

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДОПОМОГИ

(назва теми)

У ВИРОЩУВАННІ РОСЛИН. PLANT CARE SOFTWARE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-220сп

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

КОВАЛЬОВА Г.М.

(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« » 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КОВАЛЬОВОЇ Ганни Михайлівни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення допомоги у вирощуванні рослин. Plant Care Software

керівник проєкту (роботи) к.т.н., доцент СКРУПСЬКИЙ Степан Юрійович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «04» квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 1 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Реалізація програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 4 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

_____ Ганна КОВАЛЬОВА
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 92 с., 22 рис., 3 табл., 3 дод., 12 джерел.

ВИРОЩУВАННЯ РОСЛИН, ВИД РОСЛИНИ, ХВОРОБА, ЕКСПЕРТ, ПОРАДА, ВЕБДОДАТОК.

Об'єкт роботи – процес вирощування рослин. Предмет роботи – програмне забезпечення допомоги у вирощуванні рослин. Мета роботи – розробити програмне забезпечення допомоги у вирощуванні рослин для спрощення керування процесом вирощування рослин і виявленими під час нього проблемами.

Матеріали, методи та технічні засоби: мова Python, фреймворк Django, який підтримує веброзробку, система керування базами даних PostgreSQL.

Результати. Вебдодаток допомоги у вирощуванні рослин націлений на розв'язання спектру проблем, які виникають під час вирощування як домашніх рослин, так і під час садівництва як такого. Для його забезпечення виконано процедури моделювання, проєктування і реалізації, а основні рішення, отримані під час реалізації цих процедур, описані та є результатами роботи.

Висновки. Створена програма надає можливості використання її садівниками для пошуку нових рослин, для впорядкування догляду за власними рослинами, вирішення проблем, які пов'язані з захворюваннями рослин, знаходження інформації щодо правильного догляду за рослинами. Для експертів програма надає можливість ділитися своїм досвідом, знаннями, допомагати садівникам у тих проблемах, які у них виникають.

Галузь використання – садівництво.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 92 pages, 22 figures, 3 tables, 3 appendixes, 12 sources.

PLANT GROWING, PLANT SPECIES, DISEASE, EXPERT, ADVICE, WEB APPLICATION.

The object of the work is the process of growing plants. The subject of the work is plant care software. The goal of the work is to develop a plant care software to simplify the management of the plant cultivation process and the problems identified during it.

Materials, methods and technical means: Python language, Django framework which supports web development, PostgreSQL database management system.

The results. The plant care web application aims to solve a range of problems that arise when growing both house plants and gardening in general. To ensure it, modeling, design and implementation procedures were performed, and the main solutions obtained during the implementation of these procedures are described and are the results of the work.

Conclusions. The created program provides opportunities for gardeners to use it to find new plant species, to organize the care of their own plants, to solve problems concerning plant diseases, and to find information on proper plant care. For experts the program provides an opportunity to share their experience, knowledge and help gardeners with the problems they have.

The field of use is gardening.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	9
1.1 Аналіз програмних засобів допомоги у вирощуванні рослин.....	9
1.2 Визначення концепції розроблюваної програми	11
1.3 Висновки за розділом 1 та постановка завдань роботи	13
2 Розробка архітектури програми.....	14
2.1 Вибір технологій розробки	14
2.2 Моделювання вимог до програми	15
2.3 Проєктування бази даних	20
2.4 Висновки за розділом 2	35
3 Реалізація програми	37
3.1 Програмна реалізація функцій експерта.....	37
3.2 Програмна реалізація функцій садівника	44
3.3 Висновки за розділом 3	50
4 Експлуатація та тестування програми	51
4.1 Призначення програми	51
4.2 Умови виконання програми	51
4.3 Виконання і тестування програми.....	51
4.4 Висновки за розділом 4	58
Висновки	59
Перелік джерел посилання	60
Додаток А Технічне завдання	61
Додаток Б Текст програми	66
Додаток В Слайди презентації.....	86

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

MVT – Model-View-Template;

БД – база даних.

ВСТУП

Вирощування рослин – це непростий процес. Він починається ще під час вибору рослин, продовжується, коли людина саджає або купляє готову рослину, але на цьому не зупиняється, а тільки розпочинається фактично. Далі потрібно весь час доглядати рослини: поливати, вносити добрива, обрізати певним чином і т.д. Але окрім цієї монотонності, про яку потрібно не забути у відповідний момент, рослини хворіють.

До основних проблем, які виникають під час вирощування рослин, належать:

- перезволоження;
- недостатній полив;
- недостатня кількість сонця;
- надлишок сонця;
- шкідники;
- коренева гниль [1].

Потрібно розпізнати хворобу, зрозуміти, як її вилікувати. Тобто це вже величезний набір дій. Але він ще більше збільшується, чим більше рослин має людина у себе вдома. Тоді для підтримки даних процесів, для їхнього впорядкування потрібно надати зручну множину інструментів у складі програмного забезпечення, яке має бути розроблено в роботі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз програмних засобів допомоги у вирощуванні рослин

Planta [2] (рис. 1.1) є програмним додатком, який є достатньо ефективним у підтримці вирощування рослин, надаючи можливості внесення власних рослин, отримування інформації про необхідність виконання тих або інших дій з догляду за рослинами, що дозволяє власникам не забувати вчасно полити рослину, внести добрива і т.д. Також програма надає можливість отримувати різні інструкції з інформацією про, наприклад, поливання рослин.

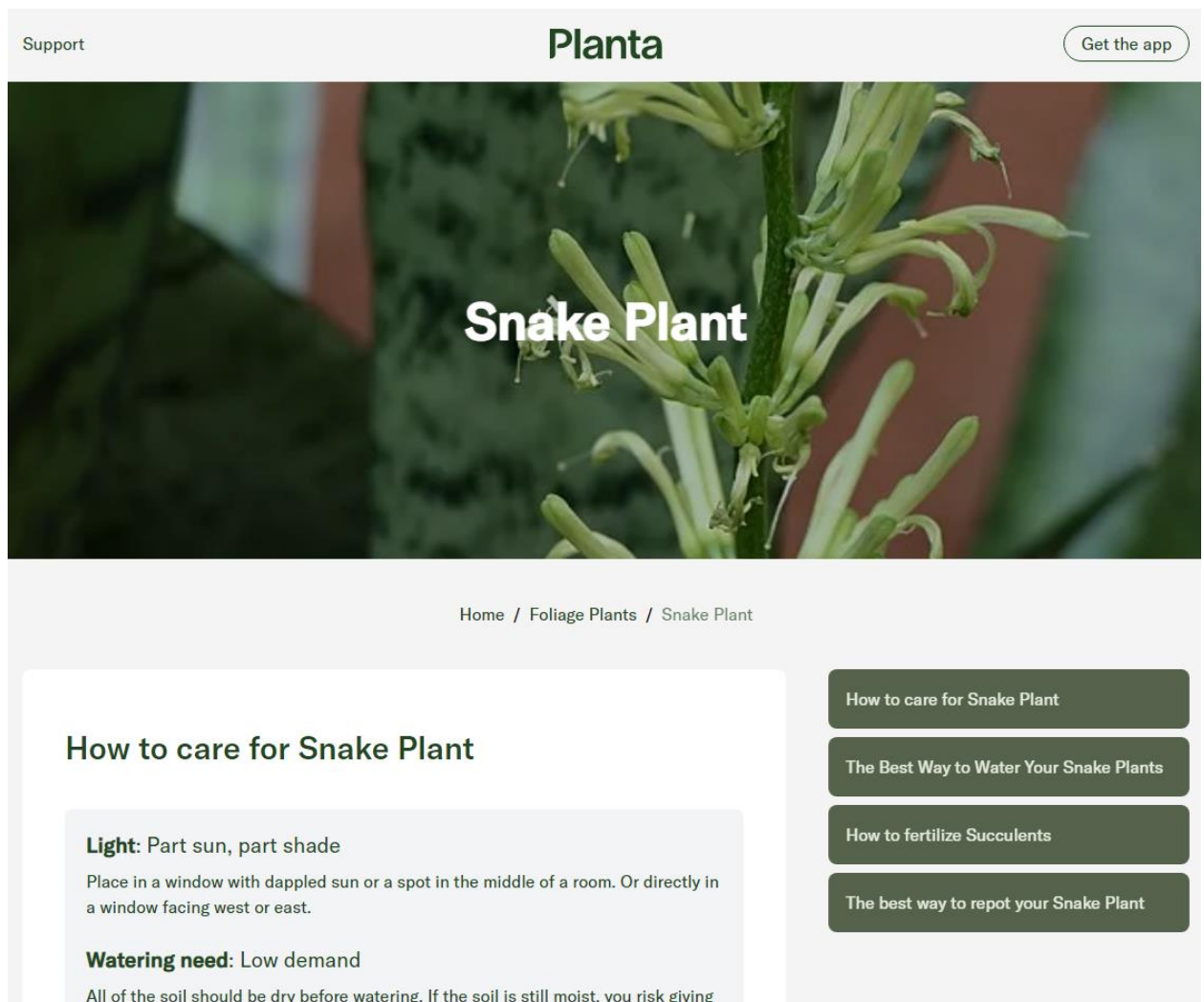


Рисунок 1.1 – Інтерфейс Planta

Стосовно даного додатку важливим є те, що Planta надає тільки частину функцій безкоштовно. Наприклад, для планування поливання рослин, отримання інструкцій про полив не потрібно сплачувати гроші, а ось за аналогічні дії для внесення добрив, обрізання рослин, пересаджування вже доведеться сплатити 7.99 \$ на місяць [3].

Рослини розділені за приміщеннями, в яких вони розташовані у власника. Тож базова функціональність додатку є достатньо потужною, що було використано від час визначення функцій програми-розробки.

Додаток Vera [4] відрізняється від попереднього тим, що він не має інструкцій, тобто загальна база знань у нього фактично відсутня. Проте він дозволяє створити індивідуальний підхід стосовно власних рослин. Він надає можливість внести всі дані, які стосуються рослин власника, визначити дії з догляду, які виконуються над ними, планувати подальші дії з отриманням відповідних нагадувань. За кожною рослиною можна додавати серію фотографій, створюючи таким чином певний щоденник, що дозволяє потім бачити, як рослина розвивалась. Це в підсумку дозволяє визначити, що саме краще підходить для конкретної рослини і виходити з цього в подальшому. Тобто підхід даного додатку ґрунтується на тому, щоб відмовитись від загальних принципів, а використовувати індивідуальні. Визначаючи ці індивідуальні принципи, користувач фактично формує свої власні інструкції щодо вирощування конкретної рослини. Програмний додаток є безкоштовним і не потребує сплати.

Додаток ThePlantMe [5] також, як і перший додаток, є небезкоштовним і за певні функції передбачає сплату. Фактично ідея цього додатку передбачає певну гейміфікацію вирощування рослини [6]. Коли користувач виконує дії з догляду і одночасно слідкує за станом рослини через фотографії, таким чином наче призводячи до покращення або погіршення стану об'єкта гри. Загалом програма передбачає виконання стандартних для цієї предметної області функцій, описаних вище. Основними точками доступу до функцій є пошук рослин та зона керування

інформацією про власну рослину, що включає швидке збереження виконання дій з догляду, перегляд історії таких дій та опис рослини. Але фактично певною незручністю є те, що щоб отримати всю цю інформацію потрібно відкрити картку з рослиною, тобто доступ відбувається за кожною рослиною. У той же час, якщо рослин багато, то відслідковувати, для якої рослини сьогодні потрібно виконати якісь дії, а для якої – ні, стає проблемним. Більш зручним у таких випадках є отримання інформації одразу про всі рослини. Тож це одна з рис, яку потрібно вказувати при створенні розробки в роботі.

Gardenia [7] є ще одним подібним додатком, але він має свої певні особливості, що є достатньо важливими. Додаток передбачає, по-перше, що дії з догляду можуть розширюватися, тобто він не концентрується тільки на поливанні та внесенні добрив, як це робить більшість попередніх додатків. Також за рослиною, яка додається у додаток, одразу надаються підказки про те, який тип ґрунту необхідний для неї, якої мінімальної температури вона потребує і т.д. Це є достатньо інформативним підходом, який дозволяє оцінювати рослину з практичних позицій і також отримувати потрібну базову інформацію швидко, що необхідно може бути в подальшому багато разів. Тобто взимку за специфічними рослинами можна відслідковувати швидко мінімальну температуру без необхідності виконання окремого пошуку.

Загалом слід визначити, що подібні додатки мають певний набір функцій, які у них повторюються і які були виділені для подальшої розробки у складі програми, як базово необхідні, адже очікуються користувачами у цій предметній області як базові.

1.2 Визначення концепції розроблюваної програми

Проведений аналіз дозволив встановити, що ця предметна область є достатньо популярною, тобто дійсно користувачі, які вирощують рослини, потребують подібного роду додатків, проте було виявлено і ряд прогалин

стосовно функціональності (табл. 1.1). Так програми не мають індивідуального підходу в підтримці садівників. Наприклад, якщо садівник зіштовхнувся з проблемою, він як раз потребує саме такого підходу. Виявлено проблему з рослиною, необхідно встановити захворювання, поради лікування, виходячи з індивідуального випадку. Однак наявні програми таких можливостей не надають. Використання цієї логіки в розробці дозволяє повернути до неї іншу сторону – експертів. Це люди, які мають знання і які можуть допомогти садівникам з одного боку, а з іншого боку – програмній системі своїми знаннями. Окрім того, використовуючи для роботи з експертами, які отримують високі оцінки, обмеження щодо необхідності потенційно платної підписки, надається можливість сплатити за роботу експертів, таким чином їх додатково зацікавлюючи. Зважаючи на це все, запропонована концепція програми може бути на практиці доволі ефективною.

Таблиця 1.1 – Результати аналізу програмних засобів допомоги у вирощуванні рослин

Ознака	Planta	Vera	Розробка
Пошук видів рослин	+	—	+
Внесення власних рослин	+	+	+
Планування процедур з догляду	+	+	+
Отримання інструкцій	+	—	+
Наявність функцій стосовно захворювань рослин	—	—	+
Отримання порад від експертів	—	—	+

Додаток Planta використаний як прототип для подальшої розробки, адже він має більше узагальнених засобів, проте деякі функціональні можливості інших додатків, зокрема Gardenia з інформативними підказками за рослинами, були визначені також для розробки у програмі.

1.3 Висновки за розділом 1 та постановка завдань роботи

Аналіз існуючих програмних засобів допомоги у вирощуванні рослин дозволив виявити основні функції таких програми, а також наявні прогалини. Це дозволило в підсумку сформулювати концепцію програми-розробки.

Мета роботи, що полягає в розробленні програмного забезпечення допомоги у вирощуванні рослин для спрощення керування процесом вирощування рослин і виявленими під час нього проблемами, передбачає виконання таких завдань:

- аналіз програмних аналогів і визначення концепції програми;
- проєктування програмного забезпечення допомоги у вирощуванні рослин;
- реалізація вебдодатку допомоги у вирощуванні рослин.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір технологій розробки

Для розв’язання завдання в другому розділі потрібно було розпочати з вибору технологій, що включають мову програмування і фреймворк, що буде використовуватися для веброзробки. Мовою програмування встановлено Python [8], адже вона є гнучкою, підтримує веброзробку на високому рівні, спрощує розробку. Для веброзробки встановлено використовувати Django, адже порівняно з Flask він має більше вбудованих засобів, таким чином краще забезпечуючи захищеність даних і функцій, дозволяє краще реагувати на ризики подальшого розширення функціональності (табл. 2.1) [9]-[10]. Вибір Django визначив використання в подальшому архітектурного патерну Model-View-Template (MVT).

Таблиця 2.1 – Порівняння основних Python-фреймворків

Ознака	Django	Flask
Обсяг розроблюваної функціональності, для якого призначений фреймворк	Великий	Малий
Захищеність	Багато засобів з автоматичним оновленням	Вимагає встановлення додаткових рішень, відсутнє автоматичне оновлення, потребує додаткових зусиль
Ризики під час розширення додатку в майбутньому	Нижчі	Вищі

2.2 Моделювання вимог до програми

Садівник є стандартним користувачем програми, який виконує пошук інформації в програмі та виконує певні дії, пов'язані з власними рослинами. Садівник може авторизуватися або ні. У будь-якому випадку такий користувач розглядається як садівник.

Садівник має наступний набір сценаріїв:

- перегляд інформації про вид рослини;
- перегляд списку керівництв з вирощування рослин;
- перегляд керівництва з вирощування рослини;
- збереження керівництва з вирощування рослин у власну теку;
- перегляд збережених керівництв з власної теки;
- пошук керівництв з вирощування рослин;
- перегляд розв'язань проблем за експертом;
- перегляд специфікацій процедур з догляду рослин, визначених експертом;
- запит поради у експерта;
- перегляд отриманих порад від експертів за рослиною;
- оформлення підписки садівником на програму;
- оформлення підписки садівником на експерта;
- робота з власними рослинами;
- перегляд оцінювань за експертом;
- оцінювання експерта;
- перегляд порад, наданих експертом;
- визначення проблеми, виявленої у рослини;
- перегляд випадків лікування за хворобою;
- пошук видів рослин;
- пошук експертів;
- робота з власними приміщеннями;
- перегляд останніх розв'язаних проблем експертами з підписки;

- авторизація;
- перегляд процедур з догляду за вказаним видом рослини;
- перегляд інформації про садівника;
- перегляд інформації про експерта.

Сценарій перегляду списку керівництв з вирощування рослин узагальнює 3 окремі сценарії:

- перегляд керівництв з вирощування рослин експертів з підписки садівника;
- перегляд керівництв з вирощування за видом рослини;
- перегляд списку керівництв з вирощування рослин експерта.

Сценарій запиту поради у експерта узагальнює таким чином 3 окремі сценарії:

- запит поради про власну рослину;
- запит загальної поради;
- запит поради про проблему.

Сценарій роботи з власними приміщеннями включає наступні сценарії:

- створення запису про приміщення;
- визначення поточних умов у приміщенні;
- перегляд власних приміщень.

Сценарій роботи садівника з власними рослинами включає наступні сценарії (рис. 2.1):

- внесення власної рослини;
- внесення виконаних дій з догляду за рослиною;
- перегляд переліку власних рослин;
- перегляд історії доглядів за рослиною;
- визначення періодичності догляду за рослиною;
- перегляд календаря догляду.

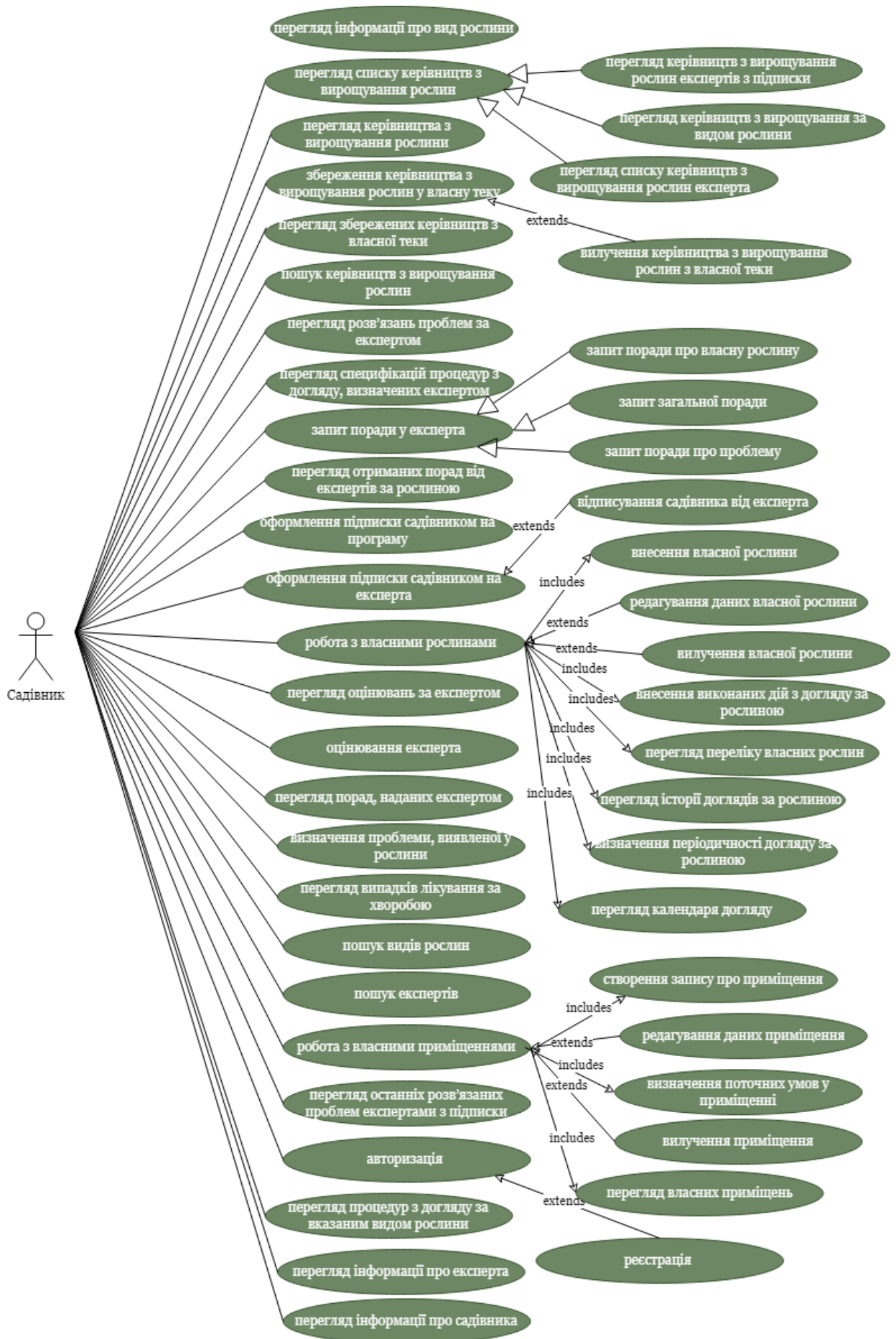


Рисунок 2.1 – Діаграма прецедентів садівника

Також виявлено наступні альтернативні сценарії:

- вилучення керівництва з вирощування рослин з власної теки, що реалізується в тому випадку, якщо збереження керівництва з вирощування рослин у власну теку неможливе, адже це керівництво вже збережене раніше;
- редагування даних власної рослини, що реалізується тоді, коли дані про власну рослину більше неактуальні і потрібно їх змінити;
- вилучення власної рослини, що реалізується тоді, коли рослини більше немає у садівника або він не хоче, щоб вона відображалась у програмі;
- редагування даних приміщення, що реалізується тоді, коли дані про певне приміщення садівника більше не актуальні і потрібно їх змінити;
- вилучення приміщення, що реалізується тоді, коли садівник не хоче, щоб серед його приміщень далі було вказане;
- реєстрація, що реалізується в тому випадку, коли садівник не має власного облікового запису.

Експерт може надавати поради садівникам щодо виявлених випадків хвороб, ділитися власним досвідом, знаннями щодо вирощування рослин (рис. 2.2).

Робота експерта реалізується через наступні сценарії:

- перегляд оцінювань за експертом;
- перегляд розв'язань проблем за експертом;
- пошук видів рослин;
- робота з керівництвами з вирощування рослин;
- пошук експертів;
- перегляд інформації про вид рослини;
- перегляд випадків лікування за хворобою;
- робота з процедурами з догляду за видами рослин;
- авторизація;
- робота з порадами з догляду;
- перегляд інформації про садівника;

– перегляд інформації про експерта.



Рисунок 2.2 – Діаграма прецедентів експерта

Сценарій роботи з керівництвами з вирощування рослин включає наступні сценарії:

- перегляд керівництва з вирощування рослини;
- перегляд списку керівництв з вирощування рослин;
- створення керівництва з догляду за рослинами;
- пошук керівництв з вирощування рослин.

Сценарій перегляду списку керівництв з вирощування рослин узагальнює 3 окремі сценарії:

- перегляд керівництв з вирощування за видом рослини;
- перегляд списку керівництв з вирощування рослин експерта;

- перегляд списку власних керівництв з догляду за рослинами.

Сценарій роботи з процедурами з догляду за видами рослин включає наступні сценарії:

- створення власної специфікації процедури з догляду за видами рослин;

- перегляд специфікацій процедур з догляду, визначених експертом;
- перегляд процедур з догляду за вказаним видом рослини.

Сценарій роботи з порадами з догляду включає наступні сценарії:

- перегляд запитаних порад;
- перегляд порад, наданих експертом;
- надання поради з вирощування.

Сценарій надання поради з вирощування узагальнює 3 окремі сценарії:

- надання поради про проблему;
- надання загальної поради;
- надання поради про власну рослину.

Також виявлено наступні альтернативні сценарії:

- редагування власного керівництва з догляду за рослинами, що реалізується в тому випадку, коли експерт хоче оновити інформацію в керівництві;

- редагування власної специфікації процедури з догляду за видами рослин, що реалізується в тому випадку, коли експерт хоче оновити інформацію у власній специфікації процедури з догляду за видами рослин;

- реєстрація, що реалізується в тому випадку, коли експерт не має власного облікового запису.

2.3 Проєктування бази даних

Для підтримки організації роботи з базою даних (БД) застосовано PostgreSQL.

Категорії рослин необхідні для узагальнення видів рослин у звичному для звичайної людини представленні. Це можуть бути квіти, трави і т.д. Такі категорії накопичуються в таблиці БД Plantcategory, для чого у ній визначені поля:

- id – ідентифікатор загальної категорії рослин;
- category_name – назва категорії рослин;
- pic – фотографія, яка візуалізує категорію рослин у програмі.

Для визначення рослин у біологічних термінах потрібно мати інформацію про сімейства рослин. Сімейства рослин накопичуються в таблиці Plantfamily, для чого у ній визначені поля:

- id – ідентифікатор сімейства рослин;
- family_name – назва сімейства рослин;
- pic – фотографія представника цього сімейства рослин.

Види рослин накопичуються в таблиці БД Plantspecies, для чого у ній визначені поля:

- id – ідентифікатор виду рослин;
- plant_category – категорія рослин, до якої належить цей вид, представлена зовнішнім ключем з таблиці Plantcategory;
- plant_family – сімейство, до якого належить рослина, представлена зовнішнім ключем з таблиці Plantfamily;
- species_name – назва цього виду рослин;
- leaf_color – колір листя у цього виду рослин;
- flower_color – колір квітки у цього виду рослин;
- growth_rate – загальна швидкість росту цього виду рослин;
- duration – тривалість життя цього виду рослин;
- water_requirements – вимогливість цього виду рослин до поливу;
- fertilization_requirements – вимогливість цього виду рослин до внесення добрив;
- soil_requirements – склад ґрунту, необхідний для цього виду рослин;

- `light_requirements` – ступінь освітлення, необхідна для цього виду рослин;
- `plant_type` – домашня рослина або садова;
- `temperature` – мінімальна температура, необхідна цьому виду рослин;
- `blooming` – частота цвітіння цієї рослини;
- `humidity` – вимогливість до вологості повітря цього виду рослин;
- `potting` – горщик, необхідний для висаджування цього виду рослин, або відсутність такої необхідності, якщо рослина одразу висаджується у відкритий ґрунт;
- `propagation` – спосіб розмноження цього виду рослин;
- `pic` – фотографія цього виду рослин для візуалізації у програмі.

Вигоди для здоров'я, якими характеризується вплив споживання або присутності рослини на людину, мають бути зафіксованими, щоб надати додаткову інформацію користувачу, але для цього потрібно спочатку визначити, на що взагалі рослини можуть впливати. Це може бути мозок, зір і т.д. Вигоди для здоров'я накопичуються в таблиці `Healthbenefit`, для чого у ній визначені поля:

- `id` – ідентифікатор вигоди для здоров'я;
- `benefit_name` – назва вигоди для здоров'я;
- `pic` – фотографія, яка використовується для візуалізації цієї вигоди у програмі.

Конкретний вплив виду рослини на вигоду для здоров'я накопичується в таблиці `PlantspeciesHealthbenefit`, для чого у ній визначені поля:

- `id` – ідентифікатор впливу виду рослини на вигоду для здоров'я людини;
- `plant_species` – вид рослин, якого стосується цей запис, представлений зовнішнім ключем з таблиці `Plantspecies`;
- `healthbenefit` – вигода для здоров'я, якої стосується цей запис і представлений зовнішнім ключем з таблиці `Healthbenefit`;

– full_info – повна інформація про вплив на цю вигоду для здоров'я цього виду рослин;

– level – числове значення ступеню впливу.

Види рослин представлені конкретними рослинами, які має садівник або людина, яка вирощує рослину чи рослини. Якщо це не домашня рослина, а садова, то це може бути не конкретна рослина, а фактично група рослин. Тобто під рослиною маються на увазі, наприклад, помідори, які ростуть на городі. Тоді звичайно, що кожен помідор не вноситься окремо, оскільки він не має індивідуальних характеристик. Якщо вдома росте декілька рослин однакового виду, то вони можуть вноситися окремо. Рослини накопичуються в таблиці БД Plant, для чого у ній визначені поля:

– id – ідентифікатор рослини;

– plant_species – вид цієї рослини, представлений зовнішнім ключем з таблиці Plantspecies;

– gardener – садівник, у якого росте ця рослина, представлений зовнішнім ключем з таблиці Gardener;

– plant_name – фактично ім'я (псевдонім) рослини або будь-яка позначка, щоб ідентифікувати окрему рослину, якщо це необхідно;

– memo – запис про цю рослину;

– gardener_location – приміщення садівника, де розташовано рослину (тип кімнати або сад, наприклад), представлений зовнішнім ключем з таблиці Location;

– adopted – момент, коли було посаджено рослину або отримано її садівником;

– pic – основна фотографія рослини;

– active – наявність рослини у садівника на цей момент: щоб не вилучати повністю дані з БД, використовується це поле, щоб визначити, чи можна відображати інформацію про цю рослину, чи вона не має використовуватися.

Рослини можуть уживатися з одними рослинами і не уживатися поруч з іншими. Це важлива інформація. Вона накопичується в таблиці Companion, для чого у ній визначені поля:

- id – ідентифікатор запису про допустимість спільного існування рослин різних видів;
- plant_species_source – перший вид рослини у парі, представлений зовнішнім ключем з таблиці Plantspecies;
- plant_species_companion – другий вид рослини у парі, представлений зовнішнім ключем з таблиці Plantspecies;
- companion_type – ці види рослин можуть уживатися поруч або навпаки (є ворогами).

Звичайні користувачі програми є садівниками, а інформація про них накопичується в таблиці Gardener, для чого у ній визначені поля:

- id – ідентифікатор садівника;
- name – ім'я садівника;
- login – логін облікового запису садівника;
- password – пароль облікового запису садівника;
- email – електронна адреса садівника;
- phone – номер телефону садівник;
- memo – ознайомчий запис про садівника;
- pic – фотографія садівника.

Садівники можуть про свої рослини створювати нотатки. Вони накопичуються в таблиці PlantNote, для чого у ній визначені поля:

- id – ідентифікатор нотатки про рослину садівника;
- plant – рослина, про яку визначена ця нотатка, представлена зовнішнім ключем з таблиці Plant;
- memo – текст нотатки про цю рослину садівника.

Іноді може бути важливим відслідкувати, як змінюється стан рослини. Це може бути корисним і під час виявлення хвороб, і під час визначення розвитку рослини, адже згадати це, особливо якщо рослин у садівника

багато, може бути проблемно. Дані про минулий стан рослин накопичуються в таблиці БД PlantHistory, для чого у ній визначені поля:

- id – ідентифікатор минулого стану рослини;
- plant – рослина, минулий стан якої визначено, представлена зовнішнім ключем з таблиці Plant;
- pic – фотографія рослини в цей момент;
- memo – опис стану рослини в цей момент;
- pic_data – дата створення фотографії, а загалом і фактично цього запису.

Дії, які виконує садівник з рослинами, включають певний набір. Це може бути поливання, внесення добрив, підрізання і т.д. Цей набір може бути різним, може розширюватися. Тому при збереженні даних ці дії винесені окремо. Дані про дії з догляду за рослинами накопичуються в таблиці Activity, для чого у ній визначені поля:

- id – ідентифікатор дії з догляду за рослиною;
- activity_name – назва дії;
- pic – фотографія, яка використовується в програмі для візуалізації дії.

Дії самі по собі важливі, але фактично в програмі садівник може використовувати їх для того, щоб визначати, які їх виконувати для своєї власної рослини. Відповідно зв'язування рослини з дією накопичуються в таблиці PlantActivity, для чого у ній визначені поля:

- id – ідентифікатор правила застосування дії з догляду до рослини;
- plant – рослина, до якої застосовується ця дія з догляду, представлена зовнішнім ключем з таблиці Plant;
- activity – дія з догляду, яка застосовується до цієї рослини, представлена зовнішнім ключем з таблиці Activity;
- periodicity – періодичність (кількість днів), з якою ця дія має виконуватися в цих умовах;
- start_period – час року, коли такий план має розпочатися;

- end_period – час року, коли такий план має закінчитися;
- next_activity – наступна дата, коли ця дія має бути виконана відносно даного моменту: кожного разу, коли дія виконується, ця дата має бути оновлена для рослини з врахуванням вказаної періодичності;
- conditions – умови, за яких до цієї рослини має застосовуватися ця дія за цим правилом: це числове значення, яке визначає особливості стану навколишнього середовища;
- memo – додатковий опис до умов виконання цієї дії.

Дії з догляду за рослинами не тільки плануються, але і зберігаються вже виконані, таким чином формуючи історію, щоб потім її можна було відслідкувати. Ці дані накопичуються в таблиці PlantActivityHistory, для чого у ній визначені поля:

- id – ідентифікатор виконаного догляду за рослиною;
- plantactivity – дія з догляду за рослиною, яку було виконано, представлена зовнішнім ключем з таблиці PlantActivity;
- done_date – момент, коли цей догляд було виконано;
- memo – запис про особливості виконання, якщо потрібно.

Окрім визначення власних запланованих або виконаних дій про рослину, яка належить садівнику, він може бути зацікавлений у пошуку інформації про те, як краще це робити. Ця інформація надається експертами або певним загальним правилом (базовий експерт – адміністратор, який такі правила має визначати) і накопичується в таблиці PlantSpeciesActivity, для чого у ній визначені поля:

- id – ідентифікатор правила застосування дії з догляду за рослиною до певного виду рослин;
- plant_species – вид рослин, до якого має застосовуватися ця дія з догляду за цим правилом, представлений зовнішнім ключем з таблиці PlantSpecies;
- activity – дія з догляду, яка має застосовуватися до цього виду рослин за цим правилом, представлений зовнішнім ключем з таблиці Activity;

– periodicity – періодичність, з якою цей догляд має виконуватися для цього виду рослин;

– start_period – час року, коли такий догляд має розпочатися;

– end_period – час року, коли такий догляд має закінчитися;

– conditions – умови, за яких до цієї рослини має застосовуватися ця дія за цим правилом: це числове значення, яке визначає особливості стану навколишнього середовища;

– memo – додаткові особливості застосування;

– adviser – експерт, який визначив це правило, представлений зовнішнім ключем з таблиці Adviser.

Захворювання рослин (самі назви) накопичуються в таблиці Disease, для чого у ній визначені поля:

– id – ідентифікатор захворювання рослин;

– disease_name – назва захворювання рослин;

– memo – опис цього захворювання.

Зв'язування захворювань з видами рослин виконується через окрему таблицю. Це необхідно через те, що деякі захворювання можуть спостерігатися у різних (наприклад, споріднених) видів рослин. І звичайно, що кожен вид рослин може мати різні хвороби, тому це потребувало виділення окремої таблиці, реалізуючи зв'язок багато-до-багатьох. Це таблиця PlantspeciesDisease, для чого у ній визначені поля:

– id – ідентифікатор хвороби у виду рослин;

– plant_species – вид рослини, у якого спостерігається ця хвороба, представлений зовнішнім ключем з таблиці Plantspecies;

– disease – хвороба, яка спостерігається у цього виду рослин, представлена зовнішнім ключем з таблиці Disease;

– precondition – ознаки хвороби у рослини;

– treatment – лікування цієї хвороби у цього виду рослин.

Конкретні випадки хвороб, які були вилікувані у рослин певного виду експертом, вносяться в окрему таблицю DiseaseCase, для чого у ній визначені поля:

- id – ідентифікатор захворювання рослини (це може бути і узагальнений досвід з лікування декількох рослин цим експертом);
- adviser – експерт, який виявив цю хворобу, представлений зовнішнім ключем з таблиці Adviser;
- plantspecies_disease – хвороба цього виду рослин, якої стосується цей випадок, представлена зовнішнім ключем з таблиці PlantspeciesDisease;
- pic – фотографія цього випадку;
- precondition – ознаки хвороби у рослини, які виявив експерт;
- treatment – лікування цієї хвороби, яке виконував експерт.

Виявлені випадки захворювань власних рослин садівники визначають як проблеми, які мають бути вирішені. Проблеми накопичуються в таблиці Problem, для чого у ній визначені поля:

- id – ідентифікатор проблеми з рослиною;
- plant – рослина, проблему з якою виявлено, представлена зовнішнім ключем з таблиці Plant;
- memo – опис випадку;
- stage – стан вирішення цієї проблеми;
- done_date – момент, коли цю проблему було виявлено;
- solved_date – момент, коли проблему було вирішено;
- treatment – лікування, яке фактично було застосовано садівником;
- result – результат, який було отримано після лікування (опис).

Для ілюстрації проблеми садівник може створювати множину фотографій, тому їх було винесено в окрему таблицю БД ProblemPic, для чого у ній визначені поля:

- id – ідентифікатор фотографії проблеми;
- problem – проблема, якої стосується ця фотографія, представлена зовнішнім ключем з таблиці Problem;

– pic – фотографія, яка ілюструє проблему.

Поради, які надають експерти для лікування з метою вирішення проблеми, що виникла з рослиною, накопичуються в таблиці ProblemAdvice, для чого у ній визначені поля:

- id – ідентифікатор поради для лікування рослини;
- problem – проблема, якої стосується ця порада, представлена зовнішнім ключем з таблиці Problem;
- adviser – експерт, який визначив цю пораду, представлений зовнішнім ключем з таблиці Adviser;
- memo – детальний опис лікування;
- done_date – момент, коли цю пораду було створено;
- disease – встановлена хвороба, представлена зовнішнім ключем з таблиці Disease.

Експерти є окремим користувачами системи, які фактично є джерелом інформації щодо вирощування, догляду за рослинами. Інформація про них накопичується в таблиці Adviser, для чого у ній визначені поля:

- id – ідентифікатор експерта;
- login – логін облікового запису експерта;
- password – пароль облікового запису експерта;
- first_name – ім'я експерта;
- last_name – прізвище експерта;
- email – електронна адреса експерта;
- phone – номер телефону експерта;
- memo – ознайомчий запис про експерта;
- pic – фотографія експерта;
- level – середній рівень експерта, визначений за оцінками садівників;
- open – експерт працює тільки за наявності підписки на програму в садівника або він відкритий до всіх звернень;
- active – доступ експерта до роботи з програмою.

На експертів можна підписуватися, щоб слідкувати за їхніми керівництвами з вирощування рослин, наданими порадами. Дані підписок на експертів накопичуються в таблиці `AdviserFollowing`, для чого у ній визначені поля:

- `id` – ідентифікатор підписки на експерта;
- `gardener` – садівник, який підписався на експерта, представлений зовнішнім ключем з таблиці `Gardener`;
- `adviser` – експерт, на якого підписався садівник, представлений зовнішнім ключем з таблиці `Adviser`.

Садівник може не очікувати відповідей на виявлені ним проблеми, а самостійно звернутися до експерта з проханням порадити йому, як проблему вирішити. Ці дані накопичуються в таблиці `AdviserAskProblem`, для чого у ній визначені поля:

- `id` – ідентифікатор запиту до експерта про пораду щодо виявленої проблеми з рослиною;
- `adviser` – експерт, до якого звернений запит, представлений зовнішнім ключем з таблиці `Adviser`;
- `problem` – проблема, якої стосується цей запит, представлена зовнішнім ключем з таблиці `Problem`;
- `stage` – стан вирішення проблеми;
- `done_date` – момент, коли було створено запит.

Запити до експертів можуть стосуватися і рослин загалом, а не тільки виявленої проблеми. Такі запити накопичуються в таблиці `AdviserAskPlant`, для чого у ній визначені поля:

- `id` – ідентифікатор запиту до експерта про рослину;
- `adviser` – експерт, до якого звернений запит, представлений зовнішнім ключем з таблиці `Adviser`;
- `plant` – рослина, якої стосується запит, представлена зовнішнім ключем з таблиці `Plant`;
- `question` – запитання до експерта;

- answer – відповідь експерта;
- open – можливість відкрити всі дані про рослину експерту;
- ask_date – момент, коли було створено запит;
- answer_date – момент, коли було отримано відповідь.

Також запити до експертів можуть стосуватися загалом садівництва, а не окремої рослини. Такі дані накопичуються в таблиці AdviserAsk, для чого у ній визначені поля:

- id – ідентифікатор загального запиту до експерта;
- gardener – садівник, який створив цей запит, представлений зовнішнім ключем з таблиці Gardener;
- adviser – експерт, до якого звернений запит, представлений зовнішнім ключем з таблиці Adviser;
- question – запитання до експерта;
- answer – відповідь експерта;
- open – можливість відкрити всі дані про всі рослини садівника експерту;
- ask_date – момент, коли було створено запит;
- answer_date – момент, коли було отримано відповідь.

Садівник може використовувати програму для пошуку нових видів рослин, які б він хотів посадити. Для цього він може вносити їх до власного списку для збереження. Ці дані накопичуються в таблиці Wishlist, для чого у ній визначені поля:

- id – ідентифікатор виду рослин у збереженні садівником;
- gardener – садівник, який зберіг цей вид рослин;
- plant_species – вид рослин, який було збережено, представлений зовнішнім ключем з таблиці Plantspecies;
- memo – запис, який деталізує плани садівника відносно цієї рослини.

Керівництва з вирощування рослин є одним з напрямків роботи експертів з програмою. Вони надають їх, демонструючи свій власний рівень,

формує навколо себе певну нішу садівників. Дані про керівництва накопичуються в таблиці Tutorial, для чого у ній визначені поля:

- id – ідентифікатор керівництва з вирощування рослин;
- adviser – експерт, який створив це керівництво, представлений зовнішнім ключем з таблиці Adviser;
- title – назва керівництва;
- all_text – текст керівництва;
- pic – фотографія, яка використовується для ілюстрації в керівництві (ілюструючі фотографії мають надаватися посиланнями в тексті керівництва, програмно підвантажуючись);
- done_date – момент, коли було створено керівництво;
- open – для експерта з доступом за підпискою визначає, чи є це керівництво відкритим: необхідно, щоб садівник, який не має підписки, міг ознайомитись з доступними можливостями.

Керівництво може стосуватися садівництва загалом, деяких дій з догляду, видів рослин. Це можуть бути різні співвідношення, тому такі відношення було винесено в окремі таблиці. Відношення між керівництвом і діями з догляду накопичуються в таблиці TutorialActivity, для чого у ній визначені поля:

- id – ідентифікатор відношення керівництва до дії з догляду;
- tutorial – керівництво, якого стосується ця дія з догляду, представлене зовнішнім ключем з таблиці Tutorial;
- activity – дія з догляду, якої стосується керівництво, представлена зовнішнім ключем з таблиці Activity.

Відношення між керівництвом і видами рослин накопичуються в таблиці TutorialPlantspecies, для чого у ній визначені поля:

- id – ідентифікатор відношення керівництва до виду рослин;
- tutorial – керівництво, якого стосується цей вид рослин, представлене зовнішнім ключем з таблиці Tutorial;

– `plant_species` – вид рослин, якого стосується це керівництво, представлене зовнішнім ключем з таблиці `Plantspecies`.

Садівники можуть собі зберігати керівництва з вирощування рослин. Ці збереження накопичуються в таблиці БД `TutorialSave`, для чого у ній визначені поля:

- `id` – ідентифікатор збереження керівництва з вирощування рослин садівником для себе;
- `tutorial` – керівництво, яке було збережено, представлене зовнішнім ключем з таблиці `Tutorial`;
- `gardener` – садівник, який зберіг це керівництво, представлений зовнішнім ключем з таблиці `Gardener`.

Для отримання доступу до керівництв з вирощування рослин та порад експертів, які надають їх з обмеженим доступом, потрібно сплатити підписку. Дані про підписки садівників накопичуються в таблиці `Subscribe`, для чого у ній визначені поля:

- `id` – ідентифікатор підписки садівника на програму;
- `gardener` – садівник, який оформив цю підписку, представлений зовнішнім ключем з таблиці `Gardener`;
- `payment` – дані платежа, за яким оформлено цю підписку;
- `sum` – сума, яку було сплачено;
- `done_date` – момент, коли було оформлено підписку;
- `due_date` – момент, до якого діє підписка.

Садівники можуть оцінювати експертів, за тим, на скільки якісно вони надали їм поради, допомогу. Це стає основою для визначення того, чи потрібна платна підписка для отримання порад або читання керівництв з вирощування рослин цього експерта. Оцінювання експерта садівниками накопичуються в таблиці `AdvicerAssess` БД (рис. 2.3).

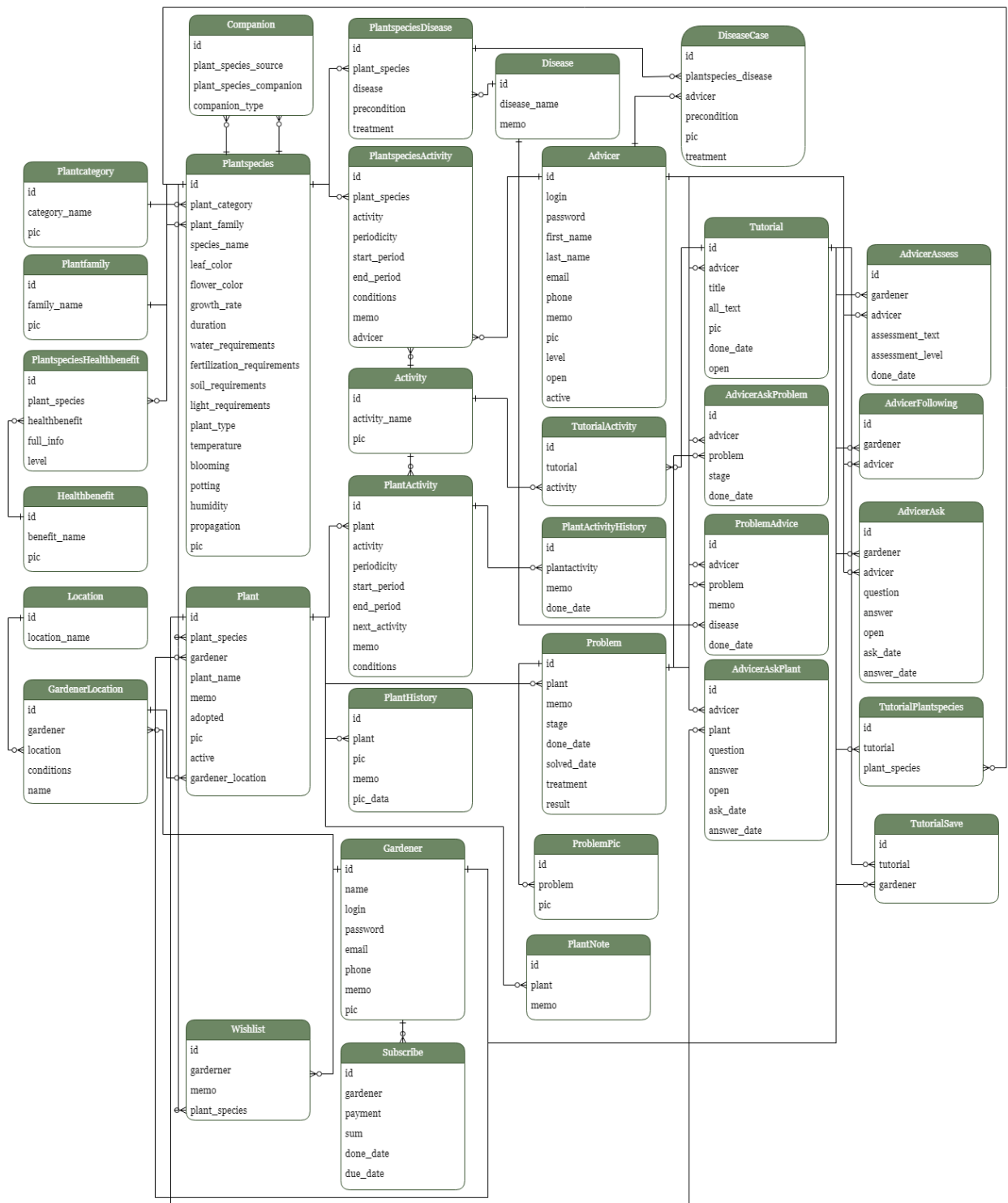


Рисунок 2.3 – Схема БД

Поля таблиці AdviserAssess:

- id – ідентифікатор оцінювання експерта садівником;
- gardener – садівник, який виконав оцінювання, представлений зовнішнім ключем з таблиці Gardener;

- `adviser` – експерт, якого було оцінено, представлений зовнішнім ключем з таблиці `Adviser`.

- `assessment_text` – текст оцінки;

- `assessment_level` – числове значення оцінки;

- `done_date` – момент, коли було виконано оцінювання.

Рослини можуть знаходитися в різних приміщеннях. Варіанти приміщень, доступні в програмі, накопичуються в таблиці БД `Location`, для чого у ній визначені поля:

- `id` – ідентифікатор приміщення (фактично типу приміщення);

- `location_name` – назва приміщення.

Конкретні приміщення, в яких можуть знаходитись рослини у садівника, накопичуються в таблиці `GardenerLocation`, для чого у ній визначені поля:

- `id` – ідентифікатор приміщення садівника;

- `gardener` – садівник, який має це приміщення, представлений зовнішнім ключем з таблиці `Gardener`;

- `location` – тип приміщення, представлений зовнішнім ключем з таблиці `Location`;

- `name` – назва приміщення в інтерпретації садівника (стандартно повторює назву варіанта приміщення);

- `conditions` – поточні умови в цьому приміщенні, які відповідно визначають поточні умови для кожної рослини, що знаходиться в цьому приміщенні.

2.4 Висновки за розділом 2

Усі необхідні завдання з проєктування програми виконано: визначено основну мову для розробки вебдодатку (Python), фреймворк, який підтримує веброзробку (Django), що визначило архітектурний патерн MVT і відповідно структуру вебдодатку в подальшому, сформульовано сценарії роботи

садівників і експертів з програмою, визначено структуру БД, включаючи потрібні сутності предметної області, представлення цих сутностей в реляційній БД на основі таблиць, характеристики цих сутностей, зв'язки між таблицями.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Програмна реалізація функцій експерта

На основі шаблону [11], патерну MVT [12] для реалізації програми було створено ряд відповідних одиниць, що зокрема включають класи моделей, які відповідають описаній структурі БД, тому не описані нижче детально.

Функція програми перегляду оцінювань за експертом реалізована створенням класу `AdviserAssessView` для представлення (рис. 4.1) і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що визначає необхідність авторизації користувача, та `ListView`, що дозволяє керувати переліком оцінок, і з застосуванням моделі `AdviserAssess` в основі. Відбираються у методі `get_queryset` усі оцінювання експерта з логіном, який передано через запит, або авторизованого експерта, якщо такі дані не передано.

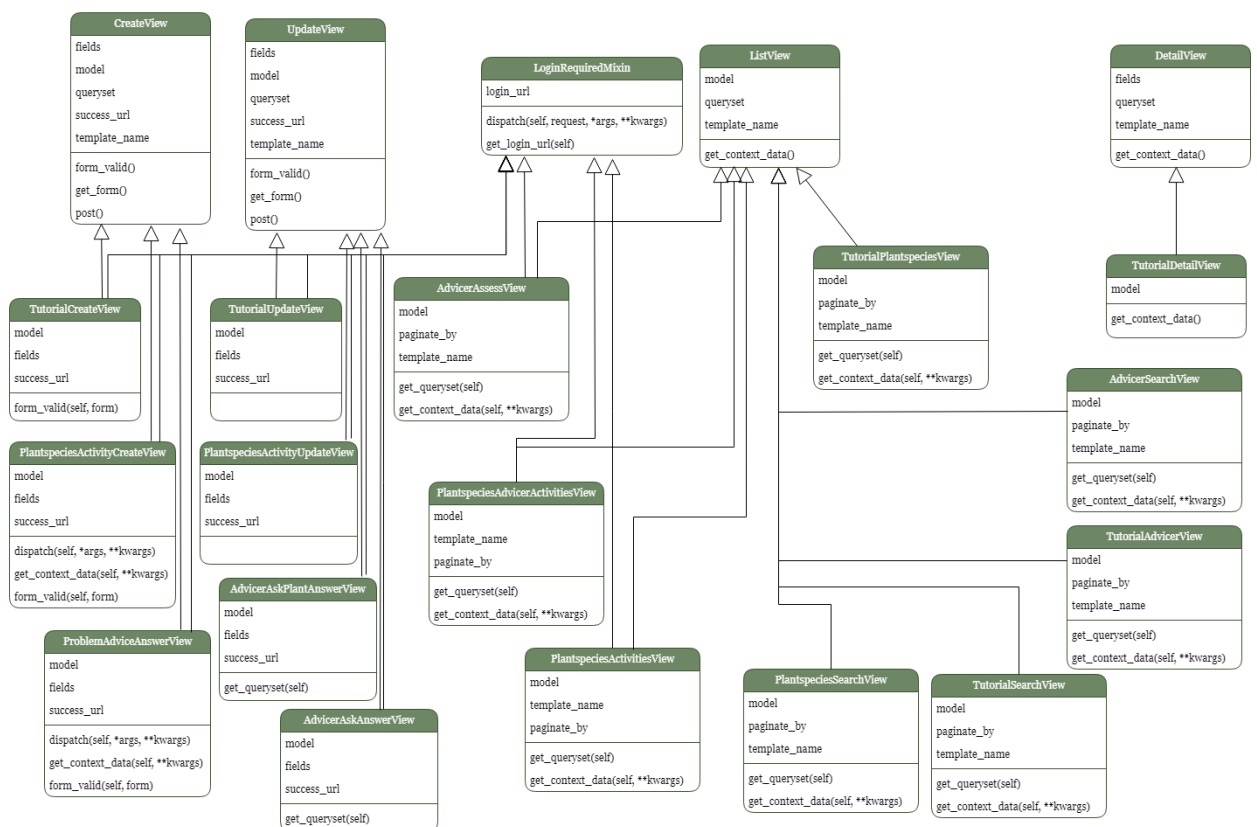


Рисунок 4.1 – Діаграма класів для експерта

Функція програми пошуку видів рослин реалізована створенням класу `PlantspeciesSearchView` для представлення і шаблону вебсторінки для цього представлення через наслідування класу `ListView`, що дозволяє керувати переліком видів рослин, і з застосуванням моделі `Plantspecies` в основі. Відбираються у методі `get_queryset` усі види рослин, які відповідають встановленим критеріям пошуку в запиті: списку категорій рослин в параметрі `category` (вказуються назвами через кому), фразі у назві рослини або описі в параметрі `name`, кольорам листя, вказаним у параметрі `leaf` (вказуються числами через кому), кольорам квітів, вказаним у параметрі `flower` (вказуються числами через кому), варіантам швидкості росту виду рослин у параметрі `growth` (вказуються числами через кому), варіантам тривалості життя рослини у параметрі `duration` (вказуються числами через кому), варіантам вимогливості рослин до поливу в параметрі `water` (вказуються числами через кому), варіантам вимогливості рослин до добрив в параметрі `fertil` (вказуються числами через кому), варіантам необхідного складу ґрунту в параметрі `soil` (вказуються числами через кому), варіантам необхідного ступеня освітлення в параметрі `light` (вказуються числами через кому), типам рослини в параметрі `type` (вказуються числами через кому), варіантам частоти цвітіння рослини в параметрі `bloom` (вказуються числами через кому), варіантам вимогливості до вологості повітря в параметрі `humid` (вказуються числами через кому), варіантам висаджування рослини в параметрі `pot` (вказуються числами через кому), способам розмноження в параметрі `propagation` (вказуються числами через кому), варіантам впливу на здоров'я, вказаних в параметрах `benefit`, де задається назва впливу, та `level` з мінімальним рівнем для відповідного впливу (вказуються в кожному параметрі через кому). У методі `get_context_data` передаються значення параметрів, для яких виконано пошук, щоб їх відобразити в результатах для експерта.

Функція програми перегляду розв'язань проблем за експертом реалізована створенням функції `adviser_solutions_views`. Використання класу

для реалізації функції було не дуже зручним, оскільки ця функція передбачає роботу з декількома моделями одночасно. Вона дозволяє відібрати всі загальні запити і запити за рослиною до експерта з вказаним в параметрі `adviser` логіном або для авторизованого експерта, якщо такий логін не вказано. Відбираються тільки ті запити, за якими поле відповіді непорожнє. Отримані списки разом з об'єктом експерта передаються в шаблон сторінки `solutions.html`.

Функція програми перегляду керівництва з вирощування рослини реалізована створенням класу `TutorialDetailView` для представлення і шаблону вебсторінки для цього представлення через наслідування класу `DetailView`, що дозволяє відтворювати дані за екземпляром моделі, і з застосуванням моделі `Tutorial` в основі. Метод `get_context_data` виконує перевірку усередненої оцінки експерта. Якщо вона вище ніж 4.6 балів, то в контексті даних повертається за ключем `adviser_access` значення `True`, інакше – `False`. У шаблоні цієї вебсторінки виконується перевірка при відображенні даних керівництва з вирощування рослин, чи є воно відкритим. Якщо так, то відображаються всі дані. Якщо ні, то для експерта перевіряється, яке значення було передано за ключем `adviser_access`, а вміст демонструється тільки в тому випадку, якщо значення дорівнює `True`.

Функція програми перегляду керівництв з вирощування за видом рослини реалізована створенням класу `TutorialPlantspeciesView` для представлення і шаблону вебсторінки для цього представлення через наслідування класу `ListView`, що дозволяє відтворювати дані за переліком керівництв, і з застосуванням моделі `Tutorial` в основі. Метод `get_queryset` відбирає всі керівництва, які стосуються виду рослини, назва якого передана через параметр запиту `plant`. Дані виду рослини додаються до контексту через метод `get_context_data`.

Функція програми перегляду списку керівництв з вирощування рослин експерта реалізована створенням класу `TutorialAdviserView` для представлення і шаблону вебсторінки для цього представлення через

наслідування класу `ListView`, що дозволяє відтворювати дані за переліком керівництв, і з застосуванням моделі `Tutorial` в основі. Метод `get_queryset` відбирає всі керівництва, які створені експертом, логін якого переданий через параметр запиту `adviser`, або авторизованим експертом, якщо логін не передано. Дані експерта додаються до контексту через метод `get_context_data`. Таким чином цей клас одночасно підтримує реалізацію і функції програми перегляду списку власних керівництв з догляду за рослинами.

Функція програми пошуку керівництв з вирощування рослин реалізована створенням класу `TutorialSearchView` для представлення і шаблону вебсторінки для цього представлення через наслідування класу `ListView`, що дозволяє керувати переліком керівництв, і з застосуванням моделі `Tutorial` в основі. Відбираються у методі `get_queryset` усі керівництва, які відповідають встановленим критеріям пошуку в запиті: фразі в назві або повному тексті керівництва в параметрі `title`, видам рослин в параметрі `plants` (назви вказуються через кому), діям з догляду в параметрі `activities` (назви вказуються через кому).

Функція програми пошуку експертів реалізована створенням класу `AdviserSearchView` для представлення і шаблону вебсторінки для цього представлення через наслідування класу `ListView`, що дозволяє керувати переліком експертів, і з застосуванням моделі `Adviser` в основі. Відбираються у методі `get_queryset` усі експерти, які відповідають встановленим критеріям пошуку в запиті: логіну або його частині в параметрі `login`, прізвищі та імені в параметрі `name` (прізвища та ім'я відділяються пробілом, можуть бути вказані в будь-якому порядку), електронній пошті експерта в параметрі `email`, мінімальному усередненому рівню за оцінкою, вказаному в параметрі `level_min`, та максимальному усередненому рівню за оцінкою, вказаному в параметрі `level_max`, кількості підписників, що має бути не меншою ніж значення, встановлене в параметрі `followers`. У методі `get_context_data`

передаються значення параметрів, для яких виконано пошук, щоб їх відобразити в результатах.

Функція програми перегляду випадків лікування за хворобою реалізована створенням функції `plant_diseases_views`, оскільки вона передбачала роботу з декількома моделями. У функції відбувається витягання даних про хворобу з назвою `disease_name`, а далі за нею відбираються всі випадки такої хвороби та поради експертів за такими хворобами, які були успішно завершені.

Функція програми створення керівництва з догляду за рослинами реалізована створенням класу `TutorialCreateView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `CreateView`, що підтримує роботу з формою створення екземпляру моделі, і з застосуванням моделі `Tutorial` в основі. Метод `form_valid` заповнює поле `adviser` з даними експерта, який створив це керівництво з вирощування рослин, за допомогою даних авторизованого експерта.

Функція програми редагування власного керівництва з догляду за рослинами реалізована створенням класу `TutorialUpdateView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `UpdateView`, що підтримує роботу з тією ж формою, що і для створення, але вже для редагування існуючого екземпляру моделі, і з застосуванням моделі `Tutorial` в основі.

Функція програми створення власної специфікації процедури з догляду за видами рослин реалізована створенням класу `PlantspeciesActivityCreateView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `CreateView`, що підтримує роботу з формою створення екземпляру процедури догляду, і з застосуванням моделі `PlantspeciesActivity` в основі. Метод `dispatch` за назвою виду рослини та дії з

догляду вносить відповідні об'єкти. Метод `get_context_data` передає ці об'єкти на сторінку, а разом з ними об'єкт, який визначає параметри для цієї дії та цього виду рослин. Метод `form_valid` заповнює поле `adviser` з даними експерта, який створив цю процедуру, за допомогою даних авторизованого експерта.

Функція програми редагування власної специфікації процедури з догляду за видами рослин реалізована створенням класу `PlantspeciesActivityUpdateView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `UpdateView`, що підтримує роботу з тією ж формою, що і для створення, але вже для редагування існуючого екземпляру моделі, і з застосуванням моделі `PlantspeciesActivity` в основі.

Функція програми перегляду специфікацій процедур з догляду, визначених експертом, реалізована створенням класу `PlantspeciesAdviserActivitiesView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що визначає необхідність авторизації користувача, та `ListView`, що дозволяє керувати переліком, і з застосуванням моделі `PlantspeciesActivity` в основі. Відбираються у методі `get_queryset` усі процедури з догляду, визначені експертом з логіном, який передано через запит, або авторизованого експерта, якщо такі дані не передано.

Функція програми перегляду процедур з догляду за вказаним видом рослини реалізована створенням класу `PlantspeciesActivitiesView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що визначає необхідність авторизації користувача, та `ListView`, що дозволяє керувати переліком, і з застосуванням моделі `PlantspeciesActivity` в основі. Відбираються у методі `get_queryset` усі процедури з догляду, що стосуються виду рослини з назвою, який передано через запит у параметрі `plant`. Якщо значення параметра `all`

встановлено, то повертаються результати за всіма експертами, а якщо ні, то тільки за авторизованим експертом.

Функція програми перегляду запитаних порад реалізована створенням функції `adviser_ask_views`, у якій спочатку отримуються дані авторизованого експерта. Далі обираються його загальні запити, відділяючи окремо запити, на які було отримано вже пораду, – `asks_answered` та запити, на які ще пораду не отримано, – `asks_new`. Окремо відділяються його запити, які стосуються рослин садівника: ті, на які отримано пораду, – `plant_asks_answered` та ті, на які не отримано пораду, – `plant_asks_new`. Далі відділяються запити до експерта за проблемою. Окремо ті, на які було отримано пораду, – `asks_answered` та ті, на які пораду не було отримано, – `asks_new`.

Функція програми перегляду порад, наданих експертом, реалізована створенням функції `adviser_advices_views`, у якій за експертом з логіном, який передано у параметрі `adviser`, або за авторизованим експертом відбираються поради за проблемами, які надав цей експерт.

Функція програми надання поради про проблему реалізована створенням класу `AdviserAskPlantAnswerView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `UpdateView`, що підтримує роботу з формою для редагування існуючого екземпляру моделі, і з застосуванням моделі `AdviserAskPlant` в основі. Метод `get_queryset` виконує перевірку, залишаючи в контексті даних тільки ті, що стосуються авторизованого експерта. Тобто якщо запит надіслано не цьому експерту, то він не зможе внести відповідь.

Функція програми надання загальної поради реалізована створенням класу `AdviserAskAnswerView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `UpdateView`, що підтримує роботу з формою для редагування існуючого екземпляру моделі, і з застосуванням моделі `AdviserAskPlant` в основі. Метод `get_queryset` виконує перевірку, залишаючи

в контексті даних тільки ті, що стосуються авторизованого експерта. Тобто якщо запит надіслано не цьому експерту, то він не зможе внести відповідь.

Функція програми надання поради про рослину реалізована створенням класу `ProblemAdviceAnswerView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `CreateView`, що підтримує роботу з формою створення поради про рослину, і з застосуванням моделі `ProblemAdvice` в основі. Метод `dispatch` за проблемою вносить відповідний об'єкт з повними даними. Метод `get_context_data` передає цей об'єкт на сторінку. Метод `form_valid` заповнює поле `adviser` з даними експерта, який створив цю пораду, за допомогою даних авторизованого експерта.

3.2 Програмна реалізація функцій садівника

Частина функцій садівника використовується на основі функцій, реалізованих для експерта. Тому їх не було описано в цьому підрозділі окремо.

Функція програми перегляду керівництв з вирощування рослин експертів з підписки реалізована створенням класу `SubscribedTutorialView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що визначає необхідність авторизації користувача, та `ListView`, що дозволяє відтворювати дані за переліком керівництв, і з застосуванням моделі `Tutorial` в основі. Метод `get_queryset` відбирає всі керівництва, які створені експертами, на яких підписався авторизований садівник.

Функція програми збереження керівництва з вирощування рослин у власну теку реалізована створенням класу `TutorialSaveView` для представлення через наслідування класів `LoginRequiredMixin`, що визначає необхідність авторизації користувача, та базового класу представлення `View`. У методі `post` перевіряється, чи авторизований садівник вніс у модель

TutorialSave об'єкт за цим керівництвом, чи ні. Якщо так, то цей об'єкт вилучається. Якщо ні, то об'єкт додається. Таким чином одночасно реалізується функція програми вилучення керівництва з вирощування рослин з власної теки.

Функція програми перегляду збережених керівництв з власної теки реалізована створенням класу SavedTutorialView для представлення і шаблону вебсторінки для цього представлення через наслідування класів LoginRequiredMixin, що визначає необхідність авторизації користувача, та ListView, що дозволяє відтворювати дані за переліком керівництв, і з застосуванням моделі Tutorial в основі. Метод get_queryset відбирає всі керівництва, які мають для авторизованого садівника об'єкти моделі TutorialSave.

Функція програми запиту поради про власну рослину реалізована створенням класу AdvicerAskPlantView для представлення і шаблону вебсторінки для цього представлення через наслідування класів LoginRequiredMixin, що вимагає авторизації, та CreateView, що підтримує роботу з формою створення запиту про пораду, і з застосуванням моделі AdvicerAskPlant в основі.

Функція програми запиту загальної поради реалізована створенням класу AdvicerAskView для представлення і шаблону вебсторінки для цього представлення через наслідування класів LoginRequiredMixin, що вимагає авторизації, та CreateView, що підтримує роботу з формою створення запиту про пораду, і з застосуванням моделі AdvicerAsk в основі. Метод form_valid заповнює поле gardener з даними садівника, який створив цю пораду, за допомогою даних авторизованого садівника.

Функція програми запиту поради про проблему реалізована створенням класу AdvicerAskProblemView для представлення і шаблону вебсторінки для цього представлення через наслідування класів LoginRequiredMixin, що вимагає авторизації, та CreateView, що підтримує

роботу з формою створення запиту про пораду, і з застосуванням моделі `AdvicerAskProblem` в основі.

Функція програми оформлення підписки садівником на експерта реалізована створенням класу `AdvicerFollowingView` для представлення через наслідування класів `LoginRequiredMixin`, що визначає необхідність авторизації користувача, та базового класу представлення `View`. У методі `post` перевіряється, чи авторизований садівник вніс у модель `AdvicerFollowing` об'єкт за цим експертом, чи ні. Якщо так, то цей об'єкт вилучається. Якщо ні, то об'єкт додається. Таким чином одночасно реалізується функція програми скасування оформлення підписки садівником на експерта.

Функція програми внесення власної рослини реалізована створенням класу `PlantCreateView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `CreateView`, що підтримує роботу з формою створення запису про власну рослину, і з застосуванням моделі `Plant` в основі. Метод `form_valid` заповнює поле `gardener` з даними садівника, який створив цей запис, за допомогою даних авторизованого садівника.

Функція програми оцінювання експерта реалізована створенням класу `AdvicerAssessCreateView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `CreateView`, що підтримує роботу з формою створення оцінювання експерта, і з застосуванням моделі `AdvicerAssess` в основі. Метод `form_valid` заповнює поле `gardener` з даними садівника, який виконав оцінювання, за допомогою даних авторизованого садівника.

Функція програми визначення проблеми, виявленої у рослини, реалізована створенням класу `ProblemCreateView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `CreateView`, що підтримує

роботу з формою створення запису про проблему в рослині, і з застосуванням моделі Problem в основі.

Функція програми редагування даних власної рослини реалізована створенням класу PlantUpdateView для представлення і шаблону вебсторінки для цього представлення через наслідування класів LoginRequiredMixin, що вимагає авторизації, та UpdateView, що підтримує роботу з формою редагування запису про власну рослину, і з застосуванням моделі Plant в основі. За тією ж логікою організована функція програми видалення власної рослини реалізована створенням класу PlantRemoveView, адже це передбачає деактивацію даних.

Функція програми перегляду отриманих порад від експертів за рослиною реалізована створенням функції gardener_plant_advices_view, у якій перевіряється, чи рослина plant, що визначається параметром функції, є рослиною авторизованого користувача. Якщо це так, то за нею вибираються всі поради від експертів за проблемами, пов'язаними з цією рослиною, з моделі ProblemAdvice та поради за цією рослиною з моделі AdvicerAskPlant.

Функція програми перегляду переліку власних рослин реалізована створенням класу GardenerPlantView для представлення і шаблону вебсторінки для цього представлення через наслідування класів LoginRequiredMixin, що визначає необхідність авторизації користувача, та ListView, що дозволяє керувати переліком рослин, і з застосуванням моделі AdvicerAssess в основі. Відбираються у методі get_queryset усі рослини, якими володіє авторизований на даний момент садівник і які є активними, тобто мають позначку True у полі active.

Функція програми оформлення підписки садівником на програму реалізована створенням класу SubscribeView через наслідування класів LoginRequiredMixin, що вимагає авторизації, та View, що через метод post підтримує внесення параметрів сплати. Класи SuccessView і CancelView реалізовані через наслідування класу TemplateView і підтримують обробку результатів сплати: успішної або неуспішної відповідно.

Функція програми внесення виконаних дій з догляду за рослиною реалізована створенням класу `PlantActivityHistoryCreateView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `CreateView`, що підтримує роботу з формою створення запису про проблему в рослині, і з застосуванням моделі `PlantActivityHistory` в основі. У методі `form_valid` одночасно виконується внесення змін в об'єкт моделі `PlantActivity` з оновленням наступного часу виконання дії у відповідності з запланованою у ньому для поточного стану в приміщенні, де знаходиться ця рослина, періодичністю.

Функція програми перегляду історії доглядів за рослиною реалізована створенням класу `PlantActivityHistoriesView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що визначає необхідність авторизації користувача, та `ListView`, що дозволяє керувати переліком записів історії доглядів, і з застосуванням моделі `PlantActivityHistory` в основі. Відбираються у методі `get_queryset` за параметром з запиту `plant` записи зі всіма діями з догляду стосовно цієї рослини, перевіряючи при цьому, що авторизований садівник володіє цією рослиною.

Функція програми перегляду календаря догляду реалізована створенням класу `CalendarPlantActivitiesView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що визначає необхідність авторизації користувача, та `ListView`, що дозволяє керувати переліком запланованих дій для рослини, і з застосуванням моделі `PlantActivity` в основі. Відбираються у методі `get_queryset` за параметром з запиту `plant` записи зі всіма діями з догляду стосовно цієї рослини, впорядковуючи їх за датою виконання наступної дії і групуючи за відповідними датами.

Функція програми визначення періодичності догляду за рослиною реалізована створенням класу `PlantActivityCreateView` для представлення і

шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `CreateView`, що підтримує роботу з формою створення запису про періодичність догляду для рослини садівника, і з застосуванням моделі `PlantActivity` в основі. Метод `dispatch` за рослиною та дією з догляду вносить відповідні об'єкти. Метод `get_context_data` передає ці об'єкти на сторінку, а разом з ними інформацію про те, чи є авторизований садівник власником цієї тварини. Якщо він не є власником, то в шаблоні вебсторінки визначається, що форма такому користувачу не відображається.

Функція програми створення запису про приміщення реалізована створенням класу `GardenerLocationCreateView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `CreateView`, що підтримує роботу з формою створення запису про приміщення садівника, і з застосуванням моделі `GardenerLocation` в основі. У методі `form_valid` одночасно виконується внесення даних авторизованого садівника у відповідне поле.

Функція програми редагування даних приміщення реалізована створенням класу `GardenerLocationUpdateView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що вимагає авторизації, та `UpdateView`, що підтримує роботу з формою редагування даних приміщення власника, і з застосуванням моделі `GardenerLocation` в основі. Метод `get_queryset` виконує перевірку, залишаючи в контексті даних тільки ті, що стосуються авторизованого садівника. Подібно реалізовано функцію програми визначення поточних умов у приміщенні через клас `GardenerLocationConditionUpdateView`.

Функція програми перегляду власних приміщень реалізована створенням класу `GardenerLocationsView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що визначає необхідність авторизації користувача, та

ListView, що дозволяє відтворювати дані за переліком приміщень, і з застосуванням моделі GardenerLocation в основі. Метод `get_queryset` відбирає всі приміщення, які створені авторизованим експертом.

Функція програми вилучення приміщення реалізована створенням класу `GardenerLocationDeleteView` для представлення через наслідування класів `LoginRequiredMixin`, що визначає необхідність авторизації користувача, та `DeleteView`, що підтримує вилучення з застосуванням моделі `GardenerLocation`.

Функція програми перегляду останніх розв'язаних проблем експертами з підписки реалізована створенням класу `SubscribedProblemsView` для представлення і шаблону вебсторінки для цього представлення через наслідування класів `LoginRequiredMixin`, що визначає необхідність авторизації користувача, та `ListView`, що дозволяє відтворювати дані за переліком проблем, і з застосуванням моделі `ProblemAdvice` в основі. Метод `get_queryset` відбирає всі поради з проблем, які стосуються проблем за рослинами авторизованого садівника і які створені експертами, на які підписався садівник.

3.3 Висновки за розділом 3

Усі необхідні завдання з реалізації програми виконано: за допомогою патерну MVT на основі фреймворку Django реалізовано функції програми через відповідні класи і функції, що подано в розділі.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для використання її садівниками для пошуку нових рослин, для впорядкування догляду за власними рослинами, вирішення проблем, які пов'язані з захворюваннями рослин, знаходження інформації щодо правильного догляду за рослинами. Для експертів програма надає можливість ділитися своїм досвідом, знаннями, допомагати садівникам у тих проблемах, які у них виникають.

4.2 Умови виконання програми

Апаратні вимоги до серверної сторони:

- процесор у мінімальній конфігурації Intel Core 2 або відповідний еквівалент;
 - оперативна пам'ять з доступом не менше ніж до 4 Гб;
 - жорсткий диск з бажаним доступним обсягом не менше ніж 100 Гб:
- у випадку меншого обсягу потрібно контролювати обсяг даних у БД при внесенні фотографій, при цьому в будь-якому випадку потрібний обсяг може і значно збільшуватися, що залежить від кількості користувачів, кількості та якості фотографій, які вони завантажують.

З програмного забезпечення вимагається наявність встановленого інтерпретатора Python 3, системи PostgreSQL.

4.3 Виконання і тестування програми

Виконання програми, описане нижче, є результатом тестування програми, яке здійснювалось за наступною процедурою. Для кожної з функцій, що була визначена за результатами моделювання для кожного користувача, виконувалось тестування з перевіркою результатів у браузері

Google Chrome та Mozilla Firefox. У випадку виявлення невідповідностей за якоюсь функцією ці невідповідності усувались, а далі процедура виконувалась заново, поки в кінці не було перевірено всі заплановані функції з отриманням очікуваних результатів.

Робота користувача з програмою реалізується в основному після авторизації, оскільки до авторизації користувач може тільки шукати види рослин і керівництва, переглядати інформацію про види рослин, керівництва, експертів. Тобто це тільки загальна інформація. Після авторизації користувач визначається як садівник. Садівник має стандартне меню (рис. 4.1) та набір посилань на кожній сторінці для використання тих або інших функцій.

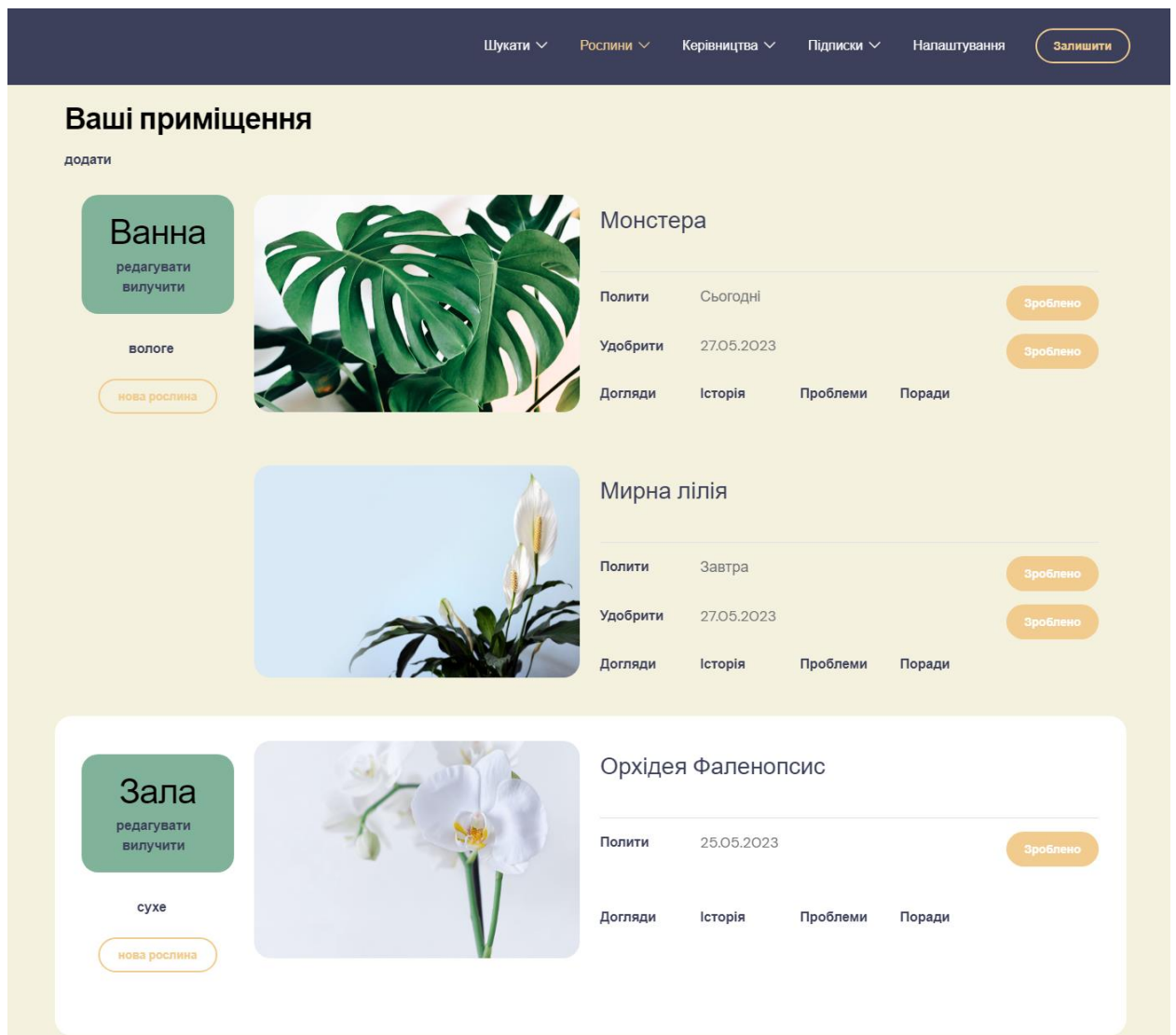


Рисунок 4.1 – Керування власними приміщеннями і рослинами

Садівник може шукати керівництва, експертів та види рослин. Вибрати конкретний варіант потрібно з випадającego переліку з пункту меню «Шукати». З результатів пошуку можна перейти до відповідної сторінки, а звідти, залежно від інформації, що наводиться, – на інші: наприклад, на сторінку експерта зі сторінки керівництва.

Меню «Рослини» містить підпункти «мої», «календар», «приміщення» та «бажані». За першим пунктом відтворюється список всіх рослин, якими володіє садівник, з можливістю редагування даних, внесення даних про проблему рослини, видалення рослини. Підпункт «календар» за допомогою вибору дати демонструє, які саме процедури з догляду мають бути виконані за кожною рослиною. Подібно виглядає пункт «приміщення», тільки структуризація відбувається за приміщеннями. Посилання «додати» на сторінці дозволяє додати нове приміщення садівнику. Зеленим відображається приміщення. За ним є можливість або редагувати його дані, або вилучити. Нижче відображається стан у приміщенні (вологе, помірне, сухе). При натисканні на нього можна обрати інший стан у спливаючому вікні. Тоді орієнтовні дати виконання процедури для кожної рослини у цьому приміщенні наступного разу будуть оновлені у відповідності з правилами, які було встановлено за рослиною у цього садівника. Тут же можна додати нову рослину у приміщення кнопкою «нова рослина».

За кожним приміщенням відображається список рослин садівника у цьому приміщенні. За кожною рослиною демонструється назва, очікувані процедури. За кожною процедурою відображається дата наступної дії. Для того щоб перейти до списку періодичності процедур у тому вигляді, якими їх визначив садівник для своєї рослини, потрібно натиснути на посилання «Догляди». Посилання «Історія» дозволяє переглянути всі минулі виконані дії з догляду за рослиною. Посилання «Проблеми» дозволяє переглянути проблеми за цією рослиною і внести нову, якщо це потрібно. Посилання «Поради» дозволяє переглянути поради від експертів за цією рослиною або перейти до порад загалом (окреме посилання).

Якщо садівник вже виконав заплановану дію, то він має натиснути на кнопку «Зроблено», одразу буде перераховано дату наступної дії та зафіксовано виконану.

Поради за проблемою міститься на сторінці проблеми (рис. 4.2). На сторінці проблеми відтворюються фотографії проблеми з можливістю перемикатися між ними, опис проблеми, дата, коли було зафіксовано садівником цю проблему, поточний стан вирішення проблеми, дата вирішення, якщо вже було вирішено. Кнопка «Редагувати» дозволяє внести зміни в ці та наступні дані, а також перейти до додавання фотографії, якщо це потрібно.

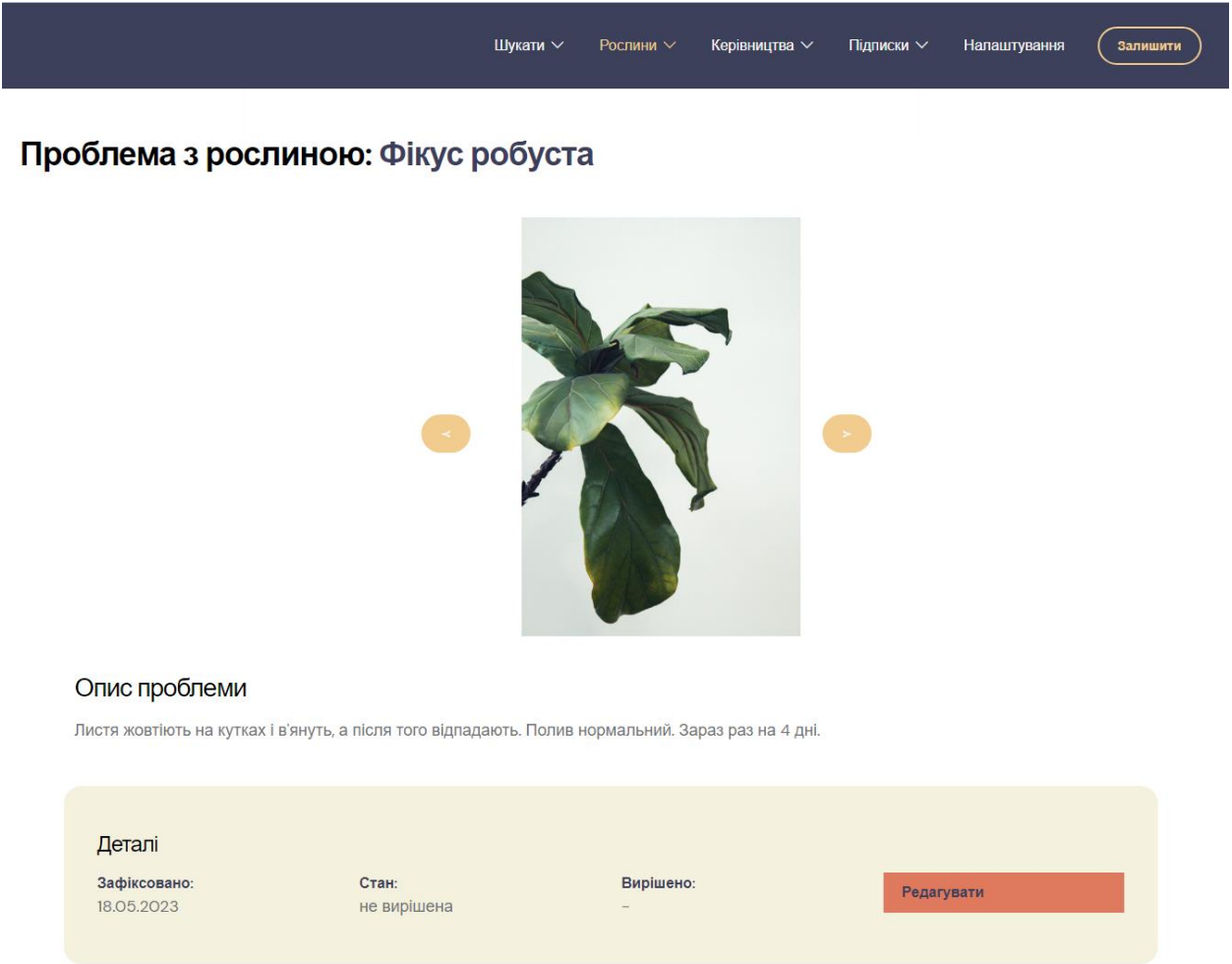


Рисунок 4.2 – Проблема за рослиною садівника

На тій же самій сторінці нижче знаходиться опис виконаного лікування та отриманого результату (рис. 4.3). Ще нижче знаходяться поради, отримані за експертом. За кожною порадою є посилання на ім'я експерта, щоб перейти до його сторінки, текст поради, дата створення та виявлена хвороба. При натисканні на назву хвороби можна перейти до виявлених випадків цієї хвороби. Так садівник може ствердитися в думці. Кнопка «оцінити» дозволяє перейти до оцінювання експерта. Ще нижче знаходяться запити до експертів садівника з проханням допомогти у вирішенні цієї проблеми.

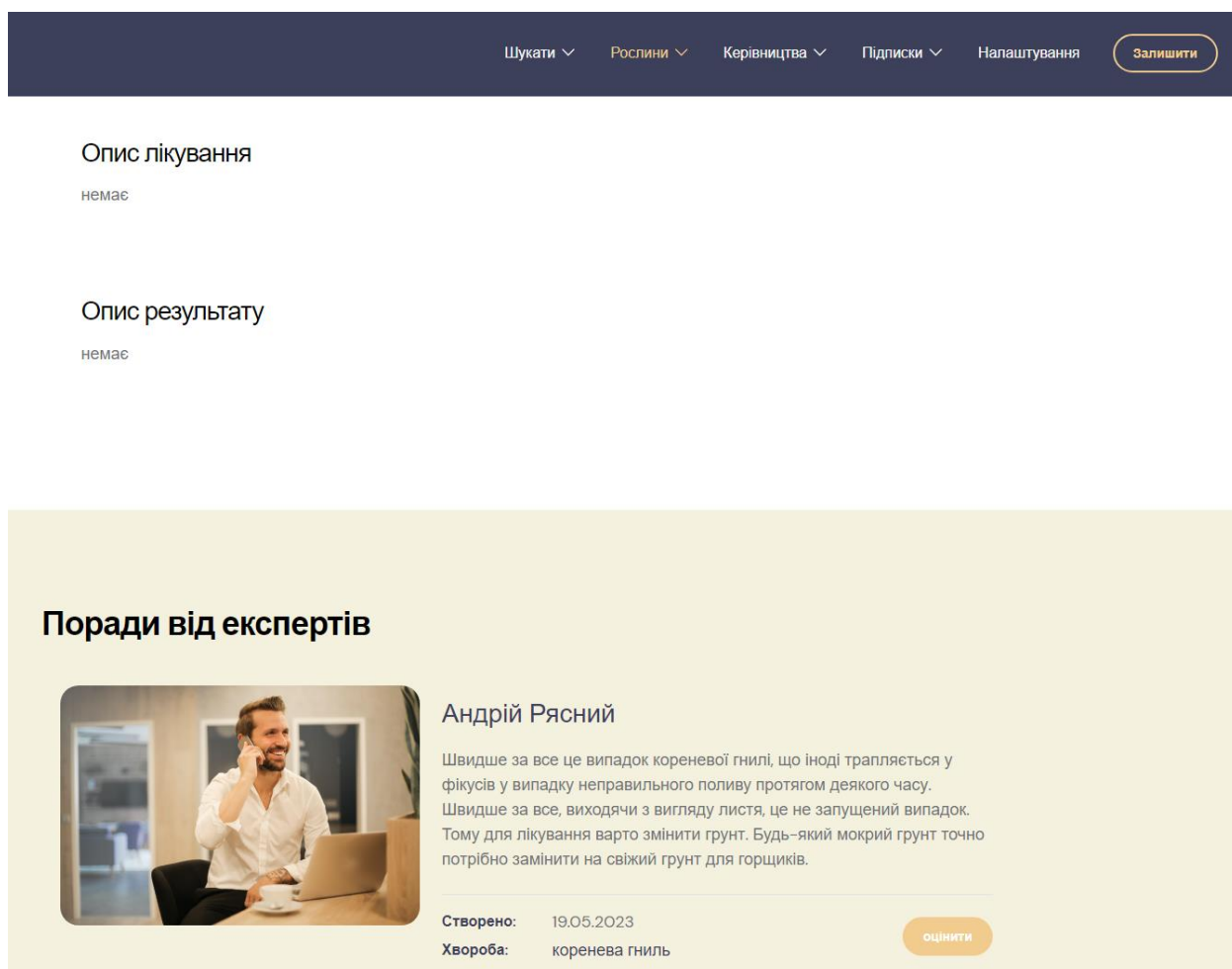


Рисунок 4.3 – Поради від експертів за виявленою проблемою рослини

Меню «Керівництва» містить підпункти «збережені» та «підписники». За кожною сторінкою керівництва є посилання для збереження керівництва, а

потім воно відтворюється в цій теці. Варіант «підписники» відображає керівництва тих експертів, на яких підписався садівник.

Меню «Підписки» містить підпункти «список» та «проблеми». Перший відтворює список експертів, на яких підписався садівник. Другий відтворює список проблем, поради за якими надавав експерт, на якого підписався садівник.

Експерт має дещо інше стандартне меню. Воно включає пункти «Шукати» для подібних до садівника дій з пошуку. Меню «Керівництва» дозволяє керувати власними керівництвами. Меню «Процедури» дозволяє визначати експерту власні специфікації процедур з догляду (рис. 4.4).

Шукати ▾ Керівництва ▾ **Процедури** Поради Хвороби Оцінювання Напаштування [Залишити](#)

Ваші специфікації процедур з догляду

Періодичність

додати процедуру

Догляд і умови	Період	Сухе	Помірне	Вологе
Орхідея Фаленопсис	1.12 – 28.02	5 ✖	6 ✔	7 ✖
Орхідея Фаленопсис	1.03 – 30.11	3 ✖	4 ✖	4 ✖
Фікус ліроподібний лірата	1.12 – 28.02	7 ✖	7 ✖	10 ✖
Фікус ліроподібний лірата	1.03 – 30.11	3 ✖	3 ✖	5 ✖

Оберіть параметри

Процедура
полити

Вид рослини
всі

[Відобразити](#)

Запис обраного догляду

Раз на місяць краще проводити полив методом гарячого душу.

[Редагувати](#)

Рисунок 4.4 – Керування власними специфікаціями процедур з догляду експертом

Експерт за наявними процедурами повинен обрати тип процедури, вид рослини або вказати всі), а тоді натиснути «Відобразити». У таблиці

відображається список процедур з назвою, періодом дії та періодичністю для кожного варіанта умов у приміщенні.

Якщо натиснути на червону іконку з хрестиком на відповідному варіанті, то відображається запис з порадою щодо цього варіанта дії. Кнопка «Редагувати» дозволяє внести зміни.

Кнопка «додати процедуру» згори дозволяє визначити нову специфікацію.

Той же самий варіант сторінки для садівника (рис. 4.5) містить можливість запитати експерта про рослину, яка належить садівнику та яка відповідає тому виду рослин, який обрано зараз з таблиці.

Шукати
Рослини
Керівництва
Підписки
Налаштування
Залишити

Специфікації процедур з догляду експерта

запитати
запитати про проблему

Експерт: Дмитро Лісний

відписатися

оцінити

Періодичність

Догляд і умови	Період	Сухе	Помірне	Вологе
Орхідея Фаленопсис	1.12 – 28.02	5	6	7
Орхідея Фаленопсис	1.03 – 30.11	3	4	4
Фікус ліроподібний лірата	1.12 – 28.02	7	7	10
Фікус ліроподібний лірата	1.03 – 30.11	3	3	5

Оберіть параметри

Процедура

полити

Вид рослини

всі

Відобразити

Запис обраного догляду

Поливати тільки тоді, коли верхній шар (2 дюйми) повністю сухий.

запитати про рослину

Рисунок 4.5 – Перегляд специфікацій експерта процедур з догляду садівником

Редагувати відповідно садівник дані не може, а на сторінці йому відображаються дані експерта, включаючи його ім'я, можливість підписатися

або відписатися від нього та перейти до його оцінювання. Також доступні посилання для створення загального запиту до цього експерта та запиту про проблему.

Експерт з меню «Поради» отримує доступ до всіх запитів до нього, як загальних, так і за рослиною або проблемою. Звідти він може формувати відповідні на запити. З меню «Хвороби» експерт може вносити описи випадків виявлення тих чи інших хвороб, які були здійснені ним. З меню «Оцінювання» експерт може переглянути, як особисто його було оцінено садівниками.

Деякі функції доступні у спеціальних режимах роботи на окремих сторінках. Наприклад, при пошуку видів рослин з результатів пошуку, а також зі сторінок цих видів рослин доступне посилання для збереження їх до власного списку. Експерту, наприклад, на сторінці проблеми доступна кнопка для створення власної поради за цією проблемою за своєю власною ініціативою.

4.4 Висновки за розділом 4

Основні сценарії роботи садівника та експерта в чіткій відповідності з виконаним раніше описом реалізації презентовані. Перед тим забезпечено тестування функцій в двох браузерях з успішним виконанням.

ВИСНОВКИ

Аналіз існуючих програмних засобів допомоги у вирощуванні рослин дозволив виявити основні функції таких програми, а також наявні прогалини. Це дозволило в підсумку сформулювати концепцію програми-розробки. Програма організована навколо ідеї об'єднання засобів пошуку інформації, необхідної при вирощуванні рослин, плануванні відповідних процедур та вирішенні проблем при виявленні захворювань рослини на основі контактування з експертами. Введення двох сторін у додаток (садівників та експертів) має сприяти розвитку ринку послуг людей, які цікавляться вирощуванням рослин, як домашніх, так і в садах.

Усі необхідні завдання з проєктування програми в роботі виконано: визначено основну мову для розробки, якою є Python, фреймворк, який підтримує веброзробку, яким є Django, сформульовано сценарії роботи садівників і експертів з програмою, визначено структуру БД, включаючи потрібні сутності, представлення цих сутностей в реляційній БД. Визначення структури БД сформувало фактично основу реалізованих класів моделей у структурі результуючого MVT-додатку. Усі необхідні завдання з реалізації програми виконано: за допомогою патерну MVT на основі фреймворку Django реалізовано функції програми через відповідні класи і функції, що подано в розділі. Основні сценарії роботи садівника та експерта в чіткій відповідності з виконаним раніше описом реалізації презентовані. Перед тим забезпечено тестування функцій в браузерях Chrome і Firefox з успішним виконанням.

Створена програма призначена для використання її садівниками для пошуку нових рослин, для впорядкування догляду за власними рослинами, вирішення проблем, які пов'язані з захворюваннями рослин, знаходження інформації щодо правильного догляду за рослинами. Для експертів програма надає можливість ділитися своїм досвідом, знаннями, допомагати садівникам у тих проблемах, які у них виникають.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Plant Troubleshooting [Electronic resource] : How to Diagnose Common Plant Care Problems. – Access mode : <https://trendspotinc.com/plant-troubleshooting-how-to-diagnose-common-plant-care-problems/>.
2. Plant care app – Keep your plants alive. Planta [Electronic resource]. – Access mode : <https://getplanta.com/>.
3. Best Free Plant Care App in 2023 [Electronic resource]. – Access mode : <https://blog.gitnux.com/best-free-plant-care-app/>.
4. Vera Plant Care App. Bloomscape [Electronic resource]. – Access mode : <https://bloomscape.com/vera/#download>.
5. ThePlantMe [Electronic resource]. – Access mode : <https://theplantme.com/>.
6. Chu, H. 7 apps to keep your plants alive and well [Electronic resource] / H. Chu. – Access mode : <https://mashable.com/article/best-apps-for-taking-care-of-plants>.
7. Gardenia [Electronic resource]. – Access mode : <http://getgardenia.co/>.
8. Downey, A. Think Python [Text] : How to Think Like a Computer Scientist / A. Downey. – Sebastopol, California : O'Reilly Media, 2016. – 289 p.
9. Flask vs. Django [Electronic resource] : Which Python Framework Is Better for Your Web Development? – Access mode : <https://www.stxnnext.com/blog/flask-vs-django-comparison/>.
10. Flask vs. Django – a comparison of the Python frameworks [Electronic resource]. – Access mode : <https://www.ionos.com/digitalguide/websites/web-development/flask-vs-django/>.
11. Tiya Golf Club Template [Electronic resource]. – Access mode : <https://templatemo.com/tm-587-tiya-golf-club>.
12. Kean, K. Django's MVT Architecture Explained [Electronic resource] / K. Kean. – Access mode : <https://www.makeuseof.com/django-mvt-architecture/>.

ДОДАТОК А
Технічне завдання

Вступ

Вирощування рослин – це непростий процес. Він починається ще під час вибору рослин, продовжується, коли людина саджає або купляє готову рослину, але на цьому не зупиняється, а тільки розпочинається. Далі потрібно весь час доглядати рослини: поливати, вносити добрива, обрізати певним чином і т.д. Але окрім цієї монотонності, про яку потрібно не забути у відповідний момент, рослини хворіють. Потрібно розпізнати хворобу, зрозуміти, як її вилікувати. Тобто це вже величезний набір дій. Але він ще більше збільшується, чим більше рослин має людина у себе вдома. Тоді для підтримки даних процесів, для їхнього впорядкування потрібно надати зручну множину інструментів у складі програмного забезпечення, яке має бути розроблено в роботі.

A.1 Підстави для розробки

Завдання на дипломну кваліфікаційну роботу визначено на підставі наказу № 87 від 4 квітня 2023 р. за Національним університетом «Запорізька політехніка» за темою «Програмне забезпечення допомоги у вирощуванні рослин».

A.2 Призначення розробки

Програма призначена для використання її садівниками для пошуку нових рослин, для впорядкування догляду за власними рослинами, вирішення проблем, які пов'язані з захворюваннями рослин, знаходження інформації щодо правильного догляду за рослинами. Для експертів програма надає можливість ділитися своїм досвідом, знаннями, допомагати садівникам у тих проблемах, які у них виникають.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Для підтримки описаного призначення програма повинна надавати користувачам наступний набір функцій для роботи:

- створення і редагування керівництва з вирощування рослин експертом;
- перегляд списку вже внесених керівництв з вирощування рослин експертом;
- перегляд керівництва з вирощування рослини;
- перегляд керівництв з вирощування за видом рослини;
- збереження керівництва з вирощування рослин у власну теку;
- вилучення керівництва з вирощування рослин з власної теки;
- перегляд збережених керівництв з власної теки;
- пошук керівництв з вирощування рослин;
- перегляд керівництв з вирощування рослин експертів, на яких підписався садівник;
- визначення процедур з догляду за видами рослин експертом, включаючи створення власної специфікації, редагування внесених даних;
- перегляд процедур з догляду за вказаним видом рослини;
- перегляд специфікацій процедур з догляду, визначених експертом;
- запит поради садівником у експерта;
- перегляд експертом запитаних у нього порад;
- перегляд отриманих порад від експертів садівником за рослиною;
- перегляд порад, наданих експертом;
- надання поради з вирощування рослин експертом садівнику;
- оформлення підписки садівником на програму;
- відписування садівника від експерта;
- оформлення підписки садівником на експерта;

– внесення власних рослин садівником, редагування даних про них і вилучення;

- перегляд переліку рослин садівника;
- визначення періодичності догляду за рослиною;
- перегляд оцінювань за експертом;
- оцінювання експерта;
- визначення проблеми, виявленої у рослині;
- перегляд випадків лікування за хворобою;
- внесення виконаних дій з догляду за рослиною;
- перегляд історії доглядів за рослиною;
- пошук видів рослин;
- пошук експертів;
- визначення садівником власних приміщень, включаючи створення,

редагування і вилучення;

- перегляд власних приміщень;
- визначення поточних умов у приміщенні;
- попередження про несумісність виду рослин з рослиною садівника;
- авторизація користувача;
- реєстрація садівника та експерта;
- перегляд останніх розв'язаних експертами, на які підписався

садівник, проблем;

- перегляд розв'язань проблем за експертом.

A.3.2 Вимоги до складу та параметрів технічних засобів

Апаратні вимоги до серверної сторони:

- процесор у мінімальній конфігурації Intel Core 2 або відповідний еквівалент;
- оперативна пам'ять з доступом не менше ніж до 4 Гб;

– жорсткий диск з бажаним доступним обсягом не менше ніж 100 Гб: у випадку меншого обсягу потрібно контролювати обсяг даних у БД при внесенні фотографій, при цьому в будь-якому випадку потрібний обсяг може і значно збільшуватися, що залежить від кількості користувачів, кількості та якості фотографій, які вони завантажують.

A.4 Порядок контролю та приймання

Основою щодо реалізації контролю та приймання дипломної кваліфікаційної роботи є завдання на роботу і безпосередньо дане технічне завдання. За успішного виконання цих завдань у встановлені строки робота після проходження всіх підготовчих процедур вноситься на захист перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

Б.1 Текст файла `advicerviews.py`

```

from django.views.generic.edit import CreateView
from .models import AdvicerAssess
from .models import Advicer
from .models import Plantspecies
from django.views.generic.edit import UpdateView
from .models import Plantcategory
from .models import PlantspeciesHealthbenefit
from .models import Healthbenefit
from .models import AdvicerAsk
from .models import AdvicerAskPlant
from django.views.generic.list import ListView
from .models import ProblemAdvice
from .models import Tutorial
from .models import TutorialSave
from .models import Activity
from .models import PlantspeciesDisease
from django.contrib.auth.mixins import LoginRequiredMixin
from .models import Disease
from .models import DiseaseCase
from .models import Problem
from django.views.generic.detail import DetailView
from django.db.models import Q

class AdvicerAssessView(LoginRequiredMixin, ListView):
    template_name = 'assesses.html'
    model = AdvicerAssess
    paginate_by = 10

    def get_queryset(self):
        advicer_login = self.request.GET.get('advicer')
        if advicer_login:
            advicer = Advicer.objects.get(user__login =
advicer_login)
        else:
            advicer = Advicer.objects.get(user =
self.request.user)
        assesses = AdvicerAssess.objects.filter(advicer =
advicer)
        assesses = assesses.order_by('-done_date')
        return assesses

    def get_context_data(self, **kwargs):

```

```

        data_all_context = super(AdvicerAssessView,
self).get_context_data(**kwargs)
        advicer_login = self.request.GET.get('advicer')
        if advicer_login:
            advicer = Advicer.objects.get(user__login =
advicer_login)
            data_all_context['advicer'] = advicer
        return data_all_context

class PlantspeciesSearchView(ListView):
    template_name = 'plant_search.html'
    model = Plantspecies
    paginate_by = 10

    def get_queryset(self):
        plant_category = self.request.GET.get('category')
        categories = plant_category.split(',')
        plant_categories = []
        for plant_category in categories:
            category =
Plantcategory.objects.filter(category_name = plant_category)
            plant_categories.append(category)
        if plant_categories:
            plantspecies_list =
Plantspecies.objects.filter(plant_category__in =
plant_categories)
            species_name = self.request.GET.get('name')
            if species_name and plantspecies_list:
                plantspecies_list =
plantspecies_list.filter(species_name__icontains = species_name)
            elif species_name:
                plantspecies_list =
Plantspecies.objects.filter(species_name__icontains =
species_name)
            leaf_color = self.request.GET.get('leaf')
            leaf_colors = leaf_color.split(',')
            plant_leaf_colors = []
            for leaf in leaf_colors:
                plant_leaf_colors.append(int(leaf))
            if plant_leaf_colors and plantspecies_list:
                plantspecies_list =
plantspecies_list.filter(leaf_color__in = plant_leaf_colors)
            else:
                plantspecies_list =
Plantspecies.objects.filter(leaf_color__in = plant_leaf_colors)
            flower_color = self.request.GET.get('flower')

```

```

        flower_colors = flower_color.split(',')
        plant_flower_colors = []
        for flower in flower_colors:
            plant_flower_colors.append(int(flower))
        plantspecies_list.filter(flower_color__in =
plant_flower_colors)
        growth_rate = self.request.GET.get('growth')
        growth_rates = growth_rate.split(',')
        plant_growth_rates = []
        for growth in growth_rates:
            plant_growth_rates.append(int(growth))
        plantspecies_list.filter(growth_rate__in =
plant_growth_rates)
        life_duration = self.request.GET.get('duration')
        life_durations = life_duration.split(',')
        plant_life_durations = []
        for life in life_durations:
            plant_life_durations.append(int(life))
        plantspecies_list.filter(duration__in =
plant_life_durations)
        water_requirements = self.request.GET.get('water')
        water_requirements = water_requirements.split(',')
        plant_water_requirements = []
        for water in water_requirements:
            plant_water_requirements.append(int(water))
        plantspecies_list.filter(water_requirements__in =
plant_water_requirements)
        fertilization_requirements =
self.request.GET.get('fertil')
        fertilization_requirements =
fertilization_requirements.split(',')
        plant_fertilization_requirements = []
        for fertilization in fertilization_requirements:

plant_fertilization_requirements.append(int(fertilization))

        plantspecies_list.filter(fertilization_requirements__in =
plant_fertilization_requirements)
        soil_requirements = self.request.GET.get('soil')
        soil_requirements = soil_requirements.split(',')
        plant_soil_requirements = []
        for soil in soil_requirements:
            plant_soil_requirements.append(int(soil))
        plantspecies_list.filter(soil_requirements__in =
plant_soil_requirements)
        light_requirements = self.request.GET.get('light')

```

```

        light_requirements = light_requirements.split(',')
        plant_light_requirements = []
        for light in light_requirements:
            plant_light_requirements.append(int(light))
        plantspecies_list.filter(light_requirements__in =
plant_light_requirements)
        plant_type = self.request.GET.get('type')
        plant_type = plant_type.split(',')
        plant_types = []
        for type in plant_type:
            plant_types.append(int(type))
        plantspecies_list.filter(plant_type__in =
plant_types)
        temperature = self.request.GET.get('temperature')
        plantspecies_list.filter(temperature__gte =
int(temperature))
        blooming = self.request.GET.get('bloom')
        blooming = blooming.split(',')
        plant_bloomings = []
        for bloom in blooming:
            plant_bloomings.append(int(bloom))
        plantspecies_list.filter(blooming__in =
plant_bloomings)
        humidity = self.request.GET.get('humid')
        humidity = humidity.split(',')
        plant_humidity = []
        for humid in humidity:
            plant_humidity.append(int(humid))
        plantspecies_list.filter(humidity__in =
plant_humidity)
        potting = self.request.GET.get('pot')
        potting = potting.split(',')
        plant_potting = []
        for pot in potting:
            plant_potting.append(int(pot))
        plantspecies_list.filter(potting__in =
plant_potting)
        propagation = self.request.GET.get('propagation')
        propagation = propagation.split(',')
        plant_propagation = []
        for propag in propagation:
            plant_propagation.append(int(propag))
        plantspecies_list.filter(propagation__in =
plant_propagation)
        benefit = self.request.GET.get('benefit')
        level = self.request.GET.get('level')

```

```

        benefit = benefit.split(',')
        levels = level.split(',')
        plant_benefit = []
        plant_levels = []
        num = 0
        condition = True
        for ben in benefit:
            plant_benefit =
Healthbenefit.objects.get(benefit_name = ben, None)
            if plant_benefit:
                level = int(levels[num])
                objects =
PlantspeciesHealthbenefit.objects.filter(healthbenefit =
plant_benefit, level__gte = level)
                condition = condition | Q(healthbenefits =
objects)

                num = num + 1
            plantspecies_list.filter(condition)
            plantspecies_list.order_by('species_name')
            return plantspecies_list

def get_context_data(self, **kwargs):
    data_all_context = super(PlantspeciesSearchView,
self).get_context_data(**kwargs)
    plant_category = self.request.GET.get('category')
    data_all_context['category'] = plant_category
    species_name = self.request.GET.get('name')
    data_all_context['species_name'] = species_name
    leaf_color = self.request.GET.get('leaf')
    data_all_context['leaf_color'] = leaf_color
    flower_color = self.request.GET.get('flower')
    data_all_context['flower_color'] = flower_color
    growth_rate = self.request.GET.get('growth')
    data_all_context['growth'] = growth_rate
    life_duration = self.request.GET.get('duration')
    data_all_context['duration'] = life_duration
    water_requirements = self.request.GET.get('water')
    data_all_context['water'] = water_requirements
    fertilization_requirements =
self.request.GET.get('fertil')
    data_all_context['fertilization'] =
fertilization_requirements
    soil_requirements = self.request.GET.get('soil')
    data_all_context['soil'] = soil_requirements
    light_requirements = self.request.GET.get('light')
    data_all_context['light'] = light_requirements

```

```

        plant_type = self.request.GET.get('type')
        data_all_context['plant_type'] = plant_type
        temperature = self.request.GET.get('temperature')
        data_all_context['temperature'] = temperature
        blooming = self.request.GET.get('bloom')
        data_all_context['blooming'] = blooming
        humidity = self.request.GET.get('humid')
        data_all_context['humidity'] = humidity
        potting = self.request.GET.get('pot')
        data_all_context['potting'] = potting
        propagation = self.request.GET.get('propagation')
        data_all_context['propagation'] = propagation
        benefit = self.request.GET.get('benefit')
        data_all_context['benefit'] = benefit
        level = self.request.GET.get('level')
        data_all_context['level'] = level
        return data_all_context

    def advicer_solutions_views(request, advicer = None):
        if not advicer:
            advicer = Advicer.objects.get(user =
self.request.user)
        else:
            advicer = Advicer.objects.get(user__login =
advicer)
        asks = AdvicerAsk.objects.filter(advicer = advicer,
answer__isnull = False).select_related("advicer").order_by("-
ask_date")
        plant_asks = AdvicerAskPlant.objects.filter(advicer =
advicer, answer__isnull =
False).select_related("advicer").select_related("plant").order_b
y("-ask_date")
        solution_data = {
            "advicer": advicer,
            "asks": asks,
            "plant_asks": plant_asks
        }
        return render(request, "plantapp/solutions.html",
solution_data)

class TutorialDetailView(DetailView):
    model = Tutorial

    def get_context_data(self, **kwargs):
        tutorial_context =
super().get_context_data(**kwargs)

```

```

        advicer = Advicer.objects.get(user =
self.request.user)
        tutorial_context["advicer_access"] = advicer.level
> 4.6
        return tutorial_context

class TutorialPlantspeciesView(ListView):
    template_name = 'plant_tutorials.html'
    model = Tutorial
    paginate_by = 10

    def get_queryset(self):
        plant_species = self.request.GET.get('plant')
        if plant_species:
            plant_species =
Plantspecies.objects.get(species_name = plant_species)
        else:
            return []
        tutorials = Tutorial.objects.filter(plantspecieses
= plant_species)
        tutorials = tutorials.order_by('-done_date')
        return tutorials

    def get_context_data(self, **kwargs):
        data_all_context = super(TutorialPlantspeciesView,
self).get_context_data(**kwargs)
        plant_species = self.request.GET.get('plant')
        if plant_species:
            plant_species =
Plantspecies.objects.get(species_name = plant_species)
            data_all_context['plant'] = plant_species
        return data_all_context

class TutorialAdvicerView(ListView):
    template_name = 'advicer_tutorials.html'
    model = Tutorial
    paginate_by = 10

    def get_queryset(self):
        advicer_login = self.request.GET.get('advicer')
        if advicer_login:
            advicer = Advicer.objects.get(user__login =
advicer_login)
        else:
            advicer = Advicer.objects.get(user =
self.request.user)

```

```

        tutorials = Tutorial.objects.filter(advicer =
advicer)

        tutorials = tutorials.order_by('-done_date')
        return tutorials

    def get_context_data(self, **kwargs):
        data_all_context = super(AdvicerAssessView,
self).get_context_data(**kwargs)
        advicer_login = self.request.GET.get('advicer')
        if advicer_login:
            advicer = Advicer.objects.get(user__login =
advicer_login)
            data_all_context['advicer'] = advicer
        return data_all_context

class TutorialSearchView(ListView):
    template_name = 'search_tutorials.html'
    model = Tutorial
    paginate_by = 10

    def get_queryset(self):
        tutorial_title = self.request.GET.get('title')
        if tutorial_title:
            tutorials =
Tutorial.objects.get(Q(title__icontains = tutorial_title) |
Q(all_text__icontains = tutorial_title))
            tutorial_plants = self.request.GET.get('plants')
            tutorial_activities =
self.request.GET.get('activities')
            advicer = Advicer.objects.get(user =
self.request.user)
            if not advicer:
                return []
            tutorial_plants = tutorial_plants.split(',')
            plant_species = []
            for plant in tutorial_plants:
                plant = Plantspecies.objects.get(species_name =
plant)

                plant_species.append(plant)
            if plant_species:
                tutorials =
Tutorial.objects.filter(plantspecieses__in = plant_species)
                tutorial_activities =
tutorial_activities.split(',')
                activities = []
                for act in tutorial_activities:

```

```

        act = Activity.objects.get(activity_name = act)
        activities.append(act)
    if activities:
        tutorials =
Tutorial.objects.filter(activities__in = activities)
    if advicer.level <= 4.6:
        tutorials = tutorials.filter(open = True)
    tutorials = tutorials.order_by('-done_date')
    return tutorials

    def get_context_data(self, **kwargs):
        data_all_context = super(TutorialSearchView,
self).get_context_data(**kwargs)
        tutorial_title = self.request.GET.get('title')
        data_all_context['title'] = tutorial_title
        tutorial_plants = self.request.GET.get('plants')
        data_all_context['plants'] = tutorial_plants
        tutorial_activities =
self.request.GET.get('activities')
        data_all_context['activities'] =
tutorial_activities
        return data_all_context

class AdvicerSearchView(ListView):
    template_name = 'search_advicer.html'
    model = Advicer
    paginate_by = 10

    def get_queryset(self):
        advicer_login = self.request.GET.get('login')
        if advicer_login:
            advicers =
Advicer.objects.filter(user__login__icontains = advicer_login)
            advicer_name = self.request.GET.get('name')
            if advicer_name:
                advicer_name = advicer_name.split(' ')
                name1 = advicer_name[0]
                name2 = advicer_name[1]
                if not advicers:
                    advicers =
Advicer.objects.filter(Q(last_name__icontains = name2,
first_name__icontains = name1) | Q(last_name__icontains = name1,
first_name__icontains = name2))
                else:
                    advicers =
advicers.filter(Q(last_name__icontains = name2,

```

```

first_name__icontains = name1) | Q(last_name__icontains = name1,
first_name__icontains = name2))
    advicer_email = self.request.GET.get('email')
    if advicer_email and advicers:
        advicers = advicers.filter(email =
advicer_email)
    elif advicer_email:
        advicers = Advicer.objects.filter(email =
advicer_email)
    advicer_level_min =
self.request.GET.get('level_min')
    advicer_level_max =
self.request.GET.get('level_max')
    if advicer_level_min:
        advicer_level_min = float(advicer_level_min)
        if advicers:
            advicers = advicers.filter(level__gte =
advicer_level_min)
        else:
            advicers =
Advicer.objects.filter(level__gte = advicer_level_min)
    if advicer_level_max:
        advicer_level_max = float(advicer_level_max)
        if advicers:
            advicers = advicers.filter(level__lte =
advicer_level_max)
        else:
            advicers =
Advicer.objects.filter(level__lte = advicer_level_max)
    advicer_followers =
self.request.GET.get('followers')
    advicers = advicers.order_by('-level')
    if advicer_followers:
        advicer_followers = int(advicer_followers)
        advicers_ready = []
        for advicer in advicers:
            followers =
AdvicerFollowing.objects.filter(advicer = advicer).count()
            if followers >= advicer_followers:
                advicer.followers = followers
                advicers_ready.append(advicer)
    return advicers_ready

    def get_context_data(self, **kwargs):
        data_all_context = super(TutorialSearchView,
self).get_context_data(**kwargs)

```

```

        advicer_login = self.request.GET.get('login')
        data_all_context['login'] = advicer_login
        advicer_name = self.request.GET.get('name')
        data_all_context['name'] = advicer_name
        advicer_email = self.request.GET.get('email')
        data_all_context['email'] = advicer_email
        advicer_level_min =
self.request.GET.get('level_min')
        data_all_context['level_min'] = advicer_level_min
        advicer_level_max =
self.request.GET.get('level_max')
        data_all_context['level_max'] = advicer_level_max
        advicer_followers =
self.request.GET.get('followers')
        data_all_context['followers'] = advicer_followers
        return data_all_context

    def plant_diseases_views(request, disease_name):
        if plant_species:
            disease = Disease.objects.get(disease_name =
disease_name)
        else:
            return render(request, "plantapp/advices.html",
{"disease": "no information", "cases": "no information"})
            plants_disease =
PlantspeciesDisease.objects.filter(disease =
disease).select_related("plant_species")
            cases =
DiseaseCase.objects.filter(plantspecies_disease__in =
plants_disease).select_related("advicer").order_by('-id')
            advice_cases = ProblemAdvice.objects.filter(disease =
disease, problem__stage =
2).select_related("problem").select_related("advicer").order_by(
'-done_date')
            cases_data = {
                "disease": disease,
                "cases": cases,
                "advice_cases": advice_cases
            }
            return render(request, "plantapp/disease.html",
cases_data)

class TutorialCreateView(LoginRequiredMixin, CreateView):
    model = Tutorial
    fields = ['title',
              'all_text',

```

```

        'pic',
        'open']
    success_url = reverse_lazy('plants:advicer_tutorials')

    def form_valid(self, form):
        advicer = Advicer.objects.get(user =
self.request.user)
        form.instance.advicer = advicer
        return super().form_valid(form)

class TutorialUpdateView(LoginRequiredMixin, UpdateView):
    model = Tutorial
    fields = ['title',
              'all_text',
              'pic',
              'open']
    success_url = reverse_lazy('plants:advicer_tutorials')

class PlantspeciesActivityCreateView(LoginRequiredMixin,
CreateView):
    model = PlantspeciesActivity
    fields = ['plant_species',
              'activity',
              'periodicity',
              'conditions',
              'start_period',
              'end_period',
              'memo']
    success_url =
reverse_lazy('plants:plantspec_activities')

    def dispatch(self, *args, **kwargs):
        self.plant_species =
Plantspecies.objects.get(species_name = kwargs['plant'], None)
        self.activity = Activity.objects.get(activity_name
= kwargs['activity'], None)
        return super(PlantspeciesActivityCreateView,
self).dispatch(*args, **kwargs)

    def get_context_data(self, **kwargs):
        plant_activity_context =
super().get_context_data(**kwargs)
        plant_activity_context["plant"] =
self.plant_species
        plant_activity_context["activity"] = self.activity

```

```

        plant_activity_context["standard"] =
PlantspeciesActivity.objects.filter(plant_species =
self.plant_species, activity = self.activity, advicer__id = 0)
        return plant_activity_context

    def form_valid(self, form):
        advicer = Advicer.objects.get(user =
self.request.user)
        form.instance.advicer = advicer
        return super().form_valid(form)

class PlantspeciesActivityUpdateView(LoginRequiredMixin,
UpdateView):
    model = PlantspeciesActivity
    fields = ['periodicity',
              'conditions',
              'start_period',
              'end_period',
              'memo']
    success_url =
reverse_lazy('plants:plantspec_activities')

class PlantspeciesAdvicerActivitiesView(LoginRequiredMixin,
ListView):
    template_name = 'advicer_activities.html'
    model = PlantspeciesActivity
    paginate_by = 10

    def get_queryset(self):
        advicer_login = self.request.GET.get('advicer')
        if advicer_login:
            advicer = Advicer.objects.get(user__login =
advicer_login)
        else:
            advicer = Advicer.objects.get(user =
self.request.user)
        activities =
PlantspeciesActivity.objects.filter(advicer =
advicer).select_related("plant_species").select_related("activit
y")
        activities = activities.order_by('-id')
        return activities

    def get_context_data(self, **kwargs):

```

```

        data_all_context =
super(PlantspeciesAdvicerActivitiesView,
self).get_context_data(**kwargs)
        advicer_login = self.request.GET.get('advicer')
        if advicer_login:
            advicer = Advicer.objects.get(user__login =
advicer_login)
            data_all_context['advicer'] = advicer
        return data_all_context

class PlantspeciesActivitiesView(LoginRequiredMixin,
ListView):
    template_name = 'plant_activities.html'
    model = PlantspeciesActivity
    paginate_by = 10

    def get_queryset(self):
        plant = self.request.GET.get('plant')
        all_mode = self.request.GET.get('all')
        activities = []
        if not all_mode:
            advicer = Advicer.objects.get(user =
self.request.user)
            activities =
PlantspeciesActivity.objects.filter(advicer =
advicer).select_related("activity").select_related("advicer")
            if plant:
                plant = Plantspecies.objects.get(species_name =
plant)
                if not activities:
                    activities =
PlantspeciesActivity.objects.filter(plant_species =
plant).select_related("activity").select_related("advicer")
                else:
                    activities =
activities.filter(plant_species = plant)
            else:
                return []
            activities = activities.order_by('-id')
            return activities

    def get_context_data(self, **kwargs):
        data_all_context = super(PlantspeciesActivity,
self).get_context_data(**kwargs)
        plant = self.request.GET.get('plant')
        if plant:

```

```

        plant = Plantspecies.objects.get(species_name =
plant)

        data_all_context['plant'] = plant
    return data_all_context

    def advicer_ask_views(request):
        advicer = Advicer.objects.get(user = self.request.user)
        asks_answered = AdvicerAsk.objects.filter(advicer =
advicer, answer__isnull =
False).select_related("gardener").order_by("-ask_date")
        asks_new = AdvicerAsk.objects.filter(advicer = advicer,
answer__isnull = True).select_related("gardener").order_by("-
ask_date")
        plant_asks_answered =
AdvicerAskPlant.objects.filter(advicer = advicer, answer__isnull
=
False).select_related("advicer").select_related("plant").order_b
y("-ask_date")
        plant_asks_new = AdvicerAskPlant.objects.filter(advicer
= advicer, answer__isnull =
True).select_related("gardener").select_related("plant").order_b
y("-ask_date")
        advices = ProblemAdvice.objects.filter(advicer =
advicer).select_related("advicer").select_related("problem").ord
er_by("-done_date")
        asks_answered =
AdvicerAskProblem.objects.filter(advicer = advicer, stage = 1)
        asks_new = AdvicerAskProblem.objects.filter(advicer =
advicer, stage = 0)
        asks_data = {
            "asks_answered": asks_answered,
            "asks_new": asks_new,
            "plant_asks_answered": plant_asks_answered,
            "plant_asks_new": plant_asks_new,
            "advices": advices,
            "asks_answered": asks_answered,
            "asks_new": asks_new
        }
        return render(request, "plantapp/asks.html", asks_data)

    def advicer_advices_views(request, advicer = None):
        if not advicer:
            advicer = Advicer.objects.get(user =
self.request.user)
        else:

```

```

        advicer = Advicer.objects.get(user__login =
advicer)
        advices = ProblemAdvice.objects.filter(advicer =
advicer).select_related("advicer").select_related("problem").ord
er_by("-done_date")
        advices_data = {
            "advicer": advicer,
            "advices": advices
        }
        return render(request, "plantapp/solutions.html",
advices_data)

```

```

class AdvicerAskPlantAnswerView(LoginRequiredMixin,
UpdateView):
    model = AdvicerAskPlant
    fields = ['answer']
    success_url =
reverse_lazy('plantapp:advicer_solutions')

    def get_queryset(self):
        updateview = super(AdvicerAskPlantAnswerView,
self).get_queryset()
        advicer = Advicer.objects.get(user =
self.request.user)
        return updateview.filter(advicer = advicer)

```

```

class AdvicerAskAnswerView(LoginRequiredMixin, UpdateView):
    model = AdvicerAsk
    fields = ['answer']
    success_url = reverse_lazy('plants:advicer_solutions')

    def get_queryset(self):
        updateview = super(AdvicerAskAnswerView,
self).get_queryset()
        advicer = Advicer.objects.get(user =
self.request.user)
        return updateview.filter(advicer = advicer)

```

```

class ProblemAdviceAnswerView(LoginRequiredMixin,
CreateView):
    model = ProblemAdvice
    fields = ['problem', 'memo', 'disease']
    success_url = reverse_lazy('plants:advicer_advices')

    def dispatch(self, *args, **kwargs):

```

```

        self.problem = Problem.objects.get(id =
kwargs['problem'], None)
        return super(ProblemAdviceAnswerView,
self).dispatch(*args, **kwargs)

    def get_context_data(self, **kwargs):
        problem_advice_context =
super().get_context_data(**kwargs)
        problem_advice_context["problem"] = self.problem
        return plant_activity_context

    def form_valid(self, form):
        advicer = Advicer.objects.get(user =
self.request.user)
        form.instance.advicer = advicer
        return super().form_valid(form)

```

Б.2 Текст файла admin.py

```

from django.contrib import admin
from .models import AdvicerAssess
from .models import Advicer
from .models import Plantspecies
from .models import Plantcategory
from .models import Plantfamily
from .models import PlantspeciesHealthbenefit
from .models import Healthbenefit
from .models import AdvicerAsk
from .models import AdvicerAskPlant
from .models import ProblemAdvice
from .models import Tutorial
from .models import TutorialSave
from .models import Activity
from .models import PlantspeciesDisease
from .models import PlantspeciesActivity
from .models import Disease
from .models import DiseaseCase
from .models import Plant
from .models import Companion
from .models import Gardener
from .models import PlantNote
from .models import PlantHistory
from .models import PlantActivity
from .models import PlantActivityHistory
from .models import Problem

```

```

from .models import AdvicerAskProblem
from .models import Subscribe
from .models import Location
from .models import GardenerLocation

admin.site.register(Plantcategory)
admin.site.register(Plantfamily)
admin.site.register(Plantspecies)
admin.site.register(Healthbenefit)
admin.site.register(PlantspeciesHealthbenefit)
admin.site.register(Companion)
admin.site.register(Gardener)
admin.site.register(PlantNote)
admin.site.register(PlantHistory)
admin.site.register(Activity)
admin.site.register(PlantActivity)
admin.site.register(PlantActivityHistory)
admin.site.register(Disease)
admin.site.register(PlantspeciesDisease)
admin.site.register(ProblemPic)
admin.site.register(Advicer)
admin.site.register(AdvicerFollowing)
admin.site.register(Wishlist)
admin.site.register(TutorialSave)
admin.site.register(Location)
admin.site.register(GardenerLocation)

class PlantspeciesActivityAdmin(admin.ModelAdmin):
    exclude = ["advicer"]

class PlantAdmin(admin.ModelAdmin):
    exclude = ["gardener"]

class DiseaseCaseAdmin(admin.ModelAdmin):
    exclude = ["advicer"]

class ProblemAdviceAdmin(admin.ModelAdmin):
    exclude = ["advicer", "done_date"]

class TutorialAdmin(admin.ModelAdmin):
    exclude = ["advicer", "done_date"]

class AdvicerAskAdmin(admin.ModelAdmin):
    exclude = ["ask_date", "answer_date"]

class AdvicerAskProblemAdmin(admin.ModelAdmin):

```

```

        exclude = ["done_date"]

class AdvicerAskPlantAdmin(admin.ModelAdmin):
    exclude = ["ask_date", "answer_date"]
class ProblemAdmin(admin.ModelAdmin):
    exclude = ["done_date"]

class SubscribeAdmin(admin.ModelAdmin):
    exclude = ["done_date"]

class AdvicerAssessAdmin(admin.ModelAdmin):
    exclude = ["done_date", "advicer"]

    admin.site.register(PlantspeciesActivity,
PlantspeciesActivityAdmin)
    admin.site.register(Plant, PlantAdmin)
    admin.site.register(DiseaseCase, DiseaseCaseAdmin)
    admin.site.register(ProblemAdvice, ProblemAdviceAdmin)
    admin.site.register(Tutorial, TutorialAdmin)
    admin.site.register(AdvicerAsk, AdvicerAskAdmin)
    admin.site.register(AdvicerAskProblem,
AdvicerAskProblemAdmin)
    admin.site.register(AdvicerAskPlant, AdvicerAskPlantAdmin)
    admin.site.register(Problem, ProblemAdmin)
    admin.site.register(Subscribe, SubscribeAdmin)
    admin.site.register(AdvicerAssess, AdvicerAssessAdmin)

```

ДОДАТОК В
Слайди презентації

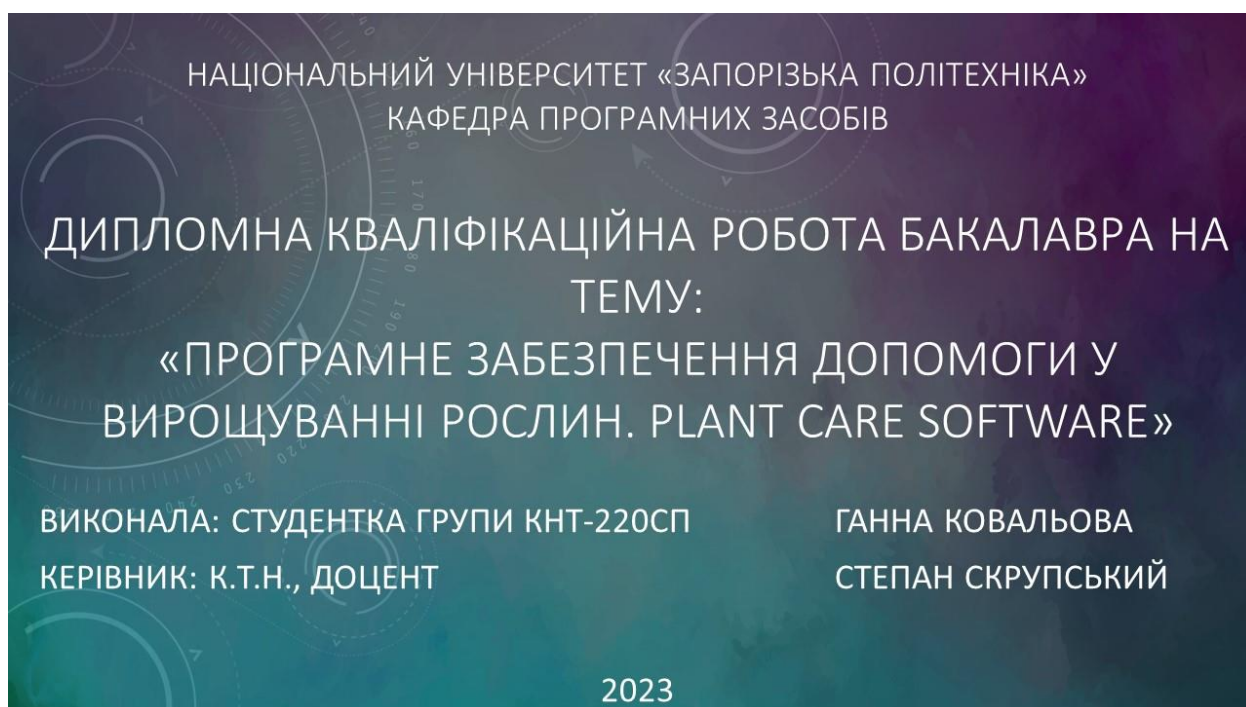


Рисунок В.1 – Слайд титульний

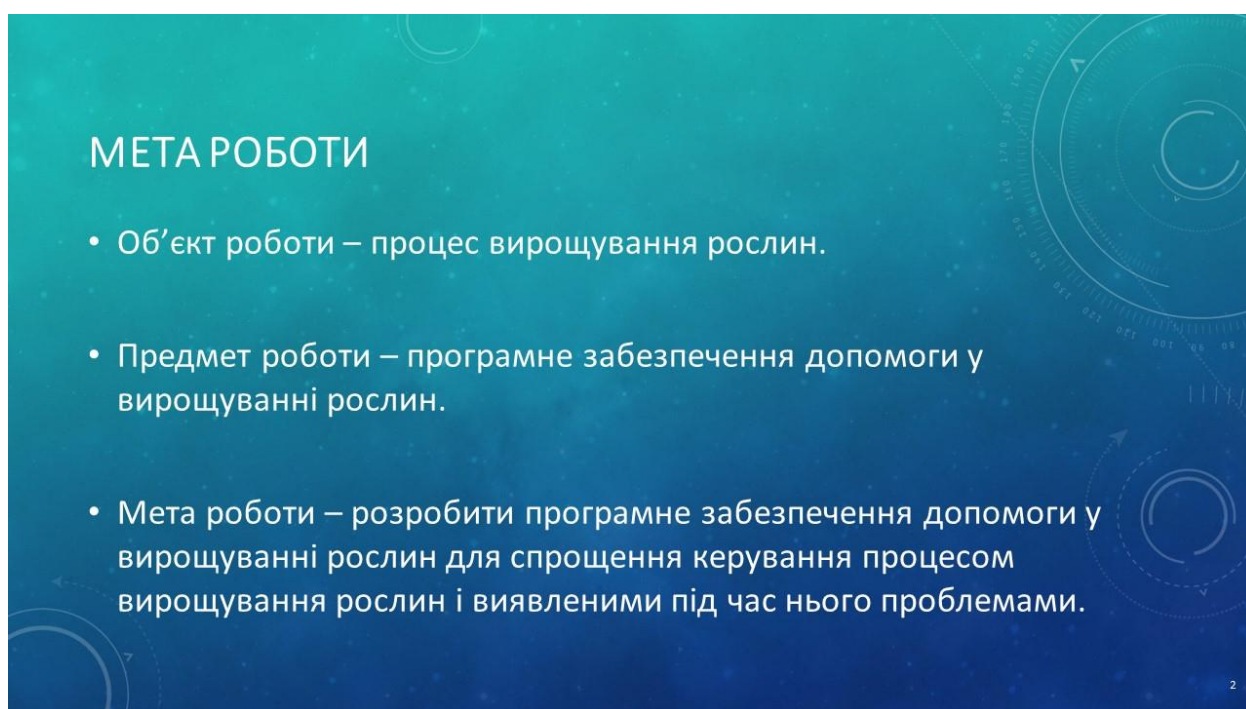


Рисунок В.2 – Мета роботи

ЗАВДАННЯ РОБОТИ

- аналіз програмних аналогів і визначення концепції програми;
- проектування програмного забезпечення допомоги у вирощуванні рослин;
- реалізація вебдодатку допомоги у вирощуванні рослин.

3

Рисунок В.3 – Завдання роботи

РЕЗУЛЬТАТИ АНАЛІЗУ ПРОГРАМНИХ ЗАСОБІВ ДОПОМОГИ У ВИРОЩУВАННІ РОСЛИН

Ознака	Planta	Vera	Розробка
Пошук видів рослин	+	–	+
Внесення власних рослин	+	+	+
Планування процедур з догляду	+	+	+
Отримання інструкцій	+	–	+
Наявність функцій стосовно захворювань рослин	–	–	+
Отримання порад від експертів	–	–	+

4

Рисунок В.4 – Результати аналізу програмних засобів допомоги у вирощуванні рослин

ДІАГРАМИ ПРЕЦЕДЕНТІВ

The image displays two precedent diagrams (flowcharts) representing expert knowledge. Both diagrams start with a central node labeled 'Спеціаліст' (Specialist) on the left and 'Експерт' (Expert) on the right, respectively. These nodes are connected to a series of green oval nodes representing various tasks and procedures. The nodes are organized into a hierarchical structure, with arrows indicating the flow of information or the sequence of actions. The left diagram is more complex, with many more nodes and connections, while the right diagram is simpler and more focused on a specific set of tasks. The nodes are labeled with Ukrainian text, such as 'переклад інформації про вид рослини' (translation of information about the plant species), 'переклад специфічної процедури з дискладу на власний розуміння' (translation of a specific procedure from a diagram to personal understanding), and 'надання поради' (providing advice).

Рисунок В.6 – Діаграми прецедентів

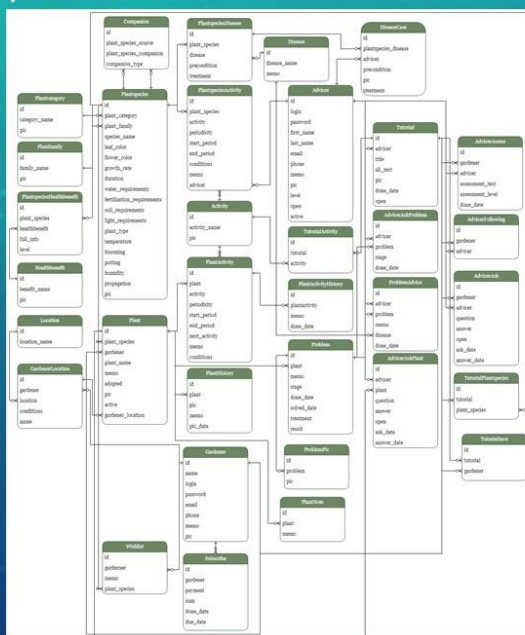


Рисунок В.7 – Схема бази даних

ДІАГРАМА КЛАСІВ

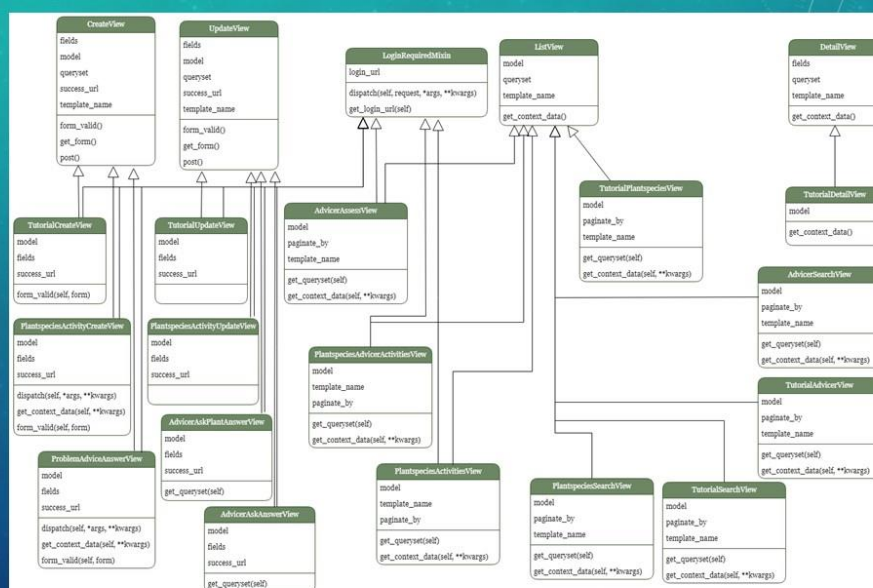


Рисунок В.8 – Діаграма класів

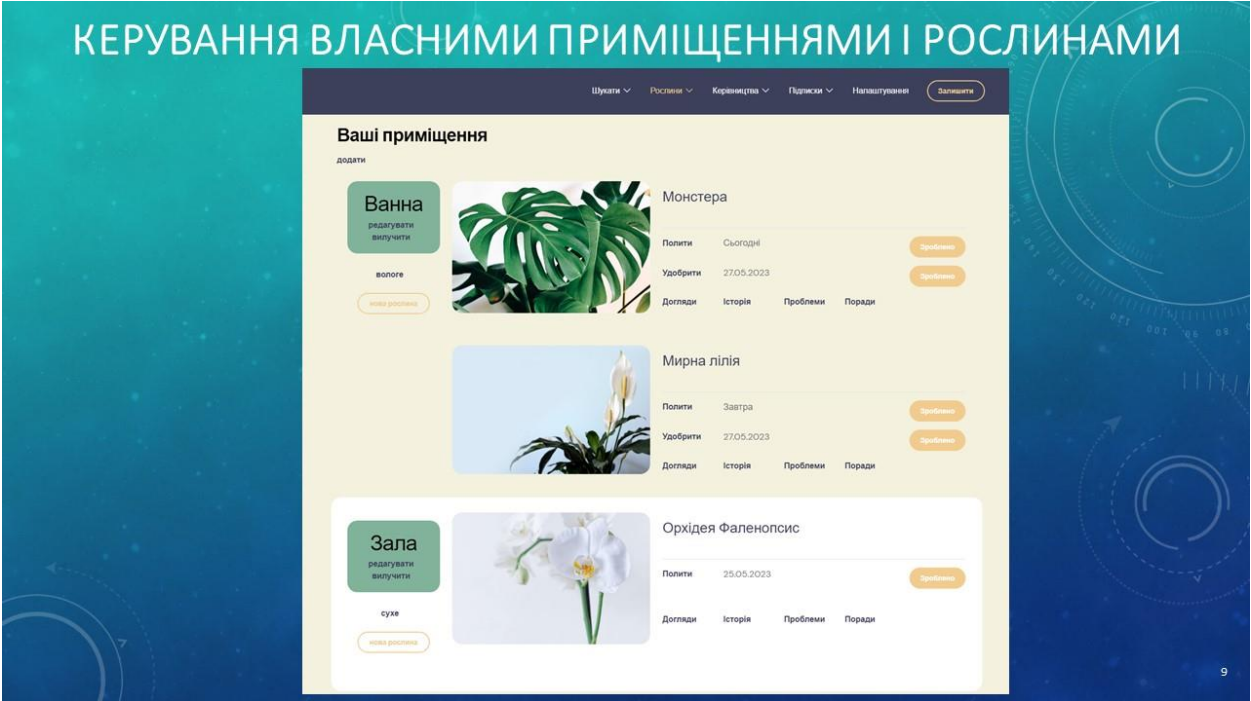


Рисунок В.9 – Керування власними приміщеннями і рослинами

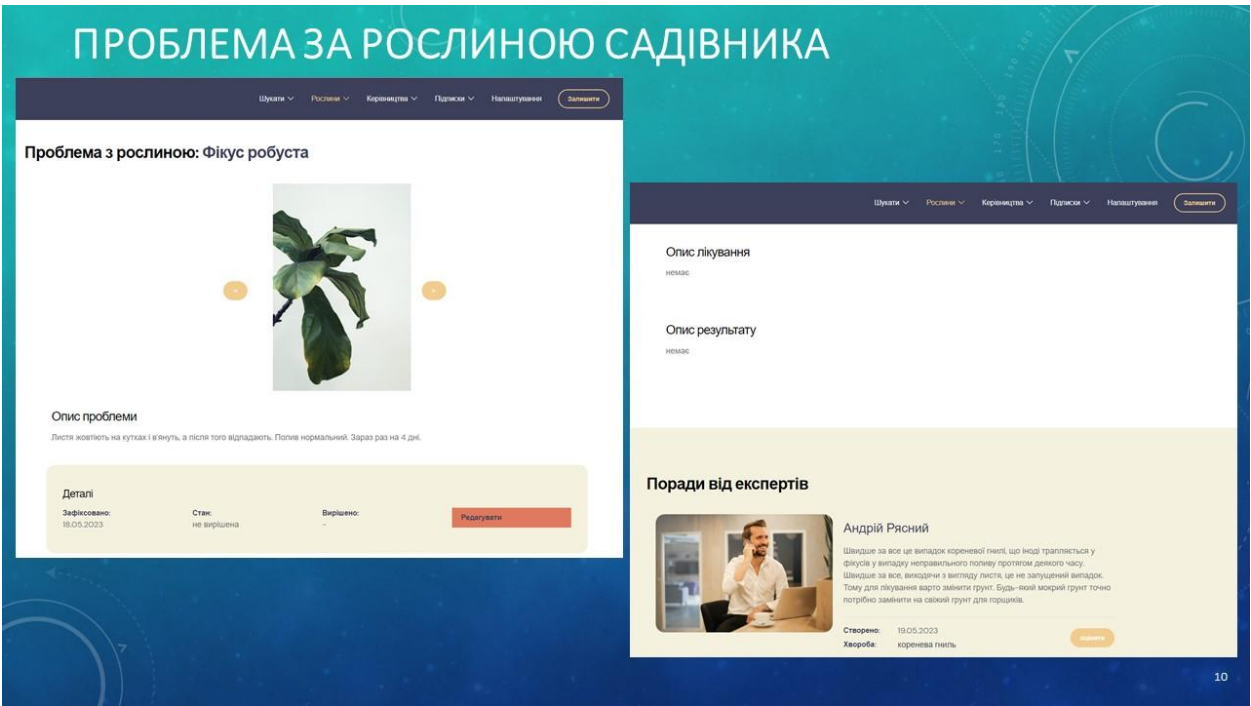


Рисунок В.10 – Проблема за рослиною садівника

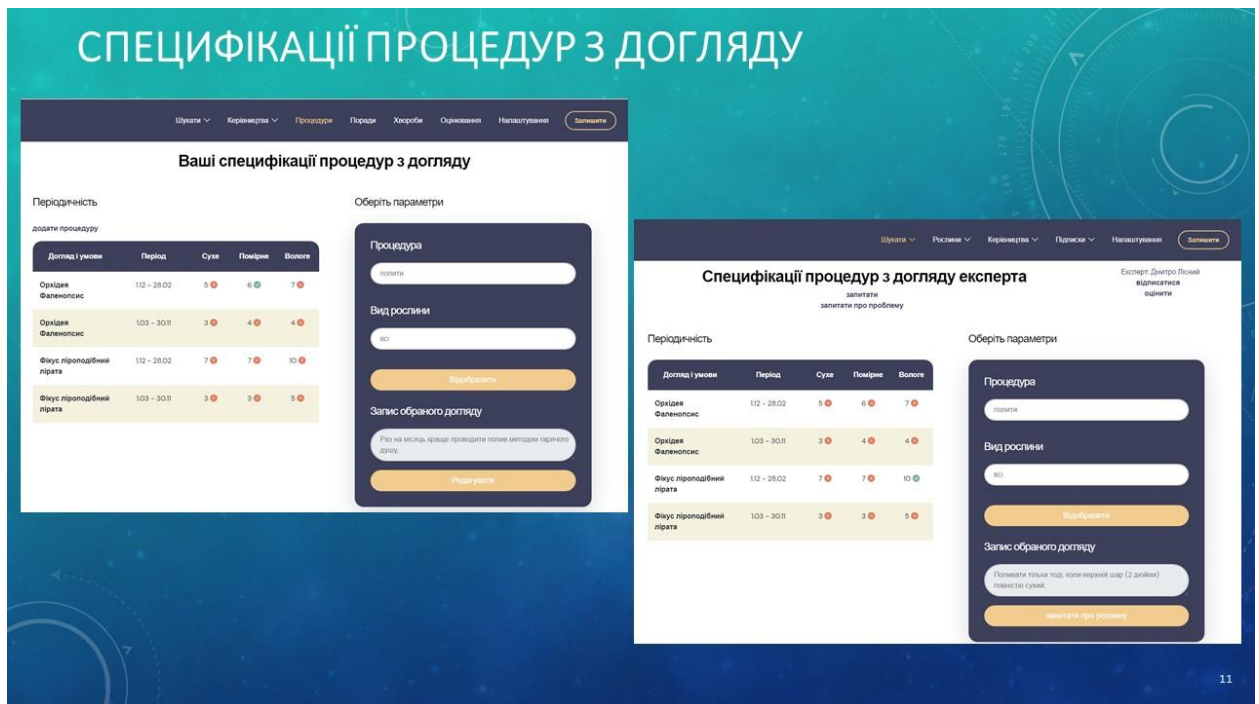


Рисунок В.11 – Специфікації процедур з догляду

ВИСНОВКИ

Аналіз існуючих програмних засобів допомоги у вирощуванні рослин дозволив виявити основні функції таких програми, а також наявні прогалини. Це дозволило в підсумку сформулювати концепцію програми-розробки.

Усі необхідні завдання з проектування програми в роботі виконано: визначено основну мову для розробки, якою є Python, фреймворк, який підтримує веброботку, яким є Django, сформульовано сценарії роботи садівників і експертів з програмою, визначено структуру бази даних, включаючи потрібні сутності, представлення цих сутностей в реляційній базі даних. Визначення структури бази даних сформувало фактично основу реалізованих класів моделей у структурі результуючого MVT-додатку.

Усі необхідні завдання з реалізації програми виконано: за допомогою патерну MVT на основі фреймворку Django реалізовано функції програми через відповідні класи і функції, що подано в розділі.

Основні сценарії роботи садівника та експерта в чіткій відповідності з виконаним раніше описом реалізації презентовані. Перед тим забезпечено тестування функцій в браузерах Chrome і Firefox з успішним виконанням.

Рисунок В.12 – Висновки