Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Левченко Дмитра Максимовича*
	(ПІБ)
академічної групи	*121М-22-2*
	(шифр)
спеціальності	*121 Інженерія програмного забезпечення*
	(код і назва спеціальності)
освітньої програми	*«Інженерія програмного забезпечення»*
	(назва освітньої програми)
на тему:	*Розробка та дослідження веб-інтерфейсу системи моніторингу грунтокліматичних параметрів агротехнічних об'єктів*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	*проф. Лактіонов І.С.*			

Рецензент				

Нормоконтролер	*проф. Лактіонов І. С.*			

Дніпро
2023

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)
_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)
« » _____ 20 23 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121М-22-2* *Левченко Дмитру Максимовичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка та дослідження веб-інтерфейсу системи моніторингу ґрунтокліматичних параметрів агротехнічних об'єктів*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 09.10.2023 р. № 1227-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес моніторингу ґрунтокліматичних параметрів агротехнічних об'єктів за допомогою веб-інтерфейсу.

Предмет досліджень – моделі, методи та інформаційні технології серверної та UI складових для забезпечення якості інструмента моніторингу ґрунтокліматичних параметрів агротехнічних об'єктів.

Мета НДР – підвищення ефективності, швидкості та складності відпрацювання сценаріїв для повного моніторингу ґрунтокліматичних параметрів за допомогою веб-інтерфейсу.

Вихідні дані для проведення роботи – теоретичні та експериментальні

дослідження для вирішення задач проєктування інформаційної системи застосунку та підходи до вибору технологій реалізації.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Новизна запропонованих рішень полягає в розробці та застосуванні сучасних методів реалізації програмного забезпечення для удосконалення існуючих засобів дистанційного моніторингу агротехнічного призначення за рахунок збільшення ступеня автоматизації та безпеки використання ПЗ з прямим доступом до бази даних.

Практична цінність результатів полягає у тому, що отримані в ході дослідження результати роботи можуть застосовуватися як і для переведення ручного моніторингу ґрунтокліматичних параметрів в більш автоматичний режим, так і у візуалізації стану наповнення бази даних для пришвидшення етапу моніторингу та аналізу ґрунтокліматичних параметрів через збільшення автоматизації процесу дослідження агротехнічних об’єктів.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень повинні бути подані у вигляді, який дозволяє побачити та оцінити безпосереднє використання технологій серверної та UI сторін розробки. В результаті роботи повинен бути розроблений веб-інтерфейс системи моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	Строки виконання робіт (початок – кінець
Аналіз теми та постановка задачі	12.09.2023-30.09.2023
Збір, дослідження та систематизація інформації щодо проектування та реалізації безпечних функціональних застосунків на Java	01.10.2023-31.10.2023
Розробка і тестування автоматизованої системи для вирішення задачі забезпечення автоматизації управління об'єктами для тестування	01.11.2023-10.12.2023

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним завдяки адаптивності даної системи до процесу аналізу та зберігання інформації продукту будь-якої установи, таким чином зменшити затрати на заробітну плату людям, які виконують мануальне моніторинг.

Соціальний ефект від реалізації результатів роботи очікується позитивним завдяки можливості вивчення та використання розробленої моделі ІС для вирішення задач проектування застосунків управління ресурсами аналізу та моніторингу в корпоративних системах. Створена на нових технологіях архітектура інформаційної системи, що має стабільну та перевірену систему авторизації та безпеки, є ефективною та дозволить легко масштабувати та інтегрувати новий функціонал.

Завдання видав _____ *Лактіонов І.С.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Левченко Д.М.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *12.09.2023 р.*

Термін подання кваліфікаційної роботи до ЕК *12.12.2023*

РЕФЕРАТ

Пояснювальна записка: 97 с., 36 рис., 3 дод., 25 джерел.

Об'єкт розробки: процес моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів за допомогою Google FireBase та Angular.

Предмет дослідження: методи, схеми та засоби автоматизації моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів.

Мета кваліфікаційної роботи: підвищення ефективності та швидкості під час повного тракту трансформації даних моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів.

Методи дослідження. Для виконання поставлених завдань були використані методи: об'єктно-орієнтоване програмування, криптографічні методи захисту інформації, система управління збереженням інформації у хмарній NoSQL базі даних, механізм гіпертекстової розмітки HTML5/SCSS, механізм структуризації та компонування веб-додатку Angular.

Наукова новизна полягає в збільшенні рівня автоматизації та безпеки методів і засобів для моніторингу стану агротехнічних об’єктів.

Практичне значення роботи. Отримані в ході дослідження результати роботи можуть застосовуватися як і для переведення ручного моніторингу інформаційної системи в більш автоматичний режим, так і у візуалізації стану наповнення бази даних для пришвидшення етапу перевірки агротехнічних об’єктів через збільшення автономності інженера з аналізу і моніторингу.

Список ключових слів: REST API, веб-застосунок, фреймворк, сервер, інтерфейс, http-запит, http-відповідь, інформаційна система, база даних, криптографія.

ABSTRACT

Explanatory note: 97 pages, 36 pictures, 3 appendices, 25 sources.

Object of research: the process of monitoring soil-climatic parameters of agrotechnical objects.

Subject of research: methods, schemes and means of automating the monitoring of soil and climatic parameters of agrotechnical objects.

Purpose of Master's thesis: increasing efficiency and speed for full monitoring of soil and climatic parameters of agrotechnical objects.

Research methods. To perform the tasks, the following methods were used: object-oriented programming, cryptographic methods of information protection, information storage management system in the cloud NoSQL database, HTML5/SCSS hypertext markup mechanism, Angular web application structuring and layout mechanism.

Originality of research consists in increasing the level of automation and safety of methods and means for monitoring the state of agrotechnical facilities.

Practical value of the results. The results obtained during the study can be used both to transfer manual monitoring of the information system to a more automatic mode, and to visualize the state of filling the database to speed up the stage of inspection of agrotechnical objects due to increasing the autonomy of the analysis and monitoring engineer.

Keywords: REST API, web application, framework, server, interface, http-request, http-response, information system, database, cryptography.

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

СУБД – система управління базою даних;

UI – user interface;

GUI – graphical user interface;

СУБД – система управління базами-даних;

SQL – structured query language;

ПК – персональний комп’ютер;

ОС – операційна система;

HTTP – hyper text transfer protocol;

API – application programming interface;

ПЗ – програмне забезпечення.

ЗМІСТ

ВСТУП

Актуальність дослідження. Сучасні технології дозволяють використовувати комп'ютери та інформаційні системи для вирішення різних завдань, у тому числі для моніторингу та управління агротехнічними об’єктами. Системи сенсорів, збір та аналіз даних, використання хмарних технологій дозволяють сільському господарству забезпечувати точне та ефективне виробництво. Інформаційні технології в агросекторі також включають в себе розробку мобільних додатків та платформ для фермерів, що полегшують процеси управління господарством та підвищують його продуктивність.

Використання сучасних інформаційних технологій для моніторингу ґрунтокліматичних параметрів стало необхідною складовою сучасного сільськогосподарського виробництва. Це дозволяє оптимізувати виробництво, зменшувати витрати та збільшувати врожайність, що є критично важливим для забезпечення продовольчої безпеки в умовах зростання населення та зміни клімату.

У кваліфікаційній роботі на основі аналізу останніх перевірених тенденцій розвитку технологій та існуючих концепцій, засобів та методів моніторингу агросполук запропоновано новий розширений метод проведення аналізу та збереження даних, а також створено необхідні засоби програмної підтримки. Розроблену програму відрізняє швидкий та інтуїтивно-зрозумілий інтерфейс що допомагає оброблювати та зберігати дані про ґрунтокліматичні параметри у захищеній хмарній СУБД.

Мета дослідження: підвищення ефективності та швидкості для повного моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів.

Завдання дослідження: Для досягнення поставленої мети в роботі були сформульовані та вирішені наступні завдання:

1. Викласти принципи процесу створення програмного продукту;
2. Дослідити особливості реалізації та захисту моніторингових програм;

3. Проаналізувати доступні інструменти підключення до сховища даних та захисту комунікації, які доступні в Google FireBase та Angular;
4. Спроектувати та розробити відповідне програмне забезпечення, яке буде здатним до розгортання на різних конфігураціях ОС;
5. Перевірити ефективність системи та зробити висновки щодо доцільності її створення.

Об’єкт дослідження: процес моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів за допомогою Google FireBase та Angular.

Предмет дослідження: методи, схеми та засоби автоматизації моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів.

Методи дослідження. Для виконання поставлених завдань були використані методи: об'єктно-орієнтоване програмування, криптографічні методи захисту інформації, система управління збереженням інформації у хмарній NoSQL базі даних, механізм гіпертекстової розмітки HTML5/SCSS, механізм структуризації та компонування веб-додатку Angular.

Наукова новизна полягає в збільшенні рівня автоматизації та безпеки методів і засобів для моніторингу стану агротехнічних об’єктів.

Практичне значення роботи. Отримані в ході дослідження результати роботи можуть застосовуватися як і для переведення ручного моніторингу інформаційної системи в більш автоматичний режим, так і у візуалізації стану наповнення бази даних для пришвидшення етапу перевірки агротехнічних об’єктів через збільшення автономності інженера з аналізу і моніторингу.

Результати дослідження можуть застосовуватися як в практиці розробки програмного забезпечення, так і поглиблення рівня самостійного опанування галузі.

Особистий внесок автора:

1. Наукові результати роботи отримані автором самостійно.

2. Вибір методів досліджень і технологій реалізації;

3. Розробка теоретичної частини роботи з дослідження і систематизування знань про існуючі підходи до аналізу та моніторингу даних а

також їх обробка та операції над ними, представлення за допомогою веб-інтерфейсу;

4.Реалізація засобів програмної підтримки для автоматизованого моніторингу даних та операцій над ними;

5. Оцінка отриманих результатів.

Структура і обсяг роботи. Робота складається з вступу, трьох розділів і висновків. Містить 97 сторінок, в тому числі 49 сторінок тексту основної частини з 36 рисунками, списку використаних джерел з 25 найменуваннями на 2 сторінках, 3 додатка на 35 сторінках.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальні відомості з предметної галузі

Дослідження та моніторинг ґрунтокліматичних параметрів агротехнічних об’єктів набувають особливої актуальності в сучасних умовах зростання світового населення та загострення проблем забезпечення продовольства. Використання веб-застосунків для збору та аналізу даних стає важливою ланкою в сучасних агротехнологіях, де інформаційні технології виявляються ключовими для підвищення ефективності сільськогосподарського виробництвах.

Одним із поширених методів дослідження ґрунтокліматичних параметрів є використання сучасних сенсорів, розташованих на агротехнічних об’єктах. Ці сенсори здатні вимірювати різноманітні параметри, такі як температура ґрунту, вологість, рівень освітлення та інші важливі характеристики. Зібрані дані потім передаються веб-застосункам для подальшого аналізу. Такий метод дозволяє отримувати точні та актуальні дані, що є критично важливим для прийняття інформованих рішень щодо сільськогосподарського виробництва [24].

Ще однією важливою складовою є використання веб-платформ для моніторингу та аналізу даних. Ці платформи надають можливість фермерам та агротехнікам в реальному часі спостерігати за змінами в ґрунтокліматичних параметрах своїх полів. Вони також можуть отримувати спеціалізовані рекомендації, що допомагають оптимізувати політику поливу, внесення добрив та інших агротехнічних аспектів. Інтеграція цих платформ з веб-застосунками дозволяє легко керувати та відслідковувати дані, забезпечуючи господарствам ефективний інструмент для прийняття рішень [25].

Застосування веб-технологій для моніторингу ґрунтокліматичних параметрів відкриває перед сільськими господарствами нові можливості. Зокрема, це дозволяє зменшити витрати на ресурси, такі як вода та добрива, завдяки більш точному та цільовому їх використанню. Крім того, це сприяє

підвищенню врожайності та якості сільськогосподарської продукції.

У наш час, коли виробництво харчових продуктів повинно бути ефективним та стійким, використання веб-застосунків для моніторингу ґрунтокліматичних параметрів стає стратегічно важливим елементом. Це не лише сприяє підвищенню продуктивності, але і забезпечує сталість сільськогосподарського виробництва в умовах непередбачуваних змін клімату та ринкових умов.

Якщо розглянути концепцію розробки, то ми побачимо, що для цього процесу нам знадобляться три мови програмування – HTML, CSS та JavaScript. Ці мови складають так звану «базу» програмування WEB-сторінок різних типів складності та різного призначення. Але не справа ще полягає в тому, що у нас час неможливо зробити конкурентоспроможний застосунок використовуючи лише ці три базові елементи. Для того, щоб сайт працював, нам потрібно щось більш точне, що більш детально давало би нам змогу втілення поставленої задачі, а також щось таке, що зробило би більш зручнішим та швидшим процес збірки окремих файлів та структур у велику цілісну систему, яка могла б автономно та незалежно функціонувати без постійної допомоги розробника. Саме для таких завдань існують такі речі, як Фреймворки, або ж англійською – Frameworks.

Framework являє собою каркас, платформу для створення веб-продуктів нового покоління, їх ефективної підтримки. Він призначений для складних, масштабних проектів, дозволяє реалізувати нестандартні рішення.

Він не формує для продукту, що розробляється жорсткі рамки. Він надає базові модулі, на основі яких створюється гнучкий сайт з широкими можливостями модернізації, розширення функціоналу за рахунок приєднання додаткових додатків в майбутньому.

Стандартна архітектура фреймворка ґрунтується на поділі трьох шарів [7]:

1. модель - відповідає за формування структури, правил бізнес-логіки;
2. уявлення - його основна функція – графічне відображення даних;
3. контролер - реалізує зв’язок з користувачем, перетворюючи

отриману від нього інформацію в команди для попередніх двох шарів.

Хочеться також зазначити, що крім цього інструменту, розробка веб-сайтів може виконуватися IT-фахівцем самостійно з написанням коду з нуля або з залученням готової CMS. Перший варіант занадто витратний за часом, вимагає маси зусиль в ході реалізації, подальшого тестування. Використання CMS дозволяє швидко створювати інтернет додатки, проте накладає на їх функціонування певні обмеження, тобто вийти за рамки «конструктора» не вийде. Такі способи розробки більше підходять для простих, статичних проектів.

Між ними фреймворк займає проміжну позицію. Робота з ним складніше, ніж з готовим движком, але результат буде отримано в рази швидше, ніж при написанні коду. Розробка сайту з його використанням отримала величезну популярність в останні роки, так як фреймворк має декілька значних переваг перед конструкторами:

- висока швидкість створення додатків, сервісів;
- можливість постійного розвитку продукту за рахунок інтеграції в структуру нових розширень;
- простий зручний супровід;
- максимальна безпека зберігання даних;
- застосування фреймворка для сайту забезпечує йому швидкість реагування, можливість витримувати значні навантаження.

Як і процес розробки, фреймворки поділяють на Back-end, Front-end та Full-stack. У рамках цієї дослідницької роботи буде розглянуто Front-end фреймворки та розробку на них.

Одразу треба зазначити, що Front-end фреймворки не пов’язані з логікою роботи програми, так як функціонують безпосередньо в браузері. Призначені для створення користувацьких інтерфейсів з різноманітною графікою, анімацією. Приклади: React, Vue, Angular.

Далі буде більш детально проведено аналіз аналогів у виборі Front-end фреймворків.

1) React

React – це відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків [10]. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

React дозволяє розробникам створювати великі веб-застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC.

Серед особливостей цього фреймворку можна зазначити:

- односторонню передачу даних;
- віртуальний DOM;
- компоненти React зазвичай написані на JSX;
- react надає змогу не лише для рендерингу HTML в браузері;
- використовує методи життєвого циклу.

2) Vue

Vue.js – це JavaScript-фреймворк що використовує шаблон MVVM для створення інтерфейсів користувача на основі моделей даних, через реактивне зв'язування даних [10].

Vue використовує синтаксис шаблонів на основі HTML, що дозволяє декларативно зв'язувати рендеринг DOM з основними екземплярами даних в Vue. Всі Vue шаблони валідні HTML, і можуть бути розпарсені браузерами та HTML парсерами. Всередині Vue компілює шаблони в рендерингові функції віртуального DOM. В поєднанні з реактивною системою, Vue здатний розумно обчислити кількість компонентів для ре-рендингу та застосувати мінімальну кількість маніпуляцій з DOM, коли стан застосунку зміниться.

У цього фреймворку ненав'язлива реактивна система. Моделі це просто плоскі JavaScript об'єкти. Це робить керування станами дуже простим та інтуїтивним. Vue надає оптимізований ре-рендеринг з коробки без потреби

робити що-небудь додатково. Кожен компонент слідкує за своїми реактивними залежностями під час рендерингу, тому система знає точно коли має відбуватись ре-рендеринг і які компоненти потрібно ре-рендерити.

Серед особливостей цього фреймворку можна зазначити:

- шаблони;
- реактивність;
- переходи. Vue надає різноманітні шляхи для застосування ефектів переходу, коли елемент додають, оновлюють або видаляють з DOM;
- роутинг.

3) Angular

Angular – це написаний на TypeScript front-end фреймворк з відкритим кодом, який розробляється під керівництвом Angular Team у компанії Google, а також спільнотою приватних розробників та корпорацій. Angular — це AngularJS, який був переосмислений та перероблений тією ж командою розробників [10].

Також варто відзначити, що Angular пропонує рендеринг на стороні сервера, який прискорює завантаження початкової сторінки і, отже, покращує SEO, спрощуючи сканування динамічних сторінок. Швидке відображення сторінок значно покращує сприйняття веб-застосунків для наступного покоління, написаних в рамках Angular. Також, за допомогою даного фреймворку можна легко тестувати код, легко створювати персоналізовані об’єктні моделі документа (DOM) та моделювати дані обмежено для використання невеликих моделей даних.

Серед особливостей цього фреймворку можна зазначити:

- angular підтримується Google;
- angular надає клієнтську MVC-інфраструктуру, яка допомагає у запуску та створенні динамічних додатків із сучасним рівнем якості;
- програми, написані на Angular, сумісні з різними браузерами. Angular автоматично обробляє код JavaScript, що підходить для кожного браузера;
- чистий і точний дизайн інтерфейсу користувача;

- зручна та проста маршрутизація;

- структура Angular полегшує розширення синтаксису HTML і легко створює повторні компоненти за директивами.

В якості фреймворку було обрано саме Angular. Також великою перевагою є використання мови TypeScript.

TypeScript – мова програмування, яка позиціонується як засіб розробки вебзастосунків, що розширює можливості JavaScript.

Переваги TypeScript над JavaScript:

- можливість явного визначення типів та статична типізація;

- підтримка використання повноцінних класів як в традиційних об'єктно-орієнтованих мовах;

- підтримка підключення модулів.

Ці нововведення мають підвищити швидкість розробки, прочитність, рефакторинг і повторне використання коду, здійснювати пошук помилок на етапі розробки та компіляції, а також швидкодію програм.

Для задання стилів будет використана мова SCSS.

SCSS – це скриптова метамова, яка інтерпретується в каскадні таблиці стилів (CSS). Спроектована Гемптоном Кетліном та розроблена Наталі Вейзенбаум. Scss призначений для підвищення рівня абстракції коду та спрощення файлів CSS.

1.2 Призначення розробки та галузь застосування

Веб-додаток розробляється з метою вдосконалення управління агротехнічними об'єктами та оптимізації процесів аналізу та моніторингу ґрунтокліматичних параметрів. Цей застосунок є відмінним інструментом для фермерів та агротехніків, які мають можливість ефективно керувати та вивчати дані про ґрунтові умови та кліматичні параметри для оптимального виробництва.

Для досягнення універсальності та гнучкості веб-додатка, була використана компонентна система, де різні сторінки та моделі функціонують

незалежно, але можуть взаємодіяти між собою при необхідності. Бек-енд сервер забезпечує універсальну базу даних, де зберігається весь необхідний інформаційний контент.

Архітектура програми сприяє уніфікації та легкості введення нових даних про ґрунтокліматичні параметри агротехнічних об'єктів. Використання методів строкової інтерполяції забезпечує гнучкість у представленні інформації про будь-який аспект агротехнічного об'єкту, що відображається чітко та системно у веб-додатку.

Додатково, в процесі розробки використовуються сучасні методи UI/UX дизайну для забезпечення інтуїтивно-зрозумілого інтерфейсу. Це робить використання програми приємним та продуктивним для кінцевих користувачів, надаючи їм доступ до зручних інструментів для аналізу та моніторингу ґрунтокліматичних умов на агротехнічних об'єктах.

1.3 Існуючі підходи до аналізу та моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів

Сучасний моніторинг ґрунтокліматичних параметрів агротехнічних об'єктів базується на використанні різноманітних інформаційних технологій, що сприяють точній збору та аналізу даних, оптимізації процесів та підвищенню ефективності сільськогосподарського виробництва.

Одним із прикладів використання інформаційних технологій є встановлення сучасних сенсорів на агротехнічних об'єктах. Ці сенсори можуть вимірювати різні параметри, такі як температура ґрунту, вологість, рівень освітлення та інші характеристики, в реальному часі. Зібрані дані передаються в хмарні системи або на сервери для подальшого аналізу та моніторингу. Такий підхід дозволяє фермерам отримувати негайну інформацію про умови на своїх полях та вчасно реагувати на будь-які зміни.

Інший ефективний метод використання інформаційних технологій полягає в застосуванні систем збору та аналізу даних, що інтегруються з

сучасними веб-додатками. Фермерам надається можливість в режимі онлайн спостерігати за динамікою ґрунтокліматичних параметрів, вносити необхідні корективи у сільськогосподарські процеси та приймати управлінські рішення на основі актуальної інформації.

Окрім того, використання сучасних технологій у сільському господарстві охоплює впровадження інтелектуальних систем аналізу даних. Методи штучного інтелекту та машинного навчання використовуються для прогнозування розвитку ґрунтових умов на основі зібраних даних. Це дозволяє агрономам визначати оптимальні стратегії управління господарством та підвищує точність прийняття рішень.

Рис. 1.1. Агротехнічний дрон-розпилювач

У сучасному світі також поширено використання дронів для моніторингу агротехнічних об'єктів. Дрони обладнані спеціалізованими сенсорами, які можуть вимірювати параметри ґрунту та відображати дані на великій площині. Це дозволяє отримувати зображення та інформацію високої роздільної здатності, що полегшує аналіз стану поля та призводить до швидшого виявлення проблем.

Усі ці технологічні рішення в сумі сприяють покращенню якості та продуктивності сільськогосподарського виробництва, а також відкривають нові можливості для точного моніторингу та управління ґрунтокліматичними параметрами агротехнічних об'єктів.

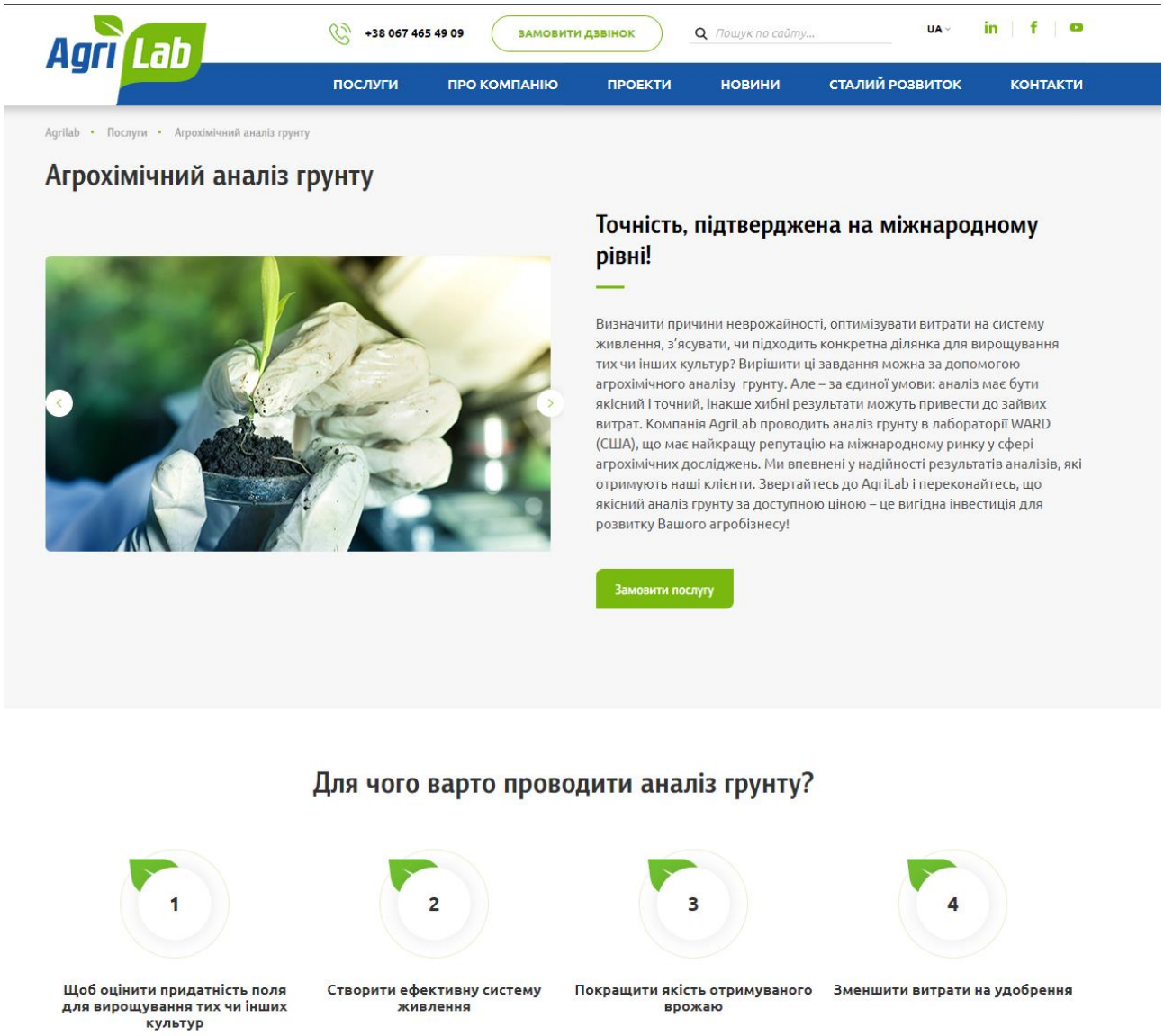

Рис. 1.2. Компанія, яка займається агрохімічним аналізом ґрунту

Далі буде наведено декілька прикладів існуючих систем. На рис. 1.2. ми бачимо сайт компанії під назвою «Agrilab» яка займається агрохімічним аналізом ґрунту.

При скролі вниз сайт пропонує нам обрати пакети обслуговування, це видно на рис. 1.3.

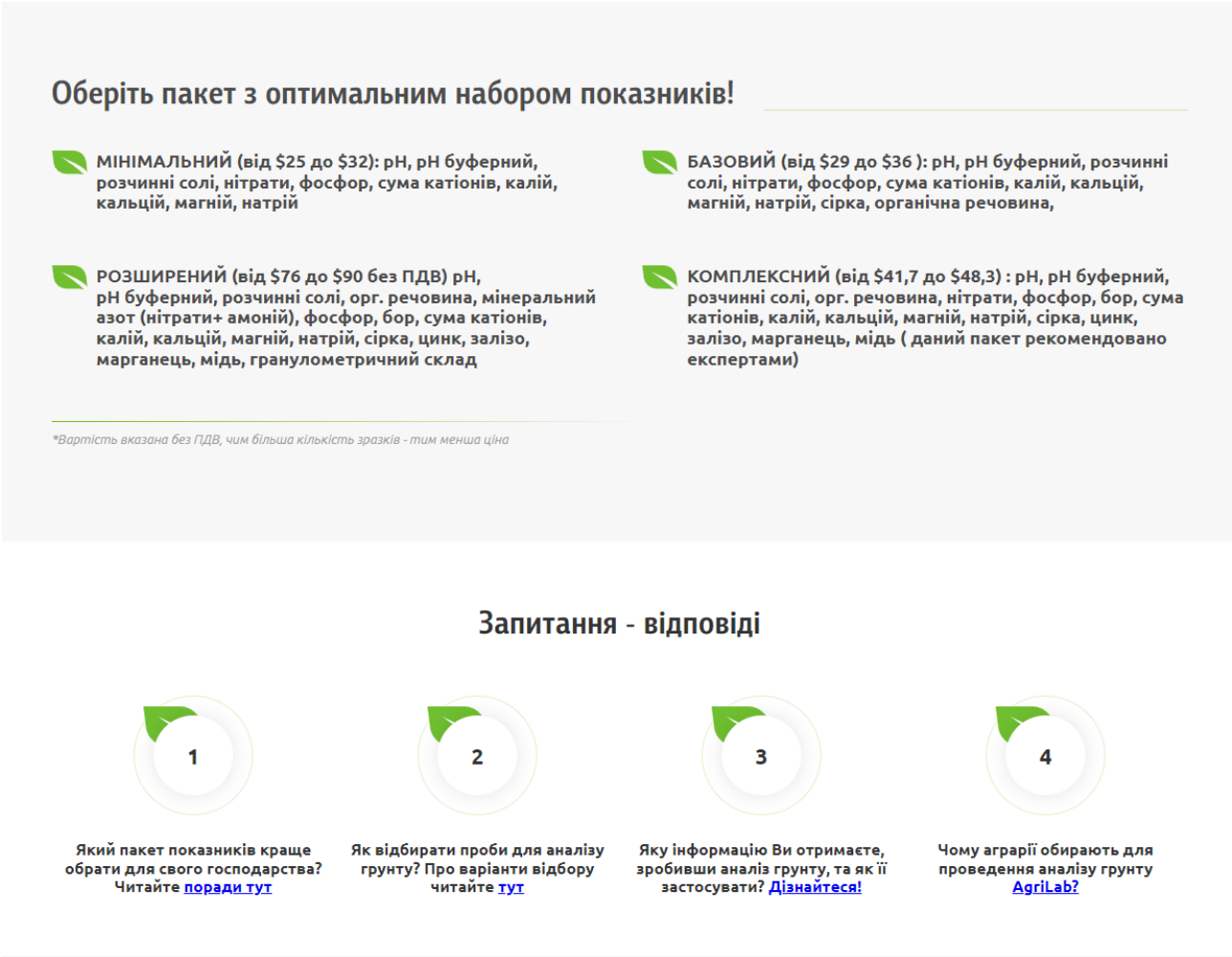

Рис. 1.3. Пакети аналізу

Далі нас зустрічає лаконічна форма, заповнивши яку можна залишити заявку на агрохімічний аналіз, це показано на рис. 1.4.

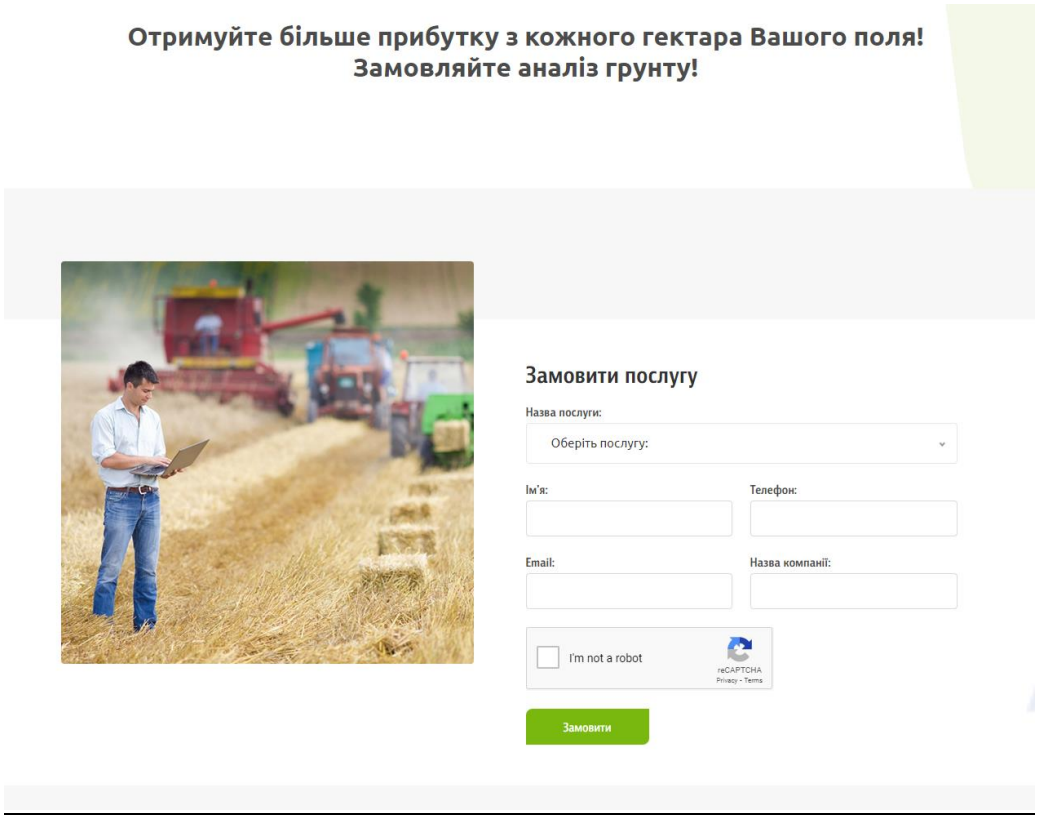

Рис. 1.4. Форма для заповнення запиту на аналіз

Далі ми бачимо портфоліо успішних кейсів цієї компанії з посиланням на її клієнтів, це видно на рис. 1.5.

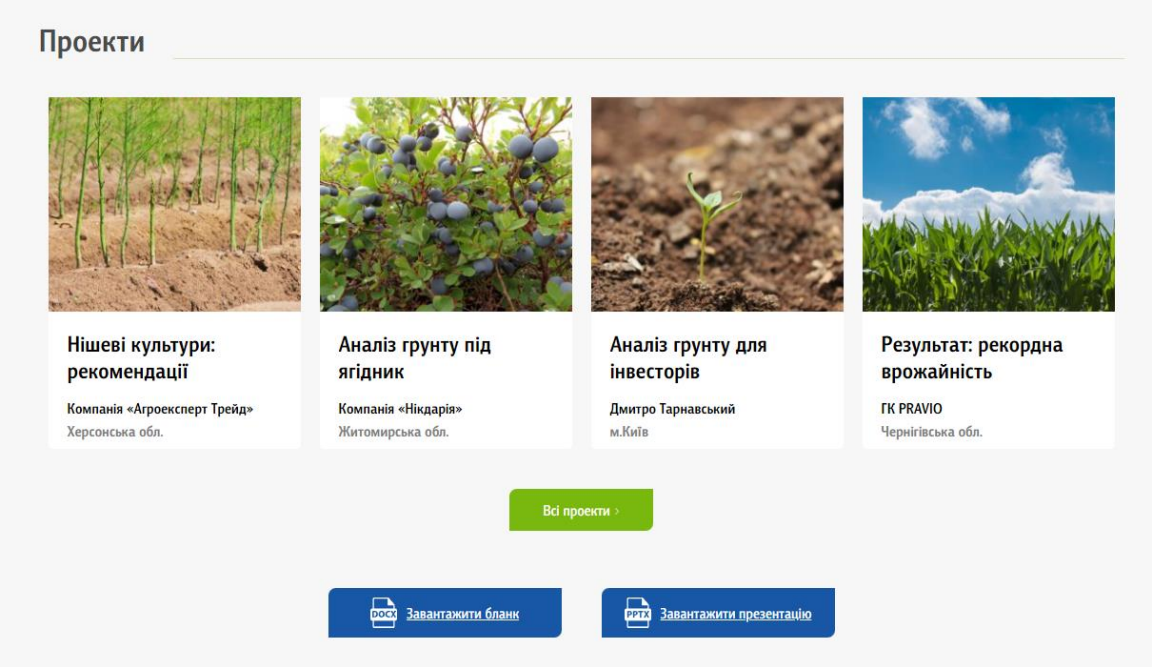

Рис. 1.5. Портфоліо успішних кейсів компанії

Fotter або ж нижня частина сайту є дуже важливою складовою, оскільки саме там повинна зберігатись уся важлива інформація про компанію та посилання для користувача для зворотнього зв’язку. Приклад організації цього компонента ми можемо бачити на рис. 1.6.

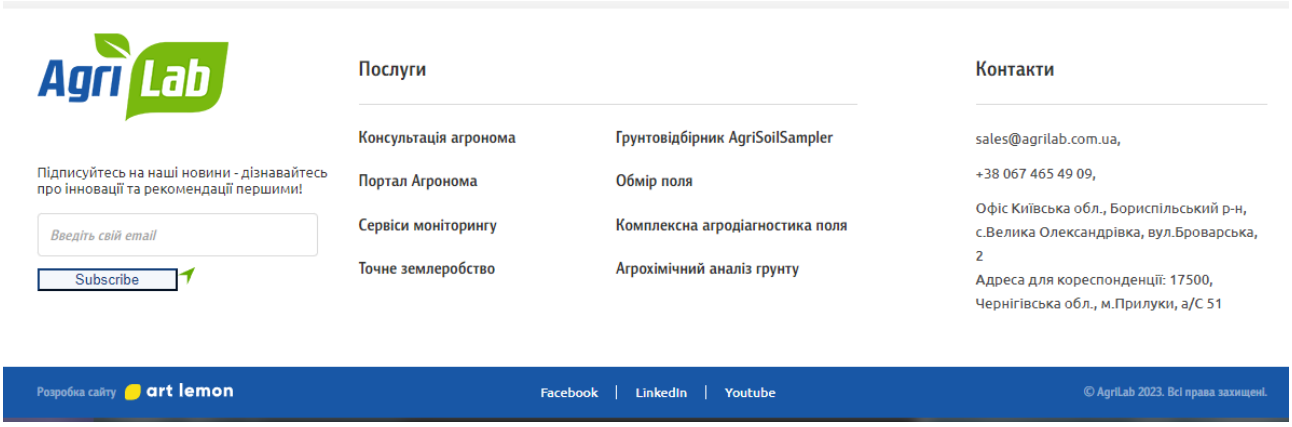

Рис. 1.6. Нижня частина сторінки сайту з корисною інформацією

Окрім простого аналізу ґрунту компанія пропонує також широкий спектр послуг, які можна побачити на рис. 1.7.

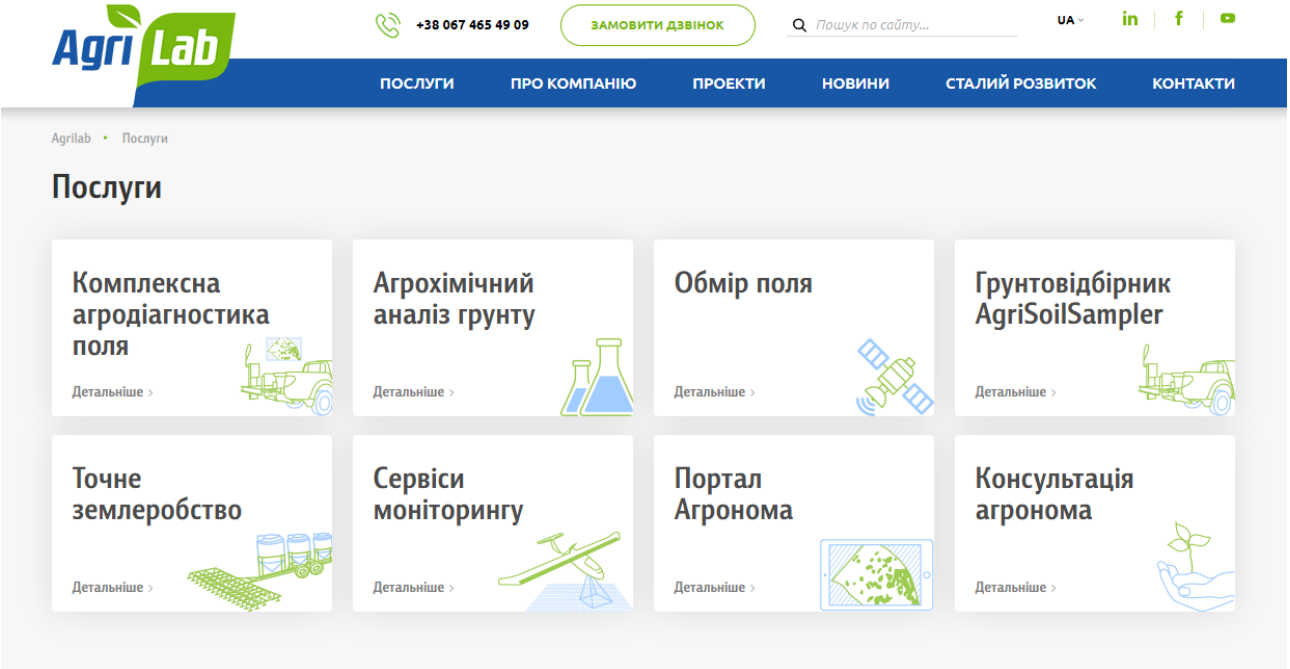

Рис. 1.7. Спектр послуг компанії

Також щоб показати, що компанія живе – на рис. 1.8. ми бачимо сторінку з новинами про життя компанії, її успішні останні кейси тощо.

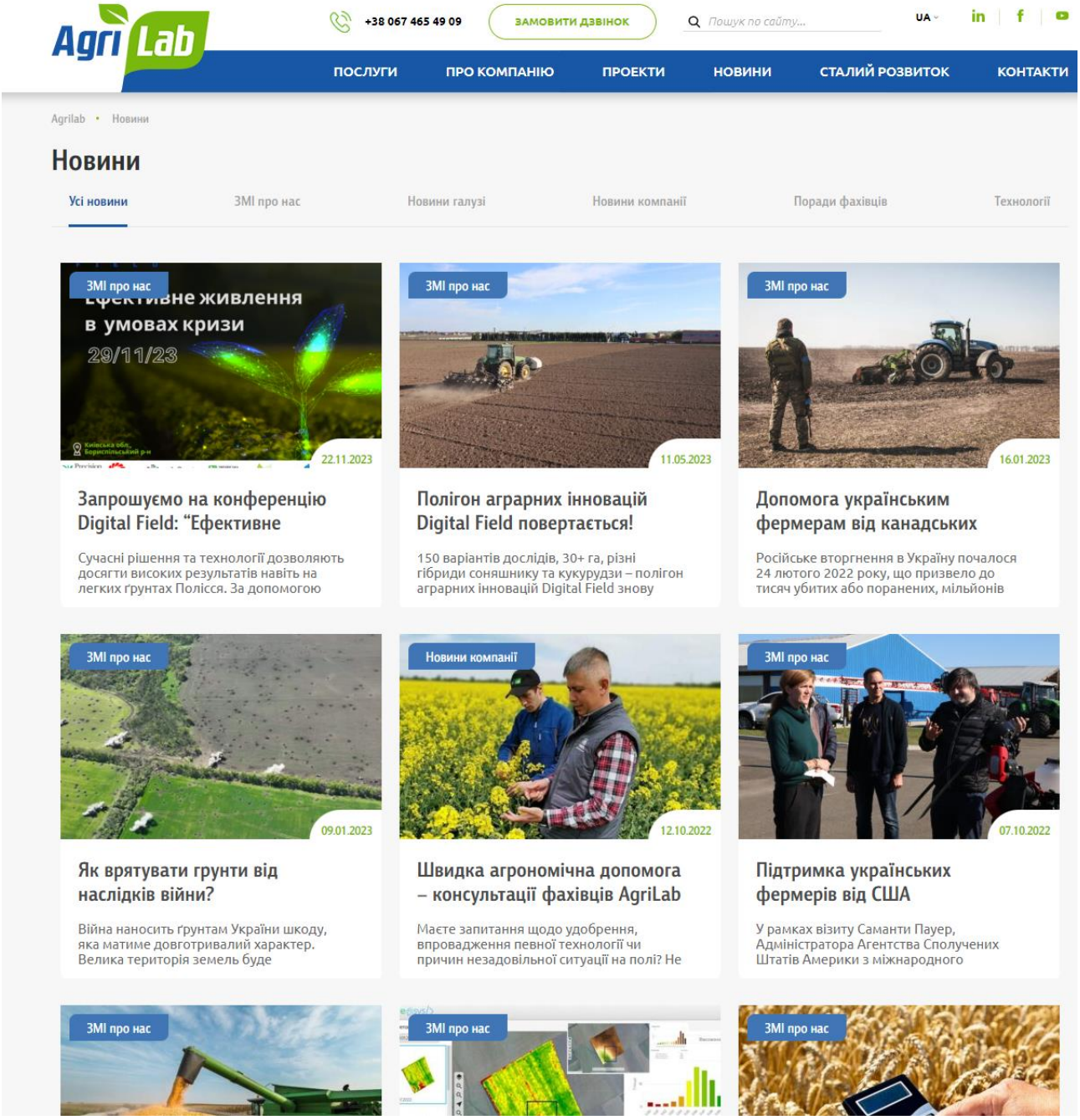

Рис. 1.8. Новини про життя компанії

В кожної перспективної компанії повинні бути офіси або місця куди можна приїхати для контакту із співробітниками та для замовлення послуг. Тому дуже важливим є зрозумілість розділу сайту, на якому вказано контаки фірми та мапа із позначками куди клієнт може під'їхати щоб отримати кваліфіковані послуги спеціалістів. Приклад такого розділу контактів позначений на рис. 1.9.

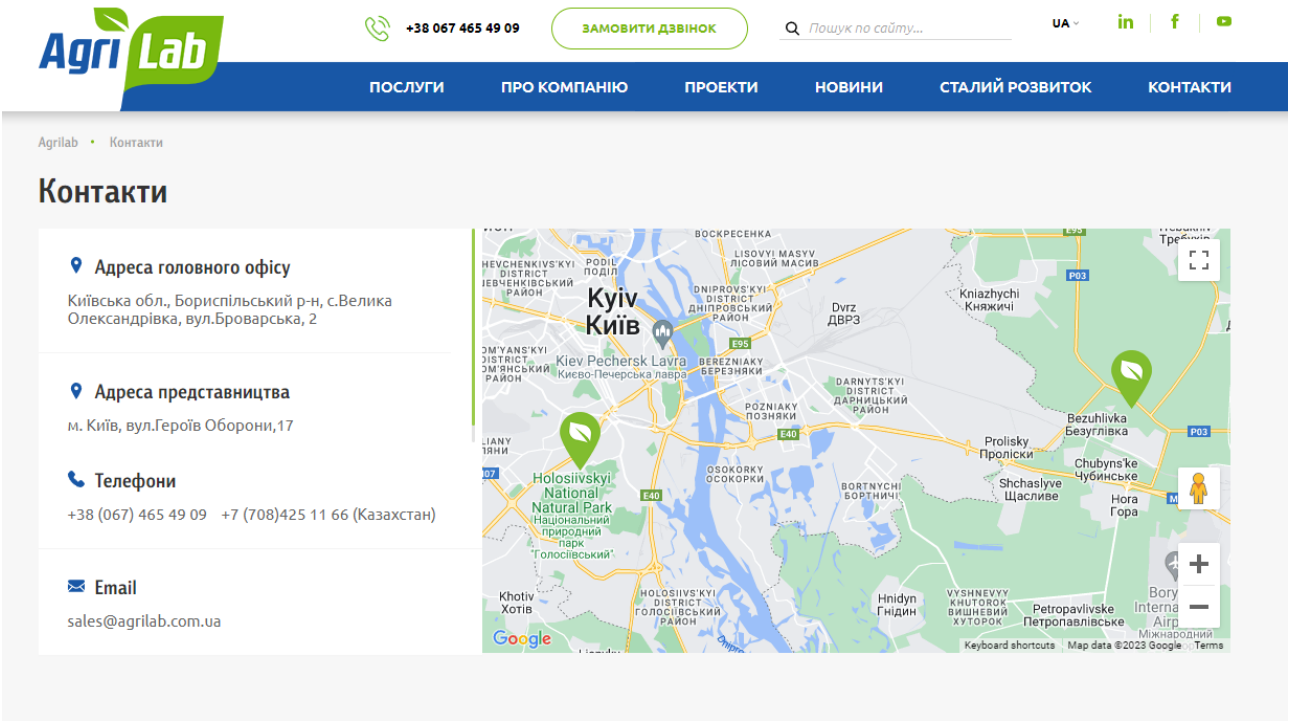

Рис. 1.9. Розділ із контактами компанії та мапою

Також у header компоненті сайту присутній пошук, номери телефонів для комунікації та посилання на соціальні мережі, це представлено на рис. 1.10.

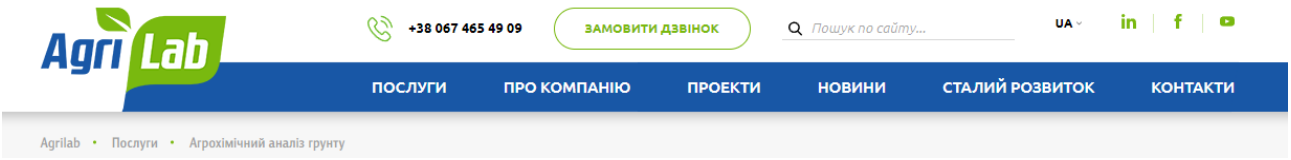

Рис. 1.10. Пошук на сайті та посилання на мережі

Для порівняння наведу приклад ще одної компанії, яка використовує архітектуру лендингів для свого сайту. Весь сайт компанії – це одна WEB-сторінка яку можна прокручувати униз та дивитись блоки з інформацією. На рис. 1.11 і 1.12 представлено загальний вид деяких проміжків лендингу.

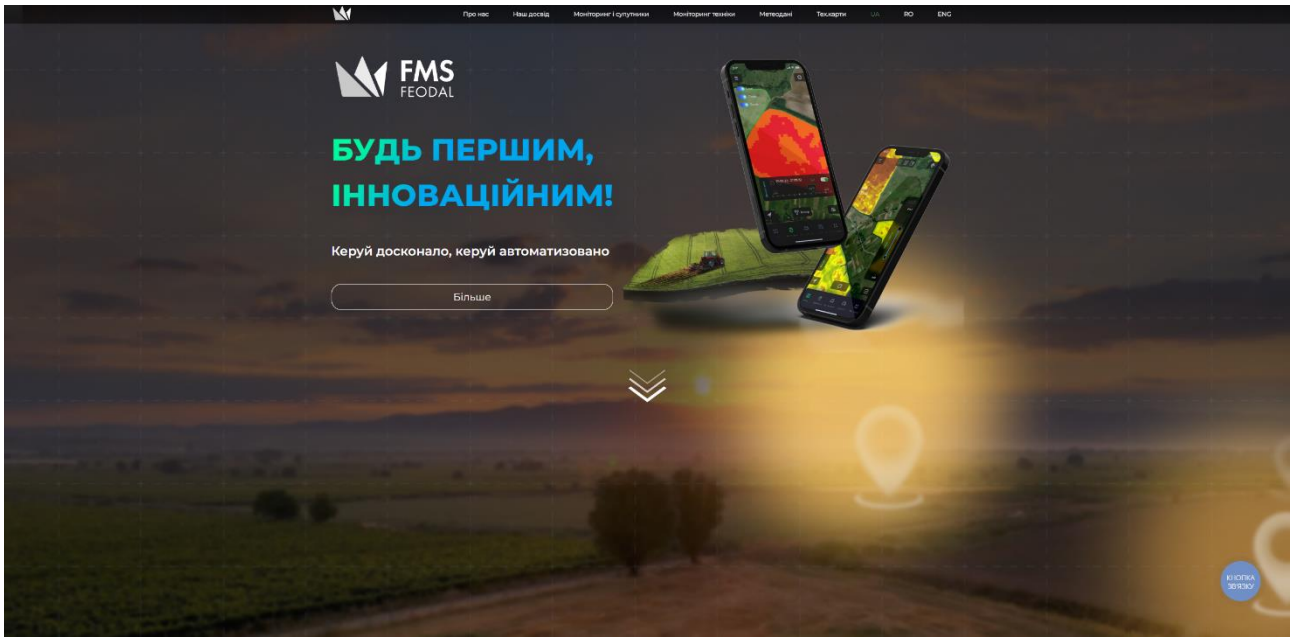

Рис. 1.11. Головний проміжок лендингу

Рис. 1.12. Типовий вигляд інших проміжків лендингу

Незважаючи на нетипічний спосіб верстки у сайта також присутній footer компонент, він представлений на рис. 1.13.

Рис. 1.13. Приклад Footer компонента у лендингу

1.4 Постановка мети і задач дослідження

Мета даного дослідження полягає в розробці front-end частини веб-додатку, спрямованого на моніторинг та аналіз ґрунтокліматичних параметрів агротехнічних об'єктів. Застосунок покликаний надавати функціонал сучасного інструменту для агрономів та фермерів, дозволяючи ефективно керувати та аналізувати дані щодо умов в ґрунті та клімату.

Для вирішення цього завдання обрано фреймворк Angular, що базується на TypeScript та HTML. Цей вибір обумовлений його здатністю виконувати необхідні функції та зручністю у компонентному підході до розробки.

Веб-додаток буде призначено для корпоративних користувачів. Вони матимуть доступ до адміністративної панелі, де вони зможуть переглядати та корегувати дані про агротехнічні сполуки, в особливості про ґрунтокліматичні показники. Користувачі зможуть спостерігати за параметрами ґрунту та клімату, використовуючи веб-інтерфейс, і вносити дані про свої об'єкти.

Головною структурою додатку буде сервер та клієнтська частина, що відповідає за веб-інтерфейс. Для адміністраторів це буде адміністративна панель для моніторингу та аналізу ґрунтокліматичних показників.

Вхідними даними для адміністраторів буде інформація про параметри ґрунту та клімату, яка подаватиметься через back-end сервер та зберігатиметься в базі даних. Користувачі отримають змогу користуватися функціоналом додатку для моніторингу та внесення своїх данних про об'єкти.

Отже, вихідним результатом дослідження та розробки буде повноцінний веб-застосунок, який надасть можливість фермерам та агрономам ефективно взаємодіяти з ґрунтокліматичними даними для підвищення якості та продуктивності сільськогосподарського виробництва.

1.5 Висновок

У сучасному світі інформаційні технології відіграють важливу роль у розвитку агросфери, зокрема у моніторингу та аналізі ґрунтокліматичних параметрів агротехнічних об'єктів. Розглянемо кілька аспектів, які ілюструють використання сучасних інформаційних технологій в цій сфері.

Початково, зазначено, що розробка веб-додатку для моніторингу ґрунтокліматичних параметрів агротехнічних об'єктів вимагає використання передових технологій для поліпшення підприємницької діяльності. Використання фреймворка Angular та компонентної архітектури дозволяє створити універсальний інструмент, який легко адаптується для різних типів даних. Застосування сучасних методів розробки UI/UX дизайну гарантує, що користувачі отримають інтуїтивно зрозумілий та зручний інтерфейс.

Подальше висвітлення дослідження та розробки веб-додатку для моніторингу ґрунтокліматичних параметрів агротехнічних об'єктів підкреслює необхідність точного та ефективного збору даних для сільськогосподарського виробництва. Використання сучасних сенсорів, інтеграція з хмарними системами та серверами дозволяють агрономам отримувати реальні дані та своєчасно реагувати на зміни у ґрунтових умовах та кліматі.

Крім того, сучасне використання дронів у сільському господарстві для моніторингу агротехнічних об'єктів є яскравим прикладом ефективного застосування інформаційних технологій. Дрони, обладнані спеціалізованими сенсорами, надають деталізовану інформацію для аналізу стану полів та вчасного виявлення проблем.

Завершуючи, обговорені питання стосуються не лише технічних аспектів розробки веб-додатків, але й важливості їх впровадження в агросферу для підвищення продуктивності та стійкості сільськогосподарського виробництва. Висвітлені приклади є свідченням того, як інформаційні технології допомагають зробити сільське господарство більш ефективним та інноваційним у вирішенні сучасних викликів.

РОЗДІЛ 2
ЗБІР, СИСТЕМАТИЗАЦІЯ ТА ДОСЛІДЖЕННЯ ІНФОРМАЦІЇ ПРО СУЧАСНІ МОДЕЛІ ВЕБ-ЗАСТОСУНКІВ НА БАЗІ СУКУПНОСТІ ТЕХНОЛОГІЙ ANGULAR ТА FIREBASE

2.1 Узагальнена структурно-функціональна організація системи

Функціональним призначенням даного додатку є функціонування WEB-додатку для моніторингу ґрунтокліматичних параметрів агротехнічних об'єктів, а саме: графічне відображення даних, які зберігаються у базі даних та надсилаються програмі через хмарний сервер. Дані повинні являти собою об’єкти з полями різних типів, які заздалегідь були зазначені у файлах моделей чи інтерфейсів. Серед даних можна виділити такі об’єкти: таблиця зареєстрованих корпоративних користувачів для авторизації, таблиця записів щодо різних складових агротехнічних сполук які містять у собі поля з інформацією для аналізу та моніторингу. Також інтерфейс повинен надавати змогу не тільки приймати дані з сервера, а і відсилати їх йому. Зокрема повинна бути присутня функція додавання нових записів у таблицю агротехнічних об’єктів, а також змога видалення цих записів. Також повинна бути присутня форма реєстрації нових корпоративних користувачів системи. Зі сторони користувача функціональне призначення полягає у змозі відображення додатком списку усіх агросполук із даними для моніторингу зі змогою їх сортування для зручного пошуку та окремого виділення агросполук у яких є загроза зараження інфекцією, а також змога ці сполуки додавати до бази даних чи видаляти з неї. Також присутня можливість реєстрації нового користувача через просту форму.

Експлуатаційне призначення програми для фермера чи співробітника агровиробництва полягає в змозі моніторингу, обліку та редагування інформації про ґрунтокліматичні параметри агротехнічних об'єктів. А також у змозі їх додавання до бази даних, або видалення з неї за допомогою зручного графічного інтерфейсу.

2.2 Модель досліджуваної системи. Опис використаної архітектури та шаблонів проектування

Архітектура програмного забезпечення, що використовується для розробки та функціонування програмного додатку має назву «клієнт-сервер».

В основі клієнт-серверної архітектури лежать два компоненти: клієнт і сервер.

Клієнт – комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій.

Сервер – більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних.

Модель такої системи полягає в тому, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат відправляється клієнтові (Рис. 2.1) [14]. Сервер може обслуговувати кілька клієнтів одночасно. Якщо одночасно приходить більше одного запиту, то вони встановлюються в чергу і виконуються сервером послідовно. Іноді запити можуть мати пріоритети. Запити з більш високими пріоритетами повинні виконуватися раніше.

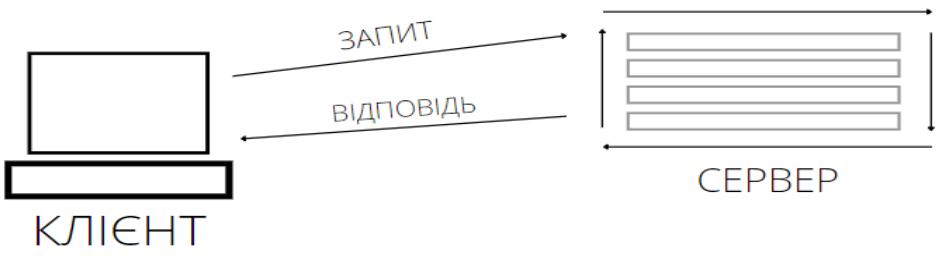

Рис. 2.1. Схема функціонування архітектури Клієнт-сервер

Функції, які реалізуються на сервері:

- зберігання, доступ, захист і резервне копіювання даних;

- обробка клієнтського запиту;
- відправлення результату (відповіді) клієнту.

Функції, які реалізуються на стороні клієнта:

- надання користувальницького інтерфейсу;
- формулювання запиту до сервера і його відправка;
- отримання результатів запиту і відправка додаткових команд (запитів на додавання, оновлення або видалення даних).

Архітектура клієнт-сервер визначає принципи спілкування між комп'ютерами, а правила і взаємодії визначені в протоколі [13].

Мережевий протокол – це набір правил, за якими відбувається взаємодія між комп'ютерами в мережі [13].

В даній дослідницькій роботі використовується протокол HTTP – це протокол передачі гіпертексту, на основі якого працюють всі сайти. Він запитує необхідні дані у віддаленій системи (веб-сторінки, файли) [13].

Наша концепція побудови системи клієнт сервер застосовує варіант з сильним клієнтом.

Сильний клієнт – концепція, в якій частина обробки інформації надається клієнтові. У такому випадку сервер виступає сховищем даних, а вся робота по обробці та подання інформації переноситься на комп'ютер клієнта [13].

Система (застосунок), яка заснована на клієнт-серверній взаємодії, включає три основних компоненти: уявлення даних, прикладний компонент, компонент управління ресурсами і їх зберігання [13].

Наша архітектура є дворівневою і складається з двох вузлів (див. рис. 2.2):

а) сервер, який відповідає за отримання запитів і відправку відповідей клієнту, використовуючи при цьому лише власні ресурси;

б) клієнт, який представляє користувацький інтерфейс.

Принцип роботи полягає в тому, що сервер отримує запит, обробляє його і відповідає безпосередньо, без використання сторонніх ресурсів.

Взаємодія клієнт-сервер дозволяє розділяти функціонал і обчислювальне навантаження між клієнтськими додатками (замовниками послуг) і серверними

додатками (постачальниками послуг). Знання архітектури додатка дозволяє тестувальнику більш якісно провести функціональне, крос-браузерне тестування, тестування юзабіліті і швидкодії.

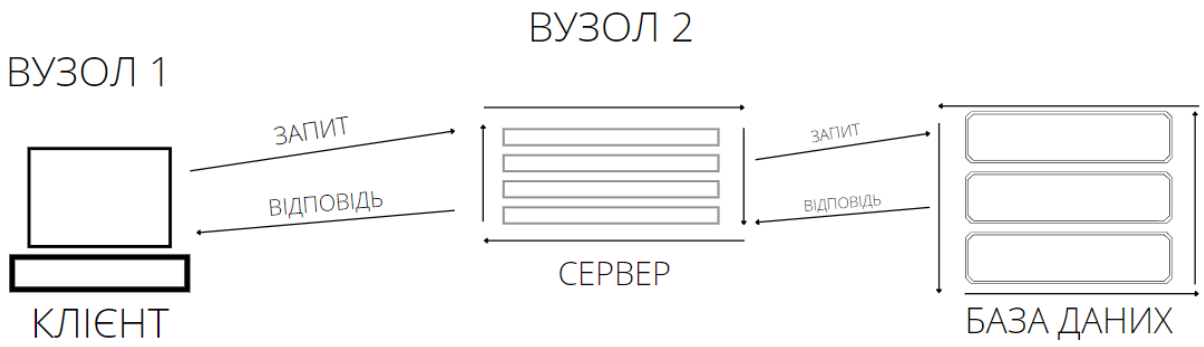

Рис. 2.2. Схема структури з двома вузлами

2.3 Опис алгоритмів роботи досліджуваної системи. Опис використаних технологій та мов програмування

Головною технологією розробки front-end частини є фреймворк під назвою Angular, як вже було зазначено у пункті 1.1 – він має низку переваг серед інших фреймворків, здебільшого це:

- програми, написані на Angular, сумісні з різними браузерами. Angular автоматично обробляє код JavaScript, що підходить для кожного браузера;
- чистий і точний дизайн інтерфейсу користувача;
- зручна та проста маршрутизація;

- структура Angular полегшує розширення синтаксису HTML і легко створює повторні компоненти за директивами.

Також на вибір фреймворку вплинуло власне бажання студента.

Як можна побачити на рис. 2.3 – Angular має досить зручну схему роботи з компонентами, розподіляючи функціонал програми на окремі модулі для більш захищеної та точної роботи функцій застосунку. Головними файлами є файл

шаблону (Template) де міститься розмітка сторінки у форматі HTML, та файл компонента (Component) написаний мовою TypeScript та відповідаючий за логіку роботи програми, і який і зв'язує шаблон з усіма іншими допоміжними сервісами, такими як моделі компоненті (Component Module) і моделі сервісів (Service Module) – вони, як правило, містять у собі правила та методи зв'язку з back-end частиною застосунку та містять чіткі форми з полями, де описуються потрібні типи даних для представлення об'єктів. Також Angular може використовувати різні допоміжні сервіси та директиви (Directive).

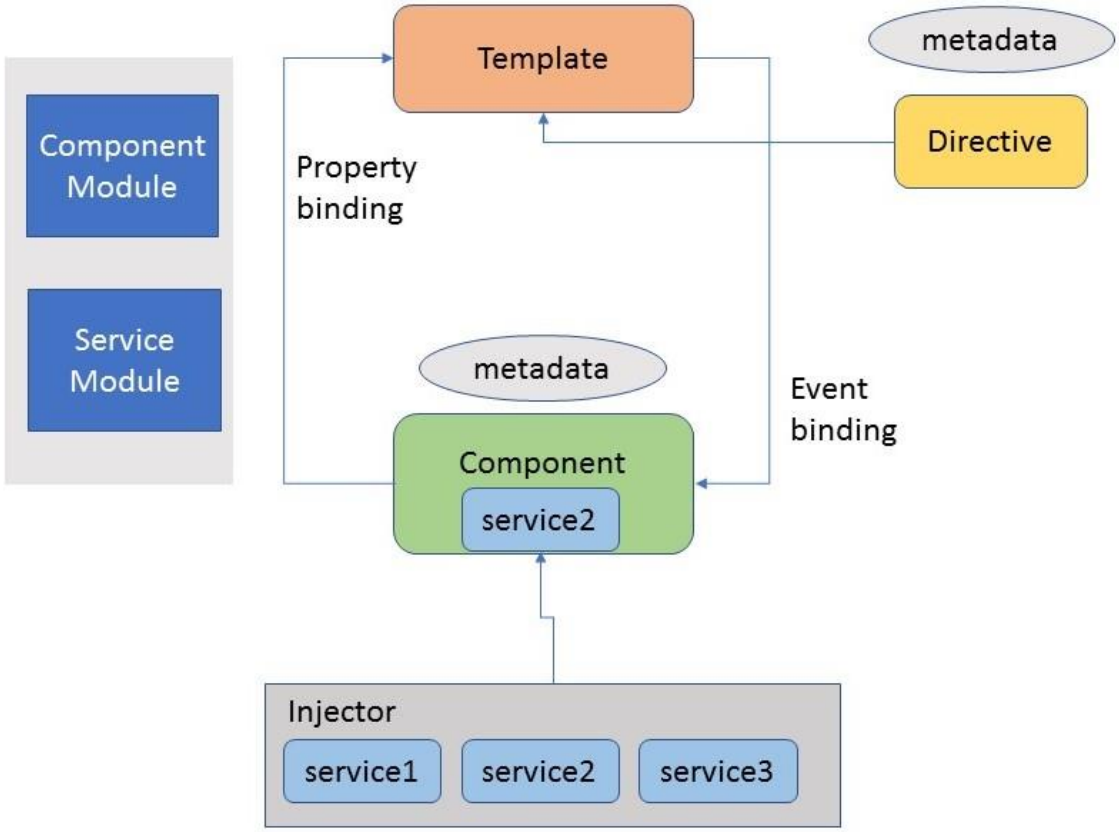

Рис. 2.3. Схема роботи будь-якого компоненту Angular

Мовою програмування логіки front-end частини застосунка була обрана мова TypeScript тому що вона є типізованим аналогом JavaScript і вносить поняття статичної типізації змінних, також дозволяє використання класів, інтерфейсів та модулів у різних структурах проекту, що робить код більш

схожим до концепції об’єктно-орієнтовного програмування, а це, в свою чергу, робить проект більш стабільним та захищеним від втручання у код зловмисників.

Мовою гіпертекстової розмітки була обрана мова HTML тому що вона є більш розповсюдженою у світі, підтримується всіма браузерами та фреймворками, а отже є універсальною і простою та ефективною у застосуванні.

Мовою каскадних таблиць та стилізації документа було обрано SCSS тому що вона також як і HTML підтримується більшістю браузерів та є більш зручною, ніж звичайний CCSS, тому що краще реалізує поняття інкапсуляції класів.

Також для передачі даних був застосований формат JSON, тому що він базується на тексті, може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується переважно для передавання структурованої інформації через мережу.

2.4 Висновок за розділом

В процесі проведених досліджень та аналізу було встановлено, що Angular — це потужний фреймворк, який визначається своєю ефективністю та зручністю у розробці веб-застосунків, зокрема на архітектурі клієнт-сервер. Його популярність визначається рядом ключових переваг, які роблять його ідеальним вибором для сучасних веб-розробок.

Однією з основних переваг Angular є його компонентний підхід до розробки. Компоненти є основними будівельними блоками додатку, і вони можуть бути використані у різних частинах програми. Це забезпечує чистоту коду, полегшує управління та підтримку проекту. Кожен компонент в Angular — це самостійна одиниця, що робить його легким у розумінні та масштабуванні.

Фреймворк також славиться своєю системою залежностей та ін'єкцією залежностей, які спрощують взаємодію між різними частинами додатку. Це дозволяє розробникам швидко та ефективно інтегрувати нові функції та забезпечує гнучкість в управлінні додатком.

Ще однією важливою рисою є двостороннє зв'язування даних, що робить роботу з інтерфейсом більш динамічною та реактивною. Зміни в інтерфейсі автоматично відображаються в даних та навпаки, що спрощує взаємодію користувача з додатком.

Angular також активно підтримує велику спільноту розробників та має обширну документацію. Це робить процес вивчення та використання фреймворку більш доступним та зручним для новачків, а також дозволяє швидко отримувати відповіді на питання та розв'язувати проблеми під час розробки.

Усі ці фактори роблять Angular ідеальним інструментом для розробки веб-застосунків на архітектурі клієнт-сервер. Його потужність, ефективність та спрощення у використанні роблять його вибором номер один для тих, хто прагне створити сучасний та ефективний веб-додаток.

РОЗДІЛ 3
РОЗРОБКА WEB-ЗАСТОСУНКУ ДЛЯ МОНІТОРИНГУ ҐРУНТОКЛІМАТИЧНИХ ПАРАМЕТРІВ АГРОТЕХНІЧНИХ ОБ'ЄКТІВ

3.1 Структура WEB-додатку для моніторингу ґрунтокліматичних параметрів агротехнічних об'єктів

Внутрішня структура створюваної системи являє собою веб-сайт, який складається з двох частин: клієнтська презентаційна (frontend) та серверна (backend). Клієнтська частина виповнює адміністративні функції з усіма супутніми правами доступу, так як планом проекту зумовлюється що до програми матимуть доступ лише зареєстровані корпоративні користувачі, тобто співробітники компанії.

Користувач системи взаємодіє безпосередньо з клієнтською частиною, через функціонал якої має змогу:

- зареєструватись;
- увійти в систему;
- переглянути усі дані про ґрунтокліматичні параметри агротехнічних об'єктів;
- видалити існуючі дані;
- додати нові дані до бази даних через спеціальну форму.

Система комунікації презентаційної і серверної частини додатку зображена на рис. 3.1.

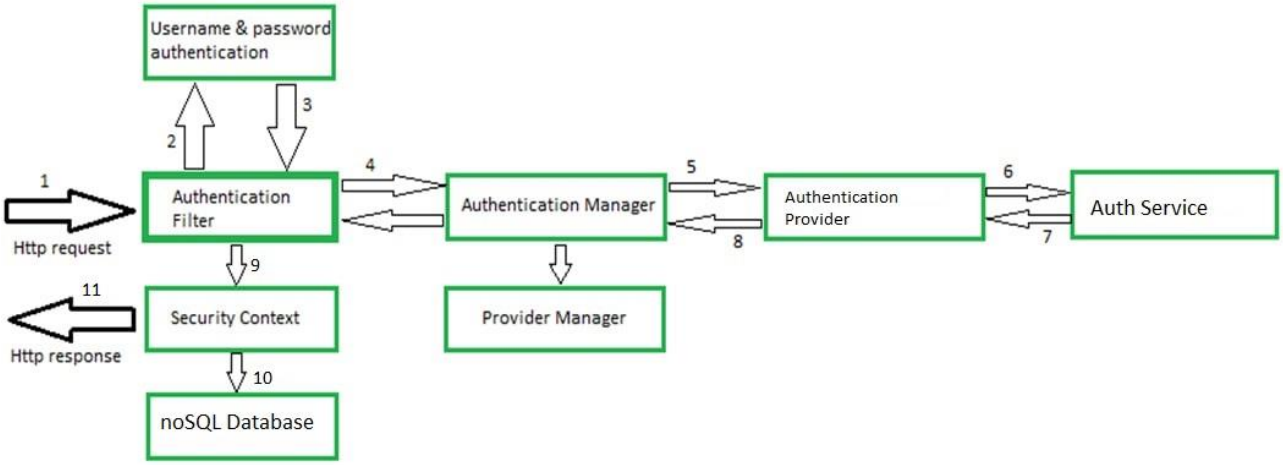

Рис. 3.1. Архітектура комунікації частин програми

Алгоритм взаємодії має наступну структуру:

1. користувач за допомгою веб-клієнта (браузера) надсилає запит певної веб-сторінки застосунку;
2. незалежно від почтакової сторінки додаток для моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів (надалі Agrolyze) перенаправляє клієнта на сторінку авторизації;
3. перед тим, як запит потрапляє до обробки бізнес-логіки та відправки результату, він спочатку перехоплюється ланцюжком фільтрів безпеки для аутентифікації та авторизації;
4. коли запит перехоплюється відповідним фільтром аутентифікації (AuthenticationFilter), він отримує ім'я користувача та пароль із запиту і створює об'єкт аутентифікації;
5. використовуючи створений об'єкт аутентифікації, фільтр викликає метод аутентифікації менеджера аутентифікації (Authentication Manager), який є лише інтерфейсом, а фактична реалізація методу автентифікації забезпечується менеджером провайдерів (ProviderManager);
6. зі свого методу автентифікації ProviderManager викликає метод автентифікації відповідного AuthenticateProvider, у відповідь отримуючи Основний об'єкт автентифікації (Principal Authentication Object), якщо автентифікація є успішною;

7. у Agrolyze для реалізації інтерфейсу з єдиним методом автентифікації AuthenticationProvider було обрано провайдер автентифікації AuthenticationProvider, який отримує дані користувача з AuthService, на цьому кроці відбувається вся фактична автентифікація;
8. використовуючи сервіс AuthService, постачальник автентифікації (AuthenticationProvider) отримує об'єкт користувача, що відповідає обліковим даним з бази даних, ці дані об'єкта користувача порівнюються з вхідними обліковими даними об'єкта автентифікації, якщо автентифікація успішна, то у відповідь повертається Основний об'єкт автентифікації (Principal Authentication Object);
9. якщо обрана бізнес-логіка потребує зв'язку з базою даних, то по налаштованому при запуску додатку підключенню до БД, відбувається взаємодія з сервером noSQL, що приймає запит до бази даних, обробляє його, а потім відправляє результати у відповідь;
10. контролер завершує виконнання обробки запиту, формуючи результат, та надає презентаційній частині програми відповідь для відображення.

3.2 Структура та опис бази даних

Google Firebase - це облачна платформа, яка надає широкий спектр інструментів для розробки веб та мобільних додатків. Однією з ключових особливостей Firebase є вбудована noSQL база даних в реальному часі, яка використовується для зберігання та синхронізації даних між різними пристроями[17].

Firebase Realtime Database працює на основі моделі документ-орієнтованої бази даних, де дані представлені у вигляді JSON-об'єктів. Вона забезпечує миттєве оновлення даних у реальному часі, що робить її ідеальним інструментом для розробників, які потребують синхронізації даних між клієнтами та сервером.

Однією з переваг Firebase Database є простота інтеграції з іншими сервісами Firebase, такими як аутентифікація користувачів, хостинг та аналітика. Це робить його зручним та компактним рішенням для розробки з приділенням менше часу на конфігурацію та обслуговування [17].

Загалом, Firebase Realtime Database - це надійний інструмент для розробників, які цінують простоту використання та потребують реальної синхронізації даних між різними платформами та пристроями.

В представленому у цій роботі проекті база даних складається з двох секторів:

- Сектор з даними про агротехнічні сполук;
- Сектор з даними про зареєстрованих у системі користувачів.

Сектор даних про агротехнічні сполуки являє собою список JSON-об’єктів ключем до яких є унікально-згенерований ID. Кожен об’єкт має певні властивості (приклад списку наведений на рис. 3.2.):

- AirTemMin – мінімаьна температура повітря для певної дати, вимірюється у °C;
- AirTempMax - максимальна температура повітря для певної дати, вимірюється у °C;
- AirTempAvg - середня температура повітря для певної дати, рахується програмно при створенні нового запису у таблиці, вимірюється у °C;
- Date – дата за яку показує інші поля, тип даних Date;
- RelHumidity – відносна вологість повітря, вимірюється у відсотках (%);
- LeafWetnness – вологість листя, вимірюється у хвилинах із-за специфіки вимірювального прибору;
- Precipitation – атмосферні опади, вимірюється у міліметрах;
- Infection – інфекованість ґрунту, вимірюється у відсотках (%);

- IsInfected – програмно-вичислювальне поле, яке э типу Boolean та ставить позначку true або false в залежності від того чи інфікований ґрунт взагалі.

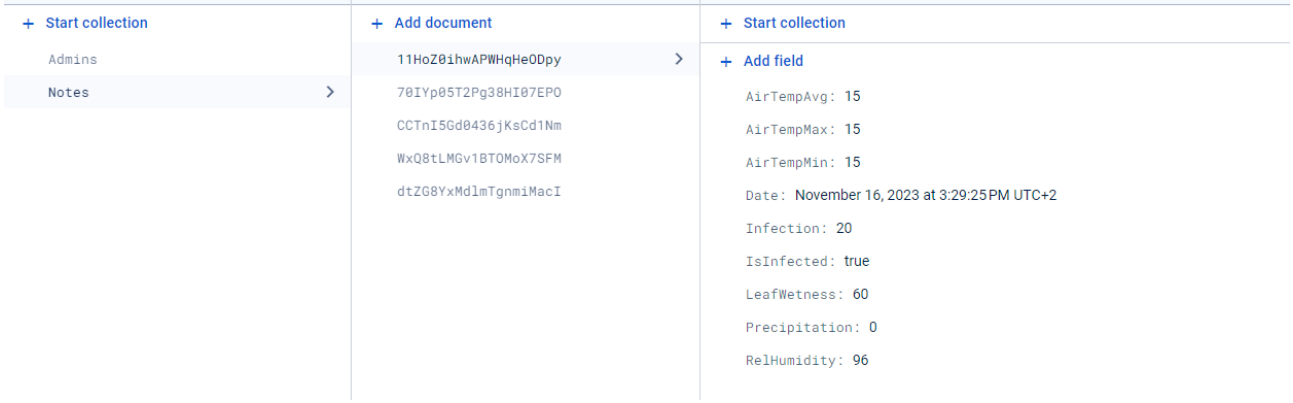

Рис. 3.2. Структура бази даних моніторингових об’єктів та їх властивості

Хочеться навести деяке пояснення щодо вологості листя: вологість листя вимірюється за допомогою спеціального датчика, який має певний алгоритм роботи. Датчик вимірює діелектричну проникність зони приблизно 1 см від поверхні датчика. Діелектрична проникність води (≈ 80) значно вище, ніж у повітря (≈1), так що вимірювана діелектрична проникність сильно залежить від наявності вологи на поверхні датчика. Датчик видає сигнал, пропорційний до діелектрика вимірюваної зони, і, відповідно, пропорційний кількості води на поверхні датчика. Говорячи простіше це кондуктометрична комірка: якщо волога є, то контакти замикаються і починається відлік часу. Як тільки лист стає сухим, то контакту немає, і відповідно, відлік часу зупиняється.

Сектор з даними про зареєстрованих у системі користувачів зберігається окремо від основної бази даних і є частиною системи FireAuth сервісу Firebase. Він представляє собою спеціальну таблицю до якої можна надсилати post та get http запити (приклад таблиці наведений на рис. 3.3).

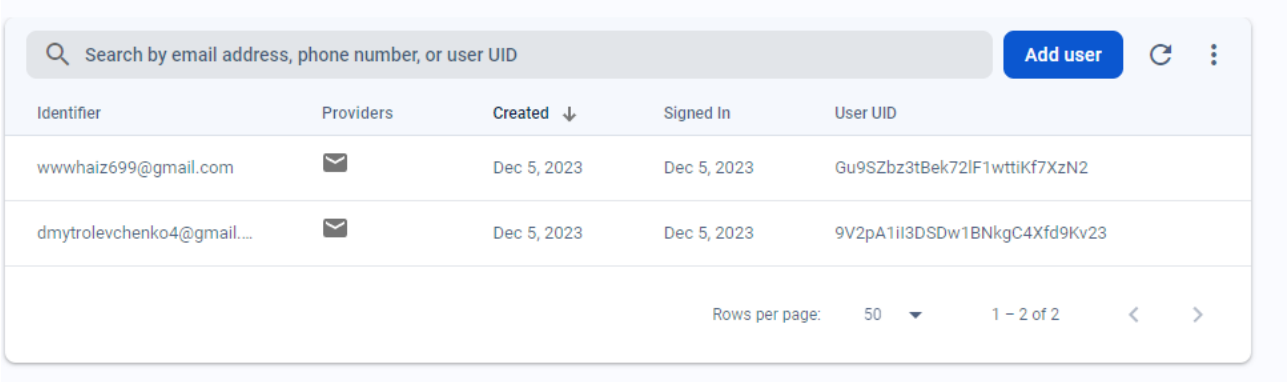

Рис. 3.3. Таблиця зареєстрованих користувачів системи

3.3 Опис системи Agrolyze та алгоритмів її функціонування

Основна обробника даних в системі відбувається в серверній частині застосунку. В якості вхідних даних в програмному комплексі використовуються текстові дані з форм, які користувач заповнює для додавання / змінення / видалення інформації про агросполуки. Вихідними даними веб-додатку можна вважати множину збережених у системі моніторингових даних, готових до використання та відображених на сторінках перегляду цих даних, сгрупованих за їх типом.

Функціональне призначення системи полягає в наданні клієнту інтуїтивно-зрозумілого і зручного інструмента для моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів із можливістю їх подальшого додавання чи вилучення з бази.

Для розробки застосунку було застосовано наступні технології:

1) Angular - відкритий фреймворк, підтримуваний Google, працюючий на мові TypeScript за технологією MVC [1];
2) TypeScript – більш чутка та типізована версія мови програмування JavaScript [1];
3) HTML – язик гіпертекстової розмітки для створення розмітки на UI [1];

4) Angular Material – створений спеціально для Angular сервіс, який дозволяє розширити функціонал фреймворку та внести органічну стилізацію зовнішнього вигляду;
5) SCSS – препроцесор мови CSS який використовується для придання додатку стилізації та більш зручної візуалізації [1];
6) Google Firebase – комплексний сервіс від Google, який включає в себе багато модулів для зберігання даних та операцій над ними а також виконує роль backend сервера;
7) FireStore – хмарне noSQL сховище даних;
8) FireAuth – хмарний сервіс для авторизації користувачів у системі;
9) AuthInterceptor – сервіс для покращення авторизації, підвищення захищеності застосунку.

Розроблений застосунок для моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів має відповідати наступним вимогам:

- інтуїтивно-зрозумілий інтерфейс для роботи з системою;
- наявна система автентифікації та авторизації;
- реєстрація користувачів;
- зручна навігація по веб-сайту;
- можливість роботи в браузерах на ПК;
- шифрування даних користувачів;
- наявність модулів створення та видалення ресурсів;
- гнучкість для модифікацій та масштабування.

Інтерфейс користувача веб-сайту складається з основних модулів:

1) Сторінка авторизації;
2) Сторінка реєстрації;
3) Сторінка з даними про моніторинг ґрунтокліматичних параметрів агротехнічних об’єктів;
4) Сторінка з формою додавання нового запису у систему моніторингу;

5) Модальне вікно з підтвердженням про видалення моніторингового запису з бази даних застосунку.

Графік компонентів та сервісів системи та як вони взаємодіють між собою зазначений на Рис. 3.4.

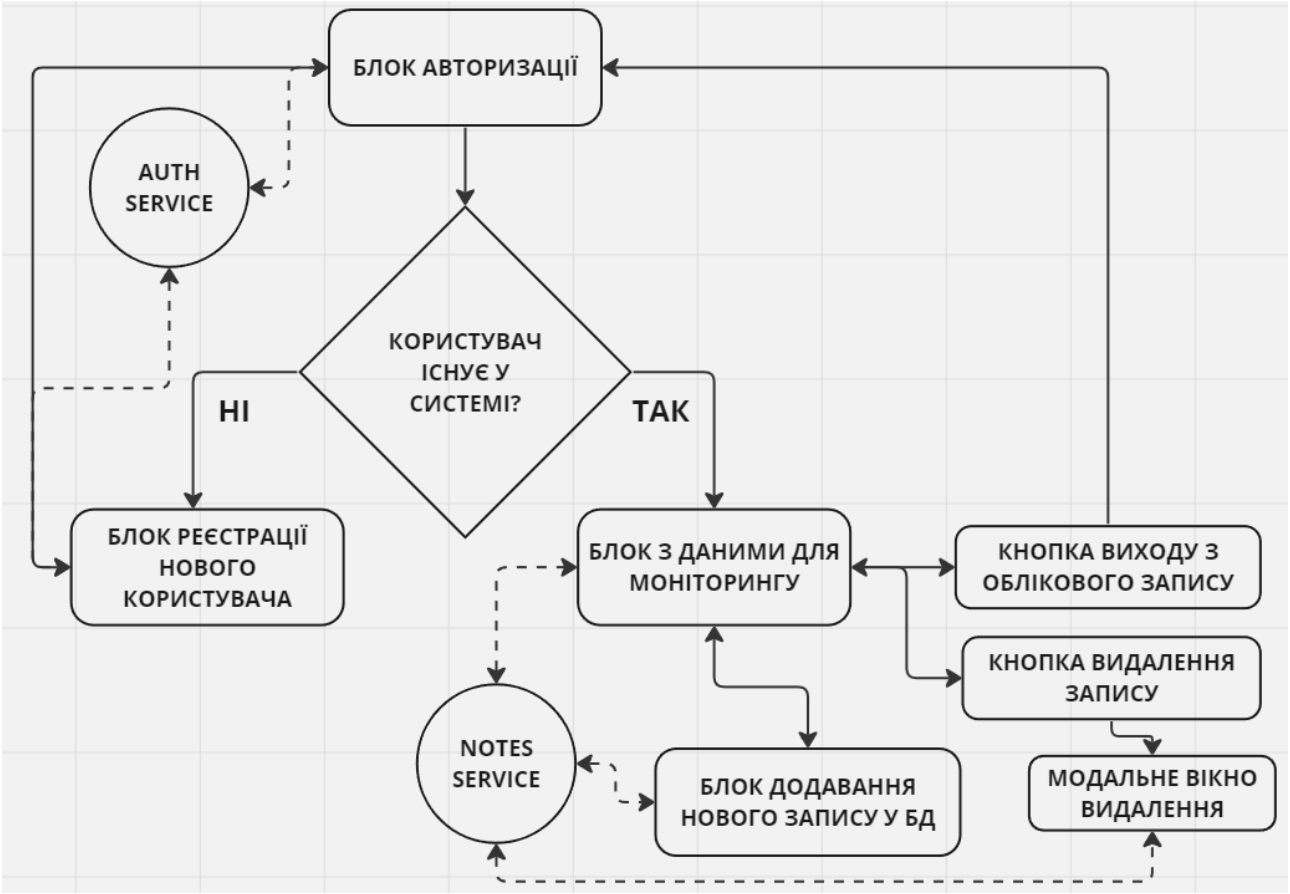

Рис. 3.4. Компоненти та сервіси застосунку Agrolyze та як вони взаємодіють між собою

При першому запуску програми користувач потрапляє на сторінку авторизації, що зображена на рис. 3.5.

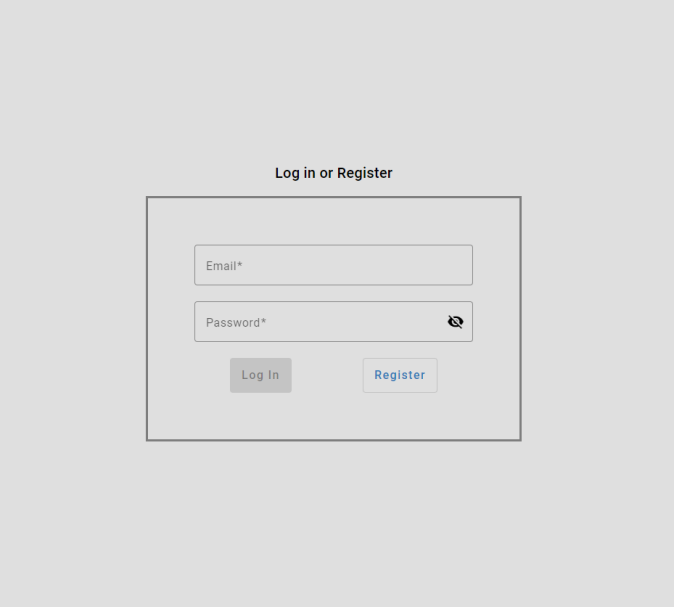

Рис. 3.5. сторінка авторизації користувача

Якщо користувач авторизований в системі, то він може увійти за посиланням «Log In», зазначивши у формі свій логін-пошту та пароль. Для реєстрації в системі потрібно натиснути на посилання «Register» та ввести облікові дані у формі на рис. 3.6.

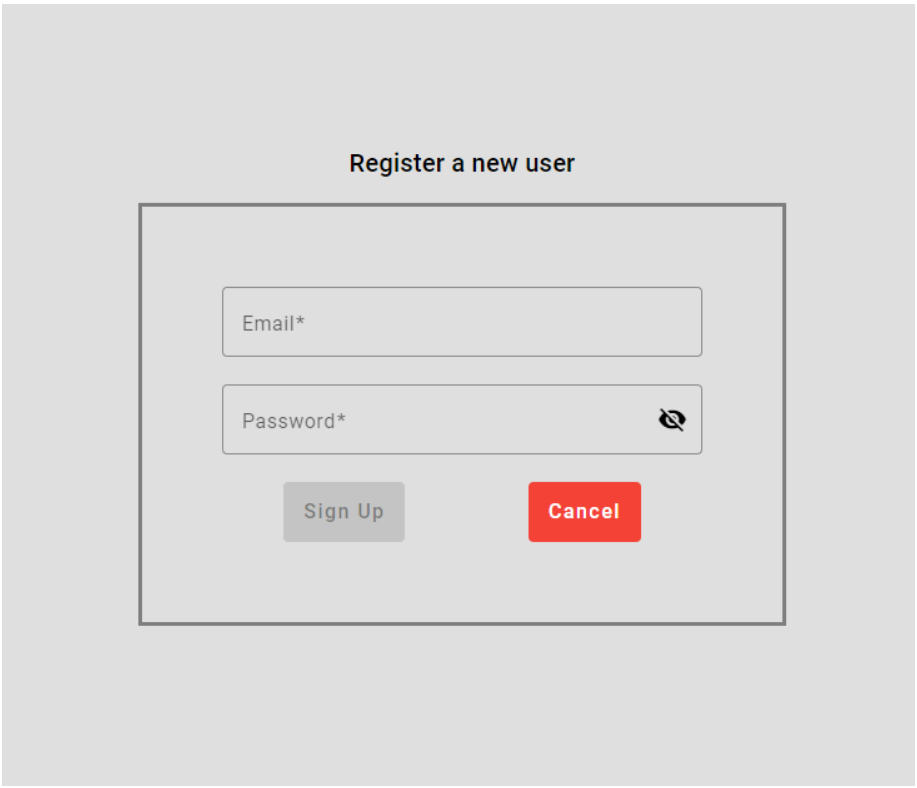

Рис. 3.6. Форма реєстрації користувача

Після успішної авторизації користувач потрапляє на сторінку доступних йому моніторингових записів, зображених на рис. 3.7.

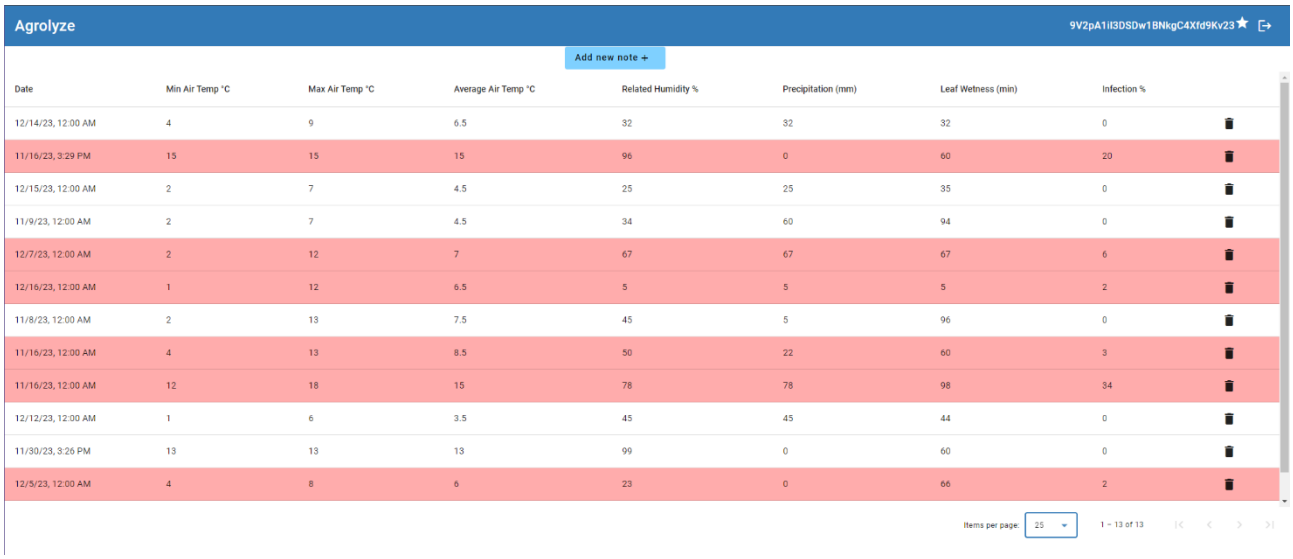

Рис. 3.7. таблиця моніторингових записів ґрунтокліматичних параметрів агротехнічних об’єктів

Якщо в даних є інформація про інфікованість досліджуваного об’єкта, тоді запис відображається полем блідо-червоного кольору – це зроблено для зручності перегляду. У таблиці присутнє сортування, яке можна увімкнути натиснувши на заголовок будь-якої колонки таблиці по якій користувач хоче виповнити сортування (процес сортування зображений на рис. 3.8). Також у таблиці присутній пейджинг та вибір кількості записів на одній сторінці (елемент пейджингу зображений на рис.3.9).

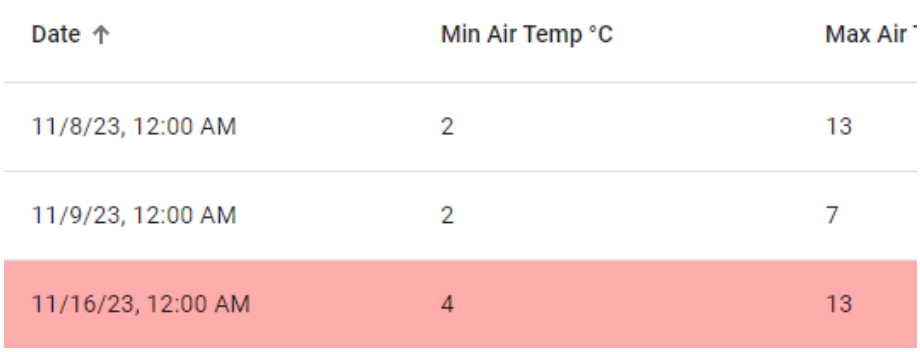

Рис. 3.8. Приклад процесу сортування

Рис. 3.9. Приклад пейджингу в додатку

Натиснувши на кнопку «Add new note +», яка зображена на рис. 3.10, користувач потрапляє на сторінку додавання нового запису у базу даних. Форма додавання нового запису зображена на рис. 3.11.

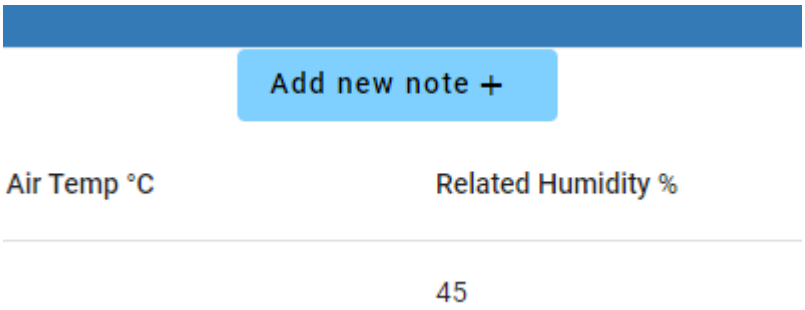

Рис. 3.10 кнопка навігації на сторінку додавання запису

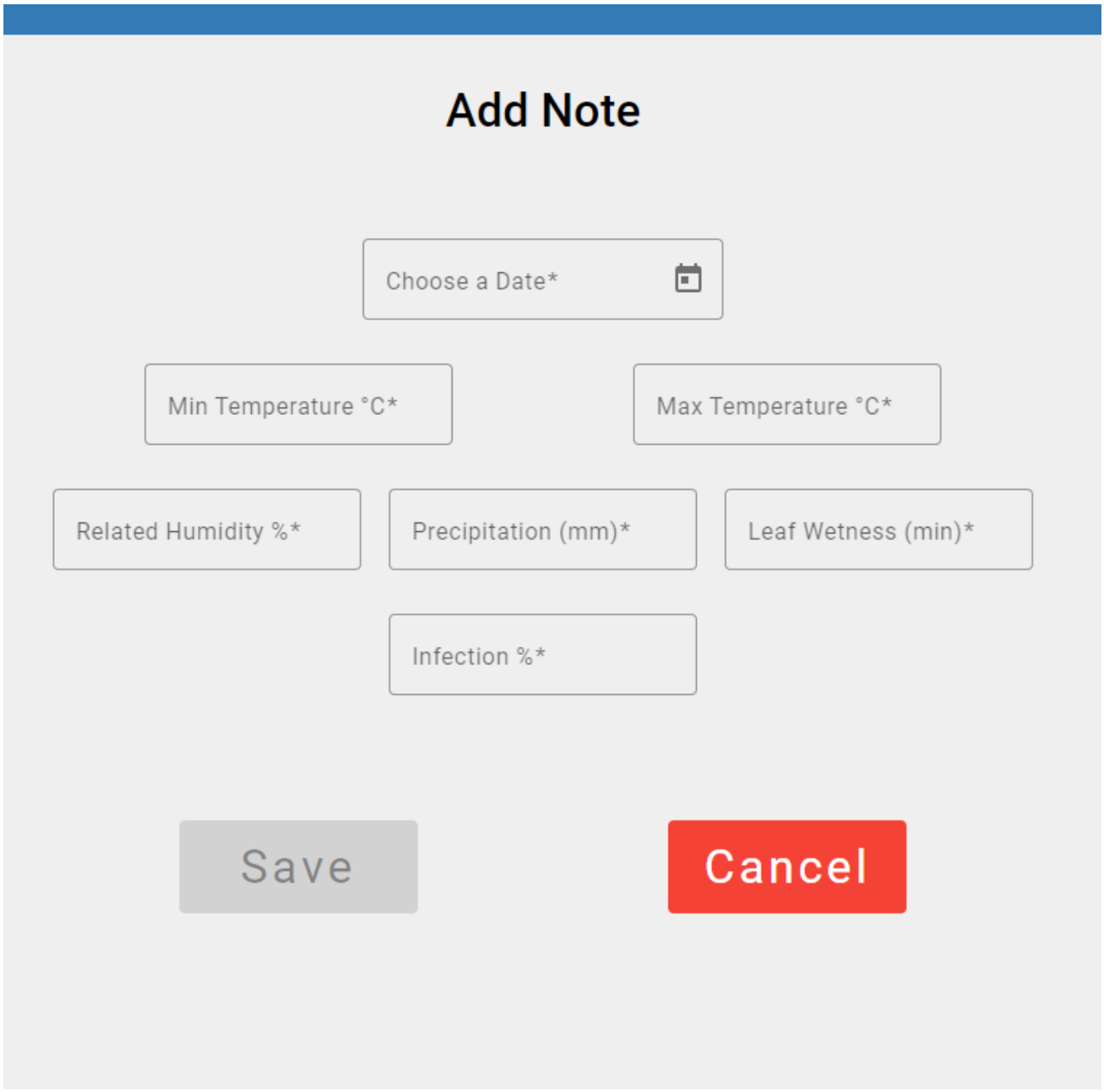

Рис. 3.11. Форма додавання нового запису у базу даних

Хочу зазначити, що у формі присутня декотра валідація полів, яку можна побачити в дії на рис. 3.12. Якщо форма не проходить валідацію, тоді кнопка збереження запису стає неактивною. Валідація відбувається динамічно в реальному часі.

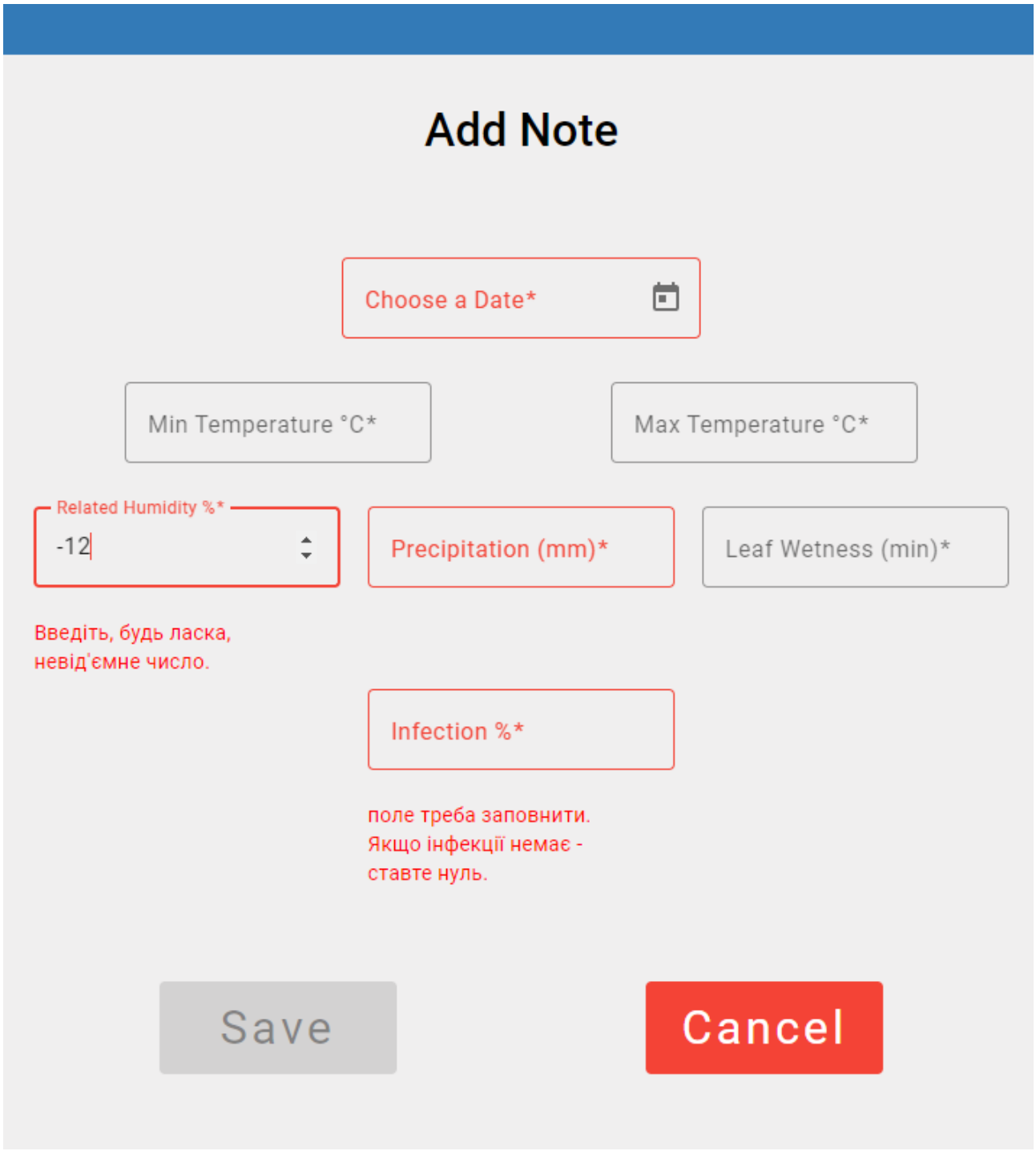

Рис. 3.12. Приклад механізму валідації форми

Якщо форма заповнена правильно і всі поля відповідають валідації – тоді кнопка «Save» стаєактивною і дозволяє зберегти дані, приклад валідної форми наведено у рис. 3.13. Кнопка «Cancel» повертає користувача на сторінку із таблицею без збереження даних.

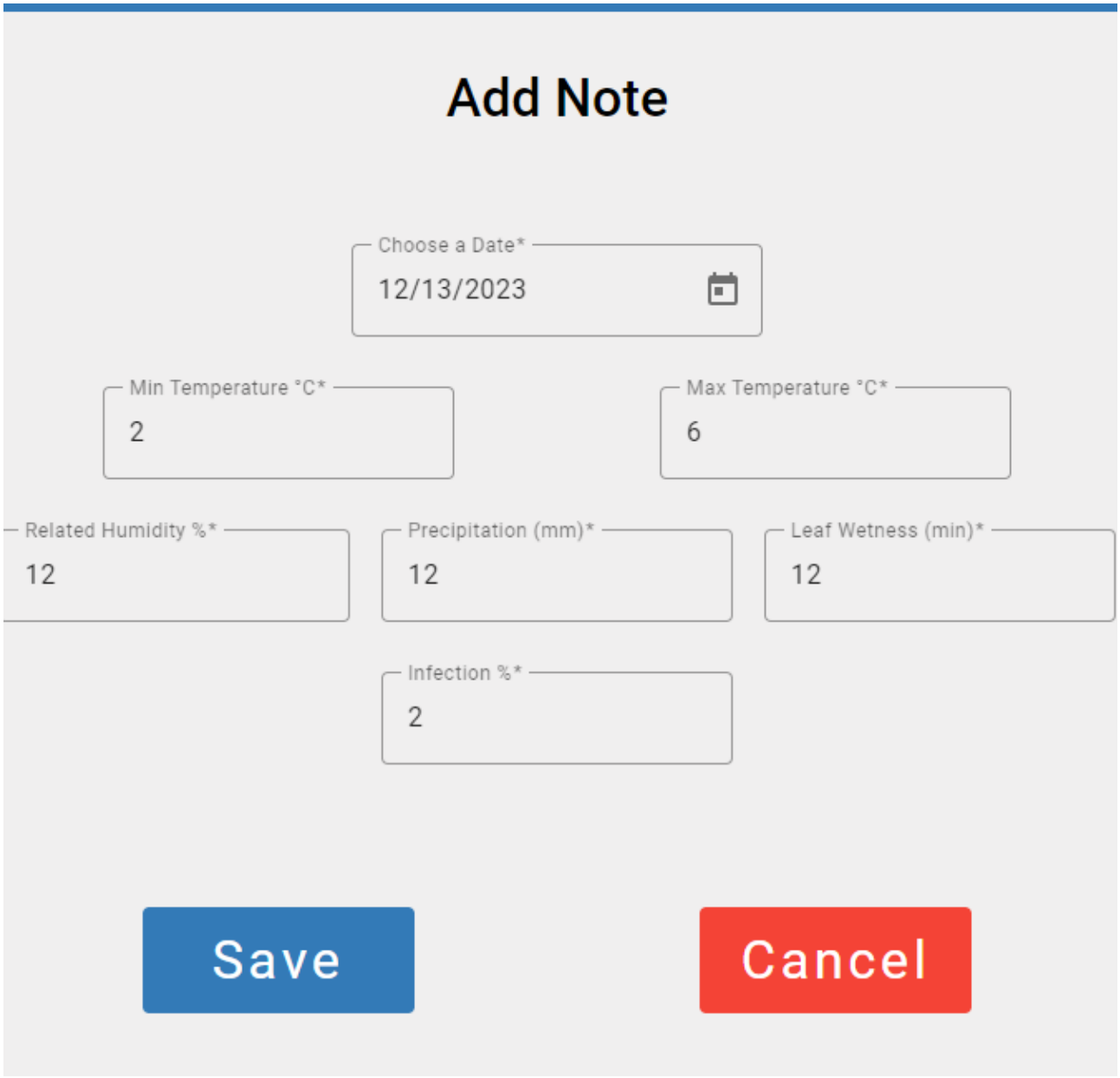

Рис. 3.13. Приклад валідної форми

Після збереження запис було додано до таблиці що ми можемо бачити на рис. 3.14.

Рис. 3.14. Збережений новий запис у таблиці

Якщо ми хочемо видалити запис тоді треба натиснути на іконку смітника в самій правій колонці таблички і тоді в нас висвітиться модальне вікно із підтвердженням видалення, воно зображене на рис.3.15.

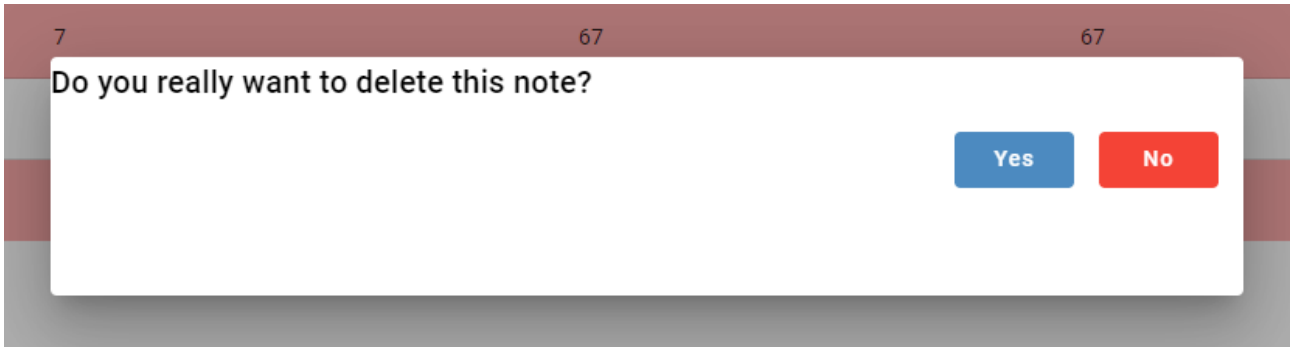

Рис. 3.15. Модальне вікно підтвердження видалення

Якщо обрати варіант «Yes» то запис видалиться, якщо обрати варіант «No» тоді модальне вікно закриється і користувача поверне на табличку.

Зправа зверху біля напису «Welcome», який свідчить про те, що користувач зареєстрований у системі ми бачимо кнопку виходу з облікового запису, зображено на рис. 3.16.

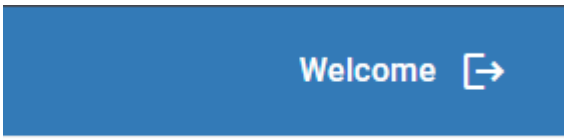

Рис. 3.16. Кнопка виходу з облікового запису

Якщо натиснути на неї – ми попадемо знову на сторінку логіну, користувача виштовхне з його облікового запису і надпис «Welcome» зникне. Ця логіка зображена на рис. 3.17.

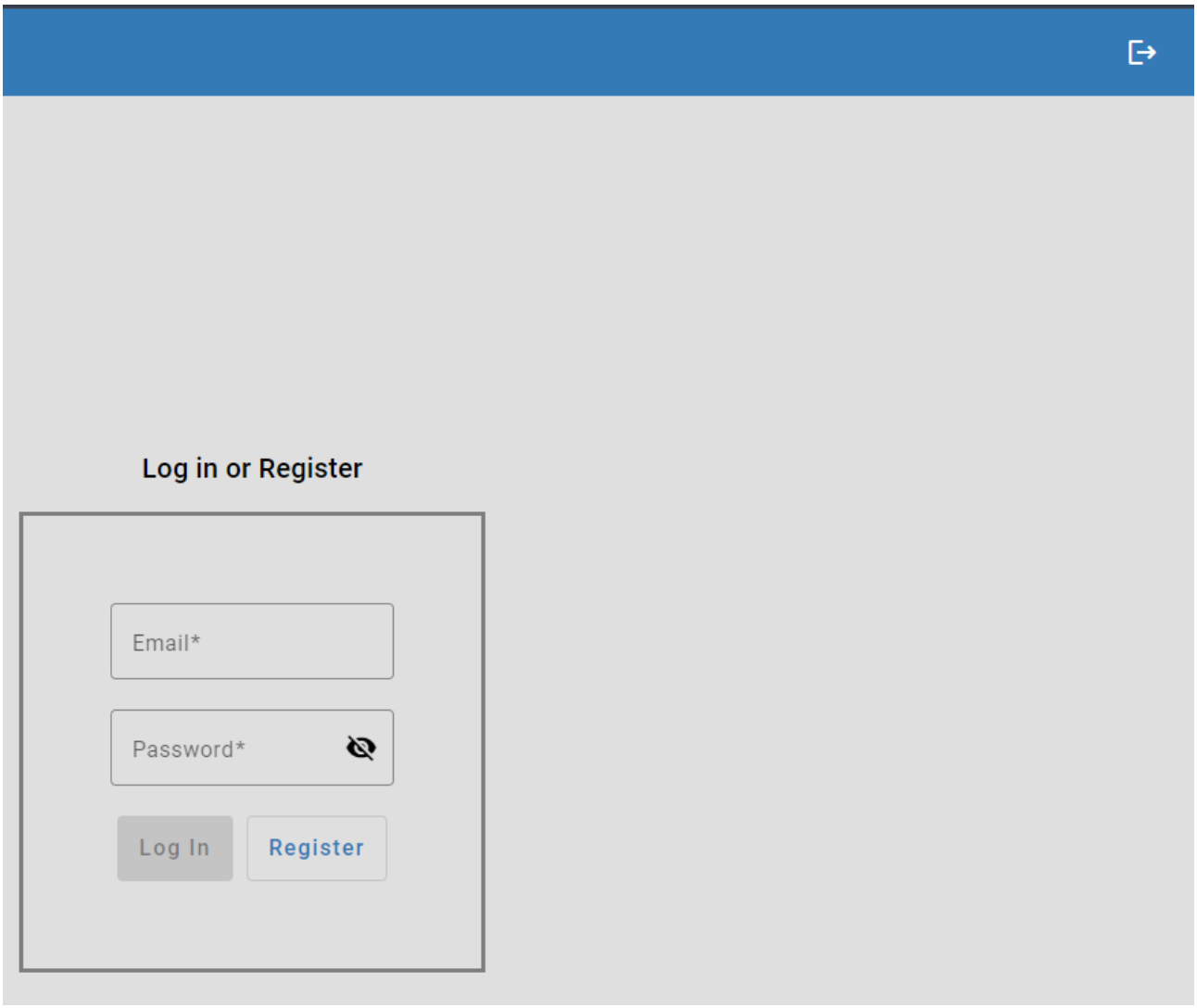

Рис. 3.17. Результат виходу з акаунту користувача

На обох формах поле введення паролю має динамічний перемикач типу для того щоб можна було піддивитись правильність введеного паролю. Приклад роботи перемикача зображений на рис. 3.18. та 3.19.

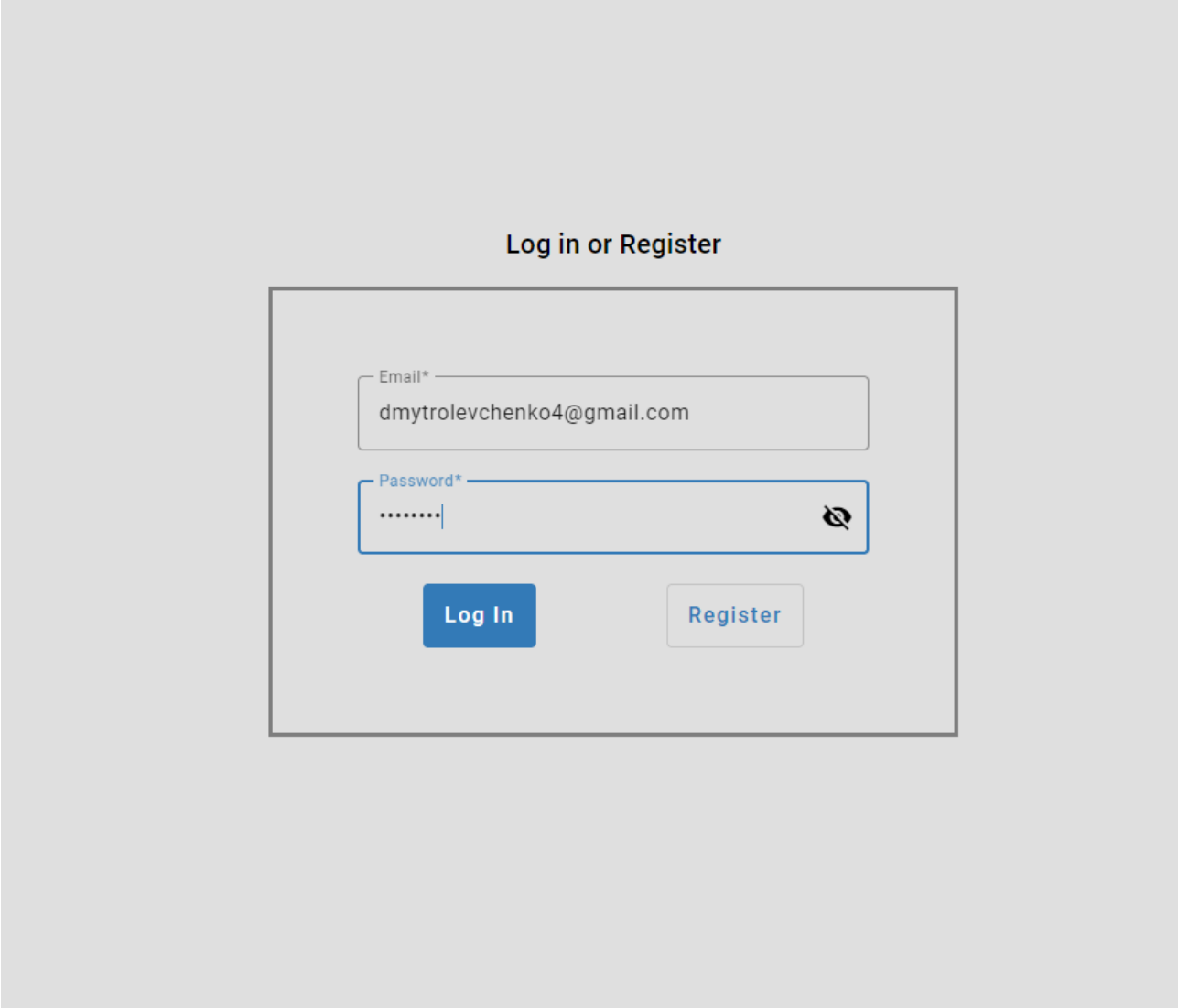

Рис. 3.18 Перемикач прозорості паролю у вимкненому положенні

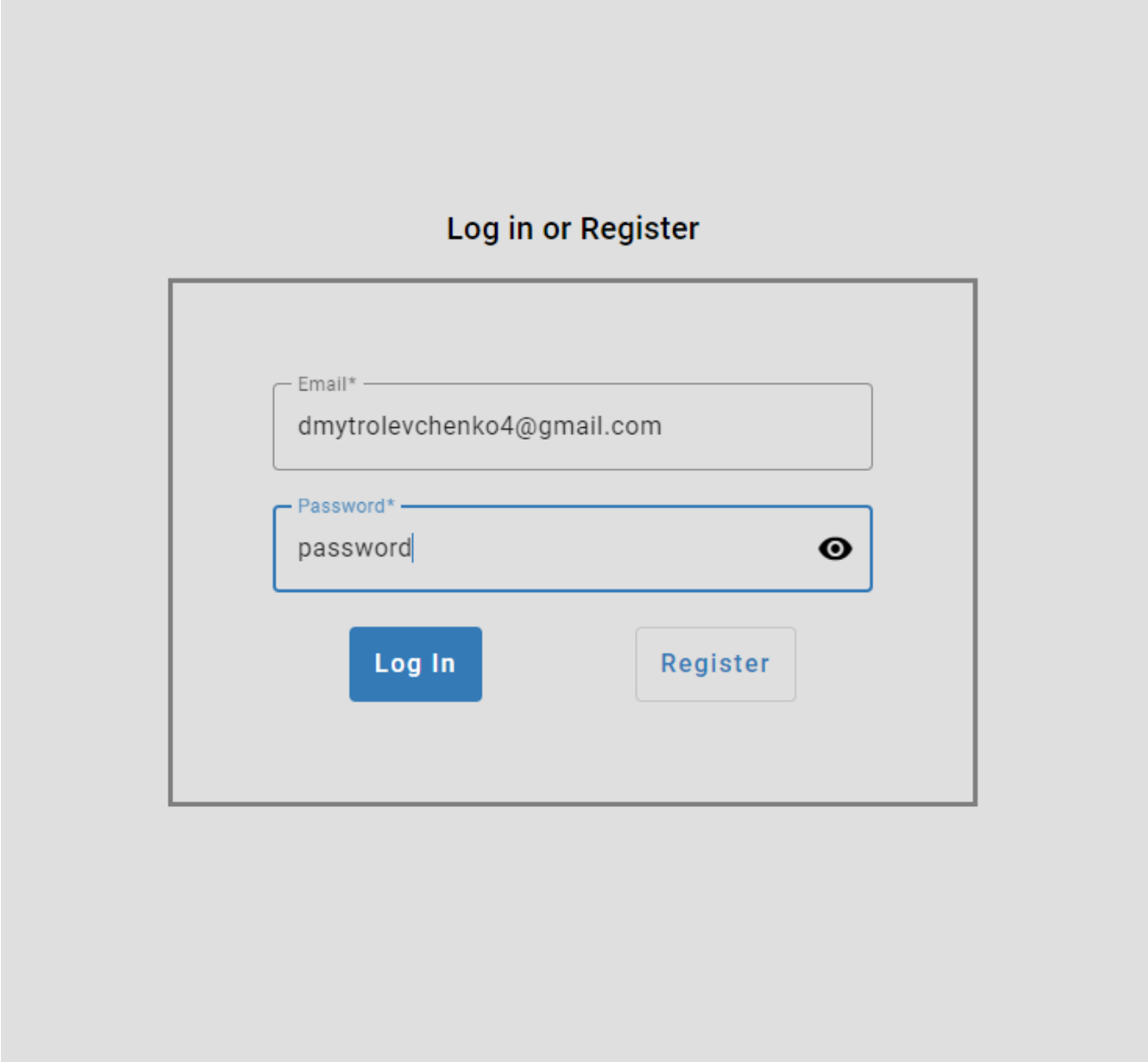

Рис. 3.19. Перемикач прозорості паролю у ввімкненому положенні

Також у застосунку є аналог Toolbar який замінює собою header частину, на якому ми можемо бачити логотип системи, який собою також являє і посилання на головну сторінку з таблицею. Показаний цей елемент на рис. 3.20.

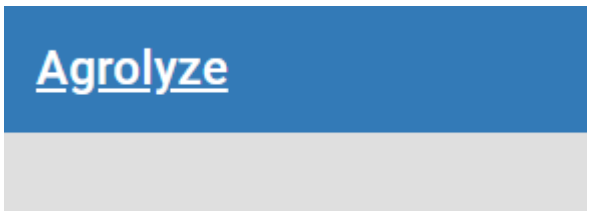

Рис. 3.20. Header логотип-посилання

3.4 Критичний аналіз розробки та напрями подальших досліджень

Розроблений застосунок Agrolyze відрізняється за функціоналом і призначенням від прикладів WEB-сайтів агрокомпаній, наведених у пункті 1.3. В той час як наведені у прикладах WEB-сайти компаній існують в більшості для того щоб нести інформаційне призначення для широкого користувача Інтернету, розповідаючи інформацію про компанію, рекламуючи її та її послуги.

Розроблений же застосунок є іншою формою WEB-додатків, Agrolyze призначений для того аби бути закритою корпоративною системою для конфіденційної комерційної діяльності, яка не призначена для використання третіми особами бо має чітку валідацію та захист за допомогою аутентифікації та системи корпоративних користувачів.

З варіантів вдосконалення системи та напрямку подальшого дослідження особливо хотілося б виділити організацію ролей із різними правами доступу для авторизованих користувачів системи, а також розгалуження моніторингових даних по різних таблицях за типом даних, або датою поточного виробничого кварталу. Гарним рішенням буде створення фільтрації даних у таблиці за допомогою окремої реактивної форми фільтра, який можна буде застосовувати до таблиці даних задля відображення записів із конкретними властивостями.

3.5 Висновки

У даному розділі було проведено проектування та розробку системи з моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів Agrolyze, в якій було використано багато технологій, які зараз є популярними на ринку WEB-застосунків.

В процесі розробки було підключено Angular та Google FireBase та використано методи сервісу Angular Material для роботи з frontend. Основний акцент було побудовано на забезпеченні швидкої, відмовостійкої, гнучкої та надійної системи. Для зміни стану записів користувачу є доступним зручний

інтуїтивно-зрозумілий інтерфейс, який без знання мови SQL допоможе створити, змінити або видалити потрібні моніторингові записи. За допомогою асинхронного програмування додано можливість динамічного відтворення деяких функцій та елементів UI.

Проведено ретельне тестування додатку, в ході якого не було знайдено очевидних можливостей зламати логіку розробленої системи. Вся система зберігає можливість масштабування і додавання нової бізнес-логіки в найкоротші строки та з мінімальними витратами.

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи є розроблений веб-застосунок для моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів «Agrolyze», що містить адміністративну та серверну частини.

У цій роботі створено надійну архітектуру та її сучасну реалізацію на основі популярного та ефективного фреймворку Angular. Клієнтська частина застосунку дозволяє без роботи з кодом виконувати всі CRUD-операції з даними для моніторингу. За допомогою асинхронного програмування додано можливість динамічного відтворення деяких функцій та елементів UI. Адміністративна частина є інтуїтивно-зрозумілою, спеціальної підготовки для користування системою не потребується.

Для функціонування додатку була створена і початково наповнена Google Firestore база даних, що надає високу оперативність за рахунок noSQL типу та постійного функціонування на хмарній платформі.

Розроблений додаток допоможе підвищити ефективність та швидкість моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів. Розроблена основа додатку є багатофункціональною, а отже отримані результати можуть застосовуватися як у прямій практиці розробки програмного забезпечення зі зміною бізнес-логіки, так і поглиблення рівня самостійного опанування галузі. Для розробки застосунку використовувались актуальні технології програмування: Angular, Angular Material, Rest API, Google Firebase, Firestore, Fireauth, HTML5, CSS3, SCSS та TypeScript.

Створений проект Agrolyze спрямований на підвищення зручності моніторингу ґрунтокліматичних параметрів агротехнічних об’єктів та зменшення дій для внесення змін в об'єкти моніторингу. Отже, він буде корисним для підтримки моніторингу в більшості корпоративних систем. Таким чином, в процесі виконання кваліфікаційної роботи було досягнуто всіх цілей та виконано усі вимоги, які висувалися при проектуванні інформаційної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statcounter URL: https://blog.statcounter.com (дата звернення: 01.10.2023).

2. Програмуваня по-українськи URL: programming.in.ua (дата звернення: 25.09.2023).

3. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення : ДСТУ 3008-95. – Чинний від 1996–01–01. – К. : Держстандарт України, 1996. – 39 с.

4. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги : ДСТУ 3582-97. – Чинний від 1998–07–01. – К. : Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

5. Михайлов С., Кулі Л. Основи дизайну. - К .: Нове знання, 1999. 106с.

6. Загрози інформаційної безпеки [Електронний ресурс] / URL: https://uk.wikipedia.org/wiki/Загрози_інформаційної_безпеки (дата звернення: 06.11.2023).

7. Статичні та динамічні web-сайти/ URL: https://armedsoft.com/ua/blog/statychni-ta-dynamichni-web-sayty (дата звернення: 11.11.2023).

8. Botha, J., Bothma, C., Geldenhuys, P. Managing E-commerce in Business. Cape Town: Juta and Company Ltd. – 2005. – p. 3.

9. Todd Fredrich. REST API Tutorial. – 2012. – p. 13-15.

10. JavaScript — Dynamic client-side scripting. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/ (дата звернення: 25.11.2023).

11. Building RESTful Web Services / URL: https://www.tutorialspoint.com/spring_boot/spring_boot_building_restful_web_seservic.htm (дата звернення: 08.10.2022).

12. Benjamin Evans, Martijn Verburg, Jason Clark. The Well-Grounded Java Developer, Second Edition – Manning, 2022. – 74 с.

13. Neal Ford, Mark Richards, Pramod Sadalage, Zhamak Dehghani. Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures – O'Reilly Media, 2021. – 459 c.

14. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Grady Booch. Design Patterns: Elements of Reusable Object-Oriented Software – Addison-Wesley Professional, 1994. – 416 c.

15. Jim Blandy, Jason Orendorff, Leonora Tindall. Programming Rust: Fast, Safe Systems Development – O'Reilly Media, 2021. – 735 c.

16. Martin Fowler. Refactoring: Improving the Design of Existing Code (2nd Edition) – Addison-Wesley Professional, 2018. – 448 c.

17. Mark Richards. Fundamentals of Software Architecture: An Engineering Approach – O'Reilly Media, 2020. – 419 c.

18. Eric Freeman. Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software 2nd Edition – O'Reilly Media, 2021. – 669 c.

19. Sam Newman. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith – O'Reilly Media, 2019. – 272 c.

20. Michael Feathers. Working Effectively with Legacy Code – Pearson, 2004. – 464 c.

21. Titus Winters, Tom Manshreck, Hyrum Wright. Software Engineering at Google: Lessons Learned from Programming Over Time – O'Reilly Media, 2020. – 599 c.

22. Merih Taze. Engineers Survival Guide: Advice, tactics, and tricks After a decade of working at Facebook, Snapchat, and Microsoft –Taze, 2021. – 245 c.

23. Sam Newman. Building Microservices: Designing Fine-Grained Systems – O'Reilly Media, 2021. – 612 c.

24. Н. Тверезовська. Інформаційні технології в агрономії. Центр навчальної літератури, 2019. 282с.

25. В.П. Горьовий, С.В. Тимчук. Менеджмент фермерських господарств. Центр учбової літератури, 2022. 336с.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Apps/Admin/src/app/app-component.html

```
<mat-toolbar color="primary">
  <span class="home-link"><a routerLink="home">Agrolyze</a></span>
  <span>
      <span class="filler"></span>
      <span class="logout-wrapper">
          <span *ngIf="authService.uid | async as uid" class="user-
name"> Welcome </span>
          <button mat-icon-button (click)="logOut()">
              <mat-icon>logout</mat-icon>
          </button>
      </span>
  </span>
</mat-toolbar>

<div class="content">
  <router-outlet></router-outlet>
</div>
```

Apps/Admin/src/app/app-component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from './services/auth.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
```

```
})
export class AppComponent {
  title = 'agrolyze';

  constructor(
    private readonly router: Router,
    public authService: AuthService
  ) {

  }

  public logOut() {
    this.authService.logout();
    this.router.navigateByUrl('sign-in');
  }
}
```

Apps/Admin/src/app/app-module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { MaterialModules } from './material-module';
import { BrowserAnimationsModule } from '@angular/platform-
browser/animations';
import { provideFirebaseApp, getApp, initializeApp } from
'@angular/fire/app';
import { getFirestore, provideFirestore } from
'@angular/fire/firestore';
import { AngularFireAuthModule } from '@angular/fire/compat/auth';
import { AngularFireModule } from '@angular/fire/compat';
import { FIREBASE_OPTIONS } from '@angular/fire/compat';
```

```
import { NotesListComponent } from './components/notes-list/notes-list.component';
import { HomeComponent } from './components/home/home.component';
import { FilterComponent } from './components/filter/filter.component';
import { NotesService } from './services/notes.service';
import { NewNoteComponent } from './components/new-note/new-note.component';
import { SignInComponent } from './components/sign-in/sign-in.component';
import { RegistrationComponent } from './components/registration/registration.component';
import { AuthService } from './services/auth.service';
import { DeleteDialogComponent } from './components/delete-dialog/delete-dialog.component';


const firebaseConfig = {
  apiKey: "AIzaSyCIPE0Ap5iwcckTkqwnfNaK2d7ikbtJeFc",
  authDomain: "agrolyze.firebaseapp.com",
  projectId: "agrolyze",
  storageBucket: "agrolyze.appspot.com",
  messagingSenderId: "1049359055230",
  appId: "1:1049359055230:web:5eb1042b8087904edd5b2f"
};

@NgModule({
  declarations: [
    AppComponent,
    NotesListComponent,
    HomeComponent,
    FilterComponent,
    NewNoteComponent,
    SignInComponent,
```

```
    RegistrationComponent,
    DeleteDialogComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    MaterialModules,
    AngularFireModule.initializeApp(firebaseConfig),
    provideFirebaseApp(() => initializeApp(firebaseConfig)),
    provideFirestore(() => getFirestore()),
    AngularFireAuthModule,
  ],
  providers: [NotesService, AuthService,
    { provide: FIREBASE_OPTIONS, useValue: firebaseConfig }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Apps/Admin/src/app/components/delete-dialog.component.html

```
<h2>Do you really want to delete this note?</h2>
<div class="dialog-content">
    <div class="dialog-actions">
        <button       mat-flat-button       (click)="DeleteNote()"
color="primary">Yes</button>
        <button         mat-flat-button         [mat-dialog-close]
color="warn">No</button>
    </div>
</div>
```

Apps/Admin/src/app/components/delete-dialog.component.ts

```
import { Component, Inject, OnInit } from '@angular/core';
import { MAT_DIALOG_DATA, MatDialogRef } from '@angular/material/dialog';
import { NotesService } from 'src/app/services/notes.service';

@Component({
  selector: 'app-delete-dialog',
  templateUrl: './delete-dialog.component.html',
  styleUrls: ['./delete-dialog.component.scss']
})
export class DeleteDialogComponent implements OnInit{

  public inputData: any;

  public delete: boolean = false;

  constructor(@Inject(MAT_DIALOG_DATA) public data: any, private ref: MatDialogRef<DeleteDialogComponent>, private notesService: NotesService ) {
  }

  ngOnInit(): void {
    this.inputData = this.data;
  }

  public DeleteNote() {
    this.delete = true;
    this.ref.close(this.delete);
  }

}
```

Apps/Admin/src/app/components/home.component.html

```
<mat-sidenav-container                          class="home-container"
[style.marginTop.px]="mobileQuery.matches ? 65 : 65">

    <mat-sidenav-content>
        <app-notes-list></app-notes-list>
    </mat-sidenav-content>
</mat-sidenav-container>
```

Apps/Admin/src/app/components/home.component.ts

```
import { MediaMatcher } from '@angular/cdk/layout';
import { ChangeDetectorRef, Component } from '@angular/core';
import { MatIconRegistry } from '@angular/material/icon';
import { DomSanitizer } from '@angular/platform-browser';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})
export class HomeComponent {

  public mobileQuery: MediaQueryList;
  public panelOpened: boolean = true;
  private _mobileQueryListener: () => void;

  constructor(
    private readonly iconRegistry: MatIconRegistry,
    private readonly sanitizer: DomSanitizer,
    private readonly changeDetectorRef: ChangeDetectorRef,
    private readonly media: MediaMatcher,
  ) {
```

```
        iconRegistry.addSvgIcon(
          'wf-icon',

sanitizer.bypassSecurityTrustResourceUrl('assets/zScripts_WF_Icon.svg'));
        this.mobileQuery = this.media.matchMedia('(max-width:
1024px)');
        this._mobileQueryListener = () =>
this.changeDetectorRef.detectChanges();
      }

      ngOnInit() {
      }
      public filterChanged(filter: any) {
        console.log(`filter changed: ${filter}`);
      }

    }
```

Apps/Admin/src/app/components/new-note.component.html

```
    <div class="form-container" *ngIf="!isLoading">
        <h1>Add Note</h1>

        <form [formGroup]="noteForm" (ngSubmit)="onSubmit()">

            <div class="formItem date">
                <mat-form-field appearance="outline">
                    <mat-label>Choose a Date</mat-label>
                    <input matInput [matDatepicker]="picker"
formControlName="date">
                    <mat-datepicker-toggle matIconSuffix
[for]="picker"></mat-datepicker-toggle>
                    <mat-datepicker #picker></mat-datepicker>
                </mat-form-field>
```

```
            </div>

            <div class="formItem temperature">
                <div class="temp-item">
                    <mat-form-field appearance="outline">
                        <mat-label>Min Temperature °C</mat-label>
                        <input          matInput          type="number"
formControlName="airTempMin">
                    </mat-form-field>
                </div>
                <div class="temp-item">
                    <mat-form-field appearance="outline">
                        <mat-label>Max Temperature °C</mat-label>
                        <input          matInput          type="number"
formControlName="airTempMax">
                    </mat-form-field>
                </div>
            </div>

            <div class="formItem specs">
                <div class="spec-item">
                    <mat-form-field appearance="outline">
                        <mat-label>Related Humidity %</mat-label>
                        <input          matInput          type="number"
formControlName="relHumidity">
                    </mat-form-field>
                    <div                                  class="error"
*ngIf="noteForm?.get('relHumidity')?.hasError('min')">
                        Введіть, будь ласка, невід'ємне число.
                    </div>
                </div>
                <div class="spec-item">
                    <mat-form-field appearance="outline">
                        <mat-label>Precipitation (mm)</mat-label>
```

```
                        <input          matInput          type="number"
formControlName="precipitation">
                    </mat-form-field>
                    <div                                  class="error"
*ngIf="noteForm?.get('precipitation')?.hasError('min')">
                        Введіть, будь ласка, невід'ємне число.
                    </div>
                </div>
                <div class="spec-item">
                    <mat-form-field appearance="outline">
                        <mat-label>Leaf Wetness (min)</mat-label>
                        <input          matInput          type="number"
formControlName="leafWetness">
                    </mat-form-field>
                    <div                                  class="error"
*ngIf="noteForm?.get('leafWetness')?.hasError('min')">
                        Введіть, будь ласка, невід'ємне число.
                    </div>
                </div>
            </div>

            <div class="formItem infection">
                <div class="infection-item">
                    <mat-form-field appearance="outline">
                        <mat-label>Infection %</mat-label>
                        <input          matInput          type="number"
formControlName="infection">
                    </mat-form-field>
                    <div                                  class="error"
*ngIf="noteForm?.get('infection')?.hasError('min')">
                        Введіть, будь ласка, невід'ємне число.
                    </div>
```

```
                    <div                                    class="error"
*ngIf="noteForm?.get('infection')?.hasError('required')                &&
noteForm?.get('infection')?.touched">
                        поле  треба  заповнити.  Якщо  інфекції  немає  -
ставте нуль.
                    </div>
                </div>
            </div>

            <div class="action-buttons">
                <button  [disabled]="noteForm.invalid"  mat-flat-button
color="primary" type="submit">
                    <span>Save</span>
                </button>
                <button   mat-flat-button   color="warn"   type="button"
(click)="onCancel()">Cancel</button>
            </div>

        </form>
    </div>
```

Apps/Admin/src/app/components/new-note.component.ts

```
    import { Component, OnInit } from '@angular/core';
    import  {  FormBuilder,  FormControl,  FormGroup,  Validators  }  from
'@angular/forms';
    import { Router } from '@angular/router';
    import { NoteModel } from 'src/app/models/note.model';
    import { NotesService } from 'src/app/services/notes.service';

    @Component({
      selector: 'app-new-note',
      templateUrl: './new-note.component.html',
      styleUrls: ['./new-note.component.scss']
```

```
})
export class NewNoteComponent implements OnInit{

  public noteForm: FormGroup;
  public isLoading: boolean;
  public newNote: NoteModel = new NoteModel();

  constructor(private formBuilder: FormBuilder, private router: Router, private notesService: NotesService) {}

  ngOnInit(): void {
    this.isLoading = true;
      this.noteForm = this.formBuilder.group({
        date: new FormControl(Date, [Validators.required]),
        airTempMin: new FormControl(null, [Validators.required]),
        airTempMax: new FormControl(null, [Validators.required]),
        leafWetness: new FormControl(null, [Validators.required, Validators.min(0)]),
        relHumidity: new FormControl(null, [Validators.required, Validators.min(0)]),
        precipitation: new FormControl(null, [Validators.required, Validators.min(0)]),
        infection: new FormControl(null, [Validators.required, Validators.min(0)]),
      })
    this.isLoading = false;
  }

  public onSubmit() {
    if(this.noteForm.valid) {
      this.newNote = {
        date: this.noteForm.get('date')?.value,
        airTempMin: this.noteForm.get('airTempMin')?.value,
        airTempMax: this.noteForm.get('airTempMax')?.value,
```

```
            airTempAvg:    (this.noteForm.get('airTempMin')?.value    +
this.noteForm.get('airTempMax')?.value)/2,
            relHumidity: this.noteForm.get('relHumidity')?.value,
            precipitation: this.noteForm.get('precipitation')?.value,
            leafWetness: this.noteForm.get('leafWetness')?.value,
            infection: this.noteForm.get('infection')?.value,
            isInfected:  this.noteForm.get('infection')?.value  >  0  ?
true : false
          };

          this.notesService.postNote(this.newNote).then((res) => {
            this.router.navigate(['/home']);
          })
        } else {
          console.log('form is not valid');
        }
      }

      public onCancel() {
        this.router.navigate(['home']);
      }
    }
```

Apps/Admin/src/app/components/new-note.component.scss

```
.form-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    width: 100%;

    h1 {
        margin: 36px 0 64px 0;
```

```
    font-size: 32px;
    font-weight: 500;
}

form {
    width: 30%;

    .formItem {
        margin: 8px 0 8px 0;

    }

    .date {
        display: flex;
        flex-direction: row;
        justify-content: center;
    }

    .temperature {
        display: flex;
        flex-direction: row;
        justify-content: space-around;
    }

    .specs {
        display: flex;
        flex-direction: row;
        justify-content: space-between;
    }

    .infection {
        display: flex;
        flex-direction: row;
        justify-content: center;
```

```
            }

            .action-buttons {
                margin: 64px 0 0 0;
                display: flex;
                justify-content: space-around;

                button {
                    min-width: 164px;
                    min-height: 64px;
                    font-size: 32px;
                    font-weight: 400;
                }
            }

            .error {
                color: red;
                max-width: 164px;
            }
        }
    }
```

Apps/Admin/src/app/components/notes-list.component.html

```
    <div class="navbar">
        <button           mat-flat-button           class="addButton"
(click)="navigateToAdd()">
            Add new note<mat-icon>add</mat-icon>
        </button>
    </div>
    <div class="table-wrapper">
        <table mat-table [dataSource]="dataSource" matSort>
            <ng-container matColumnDef="date">
```

```
                <th mat-header-cell mat-sort-header *matHeaderCellDef>
Date </th>
                <td [ngClass]="{'infected': element.isInfected}" mat-
cell *matCellDef="let element">{{element.date | date:"short"}}</td>
            </ng-container>
            <ng-container matColumnDef="airTempMin">
                <th mat-header-cell mat-sort-header *matHeaderCellDef>
Min Air Temp °C </th>
                <td [ngClass]="{'infected': element.isInfected}" mat-
cell *matCellDef="let element">{{element.airTempMin}}</td>
            </ng-container>
            <ng-container matColumnDef="airTempMax">
                <th mat-header-cell mat-sort-header *matHeaderCellDef>
Max Air Temp °C </th>
                <td [ngClass]="{'infected': element.isInfected}" mat-
cell *matCellDef="let element">{{element.airTempMax}}</td>
            </ng-container>
            <ng-container matColumnDef="airTempAvg">
                <th mat-header-cell mat-sort-header *matHeaderCellDef>
Average Air Temp °C </th>
                <td [ngClass]="{'infected': element.isInfected}" mat-
cell *matCellDef="let element">{{element.airTempAvg}}</td>
            </ng-container>
            <ng-container matColumnDef="relHumidity">
                <th mat-header-cell mat-sort-header *matHeaderCellDef>
Related Humidity % </th>
                <td [ngClass]="{'infected': element.isInfected}" mat-
cell *matCellDef="let element">{{element.relHumidity}}</td>
            </ng-container>
            <ng-container matColumnDef="precipitation">
                <th mat-header-cell mat-sort-header *matHeaderCellDef>
Precipitation (mm) </th>
                <td [ngClass]="{'infected': element.isInfected}" mat-
cell *matCellDef="let element">{{element.precipitation}}</td>
```

```
            </ng-container>
            <ng-container matColumnDef="leafWetness">
                <th mat-header-cell mat-sort-header *matHeaderCellDef>
Leaf Wetness (min) </th>
                <td [ngClass]="{'infected': element.isInfected}" mat-
cell *matCellDef="let element">{{element.leafWetness}}</td>
            </ng-container>
            <ng-container matColumnDef="infection">
                <th mat-header-cell mat-sort-header *matHeaderCellDef>
Infection % </th>
                <td [ngClass]="{'infected': element.isInfected}" mat-
cell *matCellDef="let element">{{element.infection}}</td>
            </ng-container>

            <ng-container matColumnDef="action">
                <th mat-header-cell *matHeaderCellDef></th>
                <td [ngClass]="{'infected': element.isInfected}" mat-
cell *matCellDef="let element">
                    <button                              mat-icon-button
(click)="deleteNote(element.id)">
                        <mat-icon fontIcon="delete"></mat-icon>
                    </button>
                </td>
            </ng-container>


            <tr   mat-header-row   *matHeaderRowDef="displayedColumns;
sticky: true"></tr>
            <tr     mat-row     *matRowDef="let     row;     columns:
displayedColumns;"></tr>
        </table>


    </div>
```

```
<mat-paginator [pageSizeOptions]="pageSizeOptions" showFirstLastButtons></mat-paginator>
```

Apps/Admin/src/app/components/notes-list.component.ts

```
import { Component, ViewChild, AfterViewInit, OnInit, } from '@angular/core';
import { MatDialog } from '@angular/material/dialog';
import { MatPaginator } from '@angular/material/paginator';
import { MatSort } from '@angular/material/sort';
import { MatTableDataSource } from '@angular/material/table';
import { Router } from '@angular/router';
import { map } from 'rxjs';
import { NoteModel } from 'src/app/models/note.model';
import { NotesService } from 'src/app/services/notes.service';
import { DeleteDialogComponent } from '../delete-dialog/delete-dialog.component';

@Component({
  selector: 'app-notes-list',
  templateUrl: './notes-list.component.html',
  styleUrls: ['./notes-list.component.scss']
})
export class NotesListComponent implements AfterViewInit, OnInit {

  @ViewChild(MatPaginator) paginator: MatPaginator;
  @ViewChild(MatSort) sort: MatSort;

  public totalRows: number = 0;
  public pageSize: number = 10;
  public currentPage: number = 0;
  public loading: boolean = false;
  public pageSizeOptions: number[] = [10, 25, 50];
```

```
        public dataSource: MatTableDataSource<NoteModel> = new
MatTableDataSource<NoteModel>();
        public displayedColumns: string[] = [];

        constructor(private notesService: NotesService, private router:
Router, private dialogRef: MatDialog) {
          this.displayedColumns =
["date","airTempMin","airTempMax","airTempAvg","relHumidity","precipitati
on","leafWetness","infection", "action"];
        }

        ngOnInit() {
          this.loadNotes();
        }

        ngAfterViewInit() {
          this.dataSource.sort = this.sort;
          this.dataSource.paginator = this.paginator;
        }

        public loadNotes() {
          this.notesService.getNotes().pipe(
            map((res: any[]) => {
              return res.map(item => {
                const noteModel: NoteModel = {
                  date: item.Date.toDate(),
                  airTempAvg: item.AirTempAvg,
                  airTempMax: item.AirTempMax,
                  airTempMin: item.AirTempMin,
                  relHumidity: item.RelHumidity,
                  precipitation: item.Precipitation,
                  leafWetness: item.LeafWetness,
                  infection: item.Infection,
                  isInfected: item.IsInfected,
```

```
          id: item.id,
        };
        return noteModel;
      });
    })
  ).subscribe((transformedData: NoteModel[]) => {
    this.dataSource.data = transformedData;
  });
}

public deleteNote(id: string) {
  var _dialog = this.dialogRef.open(DeleteDialogComponent, {
    width: '30%',
    height: '12%',
    enterAnimationDuration: '250ms',
    exitAnimationDuration: '250ms',
    disableClose: true,
    data: {
      id: id,
    }
  });
  _dialog.afterClosed().subscribe(item => {
    if(item) {
      this.notesService.deleteNote(id).then((res) => {
        console.log(res);
        this.loadNotes();
      })
    } else {

    }

  });
}
```

```
  public navigateToAdd() {
    this.router.navigate(['new']);
  }
}
```

Apps/Admin/src/app/components/notes-list.component.scss

```
:host {
    display: flex;
    flex: 1;
    width: 100%;
    height: calc(100% - 50px);
    flex-direction: column;
}

.table-wrapper{

    flex: 1;
    position: relative;
    min-height: 300px;
    max-height: calc(100vh - 190px);
    overflow: auto;

    table{
        font-size: 14px;
        width: 100%;

        .infected {
            background-color: rgb(255, 172, 172);
        }

        tr {

            :hover {
```

```
                cursor: pointer;
            }
        }

    }

    mat-paginator{

    }
}

.navbar {
    width: 100%;
    display: flex;
    flex-direction: row;
    justify-content: center;
    .addButton {
        margin: 0 5% 4px 0;
        background-color: rgb(127, 208, 255);
        display: flex;
        flex-direction: row-reverse;
        align-items: center;

        mat-icon{
            margin-left: 1px;
        }
    }
}
```

Apps/Admin/src/app/components/registration.component.html

```
<div class="registration-content">
    <h2>Register a new user</h2>
    <div class="form-container">
        <form [formGroup]="regForm">
            <div class="form-item">
                <mat-form-field appearance="outline">
                    <mat-label>Email</mat-label>
                    <input matInput type="email" formControlName="email">

                    <mat-error *ngIf="regForm.controls['email'].hasError('required')">Email is required</mat-error>
                    <mat-error *ngIf="regForm.controls['email'].hasError('email')">Invalid Email</mat-error>
                </mat-form-field>
            </div>
            <div class="form-item">
                <mat-form-field appearance="outline">
                    <mat-label>Password</mat-label>
                    <input matInput [type]="hide ? 'password':'text'" formControlName="password">
                    <mat-icon (click)="hide = !hide" matSuffix>{{ hide ? 'visibility_off' : 'visibility'}}</mat-icon>

                    <mat-error *ngIf="regForm.controls['password'].hasError('required')">Password is required</mat-error>
                </mat-form-field>
            </div>
```

```
                <div class="action-buttons">
                    <button   [disabled]="regForm.invalid"   mat-flat-
button color="primary" (click)="register()">
                        <span>Sign Up</span>
                    </button>
                    <button mat-flat-button color="warn" type="button"
(click)="onCancel()">Cancel</button>
                </div>
            </form>
        </div>
    </div>
```

Apps/Admin/src/app/components/registration.component.ts

```
    import { Component, OnInit } from '@angular/core';
    import { FormBuilder, FormControl, FormGroup, Validators } from
'@angular/forms';
    import { Router } from '@angular/router';
    import { AuthService } from 'src/app/services/auth.service';

    @Component({
      selector: 'app-registration',
      templateUrl: './registration.component.html',
      styleUrls: ['./registration.component.scss']
    })
    export class RegistrationComponent implements OnInit{

      public hide: boolean = true;
      public regForm: FormGroup;

      constructor(private formBuilder: FormBuilder, private router:
Router, private authService: AuthService) {}

      ngOnInit(): void {
```

```
      this.regForm = this.formBuilder.group({
        email:    new    FormControl(null,    [Validators.required,
Validators.email]),
        password: new FormControl(null, [Validators.required])
      })
    }

    public register() {
      const userData = Object.assign(this.regForm.value);

this.authService.registerWithEmailAndPassword(userData).then((res:   any)
=> {
        this.router.navigateByUrl('sign-in');
      }).catch((error: any) => {
        console.error(error);
      });
    }

    public onCancel() {
      this.router.navigateByUrl('sign-in');
    }

  }
```

Apps/Admin/src/app/components/registration.component.scss

```
  .registration-content {
      height: 900px;
      width:100%;
      background-color: rgb(223, 223, 223);
      display: flex;
      flex-direction: column;
      justify-content: center;
      align-items: center;
```

```
    .form-container {
        padding: 64px;
        width: 15%;
        border: 3px solid rgb(126, 126, 126);

        form {
            width: 100%;

            mat-form-field {
                width: 100%;
            }

            .action-buttons {
                display: flex;
                justify-content: space-around;

                button {
                    min-width: 82px;
                    min-height: 48px;
                    font-size: 16px;
                    font-weight: 500;
                }
            }
        }
    }
}
```

Apps/Admin/src/app/components/sign-in.component.html

```
<div class="sign-in-content">
    <h2>Log in or Register</h2>
    <div class="form-container">
        <form [formGroup]="logInForm">
```

```
                <div class="form-item">
                    <mat-form-field appearance="outline">
                        <mat-label>Email</mat-label>
                        <input          matInput          type="email"
formControlName="email">

                        <mat-error
*ngIf="logInForm.controls['email'].hasError('required')">Email         is
required</mat-error>
                        <mat-error
*ngIf="logInForm.controls['email'].hasError('email')">Invalid Email</mat-
error>
                    </mat-form-field>
                </div>
                <div class="form-item">
                    <mat-form-field appearance="outline">
                        <mat-label>Password</mat-label>
                        <input      matInput      [type]="hide      ?
'password':'text'" formControlName="password">
                        <mat-icon (click)="hide = !hide" matSuffix>{{
hide ? 'visibility_off' : 'visibility'}}</mat-icon>

                        <mat-error
*ngIf="logInForm.controls['password'].hasError('required')">Password   is
required</mat-error>
                    </mat-form-field>
                </div>

                <div class="action-buttons">
                    <button  [disabled]="logInForm.invalid"  mat-flat-
button color="primary" (click)="logIn()">
                        <span>Log In</span>
                    </button>
```

```
                    <button     mat-stroked-button     color="primary"
type="button" (click)="register()">Register</button>
                </div>
            </form>
        </div>
    </div>
```

Apps/Admin/src/app/components/sign-in.component.ts

```
    import { Component, OnInit } from '@angular/core';
    import { FormBuilder, FormControl, FormGroup, Validators } from
'@angular/forms';
    import { Router } from '@angular/router';
    import { AuthService } from 'src/app/services/auth.service';

    @Component({
      selector: 'app-sign-in',
      templateUrl: './sign-in.component.html',
      styleUrls: ['./sign-in.component.scss']
    })
    export class SignInComponent implements OnInit{

      public logInForm: FormGroup;
      public hide: boolean = true;

      constructor(private formBuilder: FormBuilder, private router:
Router, private authService: AuthService) {}

      ngOnInit(): void {
        this.logInForm = this.formBuilder.group({
          email:    new    FormControl(null,    [Validators.required,
Validators.email]),
          password: new FormControl(null, [Validators.required])
        })
```

```
  }

  public register() {
    this.router.navigateByUrl('register');
  }

  public logIn() {
    const userData = Object.assign(this.logInForm.value);

this.authService.signInWithEmailAndPassword(userData).then((res: any) =>
{
      this.router.navigateByUrl('home');
    }).catch((error: any) => {
      console.error(error);
    });
  }


}
```

Apps/Admin/src/app/components/sign-in.component.scss

```
.sign-in-content {
    height: 900px;
    width:100%;
    background-color: rgb(223, 223, 223);
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;

    .form-container {
        padding: 64px;
        width: 15%;
```

```
        border: 3px solid rgb(126, 126, 126);

        form {
            width: 100%;

            mat-form-field {
                width: 100%;
            }

            .action-buttons {
                display: flex;
                justify-content: space-around;

                button {
                    min-width: 82px;
                    min-height: 48px;
                    font-size: 16px;
                    font-weight: 500;
                }
            }
        }
    }
}
```

Apps/Admin/src/app/models/note.model.ts

```
export class NoteModel {
    public date: Date;
    public airTempAvg: number;
    public airTempMax: number;
    public airTempMin: number;
    public relHumidity: number;
    public precipitation: number;
    public leafWetness: number;
```

```
    public infection: number;
    public isInfected: boolean | null;
    public id?: string;
}
```

Apps/Admin/src/app/services/notes.service.ts

```
import { Injectable } from '@angular/core';
import { Firestore, addDoc, doc, collection, collectionData, deleteDoc } from '@angular/fire/firestore';
import { NoteModel } from '../models/note.model';

@Injectable({
  providedIn: 'root'
})
export class NotesService {

  constructor(private fs: Firestore) { }

  public getNotes() {
    let notesCollection = collection(this.fs, 'Notes');
    return collectionData(notesCollection, {idField: 'id'});
  }

  public postNote(note: NoteModel) {
    let data = {
      Date: note.date,
      AirTempAvg: note.airTempAvg,
      AirTempMax: note.airTempMax,
      AirTempMin: note.airTempMin,
      RelHumidity: note.relHumidity,
      Precipitation: note.precipitation,
      LeafWetness: note.leafWetness,
      Infection: note.infection,
```

```
        IsInfected: note.isInfected
      };
      let notesCollection = collection(this.fs, 'Notes');
      return addDoc(notesCollection, data);
    }

    public deleteNote(id: string) {
      let docRef = doc(this.fs, 'Notes/'+id);

      return deleteDoc(docRef);
    }
  }
```

Apps/Admin/src/app/services/auth.service.ts

```
  import { Injectable } from '@angular/core';
  import { Router } from '@angular/router';
  import { Observable, of as ObservableOf } from 'rxjs';
  import { switchMap, map } from 'rxjs/operators';
  import { GoogleAuthProvider, User } from '@angular/fire/auth';
  import { AngularFireAuth } from '@angular/fire/compat/auth';
  import { Firestore, collection, doc } from
'@angular/fire/firestore';

  @Injectable({
    providedIn: 'root'
  })

  export class AuthService {

    public uid = this.afAuth.authState.pipe(
      map(authState => {
        if(!authState) {
          return null;
```

```
            } else {
              return authState.uid;
            };
          })
        );

        public isAdmin: Observable<boolean> = this.uid.pipe(
          switchMap(uid => {
            if(!uid) {
              return ObservableOf(false);
            } else {
              let docRef = doc(this.fs, 'Admins/'+uid);
              console.log(docRef.id);
              console.log(docRef.id != uid);
              if(docRef.id != uid) {
                return ObservableOf(false);
              } else {
                return ObservableOf(true);
              }
            }
          }),map(value => !!value)
        );

        constructor(private afAuth: AngularFireAuth, private fs:
Firestore) {}

        registerWithEmailAndPassword(user: {email: string, password:
string}) {
          return this.afAuth.createUserWithEmailAndPassword(user.email,
user.password);
        }

        signInWithEmailAndPassword(user: {email: string, password:
string}) {
```

```
        return      this.afAuth.signInWithEmailAndPassword(user.email,
user.password);
      }

      logout() {
        return this.afAuth.signOut();
      }
    }
```

Apps/Admin/src/app/app-routing.module.ts

```
    import { NgModule } from '@angular/core';
    import { RouterModule, Routes } from '@angular/router';
    import { NotesListComponent } from './components/notes-list/notes-
list.component';
    import { HomeComponent } from './components/home/home.component';
    import  {  NewNoteComponent  }  from  './components/new-note/new-
note.component';
    import  {  SignInComponent  }  from  './components/sign-in/sign-
in.component';
    import        {        RegistrationComponent        }        from
'./components/registration/registration.component';
    import  {  AngularFireAuthGuard  }  from  '@angular/fire/compat/auth-
guard';

    const routes: Routes = [
      {path: 'sign-in', component: SignInComponent},
      {path: 'register', component: RegistrationComponent},
      {   path:   'home',   component:   HomeComponent,   canActivate:
[AngularFireAuthGuard]},
      {path:   'new',   component:   NewNoteComponent,   canActivate:
[AngularFireAuthGuard]},
      { path: '**', redirectTo: '/sign-in', pathMatch: 'full' },
    ];
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій
Кафедра програмного забезпечення комп'ютерних систем

ВІДГУК

Наукового керівника _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

на магістерську роботу

студента Левченко Дмитра Максимовича
(прізвище, ім'я, по батькові)

курсу II групи 121м-22-2
спеціальності 122 Комп'ютерні науки
освітньої програми «122 Комп'ютерні науки»
на тему Розробка програмного забезпечення для автоматизації зміни стану ресурсів тестування

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
Програма	
Презентація	