Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет КН
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Веб застосунок для обліку та планування замовлень по догляду за садом та газоном. Фронт-енд
(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-7

Давід КОШЕЛЬ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доцент кафедри ПІ Андрій БАБІЙ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет КН
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Кошелю Давіду Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Веб застосунок для обліку та планування замовлень по догляду за садом та газоном. Фронт-енд.
Затверджена наказом по університету від 19.05.2025 №397 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 13.06.2025
3. Вихідні дані до роботи Розробити клієнтську частину веб застосунку для обліку та планування замовлень по догляду за садом та газоном.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.06.2025	*виконано*
2	Створення специфікації ПЗ	24.06.2025	*виконано*
3	Проектування ПЗ	25.06.2025	*виконано*
4	Розробка ПЗ	27.06.2025	*виконано*
5	Тестування ПЗ	01.06.2025	*виконано*
6	Оформлення пояснювальної записки	02.06.2025	*виконано*
7	Написання заяви	02.06.2025	*виконано*
8	Підготовка презентації та доповіді	03.06.2025	*виконано*
9	Підготовка облікового запису GitHub	06.06.2025	*виконано*
10	Попередній захист	08.06.2025	*виконано*
11	Нормоконтроль, рецензування	09.06.2025	*виконано*
12	Здача роботи у електронний архів	10.06.2025	*виконано*
13	Допуск до захисту у зав. кафедри	11.06.2025	*виконано*
14	Захист кваліфікаційної роботи	13.06.2025	*виконано*

Дата видачі завдання «19» « травня » 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ доцент кафедри ПІ Андрій БАБІЙ
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 60 стор., 11 рис., 7 додатків, 6 джерел.

ПЛАНУВАННЯ, СИСТЕМА ОБЛІКУ, САДИ ТА ГАЗОНИ, ЗАМОВЛЕННЯ, JAVA SCRIPT, REACT, VISUAL STUDIO CODE.

Об’єкт розробки – веб застосунок для обліку та планування замовлень по догляду за садом та газоном.

Мета розробки – створення клієнтської частини веб застосунку для обліку та планування замовлень по догляду за садом та газоном, який забезпечує зручну взаємодію клієнта, працівника та менеджера з цифровою платформою для надання послуг з догляду за прибудинковими ділянками.

Метод рішення – використано середовище Visual Studio Code, JavaScript-бібліотеку React, а також засоби маршрутизації, керування станом та асинхронного обміну даними (Axios). Клієнтська частина реалізована із застосуванням компонентного підходу.

У результаті розробки створено клієнтську частину веб застосунку для обліку та планування замовлень по догляду за садом та газоном, через яку можна взаємодіяти з мікро-сервісами серверної частини системи. Це повноцінна інтерактивна веб-система для замовлення та управління сервісами з догляду за садом або газоном, яка підвищує прозорість та ефективність побутових послуг і забезпечує комфортну взаємодію між усіма учасниками процесу.

ABSTRACT

APPEARANCE SYSTEM, GARDENS AND LAWNS, JAVA SCRIPT, LAYOUT, PLANNING, REACT, VISUAL STUDIO CODE

The object of development is a web application for accounting and planning garden and lawn care orders.

The purpose of the development is to create a client part of a web application for accounting and planning garden and lawn care orders, which provides convenient interaction between the client, employee and manager with a digital platform for providing home care services.

Solution method – Visual Studio Code environment, React JavaScript library, as well as routing, state management, and asynchronous data exchange tools (Axios) were used. The client part was implemented using a component approach.

As a result of the development, a client part of a web application was created for accounting and planning garden and lawn care orders, through which you can interact with the micro-services of the server part of the system. This is a full-fledged interactive web system for ordering and managing garden or lawn care services, which increases the transparency and efficiency of household services and ensures comfortable interaction between all participants in the process.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface

SPA - Single Page Application

CSS – Cascading Style Sheets

DOM – Document Object Model

HTTPS – Hypertext Transfer Protocol Secure

HTTP – Hypertext Transfer Protocol

JWT – JSON Web Token

REST – Representational State Transfer

UI – User Interface

UX – User Experience

URL – Uniform Resource Locator

JSON – JavaScript Object Notation

NPM – Node Package Manager

ВСТУП

Темою кваліфікаційної роботи є веб застосунок для обліку та планування замовлень по догляду за садом та газоном. Основне завдання веб застосунку для обліку та планування – надання можливості не доглядати за садами та газонами самостійно, а зручним чином делегувати це працівникам через застосунок на постійній чи тимчасовій основі.

Метою роботи є розробка програмної системи, яка складається з клієнтської та серверної частини додатку. Веб застосунок GreenFlow почав розроблятися для того, щоб вирішити цілу низку актуальних проблем, із якими стикаються як клієнти, так і виконавці послуг. Клієнти часто не мають зручного інструменту для замовлення обслуговування ділянки, відстеження статусу робіт або здійснення онлайн-оплати. Працівники, у свою чергу, не можуть ефективно обирати завдання, планувати зайнятість або взаємодіяти з менеджментом. Компанії, які координують подібні процеси, потребують централізованої системи обліку замовлень, інвентаризації ресурсів і керування працівниками.

Веб застосунок поділяється на частини для працівників та клієнтів. Для розробки клієнтської частини продукту використовувалась середа розробки Visual Studio Code. Застосунок реалізовано як SPA з використанням сучасної JavaScript-бібліотеки React, що забезпечує зручну навігацію, високу продуктивність та адаптивний інтерфейс. Також була використана бібліотека Axios для взаємодії з серверною частиною шляхом відправлення HTTPS запитів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Сучасний ринок послуг стрімко розвивається, кількість зайнятих людей, які з початку сезону бажають зручно та без ризику делегувати догляд за своїми сади та газонами впевнено росте з кожним роком. У багатьох країнах, включно з Україною, послуги з догляду за зеленими зонами надаються малими підприємствами або самозайнятими працівниками. Водночас, у клієнтів (власників ділянок) часто немає зручного цифрового способу організувати замовлення, відстежити графік догляду, оплатити виконану роботу та залишити зворотний зв’язок. З іншого боку, працівники з неповною зайнятістю зазвичай змушені самостійно шукати клієнтів або покладатися на неструктуровані оголошення.

На даний момент на ринку подібних систем обліку існує дуже не велика кількість веб застосунків та мобільних додатків для цієї задачі, проте можна виділити “Sadi Control” - платформу для управління садовою компанією, яка планує випуск мобільного додатку у 2026 році у App Store та Google Play. Платформа поділяється на веб-застосунок для менеджерів або керівників та на мобільний додаток для безпосередньо садівників. Сторінку сайту цієї компанії ви можете побачити на скріншоті (див. рис. 1.1)

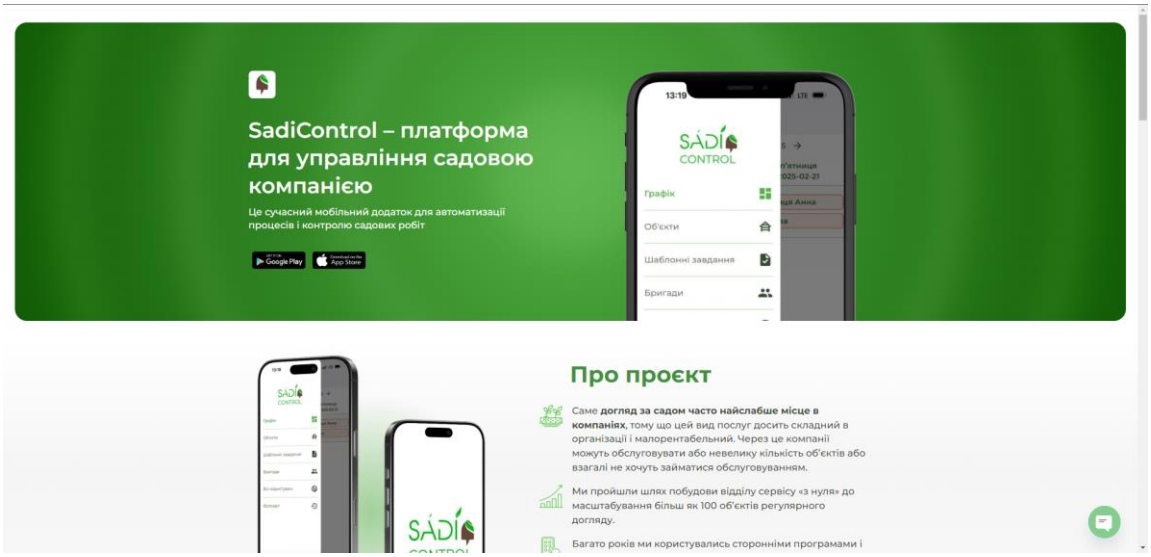

Рисунок 1.1 – Сайт платформи для управління Sadi Control

Як можна побачити, сайт виглядає досить не погано та надає інформацію усім бажаючим про плюси даної платформи та існуючий функціонал у наглядній формі.

1.2 Ключові ролі

Система матиме декілька ключових ролей, пройдемось по кожній та проаналізуємо.

1.2.1 Клієнт

Клієнт ― особа, яка реєструється в системі, додає ділянки, формує план догляду, створює замовлення і сплачує за виконані роботи. Для клієнта важливо мати можливість простого доступу до замовлень, автоматичного формування документів (актів виконаних робіт) та можливості оперативно та зручно оплачувати рахунки. У клієнтів зазвичай стоїть потреба догляду за садом, наприклад, стрижка трави, вичищення території від різного виду сміття, що псує зовнішній вигляд саду, обслуговування різних систем для вирощування або поливу врожаю та інше.

1.2.2 Працівники

Працівник ― особа, яка має обмежену зайнятість і може виконувати замовлення в системі за вільним графіком. Працівник має отримувати тільки актуальні доступні замовлення у своїй геозоні, відзначати початок та завершення роботи, а також оновлювати інформацію про видане й використане їм обладнання та інструменти. У людей з навичками садівництва, які є потенціальними працівниками стоїть потреба у пошуку зручної підробітки не маючи фіксованого графіку, щоби мати можливість виходити на заявки, коли працівнику це зручно.

1.2.3 Менеджери

Менеджер — це співробітник компанії, відповідальний за перевірку та підтвердження працівників, що відправили заявку на роботу через систему, управління працівниками, контроль запасів складу з обладнанням, перевірка різного роду звітів та створення документів оплати тощо. Крім того, менеджери слідкують за замовленнями та якістю їх виконання працівниками.

1.3 Виявлення та вирішення проблем

Багато людей хотіли би мати доглянутий сад або газон проте не витрачаючи час на догляд або пошук працівників, що могли би навести порядок замість них кожного сезону. Зайняті люди бажають просто відкрити смартфон, натиснути на кнопку замовлення та вільно займатися своїми справами будучи впевненими у безпеці та якості обслуговування.

Кількість послуг, що надаються через інтернет збільшується з великою швидкістю, проте систем обліку для догляду за садом та газоном дуже мало, особливо на ринку України.

Після огляду основного конкуренту можна назвати його плюси і мінуси :

а) плюси платформи для управління SadiControl:
   1) наявність мобільного додатку, що зручно для клієнтів
   2) простий та інтуїтивно зрозумілий інтерфейс;
   3) наявність веб застосунку для менеджерів та керівників.

б) мінуси платформи для управління SadiControl:
   1) відсутні фото об’єктів для роботи (садів та газонів);
   2) відсутня оплата рахунків.

Компанії, що надають подібні послуги догляду, зіштовхуються з низькою труднощів: пошук персоналу на неповний робочий день, облік обладнання та інструментів, контроль якості виконання тощо.

1.3.1 Неможливість оперативно призначати працівників

У традиційній моделі диспетчер вручну розподіляє замовлення, не враховуючи реального місцезнаходження працівника або його зайнятість, що викликає затримки, простої або незадоволення клієнтів.

Веб застосунок пропонує алгоритм геолокаційного відбору замовлень. Працівники бачать лише ті завдання, які знаходяться в межах їхньої зони обслуговування, та самі обирають, які замовлення брати у виконання.

1.3.2 Неструктурованість замовлень

Без єдиної системи для фіксації запитів клієнтів компанії часто покладаються на телефонні дзвінки, месенджери або паперові записи. Це призводить до втрати інформації, дублювання запитів або виконання робіт без підтвердження.

Рішення полягає у використанні централізованої база замовлень з фіксацією кожного звернення клієнта, автоматичною генерацією завдань та контролем статусу їх виконання в режимі реального часу.

1.3.3 Відсутність зворотного зв’язку та звітності

Компанії, що працюють без цифрових систем, часто не мають історії взаємодії з клієнтом, не формують актів, не зберігають чітку статистику щодо працівників і обсягів роботи.

Система вирішує цю проблему автоматичним формування звітних документів (рахунків, актів виконаних робіт), зберігання історії замовлень, аналітика по клієнтах, районах і типах робіт.

1.4 Постановка задачі

Один із актуальних напрямів є надання клієнтам можливості замовити обслуговування саду або газону через веб застосунок, який автоматизує більшість етапів взаємодії між клієнтом та компанією, що виконує ці замовлення.

Вирішення потреб користувача полягатиме у розробці централізованої веб-системи, яка дозволить значно покращити прозорість та зручність усіх процесів, таких як подача, обробка замовлень, виставлення рахунків та виїзд працівників.

1.4.1 Цільова аудиторія

Основною цільовою аудиторією даного застосунку будуть наступні люди:

- які не бажають витрачати час на догляд за садом;
- які не бажають шукати працівників кожного сезону;
- які бажають бути впевненими у якості обслуговування;
- садовники, які шукають підробіток;
- ландшафтні компанії, які шукають систему для обліку
- зайняті люди
- середнього достатку та вище
- різного віку;
- різної статі;
- різної професії;
- користувачі інтернету та ПК.

Таким чином, можна зрозуміти, що користувачами даного веб застосунку можуть майже будь які люди, котрі не мають фінансових труднощів або працівники саду чи ландшафтні дизайнери, що шукають підробіток чи постійну роботу.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Загальні вимоги

У процесі розробки клієнтської частини веб застосунку для обліку та планування догляду за садом та газоном GreenFlow основним завданням було створення зручного, адаптивного та функціонального інтерфейсу для взаємодії з платформою обліку для догляду за садом та газоном. Інтерфейс має відповідати сучасним стандартам дизайну та бути реалізованим з використання бібліотек React та Axios із компонентним підходом.

Основною метою фронтенд-частини програмного забезпечення є забезпечення зручного та інтуїтивно зрозумілого інтерфейсу для користувачів, які можуть виступати у ролі замовників (власників садів) або виконавців (працівників). Користувачі мають змогу створювати та редагувати інформацію про свої сади, переглядати доступні послуги, робити замовлення, переглядати статуси виконаних робіт, поповнювати баланс тощо.

Вебінтерфейс має бути адаптивним, але ціль це лише підтримка десктопних розширення екранів, забезпечувати захищену авторизацію та коректно взаємодіяти з бекенд-сервісами системи через REST API. Також важливою вимогою є дотримання принципів UX/UI-дизайну, що сприяє швидкому залученню користувачів та зменшенню кількості помилок при роботі із застосунком.

На момент розробки передбачено використання сучасного JavaScript-фреймворку React, який забезпечує компонентну структуру застосунку, ефективне оновлення DOM та розширюваність. Компоненти мають бути повторно використовуваними, логічно розмежованими, а логіка керування станом реалізована за допомогою хуків useState та useEffect тощо.

Загальні положення також охоплюють вимоги до інтеграції з мікросервісною архітектурою, де фронтенд є незалежним клієнтом, який взаємодіє з низкою REST API для отримання та відправлення даних у форматі JSON.

Веб система складається з клієнтської частини у вигляді веб сайту для ПК та серверної частини.

2.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики клієнтської частини вебзастосунку, які не стосуються безпосередньо реалізації окремих функцій, але впливають на загальну зручність, продуктивність, масштабованість та безпеку системи. У межах цього проєкту, який передбачає розробку фронтенду для сервісу з догляду за садами та газонами, було визначено наступні нефункціональні вимоги:

- інтерфейс повинен бути інтуїтивно зрозумілим для користувача без потреби додаткового навчання;
- всі основні дії супроводжуються підказками або повідомленнями (toast або inline alerts) ;
- інтерфейс має бути локалізованим українською мовою, з підтримкою кирилиці;
- створення та оновлення даних має відбуватись без повного перезавантаження сторінки (SPA-підхід) ;
- для кожного запиту до серверу вимагається JWT-токен, який зберігається у localStorage;
- перевірка валідності токена здійснюється перед кожною критичною дією (оновлення профілю, створення саду, замовлення) ;
- інтерфейс підтримується в останніх версіях основних браузерів: Google Chrome, Firefox, Safari, Microsoft Edge;
- висока швидкість завантаження сторінок та мінімальна затримка взаємодії;
- мінімальна залежність від сторонніх бібліотек (окрім офіційних);
- відсутність критичних помилок в інтерфейсі та обробці форм;
- безпечне зберігання даних у локальному сховищі браузера (JWT, профіль).

Крім того, користувач повинен мати ПК, доступ до мережі інтернет та володіти базовими навичками користування ПК.

2.3 Функціональні вимоги

Функціональні вимоги описують основні дії, які має виконувати клієнтська частина вебзастосунку, а також її взаємодію з користувачами та серверною частиною системи. Для системи з догляду за садами та газонами розроблено низку інтерфейсів, кожен з яких виконує чітко визначені функції в рамках загального бізнес-процесу.

Нижче наведено ключові функціональні можливості, які повинен підтримувати фронтенд-застосунок:

а) реєстрація та авторизація користувачів;

1) надання можливості користувачам зареєструвати обліковий запис з роллю замовника або працівника;

2) авторизація користувача за допомогою JWT-токенів;

3) виведення повідомлень про помилки під час автентифікації;

б) управління профілем користувача;

1) редагування персональної інформації (ПІБ, e-mail, телефон, місто) ;

2) відображення балансу та можливість його поповнення через інтеграцію з платіжною системою PayPal;

в) управління садами (для замовника) ;

1) створення нового саду із зазначенням назви, адреси, широти, довготи та опису;

2) завантаження до 12 фотографій, що зберігаються на сервері;

3) перегляд та редагування наявних садів, можливість змінити опис та координати;

4) видалення саду;

в) створення замовлення;

1) можливість вибрати сад та додати одну або декілька послуг із випадаючого списку;

2) визначення бажаної дати виконання та коментаря;

3) підтвердження замовлення та його оплата з балансу користувача;

г) перегляд власних замовлень;

1) виведення списку замовлень із детальною інформацією про послуги, дату, статус, опис та суму.

2) застосування візуальних індикаторів для різних статусів замовлень (створене, відкрите, виконане, скасоване тощо).

д) інтерфейс працівника;

1) відображення списку доступних до виконання замовлень на основі геолокації;

2) можливість прийняти замовлення, переглянути деталі, опис та координати;

3) перегляд списку замовлень, що вже взяті у роботу;

е) оренда обладнання;

1) пошук доступного обладнання за геолокацією з фільтрацією за ціною та назвою;

2) перегляд детальної інформації про кожну одиницю техніки;

3) оренда обладнання та відображення всіх активних оренд користувача.

Функціональні можливості реалізовано за допомогою React-компонентів, які взаємодіють із бекенд-сервісами через Axios та REST API. Всі запити супроводжуються відповідним заголовком авторизації.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір архітектурного підходу до фронтенд-розробки

У процесі створення клієнтської частини вебзастосунку для сервісу догляду за садами та газонами одним із ключових етапів стала побудова раціональної архітектури, яка дозволила б забезпечити масштабованість, підтримуваність та розширюваність системи. Оскільки проєкт є частиною мікросервісної системи, фронтенд мав органічно взаємодіяти з REST API кожного з мікросервісів, дотримуючись принципів незалежності, модульності та односпрямованості потоків даних. Враховуючи сучасні тенденції розробки вебінтерфейсів, було прийнято рішення реалізувати клієнтську частину із використанням бібліотеки React. React забезпечує компонентно-орієнтований підхід до проєктування інтерфейсів користувача, що відповідає потребам створення складної, інтерактивної системи з багатьма модулями: реєстрація та авторизація, замовлення послуг, керування садом, оренда техніки, перегляд історії тощо.

Архітектура фронтенду базується на шаблоні Single Page Application (SPA), що передбачає завантаження одного HTML-документа і динамічну зміну його вмісту за допомогою JavaScript без необхідності перезавантаження сторінки. Такий підхід позитивно впливає на швидкість відгуку інтерфейсу, користувацький досвід та навантаження на сервер.

Для забезпечення чіткої структури та керованого потокового обміну даними між компонентами застосовано підхід, подібний до Flux-архітектури. У якості механізму керування станом додатка обрано використання локального стану в компонентах через React useState та useEffect. У модулях, які вимагають централізованого зберігання або обміну даними, таких як обробка токенів автентифікації або глобальних фільтрів, передбачено передачу даних через React Context.

Кожен функціональний модуль застосунку організовано в межах окремого компонента або групи компонентів, що відповідають за ізольовану логіку. Наприклад, компоненти MyOrders, CreateOrder, GardenPage, CreateGarden

реалізують відповідно відображення замовлень користувача, форму створення нового замовлення, перегляд та редагування інформації про сад. Це дозволяє легко розширювати функціонал і повторно використовувати код.

З метою покращення навігації та логічної організації сторінок застосовано бібліотеку React Router. Це забезпечило підтримку маршрутизації між різними розділами сайту, а також передачу параметрів через URL, що особливо важливо для перегляду або редагування конкретного об’єкта (наприклад, /my-gardens/:id).

Таким чином, обрана архітектура клієнтської частини базується на сучасних підходах, які добре зарекомендували себе у промисловій розробці вебзастосунків, і забезпечує надійну основу для подальшого розвитку системи

3.2 Структура клієнтського застосунку та компоненти

Для забезпечення зручної навігації, повторного використання коду та спрощення підтримки, клієнтська частина системи була реалізована з використанням компонентного підходу, який пропонує React. Уся логіка користувацького інтерфейсу була структурована у вигляді функціональних компонентів, кожен з яких відповідає за окрему функціональність або представлення.

Загальна структура клієнтської частини проєкту знахотиься у папці src та має вигляд:

- components (універсальні компоненти інтерфейсу, наприклад, PhotoGallery, PhotoUploader, MapLocationSelector);
- pages (сторінки, наприклад, MyOrders.jsx, CreateOrder.jsx, GardenPage.jsx, Leasing.jsx, MyLeased.jsx);
- styles (окремі CSS-файли для кожної сторінки та UI-компонентів);
- AppRouter.jsx (маршрутизація додатку);
- App.js (головний компонент-запуск);
- index.js (точка входу в застосунок);
- context (контексти для використання у будь якій частині програми)

Внутрішня логіка компонентів реалізована за допомогою хуків: useState для керування локальним станом, useEffect для побічних ефектів (наприклад, завантаження даних при монтуванні), useRef для взаємодії з DOM (наприклад, для завантаження фото).

Усі запити до бекенду виконуються за допомогою Axios. Токен авторизації (JWT) додається до кожного запиту через заголовок Authorization. У разі помилок користувачу надається повідомлення про невдалу дію (через alert або inline-елементи).

Завдяки такій структурі забезпечується:

- логічна ізоляція функціональних блоків;
- можливість повторного використання компонентів;
- легкість масштабування;
- зручність супроводу й оновлення коду.

3.3 UML проєктування ПЗ

Перед початком розробки веб застосунку для обліку та планування догляду за садом та газоном GreenFlow були визначені основні функції для кожної ролі. Після детального аналізу була створена Use-case діаграма (див. рис. 3.1).

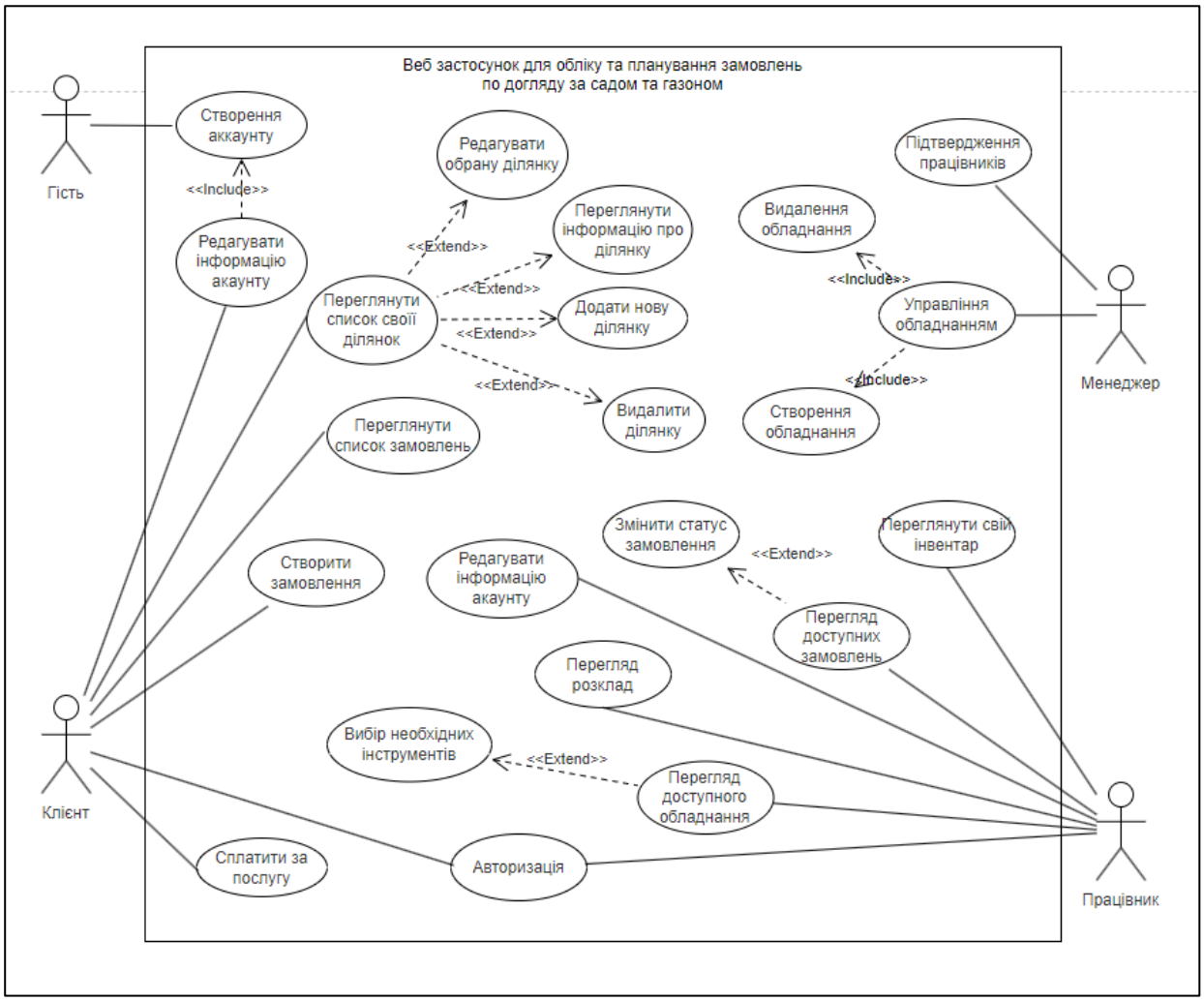

Рисунок 3.1 – Use-case діаграма веб застосунку

Клієнтська частина веб застосунку GreenFlow буде реалізована на основі компонентної архітектури з використання бібліотеки React. Загальна архітектура проекту буде клієнт-серверною, де клієнтська частина буде виступати у ролі клієнта, що взаємодіє з мікро-сервісами серверної частини через API шлюз. Взаємодія буде відбувати завдяки HTTP запитам REST API з використанням JWT-токенів для їх захисту.

Веб застосунок має 4 типу користувачів, а саме клієнт, працівник, менеджер та гість, який ще не створив свій акаунт на сайті. Клієнт буде мати можливість додати до системи ділянки з усією необхідною інформацію і фото саду чи газону, редагувати інформацію про обрані ділянки чи видаляти їх, він зможе робити замовлення на роботи над обраною ділянкою та сплачувати рахунки за працю через застосунок. Працівник в свою чергу матиме можливість переглядати

доступні та прийняті замовлення, змінювати їх статус й обирати необхідні інструменти зі списку в системі. Крім того, клієнт та працівник зможуть редагувати власну інформацію в налаштуваннях облікового запису.

3.4 Архітектура взаємодії клієнтської та серверної частини

Застосунок обмінюється даними з сервісами через API шлюз. Комунікація відбувається за протоколом HTTPS, запити надсилаються з авторизаційним заголовком, що містить JWT токен таким чином:

- користувач заходить на сайт, frontend перевіряє наявність JWT у localStorage;
- якщо токен є, то запит на перевірку прав доступу надсилається до auth-service;
- при переході на сторінку «Мої сади» — frontend надсилає GET-запит до order-service через gateway;
- відповідь обробляється, виводиться список ділянок користувача.

3.5 Загальна структура системи

Фронтенд-застосунок реалізовано за модульним принципом та включає такі логічні блоки:

- компоненти відображення, які відповідають за відображення інформації у вигляді окремих блоків (наприклад, кнопки, поля вводу, картки садів);
- контейнерні компоненти, які відповідають за логіку взаємодії з API, управління станом та передавання даних у дочірні компоненти;
- роутинг реалізований через бібліотеку React Router v6 з підтримкою захищених маршрутів для кожної ролі користувача (клієнт, працівник, менеджер);
- сторінки сформовані, як функціональні компоненти та відповідають конкретним сценаріям використання (особистий кабінет, замовлення, налаштування тощо) ;

– глобальні компоненти Navbar, Footer, глобальні повідомлення про помилки, редиректи при втраті сесії.

3.6 Валідація форм

Валідація форм дозволяє значно знизити ризики помилок з боку користувача, тому усі основні форми системи, зокрема авторизація, регістрація, створення замовлення, запиту на обладнання супроводжуються перевіркою введених користувачем даних. Валідація виконується ще до відправлення запиту на сервер у клієнтській частині. Для прикладу розглянемо код валідації форми регістрації користувача (див. додаток А)

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Вибір технологій для розробки клієнтської частини

У процесі розробки вебінтерфейсу для веб застосунку для обліку та планування замовлень по догляду за садом та газоном було прийнято рішення використовувати технології, що забезпечують високу швидкодію, масштабованість, зручність у підтримці та гнучкість інтеграції з бекендом системи.

Ключовим інструментом для реалізації клієнтської частини обрано бібліотеку React.js. Це сучасна JavaScript-бібліотека з відкритим кодом, розроблена компанією Meta, яка дозволяє створювати реактивні, компонентні інтерфейси користувача. React забезпечує ефективне оновлення стану інтерфейсу завдяки віртуальному DOM, що суттєво зменшує навантаження на браузер та покращує продуктивність вебзастосунку.

Основні переваги, які зумовили вибір React:

- Компонентний підхід, що дозволяє розбити складний інтерфейс на незалежні, багаторазові модулі
- Односторонній потік даних, який спрощує відстеження стану та логіку роботи інтерфейсу.
- Активна спільнота та велика екосистема бібліотек, що прискорює розробку.
- Легка інтеграція з REST API, що дозволяє гнучко взаємодіяти з бекендом.

Одним із створених компонентів була навігаційна панель сайту, яка використовувалась на кожній сторінці сайту завдяки можливостям фреймворку, для прикладу наведено код компоненту Navbar (див. додаток Б).

Для керування маршрутизацією в односторінковому застосунку використовувався React Router, що дозволяє реалізувати переходи між сторінками без повного перезавантаження. Це підвищує зручність користувача та дозволяє зберігати стан між переходами.

Керування станом локальних компонентів було реалізовано за допомогою вбудованих хуків React, таких як useState, useEffect та useRef. Це забезпечило високу гнучкість і простоту у створенні динамічних інтерфейсів.

Для HTTP-запитів до серверної частини використовувалась бібліотека Axios, яка забезпечує зручний синтаксис, підтримку асинхронних запитів та обробку токенів автентифікації. Зокрема, Axios активно застосовується для:

- отримання списку садів, послуг, замовлень;
- створення садів і замовлень;
- завантаження фотографій (окремим запитом для кожного зображення);
- оновлення даних профілю користувача та поповнення балансу.

Окрім того, особливу увагу було приділено адаптивності інтерфейсу. Використовуючи гнучкі CSS Flexbox-структури та медіазапити, інтерфейс було оптимізовано для використання на різних пристроях — від ноутбуків до мобільних телефонів.

Таким чином, набір прийнятих технологічних рішень на фронтенді забезпечив ефективну взаємодію користувача з системою, інтуїтивно зрозумілий інтерфейс та високу швидкодію всіх основних операцій.

4.2 Вибір архітектурного підходу та технологій для реалізації клієнтської частини

Під час проєктування клієнтської частини вебзастосунку для сервісу з догляду за садами та газонами було прийнято рішення реалізувати фронтенд у вигляді односторінкового застосунку (SPA — Single Page Application). Такий підхід дозволяє досягти високої інтерактивності, плавності користувацького досвіду та швидкості реакції інтерфейсу, оскільки взаємодія з сервером здійснюється асинхронно, без повного перезавантаження сторінки.

Для реалізації SPA обрано як раз бібліотеку React. Вона дозволяє ефективно керувати станом компонентів, розбивати інтерфейс на окремі ізольовані частини, повторно використовувати код і забезпечує високу продуктивність завдяки використанню віртуального DOM.

Для навігації між сторінками застосунку використовується бібліотека React Router, яка надає можливість визначати маршрути, пов’язувати їх з компонентами і підтримувати історію переходів. Це дозволяє гнучко організувати структуру інтерфейсу з підтримкою глибоких посилань (deep linking), що важливо для систем з особистим кабінетом користувача, динамічними таблицями, фільтрами та сторінками з параметрами.

У якості стилізації було прийнято рішення використовувати кастомну CSS-структуру з розділенням на глобальні стилі, стилі для компонентів, а також змінні кольорів, шрифтів і відступів для забезпечення візуальної консистентності та зручності підтримки коду. Крім того, були використані адаптивні стилі для підтримки різних розмірів екрану.

Клієнтська частина побудована з урахуванням взаємодії з бекендом, що реалізований на основі REST API. Для здійснення HTTP-запитів застосовується бібліотека Axios, яка підтримує обробку заголовків авторизації, обробку помилок і зручну роботу з промісами. Запити надсилаються з токеном автентифікації, що зберігається в localStorage після входу користувача.

Було реалізовано базову архітектуру компонентів, яка враховує відокремлення бізнес-логіки від представлення (UI). Основні логічні блоки винесено в окремі функціональні компоненти з використанням React-хуків, таких як useState, useEffect, useRef. Це дозволяє підтримувати чистоту коду, уникати дублікацій та покращити тестованість.

В якості засобів розробки використано Vite для створення проєкту, що забезпечує швидкий запуск дев-сервера та мінімальний час збірки. Усі компоненти системи розташовано в окремих папках відповідно до їх функціональності, наприклад: components, pages, styles, utils.

4.3 Інтеграція з бекендом та обробка даних

Одним із ключових аспектів реалізації фронтенд-частини системи стала тісна інтеграція з бекенд-сервісами через REST API. Всі основні функції, пов’язані з відображенням і взаємодією з даними, реалізовані за допомогою HTTP-запитів

до відповідних кінцевих точок API, які були спроектовані для окремих мікросервісів. Обмін даними відбувається у форматі JSON, що є стандартом для сучасних вебзастосунків.

Для виконання запитів до серверу у React-застосунку використовувалась бібліотека axios, яка дозволяє просто і гнучко керувати асинхронними запитами. Наприклад, при створенні нового саду користувачем, після валідації форми, здійснюється POST-запит до /api/v1/garden, у якому передаються дані, зібрані з відповідних полів. Успішна відповідь від бекенду містить ідентифікатор створеного об’єкта, що використовується для подальших дій — наприклад, завантаження зображень або редагування. Для прикладу розглянемо код асинхронної функції, що відправляє запит на створення саду до сервера:

```
async function createGarden(e) {
        e.preventDefault();
        if (validateForm()) {
            try {
                const response = await
axios.post('https://api.greenflow.software/api/v1/garden', garden, {
                    headers: { Authorization: `Bearer ${token}` }
                });
                console.log(response.data)
                const newGardenId = response.data.id;
                setCreatedGardenId(newGardenId);
                await uploadImages(newGardenId);
                window.location.href = "/my-gardens";
            } catch (err) {
                console.error("Помилка створення саду:", err);
            }
        }
    }
```

У всіх запитах передбачена передача токена автентифікації у заголовку Authorization. Цей токен зберігається у локальному сховищі браузера (localStorage) після успішного входу користувача, і автоматично додається до всіх

запитів через відповідну логіку у функціях виклику API. Це забезпечує контроль доступу до захищених маршрутів, зокрема — перегляд власних замовлень, створення послуг, управління садом тощо.

Особливу увагу було приділено обробці асинхронних запитів та управлінню станом інтерфейсу. У React-компонентах активно використовувалися хуки useState та useEffect: перший — для збереження та оновлення стану, другий — для ініціалізації завантаження даних при монтуванні компонента. Наприклад, при відкритті сторінки “Мої сади” автоматично виконується завантаження списку садів через GET-запит до відповідного API-методу, після чого результати відображаються на сторінці.

У складніших випадках, таких як зв’язування кількох сутностей (наприклад, замовлення + сад або техніка + оренда), реалізовувалось каскадне отримання даних: спочатку завантажувався масив ідентифікаторів, а потім для кожного з них виконувались окремі запити для отримання повної інформації. Для цього використовувався Promise.all, що дозволяло одночасно обробляти масив запитів без надмірного ускладнення логіки.

Таким чином, взаємодія фронтенду з бекендом побудована на принципах модульності, безпеки та масштабованості, що забезпечує узгоджену роботу між інтерфейсом користувача та внутрішньою логікою системи. Усі запити протестовані як у браузері, так і у середовищі Postman, що гарантує стабільну роботу застосунку.

4.4 Реалізація завантаження та редагування зображень

Однією з важливих функціональних можливостей фронтенд-частини системи стала підтримка завантаження зображень для садів користувача. Користувач має змогу при створенні саду додати до 12 зображень, які ілюструють ділянку (див. рисунок 4.1), а також у подальшому редагувати їх через окремий інтерфейс. Реалізація цієї функції вимагала розробки інтерактивного елементу з можливістю попереднього перегляду файлів до відправки на сервер, завантаження кожного файлу окремим запитом, а також надійної обробки форматів та помилок.

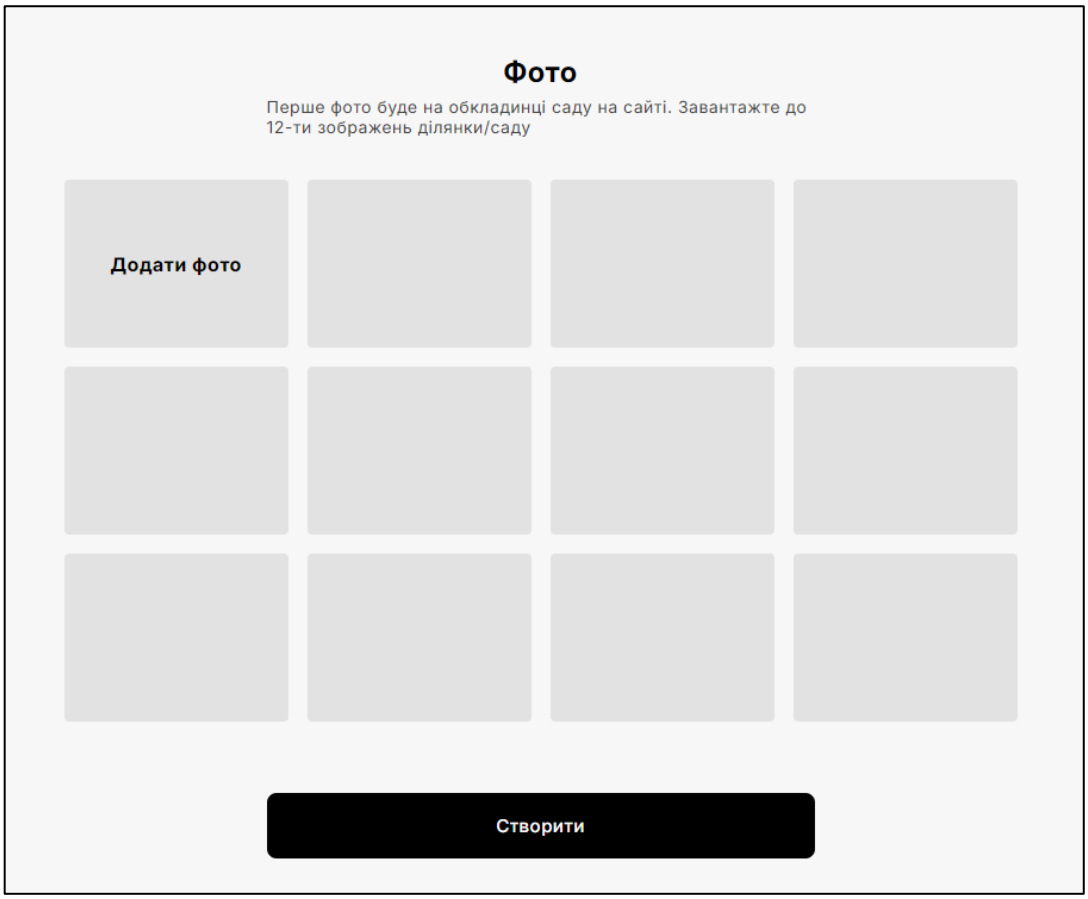

Рисунок 4.1 – Елемент для завантаження фото саду

Основним елементом для завантаження зображень став PhotoUploader — візуальний компонент, реалізований як сітка з 12 плиток, кожна з яких представляє одну позицію для зображення Перший клік по порожній плитці відкриває діалог вибору файлу за допомогою прихованого input type="file", який прив’язаний через useRef. Обрана фотографія відображається у відповідному вікні плитки завдяки створенню локального тимчасового URL через URL.createObjectURL.

Після заповнення всіх необхідних даних про сад користувач натискає кнопку “Створити”, яка викликає функцію createGarden. У разі успішного створення саду, бекенд повертає ідентифікатор нового запису, який передається у функцію uploadImages. Ця функція формує FormData для кожного зображення окремо та надсилає їх через POST-запити за допомогою axios до маршруту /api/v1/garden/images?gardenId=ID. Завантаження кожного файлу відбувається по черзі, що дозволяє відслідковувати помилки на рівні кожного запиту.

Для захисту та авторизації кожен запит містить JWT-токен у заголовку Authorization, що забезпечує обмеження доступу тільки для авторизованих користувачів. У випадку помилки (наприклад, перевищення розміру файлу, CORS-помилка або неправильний формат) система інформує користувача через повідомлення, відображене на інтерфейсі.

Окрему увагу було приділено реалізації функціоналу редагування фото. На сторінці GardenPage реалізовано модальне вікно, яке відкривається при натисканні кнопки “Редагувати фото”. Вікно містить інтерфейс, ідентичний компоненту PhotoUploader, але ініціалізується даними із garden.imagesUrl — тобто вже завантаженими фото(див. рисунок 4.2).

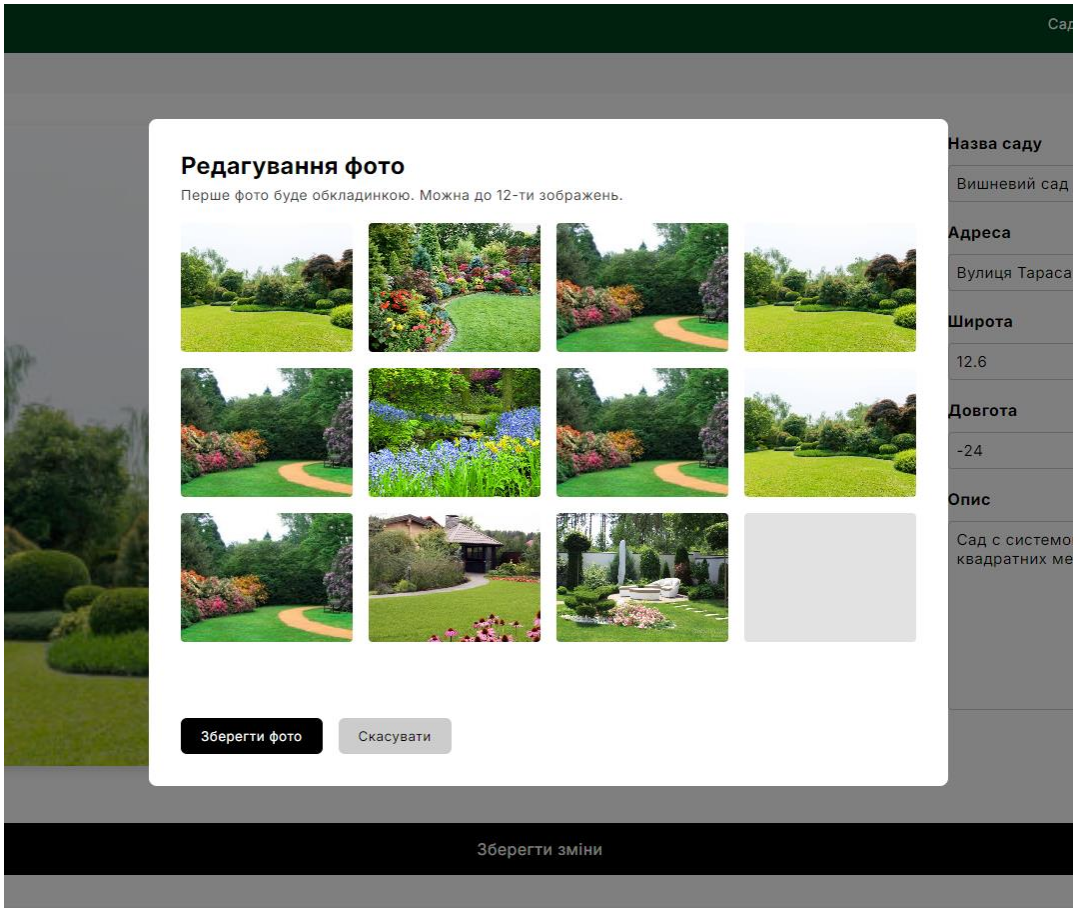

Рисунок 4.2 – Компонент PhotoUploader для редагування фото.

Користувач може змінити окремі зображення, після чого натискає кнопку “Оновити фото”, що повторно запускає процес відправки змінених файлів на сервер.

4.5 Інтерграція з геолокаційними сервісами для визначення місцезнаходження користувача

Для забезпечення функціональності пошуку найближчих замовлень та обладнання, яке можна взяти в оренду, було реалізовано інтеграцію з вбудованими можливостями геолокації браузера (див. рисунок 4.3). Ця інтеграція дозволяє в режимі реального часу визначати координати користувача (широту та довготу) та використовувати їх як параметри при запитах до бекенд-сервісів. Такий підхід покращує персоналізацію системи та дозволяє відображати тільки ті об’єкти, які справді розташовані у фізичній близькості до користувача.

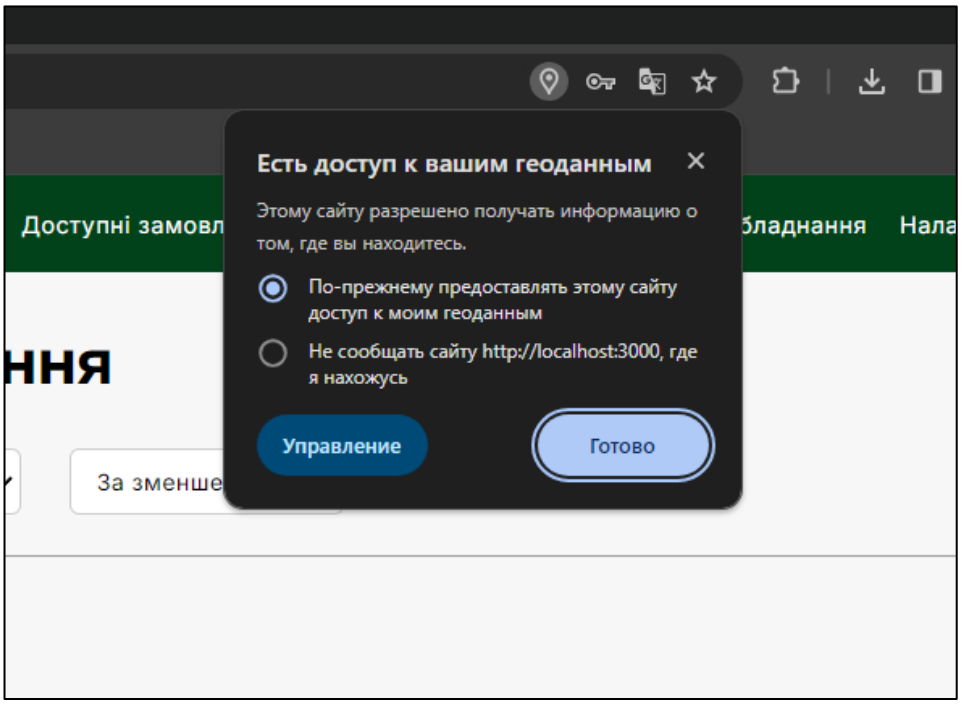

Рисунок 4.3 – Запит від браузера на використання геолокації.

Реалізація цієї можливості здійснюється через стандартне API navigator.geolocation, яке підтримується сучасними браузерами. Після завантаження компонента (наприклад, Leasing.jsx) викликається метод navigator.geolocation.getCurrentPosition:

```
useEffect(() => {
        navigator.geolocation.getCurrentPosition(
```

```
            position => {
                setCoords({
                    latitude: position.coords.latitude,
                    longitude: position.coords.longitude
                });
                console.log("lat:", position.coords.latitude,);
                console.log("lon:", position.coords.longitude);
            },
            error => {
                console.error("Geolocation error:", error);
                setError(error)
            }
        );
    }, []);
```

У разі успішного отримання географічних координат вони зберігаються в локальному стані компонента через useState (змінна coords). Якщо ж користувач не надає дозвіл на доступ до геолокації, інтерфейс виводить відповідне повідомлення з інструкціями.

Дані координати використовуються для формування запиту до бекенд-методу /api/v1/equipment/search або /api/v1/open-order з передачою широти та довготи у заголовках запиту (X-User-Latitude, X-User-Longitude). Такий підхід забезпечує гнучкість на бекенді й дозволяє уникнути передачі конфіденційних даних через URL-параметри.

Крім того, в інтерфейсі передбачено можливість фільтрації результатів за радіусом пошуку (у кілометрах), що встановлюється користувачем вручну через числове поле input. Реалізовано валідацію введення: значення радіусу не може бути менше 1 км, а зміни параметру пошуку автоматично ініціюють оновлення запиту на сервер.

Обробка помилок здійснюється як на клієнтському, так і на серверному рівнях. Наприклад, якщо геолокація не підтримується браузером або запит за координатами завершився з помилкою, користувачу виводиться зрозуміле текстове повідомлення.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Загальний підхід до тестування користувацького інтерфейсу

На етапі тестування веб-додатку особлива увага приділялася перевірці коректності роботи інтерфейсу користувача, взаємодії з серверною частиною через API, а також відповідності інтерфейсу очікуваному користувацькому досвіду. Тестування здійснювалося вручну на різних етапах розробки, із застосуванням інструментів веб-розробника в браузері Google Chrome та з використанням розширення Postman для перевірки правильності HTTP-запитів.

Оскільки веб-додаток побудований на основі бібліотеки React, яка є компонентно-орієнтованою, окремі UI-компоненти були перевірені на предмет:

- правильного відображення вхідних даних (props) ;
- обробки подій (onClick, onChange) ;
- змін локального стану (useState) ;
- динамічного оновлення даних (через useEffect) ;
- валідації форм користувача (в тому числі виведення повідомлень про помилки).

Основні сценарії ручного тестування включали:

- заповнення форми створення саду, перевірка коректності полів, відправка даних, обробка відповіді;
- завантаження зображень до саду: додавання, перегляд, видалення;
- використання геолокації для пошуку доступного обладнання;
- перевірка роботи випадаючих списків, кнопок, модальних вікон;
- оформлення оренди обладнання, перевірка правильності збережених даних;
- поповнення балансу користувача та перенаправлення на зовнішній платіжний сервіс;
- коректність відображення змін після оновлення профілю;

– валідація полів на помилки введення (порожні поля, некоректні формати координат, електронної пошти, паролів тощо);
– правильність маршрутизації між сторінками додатку.

Для прикладу наведемо процес тестування валідації форм на сторінці регістрації аккаунту. Для першого випадку тестування введемо невірний формат електронної адреси – testgmail.com. В результаті отримаємо повідомлення про невірний формат пошти (див. рисунок 5.1). У випадку введення занадто простого та короткого паролю користувач також отримує відповідне повідомлення.

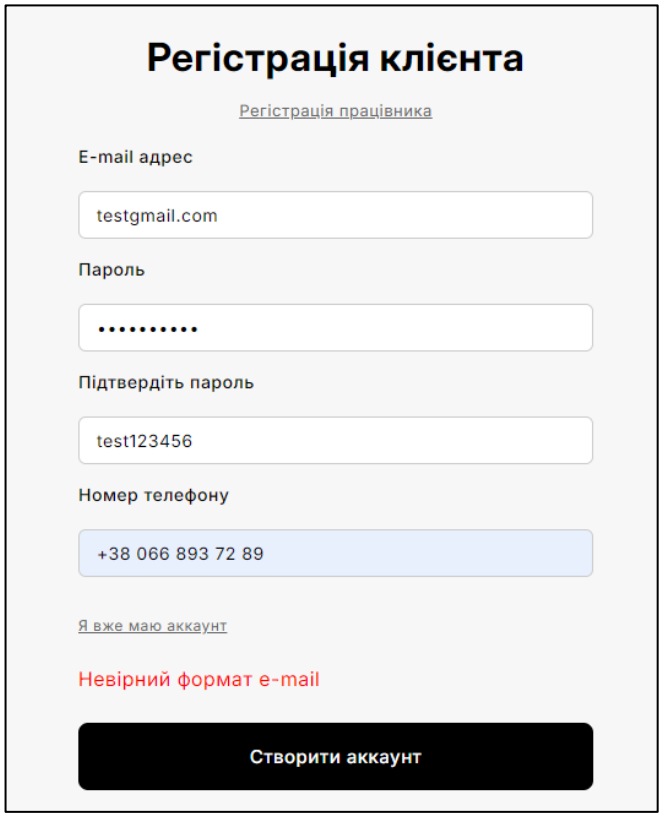

Рисунок 5.1 – Перевірка валідації форми при невірному форматі e-mail.

У випадку введення занадто простого та короткого паролю користувач також отримує відповідне повідомлення (див. рисунок 5.2).

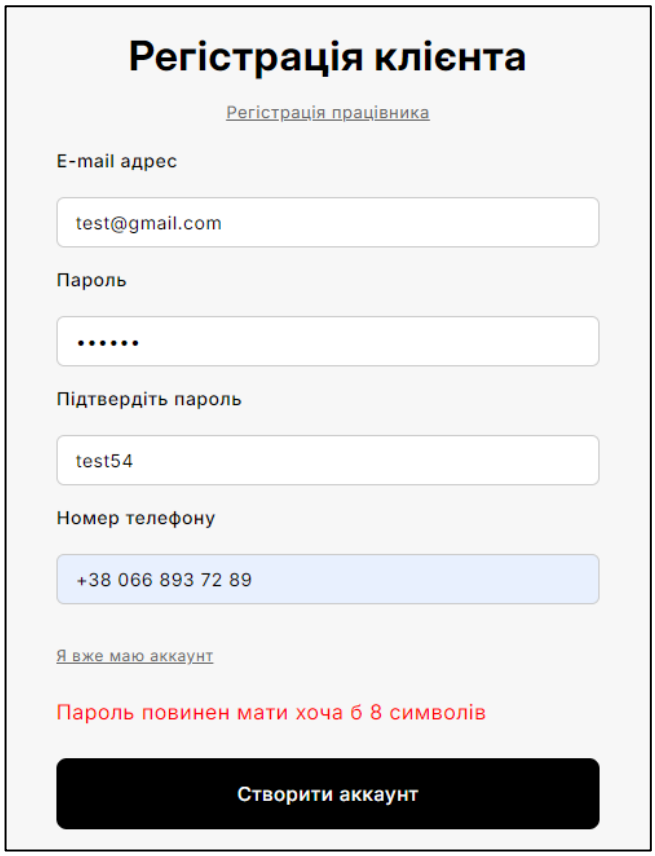

Рисунок 5.2 – Перевірка валідації форми при невірному форматі e-mail.

Крім того була протестована перевірка на заповнення усіх полів форм, співпадіння першого та другого пароля та номеру телефону.

Головною метою тестування було забезпечення стабільної роботи додатку, передбачуваної поведінки кожного компонента, зручності користування інтерфейсом та дотримання стандартів взаємодії із серверною частиною. Усі критично важливі функції були перевірені неодноразово в реальних сценаріях використання.

5.2 Тестування модуля завантаження та перегляду зображень

Модуль завантаження зображень є важливою складовою користувацького інтерфейсу системи, яка забезпечує можливість прикріплення фотографій до об’єктів типу «сад». Тестування модуля проводилося з метою перевірки стабільності, коректності обробки зображень, відповідності обмеженням (до 12 зображень), а також інтеграції з серверною частиною.

Тестування виконувалось вручну в браузері Google Chrome, а також перевірялось у Firefox та Safari з метою виявлення можливих проблем кросбраузерності.

Було протестовано обмеження кількості зображень. При спробі додати більше ніж 12 фото, система коректно ігнорує додаткові файли, не викликаючи помилок у інтерфейсі або на сервері.

Використання кнопок для перегляду поточних фото саду. При спробі натискання на кнопку «прогорнути вправо» «прогорнути вліво» зображення з галереї змінювалось на наступне та попереднє відповідно (див. рисунок 5.3).

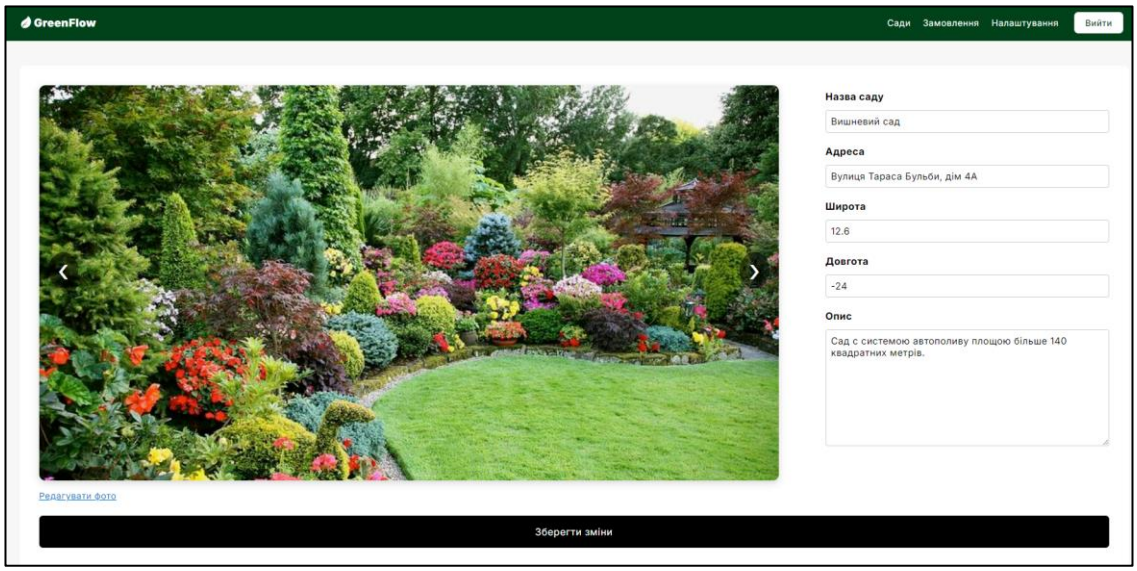

Рисунок 5.3 – Галерея фотографій на сторінці саду.

Відображення попереднього перегляду. Після вибору файлу локального зображення, у відповідній клітинці відображається його мініатюра. Було перевірено, що зображення з різним розширенням (.jpg, .png, .webp) коректно обробляються браузером та відображаються користувачеві (див. рисунок 4.2).

Заміна зображень. При кліку по завантаженому зображенню відкривається вікно вибору нового файлу, що дозволяє оновити наявне фото (див. рисунок 5.4). Ця дія була протестована на різних комірках, включно з першою (яка є обкладинкою).

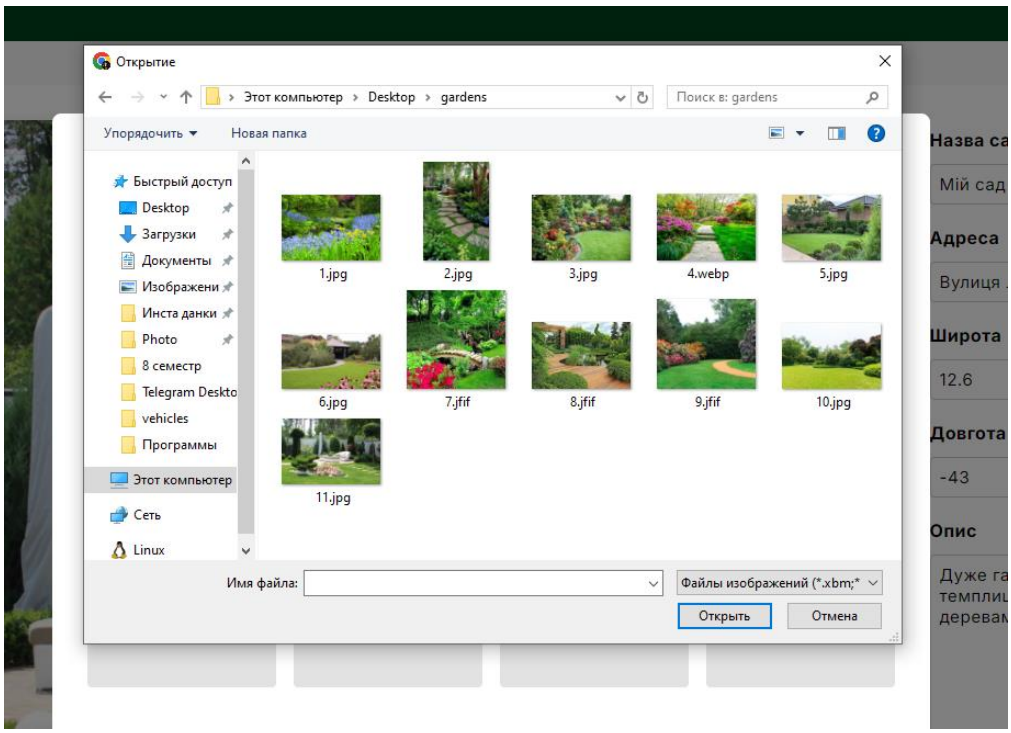

Рисунок 5.4 – Діалогове вікно для завантаження файлу

Відправка зображень на сервер. Після натискання кнопки створення саду, зображення передаються через окремі POST-запити за допомогою FormData з полем file. Була протестована передача різного розміру зображень (до 5 МБ) та знайдено недолік на стороні сервера, де він не приймає файли розміром більші ніж 1 МБ за фото.

Обробка помилок. Імітувалися ситуації втрати підключення до інтернету, відмови сервера (відповідь 500) та передавання надто великого файлу (відповідь 413 Payload Too Large (див. рисунок 5.5)). У всіх випадках було перевірено, що користувач отримує відповідне повідомлення про помилку і може повторити завантаження.

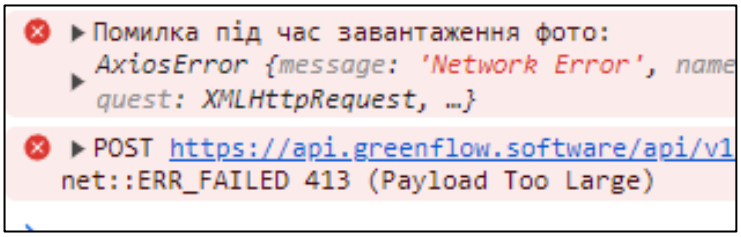

Рисунок 5.5 – Помилка при відправці завеликого файлу (3 МБ) на сервер

UX-досвід. Перевірялась інтуїтивність взаємодії користувача з елементами завантаження фото, зокрема відображення підказок, вказівок щодо обкладинки, реакції на кліки по порожніх клітинках (див. рисунок 5.6).

Рисунок 5.6 – Зовнішній вид клітинки для фото при наведенні курсора

За результатами тестування було підтверджено стабільну роботу модуля, відповідність функціоналу вимогам користувача та інтеграцію з API бекенда. Жодних критичних помилок виявлено не було. У разі неуспішної передачі файлу система дозволяє повторне завантаження.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Підготовка до розгортання клієнтської частини

Перед впровадженням клієнтської частини вебзастосунку необхідно було виконати низку дій, пов’язаних із підготовкою середовища, оптимізацією коду, налаштуванням збірки та перевіркою працездатності застосунку в умовах продуктивного оточення.

У процесі підготовки були враховані такі ключові етапи:

а) оптимізація проєкту для production-режиму:

1) команда npm run build використовується для створення фінальної збірки застосунку. Після виконання цієї команди створюється оптимізована версія React-застосунку у вигляді статичних файлів у каталозі /build.

2) до збірки включено мініфікацію JavaScript- та CSS-коду, що зменшує обсяг передаваних файлів і прискорює завантаження сторінок.

3) механізм кешування забезпечує збереження частини ресурсів на стороні клієнта, що зменшує кількість запитів до сервера під час повторних відвідувань.

б) налаштування змінних середовища:

1) для зручності зміни базового URL API використано змінні середовища REACT_APP_API_URL

2) у файлі .env.production визначено адресу серверного API, яка відрізняється від тієї, що використовується у development-режимі.

в) перевірка працездатності після збірки:

1) після створення збірки тестовий запуск здійснено за допомогою пакету serve (npm install -g serve).

2) команда serve -s build дала змогу локально перевірити коректність переходів між сторінками, доступність ресурсів, правильність обробки запитів до бекенду.

3) особливу увагу приділено перевірці динамічних маршрутів (наприклад, /my-gardens/:id) та взаємодії з API через JWT.

г) підготовка до деплою:

1) усі статичні файли після збірки призначені для розгортання на вебсервері.

2) під час налаштування вебсервера необхідно передбачити перенаправлення всіх запитів на index.html для підтримки маршрутизації React (режим BrowserRouter).

3) захист CORS був налаштований на стороні бекенду, тому клієнтська частина працює у доменах, дозволених сервером.

д) забезпечення безпеки:

1) враховано рекомендації OWASP щодо безпечної розробки SPA, наприклад, запобігання XSS через екранування даних, використання HTTPS, контроль доступу на рівні інтерфейсу.

2) усі запити до захищених ресурсів містять токен авторизації у заголовку Authorization: Bearer <JWT>.

У результаті підготовки було отримано готову до розгортання production-версію клієнтської частини, яка відповідає технічним вимогам, має зручну архітектуру та забезпечує надійну взаємодію з сервером.

6.2 Визначення плану впровадження

Після успішного створення production-збірки клієнтської частини застосунку планується здійснення її розгортання на віддаленому сервері для забезпечення доступу користувачів до системи через вебінтерфейс. У процесі розгортання буде використано сучасні підходи до публікації React-застосунків на вебсервері з урахуванням особливостей односторінкової архітектури (SPA).

Для публікації клієнтської частини було обрано хостинг на базі VPS (Virtual Private Server), що дозволяє гнучко налаштувати середовище, встановити власні правила безпеки та забезпечити масштабування.

Операційна система сервера — Ubuntu Server 22.04. У якості вебсервера використано Nginx. Підготовлючи середовище на сервері було встановлено Node.js (версія 18 LTS) та менеджер пакетів npm. У каталозі проєкту виконано

команду npm run build, у результаті чого було згенеровано папку /build із production-збіркою.

Встановлено та налаштовано вебсервер Nginx, який виступає у ролі статичного сервера для React-застосунку.

ВИСНОВКИ

У межах виконання дипломного проекту було розроблено клієнтську частину вебзастосунку, призначеного для автоматизації процесів замовлення, надання та управління послугами з догляду за садами та газонами. Основна увага приділялася створенню зручного, функціонального та адаптивного інтерфейсу користувача, який забезпечує ефективну взаємодію між клієнтами, працівниками та адміністраторами системи.

На етапі аналізу було сформовано та систематизовано функціональні й нефункціональні вимоги до інтерфейсу, які стали основою для архітектурного та компонентного проектування. З урахуванням сучасних тенденцій веброзробки було обрано стек технологій, зокрема бібліотеку React, яка дозволила реалізувати динамічні компоненти, ефективну маршрутизацію та керування станом застосунку.

Під час розробки було реалізовано ключові функціональні модулі клієнтської частини:

- модуль створення та редагування замовлень користувача;
- динамічне відображення садів і послуг із можливістю взаємодії;
- інтерфейс для працівників із переглядом доступних замовлень та можливістю їхнього прийняття;
- модуль оренди техніки з геолокаційним пошуком та фільтрацією;
- інтерфейс керування балансом користувача з інтеграцією платіжної системи;
- функціональність завантаження та редагування фотографій саду;
- особистий кабінет із можливістю оновлення профільної інформації.

Клієнтська частина побудована з дотриманням принципів компонентного підходу, що забезпечує її масштабованість, розширюваність та легкість у супроводі. Для забезпечення надійної роботи із зовнішніми API було реалізовано асинхронні запити за допомогою бібліотеки Axios, а також передбачено обробку помилок і перевірку авторизаційного токена користувача.

Після завершення етапу реалізації було проведено модульне тестування окремих частин інтерфейсу, а також комплексну перевірку взаємодії між фронтендом і бекендом. Результати тестування підтвердили коректність функціонування усіх передбачених сценаріїв користувацької поведінки.

Фінальним етапом стане успішне розгортання клієнтської частини на віддаленому сервері з використанням Nginx як статичного вебсервера, налаштуванням маршрутизації для SPA-застосунку та забезпеченням захищеного з’єднання через HTTPS. Таким чином, поставлену мету розробки клієнтської частини вебсистеми для сервісу з догляду за садами та газонами було досягнуто повною мірою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. React Documentation [Електронний ресурс] – URL: https://react.dev/learn (дата звернення: 10.05.2025)

2. Component Architecture (Hands on React) [Електронний ресурс] – URL: https://handsonreact.com/docs/component-architecture (дата звернення: 10.05.2025)

3. Frontend Integration (Microservice API Patterns) [Електронний ресурс] – URL: https://microservice-api-patterns.org/patterns/foundation/FrontendIntegration (дата звернення: 10.05.2025)

4. SPA: The Ultimate Guide for Single Page Applications [Електронний ресурс] – URL: https://www.rapidevelopers.com/blog/spa-the-ultimate-guide-for-single-page-applications (дата звернення: 10.05.2025)

5. Essential Principles for Brilliant Web App Design [Електронний ресурс] – URL: https://www.hotjar.com/web-app-design/principles (дата звернення: 10.05.2025)

6. GitHub репозиторій [Електронний ресурс] – URL: https://github.com/NureKoshelDavid/2025_B_PI_PZPI-21-7_Koshel_D_Y (дата звернення: 10.06.2025)