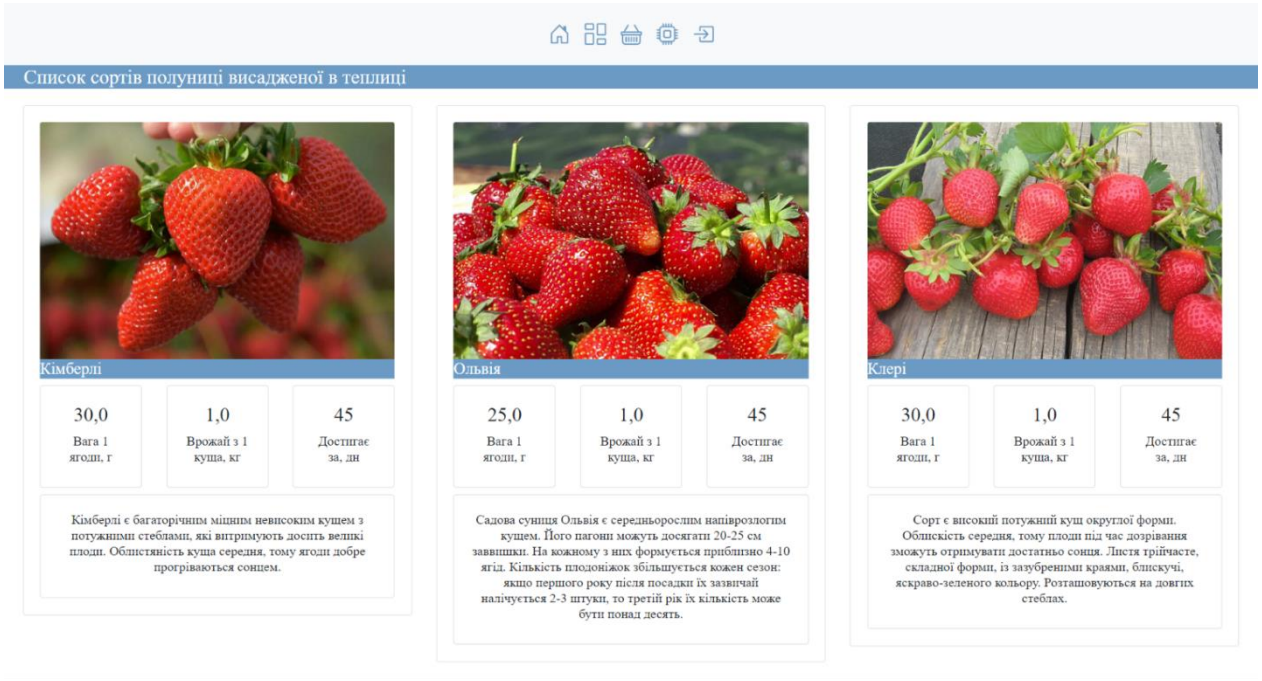

Рисунок 3.26. Сторінка списку сортів полуниці висадженої в теплиці

3.1.4. Прилади

Дана сторінка (рис. 3.27) створена для швидкого доступу до всіх наявних датчиків та реле теплиці. Інформація про датчики та реле відображається у вигляді карток Bootstrap та містить наступні дані: для датчиків – назва датчика, місце встановлення, тип датчика та останній показник; для реле – назва реле та місце встановлення.

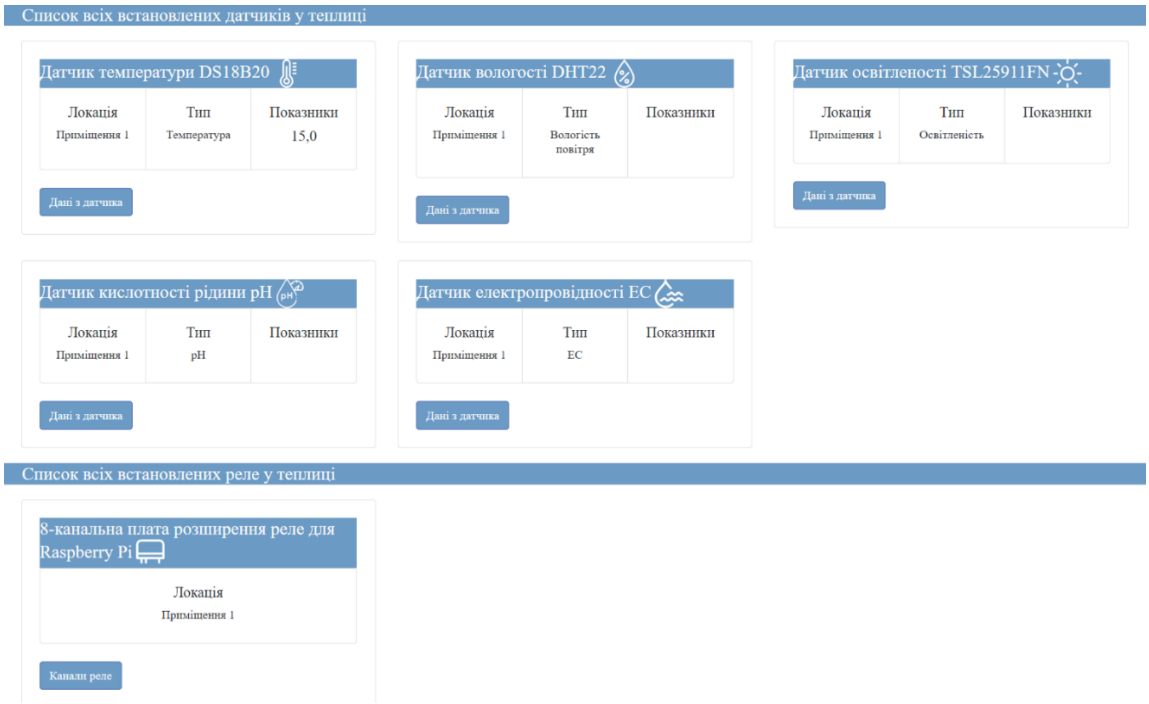

Рисунок 3.27. Сторінка зі список всіх встановлених датчиків

3.2. Інтерфейс програми адміністратора

Інтерфейс адміністратора – це вбудований модуль Django, який призначений для виконання операцій, пов'язаних з адмініструванням, що включають читання, запис, оновлення та видалення даних. За допомогою інтерфейсу, адміністратор може керувати користувачами та дозволами, пов'язаними з користувачами.

Для того, щоб відобразити таблиці в інтерфейсі адміністратора та виконувати дії з ними, необхідно в файлі admin.py імпортувати моделі з файлу models.py та викликати admin.site.register для реєстрації кожної моделі (рис. 3.28). Тобто models.py – визначає моделі бази даних, а admin.py – реєструє моделі в інтерфейсі адміністратора [37].

```
70  admin.site.register(Tulips, TulipSearch)
71  admin.site.register(Sensor_Type, Sensor_TypeAdmin)
72  admin.site.register(Sensor, SensorAdmin,)
73  admin.site.register(DataSensor, DataSensorAdmin)
74  admin.site.register(Sector,SectorAdmin)
75  admin.site.register(Planting_Sector, PlantingAdmin)
76  admin.site.register(Controller_Type,Controller_TypeAdmin)
77  admin.site.register(Controller, ControllerAdmin)
78  admin.site.register(DataController, DataControllerAdmin)
79  admin.site.register(Device, DeviceAdmin)
80  admin.site.register(ControllerPins, ControllerPinsAdmin)
```

Рисунок 3.28. Виклик та реєстрація моделей в файлі admin.py

Після цього для кожної таблиці необхідно створити клас, який наслідує клас admin.ModelAdmin. В класі можуть бути наступні об'єкти: list_display – для відображення полів таблиці, list_filter – для фільтрування даних в таблиці та search_fields – для пошуку даних в таблиці.

Натиснувши на кнопку Адміністрування в навігаційному меню, користувач потрапить до інтерфейсу адміністратора. Він дозволяє працювати з базою даних системи. Щоб адміністратор зміг зайти в систему, йому необхідно пройти авторизацію, заповнивши форму зображену на (рис. 3.29).

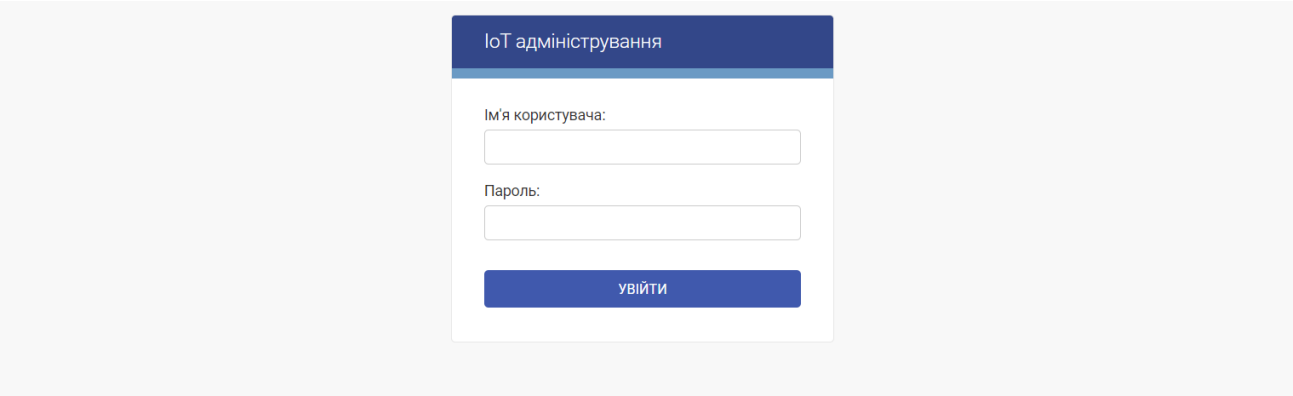

Рисунок 3.29. Форма авторизації

Після успішної авторизації, користувач побачить три логічні блоки: налаштування інтерфейсу адміністратора, головна частина, автентифікація та авторизація (рис. 3.30).

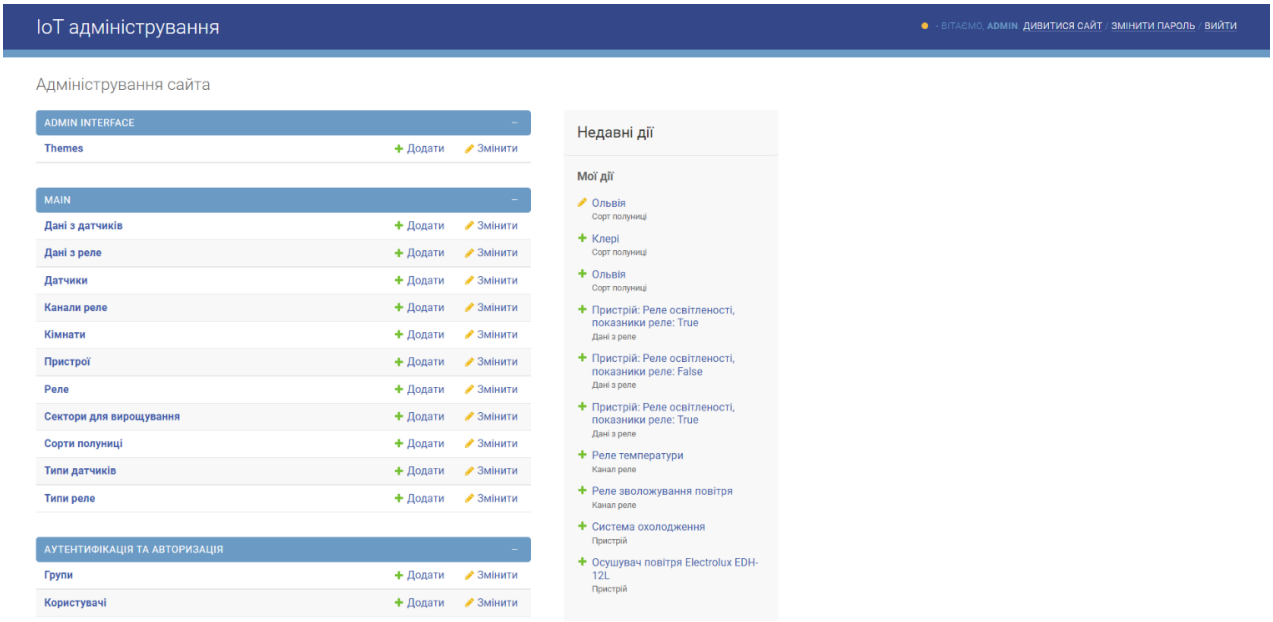

Рисунок 3.30. Сторінка адміністрування

При натисканні на назву таблиці, буде відкрито сторінка з відповідною таблицею бази даних (рис. 3.31). На ній відображено всі записи, що додано до таблиці. Користувач має змогу додати, редагувати та видаляти записи таблиць, відповідно до заданих прав.

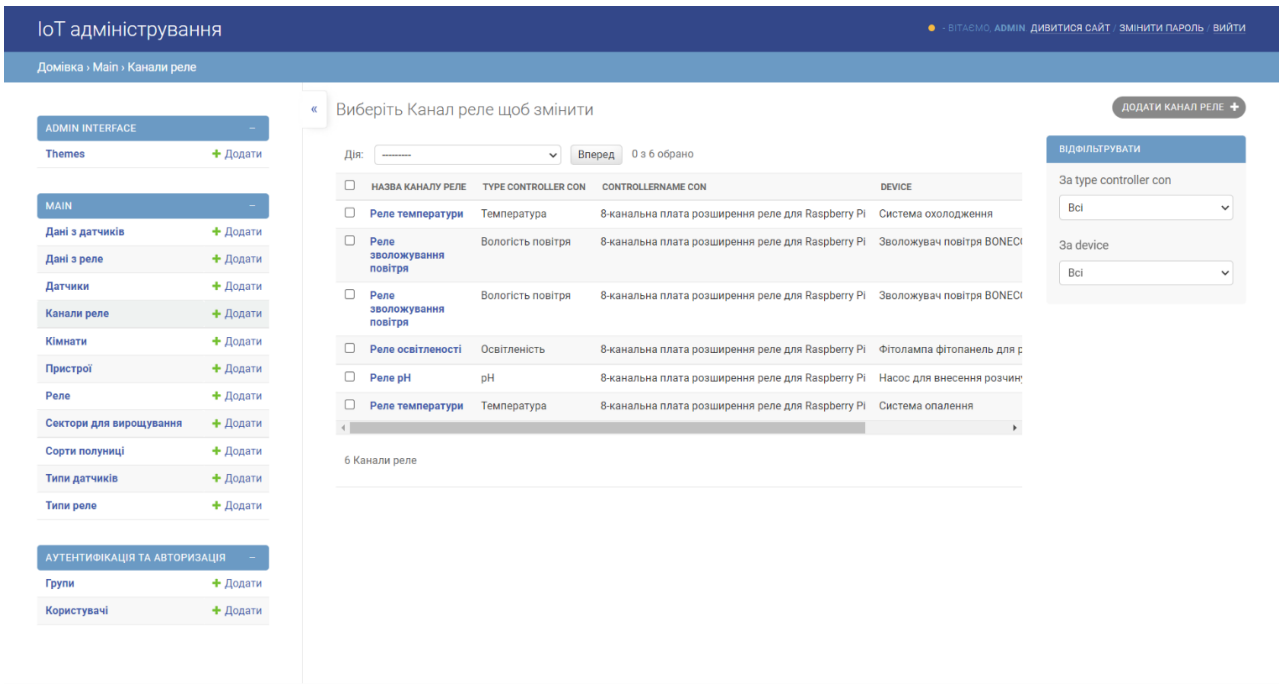

Рисунок 3.31. Сторінка перегляду даних таблиці

Для зручного досвіду користування таблицями додано фільтрацію записів (рис. 3.32). Вказавши необхідні параметри фільтрів, користувач побачить відфільтровані записи таблиць.

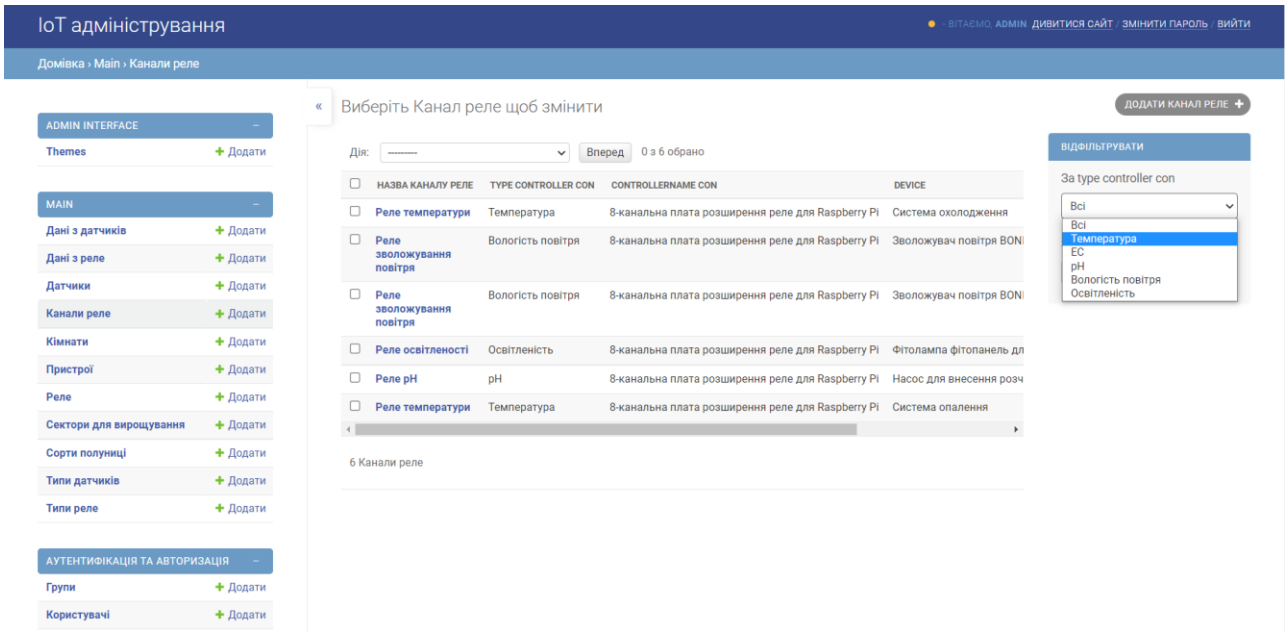

Рисунок 3.32. Фільтрація записів

Для додавання нового запису до таблиці бази даних, необхідно натиснути на кнопку Додати. Після цього відкриється форма заповнення даних (рис. 3.33).

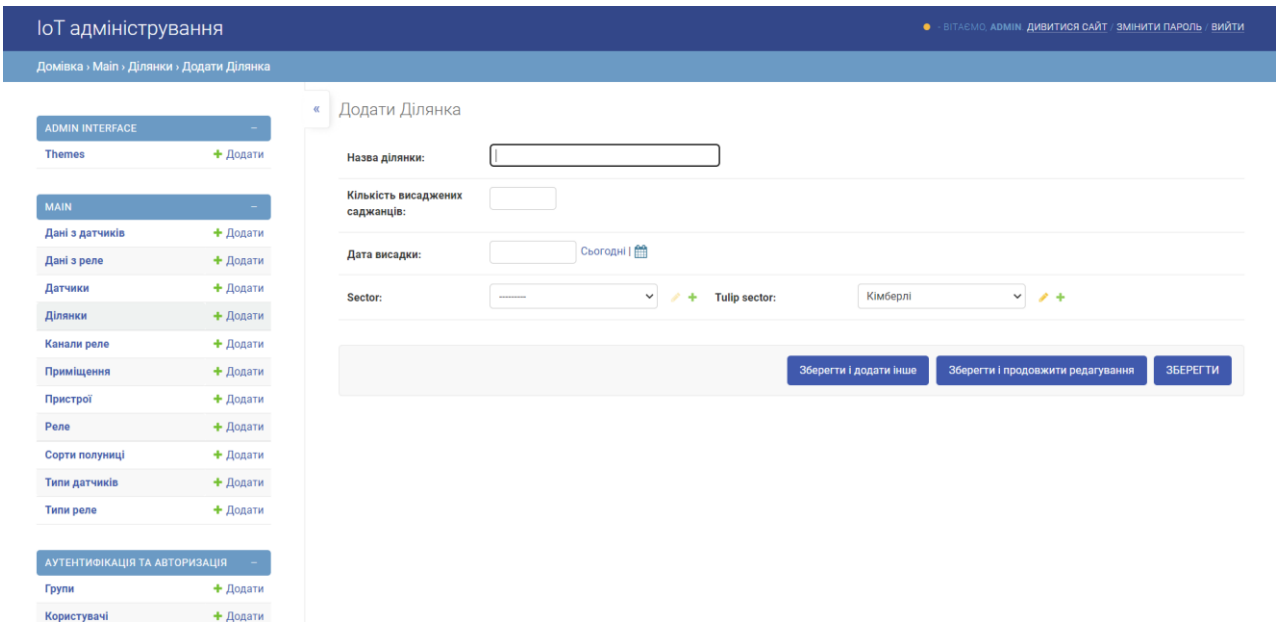

Рисунок 3.33. Форма додавання записів в таблицю

Для заповнення полів дати та часу користувач може застосувати, надані фреймворком графічні представлення (рис. 3.34), які запобігають виникненню помилок при введені даних.

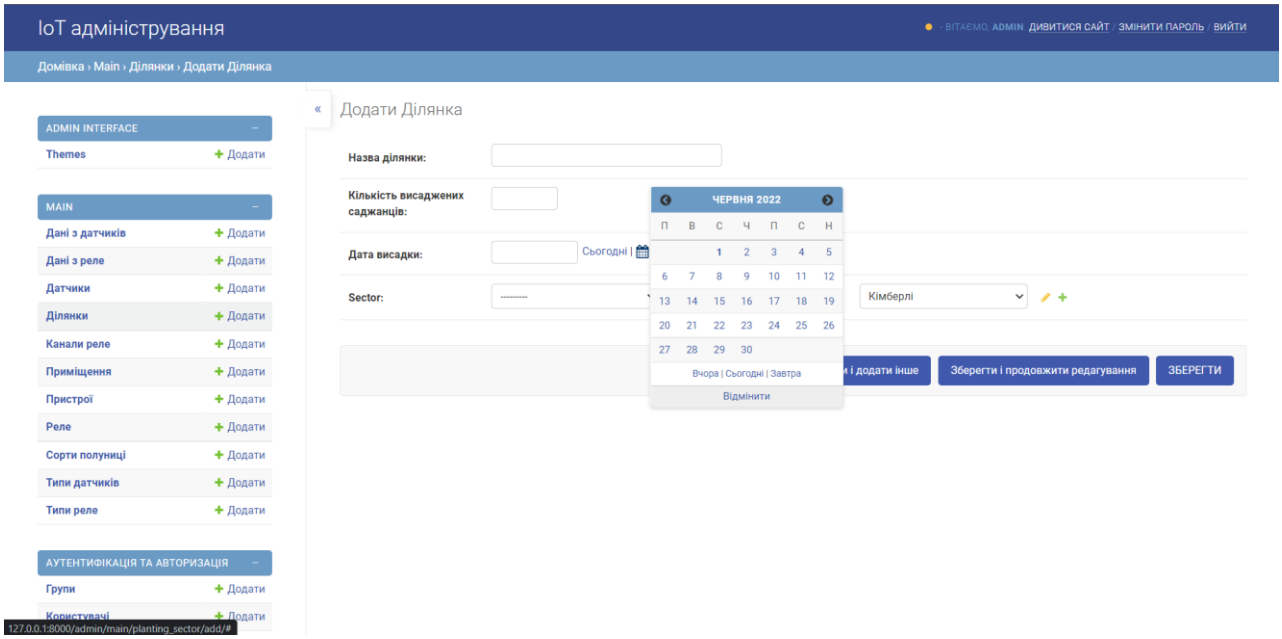

Рисунок 3.34. Графічні представлення для введення даних

Для форм введення даних присутня валідація, тому неможливо зберегти запис з некоректними даними (рис. 3.35).

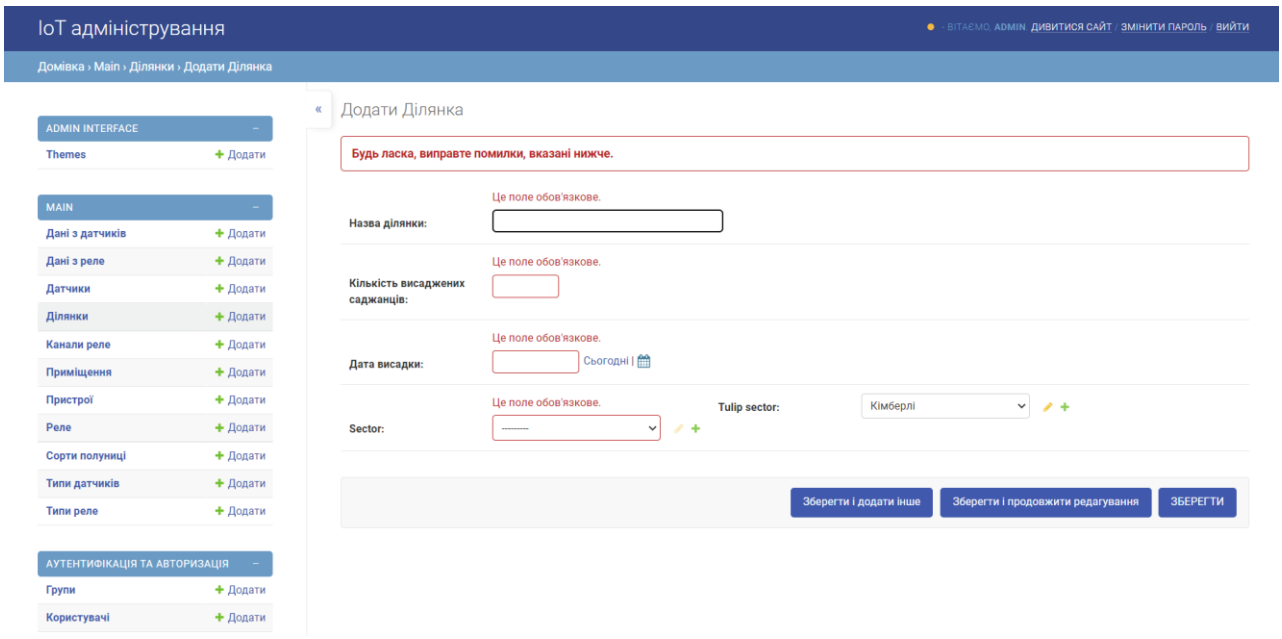

Рисунок 3.35. Валідація даних

Після додавання нового запису, дані відображаються на відповідні сторінці інтерфейсу адміністратора.

Для налаштування зовнішнього вигляду інтерфейсу адміністратора встановлено додатковий пакет – django-admin-interface. Він надає змогу користувачеві налаштувати кольорову гаму, яка йому до смаку [38]. Приклад налаштувань зображено на (рис. 3.36).

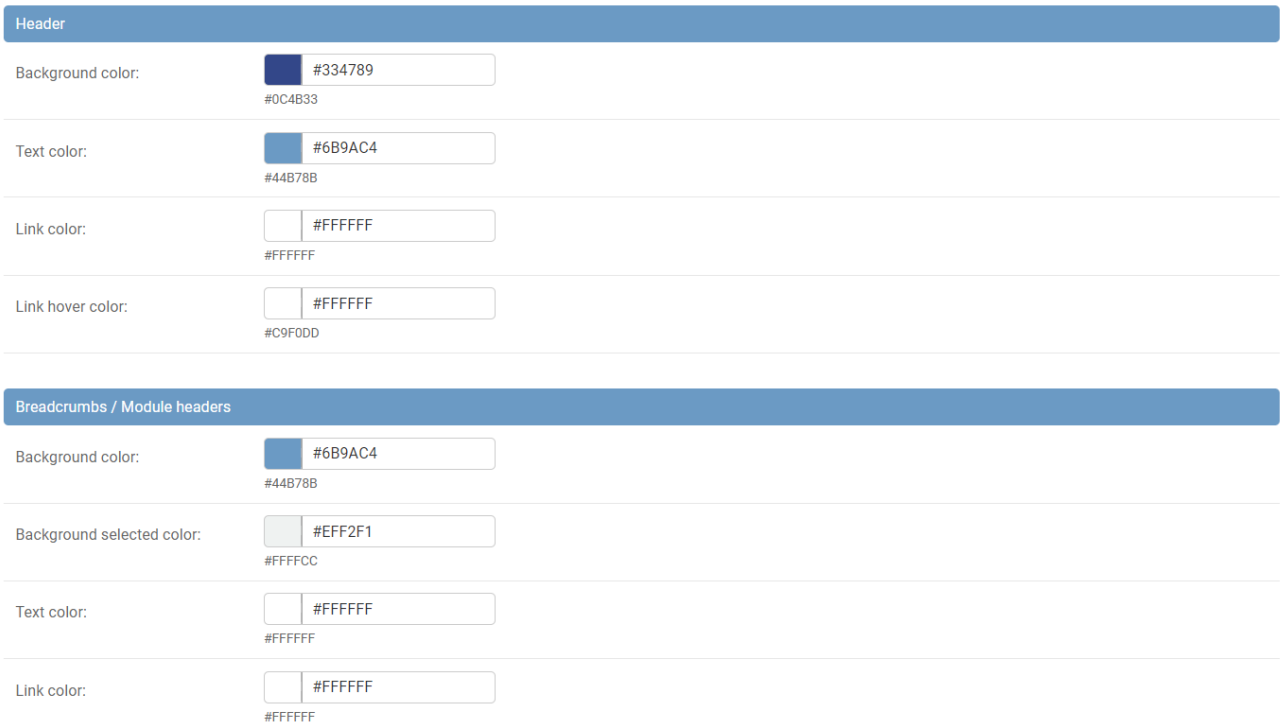

Рисунок 3.36. Налаштування зовнішнього вигляду інтерфейсу адміністратора

Щоб додати нового користувача системи, адміністратор має перейти на вкладку Користувачі та створити запис нового користувача (рис. 3.37). Для цього необхідно заповнити наступні поля, що відображено в табл 3.1:

Таблиця 3.1 – **Поля таблиці користувачі**

Поле	**Вимоги**
1	**2**
Ім’я користувача	• Необхідно: 150 або менше символів. • Тільки букви, цифри та знаки @/./+/-/_.
Пароль	• Пароль не може бути схожим особисту інформацію. • Пароль повинен містити як мінімум 8 символів. • Пароль не може бути одним із дуже поширених. • Пароль не може складається лише із цифр
Підтвердження пароля	• Ввести той же пароль, що і раніше, для підтвердження.

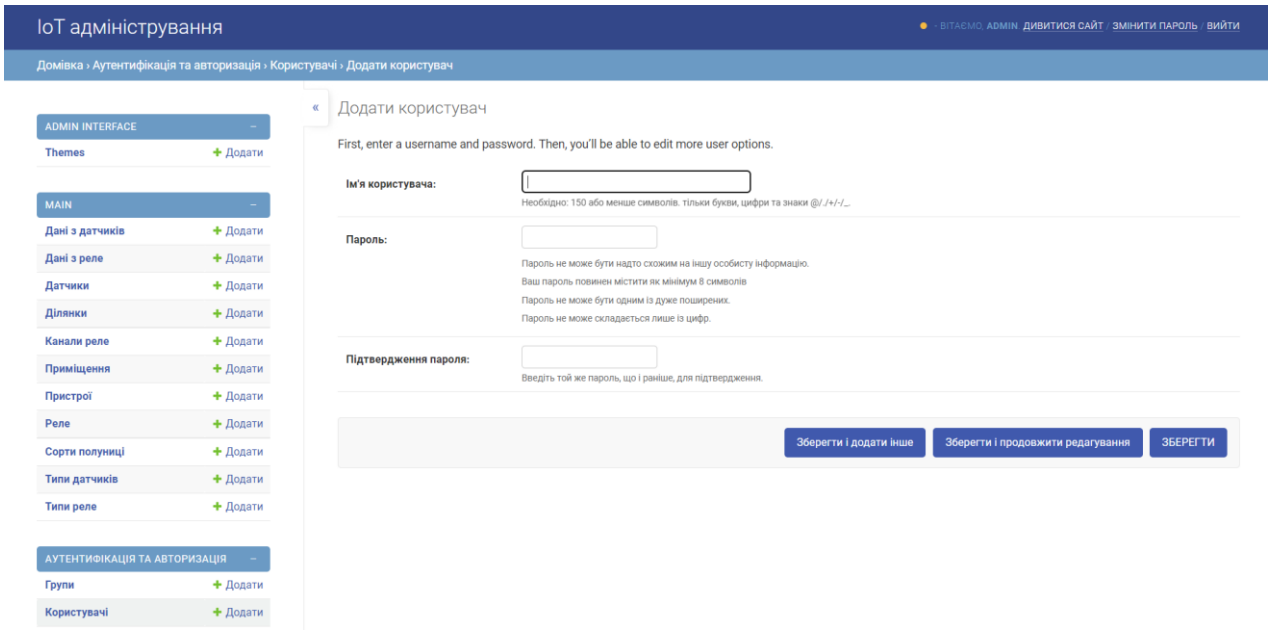

Рисунок 3.37. Форма створення нового користувача

Після натиснення кнопки Зберегти, з’явиться вікно для заповнення детальної інформації про користувача, зображеного на (рис. 3.38).

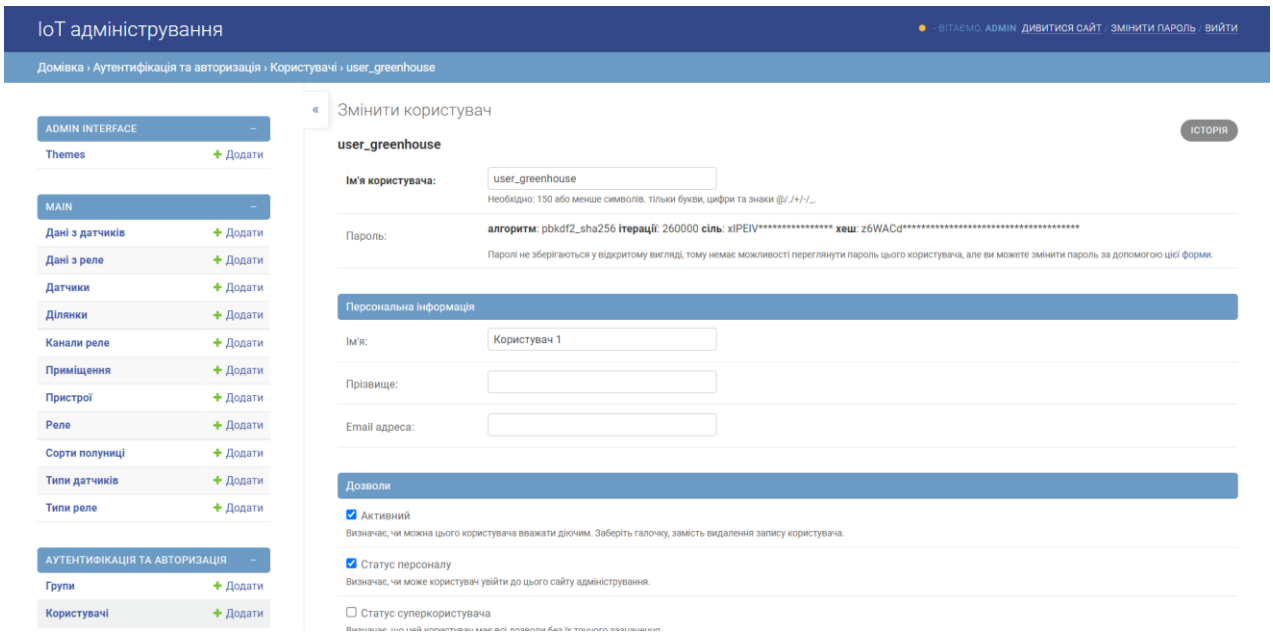

Рисунок 3.38. Форма додавання детальної інформації про користувача

В даній формі можна змінити ім’я користувача, вказати персональну інформацію: Ім’я, Прізвище, Email адреса; вказати дозволи: Активний – визначає чи можна цього користувача вважати діючим, Статус персоналу – визначає, чи може користувач увійти до цього сайту адміністрування; Статус суперкористувача – визначає, що цей користувач має всі дозволи без їх точного зазначення.

Крім того, в цьому вікні можна переглянути історію змін користувача, натиснувши кнопку Історія (рис. 3.39).

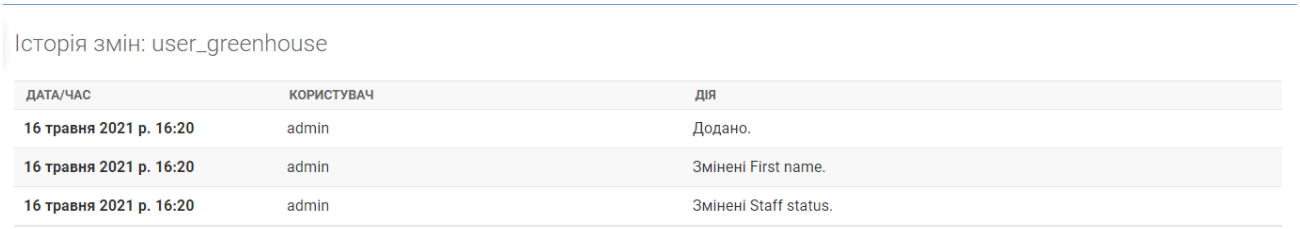

Рисунок 3.39. Історія змін користувача.

Також можна додати користувача, до групи, в яку він належить. Існує можливість додання індивідуальних дозволів для користувача.

3.3. Налаштування електронних листів IoT теплиці

Надсилання електронних листів є однією з важливих частин веб-додатку. Завдяки ним, користувач буде проінформованим у разі виникнення позаштатної ситуації.

В даному випадку, якщо датчик температури, освітленості, зволоженості повітря, кислотності рідини чи електропровідності зафіксує показник, який більше або менше норми, користувачеві буде надіслано лист на електронну пошту.

Для того, щоб надсилати користувачеві листи з веб-додатку, необхідно мати хост-сервер електронної пошти, в цьому випадку використовується SMTP-сервер Microsoft Outlook. Також необхідно мати робочий обліковий запис Microsoft Outlook. Щоб додати обліковий запис Microsoft Outlook до поштової програми, яка підтримує протокол SMTP, необхідно використати наведені нижче параметри сервера:

- Ім'я SMTP-сервера : smtp-mail.outlook.com.
- Порт SMTP : 587.
- Метод шифрування SMTP Starttls.

Далі необхідно виконати наступні налаштування в файлі проєкта settings.py. Вказано налаштування для надсилання електронної пошти в Django через Microsoft Outlook.

- EMAIL_HOST – параметр, що визначає постачальника послуг електронної пошти. В проєкті використовуються налаштування для Microsoft Outlook – smtp-mail.outlook.com.
- EMAIL_HOST_USER – обліковий запис, з якого надсилатимуться листи – greenhouse_bellamy@outlook.com.
- EMAIL_HOST_PASSWORD – пароль облікового запису електронної пошти.
- EMAIL_HOST_PORT – порт, який використовується SMTP сервером – 587.

- EMAIL_USE_TLS – параметр, що визначає чи буде використовувати електронна пошта з'єднання TLS [40] – криптографічний протокол, який надає можливості безпечної передачі даних в інтернеті для навігації, отримання пошти, спілкування, обміну файлами, тощо. Протокол використовує асиметричне шифрування і сертифікати X.509 [41].

Наведена нижче конфігурація (рис. 3.40) застосовується для відправки електронних листів користувачеві.

```
133  EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
134  DEFAULT_FROM_EMAIL = 'greenhouse_bellamy@outlook.com'
135  SERVER_EMAIL = 'greenhouse_bellamy@outlook.com'
136  EMAIL_USE_TLS = True
137  EMAIL_HOST = 'smtp-mail.outlook.com'
138  EMAIL_PORT = 587
139  EMAIL_HOST_USER = 'greenhouse_bellamy@outlook.com'
140  EMAIL_HOST_PASSWORD = '[redacted]'
```

Рисунок 3.40. Конфігурація для відправки електронних сповіщень

Виконавши необхідні налаштування, перейдемо до безпосередньої відправки електронних сповіщень IoT теплиці.

Для цього з модуля django.core.mail імпортуємо метод send_mail(), що оброблює передачу електронних листів. Даний метод має наступні параметри:

- subject – містить тему електронного листа.
- message – містить текст електронного листа.
- from_email – електронна адреса відправника, параметр має бути таким же, як було визначено в файлі settings.py.
- recipient_list – список електронних адрес відправників.
- fail_silently – необов'язковий параметр, що викликає виключення, в разі якщо сповіщення не відправлено [42].

Метод send_mail() в тексті електронного листа матиме наступні параметри (рис. 3.41), де:

- sensorname.titlesensor – назва датчика, що зафіксував зафіксує показник, що є більше або менше норми

- sensorname.sector – місце розташування датчика
- datafromPrev.measurement – показник, що зафіксував датчик
- sensorname.type_sensors.units – одиниці виміру
- datafromPrev.dateofmeasurement – дата та час, коли зафіксовано показник

```
send_mail("Сповіщення IoT теплиці",
          f'{sensorname.titlesensor}, що розміщений в {sensorname.sector}'
          f', зафіксував наступні показники {datafromPrev.measurement}{sensorname.type_sensors.units}'
          f' в {datafromPrev.dateofmeasurement}, які нижче норми.',
          "dronesmuse23@gmail.com",
          ["maxbellamy23@gmail.com"],
          fail_silently=False)
```

Рисунок 3.41. Метод send_mail()

Для відправлення сповіщень відбувається перевірка на відповідність показників згідно з табл. 3.2 наведеною нижче. Сповіщення IoT теплиці будуть надсилатись коли один з датчиків зафіксує показники, які є вище або нижче норми.

Таблиця 3.2 – **Умови вирощування полуниці**

Тип датчика	**Нормальні показники**	**Показники нижче норми**	**Показники вище норми**
1	**2**	**3**	**4**
Температура	$16 \leq t \leq 20$	$16 < t$	$t > 20$
Вологість повітря	$65 \leq \phi \leq 75$	$65 < \phi$	$\phi > 75$
Кислотність рідини	$5.8 \leq pH \leq 6.2$	$5.8 < pH$	$pH > 6.2$
Освітленість	$10000 \leq E \leq 20000$	$10000 < E$	$E > 20000$
Електропровідність рідини	$2 \leq EC \leq 3.2$	$2 < EC$	$EC > 3.2$

Надсилання електронних листів буде здійснюватися за наступним алгоритмом, роботу якого зображено на (рис. 3.42)

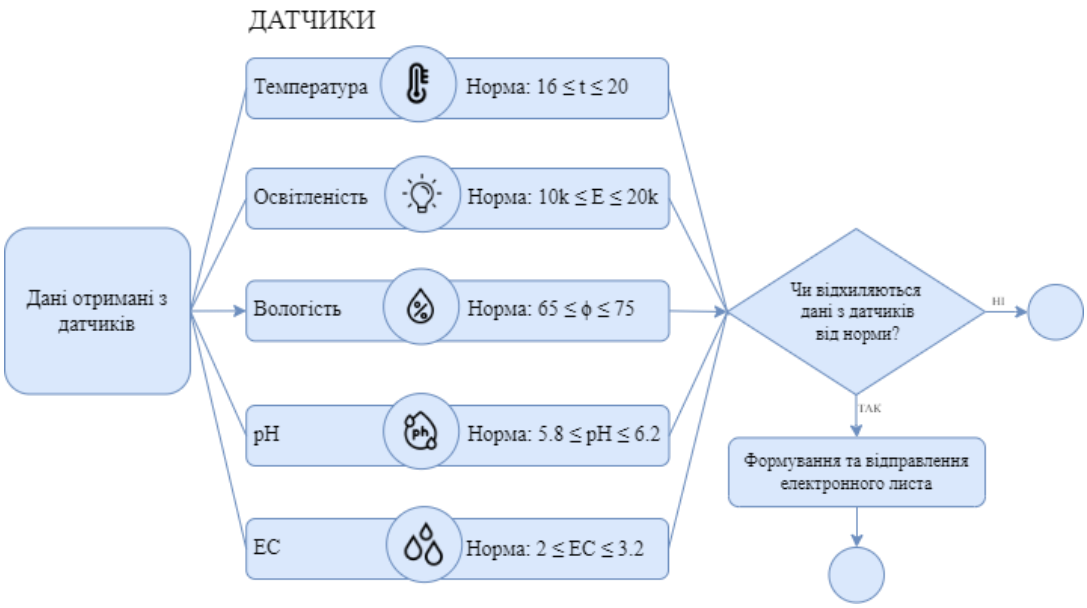

Рисунок 3.42. Алгоритм надсилання електронних листів

В результаті з пошти EMAIL_HOST_USER буде відправлено наступний лист на пошту recipient_list отримувача (рис. 3.43).

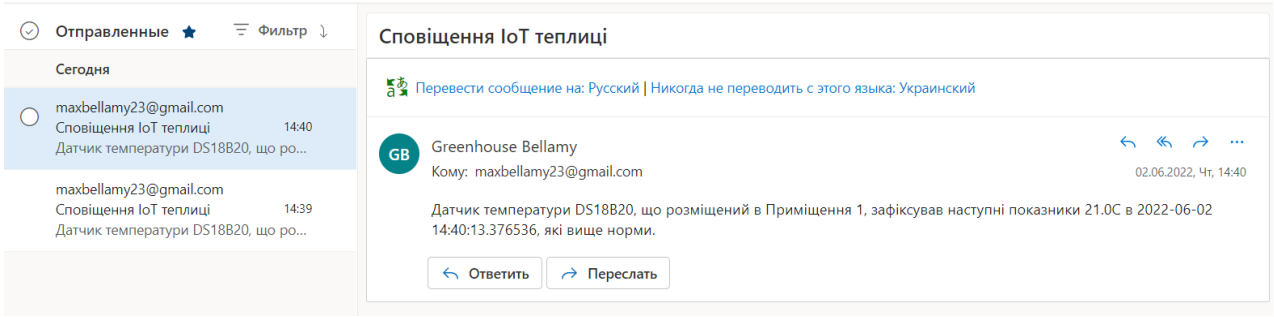

Рисунок 3.43. Відправлений лист IoT теплиці

В результаті користувач отримає наступне сповіщення (рис. 3.44). Датчик температури DS18B20, що розміщений в Приміщення 1, зафіксував наступні показники 21.0C в 2022-06-02 14:47:32.872033, які вище норми.

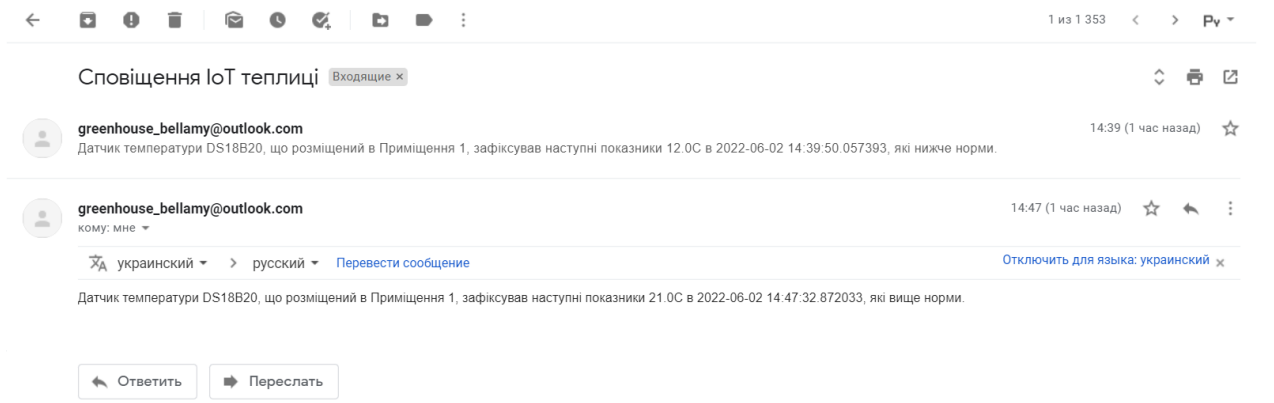

Рисунок 3.44. Отриманий лист IoT теплиці

3.4. Візуалізація отриманих даних

Візуалізація даних – графічне представлення інформації та даних. Використовуючи візуальні елементи такі як діаграми, графіки та гістограми користувач додатку зможе відслідковувати дані, отримані з датчиків в режимі реального часу.

Для відображення даних про рослин, що вирощуються на ділянках було створено гістограму. Вона дозволяє відобразити значення даних у вигляді вертикальних ліній.

Гістограму було побудовано за допомогою ChartJs – бібліотека з відкритим вихідним кодом, яка дозволяє візуалізувати дані за допомогою JavaScript. Бібліотека підтримує 8 типів діаграм, включаючи гістограми, лінійні та кругові діаграми. Для цього було створено метод planting_sector(), у якому створено об’єкт planting_sectors, що містить всі об’єкти класу Planting_Sector. Далі на сторінці, де відображається графік створено скрипт, в якому обрано тип графіку, а також надано дані та параметри для побудови, а саме:

- type: 'bar' – тип графіку, гістограма.
- labels: [{% for i in planting_sectors %}'{{i.title}}',{% endfor %}] – назви ділянок
- label: 'Кількість рослин, що висаджено' – легенда графіка.
- data: [{% for i in planting_sectors %}'{{i.number_of_planted}}',{% endfor %}] – кількість рослин, що висаджено на кожній ділянці.
- backgroundColor: ['rgba(107,154,196, 0.2)'] – колір стовпців графіка.
- borderColor: ['rgba(107,154,196, 1)'] – колір обрамлення.
- borderWidth: 1 – товщина обрамлення [43].

В результаті буде отримано наступний графік, зображений на (рис. 3.45).

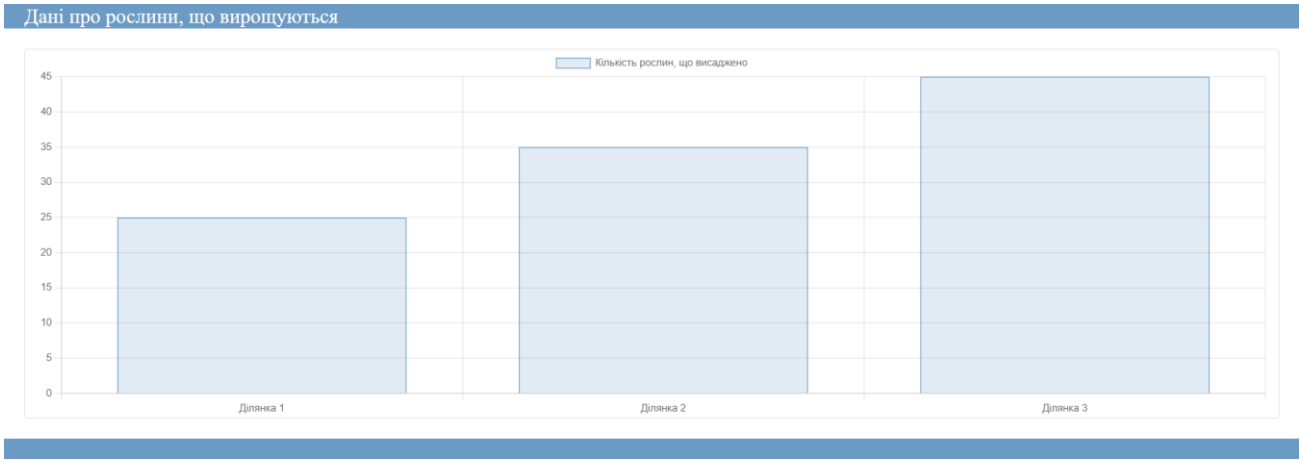

Рисунок 3.45. Гістограма з кількістю рослин, що вирощується

Таким ж чином було побудовано гістограму з даними про очікуваний врожай з ділянки (рис. 3.46).

Рисунок 3.46. Гістограма з даними про очікуваний врожай з ділянки

Для більш зручного досвіду взаємодії з гістограмою було додано підказки при наведенні на один зі стовпчиків гістограми (рис. 3.47).

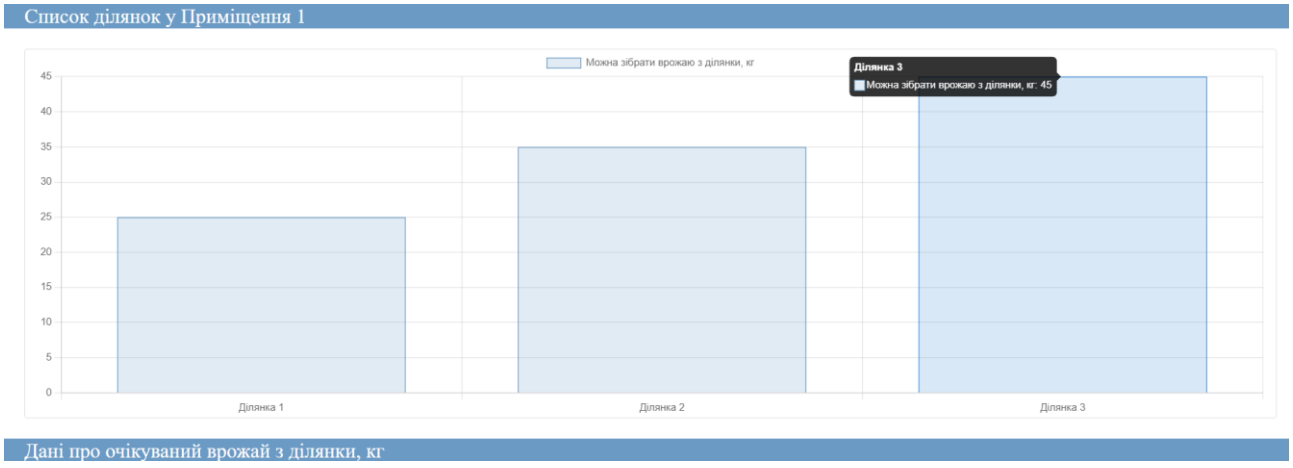

Рисунок 3.47. Підказка при наведенні на гістограму

Для відображення даних з датчику за останню годину побудовано лінійний графік, використовуючи ChartJs. З цією метою створено метод datafromsensors(), у якому створено об’єкти sensorname та datafrom1. Sensorname – містить всі об’єкти класу Sensor, а також приймає метод get з параметром id=id. Datafrom1 – містить всі об’єкти класу DataSensor, які були відфільтровані за допомогою метода filter() з параметрами sensorname=id, dateofmeasurement__gte = datetime.now() - timedelta(hours=1). Таким чином кожен графік буде відображати дані лише ті дані, що належать одному датчику за останню годину.

Після фільтрації даних, створено скрипт, в якому обрано тип графіку, а також надано дані та параметри для побудови, а саме:

- type: 'line' – тип графіку, гістограма.
- labels: [{% for i in datafrom1 %}'{{i.dateofmeasurement}}',{% endfor %}– дата та час, коли отримано показники.
- label: 'Дані з датчику за останню годину'– легенда графіка.
- data: {% for i in datafrom1 %}'{{i.measurement}}',{% endfor %}] – показники отримані з датчика.
- backgroundColor: ['rgba(107,154,196, 0.2)'] – колір точок графіка.
- borderColor: ['rgba(107,154,196, 1)'] – колір лінії.
- borderWidth: 1 – товщина лінії графіка [44].

Для побудови індикатора, що відображає останній отриманий показник було імпортовано графічну бібліотеку з відкритим вихідним кодом Plotly. В методі datafromsensors(), створено об’єкт datafromlast – містить всі об’єкти класу DataSensor, які були відфільтровані за допомогою метода filter() з параметрами sensorname=id, з використанням метода last(), який повертає останній об’єкт з наборі запитів. Для безпосередньої побудови графіка викликано метод Indicator(), з наступними параметрами:

- mode="gauge+number" – тип графіка, індикатор.
- value=datafromast.measurement – останнє отримане значення з датчика.
- domain={'x': [0, 1], 'y': [0, 1]} – координати розташування графіка

- gauge={'bar': {'color': "#6B9AC4"}} – колір графіка
- title={'text': "Показник датчика"} – заголовок графіка [45]

Побудований лінійний графік та індикатор зображено на рис. 3.48.

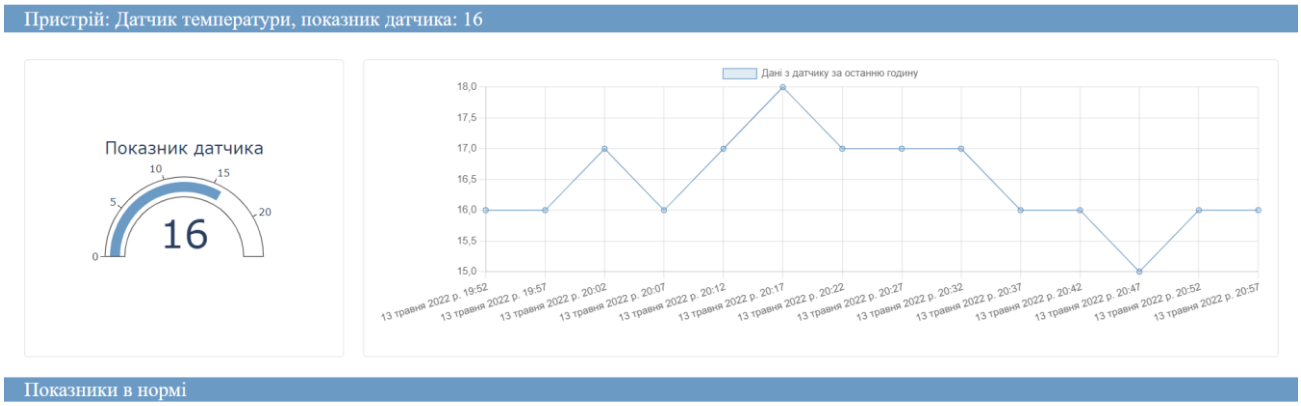

Рисунок 3.48. Лінійний графік, індикатор для відображення даних, отриманих з датчика

3.5. Реагування системи на недопустимі значення показників

Якщо датчик температури, освітленості, зволоженості повітря, кислотності рідини або електропровідності рідини зафіксує показник, що є більше або менше норми, автоматично спрацьовуватиме тригер.

Тригер – це керована подіями дія, яка виконується автоматично, коли певна операція зміни виконується в таблиці [46]. В даному випадку тригер застосовується для контролю за показниками з датчиків та реагуванням на недопустимі значення. В загальному випадку структура тригера в SQLite виглядає наступним чином (рис. 3.49):

```
CREATE TRIGGER [IF NOT EXISTS] trigger_name
   [BEFORE|AFTER|INSTEAD OF] [INSERT|UPDATE|DELETE]
   ON table_name
   [WHEN condition]
BEGIN
 statements;
END;
```

Рисунок 3.49. Загальна структура тригера

1. Спочатку вказуються назва тригера після ключових слів CREATE TRIGGER.
2. Далі визначається, коли буде спрацьовувати тригер до чи після вставлення, оновлення або видалення запису з таблиці [BEFORE|AFTER|INSTEAD OF] [INSERT|UPDATE|DELETE].
3. Потім вказується назва таблиці ON table_name, в якій має спрацювати тригер.
4. Після цього вказується подія [WHEN condition], яка спричиняє виклик тригера.
5. В блок BEGIN END поміщається логіка тригера [47].

Для доступу до даних рядка, який вставляється, оновлюється чи видаляється, використовуються посилання OLD та NEW у формі: OLD.column_name та NEW.column_name. Табл. 3.3 ілюструє дані правила:

Таблиця 3.3 – **Правила роботи OLD та NEW**

Дія	**Посилання**
1	**2**
INSERT	NEW
UPDATE	NEW, OLD
DELETE	OLD

Для реагування системи на недопустимі значення показників було реалізовано тригер check_measurement_temperature_high (рис. 3.50) для таблиці main_datasensor з наступним алгоритмом роботи:

- Після додавання нового запису до таблиці main_datasensor відбуватимуться наступні перевірки: новий запис належить датчику у якого тип датчику температура, а також чи додане значення є допустим для даного датчика.
- Якщо виконується умова додається новий запис про роботу каналу реле відповідного типу, який вмикає пристрій з необхідним режимом роботи, а

також через вказаний час, про його припинення та робиться новий запис до таблиці main_datasensor.

```
CREATE TRIGGER check_measurement_temperature_high
after INSERT on main_datasensor
when  (new.measurement > 20) AND
new.sensorname_id == (select DISTINCT md.sensorname_id from main_datasensor as md
inner join main_sensor as ms
inner join main_sensor_type as mst
on mst.id = ms.type_sensors_id and ms.id = md.sensorname_id
where mst.type == 'Температура')
BEGIN

  INSERT into main_datasensor(dateofmeasurement,sensorname_id,measurement)
  VALUES(datetime('now', '+10 minutes'),new.sensorname_id,18);

  INSERT into main_datacontroller(measurement_con,measurement_reg,dateofmeasurement_con,controllername_con_id)
  VALUES(1,0,datetime('now'),(select mc.id from main_controllerpins as mc
  inner join main_controller_type as mct
  inner join main_sensor as ms
  inner join main_sensor_type as mst
  on mc.type_controller_con_id=mct.id and ms.type_sensors_id=mst.id
  where mct.type_con == mst.type and mst.type=='Температура' and mct.type_con=='Температура'));

  INSERT into main_datacontroller(measurement_con,measurement_reg,dateofmeasurement_con,controllername_con_id)
  VALUES(0,0,datetime('now','+10 minutes'),(select mc.id from main_controllerpins as mc
  inner join main_controller_type as mct
  inner join main_sensor as ms
  inner join main_sensor_type as mst
  on mc.type_controller_con_id=mct.id and ms.type_sensors_id=mst.id
  where mct.type_con == mst.type and mst.type=='Температура' and mct.type_con=='Температура'));

end;
```

Рисунок 3.50. – Тригер check_measurement_temperature_high

Таким чином було реалізовано наступні тригери: check_measurement_lightning_high та check_measurement_lightning_low – для показників освітленості, check_measurement_humidity_high та check_measurement_humidity_low – для показників вологості, check_measurement_temperature_high та check_measurement_temperature_low – для показників, check_measurement_pH_high та check_measurement_pH_low – для показників кислотності рідини, check_measurement_ec_high та check_measurement_ec_low – для показників електропровідності рідини.

Для імітації роботи датчика температури було створено метод (рис. 3.51) generate_temperature(), який випадковим чином додає нові записи до таблиці main_datasensor.

```
def generate_temperature():
    x = random.choice([16, 17, 18, 19, 20], p=[0.15, 0.15, 0.3, 0.2, 0.2], size=(1))
    for i in range(0,24):
        room = DataSensor(dateofmeasurement=datetime.now(), sensorname_id=3, measurement=x)
        room.save()
        time.sleep(300)

generate_temperature()
```

Рисунок 3.51. Метод generate_temperature()

3.6. Висновок до третього розділу

Отже, в даному розділі було написано програмний код для роботи системи мінерального живлення рослин.

- Створено інтерфейс програми користувача, в якому передбачено наступні сторінки: дашборд, приміщення теплиці, каталог сортів полуниці та список всіх встановлених пристроїв в теплиці. Дашборд відображає інформацію про кількість сортів рослин, що вирощуються, кількість ділянок, кількість встановлених датчиків та реле, а також мінімум, в середньому, максимум та суму висаджених рослин. Також в дашборді зображено гістограму, що відображає ділянку та кількість висаджених рослин на ній. Сторінку приміщення теплиці створено для перегляду інформації про наявні приміщення, а саме ділянки, встановлені датчики та реле. Сторінки каталог сортів та список всіх встановлених пристроїв в теплиці створені реалізовано для зручного та швидкого перегляду даних.
- Реалізовано інтерфейс програми адміністратора, який призначений для виконання операцій пов’язаних з адмініструванням, що включає читання, запис, оновлення та видалення даних.
- Налаштовано розсилку електронних листів, у разі виникнення позаштатної ситуації.
- Для отриманих даних з датчиків виконано візуалізацію даних за допомогою бібліотек ChartJs та Python Plotly Library, а саме побудовано гістограми, лінійні графіки та індикатор для відображення даних.
- Створено тригери в SQLite для реагування системи на недопустимі значення показників.

ВИСНОВОК

У результаті виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано IoT систему мінерального живлення рослин. Перед тим як почати розробку було описано мету, актуальність, а також практичне значення одержаних результатів.

У першому розділі здійснено аналітичний огляд концепції гідропоніки. Проаналізовано умови вирощування полуниці за допомогою гідропоніки та досліджено переваги й недоліки використання гідропоніки. Виконано порівняння існуючих проектів в даній сфері – програмне забезпечення для моніторингу та контролю за показниками теплиці.

У другому розділі підібрано обладнання, яке підійшло би для створення системи мінерального живлення рослин. Обрано одноплатний комп'ютер Raspberry Pi 4, який лежить основі системи. Підібрано датчики температури та вологості повітря, освітленості у приміщені, кислотності та електропровідності рідини. Обрано плату розширення реле для Raspberry Pi для того щоб керувати пристроями високої напруги, 220В. Також створено базу даних для системи мінерального живлення рослин, розробленого логічну та фізичну моделі бази даних з повним описом складових даних моделей. Виконано опис засобів реалізації, а саме: обрано фреймворк для створення веб-застосунків – Django, описано процес створення моделей класів та підібрано бібліотеки для візуалізації отриманих значень з датчиків – Python Plotly Library та Chart.js.

У третьому розділі написано програмний код для роботи системи мінерального живлення рослин. Створено інтерфейси користувача та адміністратора для взаємодії з програмним забезпеченням. Налаштовано розсилку електронних листів, у разі виникнення позаштатної ситуації. Виконано аналіз та візуалізацію отриманих даних з датчиків та реалізовано реагування системи на недопустимі значення показників, отриманих з датчиків.

Під час виконання роботи, всі задачі були виконані, реалізовано IoT систему мінерального живлення рослин.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Advantages & Disadvantages of Hydroponics веб-сайт. URL: https://www.trees.com/gardening-and-landscaping/advantages-disadvantages-of-hydroponics (дата звертання 19.05.2022)
2. Hydroponic Farming is the Future веб-сайт. URL: https://greenourplanet.org/hydroponics/benefits-of-hydroponics/#:~:text=When%20Compared%20To%20Traditional%20Soil,a%20well%20managed%20hydroponic%20system (дата звертання 19.05.2022)
3. Benefits of IoT in agriculture and smart farming веб-сайт. URL: https://www.xcubelabs.com/blog/benefits-of-iot-in-agriculture-and-smart-farming/ (дата звертання 19.05.2022)
4. What is the internet of things (IoT)? веб-сайт. URL: https://www.twi-global.com/technical-knowledge/faqs/what-is-the-internet-of-things-iot (дата звертання 20.05.2022)
5. Changes in Hydroponic Water Additives That Enhance Fruiting and Flavor of Strawberries веб-сайт. URL: https://strawberryplants.org/hydroponic-water-additives-enhance-flavor/#what-makes-a-strawberry-sweet (дата звертання 20.05.2022)
6. 10 Benefits of a Smart Agriculture Solution веб-сайт. URL: https://www.c2m.net/blog/10-benefits-of-a-smart-agriculture-solution (дата звертання 20.05.2022)
7. Measuring Brix веб-сайт. URL: https://vincentcorp.com/content/measuring-brix/#:~:text=It%20can%20be%20calculated%20by,the%20equation%20for%20calculating%20Brix (дата звертання 20.05.2022)
8. iFarm Growtune - Control Centre of your vertical farm веб-сайт. URL: https://growtune.com/ (дата звертання 20.05.2022)
9. Agrowtek веб-сайт. URL: https://www.agrowtek.com/ (дата звертання 20.05.2022)
10. Raspberry Pi 4 Model B веб-сайт. URL: https://micro-pi.ru/raspberry-pi-4-model-b-rpi-4-b-bcm2711/ (дата звертання 20.05.2022)

11. Raspberry pi 4 GPIO pinout and guide for beginners веб-сайт. URL: https://nerdytechy.com/raspberry-pi-4-gpio-pinout/ (дата звертання 20.05.2022)
12. Waterproof DS18B20 Digital Temperature Sensor веб-сайт. URL: https://miniboard.com.ua/sensors/518-vlagozasshisshennyj-datchik-temperatury-ds18b20.html (дата звертання 21.05.2022)
13. DHT22 веб-сайт. URL: https://arduino.ua/prod301-datchik-vlajnosti-i-temperatyri-dht22 (дата звертання 21.05.2022)
14. CQRobot Ocean: TSL25911FN Ambient Light Sensor веб-сайт. URL: https://www.amazon.com/CQRobot-Ambient-Compatible-Raspberry-TSL25911FN/dp/B083KM51DF (дата звертання 21.05.2022)
15. Liquid PH Value Detection веб-сайт. URL: https://www.amazon.com/Detection-Monitoring-Control-Arduino-Electrode/dp/B07K8TBBFH/ref=sr_1_2?keywords=pH+Sensor+Arduino&qid=1653857390&sr=8-2 (дата звертання 21.05.2022)
16. Grove-EC Sensor Kit веб-сайт. URL: https://thepihut.com/products/8-channel-relay-expansion-board-for-raspberry-pi (дата звертання 21.05.2022)
17. How to use 5v relays with the raspberry pi веб-сайт. URL: https://www.circuitbasics.com/5v-relays-in-the-raspberry-pi/ (дата звертання 21.05.2022)
18. 8-Channel Relay Expansion Board for Raspberry Pi веб-сайт. URL: https://www.aliexpress.com/item/1005002992703788.html (дата звертання 21.05.2022)
19. Logical data models веб-сайт. URL: https://www.ibm.com/docs/en/datastudio/4.1.1?topic=SS62YD_4.1.1/com.ibm.datatools.logical.ui.doc/topics/clogmod.html (дата звертання 23.05.2022)
20. Physical Data Model веб-сайт. URL: https://www.1keydata.com/datawarehousing/physical-data-model.html (дата звертання 23.05.2022)

21. Basics of Django framework веб-сайт. URL: https://programmerprodigy.code.blog/2020/07/09/basics-of-django-framework/ (дата звертання 23.05.2022)
22. Вступ до Django веб-сайт. URL: https://metanit.com/python/django/1.1.php (дата звертання 23.05.2022)
23. Django веб-сайт. URL: https://uk.wikipedia.org/wiki/Django (дата звертання 23.05.2022)
24. Object–relational mapping веб-сайт. URL: https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping (дата звертання 23.05.2022)
25. Django Tutorial Part 3: Using models веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Models (дата звертання 23.05.2022)
26. Створення першої програми Django веб-сайт. URL: https://metanit.com/python/django/1.4.php (дата звертання 23.05.2022)
27. PyCharm веб-сайт. URL: https://uk.wikipedia.org/wiki/PyCharm (дата звертання 23.05.2022)
28. Bootstrap веб-сайт. URL: https://uk.wikipedia.org/wiki/Bootstrap (дата звертання 23.05.2022)
29. Plotly Open Source Graphing Library for Python веб-сайт. URL: https://plotly.com/python/ (дата звертання 23.05.2022)
30. About draw.io веб-сайт. URL: https://github.com/jgraph/drawio (дата звертання 23.05.2022)
31. What is a Dashboard? веб-сайт. URL: https://www.adjust.com/glossary/dashboard/ (дата звертання 15.05.2022)
32. Cards веб-сайт. URL: https://getbootstrap.com/docs/4.0/components/card/ (дата звертання 15.05.2022)
33. Bootstrap Grid System веб-сайт. URL: https://www.w3schools.com/bootstrap/bootstrap_grid_system.asp (дата звертання 15.05.2022)

34. Spacing веб-сайт. URL: https://getbootstrap.com/docs/4.0/utilities/spacing/ (дата звертання 15.05.2022)
35. Jinja веб-сайт. URL: https://jinja.palletsprojects.com/en/3.0.x/ (дата звертання 13.05.2022)
36. QuerySet API reference веб-сайт. URL: https://docs.djangoproject.com/en/4.0/ref/models/querysets/ (дата звертання 13.05.2022)
37. The Django admin site веб-сайт. URL: https://docs.djangoproject.com/en/3.0/ref/contrib/admin/ (дата звертання 18.05.2022)
38. Django-admin-interface 0.19.1 веб-сайт. URL: https://pypi.org/project/django-admin-interface/ (дата звертання 18.05.2022)
39. Sending email веб-сайт. URL: https://docs.djangoproject.com/en/4.0/topics/email/ (дата звертання 12.05.2022)
40. How to Send Emails in Django веб-сайт. URL: https://data-flair.training/blogs/django-send-email/ (дата звертання 12.05.2022)
41. Transport Layer Security веб-сайт. URL: https://en.wikipedia.org/wiki/Transport_Layer_Security (дата звертання 12.05.2022)
42. How to send emails with python django through google SMTP server for free веб-сайт. URL: https://bshoo.medium.com/how-to-send-emails-with-python-django-through-google-smtp-server-for-free-22ea6ea0fb8e (дата звертання 12.05.2022)
43. Bar Chart веб-сайт. URL: https://www.chartjs.org/docs/latest/charts/bar.html (дата звертання 13.05.2022)
44. Line Chart веб-сайт. URL: https://www.chartjs.org/docs/latest/charts/line.html (дата звертання 13.05.2022)
45. Gauge Charts in Python веб-сайт. URL: https://plotly.com/python/gauge-charts/ (дата звертання 13.05.2022)

46. SQLite Triggers веб-сайт. URL: https://www.w3resource.com/sqlite/sqlite-triggers.php (дата звертання 13.05.2022)
47. SQLite Trigger веб-сайт. URL: https://www.sqlitetutorial.net/sqlite-trigger/#:~:text=What%20is%20an%20SQLite%20trigger,issued%20against%20the%20associated%20table (дата звертання 13.05.2022)

ДОДАТОК А

Рисунок А.1. Слайд 1

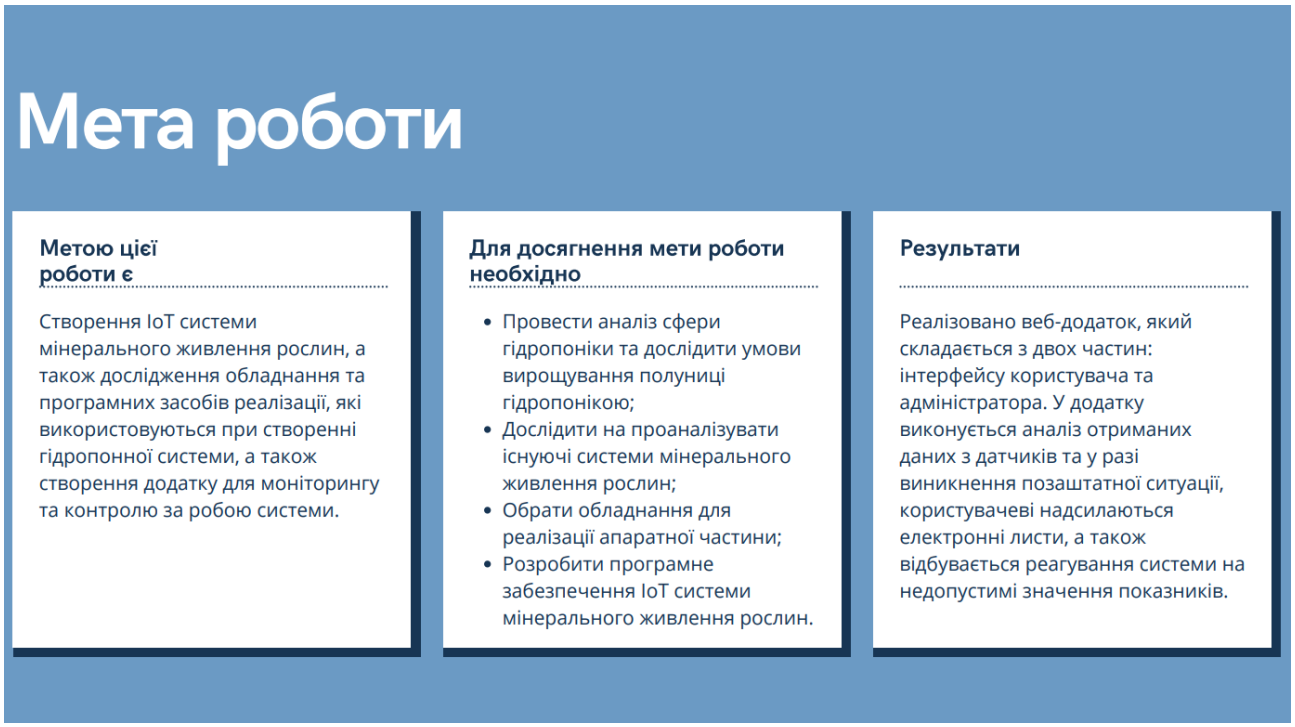

Рисунок А.2. Слайд 2

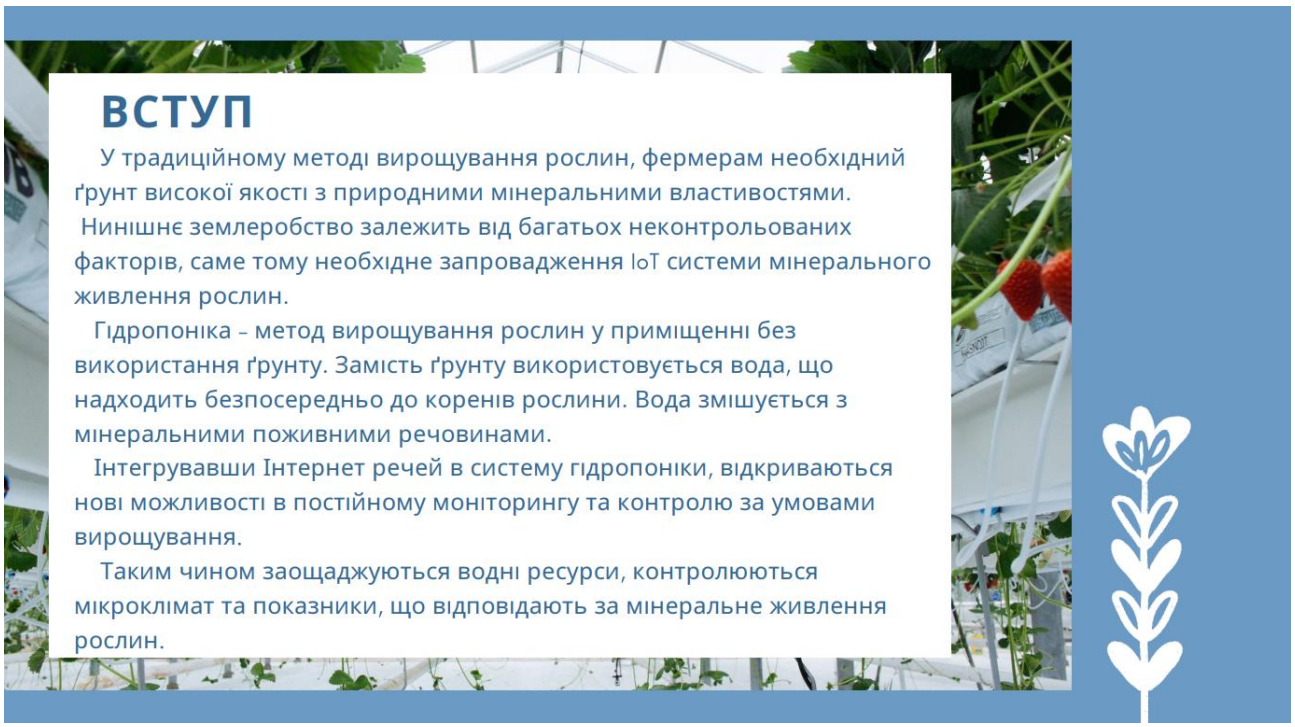

Рисунок А.3. Слайд 3

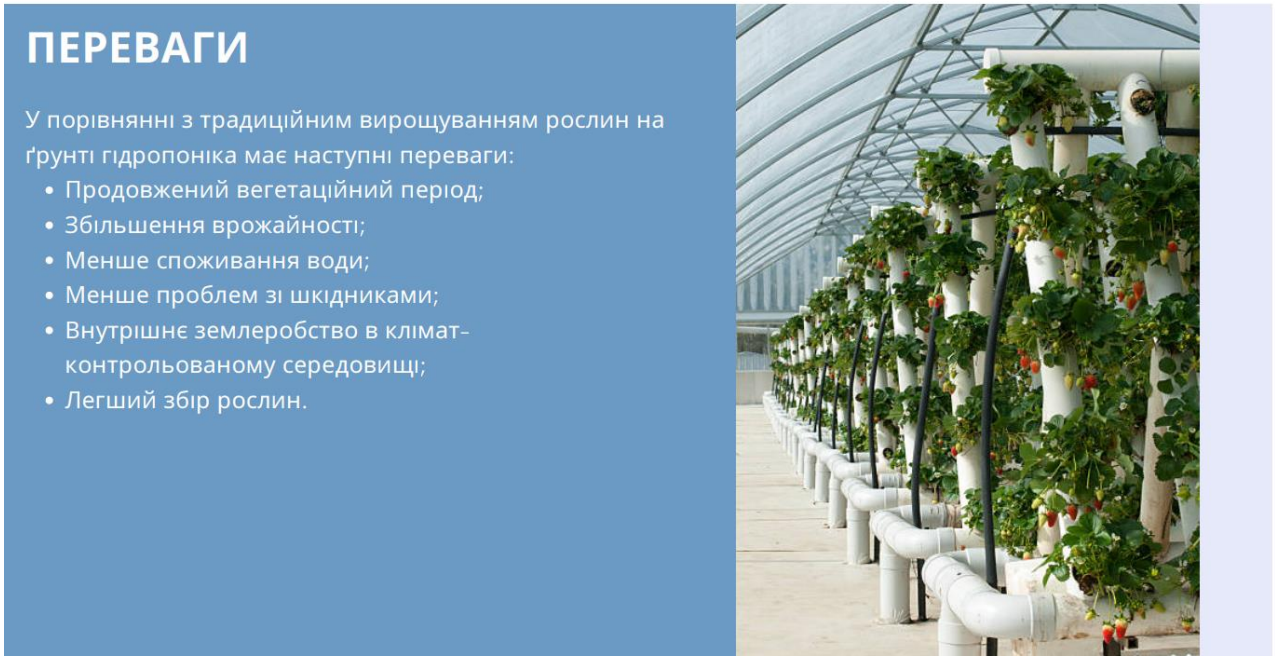

Рисунок А.4. Слайд 4

КОНЦЕПТУАЛЬНА МОДЕЛЬ АРХІТЕКТУРИ СИСТЕМИ

Архітектура системи представлена наступними компонентами: Raspberry Pi 4, вологозахищеним датчиком температури повітря DS18B20, датчиком вологості повітря DHT22, датчиком освітленості TSL25911FN, датчиком кислотності рідини pH, датчиком електропровідності рідини, 8-канальною платою розширення реле для Raspberry Pi.

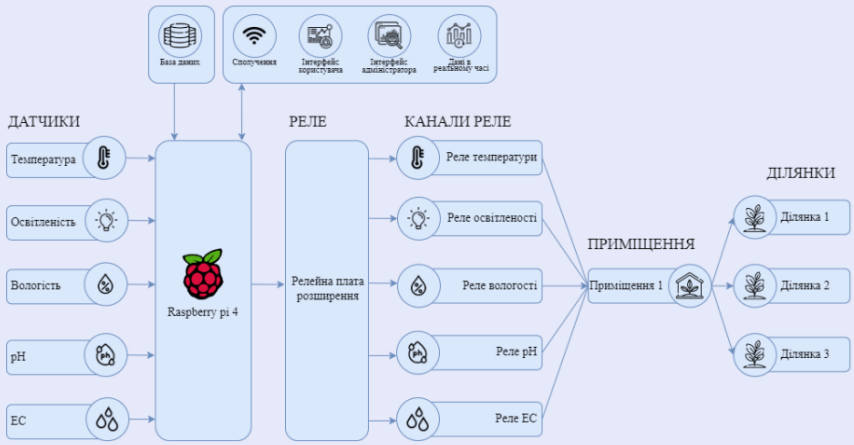

Рисунок А.5. Слайд 5

МОДЕЛЬ КЛАСІВ DJANGO

Модель являє собою клас, успадкований від django.db.models.Model. Проект має наступні наступні моделі класів: Room, GreenhouseArea, Plants, Sensor_Type, Sensor, DataSensor, Relay, RelayPins, Relay_Type, RelayPinsData, Device

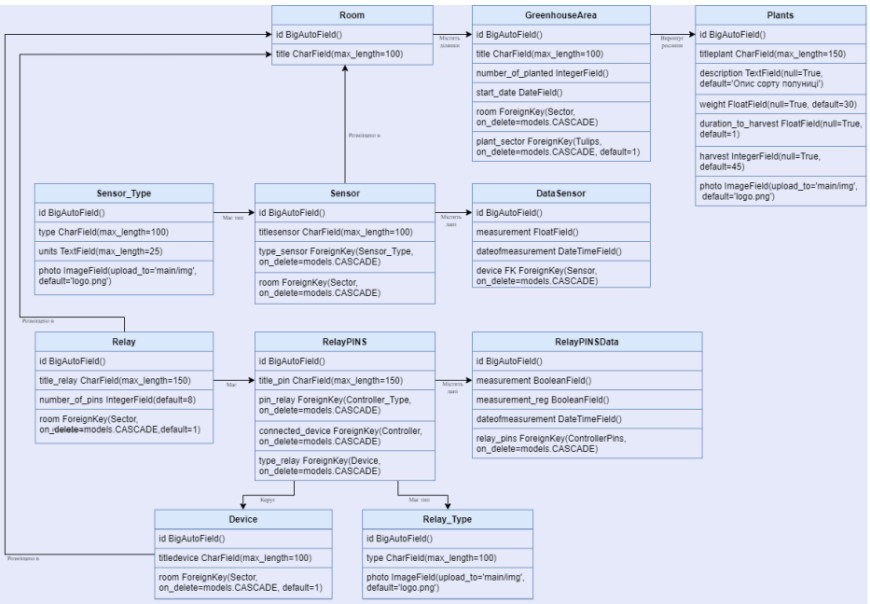

Рисунок А.6. Слайд 6

Рисунок А.7. Слайд 7

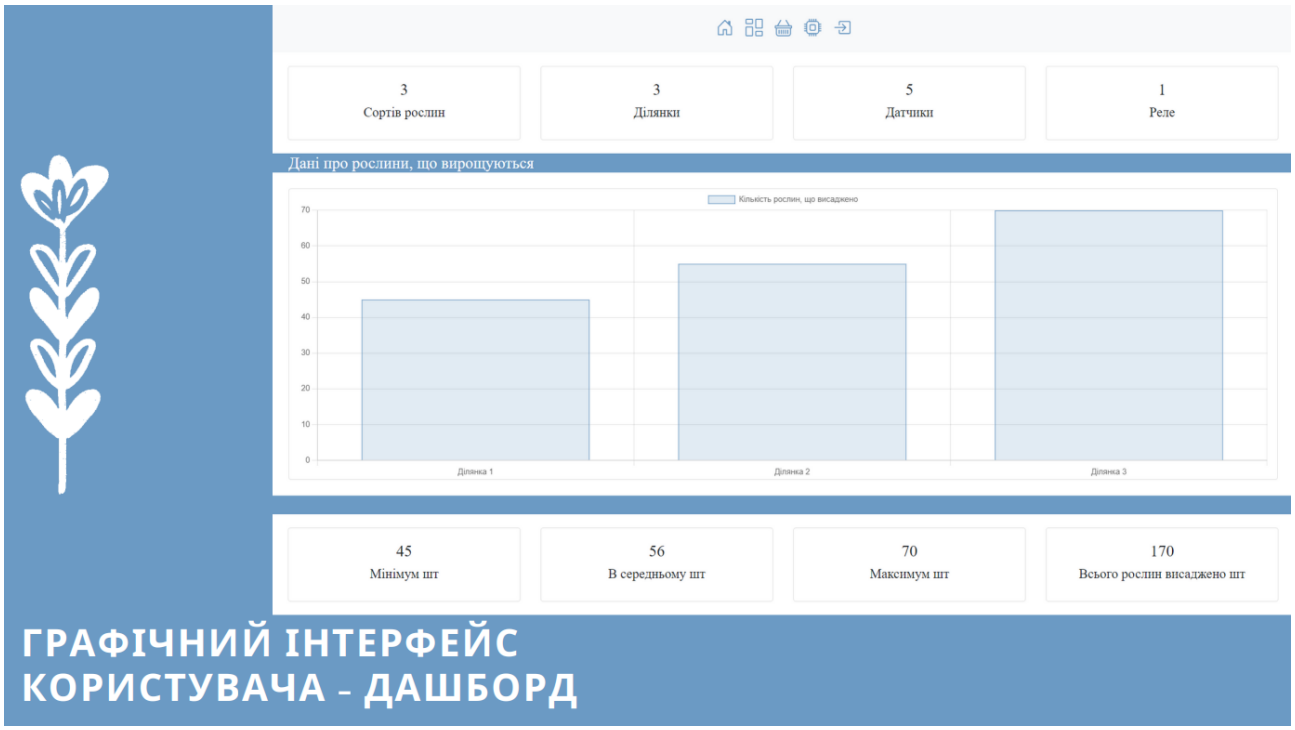

Рисунок А.8. Слайд 8

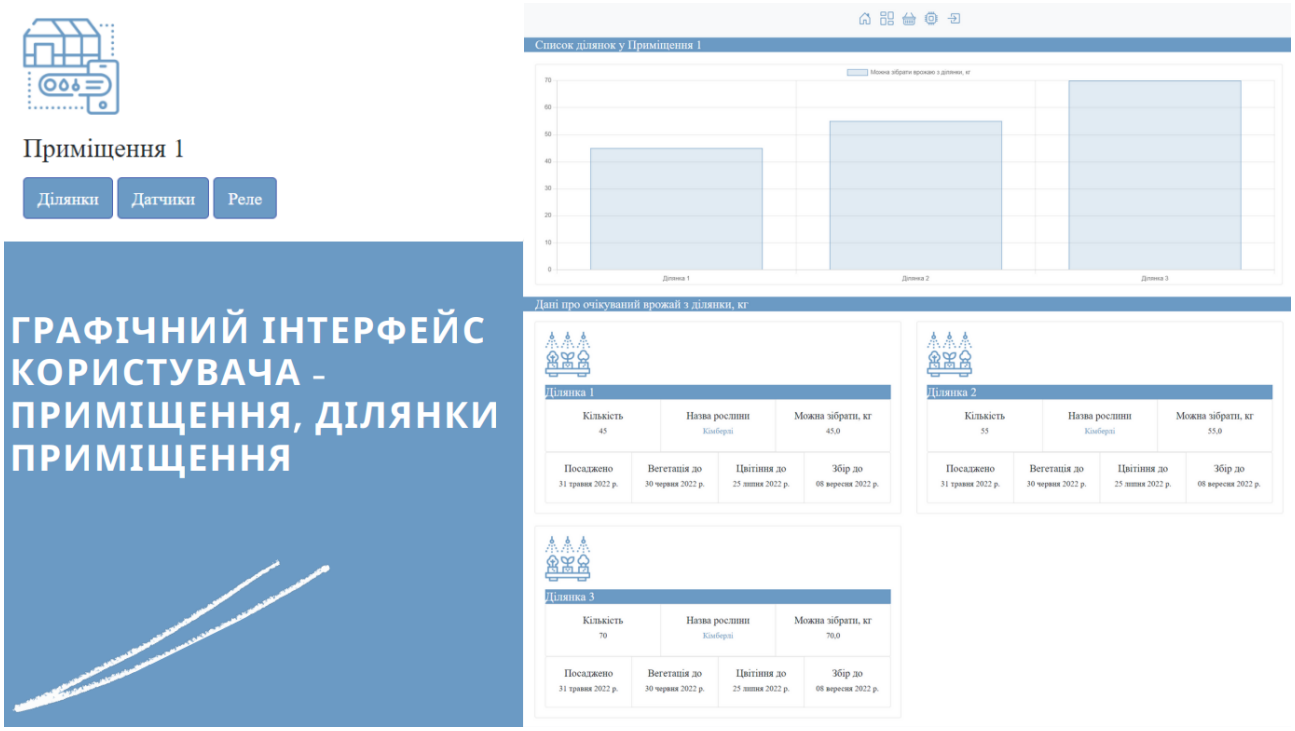

Рисунок А.9. Слайд 9

Рисунок А.10. Слайд 10

Рисунок А.11. Слайд 11

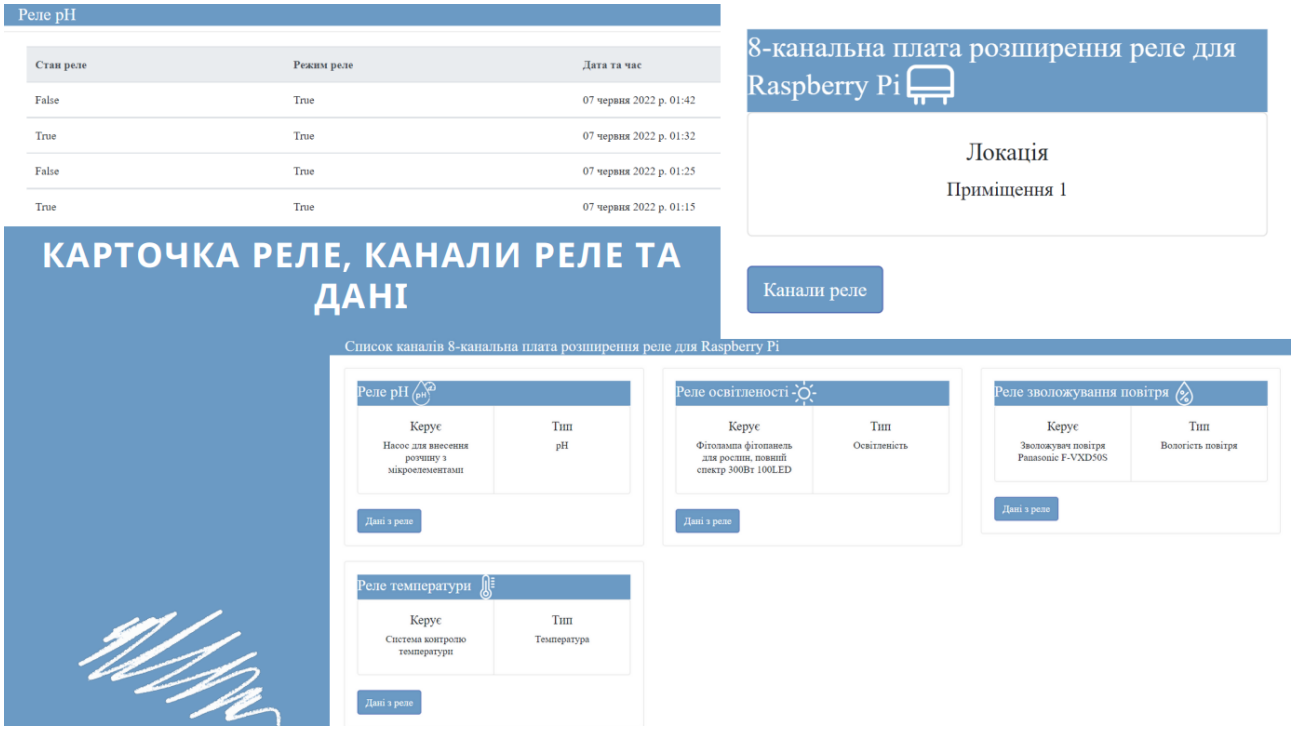

Рисунок А.12. Слайд 12

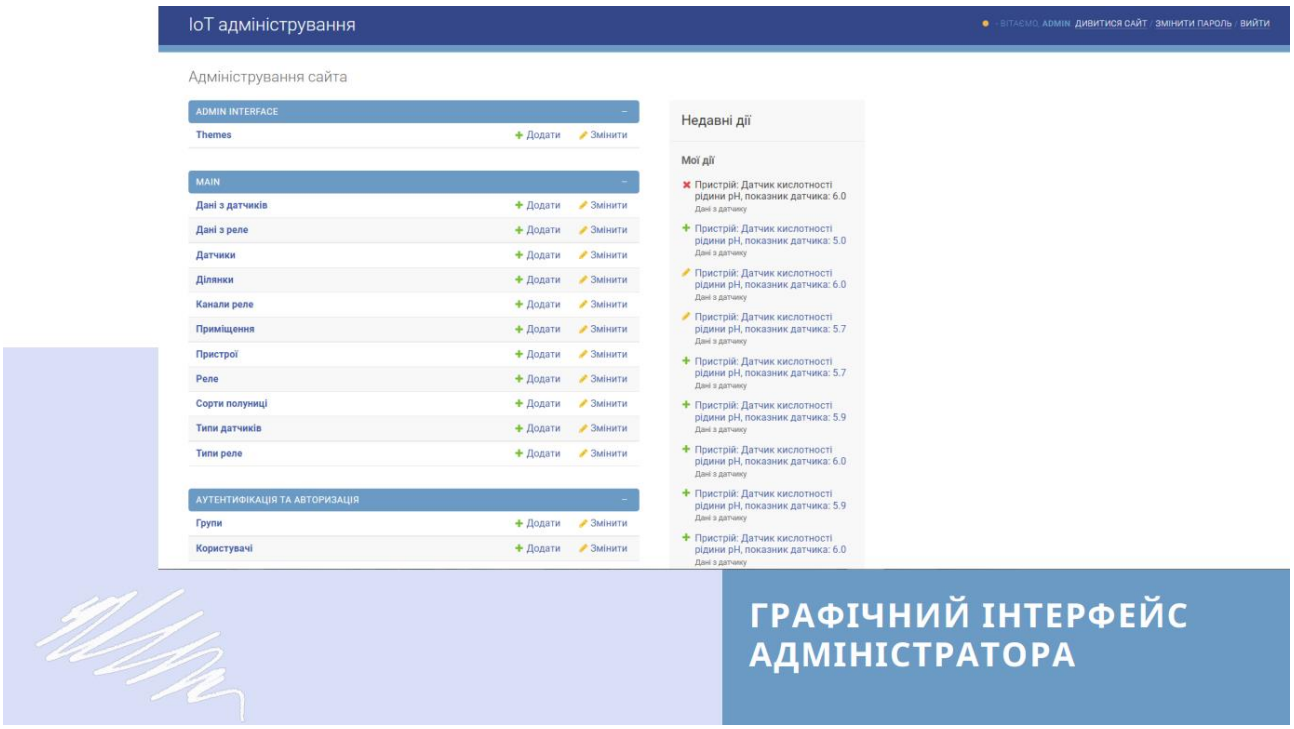

Рисунок А.13. Слайд 13

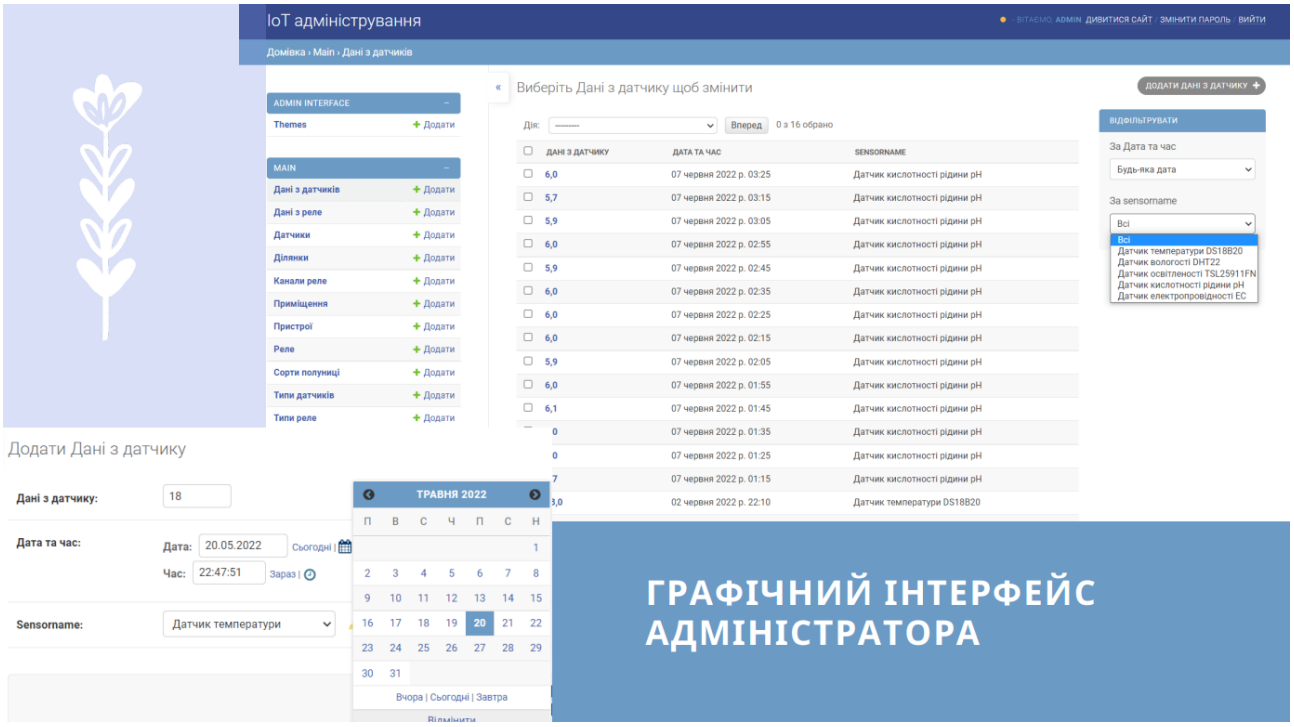

Рисунок А.14. Слайд 14

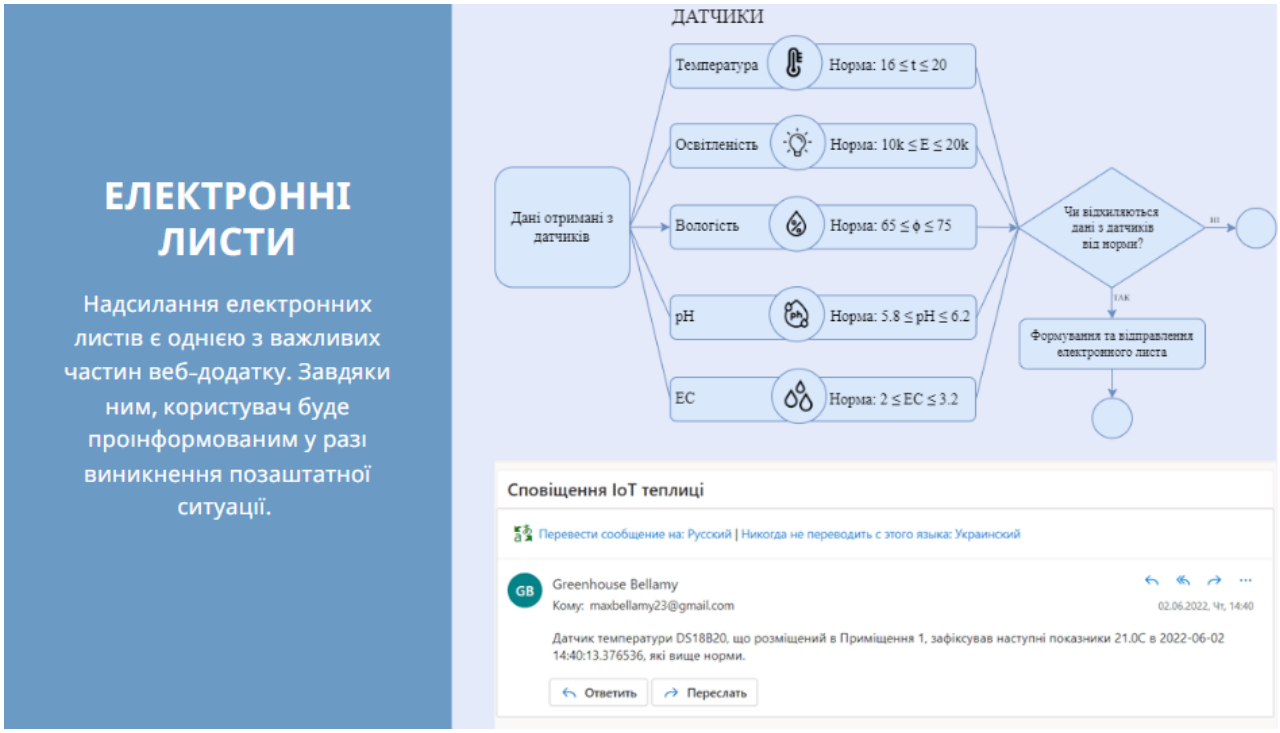

Рисунок А.15. Слайд 15

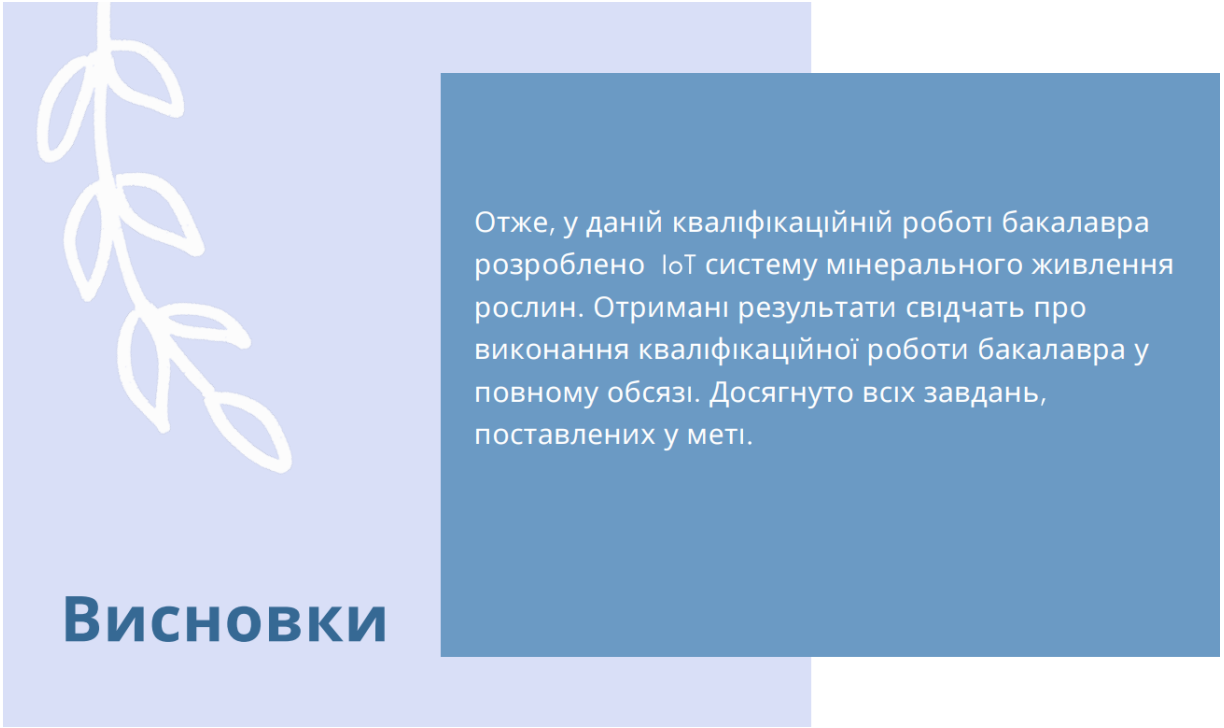

Рисунок А.16. Слайд 16

Рисунок А.17. Слайд 17

ДОДАТОК Б

Фрагмент коду програми

manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys
def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'greenhouse.settings')
    try:
        from django.core.management import
execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed
and "
            "available on your PYTHONPATH environment
variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)
if __name__ == '__main__':
    main()
```

views.py

```
from django.shortcuts import render
from .models import *
from django.core.mail import send_mail
from django.db.models import Count
from django.db.models import Avg, Max, Min, Sum
from datetime import datetime, timedelta




import plotly.graph_objects as go
from plotly.offline import plot
import plotly.express as px
import time
from numpy import random
def index(request):
    planting_sectors = Planting_Sector.objects.all()
    sensors = Sensor.objects.all()
    controllers=Controller.objects.all()
    count = Tulips.objects.all().count()
    count_sen = Sensor.objects.all().count()
    count_con = Controller.objects.all().count()
    count_plant = Planting_Sector.objects.all().count()
    count_plant_min =
Planting_Sector.objects.aggregate(a=Min('number_of_plante
d'))
    count_plant_avg =
Planting_Sector.objects.aggregate(a=Avg('number_of_plante
d', ))
    count_plant_max =
Planting_Sector.objects.aggregate(a=Max('number_of_plante
d', ))
    count_plant_sum =
Planting_Sector.objects.aggregate(a=Sum('number_of_plante
d', ))
    return
render(request,'main/index.html',{'planting_sectors':planting_
sectors, 'sensors':sensors, 'count_sen':count_sen,
'count_con':count_con, 'count_plant':count_plant,
'controllers':controllers, 'count':count,
'count_plant_sum':count_plant_sum["a"],
'count_plant_min':count_plant_min["a"],
'count_plant_avg':int(count_plant_avg["a"]),
'count_plant_max':count_plant_max["a"]})



def strawberry(request):
    strawberries = Tulips.objects.all()
    return render(request,
'main/strawberry.html',{'strawberries':strawberries})
def device(request):
    sensorname = Sensor.objects.all()
    datafromlast = DataSensor.objects.raw('''SELECT ds1.*
    FROM main_datasensor ds1 LEFT JOIN main_datasensor
ds2
     ON (ds1.sensorname_id = ds2.sensorname_id AND
ds1.dateofmeasurement < ds2.dateofmeasurement)
    WHERE ds2.dateofmeasurement IS NULL''')
    controller = Controller.objects.all()
```

```
    return render(request, 'main/devices.html', {'sensorname':
sensorname, 'datafromlast': datafromlast,
'controller':controller,})
def sector(request):
    sectors = Sector.objects.all()
    return render(request, 'main/sector.html',{'sectors':sectors})
def planting_sector_detail(request, id):
    sector = Sector.objects.get(id=id)
    planting_sectors = Planting_Sector.objects.filter(sector=id)
    d={}
    for i in planting_sectors:
        stage0 = i.start_date
        stage1 = stage0 + timedelta(days=30)#вегетація
        stage2 = stage1 + timedelta(days=25)#цвітіння
        stage3 = stage2 +
timedelta(days=i.tulip_sector.duration_to_harvest)#врожай
        kilo = i.tulip_sector.harvest * i.number_of_planted
        kiloint = int(kilo)# врожай,кг
        d[i.id]=[stage0,stage1,stage2,stage3,kilo,kiloint]



    return render(request, 'main/planting_sector.html', {'sector':
sector, 'planting_sectors': planting_sectors, 'stage0':stage0,
'stage1':stage1, 'stage2':stage2,'stage3':stage3, 'd':d,})
def planting_sector_photo(request, id):
    planting_sectors = Planting_Sector.objects.get(id=id)
    tulipphoto=Tulips.objects.filter(planting_sector=id)
    return render(request, 'main/tulips.html',
{'planting_sectors':planting_sectors,'tulipphoto': tulipphoto})
def sensore_detail(request,id):
    sector = Sector.objects.get(id=id)
    sensors = Sensor.objects.filter(sector=id)
    phototype = Sensor_Type.objects.filter(id=id)
    return render(request, 'main/sensor.html', {'sector': sector,
'sensors': sensors, 'phototype':phototype})
def datafromsensors(request, id):
    sensorname = Sensor.objects.get(id=id)
    datafrom = DataSensor.objects.filter(sensorname=id)
    datafromast =
DataSensor.objects.filter(sensorname=id).last()
    datafromPrev =
DataSensor.objects.filter(sensorname=id).order_by('-id')[1]
    datafrom1 = DataSensor.objects.filter(sensorname=id,
dateofmeasurement__gte=datetime.now() -
timedelta(hours=300))
    fig = go.Figure()
    scatter = go.Indicator(
        mode="gauge+number",
        value=datafromast.measurement,
        domain={'x': [0, 1], 'y': [0, 1]},
        gauge={'bar': {'color': "#6B9AC4"}},
        title={'text': "Показник датчика"},
    )
    fig.add_trace(scatter)
    fig.update_layout(autosize=True,height=350)
    plot_div = plot(fig, output_type='div')
    try:
        if datafromast.measurement<16 and
sensorname.type_sensors.type == 'Температура':
            send_mail("Сповіщення IoT теплиці",
                f'{sensorname.titlesensor}, що розміщений в
{sensorname.sector}, зафіксував наступні показники
{datafromast.measurement}{sensorname.type_sensors.units}
в {datetime.now()}, які нижче норми.',
                "greenhouse_bellamy@outlook.com",
                ["maxbellamy23@gmail.com"],
                fail_silently=False)
        elif datafromast.measurement>20 and
sensorname.type_sensors.type == 'Температура':
            send_mail("Сповіщення IoT теплиці",
                f'{sensorname.titlesensor}, що розміщений в
{sensorname.sector}, зафіксував наступні показники
{datafromast.measurement}{sensorname.type_sensors.units}
в {datetime.now()}, які вище норми.',
                "greenhouse_bellamy@outlook.com",
                ["maxbellamy23@gmail.com"],
                fail_silently=False)
        elif datafromast.measurement<65 and
sensorname.type_sensors.type == 'Вологість повітря':
            send_mail("Сповіщення IoT теплиці",
                f'{sensorname.titlesensor}, що розміщений в
{sensorname.sector}, зафіксував наступні показники
{datafromast.measurement}{sensorname.type_sensors.units}
в {datetime.now()}, які вище норми.',
                "greenhouse_bellamy@outlook.com",
                ["maxbellamy23@gmail.com"],
                fail_silently=False)
        elif datafromast.measurement>75 and
sensorname.type_sensors.type == 'Вологість повітря':
            send_mail("Сповіщення IoT теплиці",
                f'{sensorname.titlesensor}, що розміщений в
{sensorname.sector}, зафіксував наступні показники
```

```
{datafromast.measurement}{sensorname.type_sensors.units}
в {datetime.now()}, які вище норми.',
                "greenhouse_bellamy@outlook.com",
                ["maxbellamy23@gmail.com"],
                fail_silently=False)
    elif datafromast.measurement<5.8 and
sensorname.type_sensors.type == 'pH':
        send_mail("Сповіщення IoT теплиці",
                f'{sensorname.titlesensor}, що розміщений в
{sensorname.sector}, зафіксував наступні показники
{datafromast.measurement}{sensorname.type_sensors.units}
в {datetime.now()}, які вище норми.',
                "greenhouse_bellamy@outlook.com",
                ["maxbellamy23@gmail.com"],
                fail_silently=False)
    elif datafromast.measurement>6.2 and
sensorname.type_sensors.type == 'pH':
        send_mail("Сповіщення IoT теплиці",
                f'{sensorname.titlesensor}, що розміщений в
{sensorname.sector}, зафіксував наступні показники
{datafromast.measurement}{sensorname.type_sensors.units}
в {datetime.now()}, які вище норми.',
                "greenhouse_bellamy@outlook.com",
                ["maxbellamy23@gmail.com"],
                fail_silently=False)
    elif datafromast.measurement<55000 and
sensorname.type_sensors.type == 'Освітленість':
        send_mail("Сповіщення IoT теплиці",
                f'{sensorname.titlesensor}, що розміщений в
{sensorname.sector}, зафіксував наступні показники
{datafromast.measurement}{sensorname.type_sensors.units}
в {datetime.now()}, які вище норми.',
                "greenhouse_bellamy@outlook.com",
                ["maxbellamy23@gmail.com"],
                fail_silently=False)
    elif datafromast.measurement>80000 and
sensorname.type_sensors.type == 'Освітленість':
        send_mail("Сповіщення IoT теплиці",
                f'{sensorname.titlesensor}, що розміщений в
{sensorname.sector}, зафіксував наступні показники
{datafromast.measurement}{sensorname.type_sensors.units}
в {datetime.now()}, які вище норми.',
                "greenhouse_bellamy@outlook.com",
                ["maxbellamy23@gmail.com"],
                fail_silently=False)
    elif datafromast.measurement<2 and
sensorname.type_sensors.type == 'EC':
        send_mail("Сповіщення IoT теплиці",
                f'{sensorname.titlesensor}, що розміщений в
{sensorname.sector}, зафіксував наступні показники
{datafromast.measurement}{sensorname.type_sensors.units}
в {datetime.now()}, які вище норми.',
                "greenhouse_bellamy@outlook.com",
                ["maxbellamy23@gmail.com"],
                fail_silently=False)
    elif datafromast.measurement>3.2 and
sensorname.type_sensors.type == 'EC':
        send_mail("Сповіщення IoT теплиці",
                f'{sensorname.titlesensor}, що розміщений в
{sensorname.sector}, зафіксував наступні показники
{datafromast.measurement}{sensorname.type_sensors.units}
в {datetime.now()}, які вище норми.',
                "greenhouse_bellamy@outlook.com",
                ["maxbellamy23@gmail.com"],
                fail_silently=False)
  except:
    print('something is wrong')
  def generate_temperature():
  x = random.choice([16, 17, 18, 19, 20], p=[0.15, 0.15, 0.3,
0.2, 0.2], size=(1))
  for i in range(0,24):
 room = DataSensor(dateofmeasurement=datetime.now(),
sensorname_id=3, measurement=x)
room.save()
time.sleep(300)
generate_temperature()
  conditions = ['Показники в нормі', 'Показники менше
норми', 'Показники більше норми', 'Дані відсутні']
  try:
    if sensorname.type_sensors.type == 'Температура':
      if 16 <= datafromast.measurement <= 20:
        condition=conditions[0]
      elif datafromast.measurement<16:
        condition=conditions[1]
      elif datafromast.measurement>20:
        condition = conditions[2]
    elif sensorname.type_sensors.type == 'Вологість
повітря':
      if 65<=datafromast.measurement<=75:
        condition = conditions[0]
      elif datafromast.measurement<65:
```

```
                condition = conditions[1]
            elif datafromast.measurement > 75:
                condition = conditions[2]
        elif sensorname.type_sensors.type == 'pH':
            if 5.8<=datafromast.measurement<=6.2:
                condition = conditions[0]
            elif datafromast.measurement<5.8:
                condition = conditions[1]
            elif datafromast.measurement > 6.2:
                condition = conditions[2]
        elif sensorname.type_sensors.type == 'Освітленість':
            if 55000<=datafromast.measurement<=80000:
                condition = conditions[0]
            elif datafromast.measurement<55000:
                condition = conditions[1]
            elif datafromast.measurement > 80000:
                condition = conditions[2]
        elif sensorname.type_sensors.type == 'EC':
            if 2<=datafromast.measurement<=3.2:
                condition = conditions[0]
            elif datafromast.measurement<2:
                condition = conditions[1]
            elif datafromast.measurement > 3.2:
                condition = conditions[2]
    except:
        condition = conditions[3]
    return render(request, 'main/data.html',{'sensorname':
sensorname, 'datafrom':datafrom, 'datafromast':datafromast,
'condition':condition, 'datafrom1':datafrom1,
'plot_div':plot_div, })
def controller_detail(request,id):
    sector = Sector.objects.get(id=id)
    controllers = Controller.objects.filter(sector_con=id)
    return render(request, 'main/controller.html', {'sector':
sector, 'controllers': controllers,})
def controller_pins_detail(request,id):
    controllers = Controller.objects.get(id=id)
    controller_pins =
ControllerPins.objects.filter(controllername_con=id)
    phototype = Controller_Type.objects.filter(id=id)
    return render(request, 'main/controller_pins.html',
{'controllers': controllers,
'controller_pins':controller_pins, 'phototype':phototype})
def datafromcontroller(request, id):
    controller_pins = ControllerPins.objects.get(id=id)
    datafromcon =
DataController.objects.filter(controllername_con=id).order_b
y('-id')
    return render(request,
'main/datacon.html',{'controller_pins': controller_pins,
'datafromcon':datafromcon})
def chart(request):
    plant_data = Planting_Sector.objects.all()
    return render(request, 'main/index.html',
{'plant_data':plant_data})
def chartsen(request):
    sen_data = DataSensor.objects.all()
    return render(request, 'main/data.html',
{'sen_data':sen_data})
def chartkg(request):
    plant = Planting_Sector.objects.all()
    return render(request, 'main/planting_sector.html',
{'plant':plant})
```

main.py

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='home'),
    path('sector', views.sector, name='sector'),
    path('sector/<int:id>', views.planting_sector_detail,
name='sector-details'),
    path('sensor/<int:id>', views.sensore_detail, name='sensor-
details'),
    path('data/<int:id>', views.datafromsensors, name='data-
details'),
    path('tulips/<int:id>', views.planting_sector_photo,
name='tulip_photo'),
    path('strawberry', views.strawberry, name='strawberry'),
    path('calculator', views.calculator, name='calculator'),
    path('devices', views.device, name='device'),
    path('', views.chart, name='chart'),
    path('data', views.chartsen, name='chartsen'),
    path('planting_sector', views.chartkg, name='chartkg'),
    path('controller/<int:id>', views.controller_detail,
name='controller-details'),
    path('controller_pins/<int:id>',
views.controller_pins_detail, name='controller_pins_detail'),
    path('datacon/<int:id>', views.datafromcontroller,
name='datacon-details'),]
```

models.py

```
from django.db import models
```

```
from django.contrib import admin
class Tulips(models.Model):
    titletulip = models.CharField('Назва сорту полуниці',max_length=150)
    description = models.TextField('Опис', null=True, default='Опис сорту полуниці')
    weight = models.FloatField('Середня вага, г', null=True, default=30)
    harvest = models.FloatField('Врожайність, кг', null=True, default=1)
    duration_to_harvest = models.IntegerField('Дозріває, днів', null=True, default=45)
    photo = models.ImageField('Фото', upload_to='main/img', default='logo.png')
    def __str__(self):
        return self.titletulip
    class Meta:
        verbose_name = 'Сорт полуниці'
        verbose_name_plural = 'Сорти полуниці'
class Sector(models.Model):
    title = models.CharField('Назва приміщення', max_length=100)
    def __str__(self):
        return self.title
    class Meta:
        verbose_name = 'Приміщення'
        verbose_name_plural = 'Приміщення'
class Planting_Sector(models.Model):
    title = models.CharField('Назва ділянки', max_length=100)
    number_of_planted=models.IntegerField('Кількість висаджених саджанців')
    start_date = models.DateField('Дата висадки')
    sector = models.ForeignKey(Sector, on_delete=models.CASCADE)
    tulip_sector = models.ForeignKey(Tulips, on_delete=models.CASCADE, default=1)
    def __str__(self):
        return str(self.title) +str(self.start_date)
    class Meta:
        verbose_name = 'Ділянка'
        verbose_name_plural = 'Ділянки'
class Sensor_Type(models.Model):
    type = models.CharField('Тип датчику',max_length=100)
    units = models.TextField('Одиниці виміру', max_length=25)
    photo = models.ImageField(upload_to='main/img', default='logo.png')
    def __str__(self):
        return self.type
    class Meta:
        verbose_name = 'Тип датчику'
        verbose_name_plural = 'Типи датчиків'
class Sensor(models.Model):
    titlesensor = models.CharField('Назва датчика',max_length=100)
    type_sensors = models.ForeignKey(Sensor_Type, on_delete=models.CASCADE)
    sector = models.ForeignKey(Sector, on_delete=models.CASCADE)
    def __str__(self):
        return self.titlesensor
    class Meta:
        verbose_name = 'Датчик'
        verbose_name_plural = 'Датчики'
class DataSensor(models.Model):
    measurement = models.FloatField('Дані з датчику')
    dateofmeasurement = models.DateTimeField('Дата та час')
    sensorname = models.ForeignKey(Sensor, on_delete=models.CASCADE)
    def __str__(self):
        return 'Пристрій: ' + str(self.sensorname) + ', показник датчика: ' + str(self.measurement)
    class Meta:
        verbose_name = 'Дані з датчику'
        verbose_name_plural = 'Дані з датчиків'
class Controller_Type(models.Model):
    type_con = models.CharField('Тип реле',max_length=100)
    photo_con = models.ImageField(upload_to='main/img', default='logo.png')
    def __str__(self):
        return self.type_con
    class Meta:
        verbose_name = 'Тип реле'
        verbose_name_plural = 'Типи реле'
class Controller(models.Model):
    titlecontroller_con = models.CharField('Назва реле',max_length=150)
    number_of_pins = models.IntegerField('Кількість каналів реле', default=8)
```

```
    sector_con = models.ForeignKey(Sector,
on_delete=models.CASCADE,default=1)
    def __str__(self):
        return self.titlecontroller_con
    class Meta:
        verbose_name = 'Реле'
        verbose_name_plural = 'Реле'
class Device(models.Model):
    title = models.CharField('Назва пристрою',
max_length=100)
    sector_con = models.ForeignKey(Sector,
on_delete=models.CASCADE, default=1)
    def __str__(self):
        return self.title
    class Meta:
        verbose_name = 'Пристрій'
        verbose_name_plural = 'Пристрої'
class ControllerPins(models.Model):
    titlecontroller_con = models.CharField('Назва каналу
реле',max_length=150)
    type_controller_con =
models.ForeignKey(Controller_Type,
on_delete=models.CASCADE)
    controllername_con = models.ForeignKey(Controller,
on_delete=models.CASCADE)
    device = models.ForeignKey(Device,
on_delete=models.CASCADE)
    def __str__(self):
        return self.titlecontroller_con
    class Meta:
        verbose_name = 'Канал реле'
        verbose_name_plural = 'Канали реле'
class DataController(models.Model):
    measurement_con = models.BooleanField('Стан реле')
    measurement_reg = models.BooleanField('Режим
реле',default=1)
    dateofmeasurement_con = models.DateTimeField('Дата та
час')
    controllername_con = models.ForeignKey(ControllerPins,
on_delete=models.CASCADE)
    def __str__(self):
        return 'Пристрій: ' + str(self.controllername_con) + ',
показники реле: ' + str(self.measurement_con)
    class Meta:
        verbose_name = 'Дані з реле'
        verbose_name_plural = 'Дані з реле'
```

admin.py

```
from django.contrib import admin
from .models import Tulips
from .models import Sector
from .models import Planting_Sector
from .models import Sensor_Type
from .models import Sensor
from .models import DataSensor
from .models import Controller_Type
from .models import Controller
from .models import DataController
from .models import Device
from .models import ControllerPins
class SensorAdmin(admin.ModelAdmin):
    list_filter = ('sector', 'type_sensors')
    list_display = ['titlesensor', 'type_sensors', 'sector']
    fields = ['titlesensor', ('type_sensors', 'sector')]
class ControllerAdmin(admin.ModelAdmin):
    list_display = ['titlecontroller_con', 'number_of_pins',
'sector_con']
class DeviceAdmin(admin.ModelAdmin):
    list_display = ['title', 'sector_con']
class ControllerPinsAdmin(admin.ModelAdmin):
    list_filter = ('type_controller_con', 'controllername_con',
'device')
    list_display = ['titlecontroller_con', 'type_controller_con',
'controllername_con', 'device']
    fields = [('titlecontroller_con','device'),
('type_controller_con', 'controllername_con')]
class DataControllerAdmin(admin.ModelAdmin):
    list_filter = ('dateofmeasurement_con',
'controllername_con')
    list_display = ['measurement_con','measurement_reg',
'dateofmeasurement_con', 'controllername_con']
    fields =
[('measurement_con','measurement_reg'),'dateofmeasurement
_con','controllername_con']
class DataSensorAdmin(admin.ModelAdmin):
    list_filter = ('dateofmeasurement', 'sensorname')
    list_display = ['measurement', 'dateofmeasurement',
'sensorname']
class PlantingAdmin(admin.ModelAdmin):
    list_filter = ('sector', 'tulip_sector')
    list_display = ['title', 'number_of_planted', 'start_date',
'sector', 'tulip_sector']
```

```
    fields = ['title','number_of_planted','start_date',
('sector','tulip_sector')]
class PlantSectorInline(admin.TabularInline):
    model = Planting_Sector
class SectorAdmin(admin.ModelAdmin):
    inlines = [
        PlantSectorInline,
    ]
class TulipSearch(admin.ModelAdmin):
    search_fields = ['titletulip']
    list_display = ['titletulip', 'harvest', 'duration_to_harvest']
    fields = ['titletulip', 'description', ('weight', 'harvest',
'duration_to_harvest'), 'photo']
class ControllerSearch(admin.ModelAdmin):
    search_fields = ['titlecontroller_con']
class Sensor_TypeAdmin(admin.ModelAdmin):
    list_display = ['type','units']
class Controller_TypeAdmin(admin.ModelAdmin):
    list_display = ['type_con']
admin.site.register(Tulips, TulipSearch)
admin.site.register(Sensor_Type, Sensor_TypeAdmin)
admin.site.register(Sensor, SensorAdmin,)
admin.site.register(DataSensor, DataSensorAdmin)
admin.site.register(Sector,SectorAdmin)
admin.site.register(Planting_Sector, PlantingAdmin)
admin.site.register(Controller_Type,Controller_TypeAdmin)
admin.site.register(Controller, ControllerAdmin)
admin.site.register(DataController, DataControllerAdmin)
admin.site.register(Device, DeviceAdmin)
admin.site.register(ControllerPins, ControllerPinsAdmin)
```

urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('main.urls')),
] +static(settings.STATIC_URL,
document_root=settings.STATIC_ROOT) +
static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

index.html

```
{% extends 'main/layout.html' %}
{% block title %}Дашборд{% endblock  %}
{% block content %}
<div class="row row-cols-1 row-cols-md-4 m-2">
    <div class="col my-2 ">
        <div class="card my-1 ">
            <div class="card-body ">
                <center>
                    <h4>{{count}}</h4>
                    <h5>Сортів рослин</h5>
                </center>
            </div>
        </div>
    </div>
    <div class="col my-2 ">
        <div class="card my-1 ">
            <div class="card-body ">
                <center>
                    <h4>{{count_plant}}</h4>
                    <h5>Ділянки</h5>
                </center>
            </div>
        </div>
    </div>
    <div class="col my-2 ">
        <div class="card my-1 ">
            <div class="card-body ">
                <center>
                    <h4>{{count_sen}}</h4>
                    <h5>Датчики</h5>
                </center>
            </div>
        </div>
    </div>
    <div class="col my-2 ">
        <div class="card my-1 ">
            <div class="card-body ">
                <center>
                    <h4>{{count_con}}</h4>
                    <h5>Реле</h5>
                </center>
            </div>
        </div>
    </div>
</div>
<h4 class="datapro pl-4">Дані про рослини, що
вирощуються</h4>
<div class="card-deck-wrapper p-4">
    <div class="card-deck">
```

```
      <div class="card col-xl">
        <div class="card-block">
          <canvas id="myChartfirst" width="100"
height="30vh"></canvas>
          <script>
  const  firstctx =
document.getElementById('myChartfirst').getContext('2d');
  const  myChartfirst = new Chart(firstctx, {
    type: 'bar',
    data: {
      labels: [{% for i in planting_sectors %}'{{i.title}}',{%
endfor %}],
      datasets: [{
        label: 'Кількість рослин, що висаджено',
        data: [{% for i in planting_sectors
%}'{{i.number_of_planted}}',{% endfor %}],
        backgroundColor: ['rgba(107,154,196, 0.2)'],
        borderColor: ['rgba(107,154,196, 1)',],
        borderWidth: 1}]},
    options: {
      scales: {
        y: {
          beginAtZero: true
        }
      }
    }
  });
</script>
        </div>
      </div>
    </div>
</div>
<h4 class="datapro"><br></h4>
<div class="row row-cols-1 row-cols-md-4 m-2">
  <div class="col my-2 ">
    <div class="card my-1 ">
      <div class="card-body ">
        <center>
          <h4>{{count_plant_min}}</h4>
          <h5>Мінімум шт</h5>
        </center>
      </div>
    </div>
  </div>
  <div class="col my-2 ">
    <div class="card my-1 ">
      <div class="card-body ">
        <center>
          <h4>{{count_plant_avg}}</h4>
          <h5>В середньому шт</h5>
        </center>
      </div>
    </div>
  </div>
  <div class="col my-2 ">
    <div class="card my-1 ">
      <div class="card-body ">
        <center>
          <h4>{{count_plant_max}}</h4>
          <h5>Максимум шт</h5>
        </center>
      </div>
    </div>
  </div>
  <div class="col my-2 ">
    <div class="card my-1 ">
      <div class="card-body ">
        <center>
          <h4>{{count_plant_sum}}</h4>
          <h5>Всього рослин висаджено шт</h5>
        </center>
      </div>
    </div>
  </div>
</div>
{% endblock  %}
```

strawberry.html

```
{% extends 'main/layout.html' %}
{% block title %}Сорти полуниці{% endblock  %}
{% block content %}
<h4 class="datapro pl-4">Список сортів полуниці
висадженої в теплиці</h4>
<div class="row row-cols-1 row-cols-md-3 m-2">
  {% for el in strawberries %}
  <div class="col my-2 ">
    <div class="card my-1 ">
      <div class="card-body ">
        <img src="{{MEDIA_URL}}{{el.photo}}"
class="card-img-top">
        <h5 class="datapro">{{el.titletulip}}</h5>
        <div class="row row-cols-1 row-cols-md-3 ">
          <div class="col my-1 ">
```

```
    <div class="card my-1 ">
      <div class="card-body ">
        <center>
          <h4>{{el.weight}}</h4>
          <h6>Вага 1 ягоди, г</h6>
        </center>
      </div>
    </div>
  </div>
  <div class="col my-1 ">
    <div class="card my-1 ">
      <div class="card-body ">
        <center>
          <h4>{{el.harvest}}</h4>
          <h6>Врожай з 1 куща, кг</h6>
        </center>
      </div>
    </div>
  </div>
  <div class="col my-1 ">
    <div class="card my-1 ">
      <div class="card-body ">
        <center>
          <h4>{{el.duration_to_harvest}}</h4>
          <h6>Достигає за, дн</h6>
        </center>
      </div>
    </div>
  </div>
  <div class="col m-0 p-0">
    <div class="card  ">
      <div class="card-body ">
        <center>
          <h6>{{el.description}}</h6>
        </center>
      </div>
    </div>
  </div>
 </div>
</div>
{% endfor %}
</div>
{% endblock  %}
```