Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка автоматизованої системи моніторингу стану дерев з використанням Google Maps. Частина 2. Клієнтська частина»

Виконав: студент 4 курсу
групи ЗПІ-20б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Підгорний І.М.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Войтко В.В.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Тарновський М. Г
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«10» червня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – «Інформаційні технології»
Спеціальність 121 – «Інженерія програмного забезпечення»
Освітньо-професійна програма – «Інженерія програмного забезпечення»

УЗГОДЖЕНО
НАЧАЛЬНИК ДЕРЖАВНОЇ
ЕКОЛОГІЧНОЇ ІНСПЕКЦІЇ У
ВІННИЦЬКІЙ ОБЛАСТІ
Дубовий Ю.В.
« 12 » березня 2024 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н._____
« 12 » березня 2024 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Підгорному Ігорю Миколайовичу

1. Тема роботи – «Розробка автоматизованої системи моніторингу стану дерев з використанням Google Maps. Частина 2. Клієнтська частина»

Керівник роботи: Войтко Вікторія Володимирівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80.

2. Строк подання студентом роботи 3 червня 2024 р.

3. Вихідні дані до роботи: середовище розробки Visual Studio Code, мова розробки TypeScript, фреймворк React, Google Maps API.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану питання розробки; порівняльний аналіз аналогів; аналіз методів розв'язання задачі; постановка задач дослідження; аналіз даних, розробка структури клієнтської частини, розробка моделі клієнтської частини, розробка методу позначення дерев на карті; варіантний аналіз та вибір мови програмування, варіантний аналіз та вибір середовища розробки, розробка клієнтської частини модуля авторизації, розробка клієнтської частини модуля роботи з деревами; аналіз методів тестування, тестування розробленого програмного застосунку; висновки; список використаних

джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; ме об'єкт та предмет дослідження; задачі дослідження, новизна, практична цінні одержаних результатів; порівняльний аналіз аналогів, використані техноло метод позначення дерев на карті, тестування, висновки, апробація результа роботи і публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Войтко В.В., к.т.н., доцент кафедри ПЗ	13.03.24	03.06.24

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання розробки клієнтської частини системи	14.03.2024 – 24.03.2024	Вик.
2	Розробка структури клієнтської частини	25.03.2024 – 07.04.2024	Вик.
3	Варіантний аналіз та вибір засобів розробки	08.04.2024 – 18.04.2024	Вик.
4	Розробка клієнтської частини системи моніторингу стану дерев	19.04.2024 – 19.05.2024	Вик.
5	Тестування клієнтської частини	20.05.2024 – 24.05.2024	Вик.
6	Оформлення матеріалів до захисту БДР	25.05.2024 – 30.05.24	Вик.

Студент _____ (підпис) Підгорний І.М. (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ (підпис) Войтко В.В. (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 59 сторінок формату А4, на яких є 41 рисунків, 4 таблиці, список використаних джерел містить 21 найменування.

У бакалаврській дипломній роботі проведено аналіз стану розвитку систем для моніторингу стану дерев. Було проведено аналіз аналогів, визначено їх переваги і недоліки та доведено доцільність розробки власної системи.

Виконане обґрунтування вибору мови та середовища програмування, а також технологій, які були використані під час розробки клієнтської частини системи.

Розроблено клієнтську частину системи, що дозволяє вести моніторинг стану дерев та проводити догляд за ними.

Клієнтську частину реалізовано за допомогою мови програмування TypeScript та фреймворку React. Для розробки клієнтської частини було використано середовище розробки Visual Studio Code.

Отримані в бакалаврській дипломній роботі результати можна використати для збору даних про стан дерев, ведення їх моніторингу, автоматизації обліку та підтримкою їх стану.

Ключові слова: автоматизована система, моніторинг, Google Maps.

ANNOTATION

The bachelor's thesis consists of 59 pages of A4 format, which contain 41 drawings, 4 tables, the list of sources used contains 21 name.

In the bachelor's thesis, an analysis of the state of development of systems for monitoring the condition of trees was analyzed. Analogs were analyzed, their advantages and disadvantages were determined and the feasibility of developing their own system was proved.

The justification for the choice of language and programming environment, as well as technologies that were used in the development of the client part of the system.

The client part of the system has been developed to monitor and care for trees.

The server part is implemented using the TypeScript programming language and the React framework. Visual Studio Code development was used to develop the client part.

The results obtained from the bachelor's thesis can be used to collect trees on the status of trees, keep their monitoring, automate accounting and maintain their condition.

Keywords: automated system, monitoring, Google Maps.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Веб-розробка дозволяє створити інтерактивні візуалізації стану дерев, наприклад, інтерактивні карти або графіки. Це полегшує інтерпретацію даних та прийняття обґрунтованих рішень щодо догляду за деревами.

Системи для моніторингу стану дерев є важливим інструментом для управління зеленими насадженнями в міськиcх та природних територіях. Ці системи допомагають визначити стан здоров'я дерев, виявити хвороби та шкідників, а також контролювати ріст та розвиток дерев [1].

Одним з видів систем для моніторингу стану дерев є датчики, які встановлюються на деревах та збирають дані про їх стан. Ці датчики можуть вимірювати такі параметри, як температура, вологість, рівень освітленості, рівень кисню, рівень вуглекислого газу та інші. Дані, які збирають датчики, можуть бути передані на віддалене місце для аналізу та обробки.

Іншим видом систем для моніторингу стану дерев є візуальний моніторинг, який проводиться фахівцями з допомогою спеціальних приладів, таких як біноклі, телескопи та камери. Цей вид моніторингу дозволяє виявити візуальні ознаки хвороб та шкідників, а також оцінити стан крони дерева та його загальний стан.

Є також системи для моніторингу стану дерев, які використовують супутникові знімки та аерофотознімання. Ці системи дозволяють оцінити стан дерев на великих територіях, виявити зміни в рослинності та визначити райони, де необхідно провести додаткові дослідження.

Системи для моніторингу стану дерев допомагають ефективно керувати зеленими насадженнями, запобігати хворобам та шкідникам, а також підтримувати здоров'я та красу дерев. Вони є невід'ємною частиною сучасного міського ландшафту та важливим інструментом для збереження природних ресурсів.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою бакалаврської дипломної роботи є підвищення можливостей моніторингу стану дерев за рахунок розробки і використання клієнтської частини вебсистеми, яка включає створення графічного інтерфейсу користувача для опису дерев з використанням Google Maps, що дозволить покращити процес моніторингу стану дерев та своєчасного формування рекомендацій щодо підтримки зелених насаджень.

Відповідно до поставленої мети необхідно виконати такі **завдання:**

- розробити модель, метод та алгоритми роботи клієнтської частини вебсистеми;
- розробити модуль реєстрації та авторизації користувача;
- інтегрувати карти Google Maps в систему;
- розробити інтерфейс користувача для моніторингу стану дерев;
- забезпечити відображення дерев на карті;
- провести тестування роботи клієнтської частини вебсистеми.

Об’єктом дослідження є процес розробки клієнтської частини автоматизованої системи моніторингу стану дерев з використанням Google Maps.

Предметом дослідження є методи і засоби реалізації клієнтської частини автоматизованої системи моніторингу стану дерев з використанням Google Maps.

Методи дослідження. Під час дослідження використовувались такі методи:

- методи позначення дерев на картах для візуалізації досліджуваних об’єктів;
- методи розробки веб-застосунків для створення клієнтської частини системи;
- методи інтеграції клієнтської та серверної частин веб-застосунку для розробки цілісної вебсистеми;
- методи тестування веб-застосунків для перевірки працездатності і коректності роботи розробленої клієнтської частини вебсистеми.

Новизна отриманих результатів:

1. Подальшого розвитку отримав метод позначення дерев на карті, в якому, на відміну від відомих, дерева мають позначення різного розміру залежно від їх віку та розміру, що дозволяє наочно відрізнити старші та більші дерева від молодших та

менших у розмірі.

2. Подальшого розвитку набула модель клієнтської частини, яка, на відміну від існуючих, базується на фреймворку React та Google Maps API, що забезпечує доступ до актуальних карт місцевості, надає зручний інтерфейс для ефективного та швидкого ведення моніторингу стану дерев.

Практична цінність отриманих результатів. Практична цінність полягає у можливості використання розробленої клієнтської частини із серверною частиною, зручному інтерфейсі для ефективного ведення обліку дерев, моніторингу їх стану та проведення догляду за ними.

Особистий внесок здобувача. Усі результати, викладені в бакалаврській дипломній роботі, отримані автором особисто. У роботі [3], виконаній у співавторстві, автору належить блок-схема алгоритму роботи системи. У роботі [4], виконаній у співавторстві, автору належить модель CI/CD процесів автоматизованої системи моніторингу стану дерев.

Апробація результатів роботи. Описані у роботі положення доповідались на конференціях «LIII Науково-технічна конференція підрозділів Вінницького національного технічного університету (2024)», «PERSPECTIVES OF CONTEMPORARY SCIENCE: THEORY AND PRACTICE Proceedings of IV International Scientific and Practical Conference Lviv, Ukraine 26-28 May 2024».

Публікації. Результати роботи були опубліковані в матеріалах конференцій: LIII Науково-технічна конференція підрозділів Вінницького національного технічного університету (2024) [3] та «PERSPECTIVES OF CONTEMPORARY SCIENCE: THEORY AND PRACTICE Proceedings of IV International Scientific and Practical Conference Lviv, Ukraine 26-28 May 2024» [4].

Аналіз. У пояснювальній записці до бакалаврської дипломної роботи було розглянуто 4 розділи та було використано 21 літературне джерело.

1 АНАЛІЗ СТАНУ РОЗВИТКУ ПРОГРАМ ДЛЯ МОНІТОРИНГУ СТАНУ ДЕРЕВ

1.1 Аналіз стану питання розробки клієнтської частини системи

Розробка клієнтської частини автоматизованої системи моніторингу стану дерев з використанням Google Maps є складною та цікавою задачею, яка поєднує в собі знання з галузей IT, географії та екології.

По-перше, необхідно зрозуміти, яку мету має на меті ця система та яким чином вона буде використовуватися. Наприклад, система може бути призначена для моніторингу стану дерев у міському парку, лісовому масиві або навіть на всій території країни. Відповідно до цієї мети, необхідно визначити функціональні вимоги до системи, а також вибрати відповідні технології та інструменти для її реалізації.

Одним з ключових компонентів такої системи є картографічна основа, яка дозволяє відображати розташування дерев на карті та аналізувати їх стан. Google Maps є однією з найпопулярніших та найбільш функціональних картів, які можна використовувати для цієї мети (рисунок 1.1). Для розробки клієнтської частини системи необхідмо мати глибокі знання та навички роботи з API Google Maps, а також з іншими географічними сервісами та форматами даних, такими як GeoJSON (рисунок 1.2), KML (рисунок 1.3) та ін.

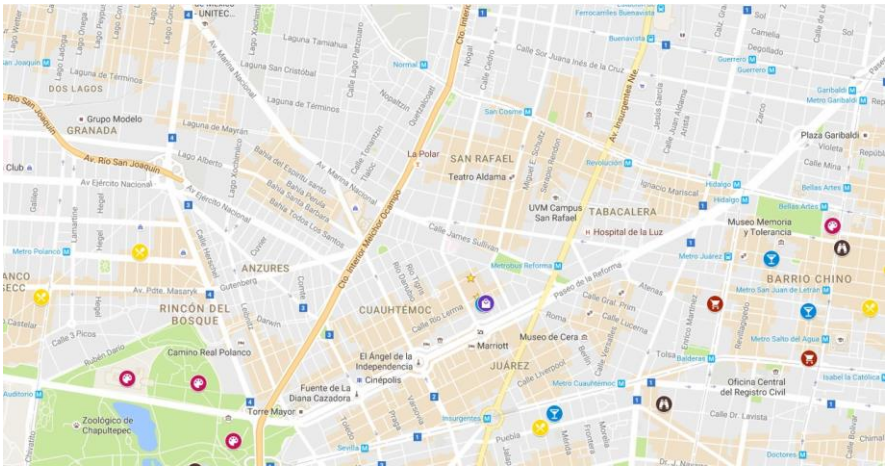

Рисунок 1.1 – Google Map

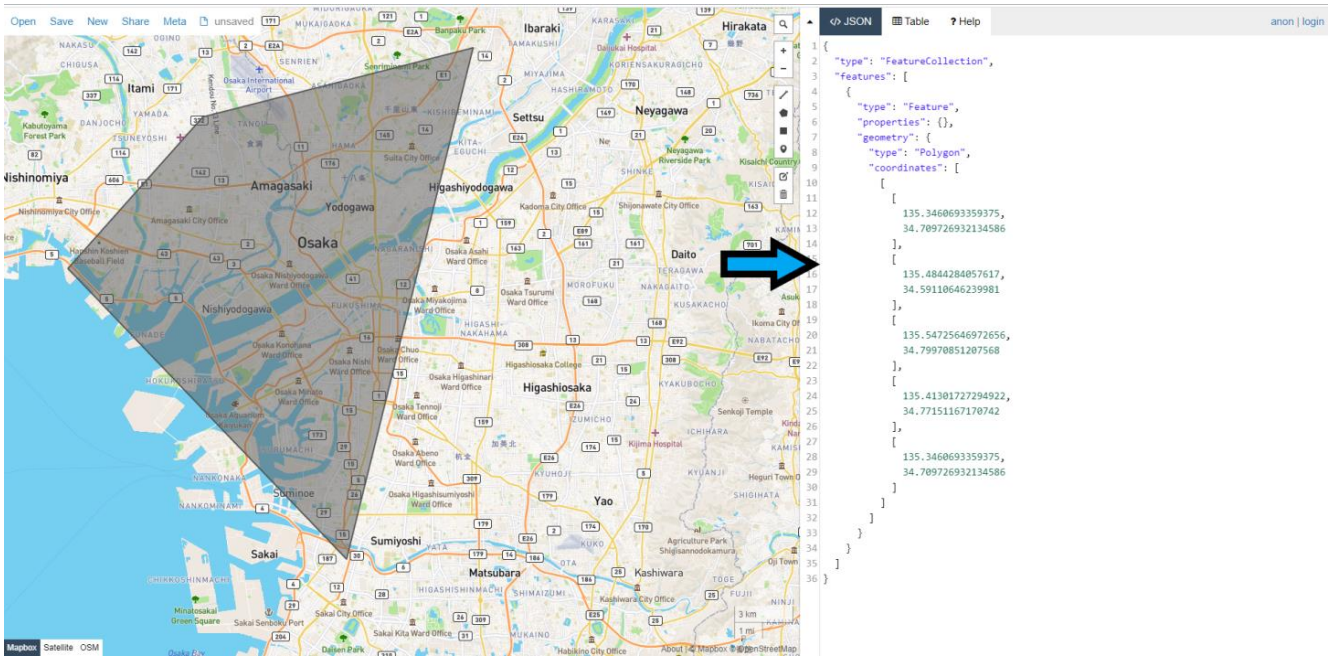

Рисунок 1.2 – GeoJSON

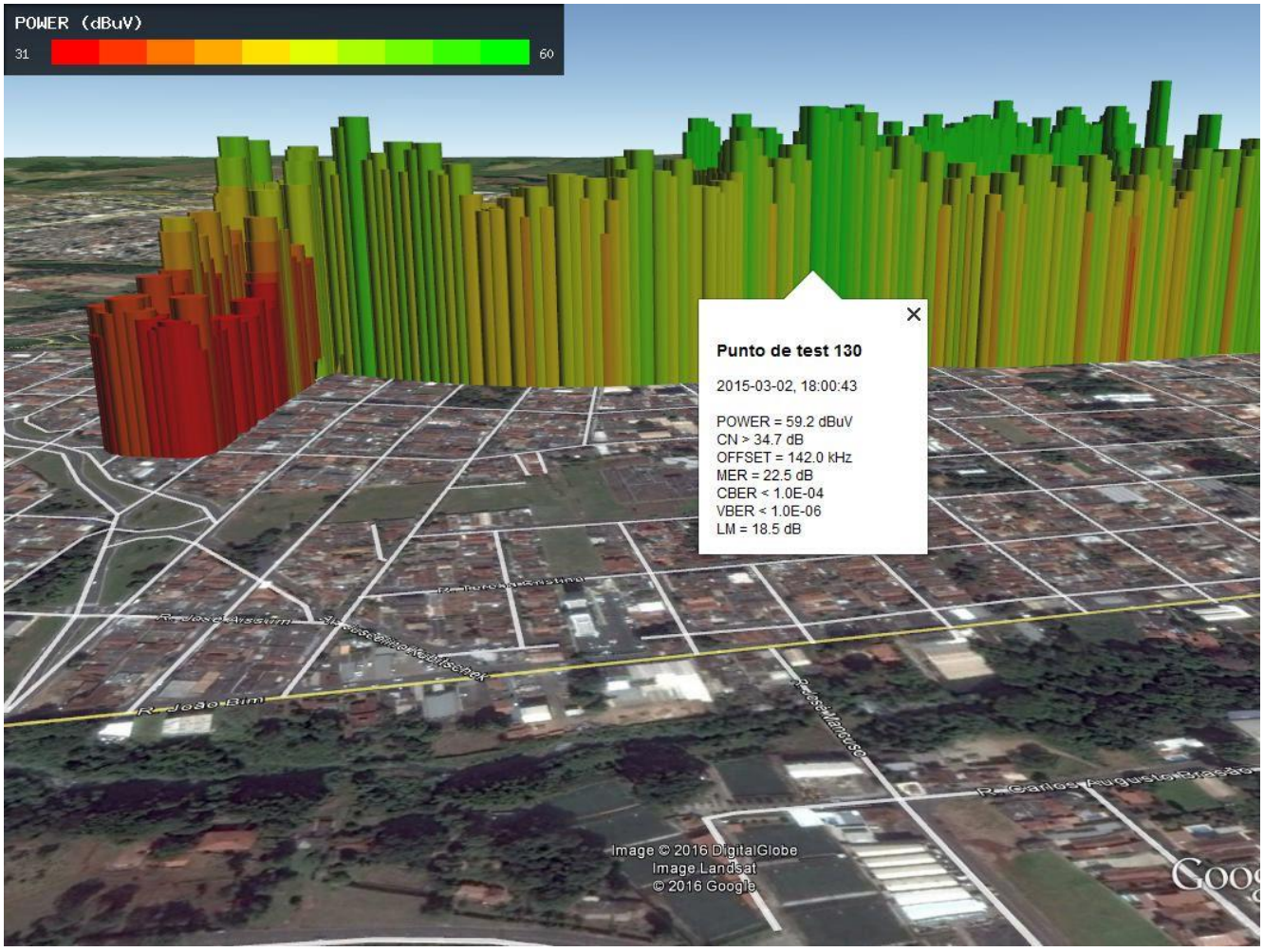

Рисунок 1.3 - KML

Крім того, необхідно передбачити можливість збору та обробки даних про

стан дерев. Це може бути здійснено шляхом використання спеціальних датчиків та пристроїв, які встановлюються на деревах та передають дані про їх стан у режимі реального часу. Також можливо використовувати візуальний моніторинг за допомогою камер та систем комп'ютерного бачення, які дозволяють визначати стан дерев за зовнішніми ознаками. В будь-якому випадку, необхідно передбачити можливість інтеграції з відповідними системами збору та обробки даних, а також розробити відповідні алгоритми та моделі для аналізу даних та визначення стану дерев.

Не менш важливим є питання безпеки та конфіденційності даних, які збираються та обробляються системою. Необхідно передбачити можливість автентифікації та авторизації користувачів, а також забезпечити захист даних від несанкціонованого доступу та зловмисних дій.

В цілому, розробка клієнтської частини автоматизованої системи моніторингу стану дерев з використанням Google Maps є складною та багатогранною задачею, яка вимагає знань та навичок у галузі географії та екології. Однак, з урахуванням всіх вимог та особливостей, ця система може стати надзвичайно корисною та ефективною для моніторингу та захисту навколишнього середовища.

1.2 Порівняльний аналіз аналогів

Розглянемо уже існуючі застосунки, які допомагають у моніторингу стану дерев.

TreePlotter [5] - це мобільний додаток, який дозволяє користувачам відстежувати стан дерев. Користувачі можуть фотографувати дерева та прикріплювати їх до геолокації на мапі. У додатку можна вести записи про характеристики дерев, такі як вид, висота, діаметр стовбура, стан здоров'я тощо. Додаток автоматично відстежує динаміку росту та змін у стані дерев з плином часу на основі збережених даних. Дані можна поділитися з іншими користувачами, дослідниками або місцевими органами влади для колективного моніторингу. TreePlotter має функції аналітики, які допомагають оцінити екологічні переваги дерев, наприклад, їх здатність поглинати вуглекислий газ.

Застосунок TreePlotter продемонстровано на рисунку 1.4.

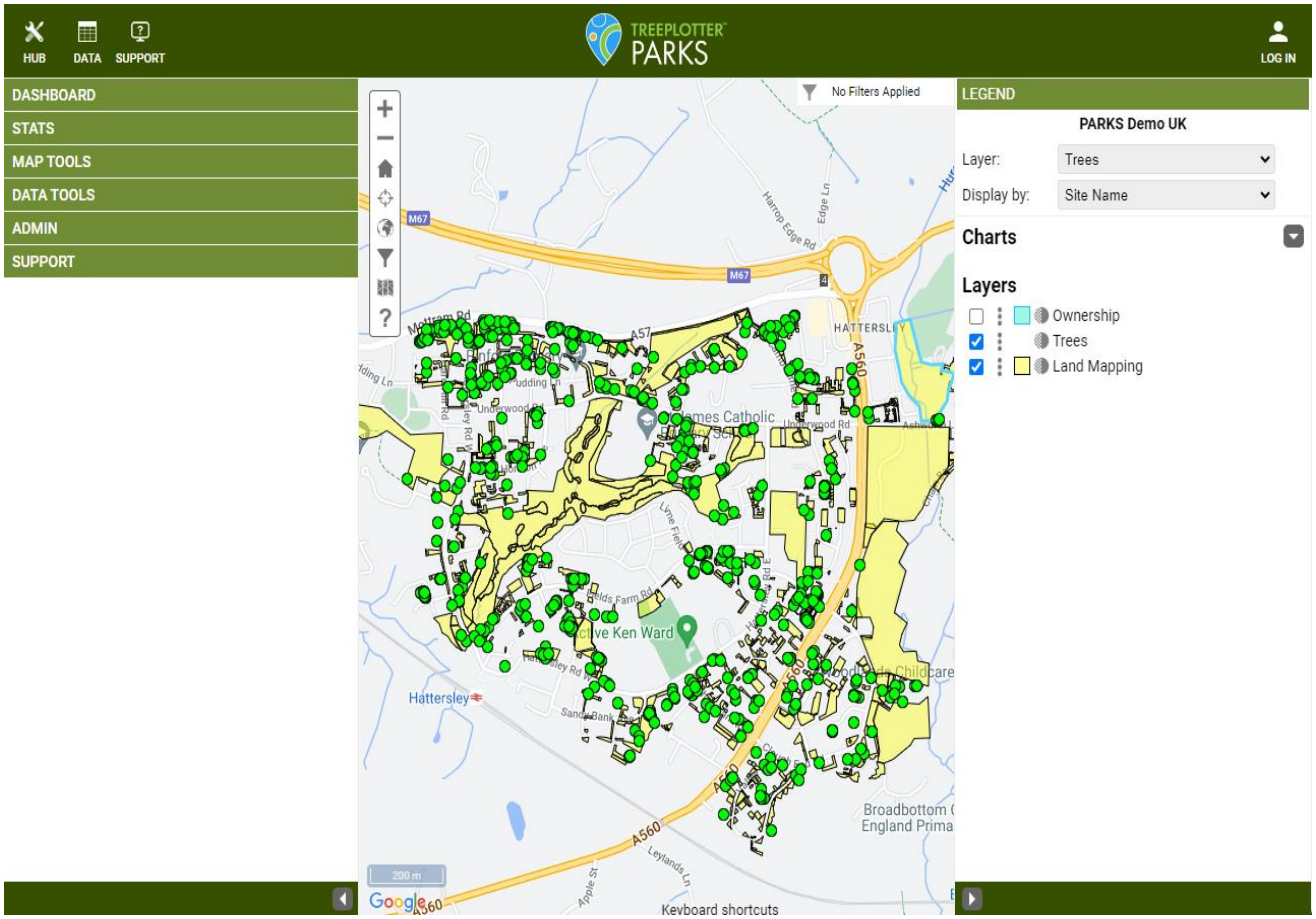

Рисунок 1.4 – TreePlotter

Arbor [6]- це додаток, який використовує супутникові зображення та передові аналітичні інструменти для моніторингу стану дерев на великих територіях.

Додаток проводить автоматичне виявлення та ідентифікацію дерев на супутникових знімках. Після цього, створює детальні карти розташування дерев на відстежуваній території. Проводить збір даних про вид, розмір, стан кожного дерева.

Крім цього, відстежує зміни у висоті, розмірі та морфології крон дерев з плином часу. Виявляє ознаки захворювань, пошкоджень або загибелі дерев. Аналізує вплив екологічних факторів, таких як погода, забруднення тощо.

Додаток Arbor продемонстровано на рисунку 1.5.

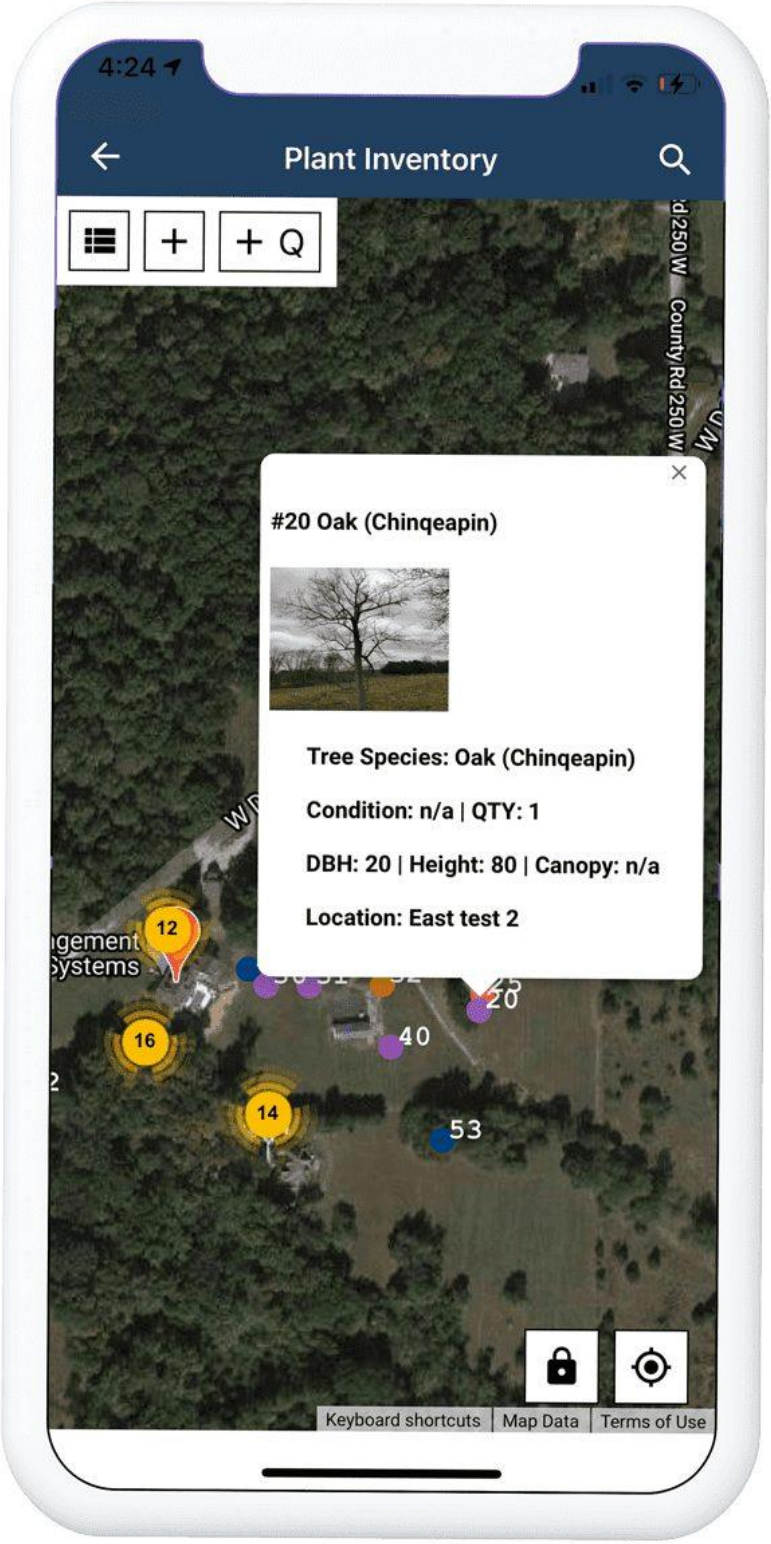

Рисунок 1.5 – Додаток Arbor

PlantsSnap [7] - це застосунок, який допомагає користувачам картографувати, досліджувати та ділитись інформацією про рослини, в тому числі про дерева.

PlantsSnap дозволяє користувачам позначати на інтерактивній карті

розташування окремих рослин, груп або цілих насаджень. До кожної позначки можна прикріпляти фотографії, записи про вид, розмір, стан та іншу інформацію.

PlantsSnap дозволяє створювати персональні або спільні колекції рослин з детальними профілями. Профілі містять дані про походження, екологічні характеристики, фенологію, історичну та культурну цінність тощо. Колекції можна організовувати за різними критеріями, наприклад, за місцем, видами чи тематикою.

Застосунок PlantsSnap продемонстрована на рисунку 1.6.

Рисунок 1.6 – Застосунок PlantsSnap

Для демонстрації відмінностей між описаними аналогами та перевагою власної розробки, було сформовано характеристики застосунків у таблицю 1.1.

Таблиця 1.1 – Порівняння характеристик додатків для моніторингу стану дерев

Характеристика	TreePlotter	Arbor	PlantsSnap	Власна розробка
Картографування дерев	1	1	1	1
Введення характеристик дерев	1	1	1	1
Відображення дерев в реальному масштабі	0	0	0	1
Відстеження динаміки змін	1	1	1	1
Постановка завдань по догляду	0	0	0	1
Загальна оцінка	3	3	3	5

Таким чином, власна розробка має кращу загальну оцінку за аналоги на 40% (100% - 3/5*100% = 40%).

1.3 Аналіз методів розв’язання задачі

Для розробки клієнтської частини автоматизованої системи моніторингу стану дерев можна розглянути кілька підходів:

1. Створення веб-застосунку, який завантажується повністю на початку, а потім оновлює відображення без перезавантаження сторінки. Цей підхід дозволяє забезпечити плавну, швидку та інтерактивну взаємодію користувача з системою. Для його реалізації можуть використовуватися фреймворки на кшталт React [8], Angular [9], Vue.js [10] для побудови компонентної архітектури (рисунок 1.7).

2. Розробка веб-застосунку з окремими сторінками, які завантажуються окремо. Може бути простішим у реалізації, особливо для простіших систем. Потребує повного перезавантаження сторінки при переході між розділами (рис.1.8).

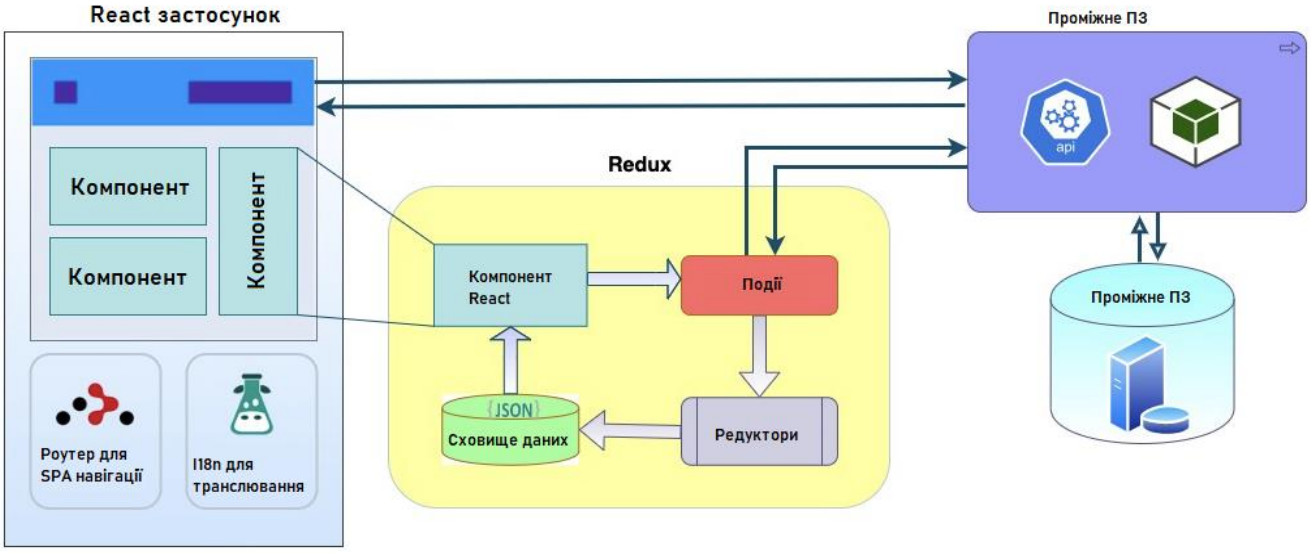

Рисунок 1.7 – Архітектура React застосунку

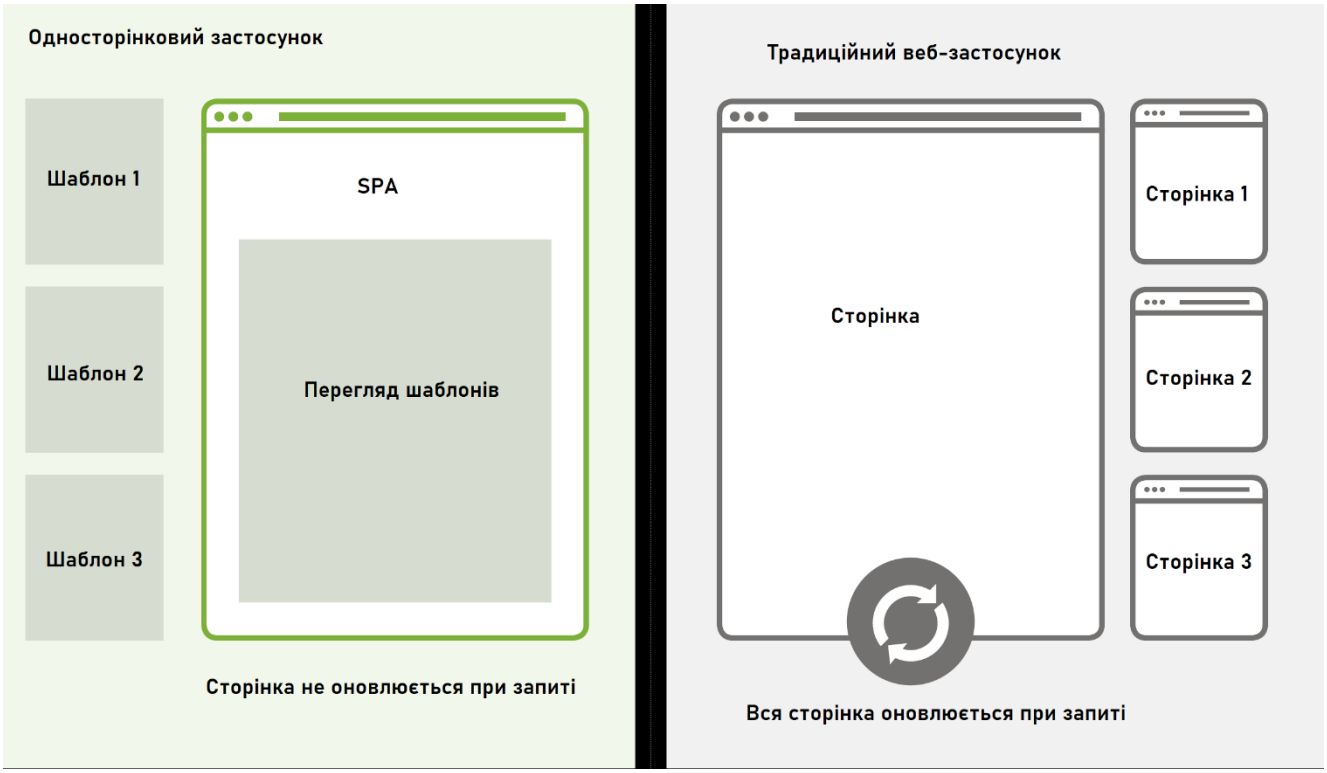

Рисунок 1.8 – Односторінковий та багатосторінковий веб-застосунки

3. Поєднання можливостей веб-сайтів та мобільних додатків. Дозволяє створювати швидкі, надійні та залучаючі застосунки. Може працювати в автономному режимі, отримувати push-сповіщення, мати доступ до апаратних можливостей пристрою. Вимагає додаткових зусиль на реалізацію, але забезпечує високу ефективність та крос-платформеність.

4. Розробка нативних мобільних застосунків для платформ iOS та Android. Дозволяє повноцінно використовувати апаратні можливості мобільних пристроїв. Може забезпечити кращий користувацький досвід, але потребує окремої розробки для кожної платформи.

Хоча клієнтська частина автоматизованої системи моніторингу стану дерев може бути реалізована із використанням різних підходів, традиційний підхід з багатосторінковим застосунком має ряд переваг:

1. Традиційний підхід з окремими сторінками, які завантажуються окремо, є більш простим у розробці та подальшій підтримці.

2. Багатосторінковий застосунок зазвичай має менші вимоги до ресурсів пристрою користувача, оскільки завантажує лише необхідні сторінки.

3. Такі системи індексуються пошуковими системами, що може бути важливо для забезпечення видимості та доступності системи.

4. Окремі сторінки з традиційною ієрархічною структурою можуть забезпечити більш зрозумілу та інтуїтивну навігацію для користувачів.

5. Такий підхід до реалізації є більш сумісним з застарілими версіями браузерів, де підтримка сучасних фреймворків може бути обмеженою.

Таким чином, було прийнято рішення розробити клієнтську частину автоматизованої системи моніторингу стану дерев за традиційним, багатосторінковим підходом.

Для розробки клієнтської частини автоматизованої системи моніторингу стану дерев було обрано традиційний підхід з багатосторінковим застосунком.

1.4 Постановка задач дослідження

У результаті аналізу стану питання розробки клієнтської частини системи моніторингу стану дерев, порівняльного аналізу аналогів та аналізу методів розв’язання задачі, було поставлено такі задачі дослідження:

- розробити модель, метод та алгоритми роботи клієнтської частини вебсистеми;

- розробити модуль реєстрації та авторизації користувача;

- інтегрувати карти Google Maps в систему;
- розробити інтерфейс користувача для моніторингу стану дерев;
- забезпечити відображення дерев на карті;
- провести тестування роботи клієнтської частини вебсистеми.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було проведено аналіз стану питання розробки клієнтської частини системи моніторингу стану дерев, порівняльний аналіз аналогів: TreePlotter, PlantsSnap, Arbor. Серед переваг аналогів варто відзначити можливість картографувати дерева, вести їх характеристику, відстежувати динаміку змін стану дерев. Водночас, аналоги не дозволяють відображати дерева в реальному масштабі та встановлювати задачі по їх догляду.

Було проведено аналіз методів розв’язання задачі дослідження. В результаті цього, було проведено постановку задач дослідження, прийнято рішення розробити клієнтську частину автоматизованої системи моніторингу стану дерев за традиційним, багатосторінковим підходом.

2 РОЗРОБКА СТРУКТУРИ, МОДЕЛІ ТА МЕТОДУ РОБОТИ КЛІЄНТСЬКОЇ ЧАСТИНИ

2.1 Аналіз даних

Під час реєстрації, користувачі надають певну особисту інформацію. Це включає ім'я, адресу електронної пошти та пароль. Ця інформація необхідна для створення окремого аккаунту користувача в системі, створення заходів безпеки входу в аккаунт та забезпечення того, що аккаунт використовує лише той користувач, якому він належить. Електронна пошта використовується для підтвердження реєстрації користувача.

Найбільший перелік даних використовується в контексті дерев. Сюди входить радіус дерева, його вік, стан здоров’я, вид дерева та його фото. Фото дерева використовується для його відображення в повному розмірі та для виділення його серед інших дерев. Знати стан здоров’я необхідно для того, аби визначити перелік робіт, які необхідно виконати для його догляду. Радіус дерева використовується для того, щоб відповідно його відобразити на карті. Чим більше дерево, тим більша умовна позначка для нього.

Також обробляється та зберігається інформація про координати дерева. Це необхідно для відображення переліку дерев на карті так, як вони розміщені в реальності, що дозволяє створювати інтерактивну карту дерев з інформацією про кожне із них. Це створює можливість обрати дерево одразу на карті та переглянути чи змінити інформацію про нього.

2.2 Розробка структури клієнтської частини

Чітка структура допомагає організувати код у логічні модулі та компоненти, що полегшує навігацію, пошук та внесення змін у майбутньому. Це особливо важливо для великих проектів із багатьма файлами і компонентами.

Коли код добре структурований, його легше підтримувати та розширювати у міру зростання проекту. Нові функції та компоненти можна додавати у відповідні папки, не порушуючи існуючу структуру.

Структура клієнтської частини наведена на рисунку 2.1.

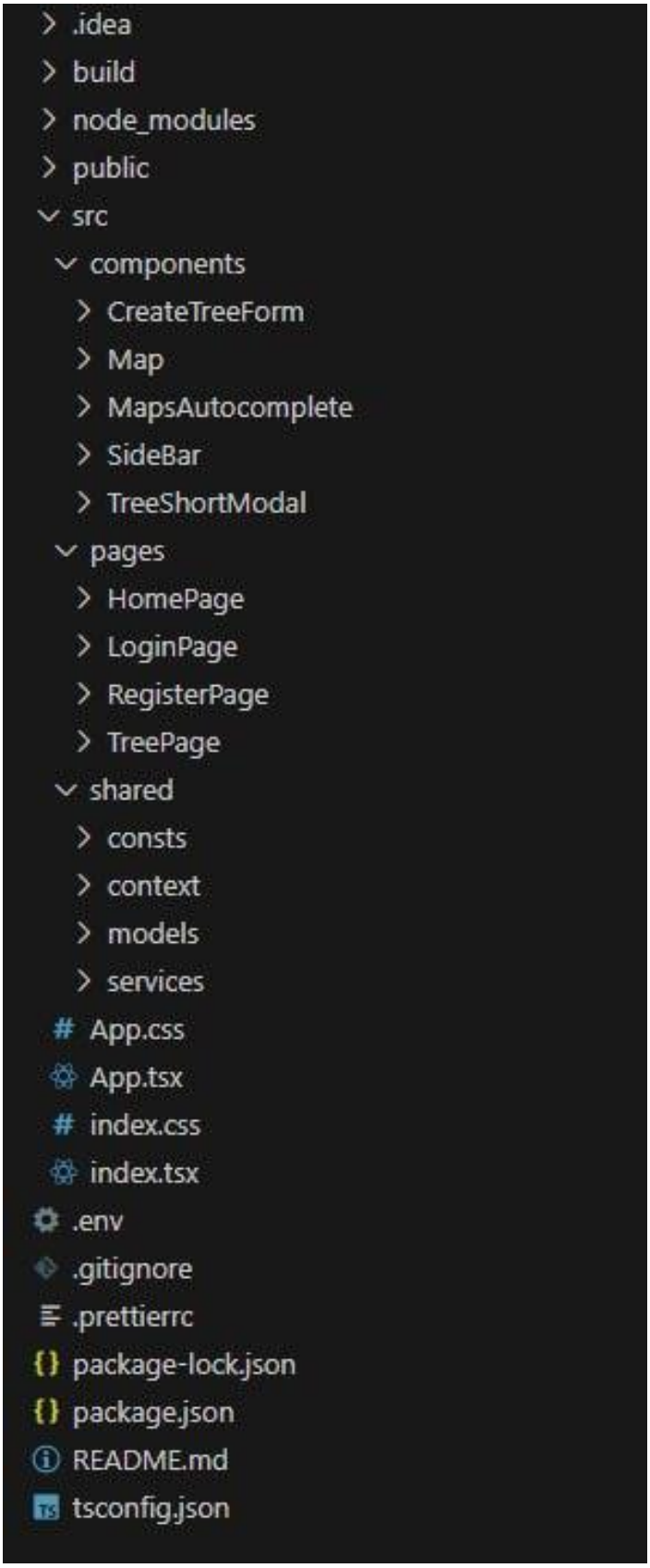

Рисунок 2.1 – Структура клієнтської частини

Ця структура папок є типовою для React проекту. Вона містить основні директорії та файли, необхідні для розробки додатку на базі React.js:

– src: головна директорія для сирцевого коду додатку;

– components: містить окремі компоненти React, такі як CreateTreeForm, Map, MapsAutocomplete, SideBar та TreeShortModal;

– pages: містить сторінки додатку, такі як HomePage, LoginPage, RegisterPage та TreePage;

– shared: загальні ресурси, використовувані в різних частинах додатку, такі як константи, контексти, моделі та сервіси;

– public: директорія для публічних статичних файлів, таких як HTML, CSS та зображення;

– node_modules: містить усі встановлені пакети npm та їх залежності;

– .env: файл для налаштування змінних оточення;

– .gitignore: файл для вказівки яких файлів та директорій слід ігнорувати в системі контролю версій Git;

– .prettierrc: конфігураційний файл для Prettier [11] (інструменту форматування коду);

– package-lock.json: файл, який фіксує точні версії встановлених пакетів та їх залежностей;

– package.json: файл, який містить метадані про проект та список залежностей;

– README.md: файл з описом проекту та інструкціями;

– tsconfig.json: конфігураційний файл для TypeScript компілятора.

Ця структура відповідає рекомендованим практикам для React проектів.

2.3 Розробка моделі клієнтської частини

Було розроблено модель системи у вигляді діаграми компонентів.

Розробка діаграми компонентів є важливою частиною процесу проектування програмного забезпечення. Ця діаграма використовується для візуалізації

організаційної структури системи шляхом відображення її компонентів, їх взаємозв'язків, а також залежностей між компонентами.

Діаграма компонентів надає загальний огляд системи, що допомагає зрозуміти її структуру, складові частини та їх взаємодію. Це корисно як для розробників, так і для інших зацікавлених сторін проекту.

Діаграма компонентів допомагає ідентифікувати залежності між різними компонентами системи. Це важливо для розуміння наслідків змін у одному компоненті на інші частини системи.

Модель клієнтської частини системи у вигляді діаграми компонентів наведено на рисунку 2.2.

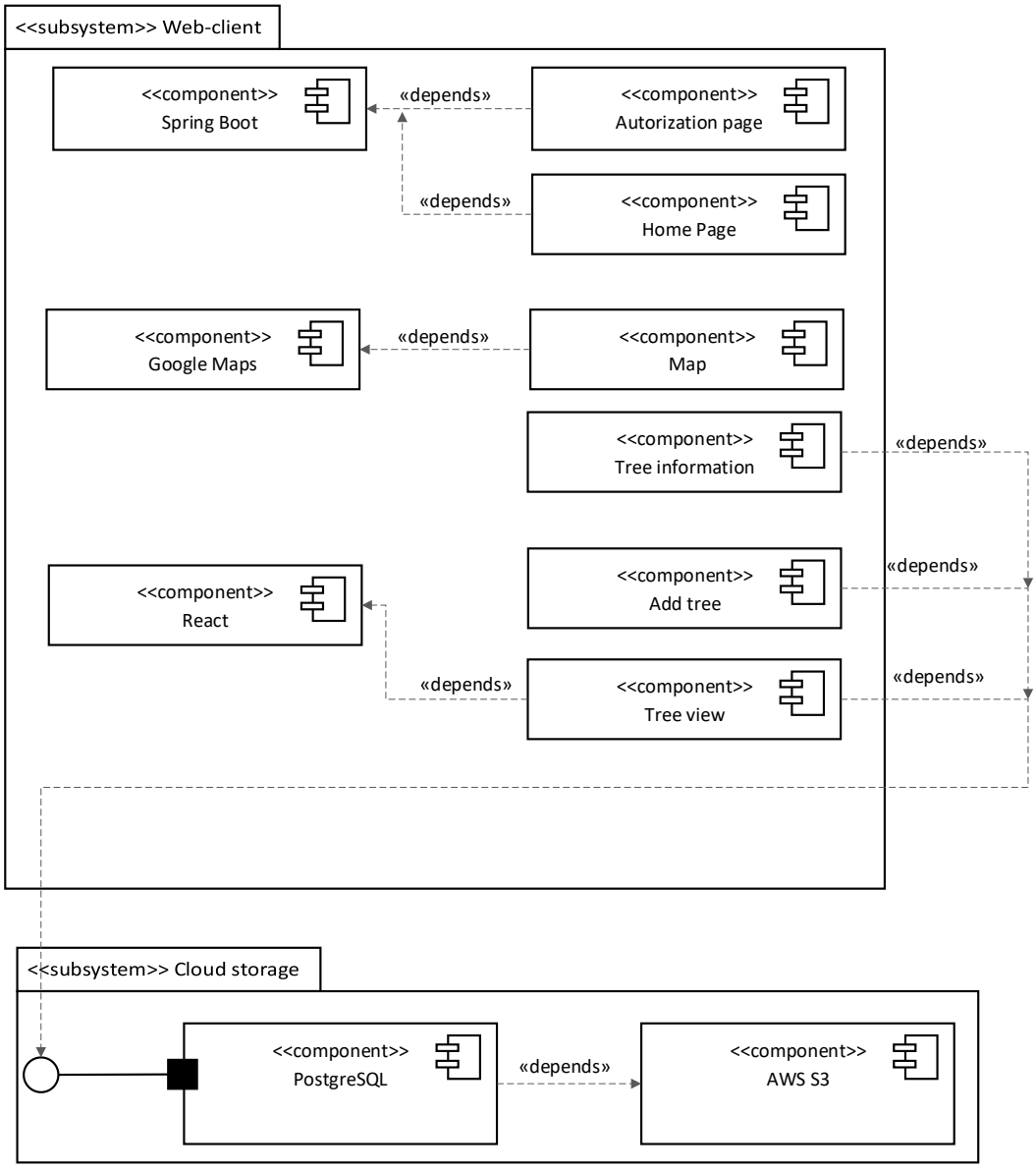

Рисунок 2.2 – Модель клієнтської частини системи

На цій моделі продемонстровані компоненти клієнтської частини системи, а також деякі компоненти серверної частини. Продемонстровано те, як серверна та клієнтська частина пов’язані між собою та як залежать один від одного.

Так, сторінка авторизації та реєстрації залежать від фреймворку серверної частини Spring Boot, який забезпечує збереження даних користувача та їх обробку.

Модуль Map залежить від карти Google Maps, адже вона використовується для відображення карти в застосунку та дерева на ній [12].

Відображення дерев залежить від фреймворку для клієнтської частини React, а також від бази даних PostgreSQL, в якій структурована вся інформація про дерева, що додані до системи. Також від цієї бази даних залежать модуль додавання дерев та інформації про них.

База даних PostgreSQL, в свою чергу, залежить від сховища даних AWS S3.

2.4 Розробка методу позначення дерев на карті

Позначення окремих дерев на карті в автоматизованій системі моніторингу стану дерев з використанням Google Maps є найбільш важливою функцією в системі.

Присвоєння унікального ідентифікатора кожному дереву дозволяє відрізняти його від інших, швидко знаходити в базі даних, що полегшує відстежування його стану, історії обробки, а також дозволяє легко пов'язувати з ним дані спостережень, виміри та іншу релевантну інформацію. Це забезпечує можливість відстежувати зміни стану конкретного дерева з часом та забезпечує чітку структуризацію даних.

Позначення дерев на карті надає точні географічні координати кожного дерева. Це дозволяє ефективно планувати маршрути та розподіляти ресурси для обслуговування, обрізки, пересадки або вирубки дерев у міській зеленій зоні. До того ж, різні позначення дерев різними кольорами залежно від їх стану або різного розміру залежно від їх реального розміру або віку.

Маючи дерева позначеними на карті, можна візуально оцінити їх розподіл, щільність, а також аналізувати взаємозв'язки між станом дерев та іншими географічними факторами, такими як забруднення, щільність населення чи

інфраструктура.

Інтерактивна карта з позначеними деревами може слугувати платформою для залучення громадськості до моніторингу та догляду за міським озелененням, надаючи можливість повідомляти про проблеми.

Метод позначення дерев на карті акумулює і виконує такі функції:

1. Інтеграція карт Google Maps з використанням Google Maps API як основу для позначення дерев.

2. Визначення географічних координат з використанням функції fetch(), яка розраховує координати області, в межах якої потрібно відобразити дерева, а саме: координату верхньої лівої точки області карти (startY), верхньої правої точки області карти (startX), нижньої лівої точки карти (endY), нижньої правої точки карти (endX).

3. Позначення на карті дерев, моніторинг яких потрібно вести, з використанням об’єкту API, який викликає функцію get для того, аби отримати список дерев, що потрібно позначити на карті.

4. Додавання такої інформації про дерева, як: вид дерева, його вік, завдання по догляду за деревом, фото дерева.

5. Позначення дерева на карті з використанням об’єкта маркера з класу Circle, який надається Google Maps API, у вигляді кола із центром в координаті, у якій він знаходиться. Радіус кола залежить від віку дерева. Колір позначення залежить від його стану: дерево здорове, потребує догляду або вирубки (рисунок 2.3).

6. Масштабування позначення дерев залежно від наближення користувачем карти.

7. Групування дерев з відповідним позначенням на карті, для якого використовується клас MarkerClusterer, яке відбувається у разі, якщо користувач сильно зменшив масштаб і неможливе позначення окремих дерев.

8. Збереження інформації про дерева у базі даних.

Користувач авторизується в системі та переходить на головну сторінку з картою. Він обирає на карті потрібну точку для додавання дерева, робить клік по ній та обирає опцію додавання дерева. Після цього вводить необхідну інформацію

та зберігає її. Після цього, на карті відображається додане дерево.

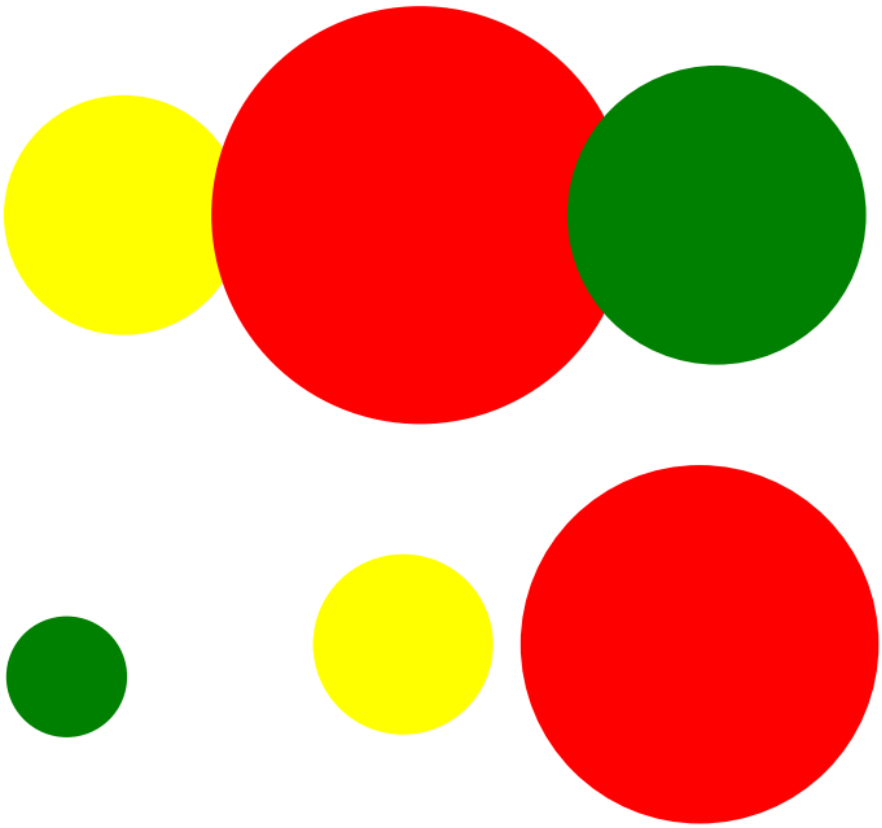

Рисунок 2.3 – Графічне представлення позначень дерев

На цьому рисунку подано зелене коло з радіусом 2, жовте коло з радіусом 4, червоне коло з радіусом 7, зелене коло з радіусом 5, жовте коло з радіусом 3, червоне коло з радіусом 6. Червоний колір означає потребу у вирубці дерева, жовтий – догляд за деревом, зелений – дерево здорове.

Опис компонентів:

1. Веб-застосунок – клієнтська частина додатку, яка забезпечує інтерфейс користувача для взаємодії з картою та даними про дерева. Він використовує Google Maps API для відображення карти та маніпуляції з позначеннями дерев.

2. Інтерфейс користувач – компонент веб-застосунку, який відповідає за відображення карти, елементів керування та форм для введення інформації про дерева.

3. Серверна частина, яка обробляє запити від клієнтської частини, взаємодіє з

базою даних та іншими сервісами (наприклад, Google Maps API).

4. Обробка запитів і маршрутизація - компонент серверної частини, який відповідає за обробку вхідних запитів, маршрутизацію та відповідь на запити від клієнтської частини.

5. Google Maps API - зовнішній сервіс, який забезпечує доступ до карт Google Maps та функціоналу для маніпуляції з картами та позначеннями.

6. База даних – сховище для зберігання інформації про дерева, включаючи їх координати, вид, вік, завдання по догляду та фотографії.

2.5 Висновки

У другому розділі було проведено аналіз даних системи моніторингу стану дерев, розроблено структуру клієнтської частини. Розроблено модель клієнтської частини, продемонстровано її складові частини. Розроблено метод позначення дерев на карті з використанням Google Maps.

3 РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ

3.1 Варіантний аналіз та вибір мови програмування

TypeScript [13] – це статично типізована мова програмування, розроблена компанією Microsoft.

Вона є надбудовою над JavaScript, додаючи статичну типізацію та інші можливості, які допомагають розробникам писати більш надійний і масштабований код.

TypeScript дозволяє виявляти помилки під час розробки, а не під час виконання коду, що допомагає заощадити час і зменшити кількість помилок у програмі.

TypeScript підтримує останні стандарти JavaScript, включаючи ES6, ES7 та ES8, і може компілюватися в будь-яку версію JavaScript, що дозволяє використовувати нові можливості мови, не турбуючись про сумісність зі старими браузерами.

TypeScript також має потужну систему інтерфейсів, яка дозволяє розробникам створювати гнучкі та масштабовні архітектури застосунків.

JavaScript [14] – це інтерпретована мова програмування, яка використовується для створення вебзастосунків. Вона є однією з трьох основних технологій вебпрограмування, разом з HTML та CSS.

JavaScript дозволяє розробникам створювати динамічні веб-сторінки, які можуть взаємодіяти з користувачем та змінювати свій контент у відповідь на дії користувача. JavaScript є клієнтською мовою програмування, тобто вона виконується на стороні користувача, у веб-браузері. JavaScript також може використовуватися для створення серверних застосунків, завдяки таким платформам, як Node.js.

Python [15] – це інтерпретована мова програмування загального призначення, яка використовується для створення веб-застосунків, наукових обчислень, штучного інтелекту та багатьох інших задач.

Python має простий і зрозумілий синтаксис, що робить його ідеальним для

початківців у програмуванні.

Python також має велику кількість бібліотек та фреймворків, які допомагають розробникам створювати вебзастосунки швидше та простіше.

Для створення вебзастосунків на Python найчастіше використовуються фреймворки Django та Flask.

Python також підтримує об'єктно-орієнтоване програмування та функціональне програмування, що дозволяє розробникам використовувати різні підходи до розробки програмного забезпечення.

Архітектура Python наведена на рисунку 3.1.

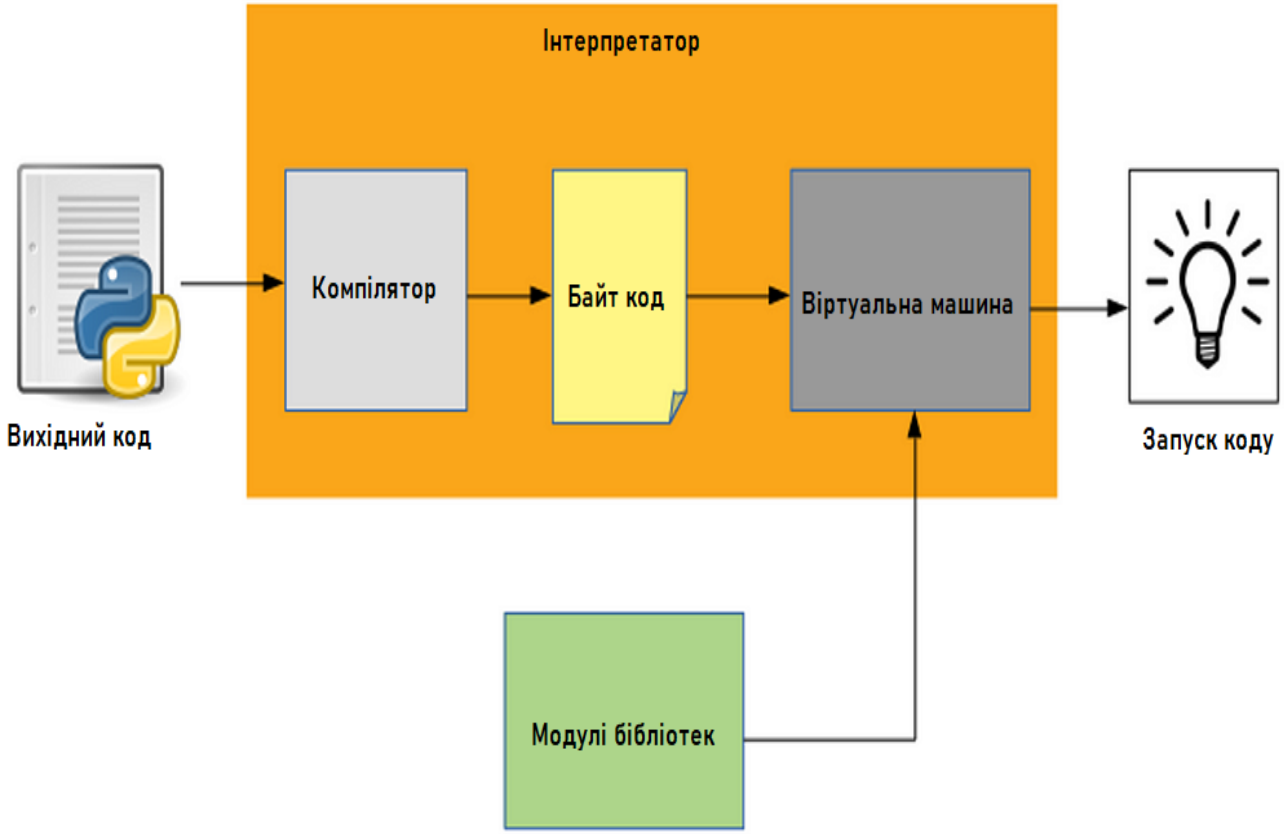

Рисунок 3.1 – Архітектура Python

Ruby [16] – це об'єктно-орієнтована мова програмування, яка використовується для створення вебзастосунків, системного програмування та інших задач. Ruby має простий та виразний синтаксис, що робить його ідеальним для початківців у програмуванні.Ruby також має велику кількість бібліотек та

фреймворків, які допомагають розробникам створювати вебзастосунки швидше та простіше. Для створення вебзастосунків на Ruby найчастіше використовується фреймворк Ruby on Rails. Ruby також підтримує функціональне програмування та метапрограмування, що дозволяє розробникам створювати гнучкі та масштабовні архітектури застосунків.

Архітектура Ruby для веб-застосунків наведена на рисунку 3.2.

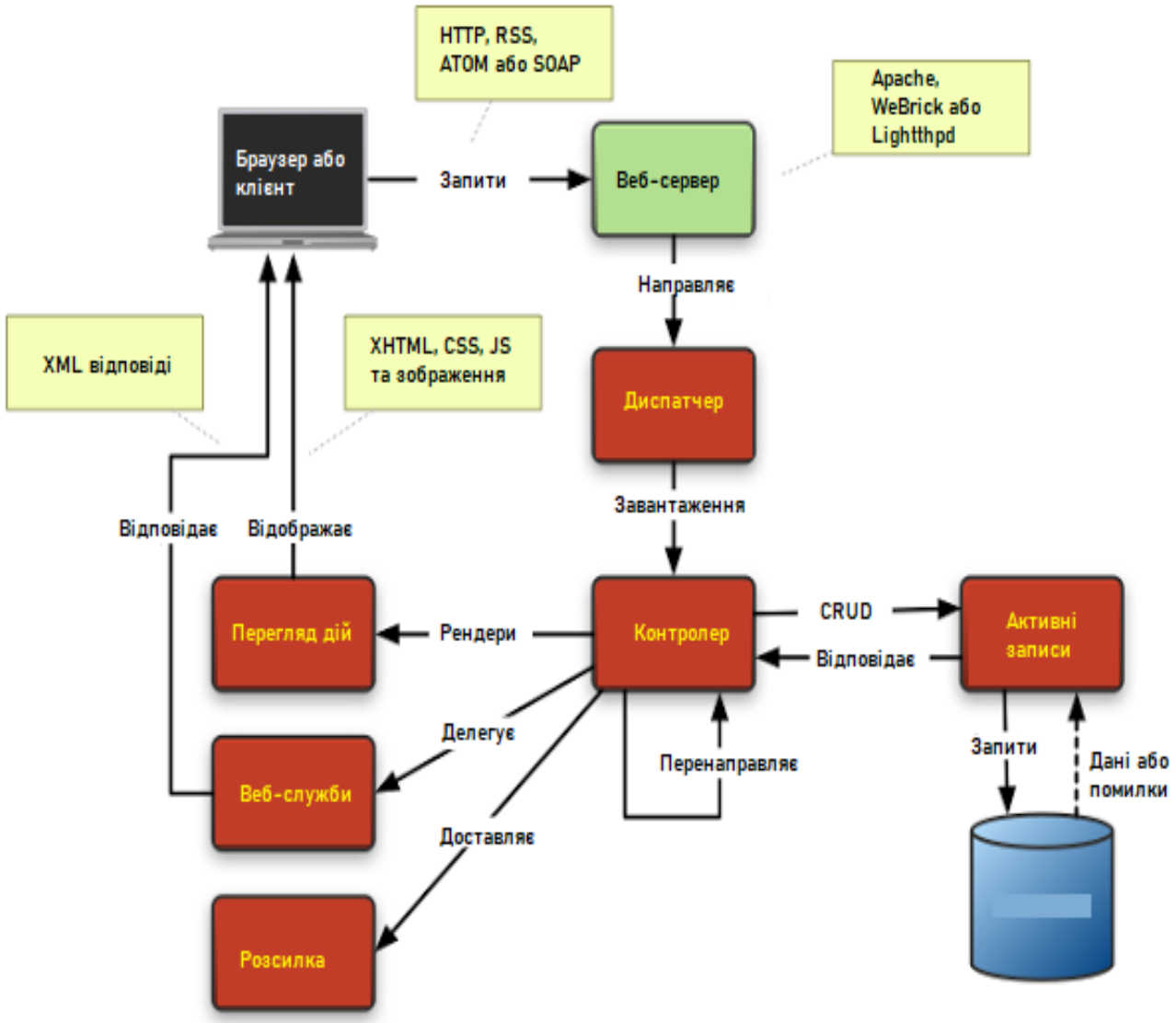

Рисунок 3.2 – Архітектура Ruby

PHP [17] – це серверна мова програмування, яка використовується для створення вебзастосунків. PHP є однією з найпопулярніших мов програмування для

веброзробки, завдяки простоті використання та великій кількості готових рішень.

PHP може використовуватися для створення динамічних вебсторінок, які можуть взаємодіяти з базами даних та іншими джерелами даних. PHP також має велику кількість бібліотек та фреймворків, які допомагають розробникам створювати вебзастосунки швидше та простіше.

Для створення вебзастосунків на PHP найчастіше використовуються фреймворки Laravel та Symfony.

Таблиця 3.1 – Таблиця порівняння характеристик мов програмування

Характеристика	TypeScript	JavaScript	Python	Ruby
Статична типізація	Так	Ні	Ні	Ні
Підтримка великих проектів	Так	Частково	Частково	Частково
Розвинені інструменти та екосистема	Так	Так	Так	Так
Швидкість виконання	Частково	Частково	Ні	Ні
Підтримка асинхронного програмування	Так	Так	Так	Частково
Простота навчання	Частково	Так	Так	Так
Популярність та ком'юніті	Так	Так	Так	Частково
Можливості для фронтенд та бекенд розробки	Так	Так	Частково	Частково
Сумарний бал	8	6	5	4

TypeScript дозволяє використовувати статичну типізацію, яка допомагає уникнути помилок під час розробки та полегшує розуміння коду. Завдяки статичній типізації, TypeScript може виявляти помилки під час компіляції, а не під час виконання коду, що допомагає заощадити час на відлагодженні.

TypeScript є надбудовою над JavaScript, тому все, що можна зробити на JavaScript, можна зробити і на TypeScript. Крім того, TypeScript додає додаткові можливості, такі як інтерфейси, класи, модулі та декоратори, які допомагають

створювати більш надійний та масштабовний код.

TypeScript має потужну систему інтерфейсів, яка дозволяє створювати гнучкі та масштабовні архітектури застосунків. Завдяки інтерфейсам, розробники можуть створювати абстрактні типи даних, які можуть бути реалізовані в різних частинах програми.

TypeScript дозволяє розробникам писати більш надійний та масштабовний код, використовуючи статичну типізацію та інші можливості мови. Крім того, TypeScript має велику спільноту та екосистему, що допомагає розробникам знайти готові рішення та отримати допомогу від інших розробників.

Отже, TypeScript є найкращим вибором для розробки клієнтської частини веб-системи для моніторингу стану дерев.

3.2 Варіантний аналіз та вибір середовища розробки клієнтської частини

Atom [18] – це безкоштовний, відкритий та гнучкий текстовий редактор для кодування, розроблений компанією GitHub. Він підтримує широкий спектр мов програмування та доступний для операційних систем Windows, macOS та Linux.

Atom має простий та інтуїтивно зрозумілий інтерфейс, який дозволяє розробникам швидко та легко налаштовувати редактор під свої потреби (рис. 3.3). Однією з ключових особливостей Atom є вбудований пакетний менеджер, який дозволяє розробникам шукати, встановлювати та оновлювати пакети (розширення) безпосередньо з редактора. Atom має велику кількість плагінів та розширень, які можна встановити для додавання нових функцій та мов програмування.

Atom дозволяє користувачам налаштовувати редактор під свої потреби, включаючи теми оформлення, шрифти, клавіатурні скорочення та інші параметри. Він має підсвітку синтаксису для широкого спектру мов програмування, що допомагає розробникам швидше розуміти та відстежувати код. Atom також має вбудовану підтримку систем керування версіями, таких як Git та SVN, що дозволяє розробникам виконувати основні операції з керування версіями безпосередньо з редактора.

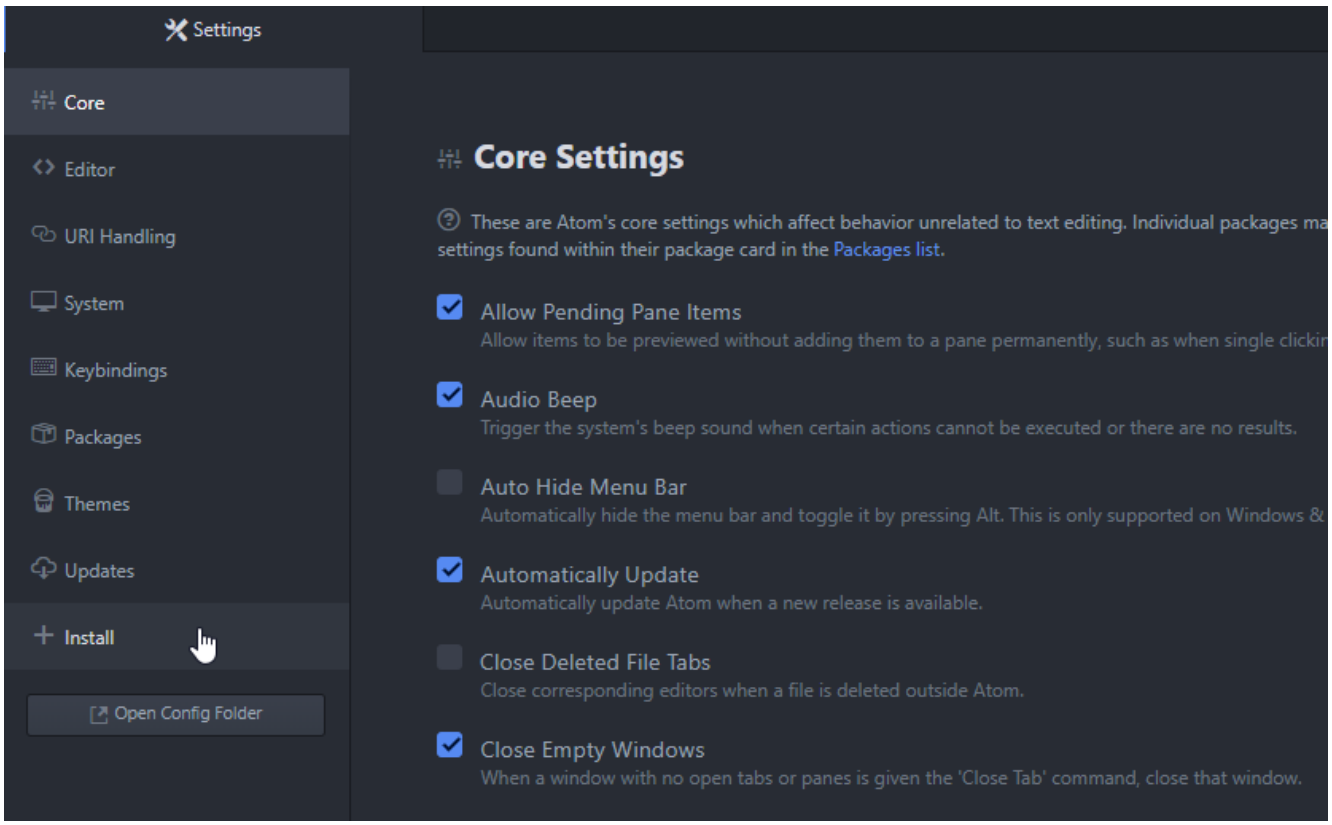

Рисунок 3.3 – Інтерфейс Atom IDE

Однією з корисних функцій Atom є множинне редагування, яке дозволяє розробникам одночасно редагувати кілька рядків коду, що значно прискорює процес редагування. Atom також має API для плагінів, яке дозволяє розробникам створювати власні плагіни та розширення для редактора. Слід зазначити, що Atom збирає дані про використання редактора, що допомагає розробникам Atom покращувати продукт.

Atom є відкритим програмним забезпеченням, що означає, що його вихідний код доступний для всіх. Він є популярним вибором серед розробників завдяки своїй гнучкості, великій кількості розширень та вбудованому пакетному менеджеру.

Visual Studio Code (VS Code) [19] – це безкоштовний, легкий та потужний текстовий редактор, розроблений компанією Microsoft. Він призначений для розробки програмного забезпечення та підтримує широкий спектр мов програмування, включаючи JavaScript, TypeScript, Python, Java, C++, C#, PHP, Go та багато інших.

VS Code має простий та інтуїтивно зрозумілий інтерфейс, який дозволяє

розробникам швидко та легко налаштовувати редактор під свої потреби. Однією з ключових особливостей VS Code є інтелектуальне завершення коду, яке допомагає прискорити написання коду та зменшити кількість помилок. VS Code також має вбудований відлагоджувач, який дозволяє розробникам встановлювати точки зупинки, переглядати змінні та стек викликів, а також виконувати код крок за кроком (рисунок 3.4).

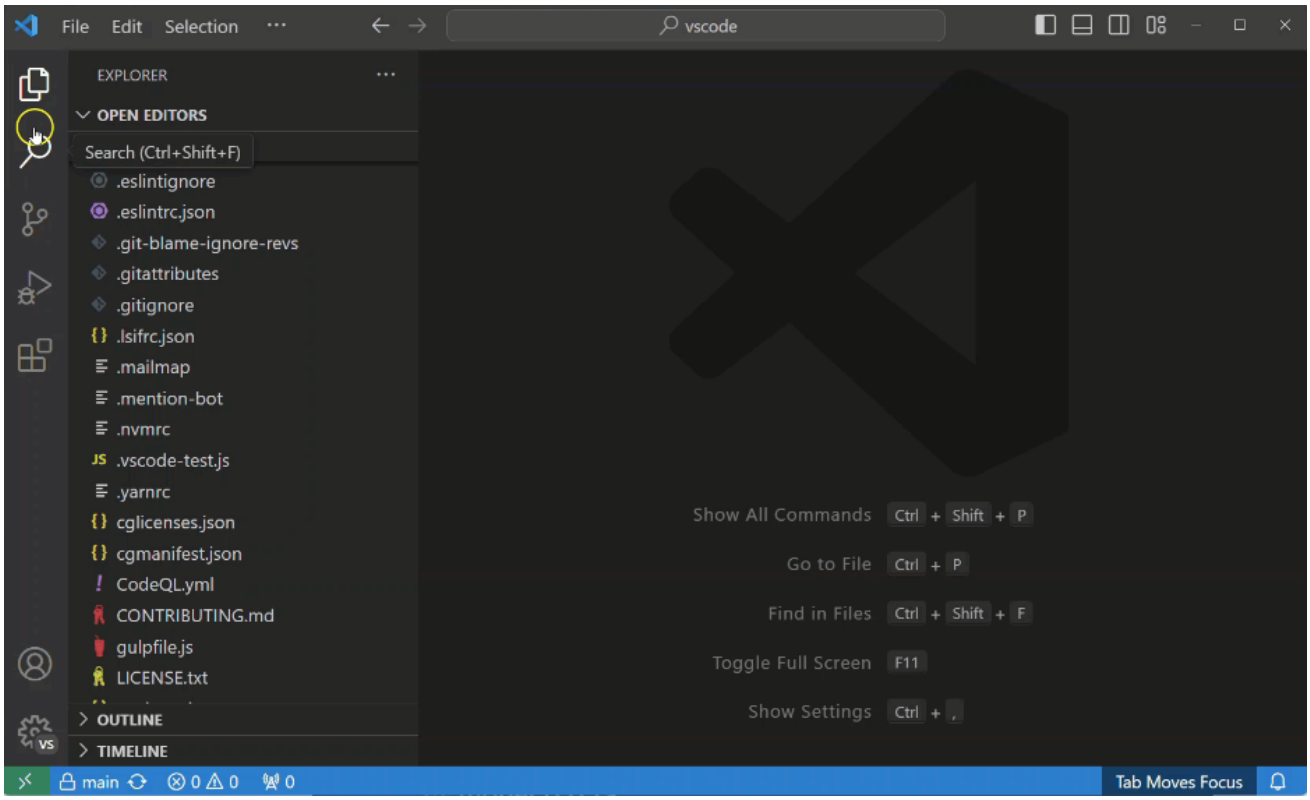

Рисунок 3.4 – Інтерфейс Visual Studio Code

VS Code має гітку інтеграцію з Git, що дозволяє розробникам виконувати основні операції з керування версіями, такі як коміти, злиття та перегляд змін, безпосередньо з редактора. VS Code також має велику кількість розширень, які можна встановити для додавання нових функцій та мов програмування. Розширення можна знайти та встановити через вбудований ринок розширень.

VS Code дозволяє користувачам налаштовувати редактор під свої потреби, включаючи теми оформлення, шрифти, клавіатурні скорочення та інші параметри. Він має вбудований термінал, який дозволяє розробникам виконувати команди

операційної системи безпосередньо з редактора. VS Code також має функцію Live Share, яка дозволяє розробникам спільно працювати над кодом у реальному часі.

VS Code використовує Language Server Protocol (LSP) для забезпечення інтелектуального завершення коду, перевірки синтаксису та інших функцій для широкого спектру мов програмування. Він є популярним вибором серед розробників завдяки своїй швидкості, гнучкості та великій кількості розширень. VS Code доступний для операційних систем Windows, macOS та Linux.

Sublime Text [20] – це популярний, швидкий та гнучкий текстовий редактор для кодування, який підтримує широкий спектр мов програмування. Він був розроблений компанією Sublime HQ та доступний для операційних систем Windows, macOS та Linux.

Sublime Text має простий та інтуїтивно зрозумілий інтерфейс, який дозволяє розробникам швидко та легко налаштовувати редактор під свої потреби. Однією з ключових особливостей Sublime Text є потужні функції пошуку та заміни, які дозволяють розробникам швидко знаходити та замінювати текст у файлах. Sublime Text також дозволяє розробникам одночасно редагувати кілька рядків коду, що значно прискорює процес редагування.

Інтерфейс Sublime Text наведено на рисунку 3.5.

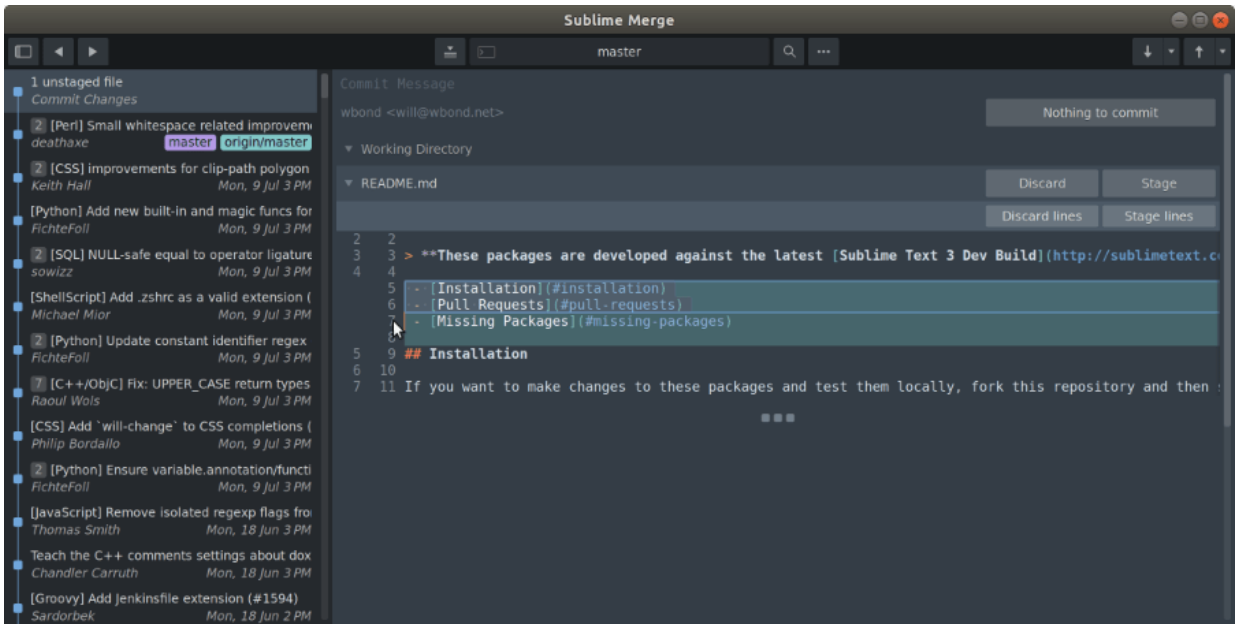

Рисунок 3.5 – Інтерфейс Sublime Text

Sublime Text має велику кількість плагінів та розширень, які можна встановити для додавання нових функцій та мов програмування. Він дозволяє користувачам налаштовувати редактор під свої потреби, включаючи теми оформлення, шрифти, клавіатурні скорочення та інші параметри. Sublime Text має підсвітку синтаксису для широкого спектру мов програмування, що допомагає розробникам швидше розуміти та відстежувати код.

Sublime Text має вбудовану підтримку систем керування версіями, таких як Git та SVN, що дозволяє розробникам виконувати основні операції з керування версіями безпосередньо з редактора. Він також має API для плагінів, яке дозволяє розробникам створювати власні плагіни та розширення для редактора.

Sublime Text є платним програмним забезпеченням, але надає безкоштовну ліцензію для ознайомлення з програмою. Він є популярним вибором серед розробників завдяки своїй швидкості, гнучкості та великій кількості розширень. Sublime Text також має функцію перегляду змін, яка дозволяє розробникам переглядати зміни, внесені в файли, та відновлювати попередні версії файлів.

У таблиці 3.1 наведено порівняння характеристик описаних середовищ розробки веб-застосунків.

Таблиця 3.2 - Порівняння характеристик описаних середовищ розробки веб-застосунків

Критерій	**Visual Studio Code (VS Code)**	**Sublime Text**	**Atom**
Висока швидкість роботи	+	+	-
Інтуїтивний інтерфейс	+	+	+
Багато плагінів та розширень	+	+	+
Широка підтримка мов програмування	+	+	+
Безкоштовна версія	+	-	+
Гнучке налаштування	+	+	+
Великий перелік документації щодо користування	+	+	-
Підтримка різних платформ	+	+	+
Багато вбудованих функцій	+	-	-

Visual Studio Code (VS Code) може бути кращим вибором для розробки клієнтської частини веб-застосунку в порівнянні з Atom та Sublime Text з декількох причин.

По-перше, VS Code має функцію інтелектуального завершення коду, яка допомагає прискорити написання коду та зменшити кількість помилок. Ця функція особливо корисна при роботі з JavaScript, який є основною мовою програмування для клієнтської частини веб-застосунків.

По-друге, VS Code має вбудований відлагоджувач, який дозволяє розробникам встановлювати точки зупинки, переглядати змінні та стек викликів, а також виконувати код крок за кроком. Ця функція допомагає розробникам швидше знаходити та виправляти помилки в коді.

По-третє, VS Code має гітку інтеграцію з Git, що дозволяє розробникам виконувати основні операції з керування версіями, такі як коміти, злиття та перегляд змін, безпосередньо з редактора. Ця функція особливо корисна для роботи з великими проектами, де керування версіями є необхідним.

По-четверте, VS Code має велику кількість розширень, які можна встановити для додавання нових функцій та мов програмування. Розширення можна знайти та встановити через вбудований ринок розширень. Наприклад, є розширення для React, Angular, Vue.js та багато інших популярних фреймворків та бібліотек для клієнтської частини веб-застосунків.

По-п'яте, VS Code має функцію Live Share, яка дозволяє розробникам спільно працювати над кодом у реальному часі. Ця функція особливо корисна для роботи з командою розробників, де кілька осіб можуть працювати над одним проектом одночасно.

Отже, VS Code має більше функцій, які є спеціально розроблені для розробки клієнтської частини веб-застосунків, в порівнянні з Atom та Sublime Text. Це робить VS Code кращим вибором для розробників, які працюють з клієнтською частиною веб-застосунків.

3.3 Розробка клієнтської частини модуля авторизації та реєстрації

Розробка клієнтської частини веб-застосунку є однією з ключових складових у створенні повноцінного веб-додатку. Клієнтська частина, також відома як фронтенд, відповідає за візуалізацію та взаємодію користувача з веб-сторінкою.

Процес розробки клієнтської частини починається з проєктування інтерфейсу користувача.

Було розроблено структурні схеми інтерфейсу користувача. Структурна схема вікна реєстрації наведена на рисунку 3.6.

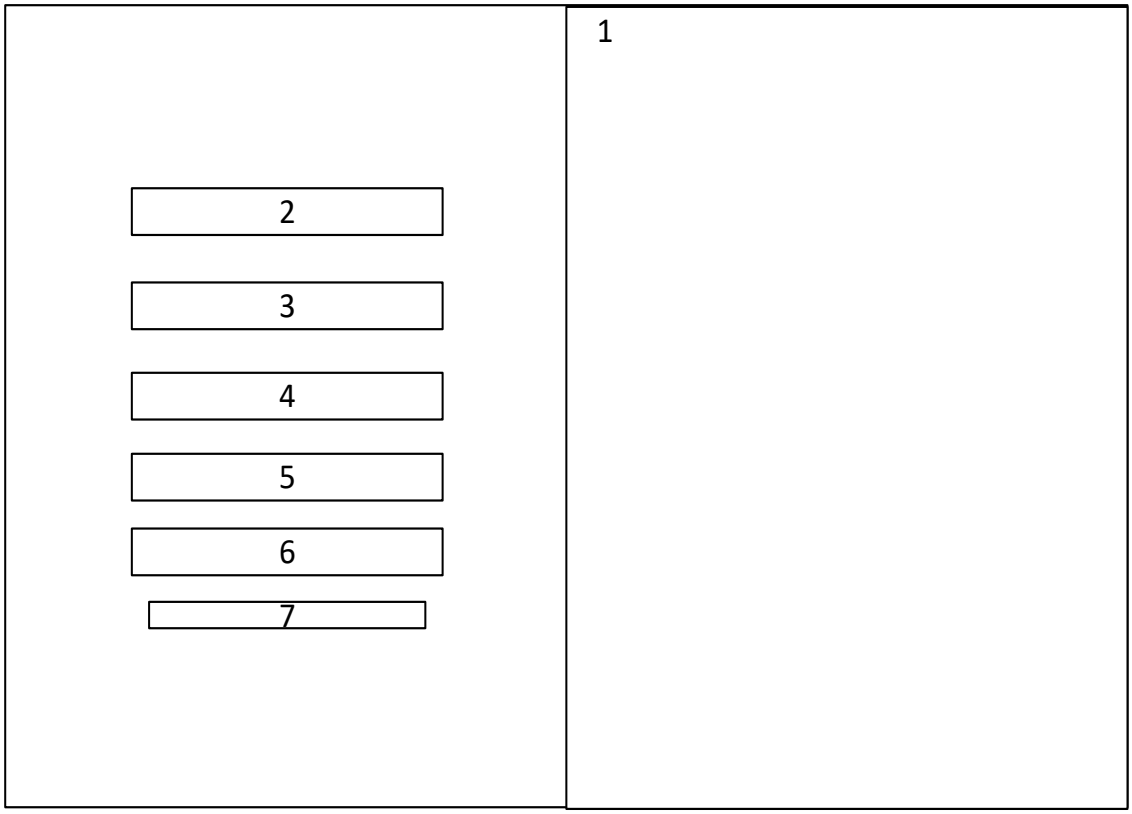

Рисунок 3.6 – Структурна схема вікна реєстрації

Елементи сторінки реєстрації:

1. Зображення з деревами.
2. Поле для імені користувача.
3. Поле для електронної пошти.
4. Поле для пароля.
5. Поле для повторного пароля.

6. Кнопка підтвердження реєстрації.
7. Кнопка переходу на вкладку для входу в аккаунт.

Було також розроблено структурну схему вікна авторизації. Вона наведена на рисунку 3.7.

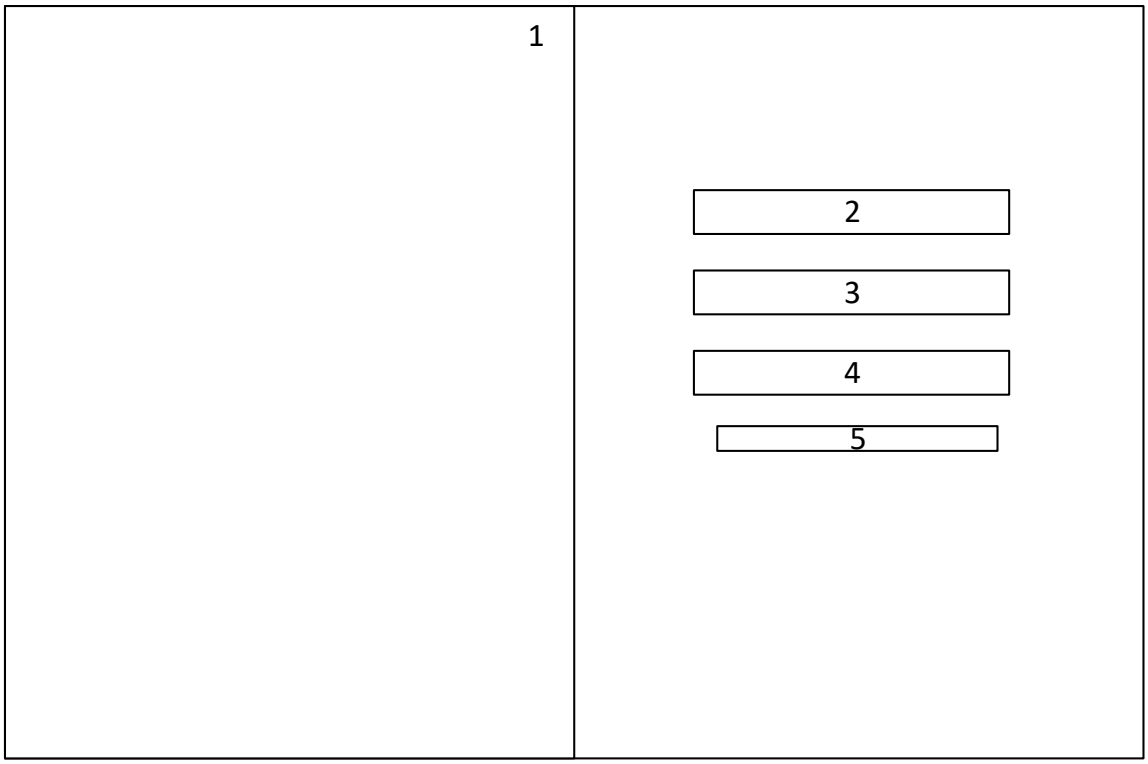

Рисунок 3.7 – Структурна схема вікна авторизації

Елементи сторінки авторизації:

1. Зображення з деревами.
2. Поле для електронної адреси.
3. Поле для пароля.
4. Поле для підтвердження входу.
5. Кнопка переходу на сторінку реєстрації.

3.4 Розробка клієнтської частини модуля роботи з деревами

Модуль роботи з деревами дозволяє додавати та редагувати інформацію про дерева, стежити за їх станом, призначати та виконувати завдання щодо їх догляду та перегляд дерев на карті.

Було розроблено загальний алгоритм роботи системи, який продемонстровано на рисунку 3.8.

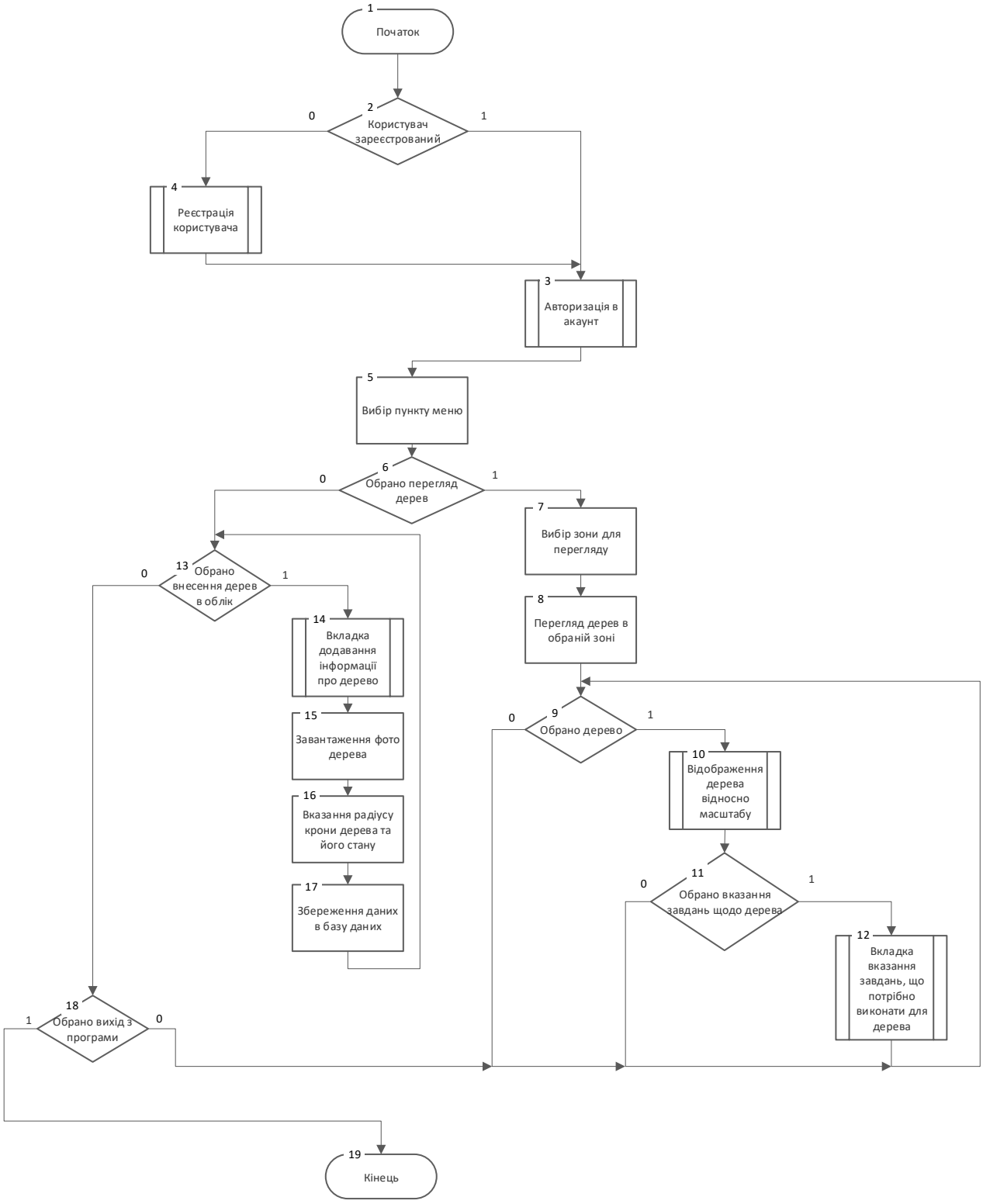

Рисунок 3.8 – Загальний алгоритм роботи системи

Було розроблено структурні схеми цього модуля. Структурна схема вкладки

детальної інформації про дерево наведена на рисунку 3.9.

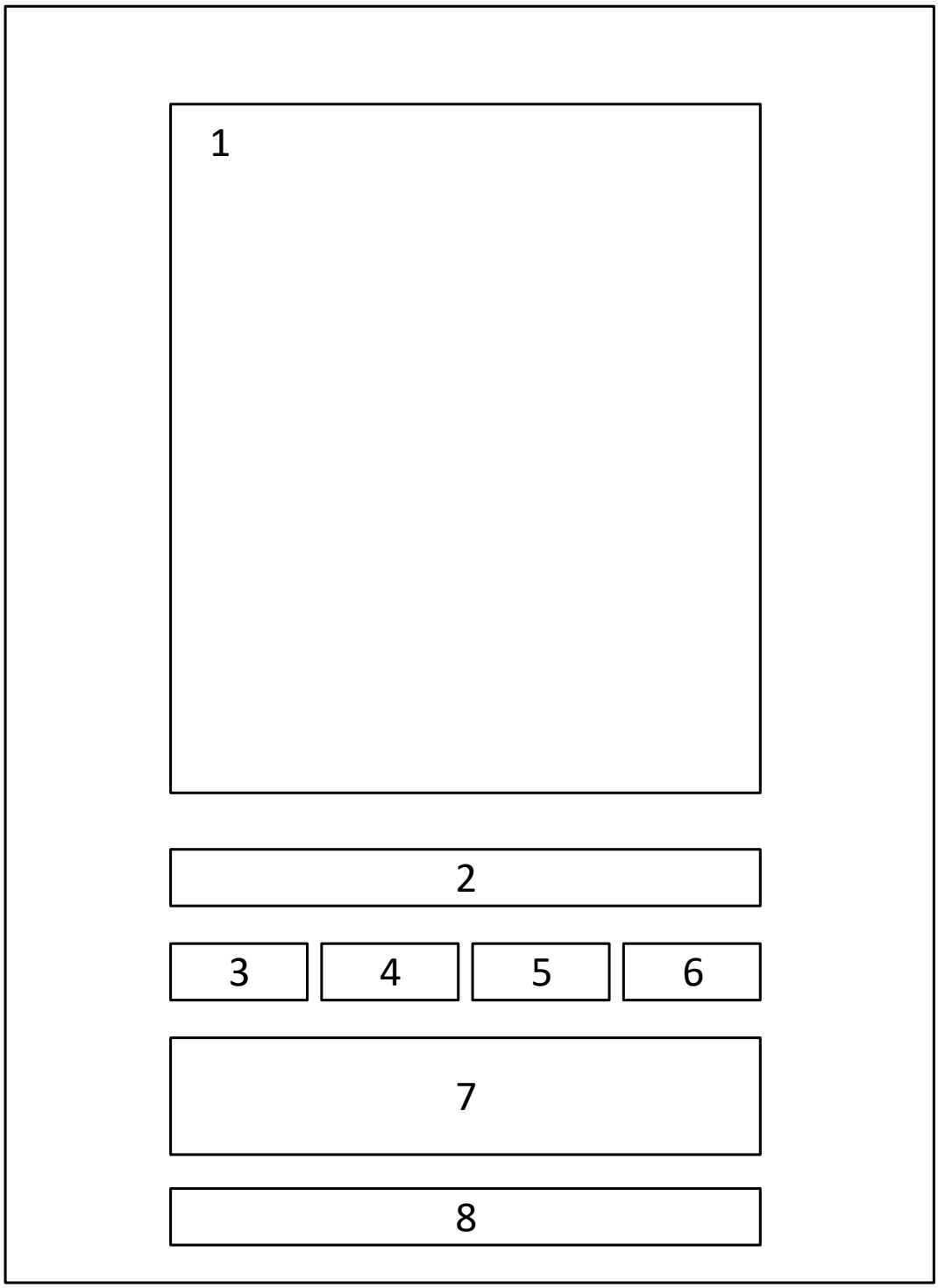

Рисунок 3.9 – Структурна схема вікна детальної інформації про дерево

Елементи вкладки детальної інформації про дерево:

1. Зображення дерева.
2. Реєстраційний номер дерева.

3. Тип дерева.
4. Радіус дерева.
5. Стан дерева.
6. Дата народження дерева.
7. Опис дерева.
8. Перелік робіт для дерева.

Крім цього, було розроблено вікно додавання нового дерева, структурна схема якого подана на рисунку 3.10.

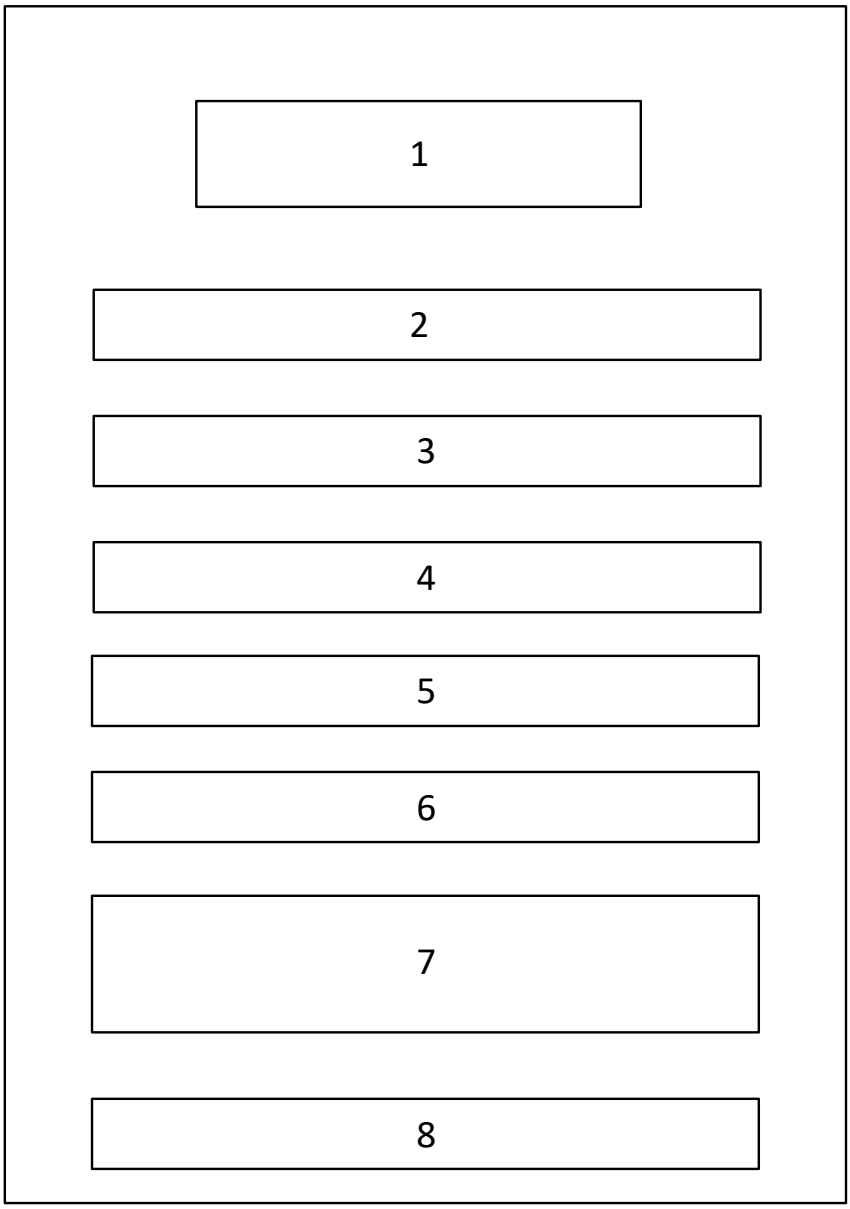

Рисунок 3.10 – Структурна схема вікна додавання нового дерева

Елементи вікна додавання нового вікна:

1. Координати дерева.
2. Радіус дерева.
3. Дата народження дерева.
4. Стан здоров’я дерева.
5. Вид дерева.
6. Завдання, що необхідно виконати по догляду за деревом.
7. Меню завантаження фото дерева.
8. Кнопка «Додати дерево».

3.5 Інтеграція клієнтської та серверної частини

Для повноцінної роботи система моніторингу стану дерев, необхідно об’єднати розроблений функціонал серверної та клієнтської частини.

Так як для завантаження даних з серверної частини системи у клієнтську необхідно перетворити надані серверною частиною JSON-об’єкти у звичайні об’єкти, потрібно для цього створити відповідні класи, об’єкти яких будуть створюватися при передачі даних.

Було створено основні класи Tree, TreeStatus, TreeType та Task, діаграма класів яких зображена на рисунку 3.11.

Клас Tree містить такі поля для ідентифікатора дерева (id), реєстраційного номера (registrationNumber), координат дерева (x,y), радіусу дерева (radius), дату посадки дерева (birthDate), посилання на фото дерева (photoUrl), стану дерева (state), завдань по догляду за деревом (tasks).

Клас TreeType містить інформацію про типи дерев та складається з ідентифікатора типу дерева (id) та назви типу (name).

Клас Task описує завдання, які необхідно виконати під час догляду за деревом та містить поля: ідентифікатор (id) та назву процедури (name).

TreeStatus перелічує усі варіанти стану дерев. Дерево може бути здоровим (HEALTHY), хворим (ILL), та таким, що вимагає зрубування (DISREPAIR).

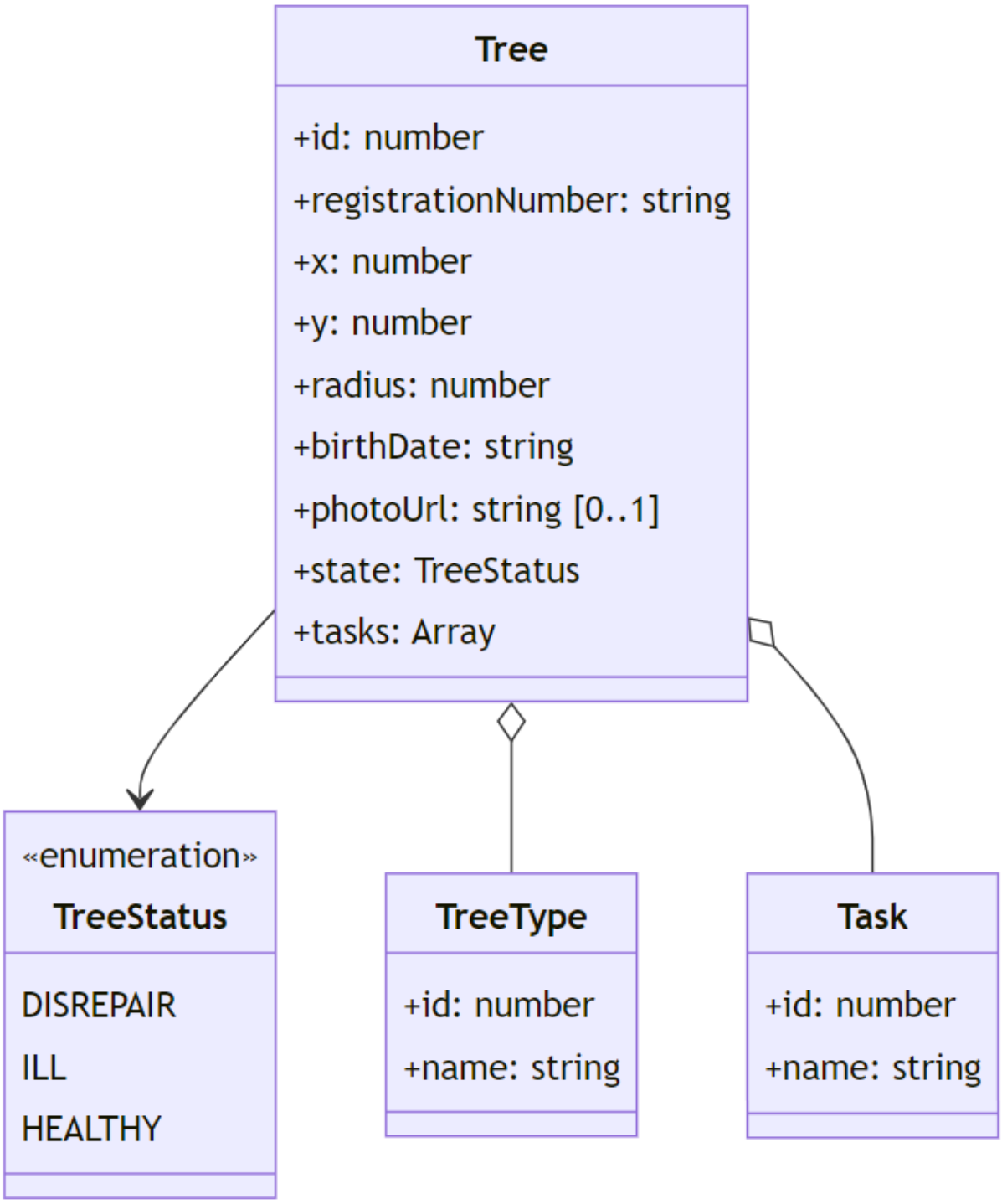

Рисунок 3.11 – Діаграма класів для опису дерев

Для повноцінної роботи системи було створено декілька інших класів: TreeShort, Coordinates, User. Діаграму цих класів наведено на рисунку 3.12.

TreeShort є скороченою версією класу Tree. Об’єкти цього класу використовуються при відображенні дерев на карті, де не потрібними є такі дані, як завдання для догляду за деревами та їх дата посадки, які не відображаються при

перегляді дерев на карті.

Клас Coordinates містить в собі довготу та широту, на якій розміщено дерево.

Клас User містить дані користувача, такі як ідентифікатор (id), повне ім’я (fullname) та електронна адреса (email).

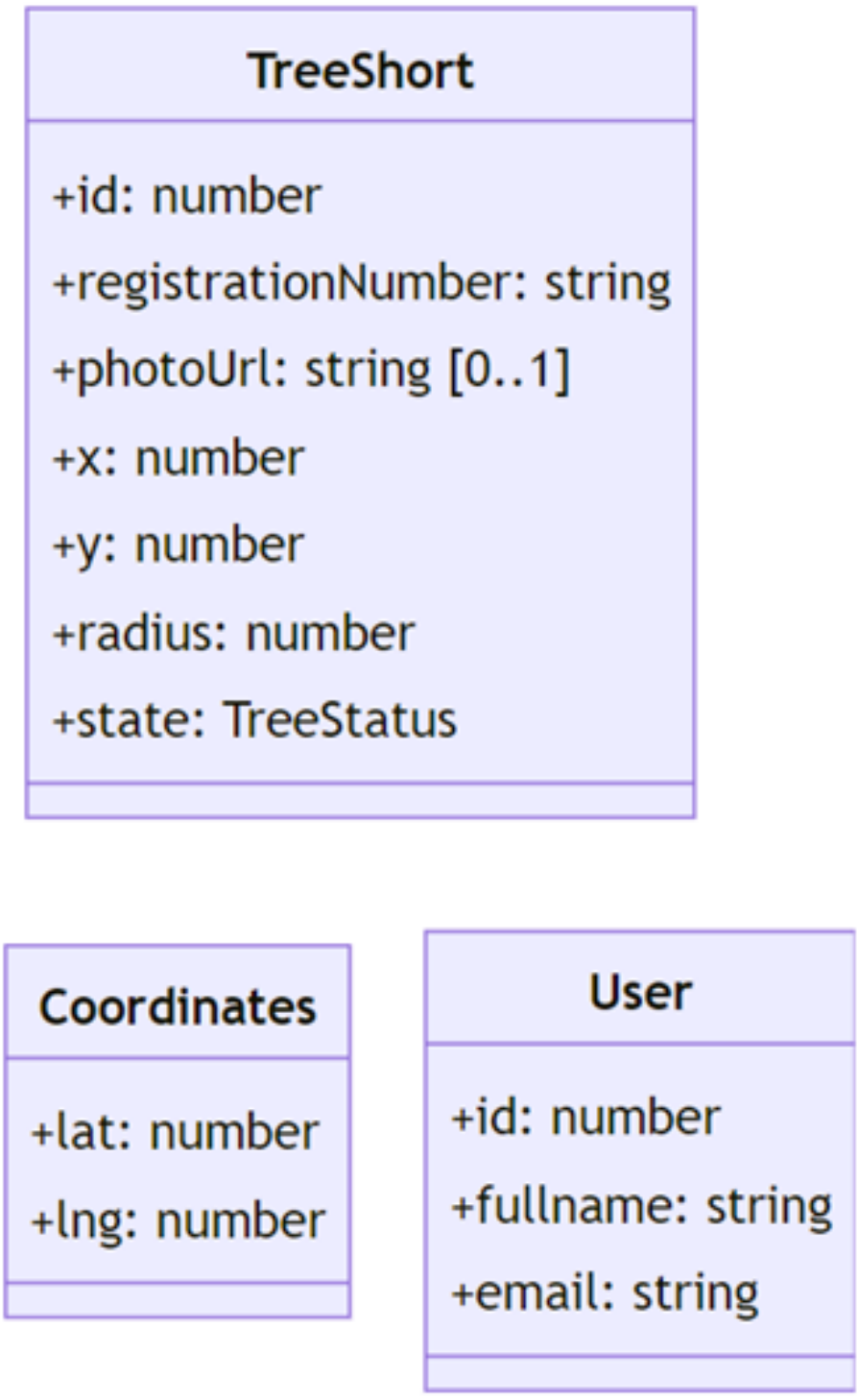

Рисунок 3.12 – Діаграма класів інших класів системи

Взаємодія між серверною та клієнтською частинами описані за допомогою діаграм послідовності. Так, діаграма послідовності процесу авторизації користувача наведена на рисунку 3.13. В ній описується процес введення даних користувача, їх перевірка на правильність з відповідним результатом, перехід на головну сторінку та завантаження карти з деревами, що додані в базу даних системи.

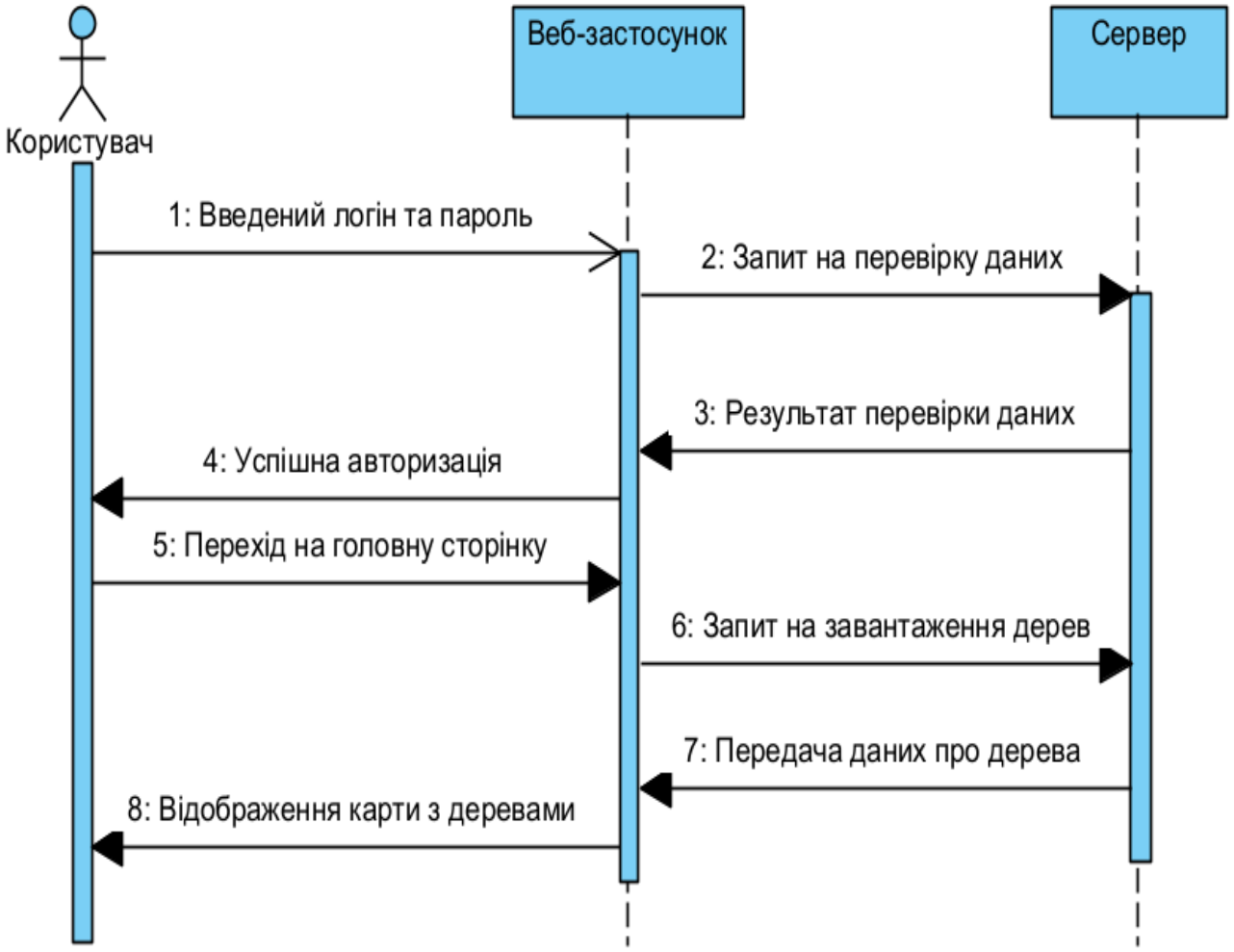

Рисунок 3.13 – Діаграма послідовності процесу авторизації користувача

Діаграма послідовності процесу реєстрації користувача наведено на рисунку 3.14. В ній описується процес введення даних для реєстрації, їх перевірка на відповідність вимогам, відправка електронного листа та створення аккаунту користувача.

Рисунок 3.14 – Діаграма послідовності процесу реєстрації

Також було створено діаграму послідовності процесу вибору дерева на карті та перегляду його деталей (рисунок 3.15).

Рисунок 3.15 – Діаграма послідовності процесу вибору дерева на карті та перегляду його деталей

Ця діаграма описує процес відкриття карти з деревами, приближення масштабу (вибір області з деревами) туди, куди йому потрібно з подальшим вибором конкретного дерева та переглядом інформації про нього.

3.5 Висновки

У третьому розділі було проведено варіантний аналіз та здійснено вибір мови програмування та середовища розробки клієнтської частини веб-системи моніторингу стану дерев з використанням Google Maps. Було розроблено клієнтську частину модуля реєстрації та авторизації та клієнтську частину модуля роботи з деревами.

4 ТЕСТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБ-СИСТЕМИ

4.1 Аналіз методів тестування

Тестування інтерфейсу користувача веб-застосунку є важливим етапом перевірки якості програмного забезпечення, який гарантує, що веб-сайт або додаток працює належним чином і забезпечує приємне та інтуїтивне використання для кінцевих користувачів. Цей процес включає перевірку візуальних елементів, функціональності, навігації та взаємодії користувача з інтерфейсом.

Під час тестування інтерфейсу користувача перевіряються такі візуальні компоненти, як розміщення елементів, кольорова гама, шрифти, зображення та відповідність дизайну загальному стилю веб-застосунку. Це допомагає забезпечити єдиний та привабливий вигляд на всіх сторінках застосунку та покращити загальне враження від продукту.

Функціональне тестування інтерфейсу користувача передбачає перевірку правильності роботи всіх елементів інтерфейсу, таких як кнопки, меню, форми введення, посилання та інші взаємодії. Це дозволяє виявити та виправити помилки та неточності, які можуть вплинути на роботу веб-застосунку та завадити користувачам виконувати бажані дії.

Навігація є ключовим аспектом інтерфейсу користувача, і її тестування має на меті перевірити, чи можуть користувачі легко та інтуїтивно переміщуватися між сторінками та розділами веб-застосунку. Це включає перевірку роботи меню, посилань, фільтрів, пошуку та інших навігаційних елементів, а також виявлення та виправлення можливих проблем, пов'язаних з навігацією.

Тестування взаємодії користувача з інтерфейсом допомагає оцінити, наскільки легко та зручно користувачі можуть виконувати бажані дії в веб-застосунку. Це включає перевірку реакції інтерфейсу на різні дії користувачів, такі як натискання кнопок, введення даних у форми, прокрутка сторінок та інші взаємодії. Тестування взаємодії допомагає виявити та виправити проблеми з відгуком інтерфейсу, швидкістю роботи та загальною зручністю використання.

Загалом, тестування інтерфейсу користувача веб-застосунку є невід'ємною

частиною забезпечення якості програмного забезпечення та покращення загального досвіду користувачів. Ретельне тестування допомагає виявити та виправити проблеми з дизайном, функціональністю, навігацією та взаємодією, що сприяє створенню приємного та інтуїтивного інтерфейсу для кінцевих користувачів.

Тестування веб-застосунків з вбудованою картою Google Maps має деякі особливості, які вимагають уваги та спеціального підходу. Основні аспекти тестування таких веб-застосунків включають перевірку коректності відображення карти, функціональності, продуктивності та сумісності з різними пристроями та браузерами.

Під час тестування веб-застосунку з вбудованою картою Google Maps необхідно перевірити, чи відображається карта коректно, з правильним масштабом, центруванням та маркерами. Також потрібно перевірити, чи працюють всі необхідні шари карти, такі як супутникові знімки, рельєф та дорожня карта.

Функціональне тестування веб-застосунку з вбудованою картою Google Maps включає перевірку роботи всіх функцій, пов'язаних з картою, таких як пошук адреси, маршрутизація, фільтри, геокодування та зворотне геокодування. Необхідно перевірити, чи працюють всі ці функції належним чином та забезпечують коректні результати.

Веб-застосунки з вбудованою картою Google Maps можуть вимагати значних ресурсів для завантаження та відображення карти, особливо при використанні великої кількості маркерів або при роботі з великими областями. Тестування продуктивності допомагає визначити, чи працює веб-застосунок швидко та плавно, і виявити можливі проблеми з продуктивністю, пов'язані з використанням карти.

Веб-застосунки з вбудованою картою Google Maps повинні працювати коректно на різних пристроях та в різних браузерах. Тестування сумісності допомагає перевірити, чи відображається та працює карта належним чином на різних типах пристроїв, таких як смартфони, планшети та настільні комп'ютери, а також в різних браузерах, таких як Google Chrome, Mozilla Firefox, Safari, Microsoft Edge та інші.

Google Maps API надає широкі можливості для взаємодії з картою та

використання різних функцій. Тестування API допомагає перевірити, чи коректно використовуються всі необхідні методи та параметри API, і виявити можливі помилки або неточності в їх використанні.

Веб-застосунки з вбудованою картою Google Maps можуть містити конфіденційну інформацію, таку як адреси та геолокацію користувачів. Тестування безпеки допомагає перевірити, чи захищена ця інформація від несанкціонованого доступу та використання.

Тестування веб-застосунків з вбудованою картою Google Maps вимагає спеціального підходу та уваги до особливостей роботи з картою. Ретельне тестування допомагає забезпечити коректну роботу веб-застосунку, високу продуктивність, сумісність з різними пристроями та браузерами та захист конфіденційної інформації користувачів.

4.2 Тестування клієнтської частини

Для перевірки реалізації клієнтської частини, проведемо тестування її функціоналу. Спершу, відкриємо сторінку реєстрації та створимо новий аккаунт. На рисунку 4.1 можна побачити, що при введенні неправильного формату електронної пошти, виводиться відповідна помилка.

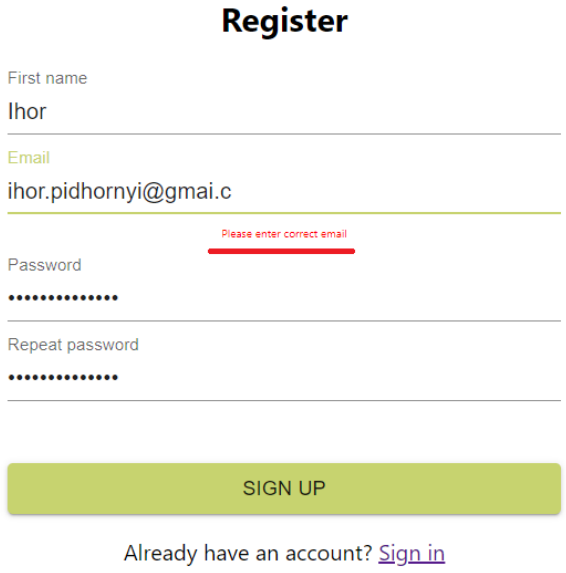

Рисунок 4.1 – Виведення помилки при введенні неправильної пошти

Виправимо цю помилку та спробуємо ще раз (рисунок 4.2). Тепер, помилка не відображається.

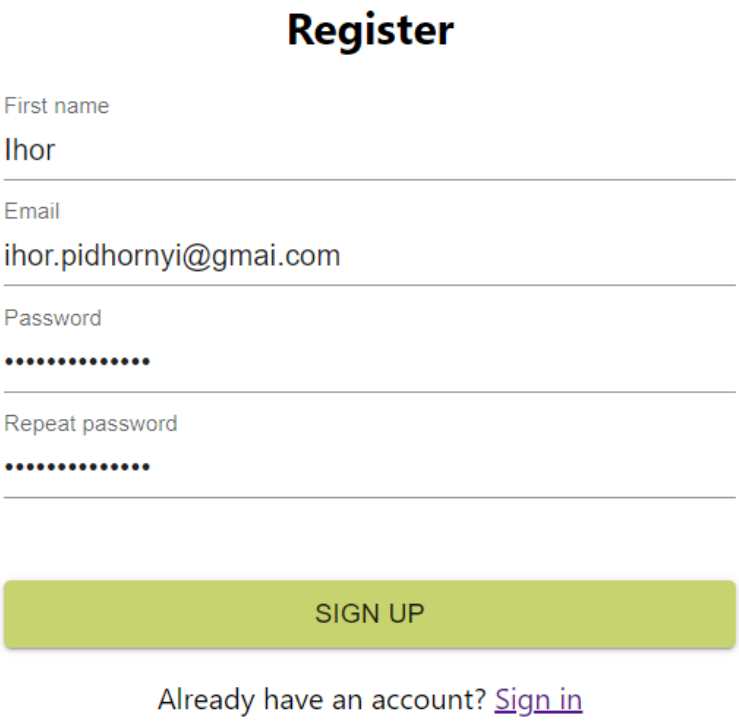

Рисунок 4.2 – Вікно реєстрації

Після успішної реєстрації, автоматично переходимо на вікно авторизації (рисунок 4.3).

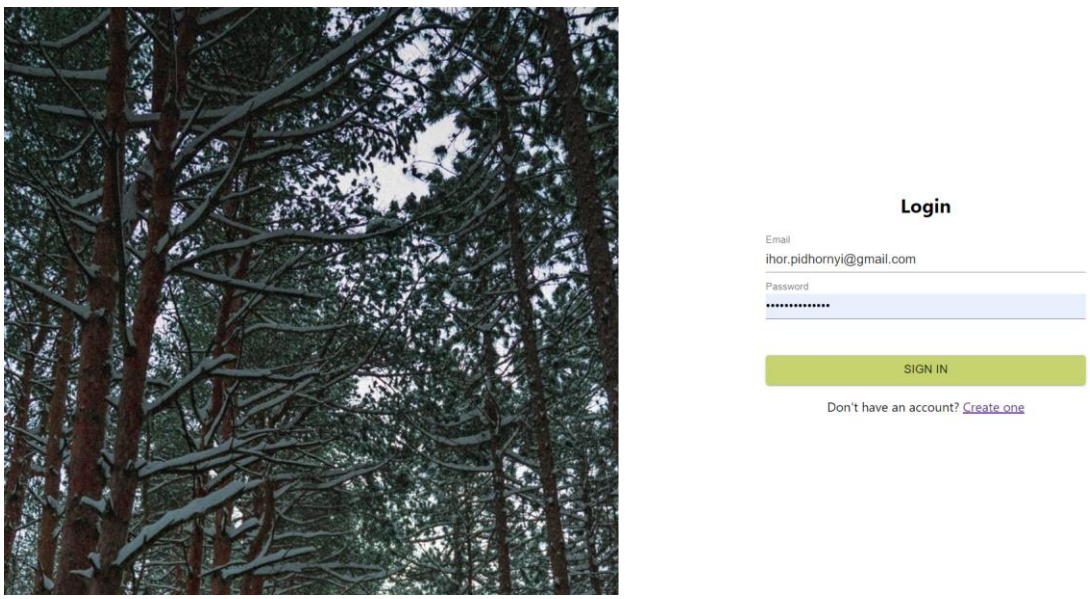

Рисунок 4.3 – Вікно авторизації

Після успішної авторизації, переходимо на головну сторінку системи (рисунок 4.4). На ній ми можемо побачити карту дерев за список дерев у лівій частині екрану.

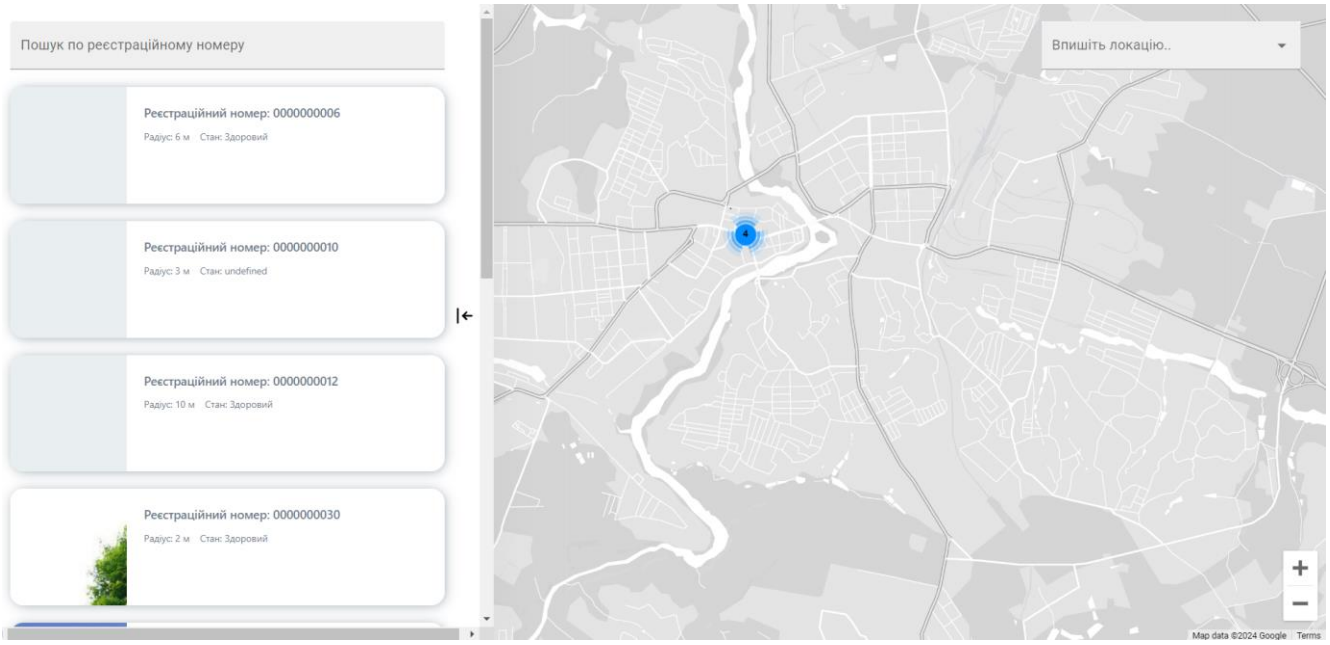

Рисунок 4.4 – Головна сторінка системи

Приблизимо карту. Можемо побачити окремі позначки дерев. На рисунку 4.5 зображено поодиноке дерево, що помічене зеленим колом.

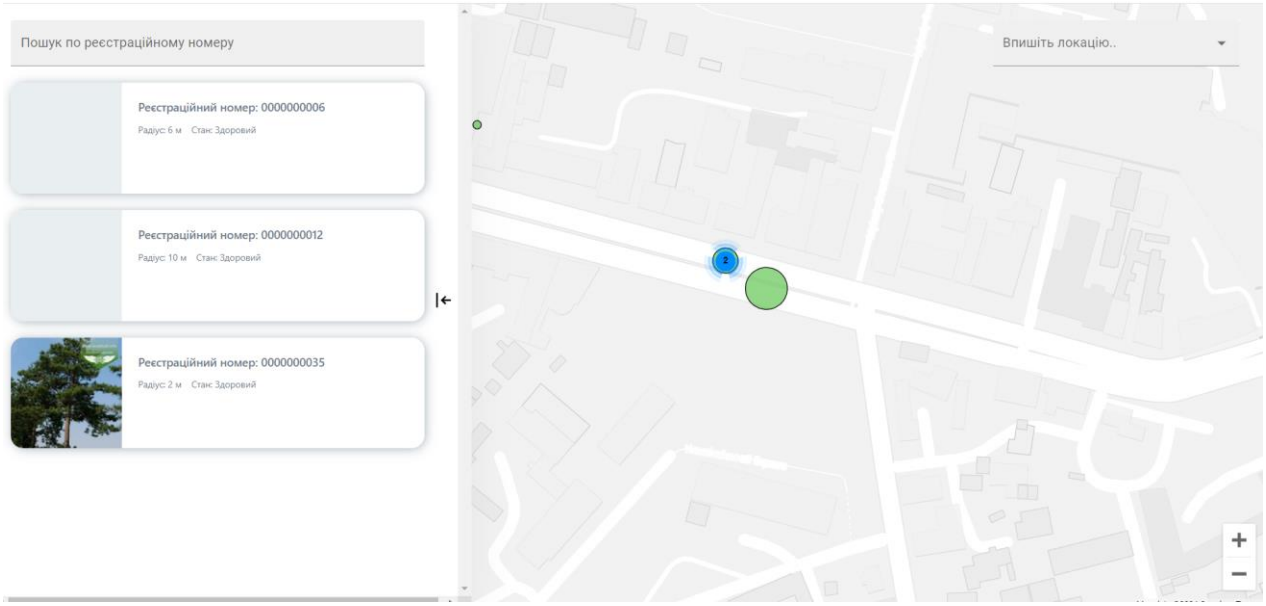

Рисунок 4.5 – Відображення окремого дерева

Оберемо це дерево. Відкриється нове вікно, як на рисунку 4.6. Воно містить детальну інформацію про дерево.

Рисунок 4.6 – Вікно детальної інформації про дерево

Додамо нове дерево. Для цього, оберемо точку на карті для нього (рис. 4.7).

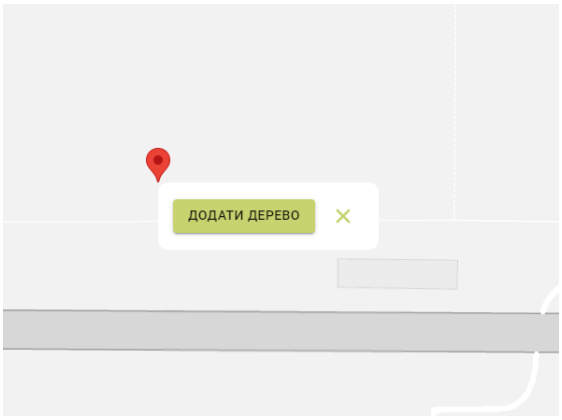

Рисунок 4.7 – Вибір точки на карті для нового дерева

Після цього, натиснемо «Додати дерево». З’явиться вікно додавання дерева (рисунок 4.8).

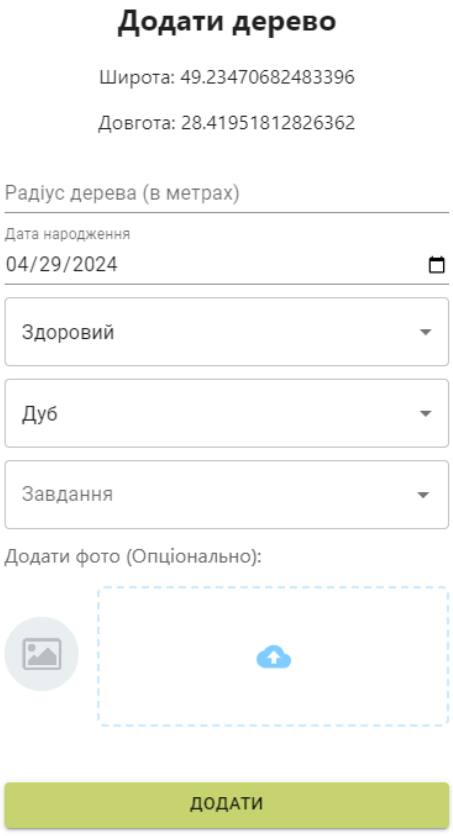

Рисунок 4.8 – Вікно додавання нового дерева

Заповнимо інформацію про дерево. Введемо його радіус, дату народження, стан, вид дерева, завдання для дерева та фото. Результат зображено на рисунку 4.9.

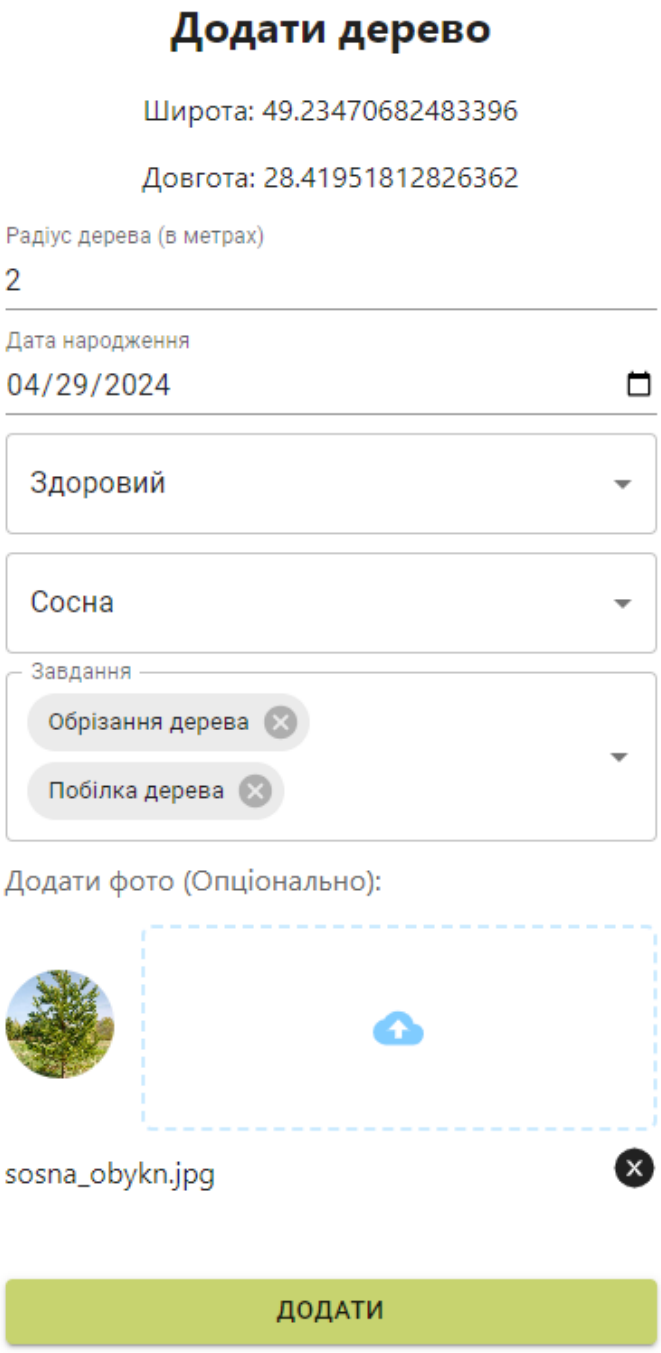

Рисунок 4.9 – Заповнена інформація про дерево

Додане на карті дерево зображене на рисунку 4.10.

Рисунок 4.10 – Додане дерево на карті

Переглянемо інформацію про дерево у окремому вікні і переконаємося, що вона відображається правильно (рисунок 4.11).

Рисунок 4.11 – Вікно інформації про дерево

При масштабуванні карти, дерева групуються так, як на рисунку 4.12.

Рисунок 4.12 – Групування дерев на карті

Дерева відображаються на карті по-різному. Різні кольори відповідають за різний стан дерев – здорове, хворе або те, що потребує зрубу (рисунок 4.13). Також, залежно від радіусу, дерева мають різний розмір на карті.

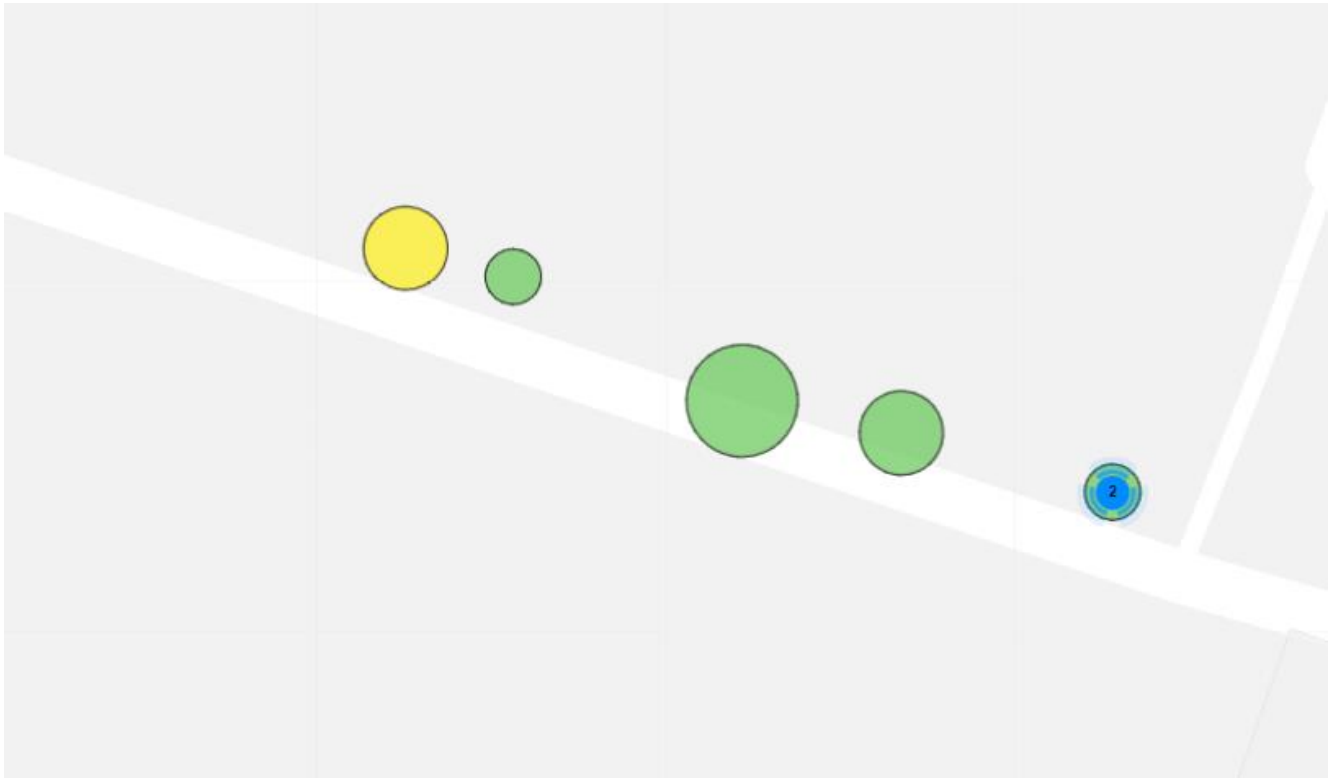

Рисунок 4.13 – Різні відображення дерев на карті

Перевіримо пошук дерев за реєстраційним номером. Приклад такого пошуку

зображено на рисунку 4.14. Тут ми можемо побачити, що при введенні в поле пошуку реєстраційного номеру дерева, воно відображається на карті та у списку дерев.

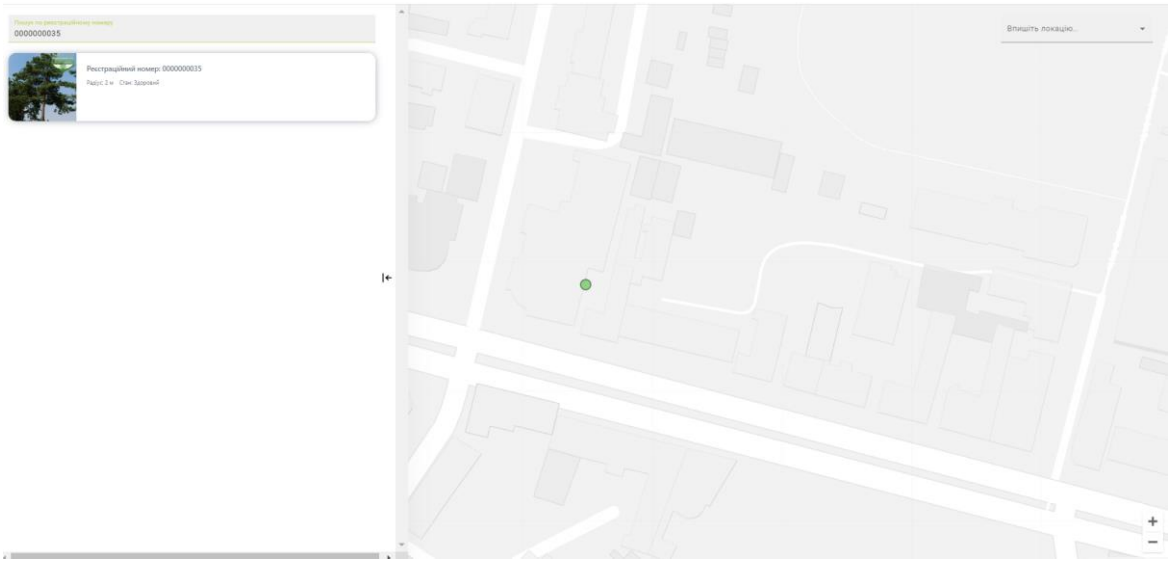

Рисунок 4.14 – Пошук дерева за реєстраційним номером

Якщо увести інший реєстраційний номер, автоматично в списку дерев відобразиться введене дерево та відбудеться переміщення курсора карти туди, де розміщене це дерево (рисунок 4.15).

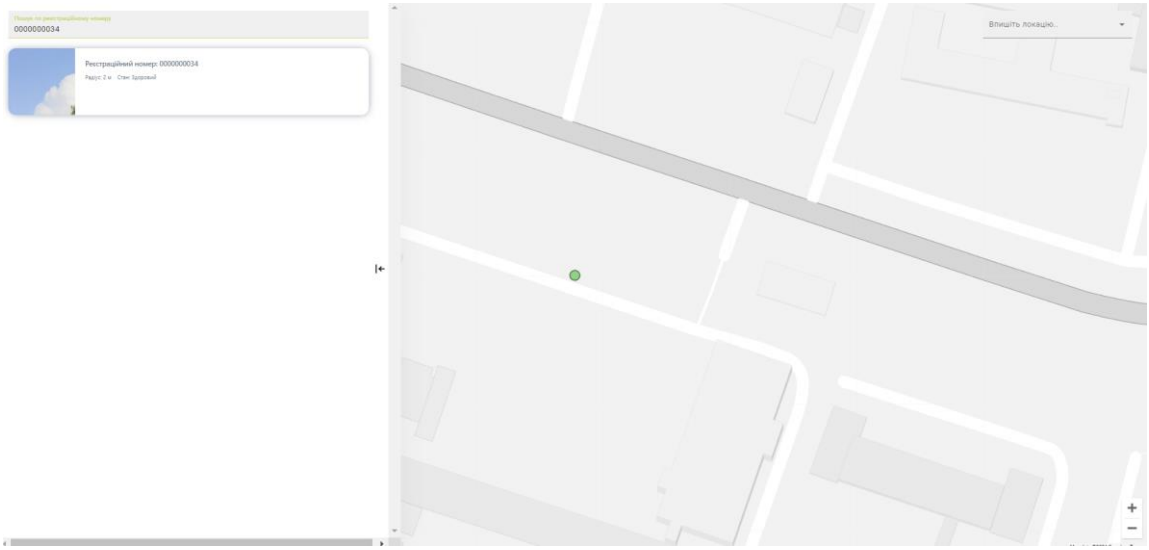

Рисунок 4.15 – Переміщення карти в нове положення

4.3 Висновок

У четвертому розділі було проведено аналіз методів тестування клієнтської частини веб-системи моніторингу стану дерев. Було проведено тестування клієнтської частини веб-системи моніторингу стану дерев, перевірено різні сценарії використання системи. Продемонстровано, що система виконує поставлені на початку розробки задачі.

ВИСНОВКИ

У результаті виконання бакалаврської дипломної роботи було розроблено модуль клієнта автоматизованої системи моніторингу стану дерев з використанням Google Maps. Розроблена клієнтська частина надає доступ до інтерфейсу користувача, з використанням якого він може додавати та змінювати інформацію про дерева, переглядати їх на карті та у вигляді списку, переглядати детальну інформацію про дерево в окремому вікні.

Для розробки було використано середовище розробки Visual Studio Code, мову програмування TypeScript та фреймоворк React.

При виконанні бакалаврської дипломної роботи було здійснено:

- розроблено модель, метод та алгоритми роботи клієнтської частини вебсистеми;
- розроблено модуль реєстрації та авторизації користувача;
- інтегровано карти Google Maps в систему;
- розроблено інтерфейс користувача для моніторингу стану дерев;
- забезпечено відображення дерев на карті;
- проведено тестування роботи клієнтської частини вебсистеми та підтверджено її працездатність.

Бакалаврська дипломна робота оформлена згідно методичних вказівок [21].

У першому розділі було проведено аналіз стану питання клієнтської частини, порівняльний аналіз аналогів, аналіз методів розв’язання задачі. На їх основі було доведено доцільність та актуальність власної розробки, поставлено задачі дослідження. У другому розділі було розроблено структуру та модель клієнтської частини, метод позначення дерев на карті, модель клієнтської частини. У третьому розділі було проведено варіантний аналіз та обґрунтування вибору мови програмування розробки та середовища розробки, розроблено програмні модулі клієнтської частини та інтерфейс користувача.У четвертому розділі було проведено аналіз методів тестування, проведено тестування розробленої клієнтської частини. Тестування довело працездатність та коректність роботи застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Tree monitoring system [Електронний ресурс] – Режим доступу до ресурсу: https://www.lifeterra.eu/en/tree-monitoring-technology

2. Street tree inventory software [Електронний ресурс] – Режим доступу до ресурсу: https://www.natural-solutions.world/blog/street-tree-inventory-software

3. Войтко В.В. Розробка автоматизованої системи моніторингу стану дерев з використанням GOOGLE MAPS / В. В. Войтко, Р.О. Ковальчук, І.М. Підгорний // Матеріали конференції «LIII Науково-технічна конференція підрозділів Вінницького національного технічного університету (2024) », Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20440/16969

4. Войтко В.В. Розгортання автоматизованої системи моніторингу стану дерев з використанням GOOGLE MAPS та впровадженням підходу CI/CD / В.В. Войтко, І.М. Підгорний, Р.О. Ковальчук // The 4th International scientific and practical conference “Perspectives of contemporary science: theory and practice” (May 26-28, 2024) SPC “Sci conf.com.ua”, Lviv, Ukraine. 2024. 1983 p. — P. 499 - 502.

5. Treeplotter [Електронний ресурс] – Режим доступу до ресурсу: https://planitgeo.com/treeplotter/

6. Arbor [Електронний ресурс] – Режим доступу до ресурсу: https://arborjet.com/app/

7. PlantsSnap [Електронний ресурс] – Режим доступу до ресурсу: https://www.plantsnap.com/

8. React [Електронний ресурс] – Режим доступу до ресурсу: https://legacy.reactjs.org/

9. Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/

10. Vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/

11. Prettier [Електронний ресурс] – Режим доступу до ресурсу: https://prettier.io/

12. Google Maps Platform [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/maps

13. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

14. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://w3schoolsua.github.io/js/index.html#gsc.tab=0

15. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/

16. Ruby [Електронний ресурс] – Режим доступу до ресурсу: https://www.ruby-lang.org/

17. PHP [Електронний ресурс] – Режим доступу до ресурсу: https://www.php.net/

18. Atom [Електронний ресурс] – Режим доступу до ресурсу: https://atom-editor.cc/

19. VS Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/

20. Sublime Text [Електронний ресурс] – Режим доступу до ресурсу: https://www.sublimetext.com/

21. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

61

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

УЗГОДЖЕНО
НАЧАЛЬНИК ДЕРЖАВНОЇ
ЕКОЛОГІЧНОЇ ІНСПЕКЦІЇ У
ВІННИЦЬКІЙ ОБЛАСТІ
Дубовий Ю.В.
« 12 » березня 2024 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 12 » березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка автоматизованої системи моніторингу стану дерев з використанням Google Maps. Частина 2. Клієнтська частина»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доц. В.В. Войтко
« 12 » березня 2024 року.

Виконав:
_____ студент гр. ІПІ-20б І.М. Підгорний
« 12 » березня 2024 року.

м. Вінниця – 2024 рік

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка автоматизованої системи моніторингу стану дерев з використанням Google Maps. Частина 2. Клієнтська частина».

Галузь застосування – веб-технології.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №17 від 20 березня 2024 р.

3. Мета та призначення розробки.

Метою бакалаврської дипломної роботи є підвищення можливостей моніторингу стану дерев за рахунок розробки і використання клієнтської частини вебсистеми, яка включає створення графічного інтерфейсу користувача для опису дерев з використанням Google Maps, що дозволить покращити процес моніторингу стану дерев та своєчасного формування рекомендацій щодо підтримки зелених насаджень.

4. Вихідні дані для проведення НДР

1. Дерева в місті: як правильно доглядати і коли необхідний спил [Електронний ресурс] – Режим доступу до ресурсу: https://nikvesti.com/ua/news/business/dereva-v-misti-yak-doglyadati-spylyuvati-neobkhidno-2024-ua

2. TREES IN OUR CITIES: 10 REASONS WE NEED TO PLANT MORE [Електронний ресурс] – Режим доступу до ресурсу: https://www.treesforcities.org/stories/trees-in-our-cities-10-reasons-we-need-to-plant-more

3. JWT [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/

4. Gerardus Blokdyk. Database Design Best Practices A Complete Guide. Бірмінгем: Packt Publishing, 2023. 294с.

5. Технічні вимоги

Вхідні дані — дані про користувача, інформація про дерева.

Вихідні дані — зображені на карті дерева та їх візуалізація.

6. Конструктивні вимоги.

Інтерфейс програми поовинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

–пояснювальна записка до бакалаврської дипломної роботи;

–технічне завдання;

–лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз стану питання розробки клієнтської частини системи	14.03.2024 – 24.03.2024
2	Розробка структури клієнтської частини	25.03.2024 – 07.04.2024
3	Варіантний аналіз та вибір засобів розробки	08.04.2024 – 18.04.2024
4	Розробка клієнтської частини системи моніторингу стану дерев	19.04.2024 – 19.05.2024
5	Тестування клієнтської частини	20.05.2024 – 24.05.2024
6	Оформлення матеріалів до захисту БДР	25.05.2024 – 30.05.24

10 Порядок контролю та прийняття.

Усі етапи бакалаврської роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

64

Додаток Б – Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка автоматизованої системи моніторингу стану дерев з використанням Google Maps. Частина 2. Клієнтська частина»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Войтко В.В.

Оригінальність	96,1%
Схожість	3,9%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Підгорний І.М.

Керівник роботи _____ Войтко В.В.

65

Додаток В – Акт впровадження (Державна екологічна інспекція у Вінницькій області)

АКТ

про впровадження результатів бакалаврської дипломної роботи «Розробка автоматизованої системи моніторингу стану дерев з використанням Google Maps. Частина 2. Клієнтська частина» студента ВНТУ Підгорного Ігоря Миколайовича

Результати бакалаврської дипломної роботи студента ВНТУ Підгорного І.М., що полягають в розробці клієнтської частини автоматизованої системи моніторингу стану дерев з використанням Google Maps, прийняті до використання у Державній екологічній інспекції у Вінницькій області.

Начальник Державної екологічної інспекції у Вінницькій області Дубовий Ю.В.

Додаток Г – Лістинг програми

Map.tsx

```
import {
  Circle,
  GoogleMap,
  Marker,
  MarkerClusterer,
} from '@react-google-maps/api'
import { useCallback, useEffect, useMemo, useRef, useState } from 'react'
import { CloseButton, Container, CreateTree } from './Map.styled'
import { Coordinates } from '../../shared/models/coordinates'
import { defaultTheme } from './Theme'
import { useNavigate } from 'react-router-dom'
import { Button, debounce } from '@mui/material'
import { TreeShort } from '../../shared/models/tree-short'
import { MapsAutocomplete } from '../MapsAutocomplete/MapsAutocomplete'
import { treeStatusColorsMap } from '../../shared/consts/treeStatusColorsMap'
import API from '../../shared/services/api'
import { useGlobalContext } from '../../shared/context/GlobalContext'

const containerStyle = {
  width: '100%',
  height: '100%',
}

const optionsClusterer = {
  imagePath:
    'https://developers.google.com/maps/documentation/javascript/examples/markerclusterer/m', // so you must have
m1.png, m2.png, m3.png, m4.png, m5.png and m6.png in that folder
}

const defaultOptions = {
  panControl: true,
  zoomControl: true,
  mapTypeControl: false,
  scaleControl: false,
  streetViewControl: false,
  rotateControl: false,
  clickableIcons: false,
  keyboardShortcuts: false,
```

```
  scrollWheel: false,
  disableDoubleClickZoom: false,
  fullscreenControl: false,
  styles: defaultTheme,
}

export const Map = ({ handleCreateTree }: any) => {
  const { trees, setTrees, filterIpn, setFilterIpn } = useGlobalContext()
  const mapRef = useRef<google.maps.Map | undefined>(undefined)
  const [createTreeLatLng, setCreateTreeLatLng] =
    useState<google.maps.LatLng | null>(null)
  const [clickXY, setClickXY] = useState<{ x: number; y: number } | null>(null)
  const [zoom, setZoom] = useState<number>(15)
  const [center, setCenter] = useState<Coordinates>({
    lat: 49.233083,
    lng: 28.468217,
  })
  const navigate = useNavigate()

  const onLoad = useCallback(function callback(map: google.maps.Map) {
    mapRef.current = map
  }, [])

  const onUnmount = useCallback(function callback() {
    mapRef.current = undefined
  }, [])

  const onClick = useCallback(function callback(
    event: google.maps.MapMouseEvent
  ) {
    setCreateTreeLatLng(event.latLng)
    if ('screenX' in event.domEvent) {
      setClickXY({ x: event.domEvent.screenX, y: event.domEvent.screenY })
    }
  },
  [])

  const fetch = useCallback(
    (
      bounds: google.maps.LatLngBoundsLiteral | null,
      filterIpn: string | null
    ) => {
```

```
    let params = {}

    if (!filterIpn && bounds) {
      const startY = Math.min(bounds.west, bounds.east)
      const startX = Math.min(bounds.north, bounds.south)
      const endY = Math.max(bounds.west, bounds.east)
      const endX = Math.max(bounds.north, bounds.south)

      params = { startY, startX, endY, endX }
    } else if (!bounds && filterIpn) {
      params = { registration_number: filterIpn }
    } else {
      return
    }

    API.get<TreeShort[]>('/trees', {
      params,
    })
      .then((res) => res.data)
      .then((trees) => {
        if (trees.length === 1 && filterIpn) {
          const tree = trees[0]
          setCenter({ lng: tree.y, lat: tree.x })
        }
        setTrees(trees)
      })
  },
  [setTrees, filterIpn]
)

const onDebounce = useMemo(
  () =>
    debounce((callback) => {
      callback()
    }, 1000),
  []
)

const onValueChanges = useCallback(
  function callback(value: Coordinates) {
    setCenter(value)
  },
```

```
  [setCenter]
)

const onZoomChanged = useCallback(
  function callback() {
    if (mapRef.current) {
      onDebounce(() => {
        const zoom = mapRef.current?.getZoom()
        zoom && setZoom(zoom)
      })
    }
  },
  [onDebounce]
)

const onBoundsChanged = useCallback(
  function callback() {
    if (mapRef.current) {
      onDebounce(() => {
        const newZoom = mapRef.current?.getZoom()
        const bounds = mapRef.current?.getBounds()?.toJSON()
        if (bounds && newZoom && newZoom <= zoom) {
          fetch(bounds, filterIpn)
          setFilterIpn(null)
        }
        newZoom && setZoom(newZoom)
      })
    }
  },
  [onDebounce, fetch, zoom, filterIpn]
)

const openCreateTree = useCallback(() => {
  handleCreateTree(createTreeLatLng?.toJSON())
}, [createTreeLatLng, handleCreateTree])

useEffect(() => {
  navigate('/' + center.lat + ',' + center.lng)
  setClickXY(null)
  setCreateTreeLatLng(null)
}, [center, navigate])
```

```
useEffect(() => {
  fetch(null, filterIpn)
}, [filterIpn])

return (
  <Container>
    {clickXY && (
      <CreateTree x={clickXY.x} y={clickXY.y}>
        <Button variant={'contained'} onClick={openCreateTree}>
          Додати дерево
        </Button>
        <Button
          sx={{
            marginLeft: '.5rem',
            height: '40px',
            padding: '6px 12px 6px 8px',
            display: 'inline-block',
            minWidth: '0px',
            span: {
              marginLeft: '0',
            },
          }}
          endIcon={<CloseButton />}
          onClick={() => {
            setClickXY(null)
            setCreateTreeLatLng(null)
          }}
        />
      </CreateTree>
    )}

    <GoogleMap
      mapContainerStyle={containerStyle}
      options={defaultOptions}
      center={center}
      zoom={zoom}
      onLoad={onLoad}
      onUnmount={onUnmount}
      onClick={onClick}
      onZoomChanged={onZoomChanged}
      onBoundsChanged={onBoundsChanged}
    >
```

```
      <div>
       <div>
        <MarkerClusterer options={optionsClusterer}>
         {(clusterer) => (
          <>
           {trees.map((tree, index) => {
            return (
             <Marker
              key={index}
              clusterer={clusterer}
              position={{ lng: tree.y, lat: tree.x }}
              options={{ visible: false }}
              children={
               <Circle
                key={index}
                onClick={() => navigate(`/tree/${tree.id}`)}
                center={{ lng: tree.y, lat: tree.x }}
                radius={tree.radius}
                options={{
                 fillColor: treeStatusColorsMap[tree.state],
                 strokeWeight: 1,
                 fillOpacity: 0.7,
                 visible: true,
                }}
               />
              }
             />
            )
           })}
          </>
         )}
        </MarkerClusterer>
       </div>

       <MapsAutocomplete valueChanges={onValueChanges} />

       {createTreeLatLng && <Marker position={createTreeLatLng.toJSON()} />}
      </div>
     </GoogleMap>
    </Container>
   )
}
```

CreateTree.tsx

```
import React, { useCallback, useRef, useState } from 'react'
import {
  Autocomplete,
  Button,
  Chip,
  Dialog,
  MenuItem,
  Select,
  TextField,
} from '@mui/material'
import { Controller, useForm } from 'react-hook-form'
import { ErrorMessage } from '@hookform/error-message'
import { toast } from 'react-toastify'
import { AxiosError } from 'axios'
import {
  ErrorText,
  PictureIcon,
  RemoveIcon,
  Wrapper,
} from './CreateTreeForm.styled'
import { createTreeFormFields } from '../../shared/consts/create-tree-form-fields'
import { Coordinates } from '../../shared/models/coordinates'
import { useGlobalContext } from '../../shared/context/GlobalContext'
import { TreeStatus } from '../../shared/models/tree-status'
import { treeStatusMap } from '../../shared/consts/treeStatusMap'
import API from '../../shared/services/api'

type FormData = {
  radius: string
  typeId: string
  birthDate: string
  state: TreeStatus
  tasks: number[]
}

export interface ICreateTreeForm {
  open: boolean
  onClose: (value: string) => void
  coords: Coordinates
}
```

```
const MAX_FILE_SIZE = 1024 * 1024 * 5

function CreateTreeForm({ open, onClose, coords }: ICreateTreeForm) {
  const [imageUrl, setImageUrl] = useState<string | null>(null)
  const { tasks, treeTypes } = useGlobalContext()

  const inputEl = useRef<HTMLInputElement | null>(null)
  const {
    reset,
    control,
    handleSubmit,
    formState: { errors },
  } = useForm<FormData>()

  const getInput = (): HTMLInputElement | null => {
    return inputEl.current instanceof HTMLInputElement ? inputEl.current : null
  }

  const getFile = (): File | null => {
    const input = getInput()

    return input ? input.files?.[0] ?? null : null
  }

  const onFileChange = () => {
    const input = getInput()
    const file = getFile()

    if (file?.size && file?.size > MAX_FILE_SIZE && input) {
      toast.error('Too large image. Max size is 5mb')
      input.value = ''
    } else {
      const reader = new FileReader()

      reader.onloadend = () => {
        setImageUrl(typeof reader.result === 'string' ? reader.result : null)
      }

      file && reader.readAsDataURL(file)
    }
  }
```

```
const removeImage = (event: unknown): void => {
  ;(event as MouseEvent)?.preventDefault()
  const input = getInput()
  if (input) input.value = ''
  setImageUrl(null)
}

const getDateString = useCallback(
  (date = new Date(), withTime = false): string => {
    const pretify = (value: number): string => {
      return value < 10 ? `0${value}` : value.toString()
    }

    return (
      `${pretify(date.getFullYear())}-${pretify(
        date.getMonth() + 1
      )}-${pretify(date.getDate())}` +
      (withTime
        ? ` ${pretify(date.getHours())}:${pretify(
            date.getMinutes()
          )}:${pretify(date.getSeconds())}`
        : '')
    )
  },
  []
)

const submit = async (data: FormData) => {
  try {
    const formData = new FormData()

    formData.append('x', coords.lat.toString())
    formData.append('y', coords.lng.toString())
    formData.append('radius', data.radius)
    formData.append('state', data.state)
    formData.append('type.id', data.typeId)
    formData.append(
      'birthDate',
      getDateString(new Date(data.birthDate), true)
    )
    data.tasks.forEach((taskId, index) => {
```

```
      formData.append(`tasks[${index}].id`, taskId.toString())
    })

    const file = getFile()
    file && formData.append('photo', file)

    const res = await API.post('/trees', formData)
    const resData = res.data

    reset()
    const input = getInput()
    if (input) input.value = ''
    setImageUrl(null)
    toast.success('Created Tree')
  } catch (error: unknown) {
    let message
    if (error instanceof AxiosError) message = error?.response?.data?.error
    else message = String(error)
    toast.error(message)
  }
}

const handleClose = () => {
  onClose('')
}

const getLabelById = useCallback(
  (id: number): string | null => {
    return tasks.find((task) => task.id === id)?.name ?? null
  },
  [tasks]
)

return (
  <Dialog onClose={() => handleClose()} open={open}>
    <Wrapper>
      <div className="half">
        <form className="form" onSubmit={handleSubmit(submit)}>
          <h2 className="title">Додати дерево</h2>
          <label className="form-item">
            <p>Широта: {coords.lat}</p>
          </label>
```

```
<label className="form-item">
  <p>Довгота: {coords.lng}</p>
</label>
{createTreeFormFields.map((el) => (
  <label key={el.name} className="form-item">
    <Controller
      name={el.name}
      rules={el.validation}
      defaultValue={el.defaultValue}
      control={control}
      render={({ field: { onChange, value } }) => (
        <>
          <TextField
            label={el.helpText}
            variant="standard"
            type={el.type}
            onChange={onChange}
            value={value}
            fullWidth={true}
          />
          <ErrorMessage
            errors={errors}
            name={el.name}
            render={({ message }: any) => (
              <ErrorText>{message}</ErrorText>
            )}
          />
        </>
      )}
    />
  </label>
))}

<label className="form-item">
  <Controller
    name={'birthDate'}
    defaultValue={getDateString()}
    control={control}
    render={({ field: { onChange, value } }) => (
      <TextField
        label="Дата народження"
        variant="standard"
```

```
        type="date"
        onChange={onChange}
        value={value}
        fullWidth={true}
       />
     )}
    />
  </label>

  <label className="form-item">
    <Controller
     name={'state'}
     defaultValue={TreeStatus.HEALTHY}
     control={control}
     render={({ field: { onChange, value } }) => (
      <Select
       fullWidth={true}
       value={value}
       label="Статус"
       onChange={onChange}
      >
       {Object.values<TreeStatus>(TreeStatus).map((el) => (
        <MenuItem key={el} value={el}>
         {treeStatusMap[el]}
        </MenuItem>
       ))}
      </Select>
     )}
    />
  </label>

  <label className="form-item">
    <Controller
     name={'typeId'}
     defaultValue={treeTypes[0]?.id + ''}
     control={control}
     render={({ field: { onChange, value } }) => (
      <Select
       fullWidth={true}
       value={value}
       label="Тип дерева"
       onChange={onChange}
```

```
      >
        {treeTypes.map((type) => (
          <MenuItem key={type.id} value={type.id}>
            {type.name}
          </MenuItem>
        ))}
      </Select>
    )}
  />
</label>

<label className="form-item">
  <Controller
    name={'tasks'}
    defaultValue={[]}
    control={control}
    render={({ field: { onChange, value } }) => (
      <Autocomplete
        fullWidth={true}
        multiple
        value={value}
        onChange={(event, newValue) => {
          onChange(newValue)
        }}
        options={tasks.map((task) => task.id)}
        getOptionLabel={(option) => getLabelById(option) ?? ''}
        renderTags={(tagValue, getTagProps) =>
          tagValue.map((option, index) => (
            <Chip
              label={getLabelById(option) ?? ''}
              {...getTagProps({ index })}
              key={index}
            />
          ))
        }
        renderInput={(params) => (
          <TextField {...params} label="Завдання" />
        )}
      />
    )}
  />
</label>
```

```
<label className="form-item">
  <div className="file-label">Додати фото (Опціонально):</div>
  <div className="file-content">
    <div className="preview">
      <img
        src={imageUrl ? imageUrl : 'img/image-placeholder.jpg'}
        alt="tree"
      />
    </div>
    <div className="upload-file-wrapper">
      <PictureIcon />
      <input
        ref={inputEl}
        onChange={onFileChange}
        className="input-file"
        type="file"
        accept="image/*,capture=camera"
      />
    </div>
  </div>
  {imageUrl ? (
    <div className="image-path">
      <div className="image-name">
        {getInput()?.value?.replace(/^.*\\/, '') ?? ''}
      </div>
      <div onClick={removeImage}>
        <RemoveIcon />
      </div>
    </div>
  ) : null}
</label>

<Button
  fullWidth={true}
  variant="contained"
  type="submit"
  sx={{
    marginTop: '35px',
  }}
>
  Додати
```

```
            </Button>
          </form>
        </div>
      </Wrapper>
    </Dialog>
  )
}

export default CreateTreeForm
```

Додаток Д

ІЛЮСТРАТИВНА ЧАСТИНА

Розробка автоматизованої системи моніторингу стану дерев з використанням Google Maps. Частина 2. Клієнтська частина

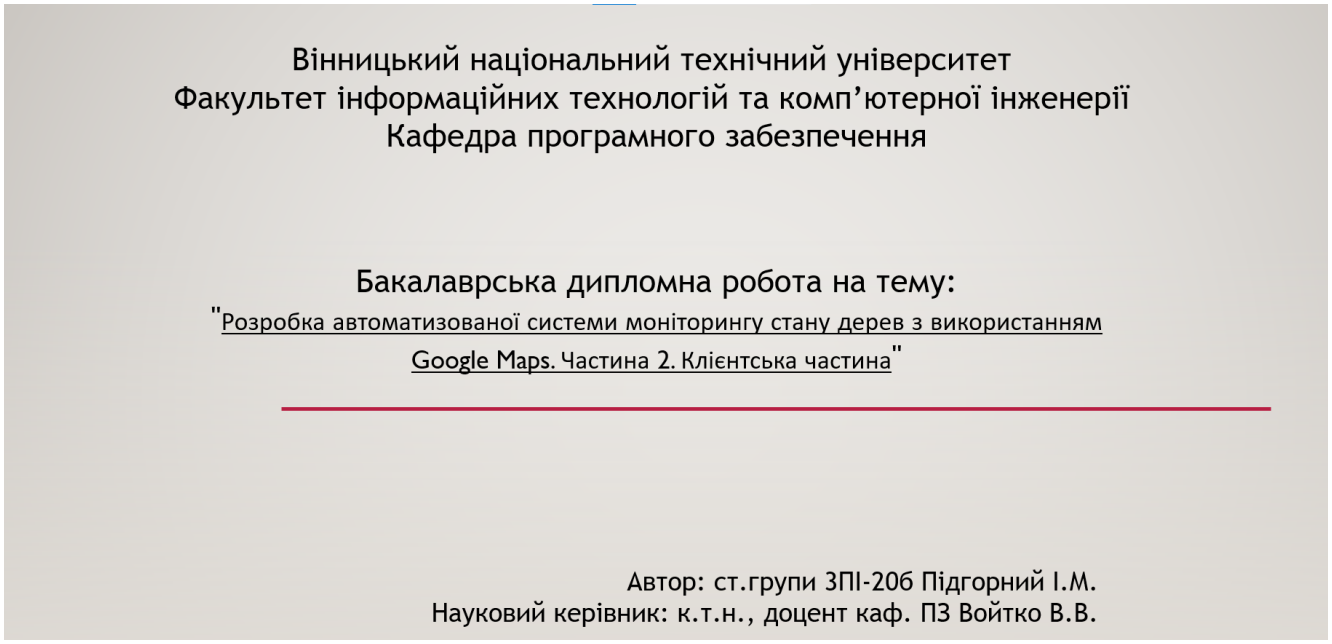

Рисунок Д.1 – Титульний слайд

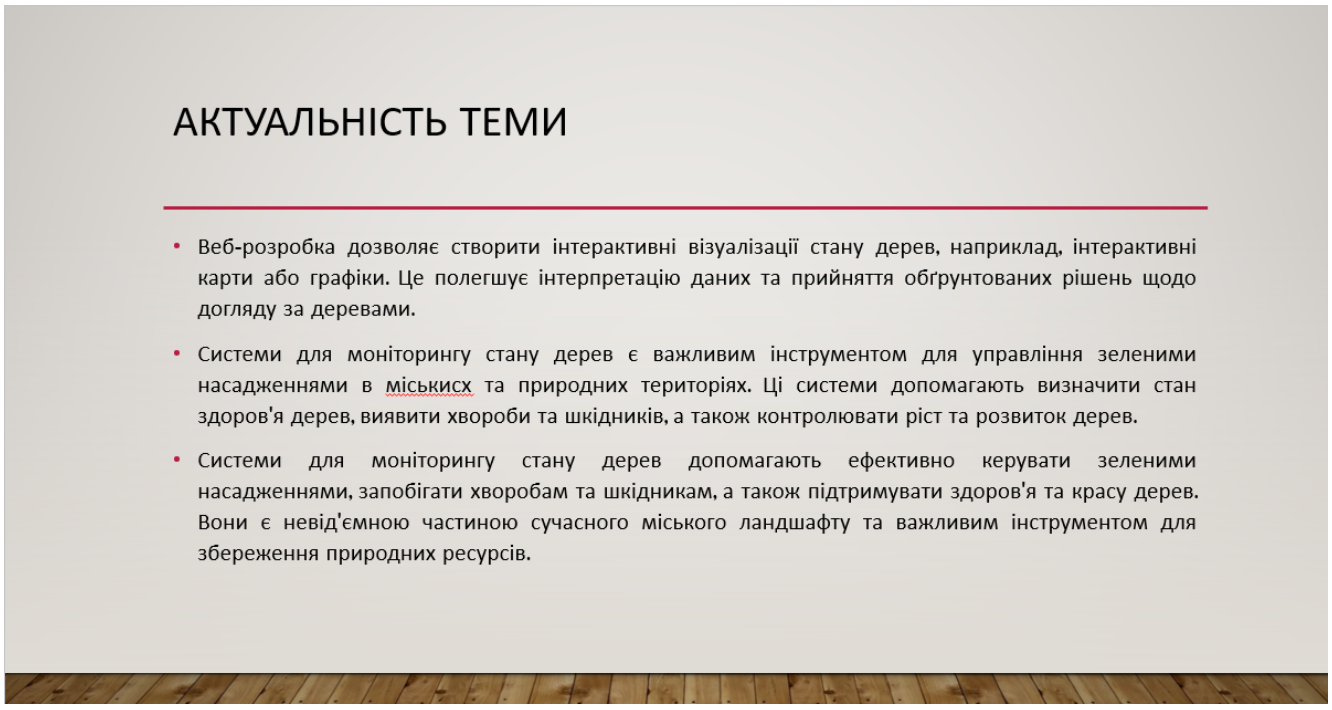

Рисунок Д.2 – Актуальність теми

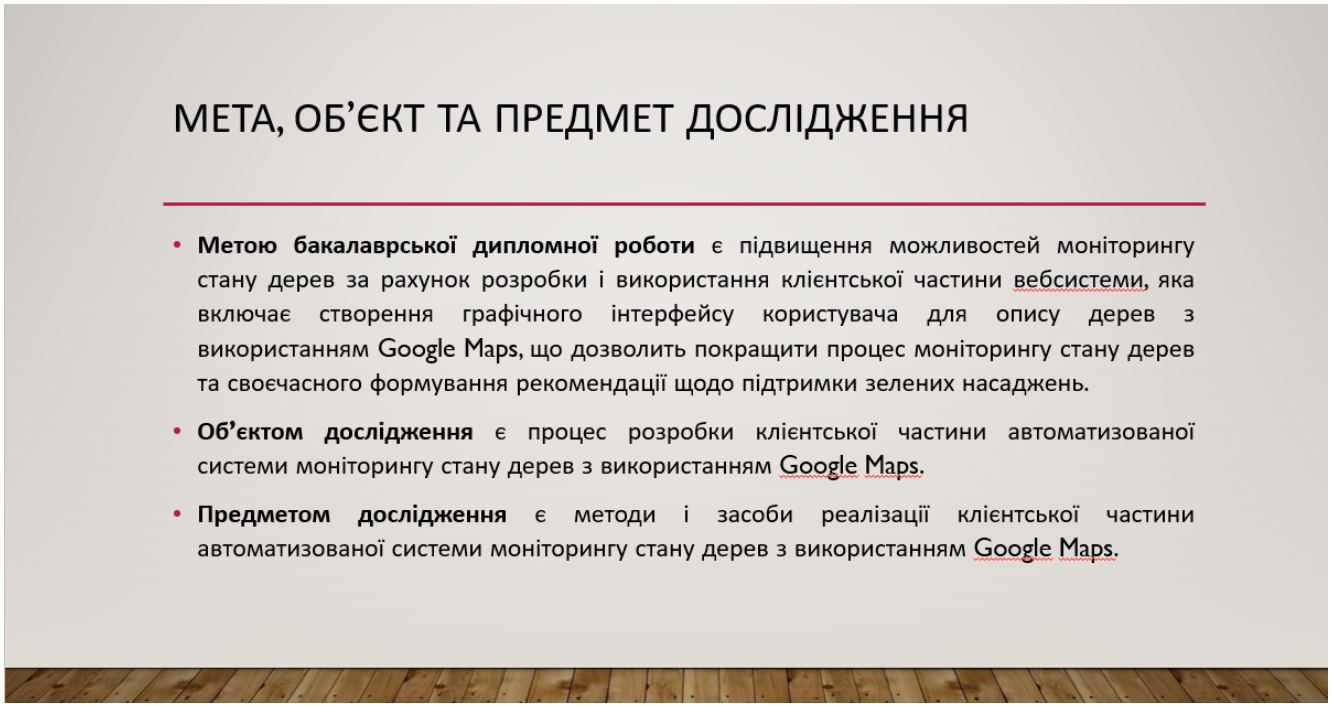

Рисунок Д.3 – Мета, об'єкт та предмет дослідження

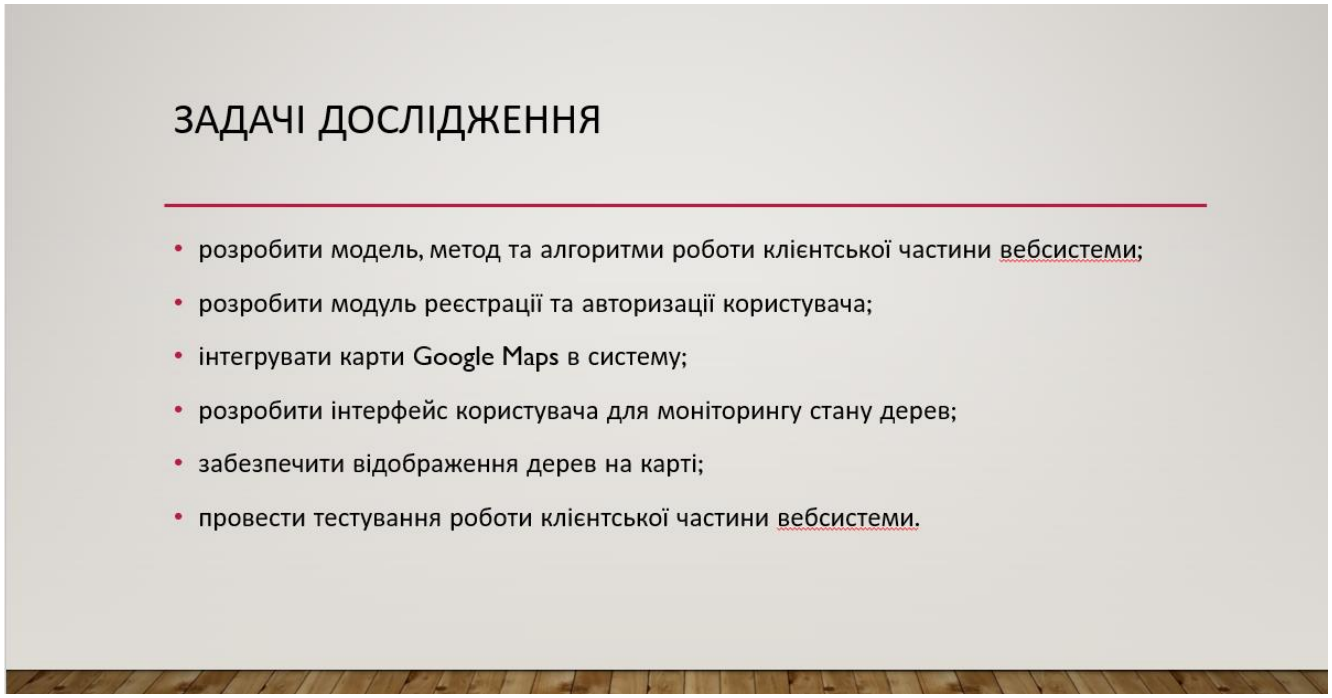

Рисунок Д.4 – Задачі дослідження

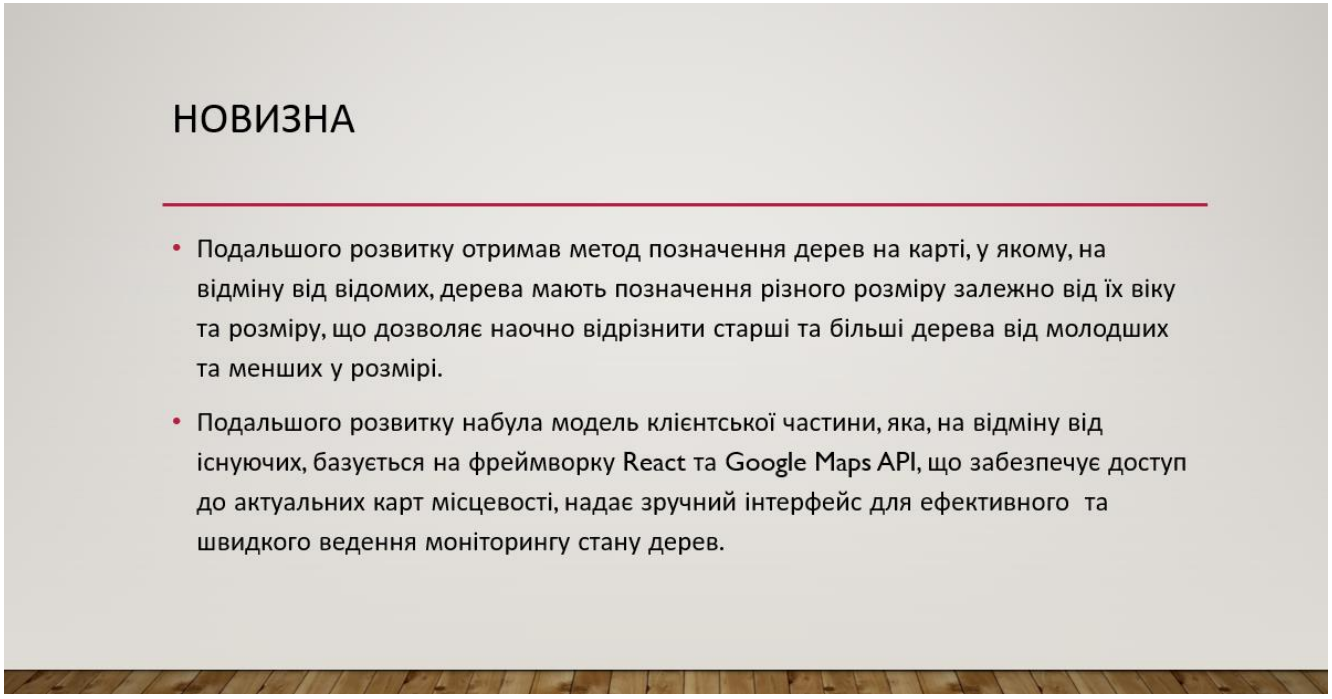

Рисунок Д.5 – Новизна

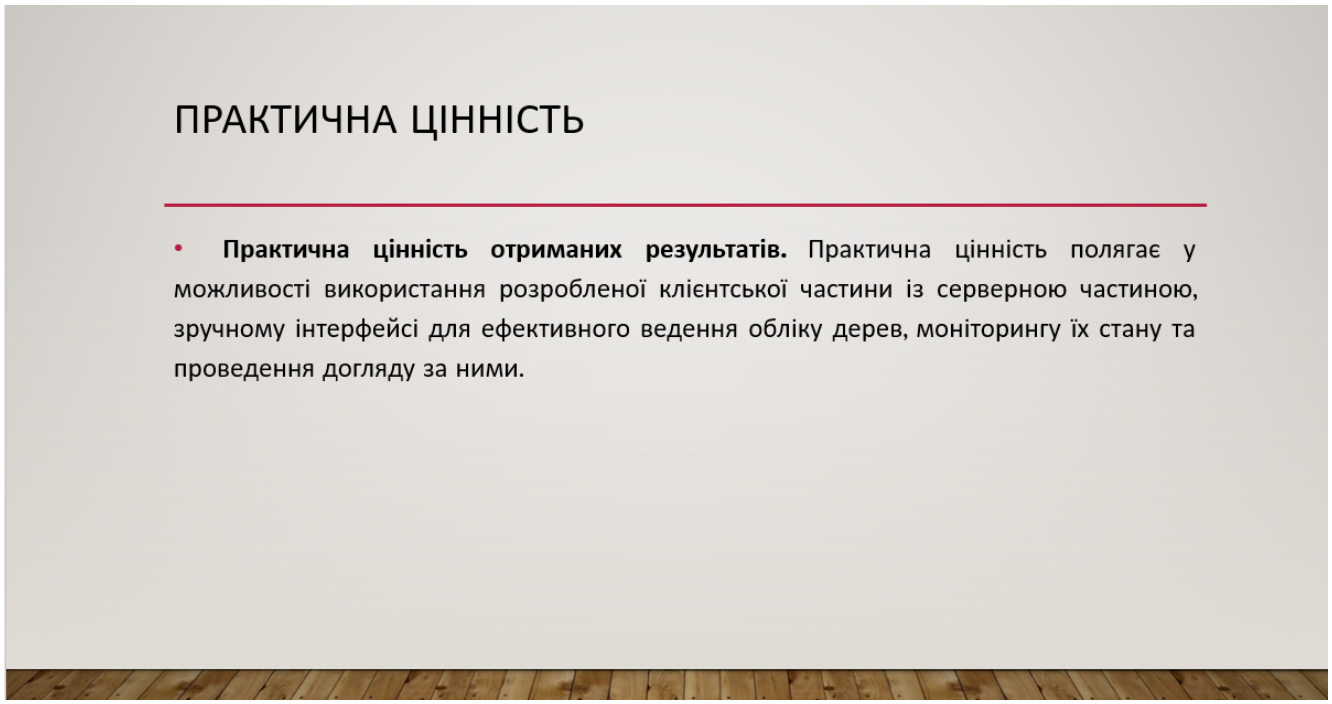

Рисунок Д.6 – Практична цінність

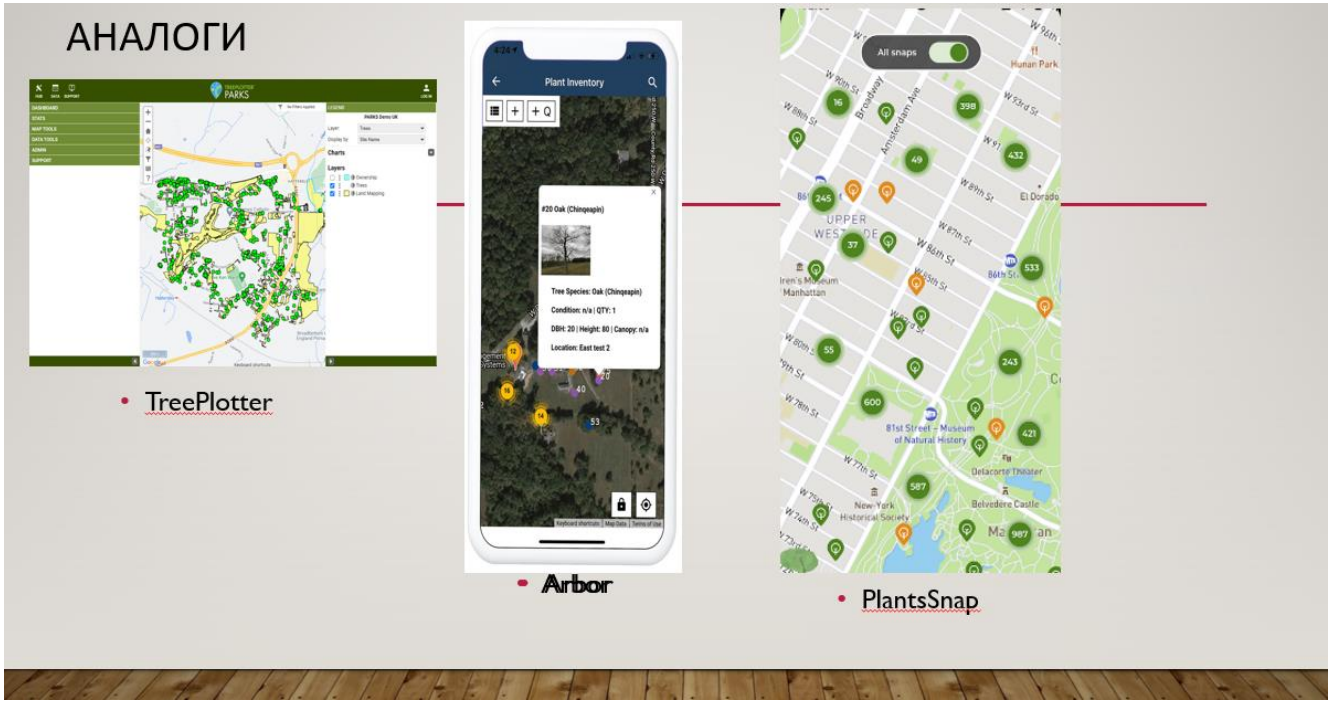

Рисунок Д.7 – Аналоги

Характеристика	TreePlotter	Arbor	PlantsSnap	Власна розробка
Картографування дерев	I	I	I	I
Введення характеристик дерев	I	I	I	I
Відображення дерев в реальному масштабі	0	0	0	I
Відстеження динаміки змін	I	I	I	I
Постановка завдань по догляду	0	0	0	I
Загальна оцінка	3	3	3	5

Рисунок Д.8 – Аналоги

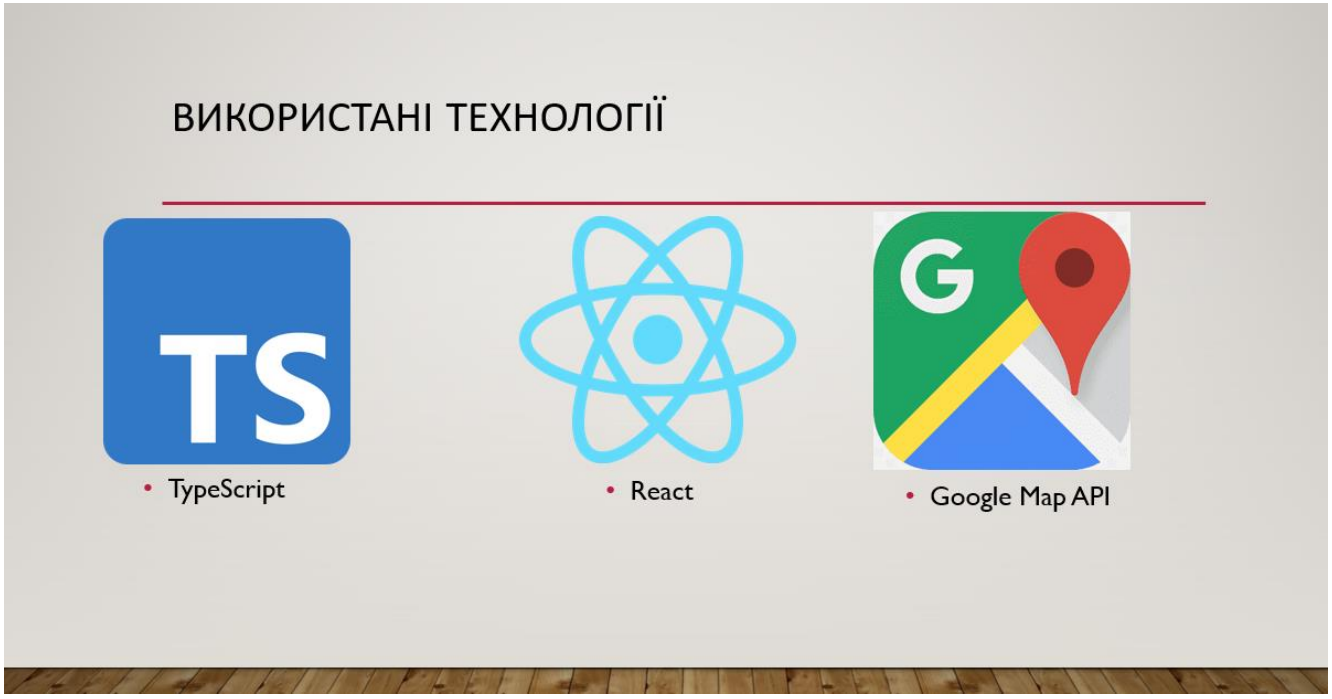

Рисунок Д.9 – Використані технології

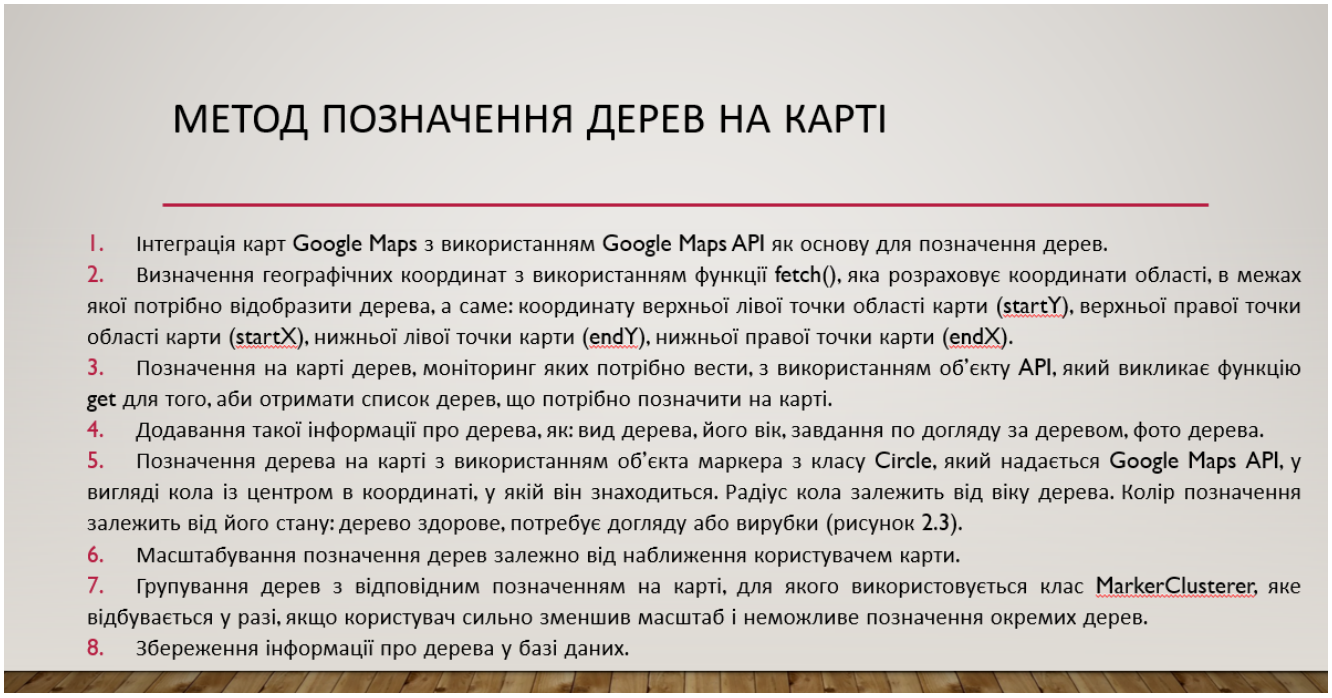

Рисунок Д.10 – Метод позначення дерев на карті

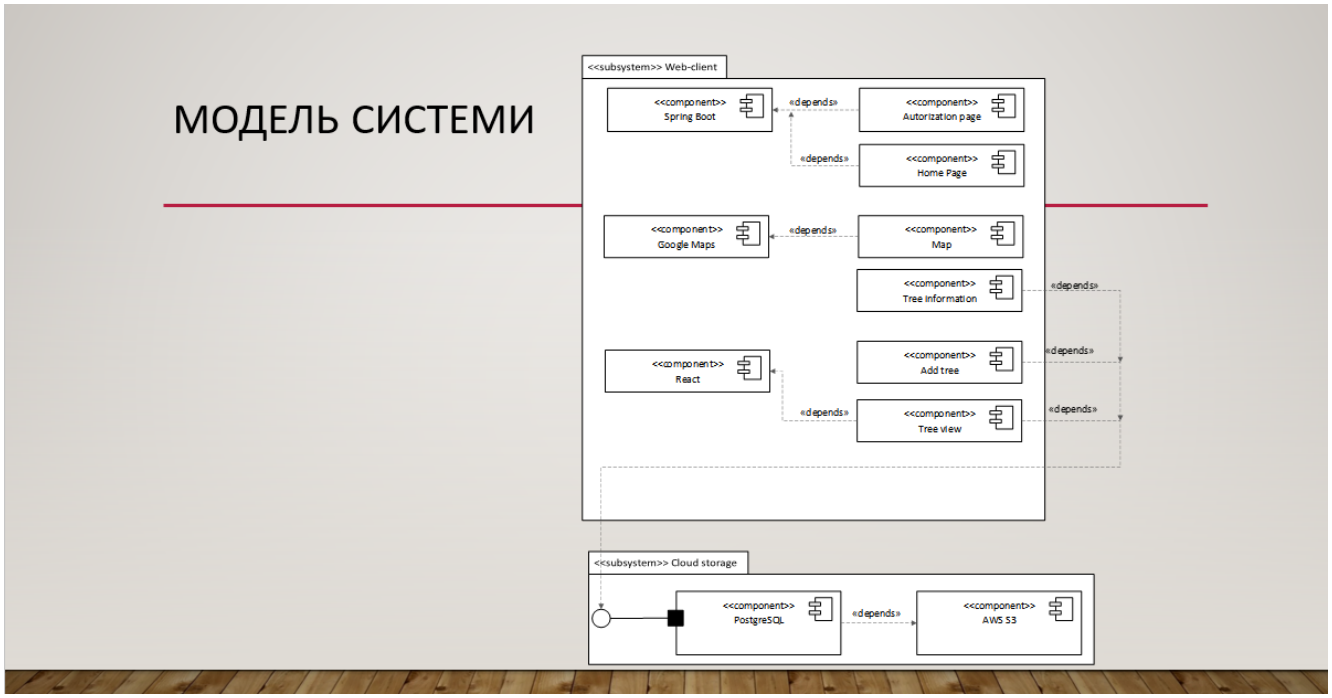

Рисунок Д.11 – Модель системи

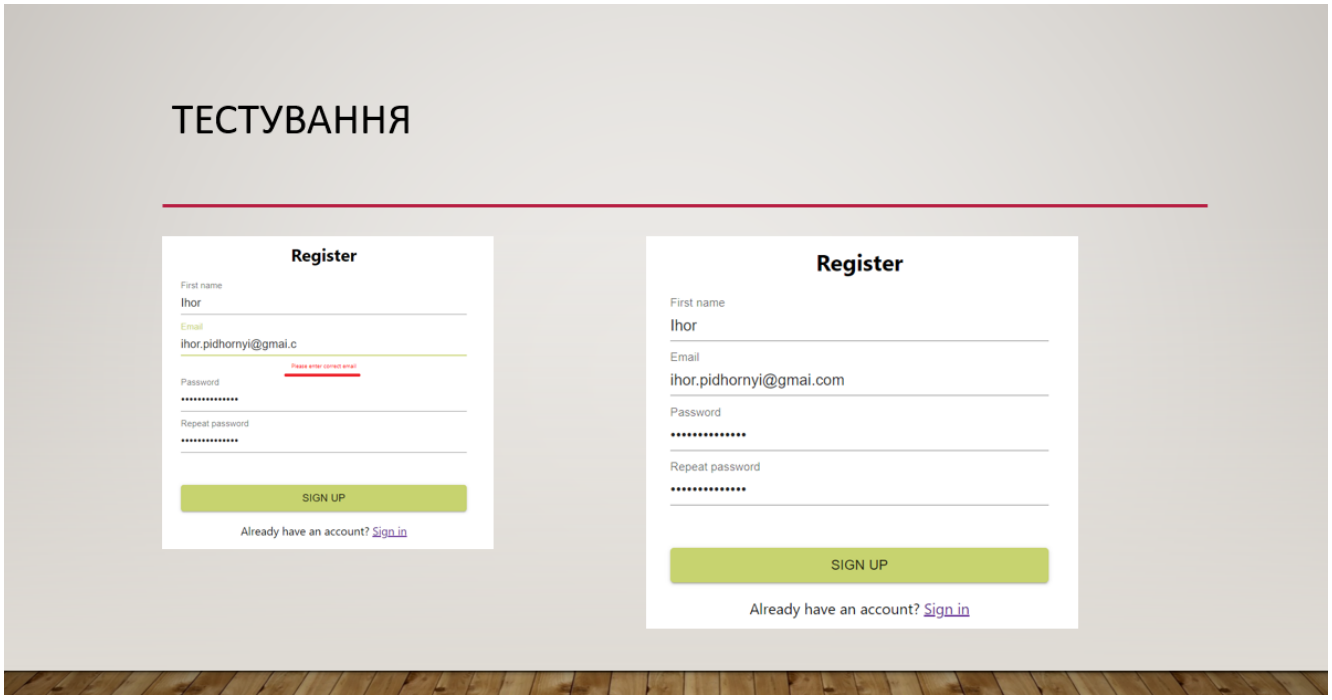

Рисунок Д.12 – Тестування реєстрації

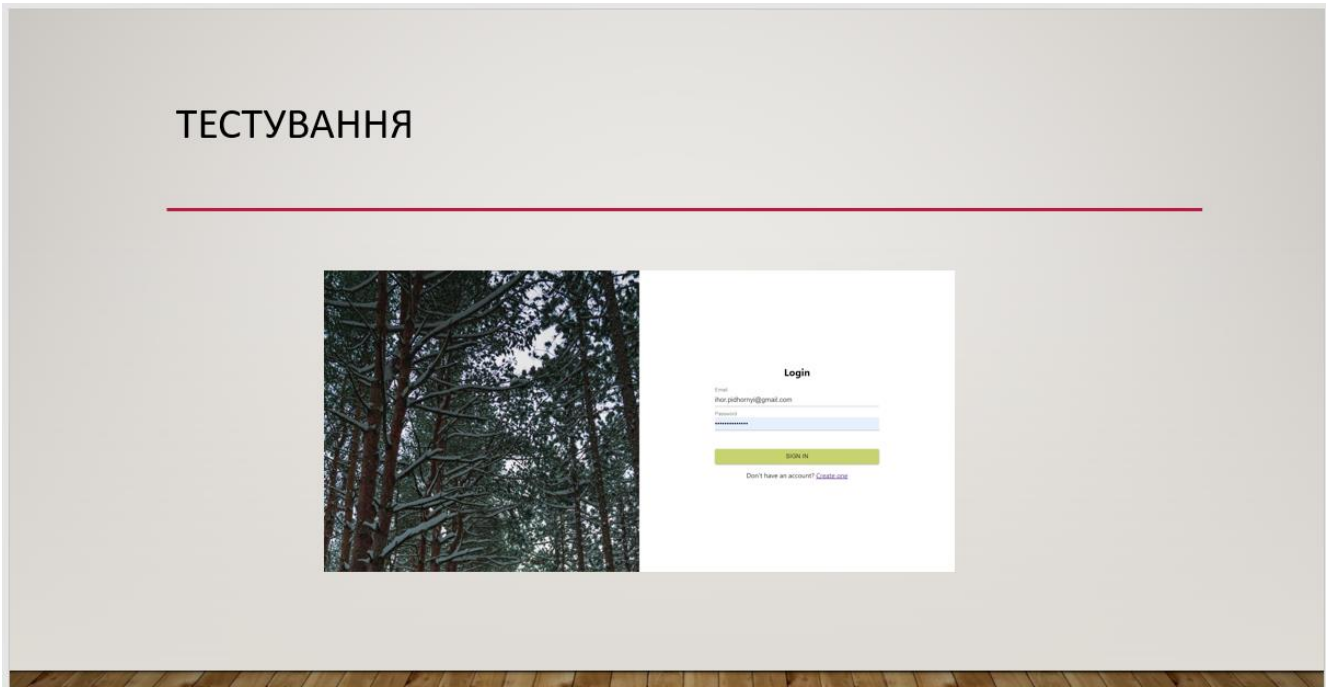

Рисунок Д.13 – Тестування авторизації

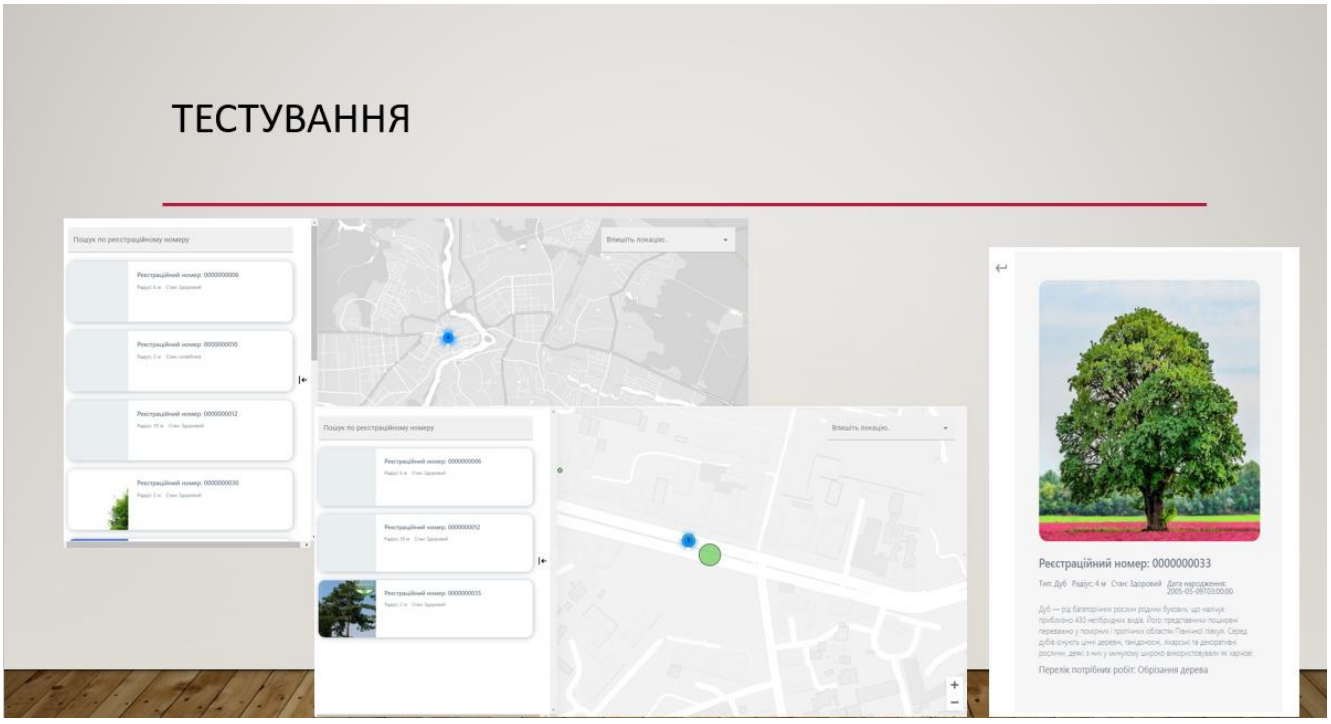

Рисунок Д.14 – Тестування пошуку дерев

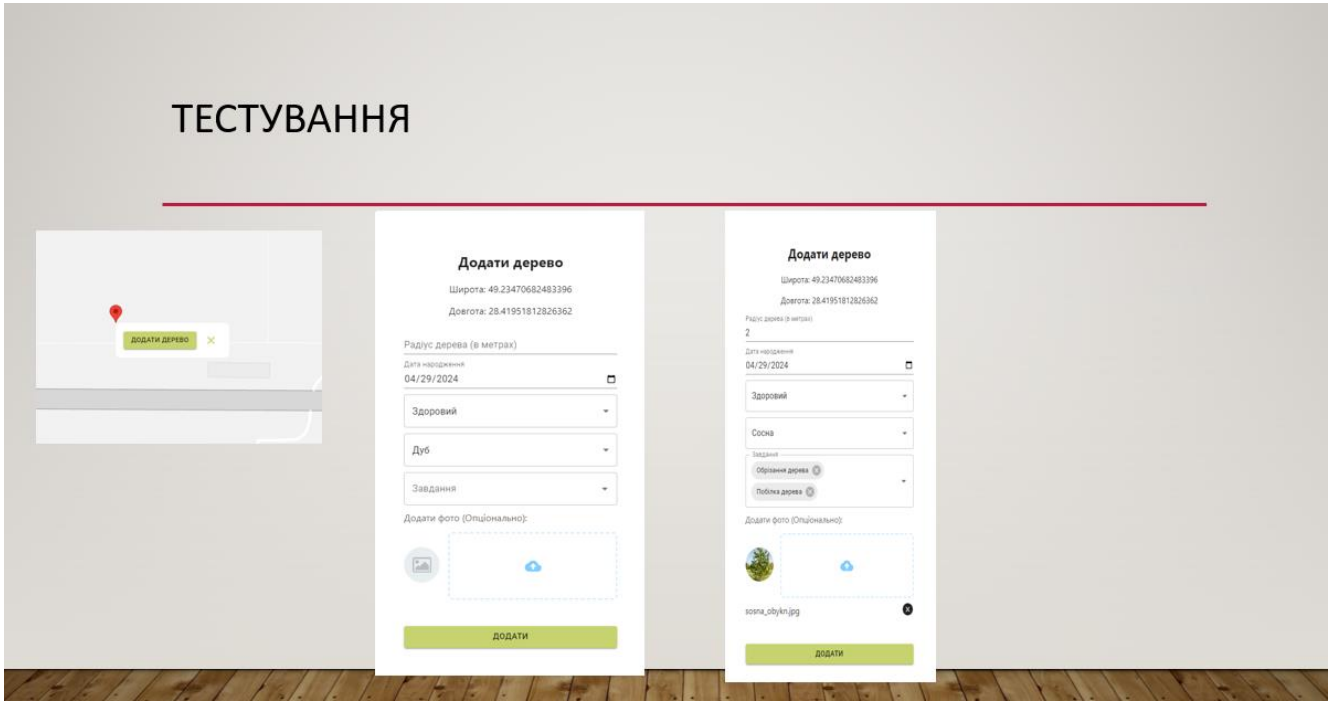

Рисунок Д.15 – Тестування додавання дерев

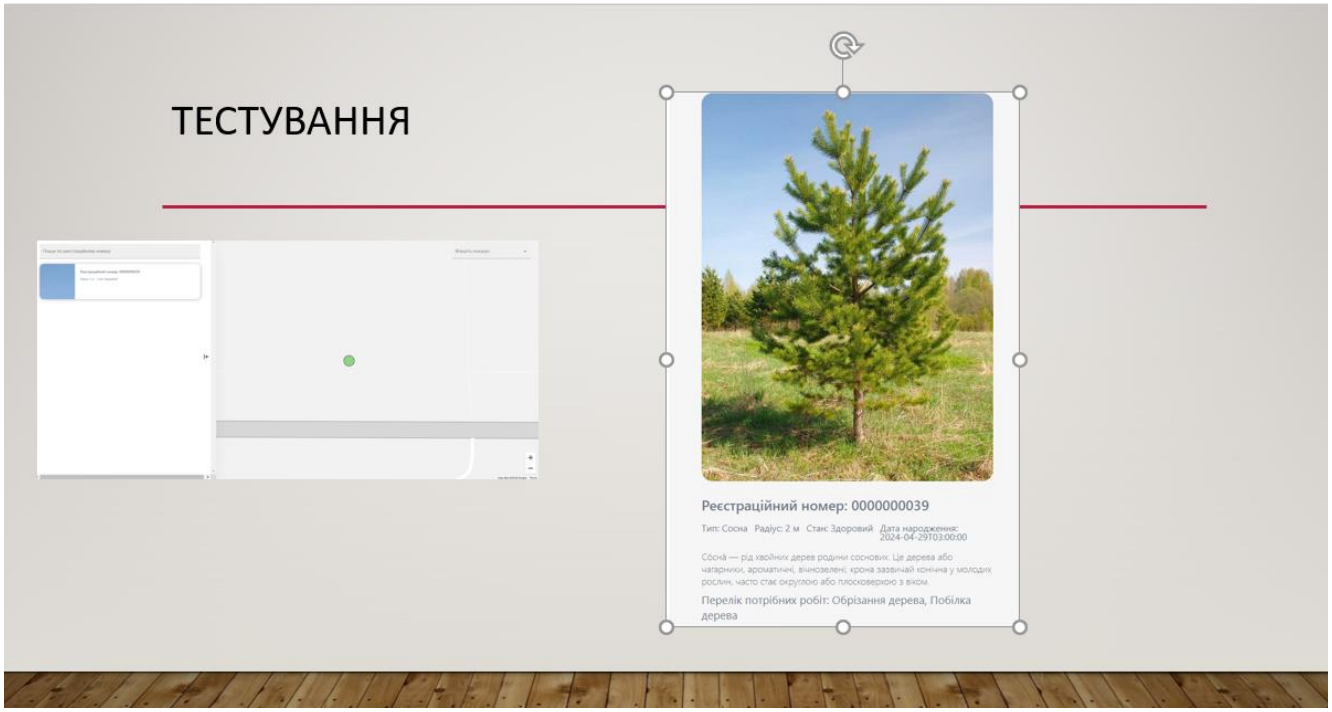

Рисунок Д.16 – Тестування перегляду детальної інформації про дерева

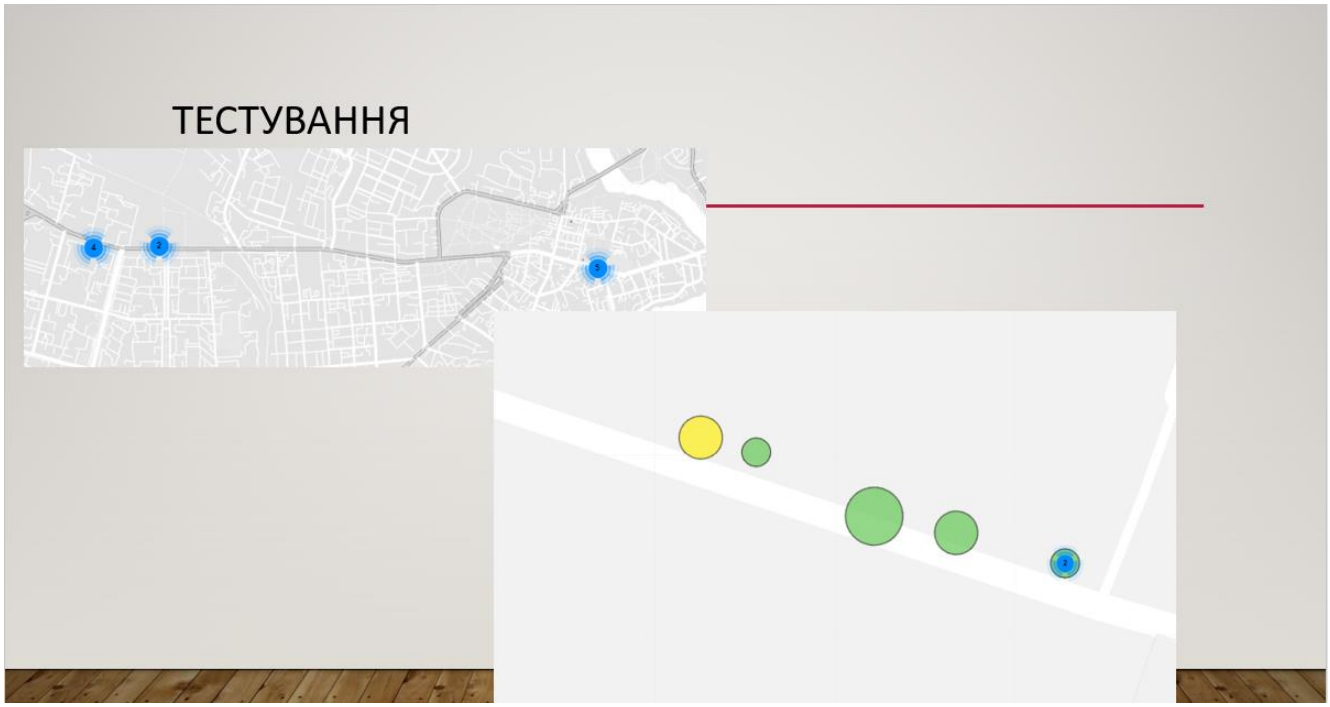

Рисунок Д.17 – Тестування відображення дерев на карті

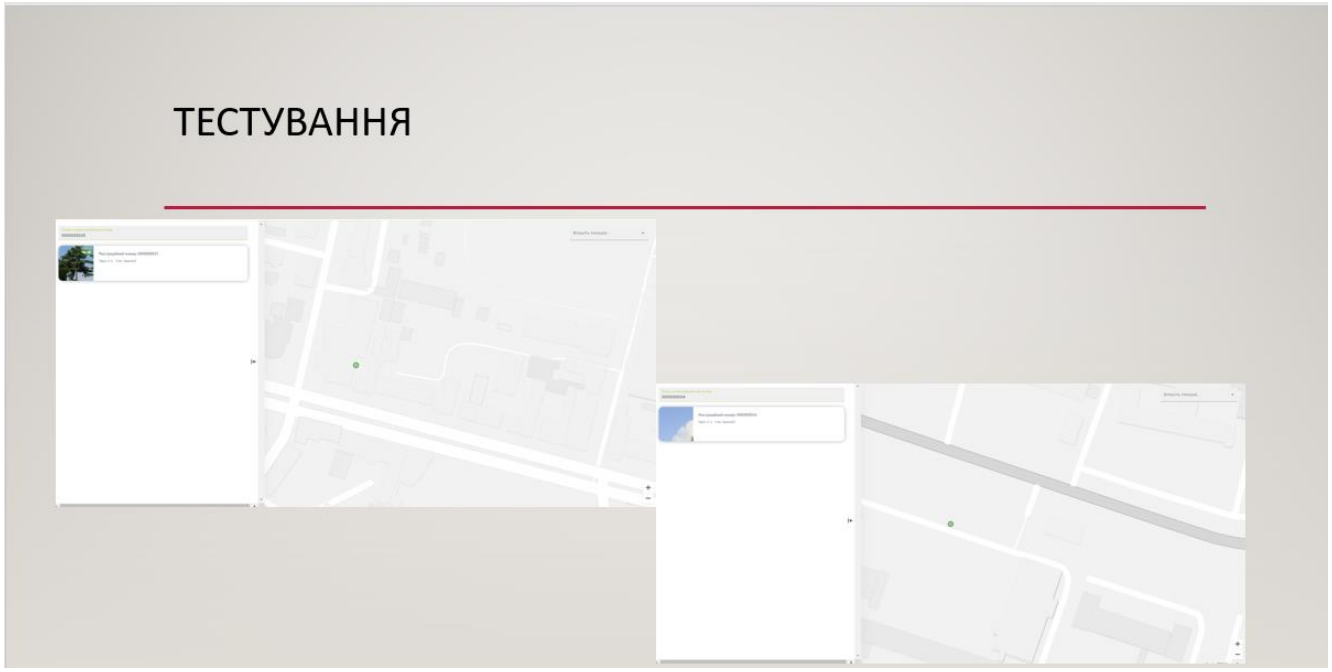

Рисунок Д.18 – Тестування пошуку дерев за номером

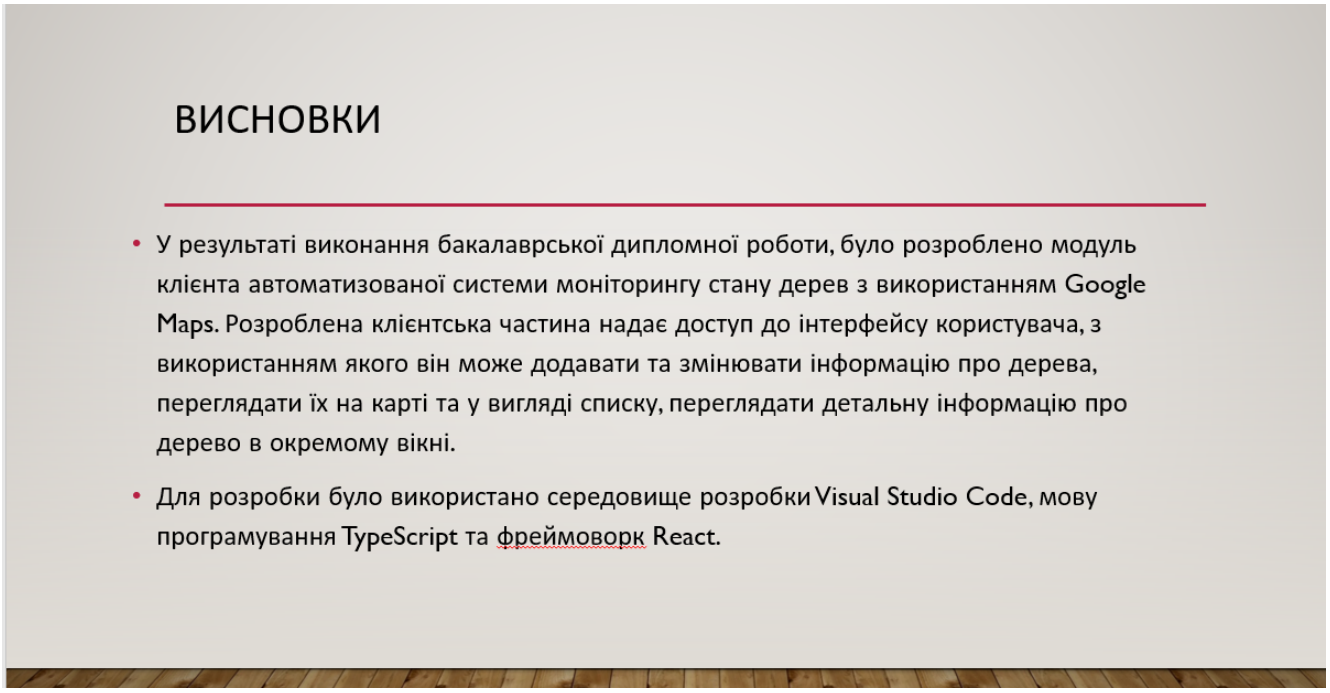

Рисунок Д.19 – Висновки

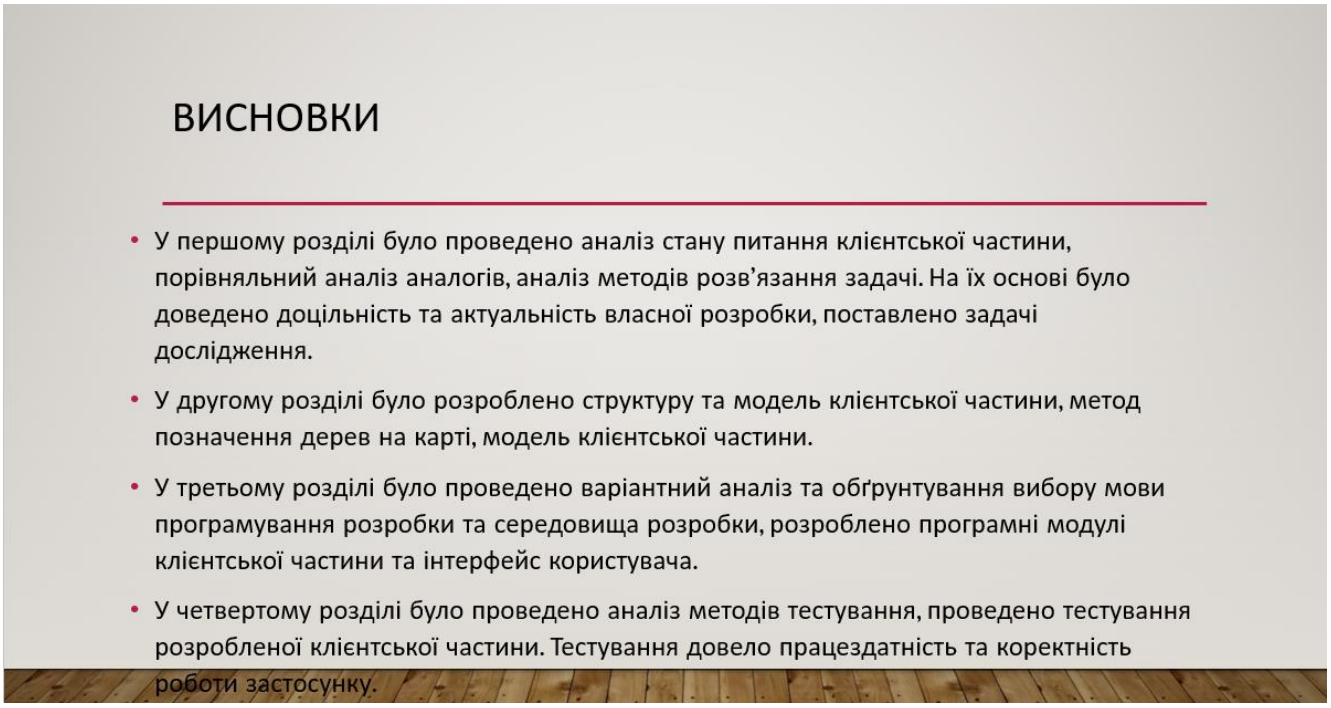

Рисунок Д.20 – Висновки (продовження)

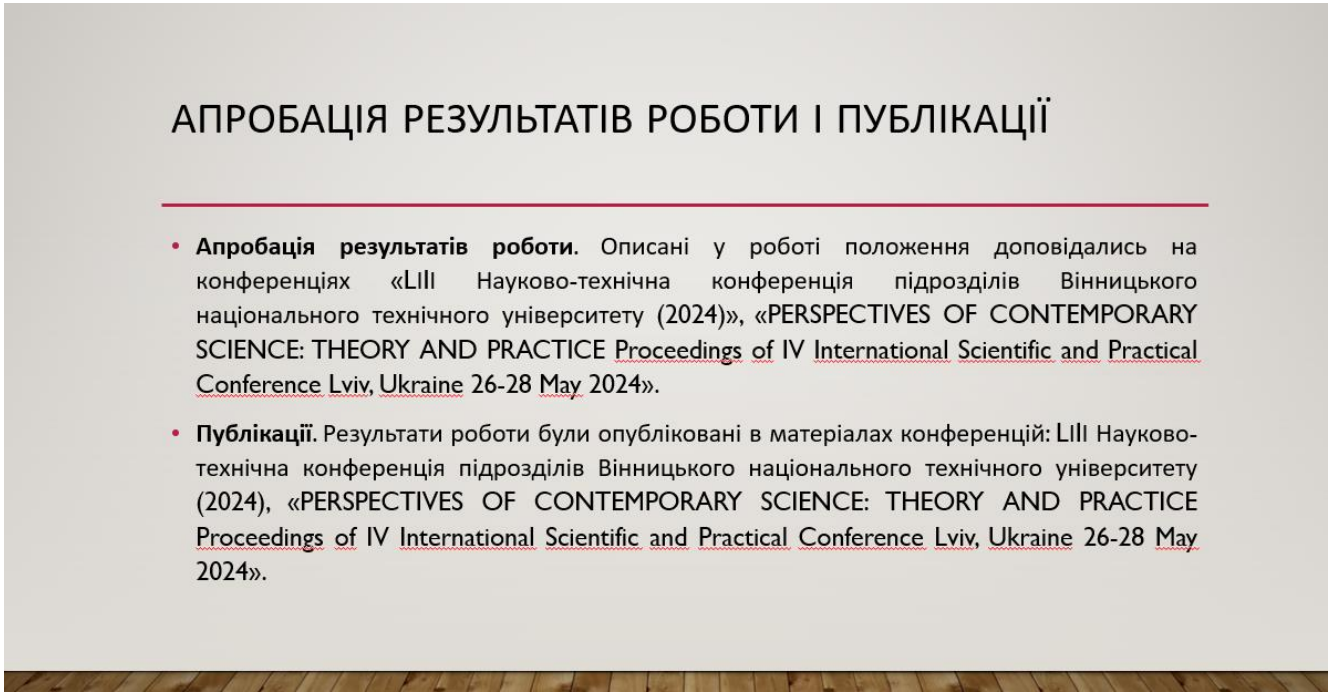

Рисунок Д.21 – Апробація результатів роботи і публікації