

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Вебсистема обліку рослин»

Виконав студент групи ВТ-22-1
САВЕНЕЦЬ Нікіта Миколайович

Керівник роботи:
ДУНЄВ Сергій Сергійович

Рецензент:
САВІЦЬКИЙ Роман Святославович

Житомир - 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **САВЕНЕЦЬ Нікіта Миколайович**

Керівник роботи: **ДУНЄВ Сергій Сергійович**

Тема роботи: **«Вебсистема обліку рослин»,**

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: **використання веб-орієнтованих технології для розробки системи обліку рослин; автоматизація управління процесами обліку рослин**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Сергій ДУНЄВ	02.03.2026	02.03.2026
2	Сергій ДУНЄВ	04.04.2026	04.04.2026
3	Сергій ДУНЄВ	02.05.2026	02.05.2026

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра присвячена проектуванню та програмній реалізації веб-орієнтованої системи для ведення обліку рослин. Робота складається з пояснювальної записки обсягом 89 сторінок та реалізованого програмного комплексу. Записка містить 38 ілюстрацій, 8 таблиць, 6 лістингів коду, 4 додатки і спирається на 33 літературних джерел.

Мета роботи - розроблення алгоритмічного та програмного забезпечення, яке автоматизує внесення, зберігання й пошук інформації про рослини у форматі веб-додатку.

У межах дослідження сформульовано перелік функціональних і нефункціональних вимог до майбутньої системи, виконано порівняльний аналіз наявних програмних аналогів та виокремлено їх сильні та слабкі сторони. Спроектовано архітектуру клієнт-серверного застосунку, описано взаємодію його модулів, побудовано об'єктну модель (діаграма класів) і структуру бази даних. Окремо подано опис користувацького інтерфейсу та результати тестування реалізованого програмного комплексу у поширених веб-браузерах.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, ОБЛІК РОСЛИН, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, БАЗА ДАНИХ.

					ІПЗ.КР.Б-121-26-ПЗ						
Зм.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Савенець Н.М.			Вебсистема обліку рослин			Літ.	Арк.	Аркушів	
Керівник		Дунєв С.С.								3	89
								ФІКТ Гр. ВТ-22-1			
Рецензент		Савіцький Р.С.									
Зав.каф.		Вакалюк Т.А.									

ABSTRACT

The bachelor final qualification work is dedicated to the design and software implementation of a web-oriented system for plant accounting. The work consists of an explanatory note of 89 pages and the implemented software package. The note includes 38 figures, 8 tables, 6 code listings, 4 appendices and relies on 33 literary sources.

The aim of the work is to develop algorithmic and software support that automates the entry, storage and search of plant information in the form of a web application.

Within the research, the list of functional and non-functional requirements for the future system is formulated, a comparative analysis of existing software analogues is performed, and their strengths and weaknesses are identified. The architecture of a client-server application is designed, the interaction of its modules is described, an object model (class diagram) and database structure are built. The description of the user interface and the testing results of the implemented software package in widely used web browsers are presented separately.

KEY WORDS: WEB APPLICATION, PLANT ACCOUNTING, USER INTERFACE, CLIENT-SERVER ARCHITECTURE, DATABASE.

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ АВТОМАТИЗАЦІЇ РОБОТИ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ ОБЛІКУ РОСЛИН.....	10
1.1. Постановка задачі.....	10
1.2. Аналіз аналогів програмного продукту	12
1.3. Вибір архітектури системи обліку рослин	18
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	23
Висновки до розділу 1	29
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ОБЛІКУ РОСЛИН	30
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	30
2.2 Розробка бази даних системи.....	33
2.3 Проектування та реалізація алгоритмів роботи системи	37
2.4 Реалізація додатку системи обліку рослин.....	40
Висновки до розділу 2	49
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ОБЛІКУ РОСЛИН	51
3.1 Порядок встановлення та налаштування системи	51
3.2 Структура інтерфейсу	54
3.3 Тестування роботи	63
Висновки до розділу 3	74
ВИСНОВКИ	76

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 78

ДОДАТКИ..... 81

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - прикладний програмний інтерфейс

БД - база даних

ПЗ - програмне забезпечення

СУБД - система управління базами даних

SDK - набір для розробки програмного забезпечення (software development kit)

MVC - схема розділення даних додатків і керуючої логіки на три окремі компоненти Model-View-Controller

CRUD - основні операції, що використовуються для управління даними (Create, Read, Update, Delete)

ВСТУП

Актуальність теми. Інформаційні технології сьогодні розвиваються надзвичайно стрімко, зокрема у напрямку збереження та опрацювання великих обсягів даних. Постійно з'являються нові інструменти, що дають змогу створювати програмні рішення, які водночас є швидкодіючими, надійними і легко масштабованими. Такі продукти мають забезпечувати оперативне внесення, пошук та обробку інформації. Сучасне виробництво, освіта, наука, культура, спорт та економіка вже не можуть існувати без використання обчислювальної техніки, яка стала невід'ємним помічником людини як у професійній діяльності, так і у повсякденному житті.

Запропонована робота присвячена систематизації та зберіганню інформації про рослини. Цільовою аудиторією додатку у першу чергу є садівники-аматори, для яких колекціонування й вирощування рослин є справою всього життя. Як правило, такі ентузіасти збирають свої колекції роками, проте не мають ані ресурсів, ані технічних навичок для створення власного веб-ресурсу для їх каталогізації. Інформація про їх рослини часто розпорошена по коментарях у тематичних спільнотах Facebook, відеороликах на YouTube та інших соціальних платформах, а пошук контактів власника рідкісної рослини зазвичай перетворюється на ланцюжок звернень до спільних знайомих. Розроблюваний у роботі веб-додаток дає таким користувачам зручний інструмент для ведення власної бази рослин, фіксування спостережень та підтримання зв'язку з іншими садівниками для обміну посадковим матеріалом. Початківці отримують зручний механізм пошуку та контактів із досвідченими садівниками, а власники колекцій - додатковий канал для популяризації своїх рослин і реалізації надлишків посадкового матеріалу.

Метою роботи є розроблення алгоритмів [1, 2] та програмна реалізація веб-орієнтованої системи обліку рослин.

Для досягнення поставленої мети у роботі необхідно вирішити такі завдання:

- виконати аналіз процесів обліку рослин та напрямків їх автоматизації;

					ІІЗ.КР.Б-121-26-ІІЗ	Арк
						8

- обґрунтовано обрати архітектурний підхід до побудови програмного комплексу;
- здійснити вибір інструментальних засобів реалізації системи;
- спроектувати структурні компоненти та алгоритми роботи системи;
- реалізувати програмний додаток;
- провести тестування розробленого програмного забезпечення.

Об’єктом дослідження є автоматизація управління процесами обліку рослин.

Предметом дослідження є застосування веб-орієнтованих технологій для побудови системи обліку рослин.

Методи дослідження. У роботі використано методи системного аналізу, об’єктно-орієнтованого проектування, веб-програмування та проектування користувацьких інтерфейсів.

Практичне значення отриманих результатів. Реалізована програмна система може бути використана садівниками-аматорами для ведення власних колекцій рослин, а також компаніями, що займаються продажем посадкового матеріалу.

Структура та обсяг роботи. Пояснювальна записка складається з вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг - 89 сторінок основного тексту, 38 ілюстрацій, 8 таблиць, 6 лістингів коду і 4 додатки. Список використаних джерел налічує 33 найменувань.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ АВТОМАТИЗАЦІЇ РОБОТИ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ ОБЛІКУ РОСЛИН

1.1. Постановка задачі

Розробка програмного продукту, який має ефективно працювати з великими обсягами даних, бути масштабованим та легко розширюваним, починається з ретельного аналізу предметної області та обґрунтованого підбору технологічного стеку. Лише за умови коректно виконаних попередніх кроків можна сформувати надійний фундамент, що відповідатиме поставленим вимогам до програмного забезпечення. Особливістю предметної області обліку рослин є те, що інформація про колекції приватних садівників майже завжди залишається розпорошеною - частина зберігається у нотатниках, частина у фотогалереях смартфонів, частина у коментарях на тематичних форумах та у відеоматеріалах соціальних мереж. Це призводить до того, що цінні знання про рідкісні сорти, особливості догляду й контактні дані власників доступні лише вузькому колу осіб, а пошук необхідного посадкового матеріалу часто перетворюється на ланцюжок особистих звернень та запитів через спільних знайомих.

Випускна кваліфікаційна робота бакалавра передбачає створення веб-орієнтованої системи, яка надасть користувачам зручний інструмент для ведення обліку власних рослин. Програмний продукт реалізується у форматі веб-додатку, що дає змогу заводити окремі картки рослин, розподіляти їх за заздалегідь визначеними категоріями або формувати власні, додавати до кожного запису текстовий опис, прикріплювати фотографію та публікувати контактні дані для подальшого спілкування з потенційними покупцями. Окрім роботи з власною колекцією, користувач має змогу здійснювати глобальний пошук рослин по всій базі, відшуковуючи цікаві екземпляри з колекцій інших садівників. Усі взаємодії з системою відбуваються через стандартний веб-браузер без необхідності встановлювати додаткове програмне забезпечення, що принципово важливо для

					ІІЗ.КР.Б-121-26-ІІЗ	Арк
						10

широкого охоплення цільової аудиторії незалежно від типу пристрою та операційної системи.

Основним джерелом наповнення бази виступають самі користувачі системи: вносити та змінювати записи мають право лише авторизовані учасники, що забезпечує контроль за достовірністю опублікованих даних. Адміністратори системи отримують додатковий рівень доступу, що дозволяє контролювати якість опублікованого контенту, керувати переліком категорій рослин та опрацьовувати звернення від користувачів. Цільовою аудиторією додатку є люди, що займаються вирощуванням рослин - як аматори, які ведуть колекції для себе, так і ті, хто веде комерційну діяльність у цій сфері, реалізуючи надлишок посадкового матеріалу або рідкісні сорти. Розроблюваний додаток одночасно вирішує дві задачі: для досвідчених садівників він стає зручним архівом колекції та каналом для популяризації своїх рослин, а для початківців - джерелом інформації про доступний посадковий матеріал та точкою контакту з компетентними виробниками.

Серед типових проблем, які мають бути усунуті в межах розробки, виокремлюються відсутність єдиного централізованого реєстру колекцій, низька якість пошуку рослин за родовою чи побутовою назвою у наявних відкритих джерелах, складність встановлення прямого контакту з власником конкретного зразка, а також брак уніфікованого формату подання інформації про рослину, що ускладнює порівняння та опрацювання даних. Окремою проблемою є відсутність у наявних аналогах механізмів автоматичної допомоги при заповненні картки рослини, через що звичайному користувачу доводиться вручну з'ясовувати наукову назву, родину та інші довідкові відомості, що є відчутним бар'єром входу для початківців. Все це формує запит на створення цілісної програмної платформи, яка зосередить розрізнені дані в одній точці доступу та надасть прості та інтуїтивно зрозумілі інструменти роботи з ними.

Розроблюване програмне забезпечення має реалізовувати реєстрацію нового облікового запису та автентифікацію наявних користувачів, редагування персональних відомостей у профілі користувача, ведення обліку рослин у

					ІІЗ.КР.Б-121-26-ІІЗ	Арк
						11

форматі повного циклу управління записами (додавання, редагування та видалення), глобальний пошук рослин і отримання контактних даних їх власників, а також надсилання відгуків та пропозицій адміністраторам системи. Підрозділ «Мої рослини» забезпечує користувачеві повний цикл управління власними записами - створення, редагування та видалення, - а опубліковані записи паралельно стають доступними іншим користувачам через механізм глобального пошуку.

1.2. Аналіз аналогів програмного продукту

З метою формування переліку функцій, бізнес-вимог та поведінкової логіки майбутньої системи проведено огляд програмних рішень, що вже представлені на ринку. Аналіз існуючих продуктів дозволив виявити їх найкращі практики й типові слабкі місця, з урахуванням яких сформульовано вимоги до розроблюваної у роботі веб-системи обліку рослин. Розглянемо два найбільш репрезентативних рішення - настільний додаток «Моя оранжерея» [32] та веб-сервіс «Thlaspi» [33], - які демонструють два принципово різних архітектурних підходи до автоматизації обліку домашніх та садових рослин.

Додаток «Моя оранжерея» [32] являє собою настільну програму, орієнтовану на квітникарів-аматорів, які бажають упорядкувати інформацію про власну колекцію рослин у зручній цифровій формі. Загальний вигляд головного вікна програми наведено на (рис. 1.1).

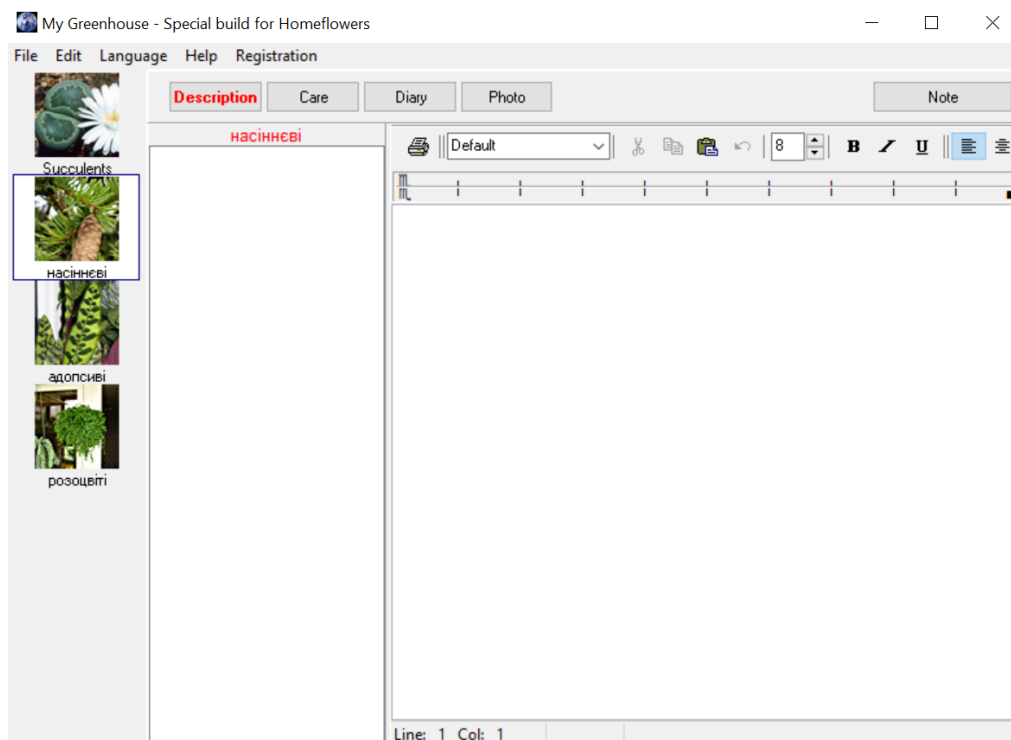


Рис. 1.1 - Головне вікно додатку

Функціонально додаток організований так, що у лівій частині вікна розташований блок зі списком рослин, у якому за допомогою контекстного меню створюються нові записи. Поряд із цим списком розташовано панель піктограм-розділів, перемикання між якими відкриває відповідний набір рослин. За замовчуванням у програмі присутні три тематичні розділи - «Декоративно-листяні», «Декоративно-квітучі» та «Сукуленти», - користувач може створювати власні розділи, видаляти наявні, перейменовувати їх та змінювати порядок у списку. Під час створення нового розділу програма пропонує обрати ілюстрацію зі стандартного набору або завантажити власне зображення, яке автоматично масштабується до потрібного розміру. У межах кожного розділу можна формувати додатковий рівень структури - папки, причому переміщення рослини між папками виконується простим перетягуванням, а зміна розділу - через контекстне меню (рис. 1.2) або шляхом перетягування запису на піктограму відповідного розділу/розділу.

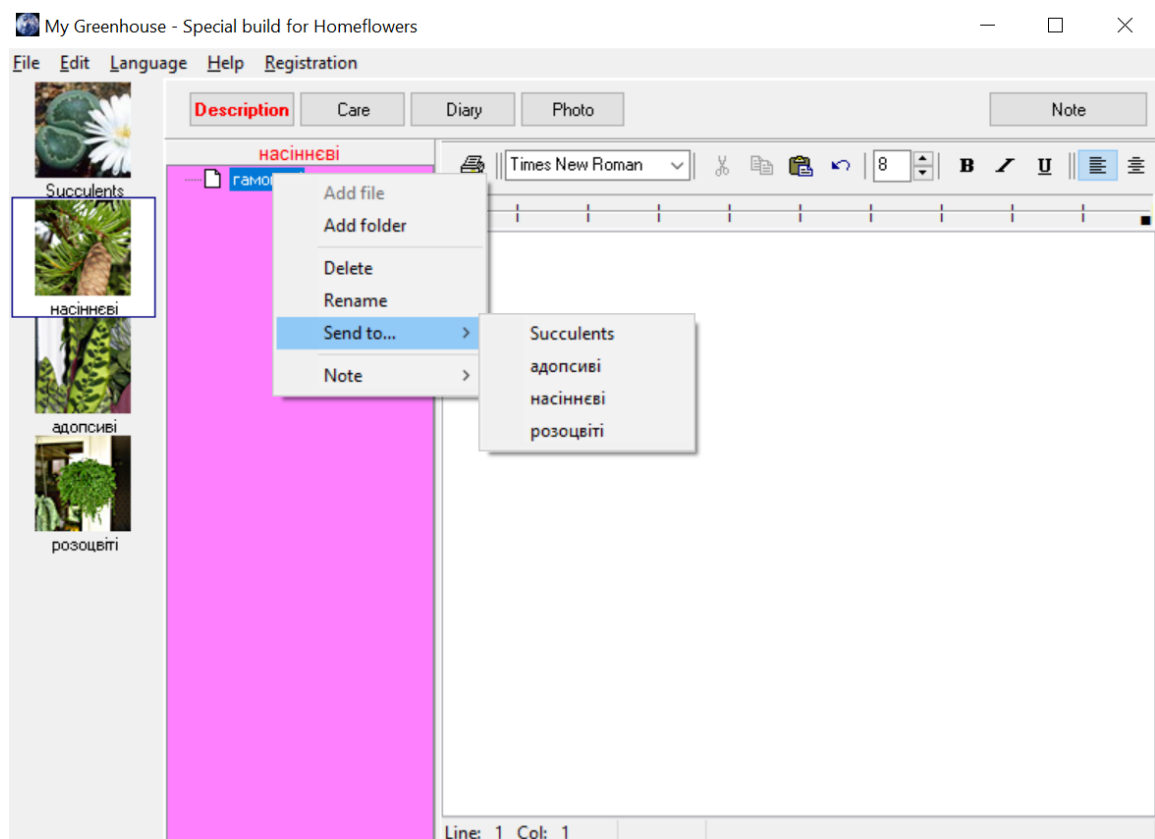


Рис. 1.2 - Меню для переміщення рослини між розділами

Окремо в додатку реалізовано систему умовних позначок стану рослини, доступних через контекстне меню (рис. 1.3): користувач може помітити запис як «Ідеальний стан», «Пересадити», «Терміново пересадити» або «Лікувати», що формує своєрідний індикатор поточного стану колекції.

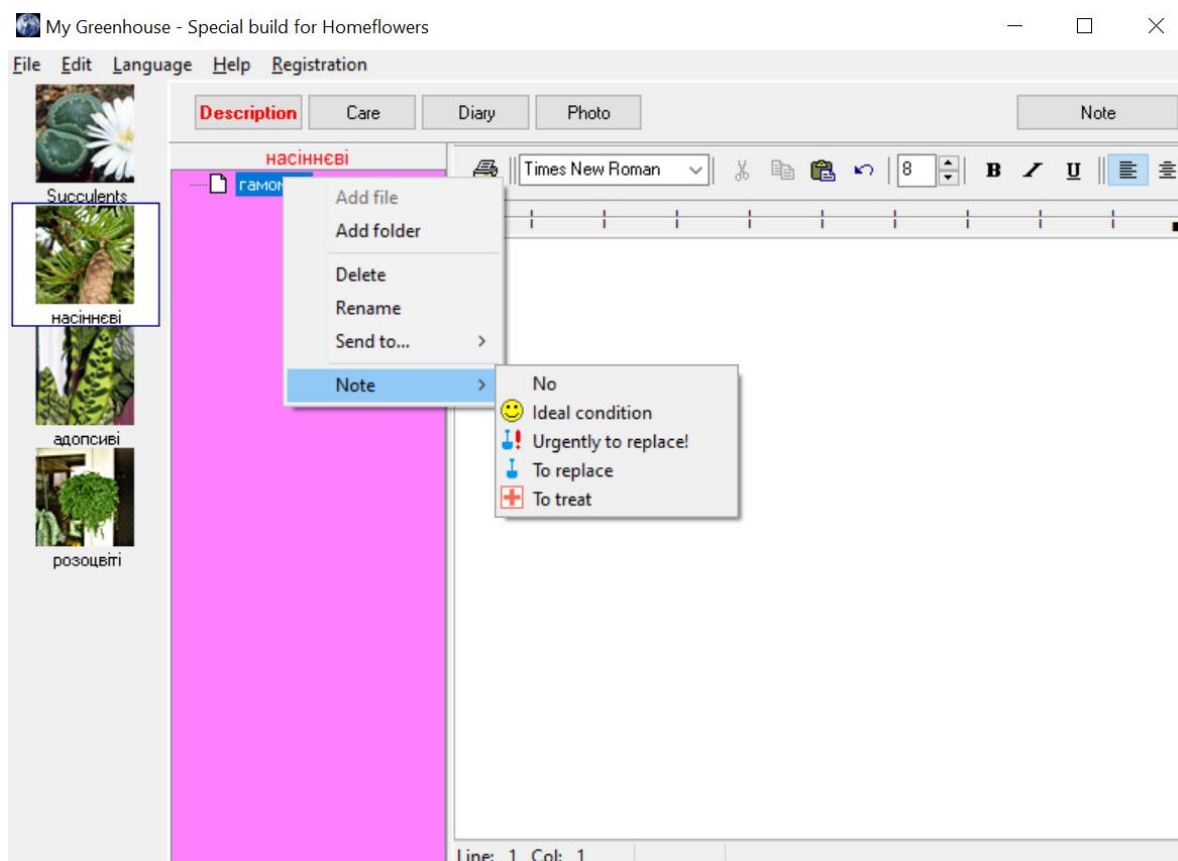


Рис. 1.3 - Додавання умовної позначки для рослини

Над основним списком розташовано чотири перемикачі, які логічно розподіляють інформацію про рослину на чотири окремі набори - опис зовнішнього вигляду, опис особливостей догляду, журнал спостережень («Щоденник») та фотоальбом. У вкладках «Догляд» та «Щоденник» користувач може фіксувати індивідуальні особливості рослини, відмічати виконані дії з догляду (пересадку, підживлення тощо), а також планувати майбутні роботи у вигляді графіка підживлень. Робота з фотоальбомом організована у вкладці «Фото» - світлини можна додавати, редагувати та видаляти за допомогою відповідних кнопок без обмеження кількості зображень, а під кожною з них розміщено поле для текстового коментаря, у якому зручно фіксувати дату знімка, вік рослини та інші супутні відомості. Опис і світлини можна додавати як до окремих файлів, так і спільно для всіх файлів однієї папки. Додатково у програмі реалізовано блок «Нотатки», що не пов'язаний із групами рослин і дозволяє вести довільні записи у власній структурі папок та файлів. Загалом додаток дозволяє сформувати повний електронний каталог особистої колекції та зосередити в

одному місці всю інформацію щодо догляду за рослинами, що робить його зручним інструментом для аматорського каталогу.

До характерних особливостей рішення слід віднести інтуїтивно зрозумілий інтерфейс, простоту освоєння користувачем-новачком та відсутність надлишкових деталей у форматі картки рослини. Водночас додаток розрахований виключно на локальне використання - для його роботи необхідне попереднє встановлення на пристрій користувача, що обмежує мобільність роботи з колекцією; обмежена глибина вкладеності папок не завжди дозволяє створити необхідну деталізацію розділів для великих колекцій; у додатку відсутній механізм публічного доступу до колекції, тож переглянути список рослин може лише власник пристрою, на якому встановлена програма, що повністю виключає взаємодію між власниками різних колекцій.

Веб-сервіс «Thlaspi» [33] є онлайн-платформою для ведення колекції рослин у режимі реального часу, обліку виконаних робіт та контролю стану насаджень. На відміну від попереднього рішення, сервіс не вимагає встановлення додаткового програмного забезпечення та доступний з будь-якого пристрою через стандартний веб-браузер. До основного функціоналу сервісу належить ведення особистого садового щоденника із зазначенням, коли і що було посаджено; формування календарного плану робіт у саду як для окремої рослини, так і для груп, з аналізом ефективності виконаних дій у багаторічній перспективі та можливістю отримання нагадувань про заплановані операції; створення повного онлайн-переліку рослин конкретного саду; публікація власних статей та читання дописів інших користувачів для обміну досвідом; створення сторінки власного саду та сторінок окремих рослин для соціальної взаємодії з іншими садівниками.

Зліва «на сторінці зареєстрованого користувача розміщено перелік розділів. Вкладка «Мій сад» (рис. 1.4) дозволяє створювати нові записи про рослини, перелік яких відображається на сторінці саду (рис. 1.5).



Рис. 1.4 - Головне вікно додатку

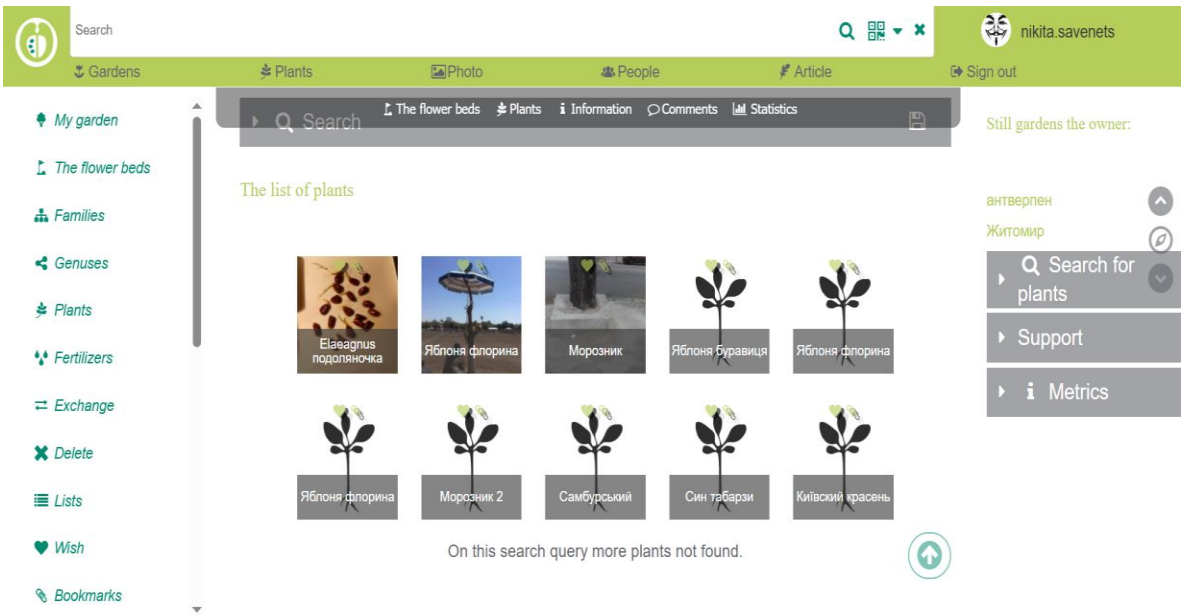


Рис. 1.5 - Список рослин на сторінці саду

Загальний перелік рослин з можливістю фільтрації та сортування за різними ознаками доступний на окремій вкладці «Рослини» (рис. 1.6).

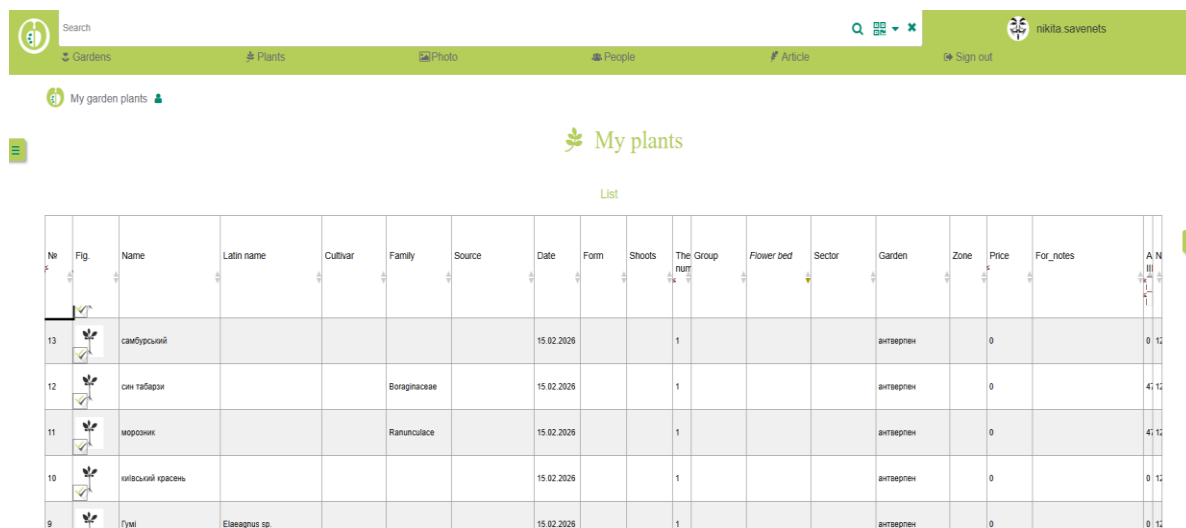


Рис. 1.6 - Список рослин на вкладці “Рослини”

У сервісу відсутня потреба у локальному встановленні та відкритість створених користувачем переліків рослин для перегляду іншим відвідувачам сайту, у тому числі неавторизованим. Водночас інтерфейс сервісу важко назвати інтуїтивно зрозумілим - навігація по сторінках непрозора та потребує тривалого освоєння, реалізація пошуку незручна та не дозволяє швидко знайти необхідний зразок, відсутнє явне групування рослин за категоріями, на сторінках відображається значна кількість сторонніх відомостей (зокрема порожні поля інших користувачів), а відсутність вбудованих інструкцій із роботи із сервісом особливо ускладнює знайомство нових користувачів із системою.

1.3. Вибір архітектури системи обліку рослин

Розроблюваний додаток умовно поділяється на дві логічні частини - клієнтську (Front-End) та серверну (Back-End). Клієнтська частина реалізована з використанням HTML, CSS, JavaScript, фреймворку Vue.js та компонентної бібліотеки Quasar [20]. Серверна частина побудована на платформі Node.js із застосуванням каркаса Express [22] та ORM-бібліотеки Sequelize [23], а зберігання даних виконує реляційна СУБД MySQL. Front-End-частина відповідає за все, що користувач безпосередньо бачить та з чим взаємодіє у браузері - відображення сторінок, обробку дій користувача, формування запитів до

серверного API. Іншими словами, фронтенд охоплює всю візуальну та інтерактивну складову веб-додатку, що виконується на стороні клієнта.

Серверна частина зосереджена в каталозі `app/router/router.js`, де описано весь набір ендпоінтів API, та у файлі `server.js`, який експортує модуль сервера. Запуск серверної компоненти передбачає послідовне виконання кількох взаємопов'язаних дій: спершу створюється та конфігурується екземпляр сервера, після чого підключаються необхідні плагіни та реєструються допоміжні обробники, що викликаються на різних етапах життєвого циклу запиту. Далі обирається стратегія автентифікації, на сервері реєструються всі ендпоінти додатку та, нарешті, запускається процес прослуховування мережових з'єднань. Завдяки можливостям Node.js та ORM-бібліотеки Sequelize встановлюється програмне з'єднання з базою даних, що у подальшому дозволяє виконувати над нею весь спектр операцій - від запитів на вибірку до операцій модифікації даних.

В основу системи покладено клієнт-серверну архітектуру [5, 9, 17]. Цей архітектурний стиль передбачає поділ функціональності програмного комплексу на окремі шари, кожен з яких може бути фізично розгорнутий на іншому пристрої. Історично такі системи розвивалися у напрямку компонентно-орієнтованого підходу, де компоненти спілкуються між собою за допомогою специфічних для платформи механізмів виклику замість обміну повідомленнями. Сфера застосування подібного архітектурного стилю охоплює як веб-додатки, так і розподілені системи різного призначення.

Програмний комплекс, побудований за принципами клієнт-серверної архітектури, складається з трьох логічно та фізично розділених рівнів: рівня представлення, рівня бізнес-логіки та рівня даних. Кожен з рівнів проектується, реалізується та супроводжується незалежно від інших. На рівні презентації формується користувацький інтерфейс програми. У більшості сценаріїв він представлений графічним інтерфейсом для роботи з товстими клієнтами та веб-інтерфейсом для роботи через браузер.

Презентаційний шар відображає інформацію, пов'язану з основними сценаріями використання, та обмінюється даними з рештою рівнів через формування запитів та обробку відповідей.

Логічний рівень, відомий також як шар бізнес-логіки, шар доступу до даних або middleware-шар, відокремлений від презентаційного та працює як самостійний рівень. Саме він реалізовує основну функціональність системи, виконуючи нетривіальну обробку даних та вирішення критично важливих бізнес-завдань. Компоненти цього шару зазвичай розташовуються на сервері, що дозволяє ефективно повторно використовувати їх для різних клієнтів. У такому шарі реалізуються бізнес-правила, а також правила цілісності даних, що забезпечують узгодженість структур у межах однієї або кількох баз. Рівень даних утворюється серверами баз даних і фактично є рівнем доступу до СУБД; звертання до нього відбувається через шар бізнес-логіки, рідше - безпосередньо з шару презентації.

Виокремлення рівня даних робить інформацію незалежною від серверів додатків та бізнес-логіки, підвищує масштабованість і продуктивність системи, дозволяє повторно використовувати ресурси та звільняє клієнтів від необхідності встановлювати на свою сторону СУБД-бібліотеки чи драйвери ODBC. Прикладом такого вузла у роботі є сервер з СУБД MySQL.

На (рис. 1.7) наведено реалізацію клієнт-серверної архітектури у розроблюваній системі.

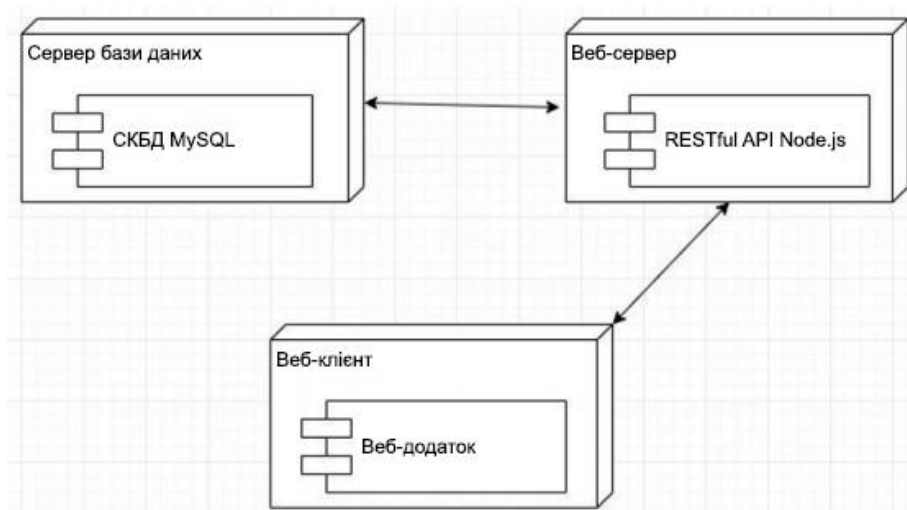
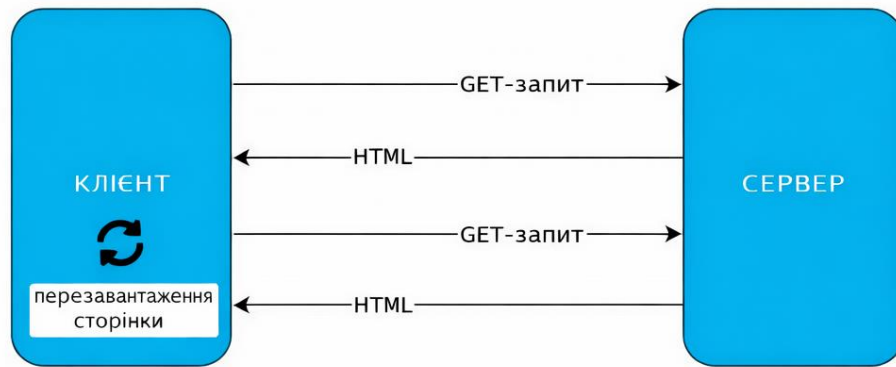


Рис. 1.7 - Архітектура системи

Обраній архітектурі належать висока масштабованість системи, гнучкість супроводу (кожен рівень управляється та масштабується незалежно від інших), а також конфігурованість, що завдяки ізольованості рівнів дозволяє швидко переконфігурувати систему у разі збоїв або під час планового обслуговування. Окремо слід відзначити високий рівень безпеки та надійності, нижчі вимоги до пропускної спроможності каналу між клієнтським терміналом та сервером додатків, а також нижчі вимоги до продуктивності та технічних характеристик клієнтських терміналів, що знижує сукупну вартість розгортання робочих місць у разі масового впровадження. Клієнт-серверної архітектури слід відзначити зростання складності серверної частини та, відповідно, витрат на її адміністрування й супровід, вищу складність розробки додатків, складніше розгортання та адміністрування, а також високі вимоги до продуктивності серверів додатків і бази даних, що збільшує вартість серверного обладнання.

Розроблюваний додаток функціонує за моделлю односторінкового застосунку (SPA, Single Page Application), у межах якої JavaScript-додаток повністю завантажується та виконується в браузері користувача. На відміну від «класичних» сайтів, у SPA рендерінг сторінок відбувається не на сервері, а на стороні клієнта. У браузер завантажується JavaScript-додаток, а контент сторінок підвантажується динамічно через AJAX-запити, без перезавантаження HTML-сторінки. Завдяки такій моделі SPA-додатки відчутно швидше реагують на дії користувача порівняно з традиційними сайтами (рис. 1.8).

Традиційні сайти



SPA-сайти

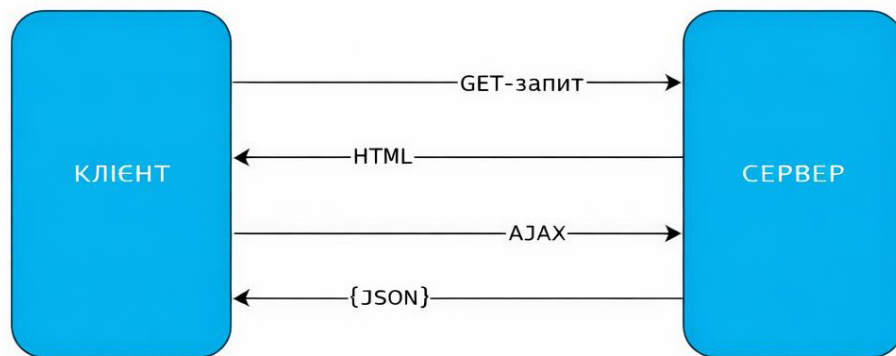


Рис. 1.8 - Схема роботи СПА

Послідовність взаємодії користувача з SPA-сайтом виглядає таким чином. Спочатку користувач звертається до URL сайту та отримує початковий HTML, після чого у браузер завантажується JavaScript-додаток. Додаток самостійно визначає, на яку сторінку зайшов користувач, та який вміст слід відобразити, після чого через AJAX-запити підвантажуються необхідні CSS, JS, HTML та текстові ресурси. На наступному кроці JavaScript-додаток обробляє отримані дані та формує DOM сторінки. При подальшій навігації по сайту перезавантажується не вся сторінка, а лише потрібні фрагменти контенту, що мінімізує мережевий обмін та забезпечує плавний користувацький досвід.

SPA-моделі належать висока швидкість роботи, прискорена розробка та можливість повторного використання коду під різні платформи, а до слабких - те, що JavaScript-контент гірше індексується пошуковими системами, а сам сайт неможливо повноцінно використати з відключеним у браузері JS.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вибір системи керування базами даних для роботи виконано з урахуванням наступних базових вимог: серверна архітектура СУБД, висока надійність, підтримка розмежування прав доступу та можливість одночасної роботи багатьох користувачів. Програмне забезпечення має функціонувати у середовищах ОС Windows та Linux. У (табл. 1.1) наведено результати порівняльного аналізу СУБД MySQL, PostgreSQL та Oracle за основними характеристиками.

Таблиця 1.1

Функціональне порівняння СУБД

Характеристика	MySQL	Oracle	PostgreSQL
Мова SQL	Добре	Відмінно	Відмінно
Швидкість роботи	Відмінно	Відмінно	Задовільно
Простота обслуговування	Відмінно	Добре	Задовільно
Безпека	Відмінно	Добре	Добре
Масштабування	Відмінно	Відмінно	Добре
Повнотекстовий пошук	Добре	Відмінно	Відмінно
Процедури, що зберігаються та тригери	Добре	Відмінно	Відмінно
Надійність	Задовільно	Відмінно	Задовільно
Підтримка транзакцій	Добре	Відмінно	Відмінно
Інтеграція з іншими СУБД	Добре	Добре	Відмінно
Забезпечення цілісності даних	Добре	Відмінно	Відмінно
Графічні інструменти	Відмінно	Відмінно	Відмінно
Підтримка різних платформ	Відмінно	Добре	Добре
Розподілена обробка транзакцій	Відмінно	Відмінно	Відмінно
Організація сховищ даних та підготовка звітів	Відмінно	Відмінно	Відмінно
Засоби завантаження	Добре	Відмінно	Добре
Засоби аналізу	Добре	Відмінно	Добре

Таблиця 1.2

Порівняння СУБД з вимогами ТЗ

Необхідні умови	MySQL	Oracle	PostgreSQL
Підтримка технології клієнт-сервер	+	+	+
Контроль цілісності БД	+	+	+
Наявність графічного інтерфейсу для адміністраторів БД	+	+	+
Імпорт та експорт таблиць БД	+	+	+
Контроль доступу до даних. Автентифікація засобами СУБД	+	+	+
Відлагоджений механізм реплікації даних	+	+	+
Підтримка утиліт резервування даних	+	+	+
Імпорт та експорт таблиць БД	+	+	+

За результатами порівняльного аналізу як цільову СУБД для роботи обрано MySQL [3, 4, 16, 18, 24].

MySQL утримує одну з провідних позицій серед серверних СУБД за популярністю, що забезпечує великий пул готових бібліотек і клієнтських адаптерів для більшості сучасних мов програмування та активне співтовариство користувачів.

До MySQL також належить висока швидкість роботи й широка кросплатформеність - система перенесена на Linux, Windows, FreeBSD, Solaris, SunOS, UnixWare та низку інших платформ.

Гнучкість MySQL забезпечується підтримкою різних типів таблиць - від MyISAM з повнотекстовим пошуком до InnoDB з транзакційною роботою на рівні окремих записів.

Для взаємодії з MySQL існують API для багатьох мов: C/C++, Java, PHP, Lisp, Python, Ruby, Perl та інших, а також бібліотеки для платформи .NET; додатково підтримується робота через ODBC за допомогою драйвера MyODBC.

Основу побудови веб-сторінок у мережі Інтернет становить стандартизована мова розмітки гіпертекстових документів HTML [1, 6, 7, 25] (англ. HyperText Markup Language). Переважна більшість сторінок Всесвітньої павутини створена засобами HTML або його розширеної версії XHTML, для

відтворення документ обробляється браузером і подається користувачеві у звичному графічному вигляді.

До основних можливостей HTML належить формування структурованого документа шляхом семантичної розмітки тексту через заголовки, абзаци, переліки, таблиці та цитати, доступ до інформації у мережі Інтернет за допомогою гіперпосилань, побудова інтерактивних форм для збору даних від користувача, а також вбудовування у документ зображень, відео, аудіо та інших мультимедійних об'єктів.

Для опису візуального представлення документів, написаних мовами розмітки, застосовується спеціалізована мова CSS [1, 11, 25] (Cascading Style Sheets, каскадні таблиці стилів). Найчастіше CSS застосовується для оформлення сторінок HTML та XHTML, проте формат також придатний для будь-яких XML-документів. Розвитком стандартів CSS опікується Консорціум Всесвітньої павутини (W3C). Існують різні рівні специфікації, серед яких CSS1, CSS2 та CSS3, причому кожен наступний з них розширює можливості попередніх. Профілі CSS об'єднують правила різних рівнів для конкретних класів пристроїв чи інтерфейсів - наприклад, для друку, мобільних пристроїв тощо. Поширення CSS дозволило відмовитись від табличної верстки на користь блочної моделі, що дозволяє відокремлювати зміст сторінки від її візуального оформлення.

У ролі мови програмування клієнтської частини веб-додатків широко використовується JavaScript (JS) [12, 14, 15, 25], що належить до групи динамічних об'єктно-орієнтованих мов та є реалізацією стандарту ECMAScript. Найчастіше використовується як скриптовий компонент браузера, що дозволяє коду на стороні клієнта взаємодіяти з користувачем, керувати браузером, обмінюватися даними з сервером в асинхронному режимі та модифікувати DOM веб-сторінки. Поряд з браузерним середовищем JavaScript використовується для розробки серверних додатків (за прикладом Java або C#), створення ігор, мобільних та десктопних застосунків, написання сценаріїв у прикладному ПЗ (наприклад, Adobe Creative Suite), а також у середині PDF-документів.

За парадигмою JavaScript класифікується як прототипна (підмножина

					ІІЗ.КР.Б-121-26-ІІЗ	Арк
						25

об'єктно-орієнтованої) скриптова мова з динамічною типізацією. Окрім прототипного підходу, мова частково підтримує імперативну та функціональну парадигми, а серед її ключових властивостей - динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування і функції як об'єкти першого класу.

Для розробки користувацьких інтерфейсів на основі моделей даних із використанням механізму реактивного зв'язування у роботі застосовується JavaScript-фреймворк Vue.js [13, 19], побудований за патерном MVVM. Vue використовує синтаксис шаблонів, заснований на HTML, що дозволяє декларативно описувати рендерінг DOM з прив'язкою до даних. Будь-який Vue-шаблон є валідним HTML і коректно обробляється браузером та парсерами. Внутрішньо фреймворк компілює шаблони у render-функції віртуального DOM. Поєднання реактивної системи з віртуальним DOM дозволяє Vue точно визначати, які саме компоненти необхідно повторно рендерити, та виконувати мінімально необхідну кількість DOM-операцій при зміні стану додатка.

Vue.js також підтримує запис компонентів безпосередньо у вигляді render-функцій з використанням JSX замість шаблону, що відкриває простір для гнучких патернів проєктування на базі компонентів.

Однією з найхарактерніших особливостей Vue є ненав'язлива реактивна система. Моделі у Vue реалізовано на основі звичайних JavaScript-об'єктів, що робить керування станом простим та інтуїтивним. Vue з коробки забезпечує оптимізований ре-рендеринг без додаткових налаштувань: кожен компонент відстежує свої реактивні залежності та повідомляє системі, коли і які саме частини DOM мають бути оновлені.

Для організації переходів між станами елементів у Vue передбачено різноманітні механізми, серед яких автоматичне застосування CSS-класів для переходів та анімацій, інтеграція сторонніх бібліотек CSS-анімацій (наприклад Animate.css), використання JavaScript для безпосередньої маніпуляції DOM під час переходу, а також інтеграція сторонніх JavaScript-бібліотек анімацій, таких як Velocity.js.

Сам Vue не включає засобів маршрутизації, проте офіційний пакет vue-router закриває цю потребу - він підтримує зв'язування вкладених маршрутів з вкладеними компонентами та надає деталізований контроль над переходами. У підсумку Vue дозволяє будувати складні компонентні додатки, у яких vue-router забезпечує прив'язку компонентів до маршрутів і керує їх відображенням.

Серверне крос-платформне середовище виконання JavaScript-коду з відкритим вихідним кодом Node.js [8, 21] орієнтоване на побудову високопродуктивних мережових застосунків. До появи Node.js мова JavaScript застосовувалася переважно для обробки даних на стороні клієнта у браузері, тоді як Node.js дозволив виконувати JavaScript-сценарії на сервері та формувати відповідь, що передається клієнту. Підхід базується на однопотоковій моделі обробки запитів із подієво-орієнтованою архітектурою, що добре зарекомендувала себе у задачах одночасного обслуговування великої кількості мережових з'єднань [8].

До характерних особливостей платформи належать асинхронна неблокуюча модель введення-виведення, JavaScript-рушій V8 від Google, модульна система CommonJS та пакетний менеджер npm для встановлення і керування залежностями [21]. Node.js застосовується не лише для побудови серверних HTTP-застосунків, але й для створення утилітарних інструментів командного рядка, мережних сервісів та повноцінних серверних або клієнтських програмних продуктів [8]. У межах розроблюваної системи на платформі Node.js реалізовано серверну частину додатку обліку рослин разом з REST-API на основі каркасу Express та ORM-бібліотеки Sequelize.

Поглиблення базової функціональності веб-системи обліку рослин передбачає залучення двох додаткових технологічних напрямків - машинного навчання для автоматичної класифікації зображень рослин та геоінформаційних систем для просторового представлення даних користувачів.

Сучасні згорткові нейронні мережі дозволяють з високою точністю визначати вид рослини за фотографією окремого органу - листа, квітки, плоду чи кори. Серед публічно доступних API виокремлюються три найпоширеніші

рішення. PlantNet API [10, 29] являє собою некомерційний проєкт французького наукового центру CIRAD, база якого охоплює понад сорок п'ять тисяч видів та використовує згорткову модель сімейства Inception, навчену на колекції PlantCLEF; API повертає топ-N кандидатів з оцінкою впевненості, біологічну таксономію (родина, рід, вид) та поширені назви декількома мовами. iNaturalist Computer Vision API є глобальною платформою спостережень з мільйонами фотографій диких рослин і автоматичним географічним сортуванням пропозицій. Plant.id, на відміну від попередніх, є комерційним API з акцентом на діагностику хвороб рослин, для якого також доступний обмежений безкоштовний тариф. Для розроблюваного програмного комплексу обрано PlantNet API, оскільки воно поєднує високу якість розпізнавання, відсутність обмежень на тип некомерційного використання та достатній безкоштовний ліміт у п'ятсот запитів на день.

Для побудови карти користувацьких садів використано бібліотеку Leaflet у комбінації з шаром плиток OpenStreetMap. Leaflet [30] являє собою JavaScript-бібліотеку з відкритим вихідним кодом для побудови інтерактивних мап у веб-середовищі, що підтримує накладання маркерів, кластеризацію точок, спливаючі вікна та обмеження області перегляду заданим географічним регіоном. OpenStreetMap [31] забезпечує безкоштовний шар тайлів без необхідності реєстрації API-ключа, що принципово важливо для дипломної роботи.

Серед альтернативних підходів до зберігання фотографій рослин розглянуто два варіанти - окремі файли на дисковому сховищі сервера або Base64-кодоване подання в полі бази даних типу LONGTEXT. Перший варіант економніший за обсягом дискового простору, проте вимагає узгодженого резервного копіювання БД та файлового сховища. Другий варіант забезпечує атомарність даних та простіше горизонтальне масштабування, що особливо актуально для відносно невеликих за обсягом фотографій рослин (близько ста кілобайт на зображення після стиснення). У роботі обрано Base64-підхід з обов'язковим попереднім стисненням зображень на стороні клієнта до розміру 1280×1280 пікселів і якості JPEG 85%. Додатково створюється зменшена

					ІІЗ.КР.Б-121-26-ІІЗ	Арк
						28

мініатюра 200×200 пікселів, яку завантажують списки рослин; повне фото підвантажується лише при відкритті картки.

Висновки до розділу 1

У межах першого розділу проведено комплексний аналіз предметної області, який дозволив окреслити ключові процеси ведення обліку рослин та сформулювати зміст автоматизації. Опрацювання наявних на ринку аналогів і зіставлення їхніх можливостей з реальними потребами користувачів показало, що серед обов'язкових властивостей майбутньої системи мають бути: інтуїтивно зрозумілий користувацький інтерфейс, чітко структуроване навігаційне меню та висока швидкодія.

На основі порівняльного аналізу архітектурних підходів обґрунтовано доцільність використання клієнт-серверної моделі побудови застосунку та надано перевагу односторінковій (SPA) реалізації клієнтської частини. Як цільову СУБД обрано MySQL, що відповідає висунутим вимогам до надійності, швидкодії та крос-платформеності. Технологічний стек реалізації включає HTML5, CSS3, JavaScript, фреймворк Vue.js разом з компонентною бібліотекою Quasar на стороні клієнта, а також Node.js, каркас Express та ORM Sequelize - на стороні сервера. Окремо обґрунтовано залучення зовнішнього API PlantNet для автоматичного розпізнавання рослин за фотографією та стеку Leaflet+OpenStreetMap для інтерактивного картографічного представлення садів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ОБЛІКУ РОСЛИН

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Розроблювана система автоматизації обліку рослин створюється з метою забезпечення користувачам зручного інструмента для ведення власних колекцій рослин та пошуку рослин у загальному переліку. Перед безпосереднім проєктуванням архітектурних компонентів виконано аналіз вимог користувачів до системи, на основі якого сформовано перелік варіантів використання, а також виокремлено сутності та зв'язки між ними у вигляді об'єктно-орієнтованої структури.

Залежно від ролі користувача у системі визначено два класи учасників - зовнішні користувачі та адміністратори. Зовнішні користувачі мають доступ до авторизації та реєстрації в системі, надсилання відгуків та пропозицій адміністраторам, перегляду контактних даних власників рослин, пошуку рослин у загальному переліку, зміни особистих даних профілю, додавання та редагування записів власних рослин, а також створення нових категорій рослин. Адміністратори системи додатково мають доступ до перегляду повного списку зареєстрованих користувачів, перегляду повідомлень від користувачів, перегляду контактів власників рослин, пошуку рослин у загальному переліку та зміни особистих даних.

За результатами аналізу вимог користувачів сформовано перелік варіантів використання системи обліку рослин, діаграму яких наведено на (рис. 2.1).

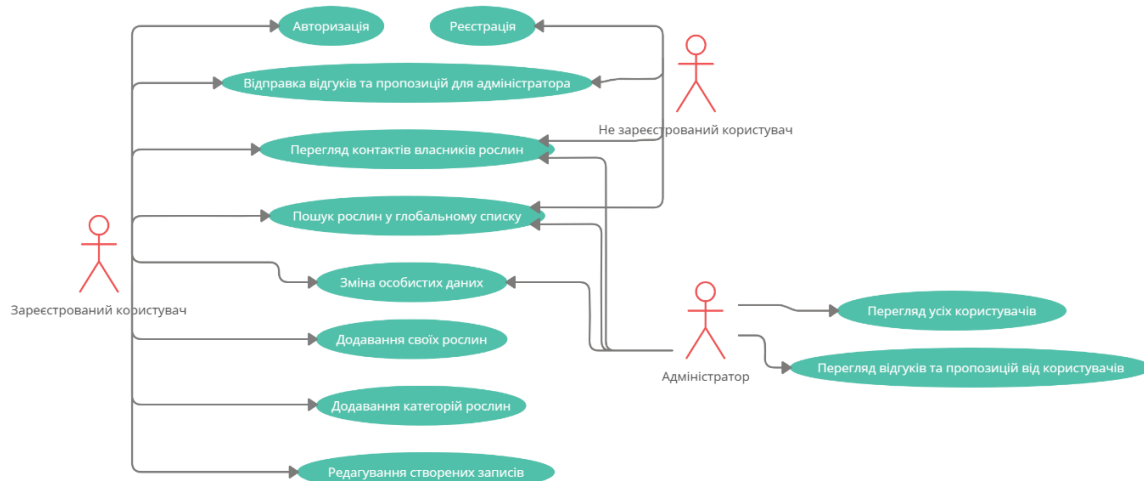


Рис. 2.1 - Варіанти використання

Функціональні вимоги до системи передбачають автентифікацію користувачів та призначення відповідної ролі (адміністратор, користувач, клієнт), ведення обліку рослин у форматі повного циклу управління записами з можливістю їх додавання, редагування та видалення, відображення детальної інформації про рослини, пошук даних у системі та редагування персональних даних користувача. Перелік цих вимог сформовано на основі огляду наявних аналогів та аналізу типових сценаріїв використання системи обліку рослин.

Нефункціональні вимоги до системи групуються за трьома основними категоріями - вимогами до сприйняття, вимогами до надійності та вимогами до продуктивності. У межах вимог до сприйняття встановлено, що середній час, необхідний для опанування основних функцій системи, для пересічного користувача складає до 25 хвилин, а для досвідченого - до 15 хвилин; час відгуку системи на типові запити не перевищує 0,02 секунди, на складні - 0,03 секунди; інтерфейс має бути інтуїтивно зрозумілим і не вимагати від користувача додаткової підготовки. У межах вимог до надійності час, відведений на технічне обслуговування системи, не повинен перевищувати 5 % загального часу роботи; середній час безперервної роботи становить від 10 годин, а максимально допустима норма помилок не перевищує двох помилок на 2000 запитів користувача. Вимоги до продуктивності передбачають здатність системи

безперебійно обслуговувати користувача протягом штатного режиму роботи із збереженням заявлених показників часу відгуку.

За результатами аналізу функціональних вимог визначено основні сутності для програмної реалізації додатку. Об'єктну структуру системи у вигляді діаграми класів наведено на (рис. 2.2). Вона відображає програмні сутності та взаємозв'язки між ними.

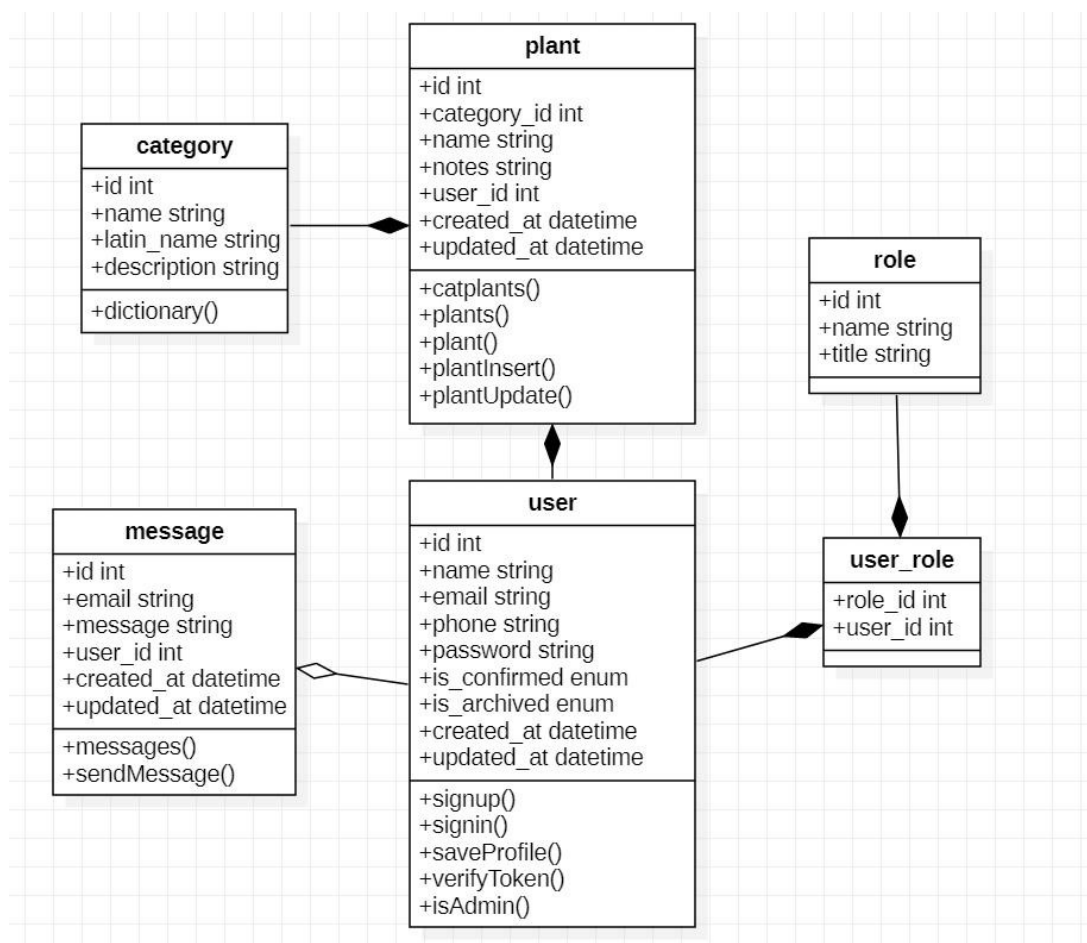


Рис. 2.2 - Діаграма класів

Ключовим компонентом для роботи Sequelize з базою даних є моделі. Моделі описують структуру даних, які зберігаються у таблицях, та зв'язки між таблицями; саме через моделі здійснюється основна взаємодія додатку з базою даних.

Клас Plant забезпечує роботу з таблицею plant, для взаємодії з ним використовуються методи контролера catplants (вибірка всіх рослин за заданою категорією для виведення у блоці пошуку), plants (вибірка списку рослин конкретного користувача), plant (отримання даних окремої рослини за її

ідентифікатором), plantInsert (додавання нового запису рослини користувачем) та plantUpdate (оновлення інформації про вже існуючу рослину).

Клас Category призначений для роботи з таблицею category - взаємодія здійснюється через метод контролера Dictionary, який повертає повний перелік категорій рослин.

Клас Message обслуговує таблицю message; через нього проходять два сценарії - отримання списку усіх повідомлень користувачів через метод messages з доступом, обмеженим роллю admin, та надсилання відгуків і пропозицій адміністраторам системи від користувачів з роллю user через метод sendMessage.

Клас User забезпечує роботу з таблицею user і підтримує методи signup для реєстрації нового користувача в системі, signin для автентифікації, saveProfile для збереження персональних даних профілю, verifyToken для перевірки коректності ідентифікаційного токена та isAdmin для перевірки наявності у користувача прав адміністратора.

Таким чином, описана структура моделей разом із набором методів контролера повністю забезпечує функціонал програмного інтерфейсу веб-додатку обліку рослин.

2.2 Розробка бази даних системи

За результатами проектування до бази даних увійшли такі таблиці: category, plant, role, user, user_role, message. Загальна структура бази даних з відображенням зв'язків між таблицями наведена на (рис. 2.3).

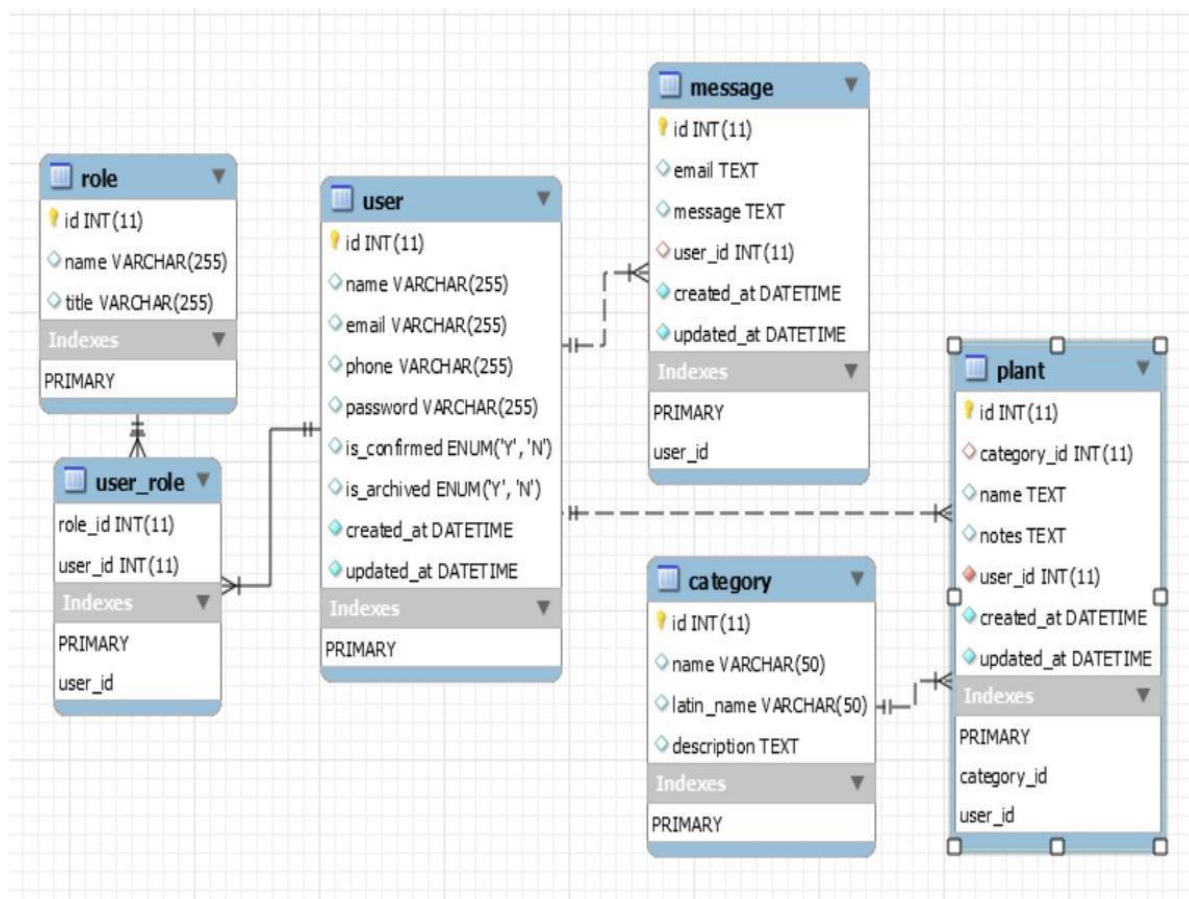


Рис. 2.3 - Структура бази даних

Нижче подано опис основних таблиць бази даних.

Таблиця category (табл. 2.1) використовується для збереження відомостей про категорії рослин: ідентифікатор запису, назва категорії, латинська назва, текстовий опис.

Таблиця 2.1

Структура таблиці category

Поле	Тип даних	PK	FK	Опис
id	INT	+		Ідентифікатор запису
name	VARCHAR(50)			Назва категорії
latin_name	VARCHAR(50)			Латинська назва
description	TEXT			Опис категорії

Таблиця plant (табл. 2.2) зберігає інформацію про окремі рослини: ідентифікатор запису, ідентифікатор категорії, назву рослини, текстові примітки, ідентифікатор власника-користувача, мітки часу створення та останнього оновлення запису.

Таблиця 2.2

Структура таблиці plant

Поле	Тип даних	PK	FK	Опис
id	INT	+		Ідентифікатор запису
category_id	INT	+	+	Ідентифікатор категорії
name	TEXT			Назва рослини
notes	TEXT			Опис категорії
user_id	INT	+	+	Ідентифікатор користувача
created_at	DATETIME			Дата створення запису
updated_at	DATETIME			Дата останнього оновлення запису

Таблиця message (табл. 2.3) використовується для збереження повідомлень, що надходять адміністраторам: ідентифікатор повідомлення, електронна адреса автора (для випадку анонімного звернення), сам текст повідомлення, ідентифікатор користувача (якщо повідомлення надіслане з-під авторизованого облікового запису), а також мітки створення та оновлення.

Таблиця 2.3

Структура таблиці message

Поле	Тип даних	PK	FK	Опис
id	INT	+		Ідентифікатор запису
email	TEXT			Електронна адреса
message	TEXT			Текст повідомлення
user_id	INT	+	+	Ідентифікатор користувача
created_at	DATETIME			Дата створення запису
update_at	DATETIME			Дата останнього оновлення запису

Таблиця role (табл. 2.4) описує перелік ролей у системі та містить ідентифікатор і назву ролі.

Таблиця 2.4

Структура таблиці role

Поле	Тип даних	PK	FK	Опис
id	INT	+		Ідентифікатор запису
name	VARCHAR(50)			Назва ролі

Таблиця user (табл. 2.5) зберігає базові відомості про користувача: ідентифікатор, ім'я, електронну адресу, хеш пароля [27], прапорець підтвердження поштової скриньки, прапорець статусу облікового запису (активний/архівний), мітки часу створення та оновлення запису.

Таблиця 2.5

Структура таблиці user

Поле	Тип даних	PK	FK	Опис
id	INT	+		Ідентифікатор запису
name	VARCHAR(255)			Ім'я користувача
email	VARCHAR(255)			Поштова скринька користувача
password	VARCHAR(255)			Пароль користувача
is_confirmed	ENUM			Підтверджена або ні пошта
is_archived	ENUM			Статус користувача
created_at	DATETIME			Дата створення запису
updated_at	DATETIME			Дата останнього оновлення інформації

Таблиця user_role (табл. 2.6) реалізує зв'язок «багато-до-багатьох» між користувачами та ролями і містить два зовнішні ключі - ідентифікатор користувача та ідентифікатор ролі.

Таблиця 2.6

Структура таблиці user_role

Назва	Тип даних	PK	FK	Опис поля
Role_id	INT		+	Інформація про роль
User_id	INT		+	Інформація про користувача

Таким чином, спроектована база даних відповідає вимогам реляційної моделі та надає програмному додатку повний набір засобів для збереження й вибірки потрібних даних. Загалом база даних складається з шести таблиць.

Розроблену структуру бази даних можна вважати нормалізованою, оскільки усі її таблиці перебувають у третій нормальній формі: жоден неключовий атрибут таблиці не залежить від іншого неключового атрибута, а лише від первинного ключа. Як наслідок, таблиці одночасно задовольняють вимоги першої та другої нормальних форм. Перша нормальна форма забороняє повторювані та множинні стовпці і вимагає визначення первинного ключа для кожної таблиці. Друга нормальна форма передбачає, що неключові стовпці залежать від первинного ключа в цілому, а не від окремих його частин. Третя нормальна форма забороняє наявність транзитивних залежностей між неключовими атрибутами; у спроектованій базі даних ця умова дотримана, оскільки всі залежності виражено через посилання на первинні ключі. Завдяки нормалізованій структурі забезпечується одноразове зберігання кожного факту, що мінімізує надлишковість даних та усуває аномалії оновлення, видалення й вставлення. Окремо реалізовано підтримку посилкової цілісності на рівні зовнішніх ключів `category_id`, `user_id` та `role_id`, що захищає базу від появи висячих посилань на неіснуючі записи.

2.3 Проектування та реалізація алгоритмів роботи системи

Узагальнену схему взаємодії компонентів розроблюваного веб-додатку наведено на (рис. 2.4).

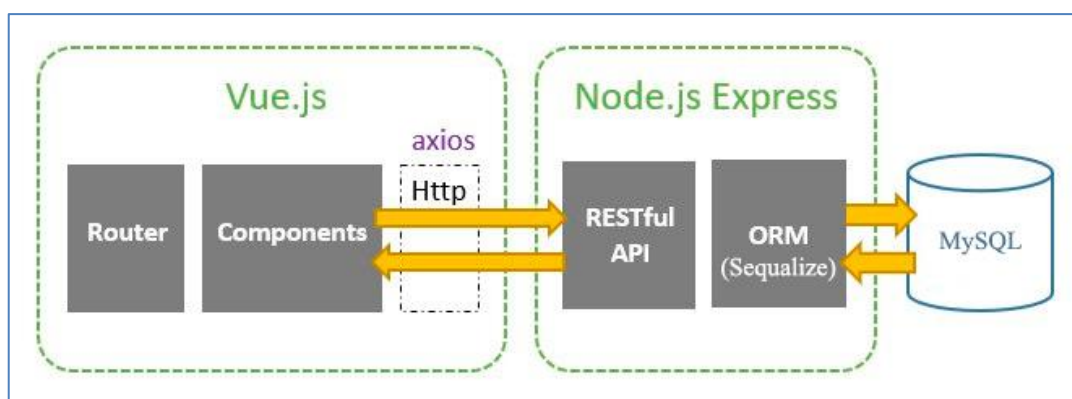


Рис. 2.4 - Архітектура сайту

Клієнтська частина додатку базується на роутерах та компонентах фреймворку Vue.js. Підсистема маршрутизації аналізує URL і перенаправляє користувача на відповідний компонент. Кожен Vue-компонент об'єднує HTML-розмітку, CSS-стилі та JavaScript-логіку; для обміну даними із серверною частиною компоненти використовують HTTP-клієнт axios, через який звертаються до RESTful API.

Серверна частина побудована на платформі Node.js із використанням каркаса Express. Її основою є RESTful API [22], яке обробляє вхідні запити та повертає клієнту необхідні дані у форматі JSON. Для взаємодії з базою даних застосовано ORM-бібліотеку Sequelize, яка дозволяє абстрагуватись від ручного формування SQL-запитів і працювати з даними у термінах моделей. Загальний цикл обробки запиту має такий вигляд: запит з клієнтського додатку потрапляє на серверний роутер, який передає його відповідному методу контролера; контролер, у свою чергу, звертається до моделей (моделі описують таблиці бази даних та зв'язки між ними), отримує потрібні дані та повертає їх клієнту у вигляді JSON-відповіді.

Алгоритм автентифікації користувача побудовано на основі стандарту JSON Web Token. На вході система отримує електронну адресу та пароль користувача, після чого виконує пошук відповідного запису у таблиці user; у разі знаходження користувача система звіряє хеш-значення пароля (отримане з використанням бібліотеки bcrypt) із збереженим у базі даних. У випадку успішної перевірки сервер формує JWT-токен, який містить ідентифікатор користувача та має обмежений термін дії, після чого повертає його клієнту разом з основними полями профілю. Клієнт зберігає отриманий токен у локальному сховищі браузера та підставляє його у заголовок x-access-token для кожного наступного запиту до серверного API. На стороні сервера для кожного захищеного ендпоінту виконується перевірка отриманого токена через спеціалізоване проміжне програмне забезпечення; у разі відсутності або некоректності токена сервер повертає клієнту HTTP-відповідь зі статусом 401 та відповідним повідомленням про помилку автентифікації. Для розмежування

					ІІЗ.КР.Б-121-26-ІІЗ	Арк
						38

доступу до адміністративних функцій додатково реалізовано перевірку наявності у користувача ролі admin, що зберігається у зв'язувальній таблиці user_role та запитується через метод getRoles об'єктної моделі.

Алгоритм реєстрації нового користувача починається з валідації вхідних даних форми, у межах якої перевіряється унікальність електронної адреси у системі та коректність формату обов'язкових полів - імені, прізвища, номера телефону, міста та паролю. У разі успішної валідації сервер виконує хешування пароля через bcrypt, створює новий запис у таблиці user із статусом активного облікового запису та автоматично призначає роль user через створення відповідного запису у таблиці user_role. Після завершення реєстрації серверна частина повертає клієнту повідомлення про успішне створення запису, а клієнт автоматично переходить до форми автентифікації.

Алгоритм додавання нової рослини охоплює послідовну обробку даних на стороні клієнта та сервера. На стороні клієнта перевіряється заповненість обов'язкових полів - назви, категорії та (за наявності) фотографії - після чого компонент завантаження фото виконує валідацію та стиснення прикріпленого зображення до робочих розмірів повної версії 1280×1280 пікселів та мініатюри 200×200 пікселів. Підготовлений об'єкт надсилається на серверний ендпоінт plantSave, який залежно від наявності ідентифікатора запису диспетчеризує обробку на сценарій додавання або сценарій оновлення. У сценарії додавання сервер виконує операцію findOrCreateCategory, що шукає категорію без врахування реєстру в локальній базі та створює нову лише за відсутності збігу; у сценарії оновлення додатково перевіряється приналежність запису поточному користувачу. Після успішного збереження сервер повертає клієнту ідентифікатор створеного або оновленого запису.

Алгоритм автоматичного розпізнавання рослини активується після натискання користувачем кнопки «Розпізнати рослину» у формі заповнення картки рослини. Клієнт надсилає до серверного ендпоінту identifyPlant запит з прикріпленим Base64-кодованим зображенням; сервер декодує отриману картинку, формує multipart/form-data запит до зовнішнього сервісу PlantNet API

[29] та очікує його відповіді. Отриманий перелік кандидатів проходить через локальний словник перекладу родових і видових назв на українську побутову назву, після чого сервер повертає клієнту топ-3 кандидати з оцінкою впевненості, науковою та поширеною назвою, біологічною таксономією та ідентифікатором локальної категорії або пропонованою новою назвою. Користувач у клієнтському компоненті обирає один з варіантів - відповідні поля картки рослини заповнюються автоматично, з можливістю подальшого редагування вручну.

Алгоритм формування JSON-відповіді для відображення карти користувачів базується на запиті до ендпоінту `usersMap`, який повертає всіх неархівованих користувачів із заповненими координатами разом з вкладеним переліком належних їм рослин. Для зменшення обсягу мережевого обміну поле `photo` з результатів запиту виключається, а у відповідь включаються лише поля `photo_thumb` (мініатюра) разом з необхідними для побудови маркерів метаданими. Клієнтська частина обробляє отриманий JSON та формує маркери на основі бібліотеки `Leaflet`, додаючи невелике зміщення координат для маркерів у межах одного міста, щоб уникнути їх накладання.

2.4 Реалізація додатку системи обліку рослин

Підключення до бази даних організовано через файл оточення `env.js`, який зчитує конфіденційні параметри з системних змінних оточення (Лістинг 2.1). Така схема дозволяє вести розробку та промислову експлуатацію з різними параметрами підключення, не вносячи змін у вихідний код. Бібліотека `dotenv` забезпечує автоматичне підвантаження значень змінних з файлу `.env` у поточне середовище виконання `Node.js`. У складі параметрів передаються адреса хоста СУБД, ім'я бази даних, обліковий запис, пароль, діалект SQL, а також налаштування пулу з'єднань (мінімальна та максимальна кількість одночасних з'єднань, час очікування під час їх отримання й вивільнення).

Лістинг 2.1 - Файл env.js (модуль конфігурації середовища)

```
const env = {  
  database: process.env.DB_NAME,  
  username: process.env.DB_USER,  
  password: process.env.DB_PASSWORD,  
  host: process.env.DB_HOST,  
  dialect: 'mysql',  
  pool: {  
    max: 5,  
    min: 0,  
    acquire: 30000,  
    idle: 10000  
  },  
  plantnetApiKey: process.env.PLANTNET_API_KEY || "  
}  
  
module.exports = env
```

Подальша ініціалізація з'єднання з базою даних виконується в модулі db.config.js (Лістинг 2.2). На вхід передаються параметри з модуля env.js, після чого створюється екземпляр Sequelize з налаштованим пулом з'єднань. Каталог app/model сканується автоматично, кожен файл моделі імпортується через sequelize.import та реєструється в об'єкті db; якщо в моделі визначено статичний метод associate, він викликається після завершення сканування для встановлення зв'язків між моделями. Окремо налаштовується багатозначний зв'язок «багато-до-багатьох» між таблицями user та role через зв'язувальну таблицю user_role, що дозволяє одному користувачу мати кілька ролей у системі.

Лістинг 2.2 - Фрагмент db.config.js (ініціалізація Sequelize та реєстрація моделей)

```
const sequelize = new Sequelize(env.database, env.username, env.password, {  
  logging: false,  
  host: env.host,
```

```

    dialect: env.dialect,
    pool: env.pool
  })
  const db = {}
  const modelPath = path.join(global.appRoot, 'app/model')
  fs.readdirSync(modelPath).filter((f) => f.slice(-3) === '.js').forEach((file) => {
    const model = sequelize.import(path.join(modelPath, file))
    db[model.name] = model
  })
  Object.keys(db).forEach((m) => { if (db[m].associate) db[m].associate(db) })
  db.role.belongsToMany(db.user, { through: 'user_role', foreignKey: 'role_id',
otherKey: 'user_id', timestamps: false })
  db.user.belongsToMany(db.role, { through: 'user_role', foreignKey: 'user_id',
otherKey: 'role_id' })
  module.exports = db

```

Модель Category (Лістинг 2.3) визначає структуру таблиці category, до якої входять цілочисельний ідентифікатор, назва українською, латинська назва та текстовий опис категорії. Поле id оголошено як автоматично інкрементований первинний ключ цілого типу INTEGER(11) без знаку. Параметр timestamps:false вимикає автоматичне додавання технічних полів createdAt/updatedAt, оскільки для категорій вони не потрібні; параметр tableName фіксує точну назву таблиці у базі даних незалежно від назви моделі у внутрішньому об'єкті db.

Лістинг 2.3 - Модель Category (файл category.js)

```

module.exports = (sequelize, DataTypes) => {
  class Category extends sequelize.constructor.Model {}
  Category.init({
    id: {
      type: DataTypes.INTEGER(11).UNSIGNED,
      autoIncrement: true,
      primaryKey: true,

```

```

        allowNull: false
      },
      name: DataTypes.STRING(50),
      latin_name: DataTypes.STRING(50),
      description: DataTypes.TEXT()
    }, {
      sequelize,
      modelName: 'category',
      tableName: 'category',
      timestamps: false
    })
    return Category
  }

```

Модель Plant (Лістинг 2.4) описує основну сутність системи - рослину. Структура містить такі поля: ідентифікатор, ідентифікатор категорії, назву, наукову назву, текстові примітки, повна фотографія та її мініатюра у форматі Base64, ідентифікатор користувача-власника. Метод `associate` налаштовує зв'язки `belongsTo` з таблицями `category` (через `category_id`) та `user` (через `user_id`), а аліаси `Category` та `User` дозволяють включати ці моделі при формуванні відповіді клієнту через опцію `include`. Параметр `timestamps:true` з відповідними іменами `createdAt/updatedAt` забезпечує автоматичне заповнення міток часу при кожній операції з записом.

Лістинг 2.4 - Модель Plant (файл `plant.model.js`)

```

module.exports = (sequelize, DataTypes) => {
  class Plant extends sequelize.constructor.Model {
    static associate(db) {
      this.belongsTo(db.category, { foreignKey: 'category_id', as: 'Category' })
      this.belongsTo(db.user, { foreignKey: 'user_id', as: 'User' })
    }
  }
}

```

```

Plant.init({
  id: { type: DataTypes.INTEGER(11).UNSIGNED, autoIncrement: true,
primaryKey: true },
  category_id: DataTypes.INTEGER(11).UNSIGNED,
  name: DataTypes.TEXT,
  notes: DataTypes.TEXT,
  photo: DataTypes.TEXT('long'),
  photo_thumb: DataTypes.TEXT('long'),
  scientific_name: DataTypes.STRING(255),
  user_id: { type: DataTypes.INTEGER(11).UNSIGNED, allowNull: false }
}, { sequelize, modelName: 'plant', tableName: 'plant', timestamps: true,
  createdAt: 'created_at', updatedAt: 'updated_at' })
return Plant
}

```

Модель User (Лістинг 2.5) описує облікові записи користувачів системи. Окрім базових полів імені, поштової адреси, телефону та паролю, модель містить географічні поля city, lat, lng, що використовуються для побудови інтерактивної карти користувачів. Прапорці is_confirmed та is_archived типу ENUM зі значеннями 'Y'/'N' використовуються для розмежування активних, підтверджених та архівних облікових записів. Асоціація hasMany(plant) з аліасом Plants дозволяє отримати весь перелік належних користувачу рослин у одному запиті за допомогою опції include.

Лістинг 2.5 - Модель User (файл user.model.js)

```

module.exports = (sequelize, DataTypes) => {
  class User extends sequelize.constructor.Model {
    static associate(db) {
      this.hasMany(db.plant, { foreignKey: 'user_id', as: 'Plants' })
    }
  }
  User.init({

```

```

    id: { type: DataTypes.INTEGER(11).UNSIGNED, autoIncrement: true,
primaryKey: true },
    name: DataTypes.STRING,
    email: DataTypes.STRING,
    phone: DataTypes.STRING,
    city: DataTypes.STRING(100),
    lat: DataTypes.DECIMAL(9, 6),
    lng: DataTypes.DECIMAL(9, 6),
    password: DataTypes.STRING,
    is_confirmed: { type: DataTypes.ENUM, values: ['Y','N'], defaultValue: 'N' },
    is_archived: { type: DataTypes.ENUM, values: ['Y','N'], defaultValue: 'N' }
  }, { sequelize, modelName: 'user', tableName: 'user', timestamps: true,
    createdAt: 'created_at', updatedAt: 'updated_at' })
  return User
}

```

Робота з моделями організована через єдиний модуль controller.js, який містить набір обробників-методів для кожного ендпоінту REST API. Як приклад типового сценарію взаємодії з моделями та формування відповіді клієнту на (Лістинг 2.6) наведено метод signup, що реалізує процес реєстрації нового користувача. Метод спочатку створює запис у таблиці user, попередньо хешуючи пароль через бібліотеку bcrypt з вісьмома раундами хешування, після чого знаходить роль user у довіднику ролей та призначає її новоствореному користувачу через метод setRoles. Після цього формується JWT-токен з ідентифікатором користувача та терміном дії 24 години, який повертається клієнту як поле accessToken. Усі помилки обробляються через catch-блоки з відповідним HTTP-статусом 500 та описом помилки в тілі відповіді.

Лістинг 2.6 - Метод signup модуля controller.js (реєстрація користувача)

```

exports.signup = (req, res) => {
  db.user.create({
    name: req.body.name,

```

```

email: req.body.email,
phone: req.body.phone,
city: req.body.city || null,
lat: req.body.lat || null,
lng: req.body.lng || null,
password: bcrypt.hashSync(req.body.password, 8)
}).then((user) => {
  db.role.findAll({
    where: { name: { [Op.or]: ['user'] } }
  }).then((roles) => {
    user.setRoles(roles).then(() => {
      const token = jwt.sign({ id: user.id }, config.secret, { expiresIn: 86400 })
      res.status(200).send({ accessToken: token })
    })
  }).catch((err) => res.status(500).send(`Error: ${err}`))
}).catch((err) => res.status(500).send(`Fail! Error: ${err}`))
}

```

Наведений приклад методу `signup` ілюструє загальний підхід до побудови інших методів контролера: вхідні дані витягуються з `req.body`, потім послідовно виконуються операції з моделями через ланцюжок `then/catch`, а кінцевий результат повертається клієнту через об'єкт `res`. Аналогічну структуру мають методи `signin` (автентифікація), `saveProfile` (збереження профілю), `plantSave` (додавання/оновлення рослини), `identifyPlant` (виклик зовнішнього сервісу PlantNet) та інші обробники, що разом утворюють повний REST API розроблюваної системи.

Кожен запис рослини має два графічних поля: повне фото (`photo`) та мініатюра (`photo_thumb`), обидва типу `LONGTEXT`, що зберігають Base64-кодоване зображення у форматі `data URL`. Розділення на повне фото та мініатюру дозволяє оптимізувати завантаження списків рослин: серверні контролери `plants` та `catplants` явно виключають важке поле `photo` з відповіді через опцію

attributes.exclude, повертаючи клієнту лише photo_thumb обсягом близько тридцяти кілобайт. Повне фото підвантажується лише при відкритті картки конкретної рослини.

Клієнтська обробка зображень реалізована як окремий Vue-компонент PhotoUploader. Компонент приймає файл від користувача, виконує валідацію (тип JPEG або PNG, мінімальний розмір 256×256 пікселів, максимальний обсяг вхідного файлу 10 МБ) та паралельно стискає зображення через бібліотеку browser-image-compression у двох режимах: повне фото - обмежене за більшою стороною 1280 пікселями з цільовим обсягом не більше 0,7 МБ; мініатюра - 200×200 пікселів з цільовим обсягом 0,05 МБ. Обидва результати кодуються у Base64 та через v-model передаються до батьківського компонента форми рослини.

Серверна частина приймає об'єкт через ендпоінт POST /api/plantSave і зберігає поля photo, photo_thumb та scientific_name у відповідних колонках таблиці plant. Розмір тіла HTTP-запиту обмежено п'ятьма мегабайтами на рівні body-parser, що з достатнім запасом покриває обсяг навіть для нестиснених повних фото з мобільних пристроїв. EXIF-орієнтація фотографій з телефонних камер обробляється бібліотекою стиснення автоматично, що усуває типову проблему перевернутих знімків.

Підсистема розпізнавання інтегрує веб-додаток із зовнішнім API PlantNet та забезпечує автоматичне визначення виду рослини за фотографією. Серверна реалізація винесена у окремий модуль app/services/plantnet.js, що відповідає за формування multipart/form-data запиту до PlantNet, парсинг відповіді та перетворення біологічної таксономії на локальну модель категорій. Звернення до зовнішнього API виконується через бібліотеку axios [28] з тайм-аутом тридцять секунд.

Локальний словник GENUS_UA забезпечує переклад наукової назви роду рослини на українську побутову назву, наприклад: Hippophae → Обліпиха, Cornus → Кизил, Magnolia → Магнолія, Hibiscus → Гібіскус. Для родів, що містять кілька видів з відмінними українськими назвами, застосовується словник

SPECIES-UA з парами «вид → назва»: *Elaeagnus multiflora* → Гумі, тоді як *Elaeagnus angustifolia* → Дика маслина. У разі, коли локальний словник не містить мапінгу, система пропонує створити нову категорію з назвою латинського роду; ця категорія автоматично додається до бази при першому виборі користувачем.

Контролер identifyPlant приймає фото у форматі Base64 від клієнтського компонента, передає буфер до сервісу plantnet.identify та повертає клієнту топ-3 кандидати з полями: оцінка впевненості (від нуля до одиниці), наукова назва, поширена назва українською, родина, рід, ідентифікатор існуючої локальної категорії або пропонується нова назва. Користувач у компоненті MyInfo обирає один із запропонованих варіантів - відповідні поля картки рослини («Назва», «Категорія», «Наукова назва») заповнюються автоматично, з можливістю подальшого редагування вручну. Ендпоінт захищено JWT-автентифікацією [26] - викликати розпізнавання може лише авторизований користувач, що запобігає вичерпанню денного ліміту PlantNet анонімними запитами.

З метою візуалізації географічного розподілу користувачів-садівників та їхніх колекцій реалізовано окрему інтерактивну карту України. Підсистема використовує безкоштовний стек Leaflet та OpenStreetMap, що не потребує реєстрації API-ключа.

Геоприв'язка реалізована на рівні користувача: при реєстрації користувач обирає своє місто зі списку ста п'ятдесяти найбільших населених пунктів України (front/src/data/ukraine-cities.json), кожному з яких заздалегідь відповідають координати у форматі decimal degrees. Поля city, lat, lng у таблиці user обов'язкові - без міста реєстрація неможлива.

На стороні сервера ендпоінт GET /api/usersMap повертає JSON-масив усіх користувачів з полями id, name, city, lat, lng, доповнений вкладеним списком належних користувачу рослин (id, name, категорія). Для встановлення зв'язку між таблицями user та plants до моделі User додано асоціацію hasMany(Plant) з аліасом Plants. Поле photo до карти не передається - попап показує лише назви

рослин з гіперпосиланнями на повну картку, що утримує обсяг JSON-відповіді в межах декількох десятків кілобайт навіть при великій кількості користувачів.

Клієнтський компонент PlantMap ініціалізує карту через виклик `L.map()` з обмеженням географічної області координатами 44.0 та 22.0 на південному заході та 52.5 і 40.5 на північному сході, що відповідає приблизним межам території України. Шар плиток підвантажується з `tile.openstreetmap.org`. Маркери розміщуються у точках кожного користувача; для уникнення накладення кількох садівників в одному місті застосовано легке зміщення координат другого та наступних маркерів. При натисканні на маркер відкривається попап з ім'ям користувача, містом, кількістю рослин та переліком назв, кожна з яких є гіперпосиланням на сторінку картки конкретної рослини. Для зручності користувача додано обмеження масштабу та параметр `autoPan`, що автоматично зміщує карту так, щоб попап завжди повністю поміщався у видиму область

Висновки до розділу 2

У межах другого розділу проаналізовано структуру та вимоги до даних, з якими оперує система обліку рослин, визначено варіанти використання та функціональні і нефункціональні вимоги, спроектовано загальний алгоритм роботи додатку та структуру бази даних.

Завдяки побудові діаграми класів та схеми послідовності виокремлено ключові об'єкти системи й зв'язки між ними. Це дозволило збалансовано оцінити структуру програмного комплексу та заздалегідь усунути потенційні архітектурні проблеми, що могли б виникнути під час реалізації.

Спроектовано та реалізовано три ключові підсистеми, що відрізняють розроблюваний додаток від базового CRUD-аналога: підсистему завантаження та зберігання фото рослин з двома рівнями деталізації (повне фото та мініатюра), підсистему автоматичного розпізнавання рослин на основі інтеграції з PlantNet API, а також підсистему геоінформаційного представлення на базі стеку Leaflet + OpenStreetMap. Кожна з підсистем має окремий бізнес-модуль на стороні

сервера, відповідні Vue-компоненти на стороні клієнта та власний REST-ендпоінт.

Описано логіку роботи системи обліку рослин з акцентом на її ключових сценаріях. Окрім того, представлено фрагменти програмного коду, що демонструють основні етапи реалізації функціонала.

взаємодіє з ним через браузер. Для роботи з веб-додатком користувачеві потрібні лише браузер, що підтримує сучасні веб-стандарти, та доступ до мережі Інтернет; зберігання даних та основну обробку запитів виконує віддалений сервер (рис. 3.2).

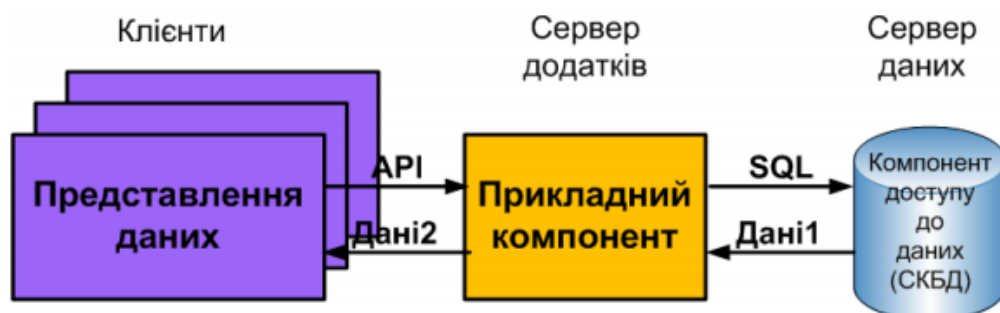


Рис. 3.2 - Схема роботи веб-додатку

Принциповою перевагою такої моделі є незалежність від операційної системи кінцевого пристрою - і інтерфейсом, і операційним середовищем для веб-додатку є браузер. Тому користувач має можливість працювати з тим самим додатком і на офісному стаціонарному комп'ютері, і на ноутбуці, і на планшеті чи смартфоні без жодних змін.

Уся програмна логіка зосереджена в одному місці (на сервері), а до користувача надходить лише користувацький інтерфейс - невелика частина функціональності, представлена через браузер. Завдяки цьому процес розгортання нової версії додатку гранично спрощений: достатньо оновити серверну копію, і всі користувачі автоматично отримують актуальну версію без додаткових дій з їхнього боку.

Для користувача робота з веб-додатком зведена до відкриття браузера та переходу за відповідним URL, а жодне налаштування, оновлення чи технічне обслуговування з його боку не потрібне. Усі ці завдання покладено на розробника та адміністратора системи.

З погляду використання обчислювальних ресурсів ще однією перевагою є те, що користувачеві не потрібно завантажувати додаток повністю - у браузері відображаються лише ті фрагменти інтерфейсу, що стосуються поточного сценарію роботи. Завдяки цьому веб-додатки запускаються швидко й оперативно

реагують на дії користувача навіть при невисокій швидкості мережевого з'єднання.

Розгортання серверної частини системи виконується адміністратором у кілька послідовних кроків. На першому етапі на хост-машині встановлюються необхідні системні залежності - середовище виконання Node.js версії 18 та сервер бази даних MySQL версії 8.0 або сумісна реляційна СУБД (наприклад, MariaDB). На етапі підготовки оточення в кореневому каталозі серверної частини створюється файл `.env`, у якому вказуються параметри підключення до бази даних (адреса хоста, назва бази, ім'я користувача та пароль), порт прослуховування серверного процесу, а також секретні значення для підпису JWT-токенів і API-ключ зовнішнього сервісу PlantNet. Для виробничого середовища додатково передбачено окремий файл `.env.production`, який автоматично підключається при встановленні змінної `NODE_ENV` у значення `production`. Після підготовки конфігурації виконується команда `npm install`, що завантажує всі визначені у `package.json` залежності серверного додатку, після чого через виклик команди `node server.js` або `npm run dev` (у разі використання утиліти `nodemon` для розробки) запускається серверний процес.

При першому запуску ORM-бібліотека Sequelize виконує операцію синхронізації структури бази даних, яка створює всі таблиці згідно з визначеними у каталозі `app/model` моделями та формує зв'язки між ними. У режимі первинного заповнення бази даних адміністратор тимчасово перемикає прапорець синхронізації у режим перебудови (через змінну середовища `DB_FORCE_SYNC`), завдяки чому модуль `app/config/db.initialize.js` створює базовий набір ролей, користувача-адміністратора, набір категорій рослин та демонстраційні записи. Після завершення первинної ініціалізації прапорець повертається у стандартне положення, що запобігає повторному видаленню даних при наступних запусках сервера.

Розгортання клієнтської частини виконується у кілька кроків. У кореневому каталозі клієнтської частини виконується команда `npm install` для встановлення залежностей, після чого через виклик команди `quasar build`

збирається оптимізована версія додатку у форматі односторінкового застосунку. Зібрані статичні файли (HTML, CSS, JavaScript-бандл, шрифти, піктограми та зображення) розміщуються у каталозі dist/spa і готові до публікації веб-сервером загального призначення. У типовій виробничій конфігурації перед серверною частиною додатково розгортається проксі-сервер nginx, який обслуговує статичні файли клієнтської частини та перенаправляє запити з префіксом /api на серверну частину додатку. Така конфігурація дозволяє ефективно використовувати кешування браузера для статичних ресурсів та одночасно забезпечує єдину точку входу для запитів до API. Додатковим перевагою такої схеми є можливість підключення HTTPS-шифрування на рівні nginx з використанням сертифікатів від безкоштовного центру сертифікації Let's Encrypt, що принципово важливо для дотримання сучасних вимог до безпеки веб-додатків.

3.2 Структура інтерфейсу

Після відкриття сайту неавторизований користувач потрапляє на головну сторінку (рис. 3.3). У верхній частині сторінки розташовано основну панель навігації з логотипом системи та посиланнями на ключові розділи, у правому куті якої знаходиться кнопка «Увійти», що відкриває форму входу. Центральну частину займає сторінка «Про проєкт» з коротким описом призначення додатку, переліком основних можливостей та закликом до реєстрації. У режимі неавторизованого користувача навігаційне меню містить обмежений набір вкладок: «Про проєкт» з описом призначення додатку, «Пошук» для глобального перегляду колекцій рослин інших користувачів та «Зворотний зв'язок» для надсилання повідомлення адміністраторам системи. Вкладки «Мої рослини», «Карта» та інші, що вимагають авторизації, у цьому режимі недоступні.

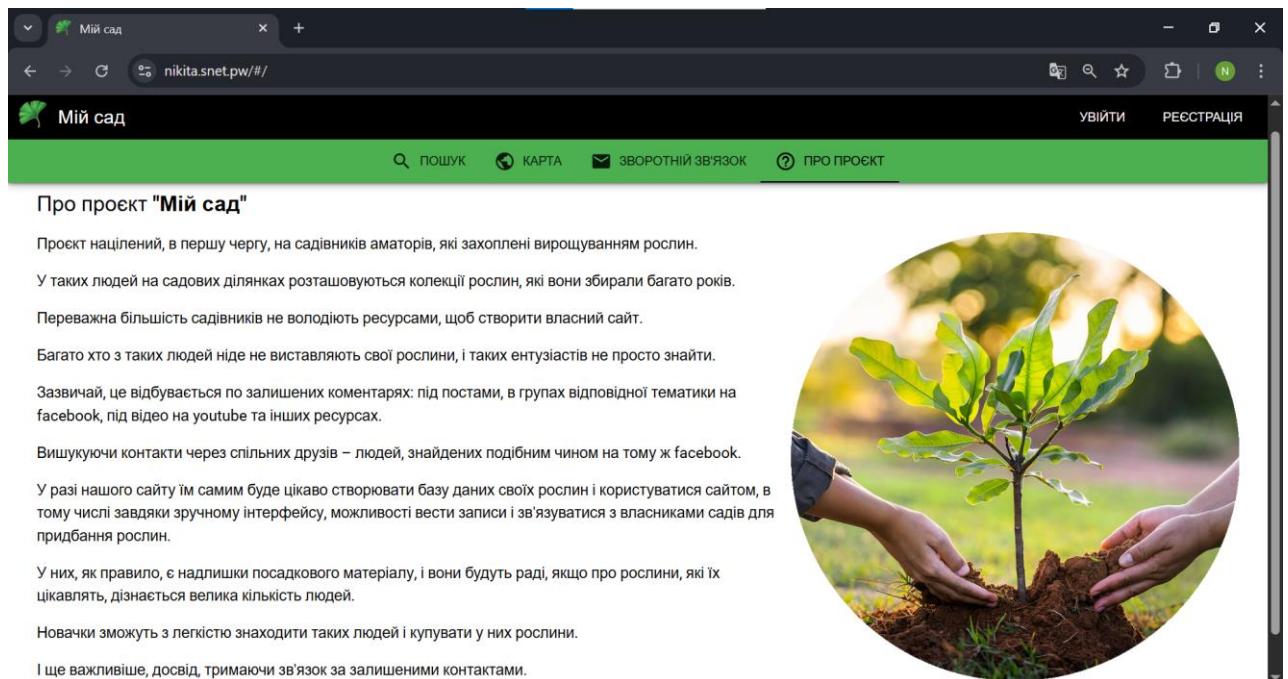


Рис. 3.3. Інтерфейс не автентифікованого користувача на головній сторінці

Для отримання повного набору можливостей користувачу необхідно зареєструватись у системі (рис. 3.4) та пройти процедуру автентифікації (рис. 3.5). Форма реєстрації побудована як єдина екранна анкета, до якої входять поля для імені та прізвища користувача, електронної поштової адреси, контактного номера телефону у міжнародному форматі, паролю з повтором та обов'язкового поля «Місто» з випадаючим списком ста п'ятдесяти найбільших населених пунктів України. Поряд з кожним полем розташовано підказку щодо очікуваного формату даних, а помилки валідації виводяться під відповідним полем червоним кольором без перезавантаження сторінки. Кнопка «Зареєструватися» стає активною лише після того, як усі обов'язкові поля заповнено та пройдено валідацію.

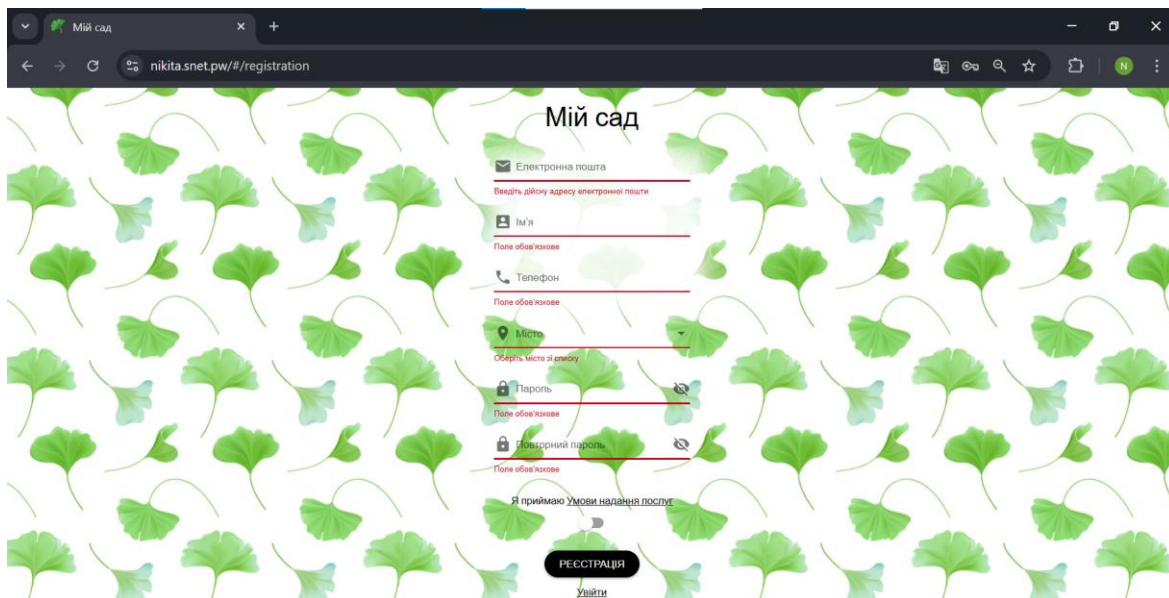


Рис. 3.4 - Інтерфейс користувача для проходження процедури реєстрації

Форма автентифікації містить два поля - електронна адреса та пароль - а також кнопку «Увійти» та посилання «Створити обліковий запис», що повертає користувача до форми реєстрації. У разі введення невірному паролю система виводить повідомлення про помилку без вказання, чи існує користувач з відповідною адресою, що відповідає поширеним практикам безпеки. При успішній автентифікації клієнт отримує JWT-токен і автоматично перенаправляється на головну сторінку зареєстрованого користувача.

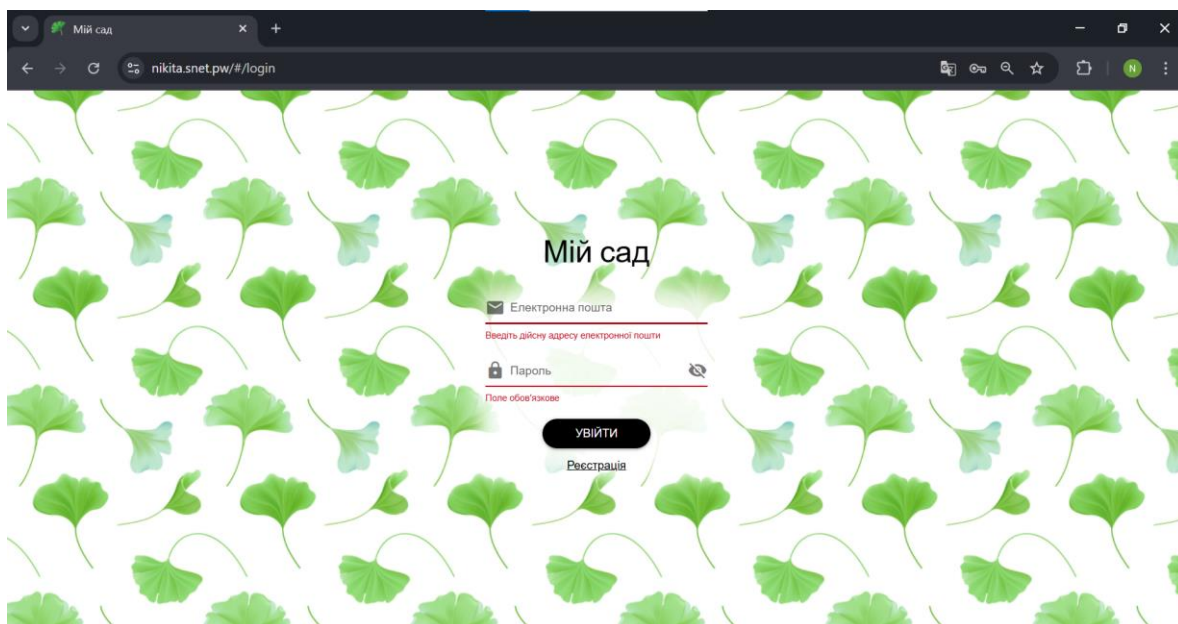


Рис. 3.5 - Інтерфейс користувача для проходження процедури автентифікації

Після успішної автентифікації користувач переходить на головну сторінку зареєстрованого користувача (рис. 3.6), яка надає швидкий доступ до всіх ключових функцій системи. У верхній частині сторінки відображається ім'я поточного користувача та кнопка «Вийти», а ліве бокове меню розширюється новими вкладками - «Мої рослини», «Карта», «Профіль» - що стали доступними після входу в систему. Центральна область сторінки відображає вітальне повідомлення та коротку статистику особистої колекції користувача: кількість записів, кількість категорій та посилання на нещодавно додані рослини.

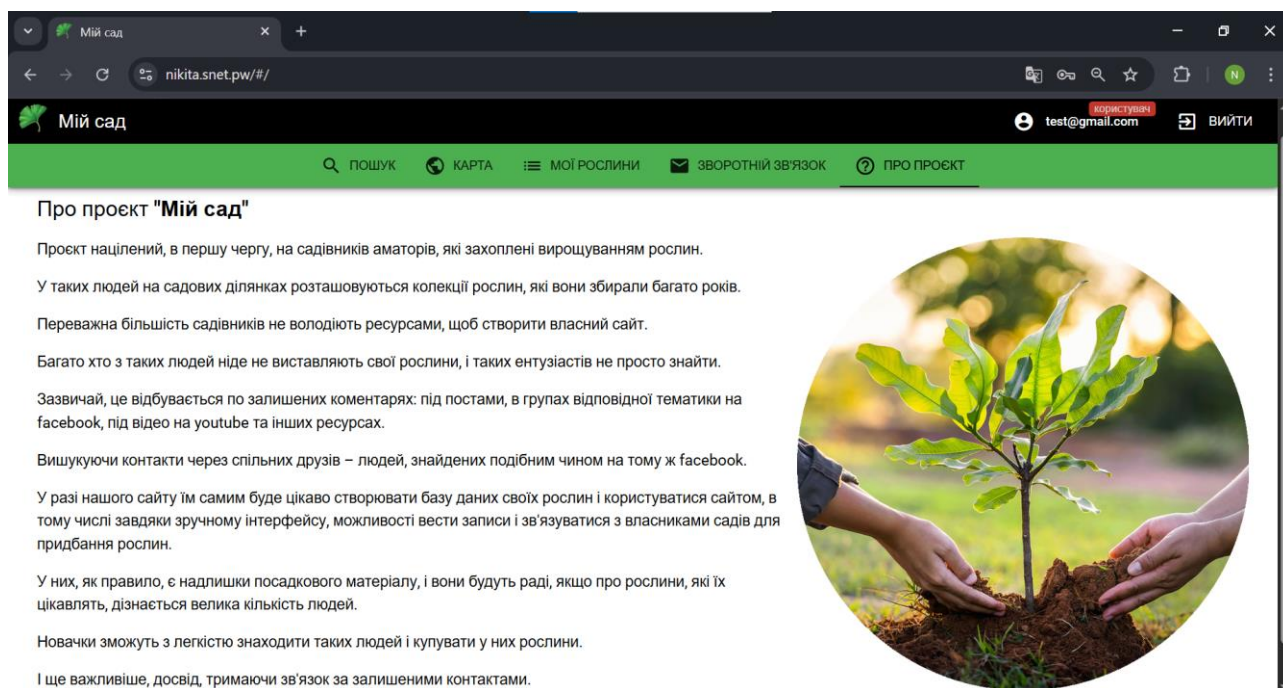


Рис. 3.6 - Інтерфейс автентифікованого користувача системи

Адміністратор системи має додаткові можливості, до яких належить перегляд списку всіх зареєстрованих користувачів, перегляд повідомлень від користувачів та пошук рослин у загальному переліку (рис. 3.7). В інтерфейсі адміністратора у бокове навігаційне меню додано дві спеціальні вкладки - «Користувачі» та «Повідомлення», що недоступні звичайним користувачам системи. Сторінка користувачів відображає таблицю з полями електронної адреси, імені, контактного телефону, дати реєстрації, дати останнього оновлення, з можливістю фільтрації та сортування за будь-яким полем. Кожен запис у таблиці клікабельний - натискання на електронну адресу відкриває поштовий клієнт зі сформованим листом до користувача.

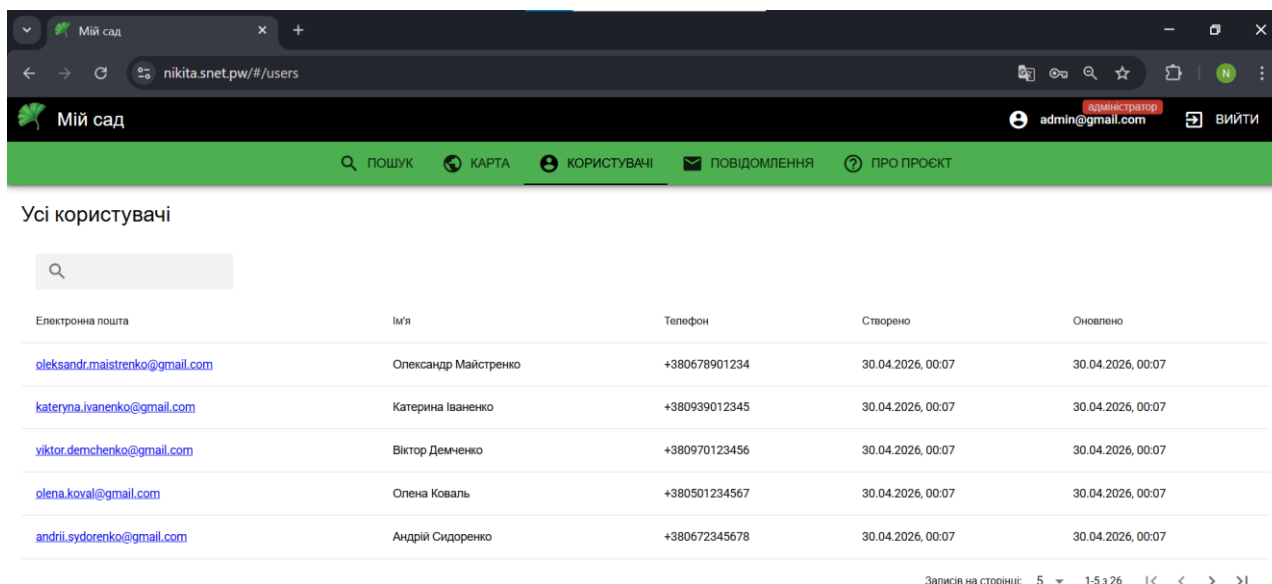


Рис. 3.7 - Інтерфейс адміністратора

Для додавання нової рослини (рис. 3.8) користувач використовує кнопку «Додати рослину» на вкладці «Мої рослини» бокового навігаційного меню. Форма заповнення нового запису побудована таким чином, щоб користувач міг швидко й зручно ввести всі необхідні дані. У верхній частині форми розташовано компонент завантаження фотографії з кнопками «Обрати фото» та «Видалити фото», нижче - кнопка «Розпізнати рослину» з іконкою чарівних зірочок. Далі йдуть поля «Категорія» (випадаючий список з кнопкою «+» для створення нової), «Назва» з валідацією на мінімальну довжину три символи, «Наукова назва (латина)» та багаторядкове поле «Примітки» для довільного опису. Кнопки «Зберегти» та «Скинути» розташовано у верхньому правому куті форми.

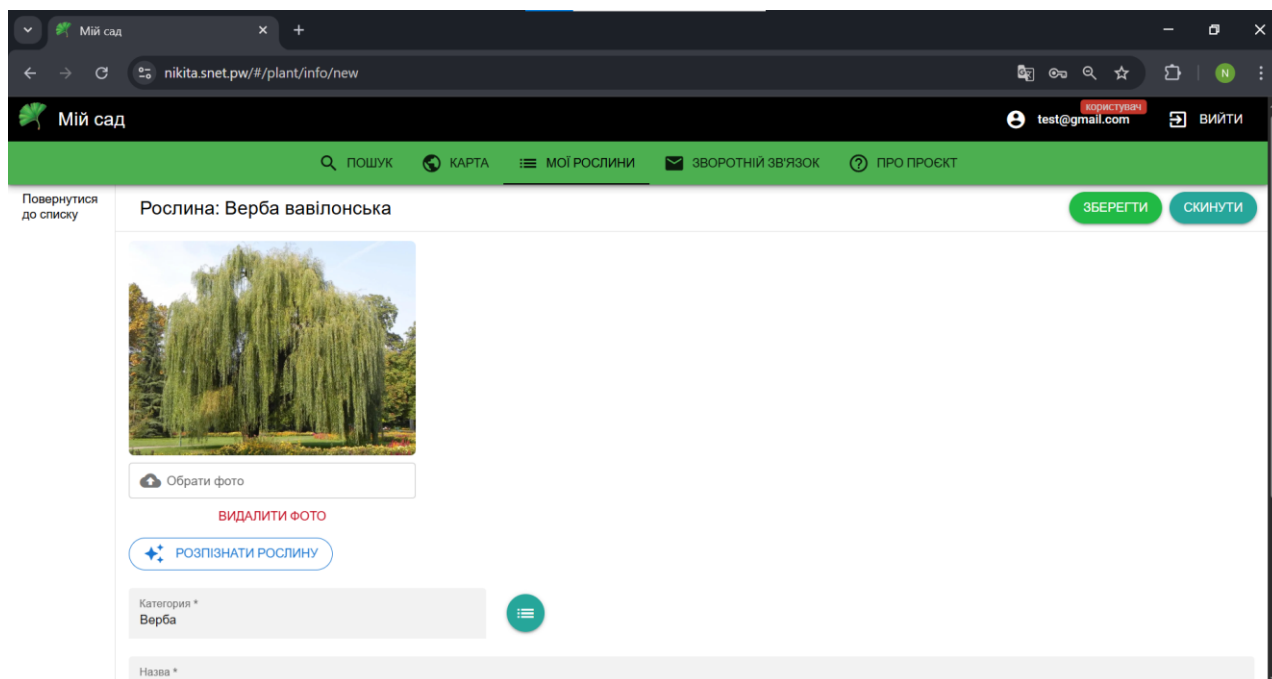


Рис. 3.8 - Сторінка для додавання нового запису

При створенні нового запису рослину можна додати до однієї з наявних категорій (рис. 3.9) або створити нову категорію. Поле «Категорія» реалізовано як компонент q-select бібліотеки Quasar з можливістю пошуку за частковим збігом по літерах назви категорії: при початку введення в поле список звужується до варіантів, що містять введений текст. Порядок категорій у списку відповідає внутрішньому ідентифікатору в таблиці category, але відображення йде українською назвою категорії, що зручніше для користувача.

У структурі форми додавання рослини, окрім стандартних полів назви та опису, передбачено компонент завантаження фотографії, що дозволяє прикріпити до картки рослини зображення з пристрою користувача або зробити знімок камерою на мобільному. Поряд з компонентом завантаження розташовано кнопку «Розпізнати рослину», яка ініціює виклик зовнішнього сервісу машинного навчання та автоматичне заповнення полів картки на основі результатів розпізнавання. Зразок зовнішнього вигляду форми з прикріпленим фото та активною кнопкою розпізнавання представлено на (рис. 3.22). На прев'ю фото видно фактично завантажене зображення зі збереженим співвідношенням сторін, а кнопка «Розпізнати рослину» оформлена з рамкою синього кольору, що привертає увагу користувача.

Після натискання кнопки система передає фотографію до сервісу PlantNet та відображає у модальному діалозі топ-3 найбільш імовірних кандидатів з відповідними оцінками впевненості (рис. 3.23). Кожен запис у діалозі містить наукову (латинську) назву виду, поширену українську назву та посилання на існуючу або пропоновану нову категорію; вибір користувачем потрібного варіанту автоматично заповнює поля «Назва», «Категорія» та «Наукова назва» картки рослини. Оцінка впевненості відображається у вигляді синього кола з відсотками: чим вища впевненість моделі, тим вищий пріоритет розміщення кандидата у списку. У нижній частині діалогу виводиться кількість запитів, що залишилися на поточну добу в межах безкоштовного ліміту PlantNet API.

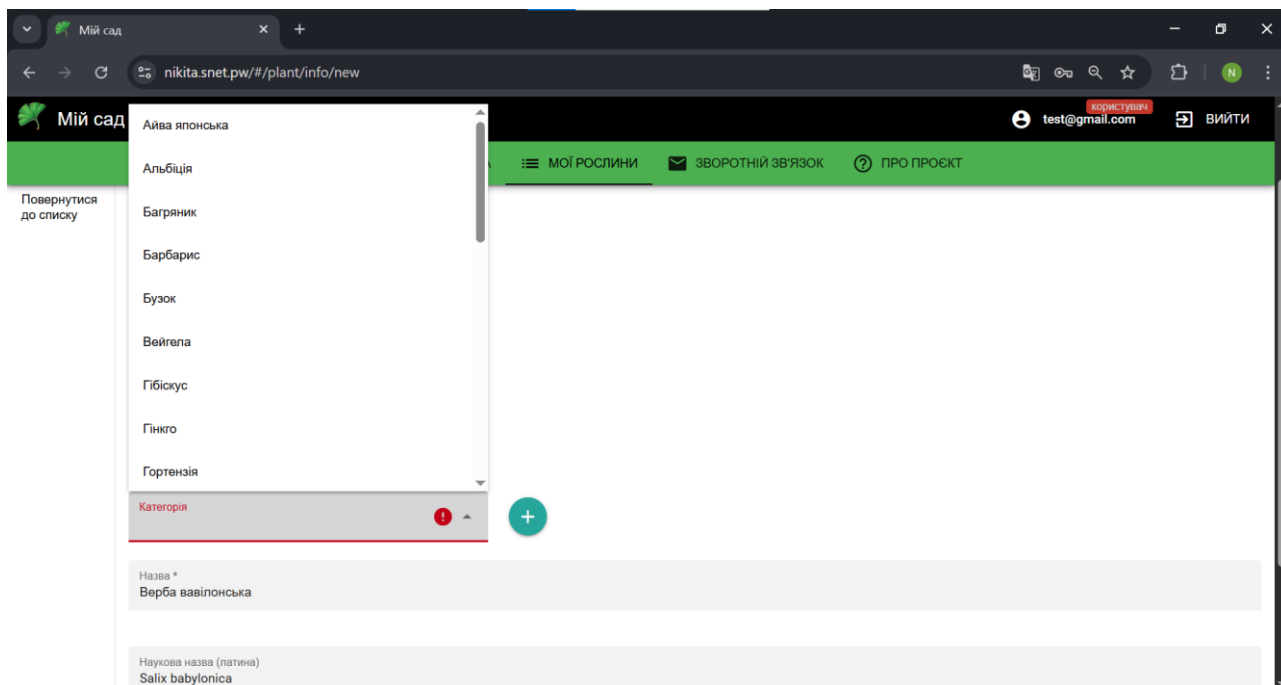


Рис 3.9 - Поле для вибору категорії в яку потрібно додати рослину

Для створення нової категорії передбачено окрему кнопку «+» поряд із полем вибору категорії (рис. 3.10). Натискання на цю кнопку перемикає поле «Категорія» зі списку у режим введення тексту, у якому користувач може ввести довільну назву нової категорії - наприклад, «Магнолія» або «Кедр атласький». Поряд з полем активною стає кнопка повернення до списку (іконка «list») на випадок, якщо користувач передумав і хоче обрати існуючу категорію. Введена назва проходить валідацію на мінімальну довжину три символи, і нова категорія створюється у базі даних лише в момент збереження картки рослини.

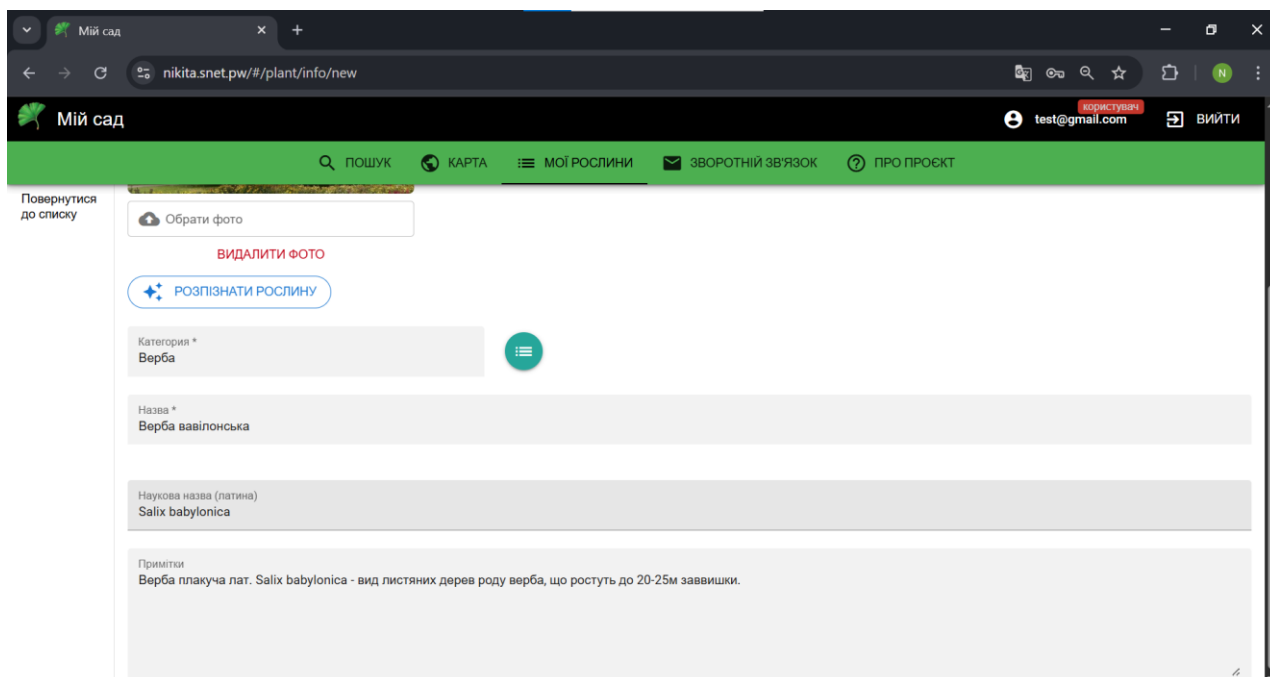


Рис. 3.10 - Поле для створення категорії

Після збереження запису користувач має змогу перейти на сторінку його редагування (рис. 3.11). У режимі перегляду всі поля відображаються в режимі «тільки для читання» з підсвіченим фоном, а у верхньому правому куті розташовано кнопку «Редагувати», яка перемикає форму у режим редагування з кнопками «Зберегти», «Скасувати» та «Видалити». Кнопка «Видалити» відкриває діалогове вікно з підтвердженням операції, що запобігає випадковому видаленню записів; після підтвердження видалення запис повністю прибирається з бази даних, а якщо це був останній запис у відповідній категорії, видаляється і сама категорія. Для пошуку рослин у загальному переліку слугує окрема вкладка «Пошук» головного навігаційного меню.

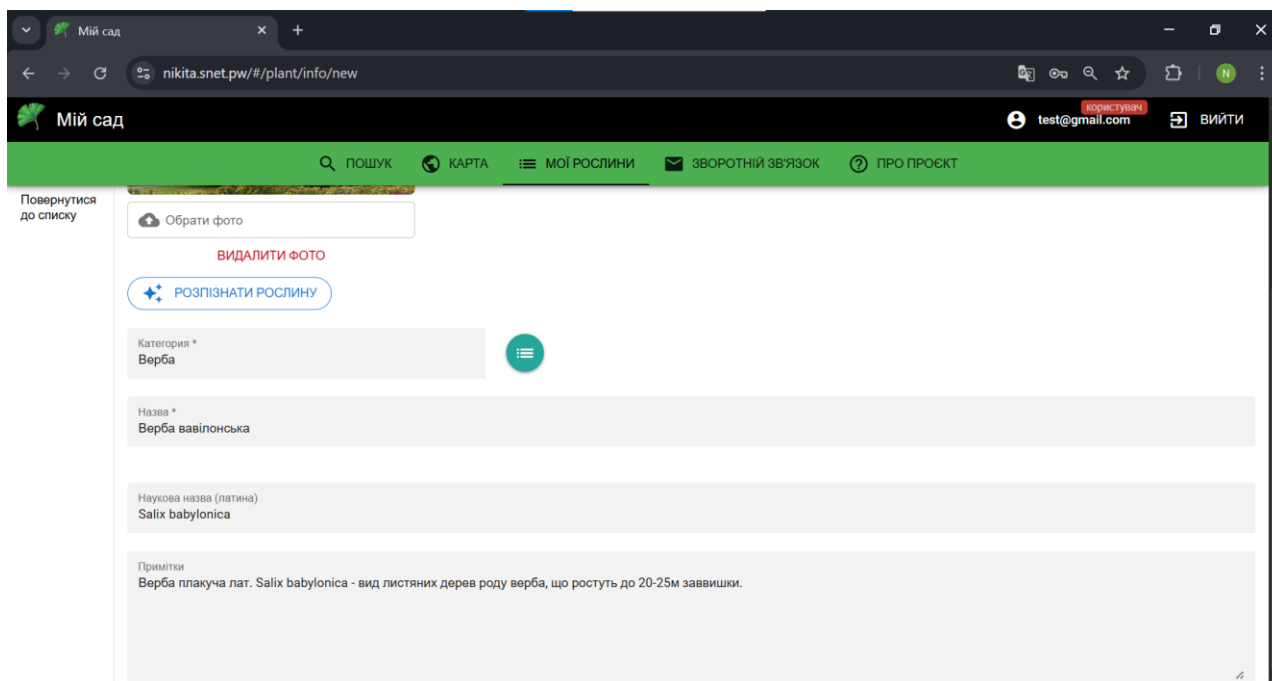


Рис. 3.11 - Сторінка для редагування запису.

Через вкладку «Карта», розташовану в головному навігаційному меню, користувач відкриває інтерактивну мапу України з маркерами всіх зареєстрованих користувачів. Загальний вигляд карти представлено на (рис. 3.24). Карта побудована на основі бібліотеки Leaflet з шаром тайлів OpenStreetMap та обмежена географічними межами України, що запобігає зміщенню маркерів за межі релевантної області перегляду. Над картою відображається статистика з двома чіпами - кількість зареєстрованих садів та загальна кількість рослин у системі. У правому нижньому куті карти автоматично виводиться текст ліцензії OpenStreetMap відповідно до умов використання картографічних даних.

При натисканні на маркер відкривається спливаюче вікно з ім'ям садівника, назвою його міста, кількістю рослин у колекції та переліком назв рослин у форматі гіперпосилань. Натиснення на ім'я садівника відкриває поштовий клієнт зі сформованим листом на електронну адресу власника, а натиснення на назву рослини веде на її повну картку. Приклад спливаючого вікна показано на (рис. 3.25). Структура попапа складається з трьох змістових блоків: ім'я садівника жирним шрифтом у вигляді mailto-посилання, рядок з містом та загальною кількістю рослин маленьким сірим шрифтом, а нижче маркований перелік

рослин з назвами рослин та назвою категорій у дужках.

Важливою частиною системи є зворотний зв'язок із розробниками. Користувач має змогу надсилати свої відгуки про роботу додатку, повідомляти про виявлені недоліки та пропонувати напрямки удосконалення; вікно для зворотного зв'язку наведено на (рис. 3.12). Форма зворотного зв'язку містить два поля - електронна адреса (автоматично підставляється для авторизованих користувачів) та багаторядкове поле повідомлення, нижче розташовано кнопку «Надіслати». Усі надіслані повідомлення зберігаються в таблиці message бази даних з прив'язкою до ідентифікатора користувача (якщо автор авторизований) та доступні адміністратору системи через окрему вкладку «Повідомлення».

Рис. 3.12 - Сторінка для зворотного зв'язку.

3.3 Тестування роботи

Розроблений веб-додаток пройшов комплексне тестування, у межах якого виконано перевірку коректності гіперпосилань, перевірку відповідності текстового наповнення заданих тематиці, перевірку відповідності функціональності індивідуальному завданню, перевірку кросбраузерності, оцінку зрозумілості інтерфейсу та коректності введення даних, а також послідовну перевірку ключових сценаріїв роботи з основним функціоналом -

					ІІЗ.КР.Б-121-26-ІІЗ	Арк
						63

ведення обліку рослин, глобального пошуку, автоматичного розпізнавання рослин за фотографією, відображення інтерактивної карти садів та керування персональним профілем.

За результатами перевірки інформаційного наповнення підтверджено повну відповідність текстового матеріалу заявленій тематиці; орфографічних, стилістичних чи пунктуаційних помилок не виявлено. Тестування на юзабіліті показало, що сайт зручний у використанні, має просте та інтуїтивно зрозуміле меню, приємну кольорову гаму та логічну структуру навігації.

За Кросбраузерне тестування проведено у трьох найпоширеніших браузерях - Google Chrome, Mozilla Firefox та Microsoft Edge. Першим перевірено браузер Google Chrome як найбільш популярний на ринку; результат перевірки головної сторінки наведено на (рис. 3.13). На скріншоті видно, що сторінка повністю відображається без горизонтальної прокрутки, шрифти виводяться у відповідності до заданих CSS-стилів, а навігаційне меню та інтерактивні елементи реагують на дії користувача без затримок. У ході тестування у Chrome додатково перевірено роботу динамічних компонентів - модальних діалогових вікон, випадаючих списків та компонента карти на основі Leaflet.

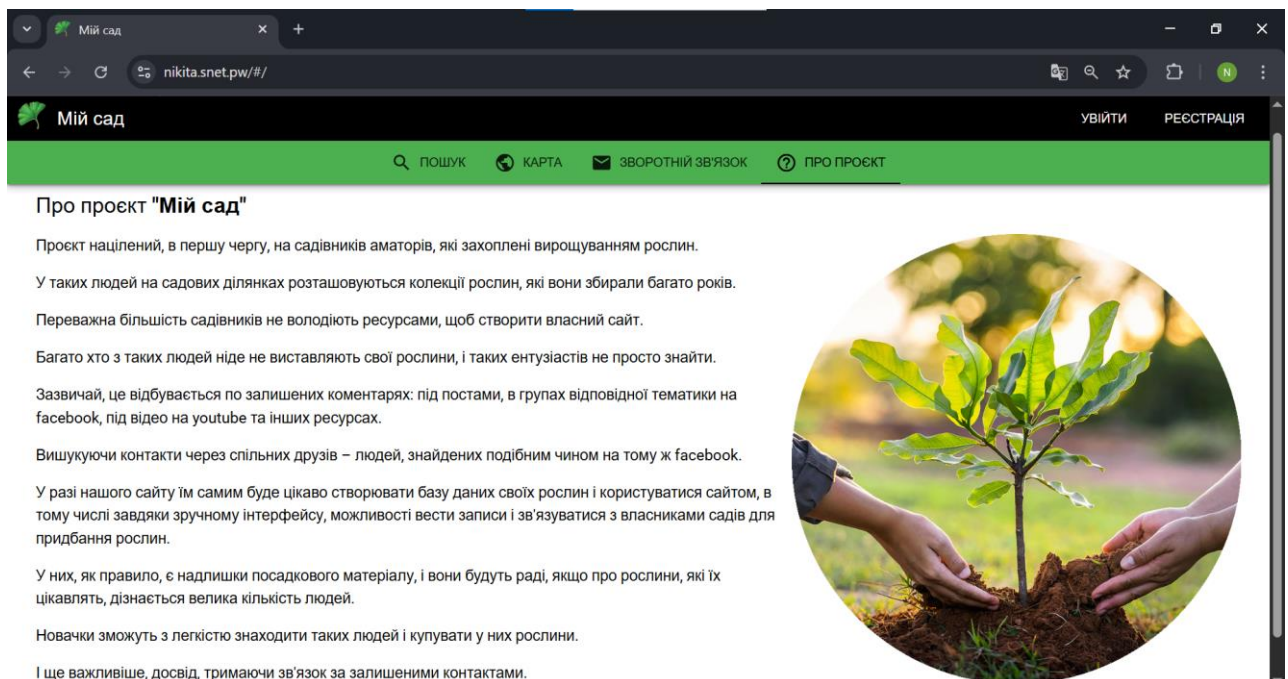


Рис. 3.13 - Тестування головної сторінки в браузері Google Chrome

Аналогічну перевірку у браузері Mozilla Firefox наведено на (рис. 3.14) - жодних відхилень у відображенні чи в роботі інтерактивних елементів не зафіксовано. У Firefox перевірено ту саму послідовність дій користувача, що і в Chrome: відкриття головної сторінки, перехід до форми авторизації, перегляд глобального переліку рослин та відкриття інтерактивної карти. Усі елементи відображаються коректно, а час реакції на дії користувача залишається в межах прийнятних значень.

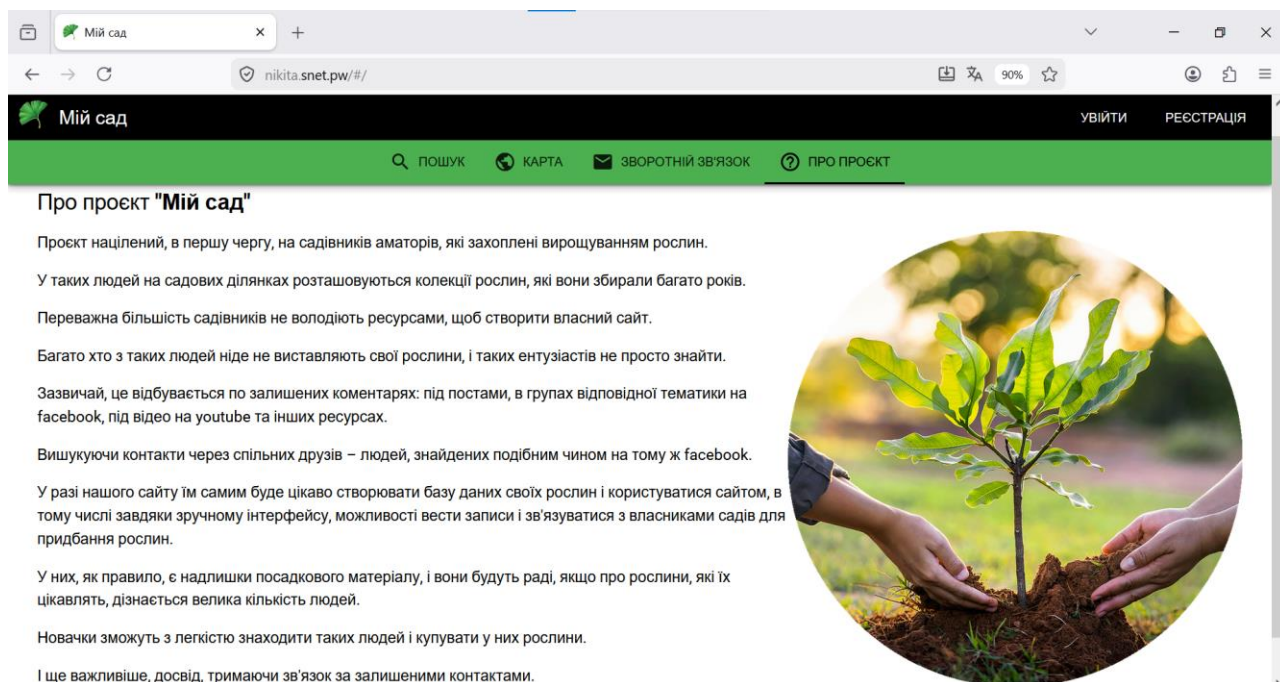


Рис. 3.14 - Тестування головної сторінки в браузері Mozilla Firefox

Перевірка у браузері Microsoft Edge (рис. 3.15) також не виявила помилок чи некоректностей. На скріншоті видно ідентичне з попередніми браузерами відображення головної сторінки, що підтверджує крос-браузерну сумісність розробленої системи. У всіх трьох браузерах сторінки відображаються однаково, а інтерактивні елементи реагують на дії користувача однаково швидко, що свідчить про коректне використання стандартних HTML5/CSS3/JavaScript-технологій без застосування браузер-специфічних розширень.

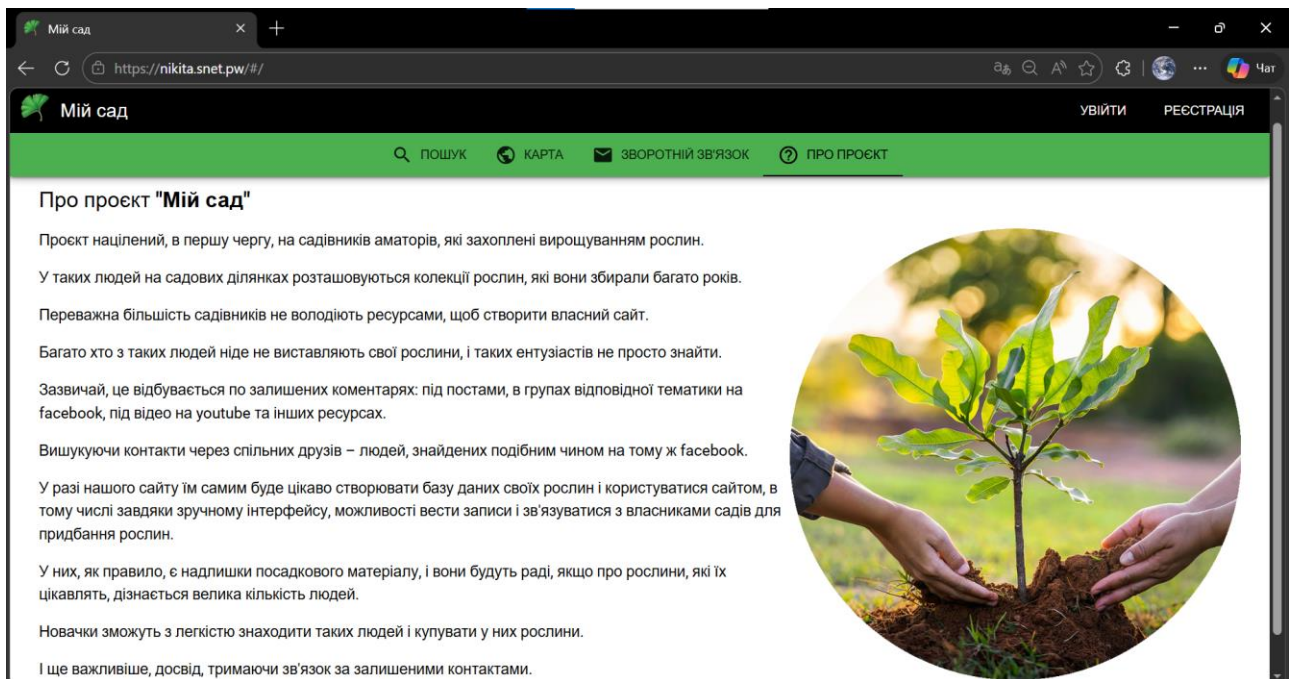


Рис. 3.15 - Тестування головної сторінки в браузері Microsoft Edge

Окремий блок тестових сценаріїв стосувався перевірки коректності модуля автентифікації та авторизації. У межах цього блоку виконано позитивні сценарії (введення правильних облікових даних, отримання токена доступу, перехід на сторінку авторизованого користувача) та негативні сценарії (введення неправильного пароля, спроба доступу до закритої сторінки без токена, спроба використання простроченого токена). У разі введення некоректних облікових даних система повторно показує користувачеві форму для введення (рис. 3.16) до моменту введення правильних логіна та пароля зареєстрованого облікового запису, при цьому відображається відповідне повідомлення про помилку без розкриття деталей того, чи існує користувач з відповідною адресою.

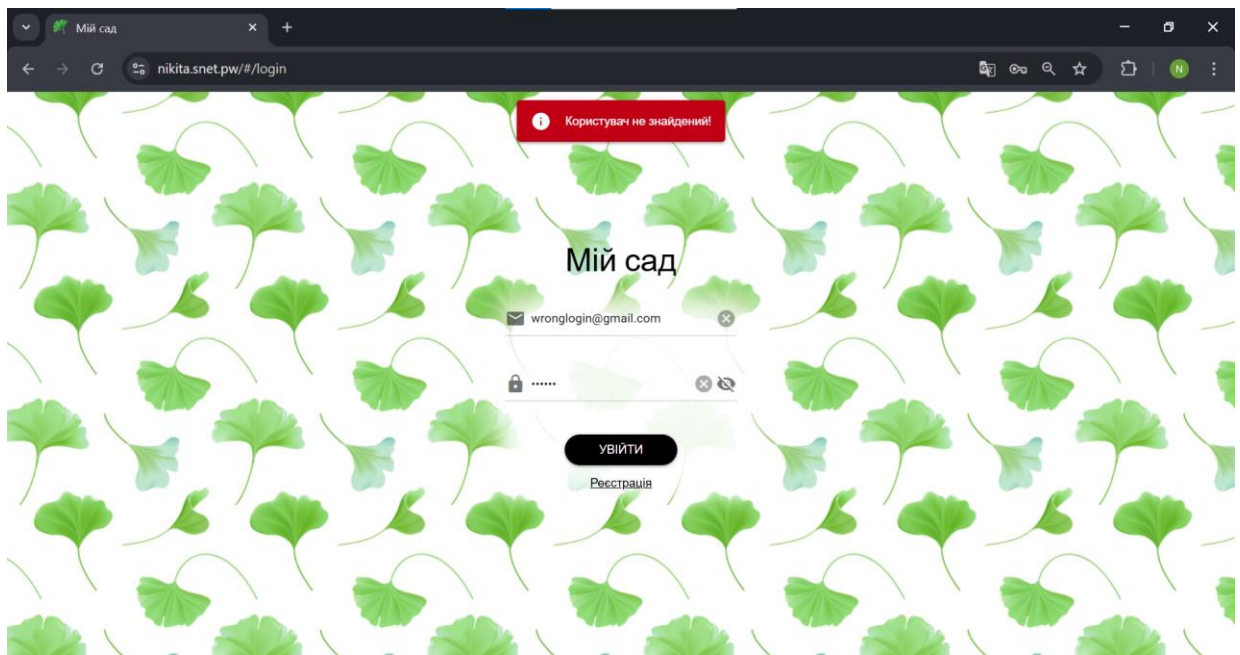


Рис. 3.16 - Інтерфейс користувача для проходження процедури автентифікації

Перевірено сценарій додавання нової рослини до колекції користувача. Користувач переходить до відповідного пункту меню та формує новий запис (рис. 3.17), заповнюючи поля назви, категорії, наукової назви та приміток. У межах тестового сценарію особлива увага приділялась обов'язковим полям та валідації введення: при незаповнених полях «Назва» та «Категорія» кнопка «Зберегти» лишається неактивною, що запобігає створенню некоректних записів.

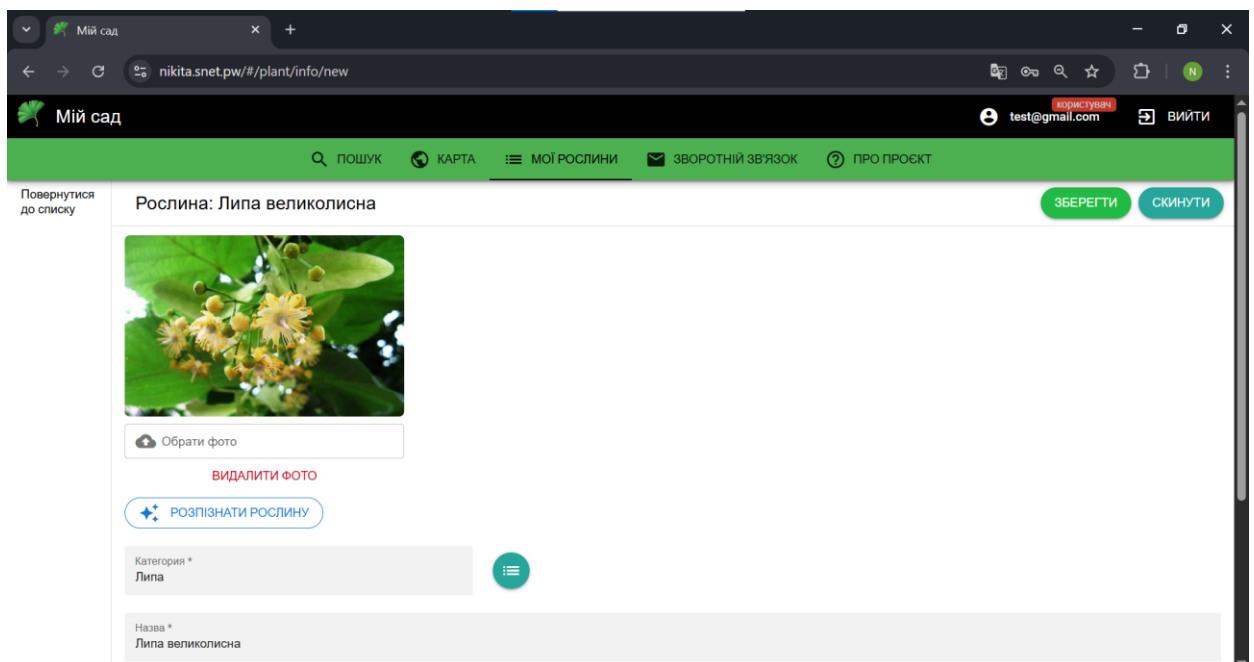


Рис. 3.17 - Додавання нового запису

Після збереження запис автоматично з'являється в загальному переліку рослин (рис. 3.18), а також в особистому переліку «Мої рослини», що підтверджує коректність синхронізації даних між персональним і глобальним відображенням. На скріншоті глобального переліку видно колонки з фотографією-мініатюрою, категорією, назвою, примітками, датою створення та датою останнього оновлення; перехід на повну картку конкретної рослини виконується одинарним кліком по будь-якому рядку таблиці.

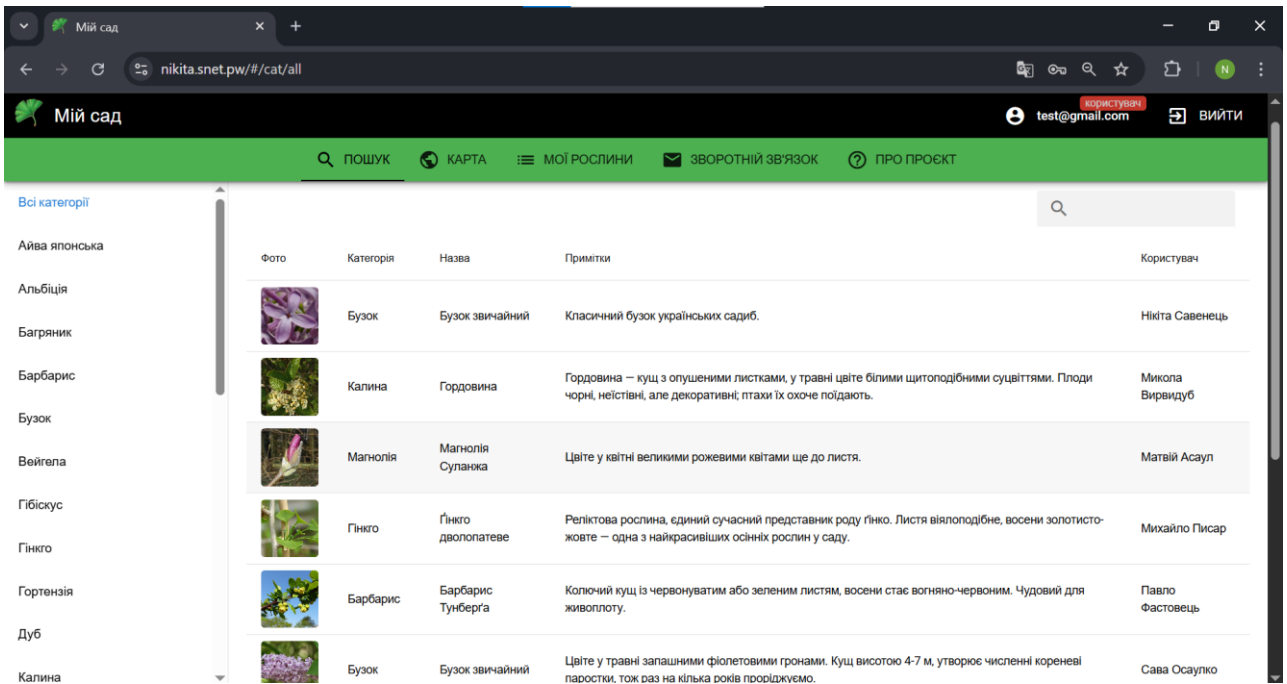


Рис. 3.18 - Глобальний список рослин

Для перевірки роботи функції глобального пошуку у відповідну вкладку було введено пошуковий запит (рис. 3.19). Система коректно повернула релевантні результати, що відповідають введеному фрагменту назви як за точним входженням, так і за частковим. Також у іншому тестовому сценарії перевірено пошук за українською назвою (наприклад, «магнолія»), за латинською назвою (наприклад, «Magnolia») та за категорією; у всіх трьох випадках система видала прогнозовані результати без затримок та помилок.

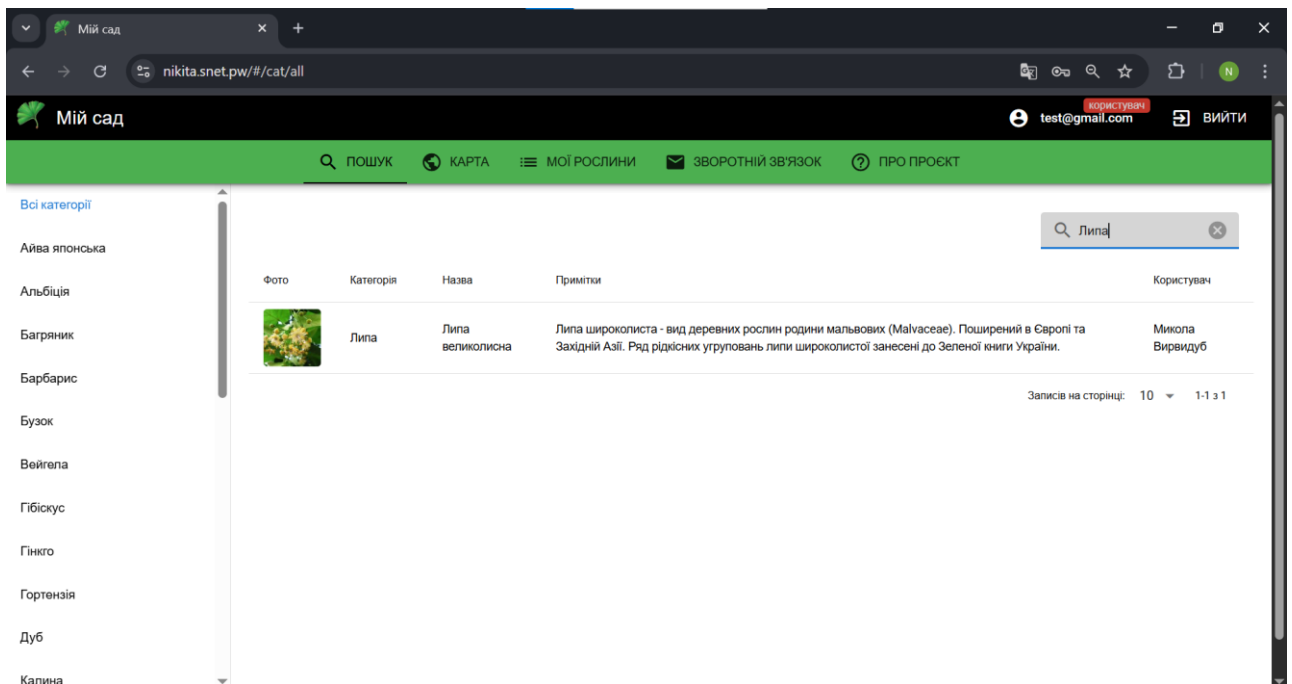


Рис. 3.19 - Пошук записів

Сценарій зворотного зв'язку перевірено шляхом надсилання відгуку через спеціальну форму на вкладці «Повідомлення» (рис. 3.20). У формі введено електронну адресу автора повідомлення та текст відгуку, після чого натиснуто кнопку «Надіслати»; система повернула підтвердження про успішне надсилання повідомлення без перезавантаження сторінки.

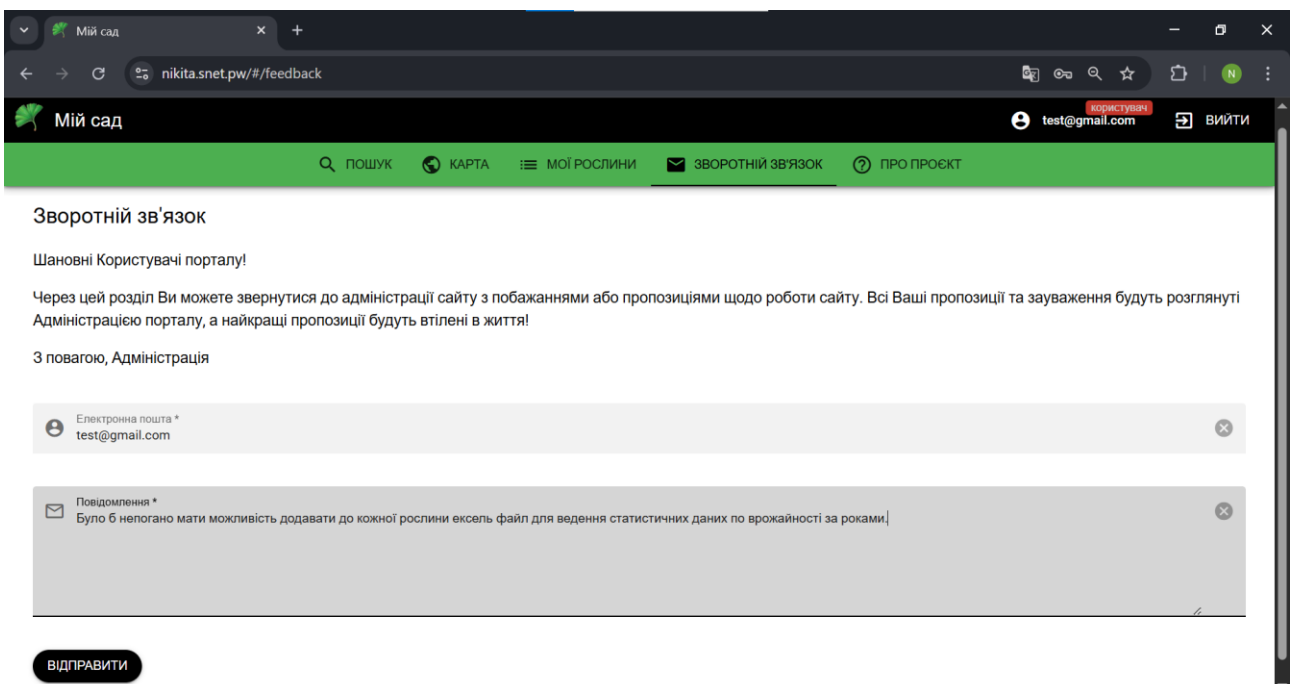


Рис. 3.20 - Зворотній зв'язок

Після цього виконано вхід у систему під обліковим записом адміністратора та переглянуто список повідомлень від користувачів (рис. 3.21) - тестове повідомлення коректно дісталось адресата та відображається в таблиці зі стандартними полями дати, користувача, тексту повідомлення та контактних даних. Перевірено, що електронна адреса автора відображається як активне mailto-посилання, натискання на яке відкриває поштовий клієнт зі сформованим листом-відповіддю

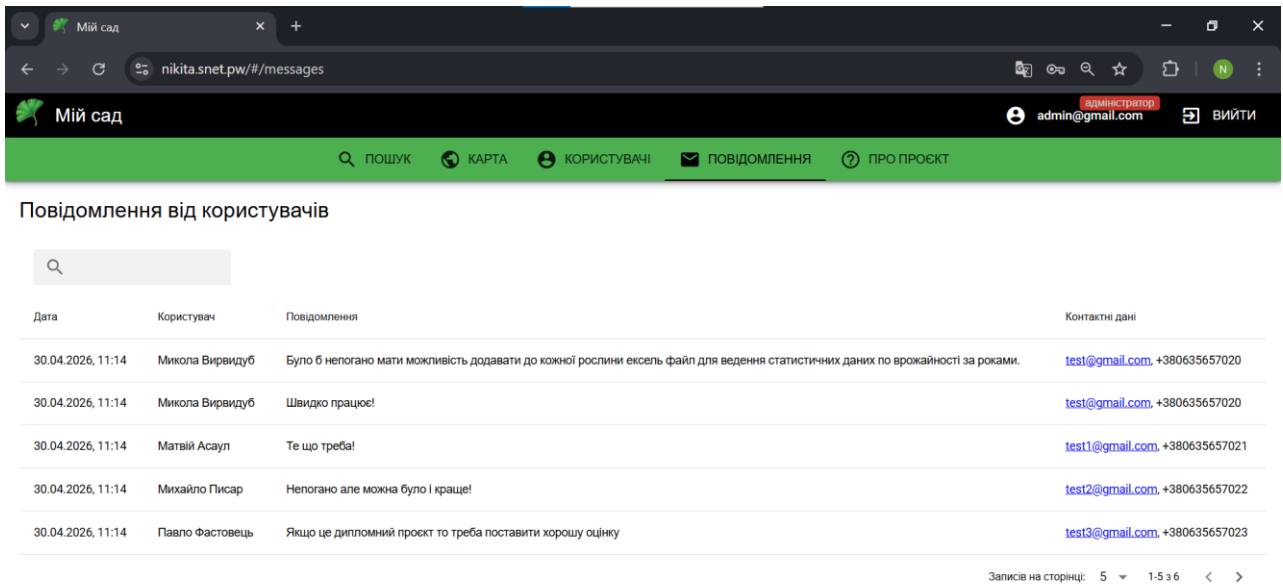


Рис. 3.21 - Зворотній зв'язок

Перевірка гіперпосилань у межах сайту не виявила помилок: усі посилання ведуть на цільові сторінки, перехід між сторінками доступний з будь-якого розділу. Окремо протестовано перехід на головну сторінку через клік на логотип сайту - він також працює коректно та не призводить до розриву сесії авторизованого користувача.

Окремим блоком виконано тестування функціоналу прикріплення фотографії до картки рослини. Перевірено сценарій, коли користувач натискає поле «Обрати фото», обирає файл з диску свого пристрою або робить знімок камерою на мобільному; компонент виконує валідацію файлу (тип JPEG/PNG, мінімальний розмір 256×256 пікселів, максимальний обсяг 10 МБ), стискає його до робочого розміру та одразу відображає прев'ю над формою. Зразок

зовнішнього вигляду форми з прикріпленим фото та кнопкою розпізнавання представлено на (рис. 3.22). У всіх протестованих випадках завантаження файлів встановлених форматів і допустимого розміру виконується коректно; файли неприпустимого типу чи розміру блокуються з виведенням відповідного повідомлення про помилку, причому повідомлення відрізняється залежно від типу проблеми (неправильний формат, занадто малий розмір, занадто великий обсяг). Окремо перевірено сценарій з фотографіями, зробленими камерою мобільного телефону у портретній орієнтації: бібліотека browser-image-compression автоматично враховує EXIF-метадані та повертає коректно орієнтоване зображення.

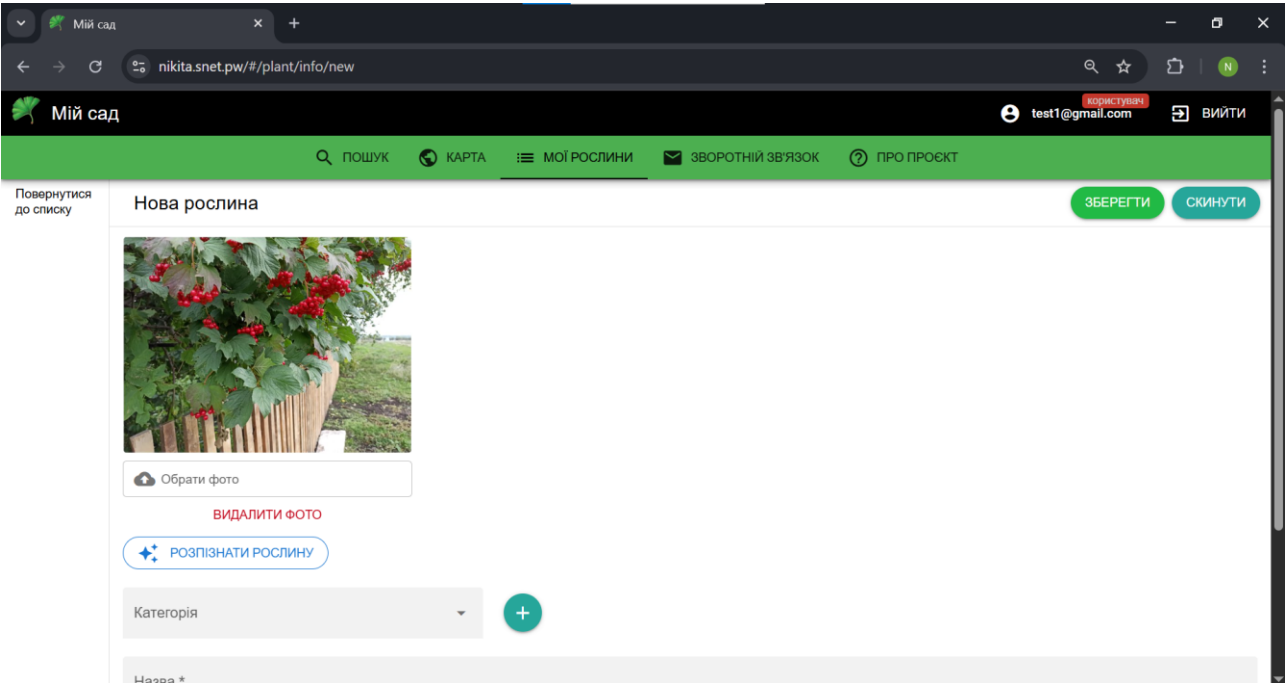


Рис. 3.22 - Форма додавання рослини з прикріпленим фото та кнопкою «Розпізнати рослину»

Протестовано сценарій автоматичного розпізнавання рослини. Після натискання кнопки «Розпізнати рослину» система передає фотографію до зовнішнього сервісу PlantNet, отримує перелік найбільш імовірних кандидатів та відображає їх у модальному діалоговому вікні. Кожен запис містить оцінку впевненості у відсотках, наукову (латинську) назву, поширену українську назву та посилання на існуючу або пропонувану нову категорію. Приклад вікна з результатами наведено на (рис. 3.23). Перевірено, що при виборі користувачем

потрібного рядка поля картки рослини автоматично заповнюються відповідними значеннями, а якщо запропонована назва категорії вже є в локальній базі, вона використовується без створення дублікату (порівняння виконується без врахування реєстру). У межах тестового сценарію перевірено розпізнавання різних видів рослин (магнолія, кедр, дуб) з фотографій листя, квіток та загального вигляду рослини; у переважній більшості випадків система коректно ідентифікувала вид з впевненістю понад 70 відсотків).

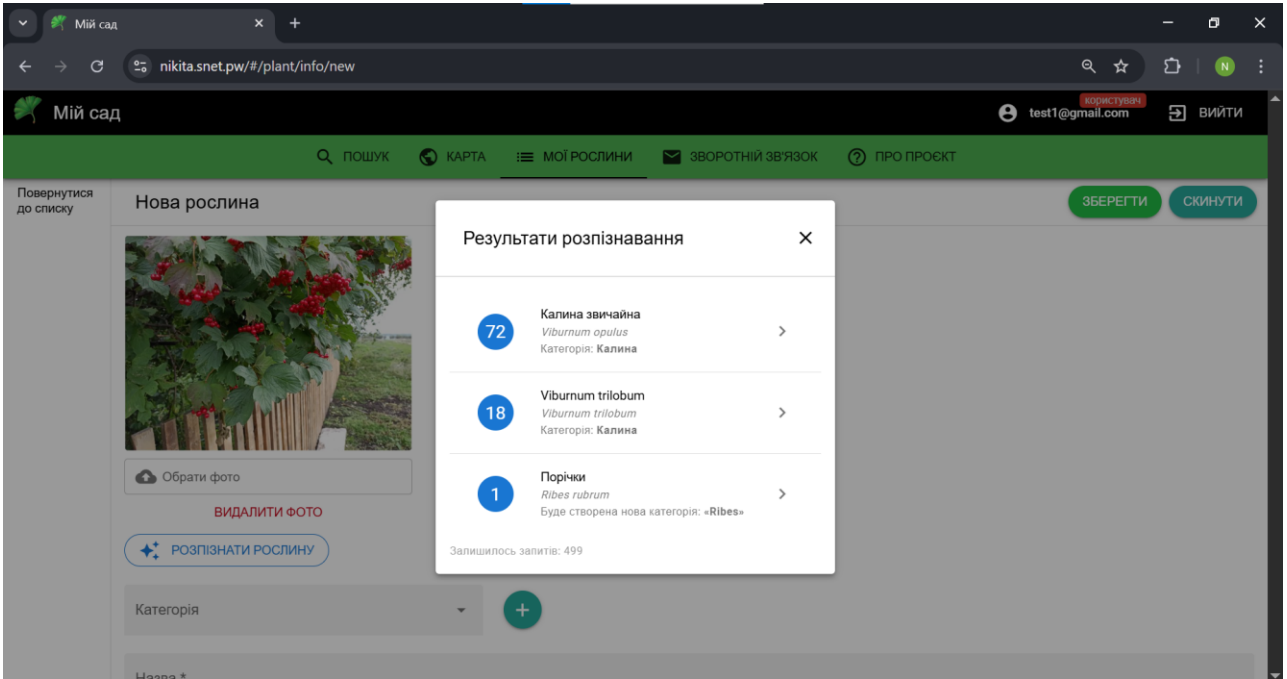


Рис. 3.23 - Діалогове вікно з результатами розпізнавання та трьома кандидатами

Перевірку інтерактивної карти садів виконано через вкладку «Карта», що відкриває мапу України з маркерами усіх зареєстрованих користувачів. Зразок загального вигляду карти представлено на (рис. 3.24). Підтверджено, що при натисканні на маркер коректно відкривається спливаюче вікно з ім'ям садівника, назвою його міста, кількістю рослин у колекції та переліком назв рослин у форматі гіперпосилань. Натиснення на ім'я садівника відкриває поштовий клієнт зі сформованим листом на електронну адресу власника, а натиснення на назву рослини веде на її повну картку. Окремо перевірено сценарій з кількома користувачами з одного міста: завдяки автоматичному зміщенню координат маркери не накладаються один на одного, що зберігає можливість натискання на кожен окремий маркер.

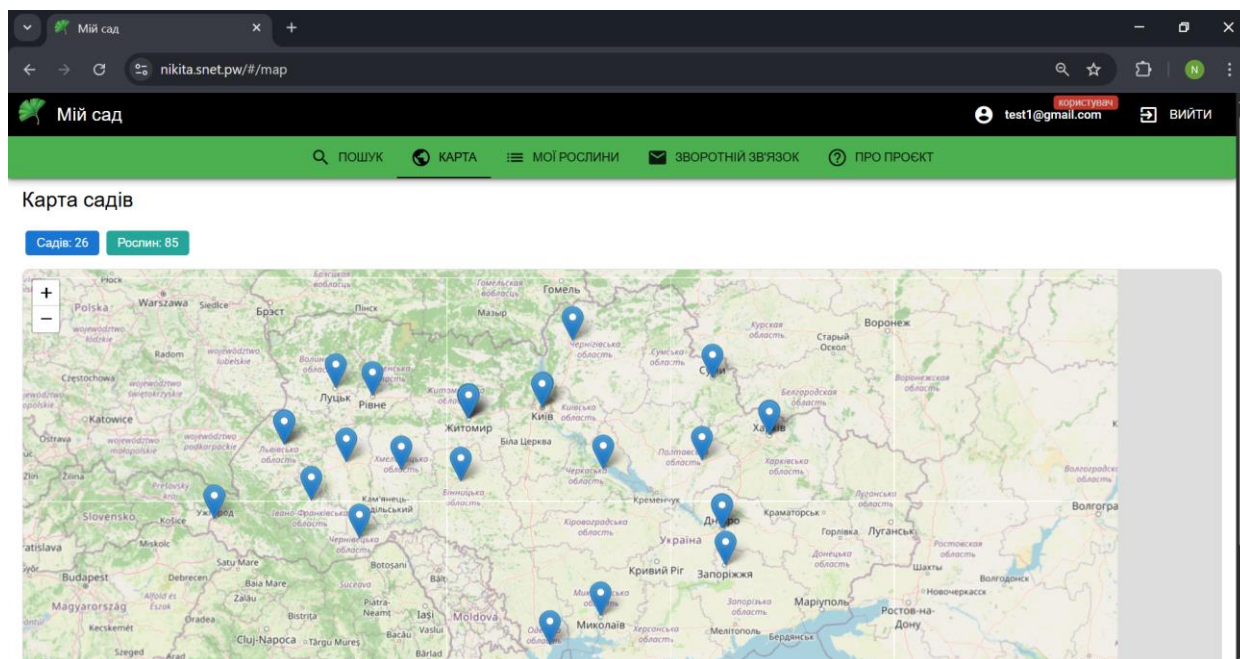


Рис. 3.24 - Загальний вигляд карти садів з маркерами користувачів

Приклад спливаючого вікна показано на (рис. 3.25). На скріншоті видно структуру попапа: жирним шрифтом виведено ім'я садівника як [mailto-](mailto:mailto:)посилання, нижче сірим шрифтом - назва міста і кількість рослин в колекції, далі маркований перелік з назвами рослин (у кожного рядка вказана назва рослини та назва категорії у дужках). Перевірено клікабельність кожного елементу попапа: натискання на ім'я відкриває поштовий клієнт, натискання на назву рослини веде на її повну картку у системі

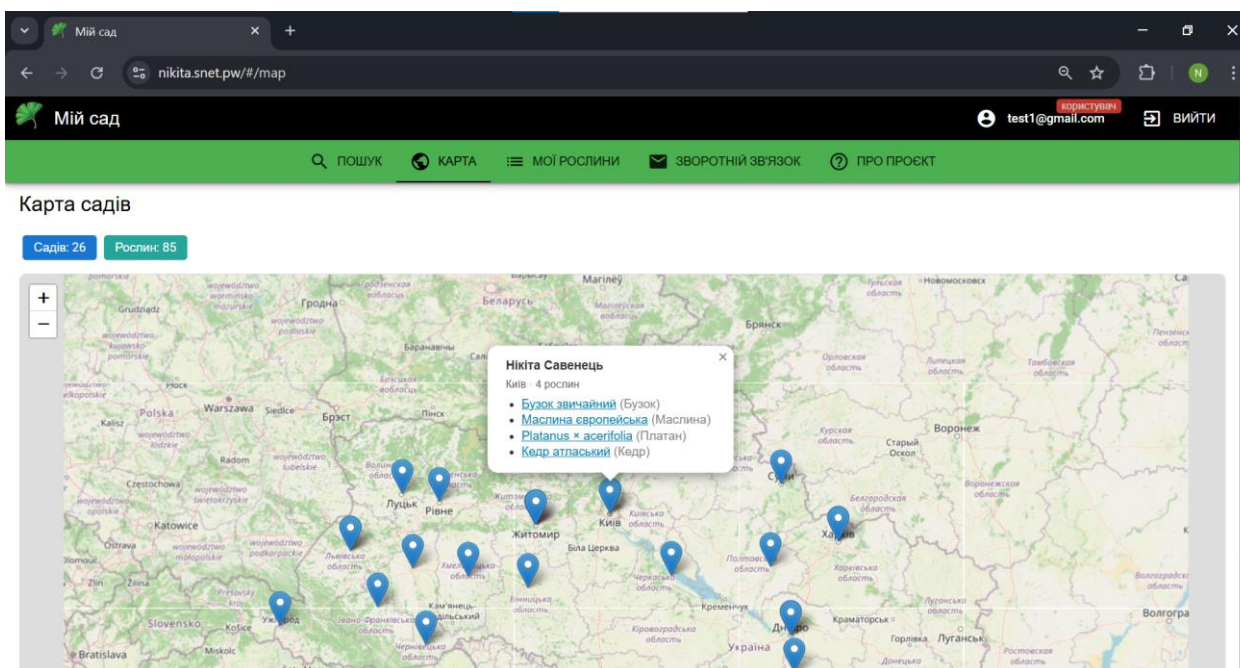


Рис. 3.25 - Спливаюче вікно маркера з переліком рослин користувача

Для коректного відображення точки на мапі у формі реєстрації перевірено роботу обов'язкового поля «Місто» з випадаючим списком ста п'ятдесяти найбільших населених пунктів України (рис. 3.26). Підтверджено, що для зміни міста зареєстрованому користувачу достатньо відкрити сторінку профілю та обрати інший пункт зі списку - координати оновлюються автоматично, і маркер на мапі переміщується після збереження. Окремо перевірено пошук міста за частковим збігом: при початку введення тексту список звужується до варіантів, що містять введений фрагмент; пошук працює як для україномовних назв («Київ», «Львів»), так і для англomовних транслітерацій («Kyiv», «Lviv»).

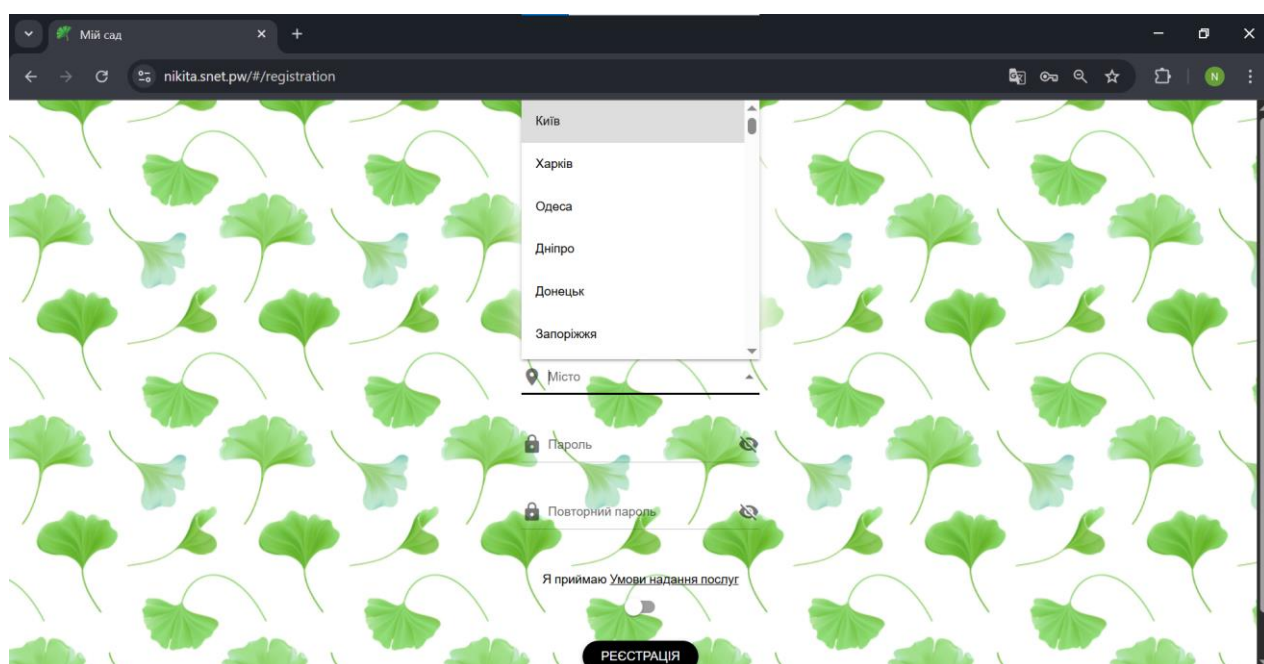


Рис. 3.26 - Поле «Місто» з випадаючим списком українських населених пунктів у формі реєстрації

Загалом, проведене тестування підтвердило, що розроблений веб-додаток обліку рослин відповідає поставленим до нього вимогам та готовий до експлуатації.

Висновки до розділу 3

У межах третього розділу досліджено особливості розгортання, структуру користувацького інтерфейсу та працездатність розробленого програмного

комплексу. Наведено покрокові інструкції роботи з додатком, проілюстровані екранними копіями основних сценаріїв.

Описано методичні засади тестування веб-додатку та виконано повноцінне тестування реалізованої системи. Перевірка кросбраузерності у Google Chrome, Mozilla Firefox та Microsoft Edge показала однотипне відображення інтерфейсу та коректну роботу функціонала у всіх трьох середовищах. Юзабіліті-тестування підтвердило зручність та інтуїтивну зрозумілість навігації, логічну побудову інтерфейсу й цілісність користувацького досвіду.

За результатами проведених тестових сценаріїв зроблено висновок, що розроблений веб-додаток обліку рослин працює коректно та повністю відповідає функціональним вимогам, сформульованим у попередніх розділах.

ВИСНОВКИ

У межах випускної кваліфікаційної роботи бакалавра спроектовано та реалізовано веб-орієнтовану систему обліку рослин з розширеним функціоналом, що виходить за межі типового CRUD-додатку. У процесі виконання роботи розв'язано низку завдань.

Виконано аналіз предметної області, що дозволив окреслити основні аспекти ведення обліку рослин у форматі веб-додатку. Розгляд наявних на ринку аналогів та зіставлення їх можливостей з реальними потребами цільових користувачів показали, що ключовими функціями розроблюваної системи є наявність інтуїтивно зрозумілого користувацького інтерфейсу, послідовно структурованого навігаційного меню та достатньо висока швидкодія. На основі порівняння архітектурних підходів обґрунтовано вибір клієнт-серверної моделі побудови системи з SPA-реалізацією клієнтської частини. Як цільову СУБД обрано MySQL, на боці клієнта застосовано HTML5, CSS3, JavaScript, Vue.js та Quasar, на боці сервера - Node.js, Express та Sequelize.

Проаналізовано структуру та вимоги до даних, з якими оперує система; спроектовано загальний алгоритм роботи додатку, описано взаємодію його модулів та сформовано фізичну схему бази даних. Завдяки побудові діаграми класів виокремлено основні об'єкти системи й зв'язки між ними, що дало змогу заздалегідь усунути потенційні архітектурні проблеми. Описано логіку роботи системи з акцентом на ключових сценаріях, наведено фрагменти програмного коду, що демонструють основні моменти реалізації.

Реалізовано три ключові підсистеми, що відрізняють розроблюваний додаток від базового аналога. По-перше, підсистема завантаження та зберігання фото рослин з двома рівнями деталізації - повне фото та мініатюра, що оптимізує мережевий трафік при роботі зі списками. По-друге, підсистема автоматичного розпізнавання рослин на основі інтеграції з зовнішнім API PlantNet, що повертає топ-3 кандидати з оцінкою впевненості та біологічною таксономією. По-третє, підсистема геоінформаційного представлення на базі бібліотеки Leaflet та шару

					ІІЗ.КР.Б-121-26-ІІЗ	Арк
						76

тайлів OpenStreetMap, що візуалізує географічний розподіл садів користувачів по території України та надає засоби швидкого переходу на картку конкретної рослини.

Розглянуто архітектуру та працездатність програмного комплексу, наведено інструкції роботи з додатком та проведено повноцінне тестування реалізованого функціонала. Тестування у браузерях Google Chrome, Mozilla Firefox та Microsoft Edge підтвердило однотипне відображення сайту та коректність роботи інтерактивних елементів у всіх середовищах. Перевірка інтерфейсу на зрозумілість та зручність показала, що сайт має зручну навігацію, інтуїтивну логічну структуру та приємний візуальний дизайн.

Реалізований програмний комплекс готовий до промислової експлуатації. Систему можна застосовувати як для ведення особистих колекцій рослин садівниками-аматорами, так і для роботи компаній, що займаються вирощуванням та реалізацією посадкового матеріалу. Інтеграція з PlantNet знижує бар'єр входу для початківців, які не володіють ботанічною номенклатурою, а карта садів забезпечує мережевий ефект - спільноту користувачів зі схожими інтересами в межах регіону.

У подальшому розробку можна розширити шляхом впровадження додаткових функціональних можливостей: інтеграції з зовнішніми сервісами оплати, додавання повідомлень-нагадувань про сезонні роботи в саду, реалізації мобільного клієнта на основі вже наявної Cordova-обгортки та модуля автоматичного розпізнавання хвороб рослин за фото з використанням аналогічного підходу до підсистеми ідентифікації виду. Завдяки обраній клієнт-серверній архітектурі розширення системи відбуватиметься оперативно й без значних змін у вже реалізованих компонентах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Юськович-Жуковська В. І. Вебпрограмування : навч. посібник / В. І. Юськович-Жуковська, О. М. Богут. - Рівне : Волинські обереги, 2023. - 384 с.
2. Васильєв О. М. Алгоритми. - Київ : Ліра-К, 2022. - 424 с.
3. Доценко С. І. Організація та системи керування базами даних : навч. посібник / С. І. Доценко. - Харків : УкрДУЗТ, 2023. - 196 с.
4. Жученко А. І. Бази даних : навч. посібник / А. І. Жученко, Л. Д. Ярошук. - Київ : КПІ ім. Ігоря Сікорського, 2022. - 180 с.
5. Добровольський Ю. Г. Стандартизація в інженерії програмного забезпечення : навч. посібник / Ю. Г. Добровольський. - Чернівці : Чернівецький національний університет ім. Ю. Федьковича, 2022. - 140 с.
6. Ковалюк Д. О. Технології розроблення програмного забезпечення : практикум / Д. О. Ковалюк. - Київ : КПІ ім. Ігоря Сікорського, 2022. - 96 с.
7. Баран С. В. Основи web-програмування : навч. посібник / С. В. Баран. - Кривий Ріг : Державний університет економіки і технологій, 2023. - 316 с.
8. Mammino L. Node.js Design Patterns : Level up your Node.js skills and design production-grade applications using proven techniques / L. Mammino, M. Casciaro. - 4th ed. - Birmingham : Packt Publishing, 2025. - 660 p.
9. Newman S. Building Microservices : Designing Fine-Grained Systems / S. Newman. - 2nd ed. - Sebastopol : O'Reilly Media, 2021. - 612 p.
10. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems / A. Géron. - 3rd ed. - Sebastopol : O'Reilly Media, 2022. - 861 p.
11. Meyer E. A. CSS : The Definitive Guide : Web Layout and Presentation / E. A. Meyer, E. Weyl. - 5th ed. - Sebastopol : O'Reilly Media, 2023. - 1062 p.
12. Sarrion E. JavaScript from Frontend to Backend : Learn full stack JavaScript development using the MEVN stack with quick and easy steps / E. Sarrion. - Birmingham : Packt Publishing, 2022. - 336 p.

13. Garaguso P. D. Vue.js 3 Design Patterns and Best Practices : Develop scalable and robust applications with Vite, Pinia, and Vue Router / P. D. Garaguso, O. Zander. - Birmingham : Packt Publishing, 2023. - 296 p.

14. Haverbeke M. Eloquent JavaScript / M. Haverbeke. - 4th ed. - San Francisco : No Starch Press, 2024. - 472 p.

15. Frisbie M. Professional JavaScript for Web Developers / M. Frisbie. - 5th ed. - Indianapolis : Wrox, 2024. - 1200 p.

16. Smirnova S. MySQL Cookbook : Solutions for Database Developers and Administrators / S. Smirnova, A. Tezuysal. - 4th ed. - Sebastopol : O'Reilly Media, 2022. - 1030 p.

17. Ford N. Software Architecture : The Hard Parts : Modern Trade-Off Analyses for Distributed Architectures / N. Ford, M. Richards, P. Sadalage, Z. Dehghani. - Sebastopol : O'Reilly Media, 2021. - 462 p.

18. Zhao A. SQL Pocket Guide : A Guide to SQL Usage / A. Zhao. - 4th ed. - Sebastopol : O'Reilly Media, 2021. - 354 p.

19. Vue.js Guide [Электронный ресурс] / Vue.js Team. - Режим доступа : <https://vuejs.org> - Дата звернення : 08.06.2026

20. Quasar Framework Documentation [Электронный ресурс] / Quasar Framework Team. - Режим доступа : <https://quasar.dev> - Дата звернення : 08.06.2026

21. Node.js Documentation [Электронный ресурс] / OpenJS Foundation. - Режим доступа : <https://nodejs.org/en/docs> - Дата звернення : 08.06.2026

22. Express.js Guide [Электронный ресурс] / OpenJS Foundation. - Режим доступа : <https://expressjs.com/en/guide/routing.html> - Дата звернення : 08.06.2026

23. Sequelize Documentation [Электронный ресурс] / Sequelize Contributors. - Режим доступа : <https://sequelize.org/docs/v6> - Дата звернення : 08.06.2026

24. MySQL 8.0 Reference Manual [Электронный ресурс] / Oracle Corporation. - Режим доступа : <https://dev.mysql.com/doc/refman/8.0/en> - Дата звернення : 08.06.2026

25. MDN Web Docs [Електронний ресурс] / Mozilla Foundation. - Режим доступу : <https://developer.mozilla.org/uk/docs/Web> - Дата звернення : 08.06.2026
26. JSON Web Tokens. Introduction [Електронний ресурс] / Auth0 Inc. Режим доступу : <https://jwt.io/introduction> - Дата звернення : 08.06.2026
27. bcrypt - Node.js package [Електронний ресурс] / Daniel Wirtz, Node.js bcrypt Contributors. - Режим доступу : <https://www.npmjs.com/package/bcrypt> - Дата звернення : 08.06.2026
28. Axios HTTP Client. Documentation [Електронний ресурс] / Matt Zabriskie, Axios Contributors. - Режим доступу : <https://axios-http.com/docs/intro> - Дата звернення : 08.06.2026
29. PlantNet API Documentation [Електронний ресурс] / Pl@ntNet, CIRAD, INRAE, IRD. - Режим доступу : <https://my.plantnet.org/account/doc> - Дата звернення : 08.06.2026
30. Leaflet - an open-source JavaScript library for mobile-friendly interactive maps [Електронний ресурс] / Volodymyr Agafonkin. - Режим доступу : <https://leafletjs.com> - Дата звернення : 08.06.2026
31. OpenStreetMap [Електронний ресурс] / OpenStreetMap Foundation. - Режим доступу : <https://www.openstreetmap.org> - Дата звернення : 08.06.2026
32. Моя оранжерея : програма для каталогізації колекції кімнатних рослин [Електронний ресурс] / Mapix Software. - Режим доступу : <http://plants-garden.com/product/moya-oranzhereya> - Дата звернення : 08.06.2026
33. Thlaspi : онлайн-додаток для ведення колекції рослин, садового щоденника та обліку садових робіт [Електронний ресурс]. - Режим доступу : <https://thlaspi.com> - Дата звернення : 08.06.2026

ДОДАТКИ

					ІІЗ.КР.Б-121-26-ІІЗ	Арк
						81

Лістинг коду, модулю vue.js, для відображення детальної інформації по рослині у пошуку

```

<template>
  <q-page>
    <q-form @submit="onSubmit()">

      <q-toolbar>
        <q-toolbar-title>
          Рослина: {{data.name}}
        </q-toolbar-title>
      </q-toolbar>

      <q-separator></q-separator>

    <template>
      <div class="row">

        <div class="col-xs-12 col-sm-6 col-md-4">
          Категорія: {{data.Category ? data.Category.name : ""}}
        </div>

        <div class="col-xs-12 col-sm-12 col-md-12">
          Назва: {{data.name}}
        </div>

        <div class="col-xs-12 col-sm-12 col-md-12">
          Примітки: {{data.notes}}
        </div>

        <div class="col-xs-12 col-sm-12 col-md-12">
          Створено: {{data.created_at}}
        </div>

        <div class="col-xs-12 col-sm-12 col-md-12">

```

```

        Оновлено: {{data.updated_at}}
    </div>

    <div class="col-xs-12 col-sm-12 col-md-12">
        Користувач: {{data.User ? data.User.name : ""}}
    </div>

    <div class="col-xs-12 col-sm-12 col-md-12">
        Електронна пошта: {{data.User ? data.User.email : ""}}
    </div>

    <div class="col-xs-12 col-sm-12 col-md-12">
        Телефон: {{data.User ? data.User.phone : ""}}
    </div>

</div>
</template>

</q-form>
</q-page>
</template>

<script>
import helpers from '../utils/helpers'

export default {
    name: 'SearchInfo',
    mixins: [
        helpers
    ],
    props: {
        id: String
    },
    data () {
        return {

```

```

        data: {}
    }
},
methods: {
    async getData () {
        this.showLoading()

        try {
            const response = await this.$axios.get('/plant/' + this.id)
            this.data = response.data
            this.data.created_at = new Date(this.data.created_at).toLocaleDateString(undefined, { hour:
'numeric', minute: 'numeric' })
            this.data.updated_at = new Date(this.data.updated_at).toLocaleDateString(undefined, { hour:
'numeric', minute: 'numeric' })
        } catch (error) {
            console.log(error.response.data.message)
        }

        this.hideLoading()
    }
},
watch: {
    id: function () {
        this.getData()
    }
},
mounted () {
    this.getData()
}
}
</script>

<style scoped>
.row > div {
    padding: 10px 15px;

```

```
}  
.row {  
  font-size: 17px;  
}  
</style>
```

					ИПЗ.КР.Б-121-26-ИЗ	Арк
						85

Лістинг коду роутінку фронтенд частини

```

const routes = [
  { path: '/', component: () => import('layouts/AppLayout.vue'),
    children: [
      { path: '/', component: () => import('pages/About.vue'), name: 'About' },
      { path: '/users', component: () => import('pages/Users.vue'), name: 'Users' },
      { path: '/feedback', component: () => import('pages/Feedback.vue'), name: 'Feedback' },
      { path: '/messages', component: () => import('pages/Messages.vue'), name: 'Messages' },
      { path: '/profile', component: () => import('components/Profile.vue'), name: 'Profile' },
      { path: '/plant', component: () => import('pages/plant/MyList.vue'), name: 'MyList' },
      { path: '/plant', component: () => import('pages/plant/MyMenu.vue'),
        children: [
          { path: '/plant/info/:id', component: () => import('pages/plant/MyInfo.vue'), name: 'MyInfo',
            props: true }
        ]
      },
      { path: '/cat', component: () => import('pages/plant/SearchMenu.vue'),
        children: [
          { path: '/cat/:id', component: () => import('pages/plant/SearchList.vue'), name: 'SearchList',
            props: true },
          { path: '/cat/info/:id', component: () => import('pages/plant/SearchInfo.vue'), name:
            'SearchInfo', props: true }
        ]
      },
    ]
  },
  { path: '/login', component: () => import('components/Login.vue'), name: 'Login' },
  { path: '/registration', component: () => import('components/Registration.vue'), name: 'Registration'
}
]

// Always leave this as last one
if (process.env.MODE !== 'ssr') {
  routes.push({
    path: '*',

```

```
component: () => import('pages/Error404.vue')
  })
}
```

export default routes

Лістинг коду роутінку бекенд частини

```

const verifySignUp = require('./verifySignUp')
const authJwt = require('./verifyJwtToken')

const controller = require('../controller/controller.js')

module.exports = function router(app) {
  app.post('/api/auth/signup', [verifySignUp.checkDuplicateEmail], controller.signup)
  app.post('/api/auth/signin', controller.signin)
  app.get('/api/auth/access', [authJwt.verifyToken], controller.access)
  app.post('/api/saveProfile', [authJwt.verifyToken], controller.saveProfile)
  app.post('/api/sendMessage', [authJwt.verifyTokenIfExists], controller.sendMessage)
  app.get('/api/messages', [authJwt.verifyToken, authJwt.isAdmin], controller.messages)

  app.get('/api/dictionary', controller.dictionary)
  app.get('/api/plants', [authJwt.verifyToken], controller.plants)
  app.get('/api/plant/:id', controller.plant)
  app.post('/api/plantSave', [authJwt.verifyToken], controller.plantSave)
  app.get('/api/catplants/:id', controller.catplants)
}

```

Лістинг коду контролеру API, яке використовується для відображення
детальної інформації по рослині у пошуку

```
exports.plant = (req, res) => {
  db.plant.findOne({
    where: {
      id: req.params.id
    },
    include: [
      {
        model: db.category,
        as: 'Category',
        attributes: ['name']
      },
      {
        model: db.user,
        as: 'User',
        attributes: ['name']
      }
    ]
  }).then((plant) => {
    res.status(200).send(plant)
  }).catch((err) => {
    res.status(500).send(`Error: ${err}`)
  })
}
```