Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)
Програмна система для автоматизованого управління освітленням для вирощування рослин. Фронт-енд
(тема)

Виконав:
здобувач 4 року навчання,
групи ПЗПІ-21-10

Богдан ІЛЬЄНКО
(власне ім'я, прізвище)

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

Керівник к.т.н., доц. кафедри ПІ
Дмитро КОЛЕСНИКОВ
(посада, власне ім'я, прізвище)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (власне імя, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Ільєнку Богдану Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для автоматизованого управління освітленням для вирощування рослин. Фронт-енд
Затверджена наказом по університету від 19.05.2025р. № 397Ст
2. Термін подання студентом роботи до екзаменаційної комісії 10.06.2025
3. Вихідні дані до роботи Розробити серверну частину програмної системи для автоматизованого управління освітленням для вирощування рослин, що допоможе фермерам вирощувати рослини і змінювати параметри відповідно до умов вирощування рослини.
4. Перелік питань, що потрібно опрацювати в роботі Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	09.04.2025	*виконано*
2	Проектування ПЗ	14.04.2025	*виконано*
3	Розробка ПЗ	29.04.2025	*виконано*
4	Тестування ПЗ	12.05.2025	*виконано*
5	Оформлення пояснювальної записки	20.05.2025	*виконано*
6	Підготовка презентації та доповіді	27.05.2025	*виконано*
7	Нормоконтроль, рецензування	01.06.2025	*виконано*
9	Здача роботи у електронний архів	04.06.2025	*виконано*
9	Допуск до захисту у зав. кафедри	07.06.2025	*виконано*
10	Попередній захист	08.06.2025	*виконано*

Дата видачі завдання 9 квітня 2025 р.

Здобувач _____

(підпис)

Керівник роботи _____ к.т.н., доц. кафедри ПІ Дмитро КОЛЕСНИКОВ

(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 59 сторінок, 10 рисунків, 12 джерел.

АВТОМАТИЗОВАНІ СИСТЕМИ, ДИСТАНЦІЙНЕ КЕРУВАННЯ, ІНТЕЛЕКТУАЛЬНЕ КЕРУВАННЯ ОСВІТЛЕННЯМ, МОНІТОРИНГ СТАНУ РОСЛИН, РОСЛИННИЦТВО, NODE.JS, NESTJS, POSTGRESQL, REACT, MQTT.

Об'єктом дослідження виступає сільське господарство, зокрема процес вирощування рослин, і вдосконалення систем автоматичного керування освітленням на фермах і плантаціях. Традиційні підходи до штучного освітлення нерідко не відповідають реальним потребам рослин, що негативно позначається на їхньому розвитку та врожайності. Для вирішення цієї проблеми передбачено створення системи, яка автоматично регулюватиме освітлення відповідно до фаз розвитку рослин і показників середовища.

Метою роботи є проєктування та розробка програмної системи для ефективного моніторингу й управління освітленням на аграрних об'єктах. Система відповідатиме за збір і збереження інформації про стан рослин, їхні потреби в освітленні та оптимальні режими, з можливістю автоматичного регулювання параметрів світла для забезпечення сприятливих умов росту.

Програмний комплекс розроблено з використанням Node.js, NestJS та TypeORM для серверної частини (взаємодія з PostgreSQL). Клієнтська частина реалізована за допомогою React, TypeScript та RTK Query. Збір даних із сенсорного обладнання відбувається через MQTT-протокол за допомогою IoT-контролера на базі Raspberry Pi.

У підсумку буде створено повноцінну програмну систему, що забезпечить автоматизоване керування освітленням у сільському господарстві. До складу системи входитиме серверна частина для обробки і зберігання даних, клієнтський інтерфейс для роботи користувачів та IoT-пристрій для моніторингу стану рослин.

ABSTRACT

AUTOMATED SYSTEMS, REMOTE CONTROL, INTELLIGENT LIGHTING MANAGEMENT, PLANT CONDITION MONITORING, CROP PRODUCTION, NODE.JS, NESTJS, POSTGRESQL, REACT, MQTT.

The object of research is agriculture, in particular the process of plant cultivation, and the improvement of automatic lighting control systems on farms and plantations. Traditional approaches to artificial lighting often do not meet the actual needs of plants, which negatively affects their development and yield. To solve this problem, a system is proposed that will automatically adjust lighting according to plant growth phases and environmental indicators.

The aim of the work is the design and development of a software system for effective monitoring and lighting control in agricultural facilities. The system will be responsible for collecting and storing information about the condition of plants, their lighting requirements, and optimal modes, with the possibility of automatically adjusting lighting parameters to ensure favorable growth conditions.

The server part is developed using Node.js, NestJS, and the TypeORM library to interact with a PostgreSQL database. The client part is implemented using the React library, TypeScript, and RTK Query for efficient data exchange. The transmission of information from real sensor equipment is based on the use of the MQTT protocol through an IoT controller based on Raspberry Pi.

As a result, a complete software system will be created that provides automated lighting control in agriculture. The system will include a server component for data processing and storage, a client interface for user interaction, and an IoT device for monitoring the condition of plants.

ЗМІСТ

ВСТУП

Основною метою даної кваліфікаційної роботи є створення програмної системи, що забезпечує автоматизацію та оптимізацію процесів керування освітленням на фермах і плантаціях. Розроблена система має серверну і клієнтську частини та реалізує наступний набір функціональних можливостей:

– Отримання, збереження і передавання інформації про рослини, сенсорні пристрої та освітлювальні елементи між сервером і клієнтом;

– Додавання нових рослин, сенсорів та джерел світла через зручний інтерфейс користувача з подальшим збереженням у базі даних;

– Інтерактивне регулювання параметрів освітлення, таких як яскравість і спектральний склад, через веб-застосунок;

– Автоматичне налаштування роботи освітлювальних пристроїв на основі актуальних даних із сенсорів у режимі реального часу;

– Механізми резервного копіювання та відновлення даних користувачів;

– Фільтрація й аналітична обробка показників сенсорів для моніторингу умов вирощування рослин.

Під час реалізації програмної системи використовувалися такі технології: Node.js [1], NestJS [2], TypeORM [3], PostgreSQL [4] та MQTT [5].

Серверна частина розроблена із застосуванням Node.js [1], NestJS [2], TypeORM [3] для роботи з PostgreSQL [4], Redis [6] для зберігання даних та MQTT-протоколу [5] для інтеграції з IoT-пристроями. Клієнтська частина створена за допомогою React [7], TypeScript [8], RTK Query [9], Redux Toolkit [10] і React Router [11] для зручної роботи з даними та навігації.

Запропонована система забезпечує ефективне керування освітленням і підвищує якість вирощування рослин завдяки автоматизації процесів та аналізу даних сенсорів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Сучасні тенденції в аграрному секторі свідчать про зростаючу потребу у впровадженні інтелектуальних систем управління освітленням у процесах вирощування культур у контрольованому середовищі. Застосування традиційних методів, при яких освітлення працює за сталим графіком або вручну, без врахування реальних біологічних потреб рослин, часто не дає очікуваного результату. Подібні підходи не забезпечують адаптації до специфіки різних видів культур, що може призводити до зниження врожайності, уповільнення росту та перевитрати енергоресурсів.

У зв’язку з цим виникає необхідність у розробці адаптивної системи, здатної в автоматичному режимі змінювати інтенсивність і спектр освітлення відповідно до стадії розвитку рослин, рівня освітленості, вологості та інших важливих параметрів середовища. Подібне рішення має бути орієнтоване не лише на покращення умов вирощування, а й на підвищення ефективності аграрного виробництва загалом — з акцентом на оптимізацію ресурсів, зниження витрат і забезпечення стабільних врожаїв незалежно від зовнішніх факторів.

Реалізація такої системи дозволить зробити аграрні процеси більш передбачуваними, контрольованими та екологічно збалансованими, що є вкрай важливим в умовах змін клімату та зростання попиту на сільськогосподарську продукцію.

1.2 Аналіз конкурентів

Для того, щоб зрозуміти особливості ринку сучасних систем автоматизованого освітлення, розглянемо рішення, які пропонують провідні компанії цієї галузі.

Fluence Bioengineering [4] (див. рис. 1.1) — компанія, що пропонує інтелектуальні світлодіодні системи для комерційного агровиробництва. Вони надають можливість налаштовувати спектр світла під конкретні потреби культур, що дозволяє досягти вищих результатів у вирощуванні. Проте рішення Fluence орієнтовані на великі аграрні комплекси, що супроводжується високою вартістю обладнання та складністю впровадження для малих фермерських господарств.

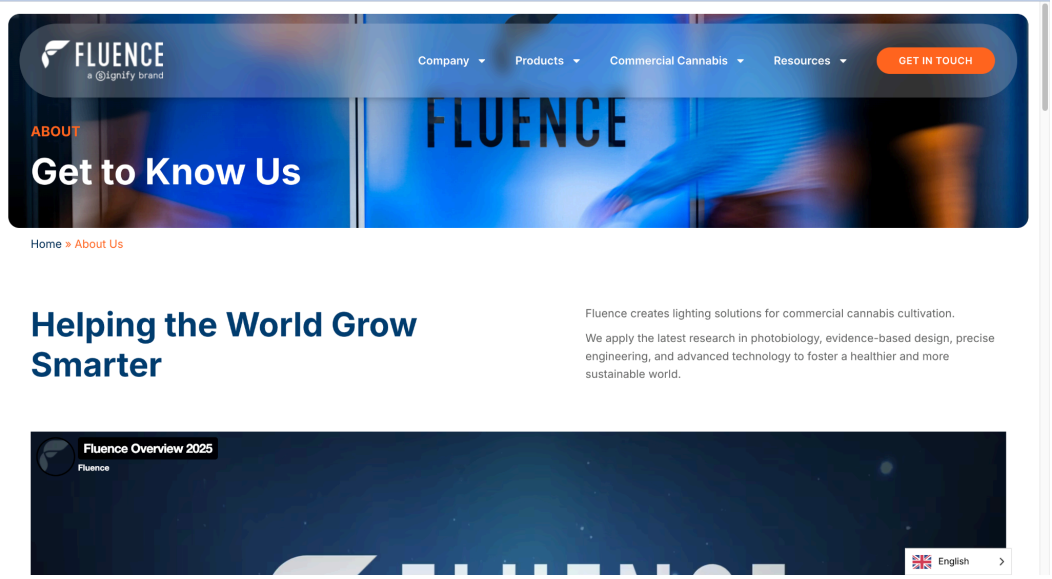

Рисунок 1.1 – Головна сторінка сайту Fluence Bioengineering [4]

Valoya [5] (див. рис. 1.2) — спеціалізується на виробництві світлодіодних ламп для теплиць і лабораторій. Системи компанії вирізняються стабільністю та високою ефективністю. Однак їхнім недоліком є обмежена інтеграція з зовнішніми сенсорними системами, відсутність автоматизації процесів та віддаленого моніторингу через централізовану платформу.

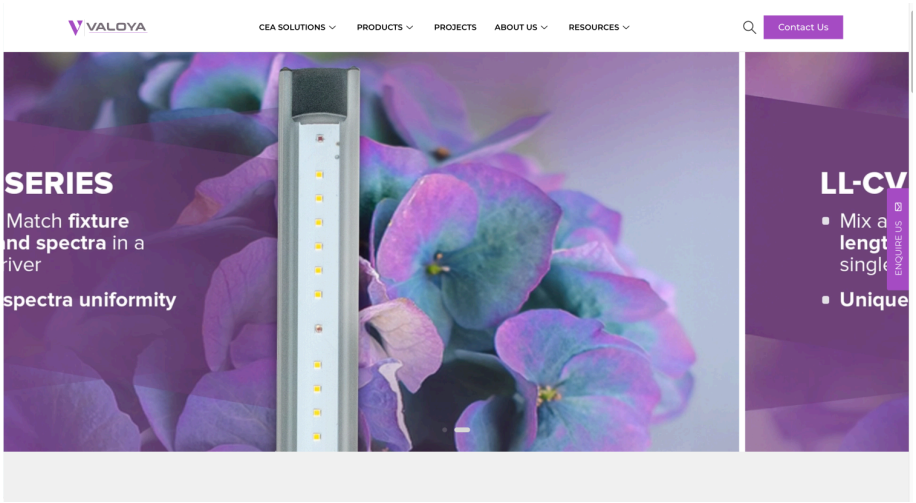

Рисунок 1.2 – Головна сторінка сайту Valoya [5]

Current (раніше GE Lighting) [6] (див. рис. 1.3) — постачає LED-освітлення для агросектора, орієнтуючись на енергоефективність. Хоча якість обладнання залишається високою, відсутність власної автоматизованої платформи для керування параметрами середовища ускладнює інтеграцію з іншими системами та потребує додаткових ресурсів для впровадження.

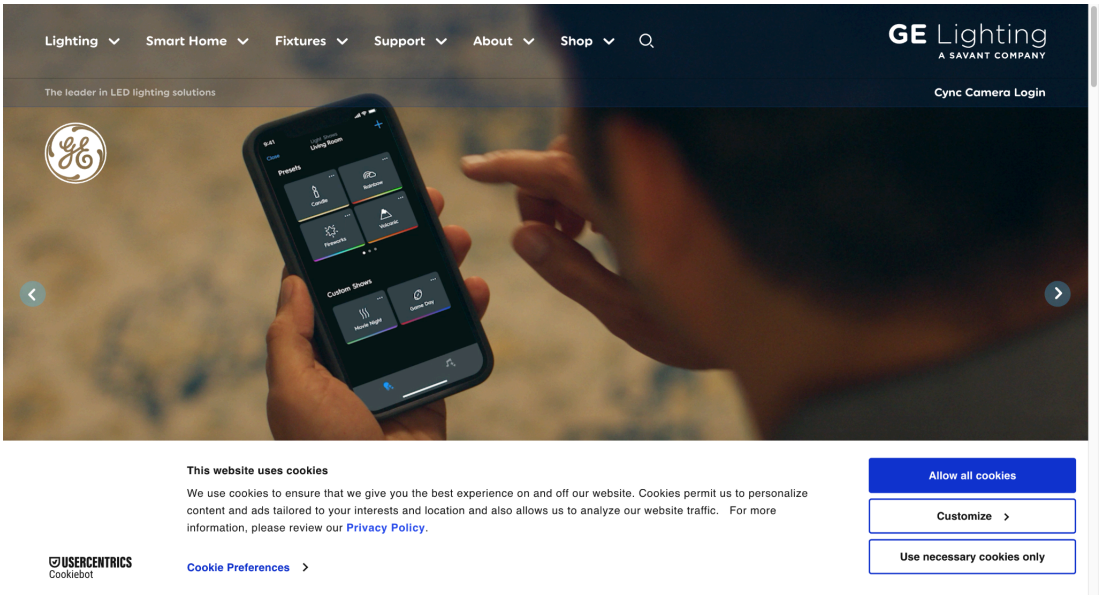

Рисунок 1.3 – Головна сторінка сайту Current [6]

Lumigrow [7] (див. рис. 1.4) — компанія, що розробляє LED-системи з функціями дистанційного керування. Вони дозволяють адаптувати освітлення відповідно до стадії розвитку культур. Попри гнучкість у налаштуваннях, рішення від Lumigrow в основному розраховані на великі агропідприємства, тому малим господарствам такі продукти є малодоступними через високу ціну й складну інтеграцію.

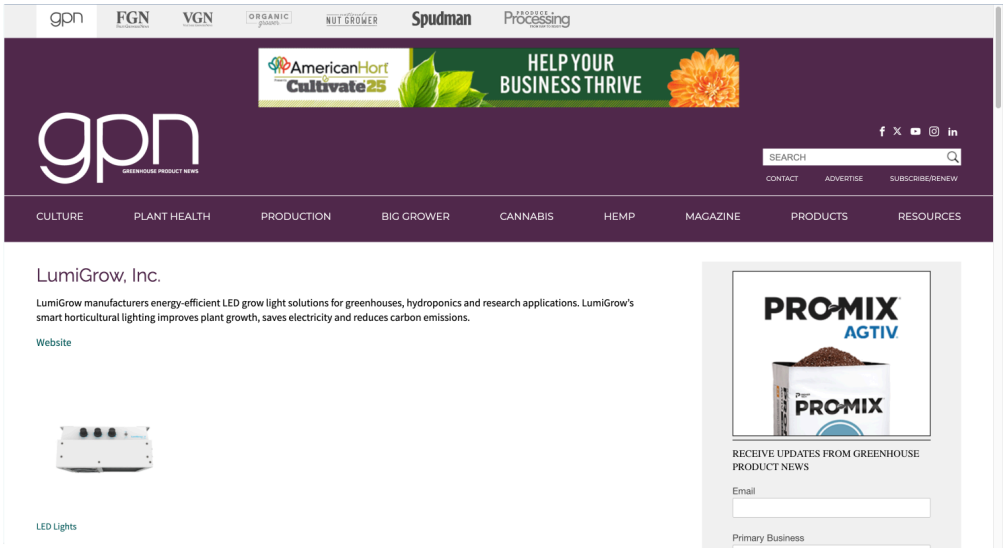

Рисунок 1.4 – Головна сторінка сайту Lumigrow [7]

Отже, проаналізуємо конкурентів, виділивши їхні переваги та недоліки (див. табл. 1.1).

Таблиця 1.1 – Порівняння аналогів (таблиця виконана самостійно)

Назва компанії	Переваги	Недоліки
Fluence Bioengineering	Висока якість освітлення; адаптація спектру під різні стадії розвитку рослин	Висока вартість впровадження; орієнтація на великі агропідприємства
Valoya	Стабільність роботи; підходить для різних типів культур	Обмежена інтеграція з автоматизованими системами; відсутність збору даних із сенсорів

Кінець таблиці 1.1

Current (раніше GE Lighting)	Енергоефективні LED-рішення; надійність обладнання	Відсутність комплексної автоматизації середовища
Lumigrow	Можливість дистанційного налаштування освітлення; оптимізація під тип рослин	Висока вартість; орієнтація на великі тепличні комплекси; складність впровадження

Після аналізу аналогів можна зробити висновок: при розробці власної системи важливо забезпечити доступність для малих господарств, просту інтеграцію сенсорів і повну автоматизацію процесів вирощування.

1.3 Можливі проблеми

У процесі створення програмної системи для автоматизованого управління освітленням на фермах та плантаціях можуть виникнути кілька важливих викликів.

Однією з головних проблем є необхідність інтеграції з різними типами сенсорних пристроїв та обладнання, що можуть працювати за несумісними протоколами або мати обмежену технічну документацію. Це ускладнює процес налаштування системи в реальних умовах господарств.

Ще однією суттєвою перепоною є нестабільне або слабке підключення до Інтернету в сільських регіонах, що може впливати на своєчасну синхронізацію даних та можливість віддаленого керування. Тому важливо передбачити можливість автономної роботи системи без постійного доступу до мережі.

Крім того, система повинна бути інтуїтивно зрозумілою для користувача, без надмірної кількості складних функцій, які можуть ускладнити її використання.

Також слід врахувати ризики, пов’язані із збереженням та передачею даних: можливі втрати показників із сенсорів або помилки при обробці інформації, що безпосередньо може вплинути на ефективність прийняття рішень.

1.4 Постановка задачі

Для розроблення цієї програмної системи необхідно вирішити низку важливих завдань.

Розробити серверну частину системи, яка відповідатиме за оброблення, зберігання даних і координацію взаємодії між усіма компонентами. Для реалізації серверної логіки буде застосовано Node.js у поєднанні з фреймворком NestJS, що дозволяє створювати масштабовані та надійні веб-сервіси.

Реалізувати API, яке забезпечить обмін даними між клієнтськими застосунками, сенсорами та сервером. Передбачається використання REST API для стандартної взаємодії, а також можливість застосування протоколів MQTT або WebSocket для оброблення даних у реальному часі в рамках IoT-рішень.

Розробити механізм авторизації користувачів із використанням імені користувача та пароля із забезпеченням захищеного зберігання облікових даних.

Організувати зберігання структурованої інформації про рослини, фази росту, типи освітлення, показники сенсорів і історію змін через базу даних PostgreSQL із використанням ORM-бібліотеки TypeORM. Окрім того, для зберігання проміжних або тимчасових даних (наприклад, сесій користувачів або оперативних показників сенсорів) передбачається використання Redis.

Реалізувати оброблення вхідних даних від сенсорів у режимі реального часу з подальшим керуванням інтенсивністю та тривалістю роботи освітлювальних пристроїв відповідно до поточних параметрів середовища.

Забезпечити можливість створення резервних копій даних користувачів і налаштувань системи, а також розробити механізми для їхнього відновлення.

Розробити клієнтську частину системи з використанням React і TypeScript, забезпечивши інтуїтивно зрозумілий та зручний інтерфейс для керування основними функціями системи. Для оптимізації обміну даними з сервером застосувати бібліотеку RTK Query.

Для організації управління залежностями, тестування та автоматизації процесів збирання застосунку буде використано інструменти екосистеми Node.js, зокрема пакетний менеджер npm.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Огляд частин програмної системи

Програмна система складатиметься з двох основних компонентів:

– Back-end частина;

– Програмний модуль інтеграції з IoT-пристроями;

– Front-end частина.

2.2 Серверна частина програмної системи

Серверна частина даної програмної системи створена з використанням мови програмування JavaScript/TypeScript, фреймворку NestJS та ORM-бібліотеки TypeORM, що реалізує об'єктно-реляційне відображення для роботи з базою даних PostgreSQL. Для написання та тестування коду застосовується сучасне інтегроване середовище розробки Visual Studio Code.

Управління залежностями, збирання проєкту та налаштування середовища виконуються за допомогою пакетного менеджера npm, що забезпечує зручність конфігурації та підтримку модулів.

NestJS надає потужні інструменти для створення масштабованих, продуктивних і стабільних серверних застосунків. Його архітектура дозволяє легко реалізовувати структуровану бізнес-логіку, організовувати модульність проєкту та впроваджувати різні протоколи взаємодії (HTTP, WebSocket, MQTT).

Завдяки використанню TypeORM робота з базою даних PostgreSQL стає більш ефективною та автоматизованою, що дозволяє мінімізувати необхідність ручної обробки запитів і полегшує зберігання інформації.

Крім того, для оптимізації обробки сесій користувачів та збереження проміжних даних використовується система кешування Redis.

Обраний набір технологій забезпечує надійну, масштабовану та зручну для супроводження серверну архітектуру, що є основою стабільного

функціонування всієї програмної системи.

2.3 IoT частина програмної системи

Для реалізації IoT-компонента буде використано мікрокомп'ютер Raspberry Pi із встановленим спеціалізованим програмним забезпеченням. Цей модуль відповідатиме за зчитування або генерацію даних від сенсорів та їхню передачу на сервер. Також він дозволить виконувати локальну обробку частини даних перед їх відправленням, що зменшить навантаження на серверну частину. Взаємодія з периферійними пристроями (сенсорами, виконавчими механізмами) здійснюватиметься через стандартні апаратні інтерфейси Raspberry Pi.

Для забезпечення надійного двонаправленого обміну даними між IoT-пристроєм та серверною частиною буде використовуватися протокол MQTT, який дозволяє ефективно працювати в умовах обмежених ресурсів і забезпечує комунікацію в реальному часі. Використання MQTT дає змогу не лише передавати дані від пристрою на сервер, а й отримувати керуючі команди від сервера на пристрій.

Програмне забезпечення для Raspberry Pi розроблятиметься із застосуванням бібліотек Node.js, що дозволяють налаштовувати роботу з мережевими протоколами та інтеграцію з серверною частиною системи.

2.4 Клієнтська частина програмної системи

Клієнтська частина програмної системи буде створена з використанням бібліотеки React і мови програмування TypeScript. Вона забезпечуватиме інтерфейс для взаємодії користувачів із функціоналом системи, відображення даних про стан рослин, сенсори та налаштування освітлення.

Для обміну даними із серверною частиною застосовуватиметься технологія RTK Query, що забезпечить ефективну роботу з REST API та оптимальне кешування запитів. Авторизація користувачів і управління

сесіями будуть реалізовані на рівні клієнтського застосунку.

Навігація між розділами здійснюватиметься через бібліотеку React Router. Для побудови зручного та сучасного інтерфейсу буде використано компонентну бібліотеку Ant Design (antd).

Особлива увага приділятиметься адаптивності інтерфейсу для коректної роботи на різних пристроях, що дозволить використовувати систему як на ПК, так і на мобільних пристроях.

2.5 Основні функції

Програмна система передбачає реалізацію наступних функціональних можливостей:

Функціонал адміністратора:

– MF-1: Створення нового облікового запису користувача;

– MF-2: Авторизація користувачів у системі;

– MF-3: Перегляд повного списку зареєстрованих користувачів;

– MF-4: Перегляд ролей, закріплених за користувачами;

– MF-5: Призначення нової ролі користувачу;

– MF-6: Скасування наданої ролі;

– MF-7: Перегляд детальної інформації про конкретного користувача;

– MF-8: Видалення облікового запису користувача;

– MF-9: Імпорт даних користувачів із JSON-файлів;

– MF-10: Експорт інформації про користувачів у JSON-форматі.

Функціонал технічного персоналу:

– MF-11: Перегляд списку всіх зареєстрованих рослин;

– MF-12: Додавання нової рослини до системи;

– MF-13: Оновлення даних існуючої рослини;

– MF-14: Видалення рослини із системи;

– MF-15: Перегляд списку сенсорів, прив'язаних до рослин;

– MF-16: Додавання нового сенсорного пристрою;

– MF-17: Видалення сенсора із системи;

– MF-18: Додавання даних, отриманих із сенсорів;

– MF-19: Перегляд переліку джерел освітлення;

– MF-20: Додавання нового джерела світла;

– MF-21: Видалення джерела освітлення;

– MF-22: Налаштування яскравості та спектру освітлення;

– MF-23: Перегляд джерел освітлення, закріплених за певним сенсором.

Функціонал працівника служби підтримки (Support):

– MF-24: Моніторинг статусу підключення сенсорів та пристроїв;

– MF-25: Перегляд журналу попереджень та помилок системи;

– MF-26: Надсилання повідомлень користувачам про виявлені проблеми;

– MF-27: Ініціювання запиту на обслуговування сенсора або пристрою.

Функціонал звичайного користувача:

– MF-28: Перегляд інформації, отриманої від сенсорів;

– MF-29: Перегляд попереджень, що стосуються конкретної рослини;

– MF-30: Отримання списку усіх доступних сенсорів у системі.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

На наступній діаграмі прецедентів буде відображено, як різні ролі користувачів взаємодіють із запропонованим функціоналом системи (див. рис. 3.1).

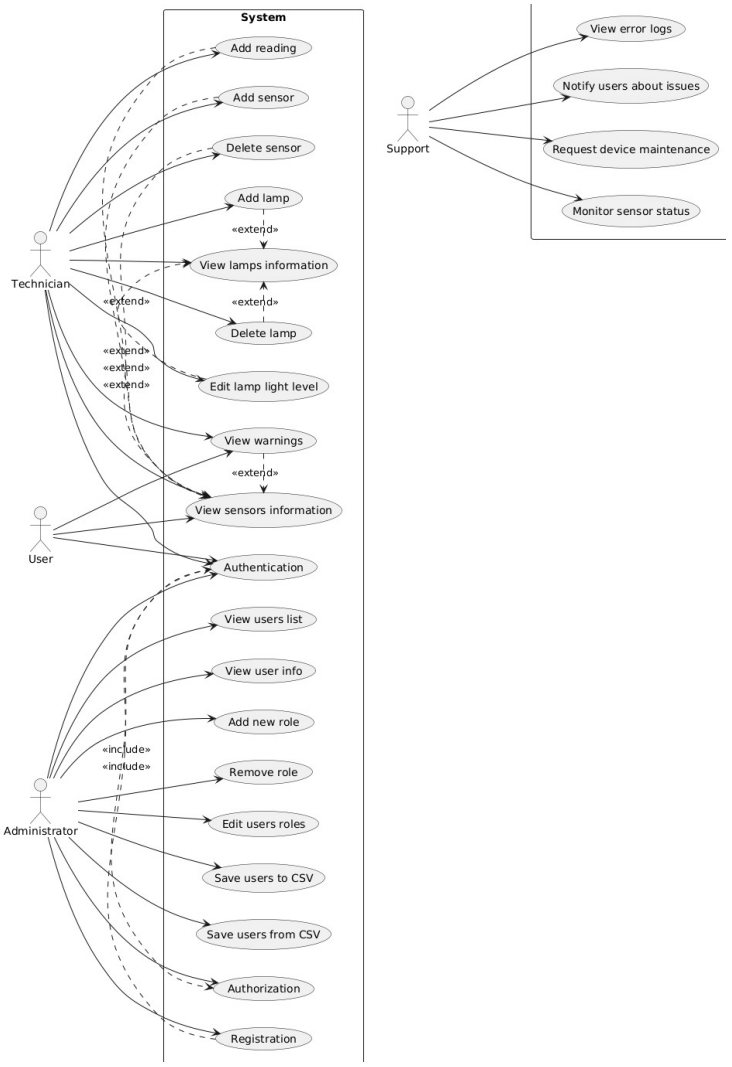

Рисунок 3.1 – Діаграма прецедентів (UML Use Case Diagram)

На діаграмі прецедентів представлені чотири основні ролі користувачів програмної системи:

– Звичайний користувач — має можливість зареєструвати обліковий запис, виконати авторизацію та переглядати список наявних сенсорів разом із даними, що надходять від них;

– Адміністратор — відповідає за адміністрування облікових записів користувачів, включаючи перегляд їх даних, зміну ролей, видалення облікових записів та імпорт/експорт інформації у форматі JSON;

– Технічний працівник — здійснює керування даними про рослини, сенсори та освітлювальні пристрої, маючи можливість додавати нові об'єкти до системи, редагувати існуючі записи та видаляти їх у разі потреби;

– Працівник служби підтримки — контролює стан підключення пристроїв, переглядає журнали помилок та надсилає сповіщення користувачам про виявлені технічні несправності або необхідність обслуговування обладнання.

3.2 Проєктування архітектури ПЗ

Для розробки програмної системи автоматизованого управління освітленням для вирощування рослин буде використано інтегроване середовище розробки Visual Studio Code із необхідними розширеннями для роботи з мовами програмування TypeScript і JavaScript.

Серверну частину системи буде реалізовано за допомогою мови програмування TypeScript із використанням фреймворку NestJS для організації структури застосунку та TypeORM для спрощеної взаємодії з базою даних. До складу технологій серверної частини увійдуть такі бібліотеки: TypeORM — для реалізації об'єктно-реляційного відображення (ORM) між сутностями та базою даних; Passport.js — для побудови механізмів автентифікації користувачів; JsonWebToken (JWT) — для створення та валідації токенів безпечної авторизації; Redis — для збереження сесій користувачів та оптимізації обробки даних; MQTT.js — для організації двосторонньої

взаємодії з IoT-пристроями; Swagger (OpenAPI) — для автоматичної генерації документації для API.

Фронтенд частина системи буде реалізована за допомогою бібліотеки React та мови програмування TypeScript. Для організації обміну даними з сервером буде застосовано технологію RTK Query, яка дозволяє ефективно працювати із запитами та кешуванням. Інтерфейс користувача буде створено на основі компонентної бібліотеки Ant Design (antd), яка забезпечує розробку адаптивного, сучасного та функціонального UI.

Для роботи з фізичними IoT-пристроями буде використано мікрокомп'ютер Raspberry Pi із встановленим спеціалізованим програмним забезпеченням на базі Node.js. Передача даних між пристроєм та серверною частиною здійснюватиметься за допомогою протоколу MQTT для забезпечення стабільної комунікації у режимі реального часу.

Серверна частина програмної системи взаємодіятиме з базою даних PostgreSQL — сучасною системою управління реляційними базами даних, яка відзначається високою продуктивністю, надійністю та широким функціоналом для роботи зі складними запитами й транзакціями. Інтеграція PostgreSQL із ORM-фреймворком TypeORM дозволяє спростити обробку даних, автоматизувати основні операції взаємодії з базою даних та значно зменшити обсяг рутинного коду, що підвищує ефективність і масштабованість системи.

У структурі бази даних буде передбачено встановлення зв'язків між таблицями для забезпечення системної організації даних, підтримки їхньої цілісності та спрощення обробки запитів.

3.3 Проєктування структури зберігання даних

У результаті аналізу предметної області були визначені такі основні об’єкти (сутності):

У процесі аналізу предметної області було визначено основні сутності, які формують логічну структуру програмної системи:

– Користувачі (Users) — зберігають інформацію про кожного користувача системи, включаючи логін, зашифрований пароль і дату створення облікового запису. Користувач може мати одну або кілька ролей, наприклад, роль адміністратора чи технічного працівника.

– Ролі (Roles) — містять перелік можливих ролей у системі, що визначають права доступу користувачів, зокрема ролі адміністратора, технічного працівника або звичайного користувача.

– Призначення ролей (UserRoles) — описують взаємозв'язок між користувачами та їхніми ролями, що дозволяє підтримувати можливість надання декількох ролей одному обліковому запису.

– Рослини (Plants) — зберігають дані про окремі рослини, включаючи їх назву, допустимі межі освітленості (мінімальні та максимальні значення), а також параметри росту.

– Сенсори (Sensors) — містять відомості про сенсорні пристрої, встановлені для моніторингу умов вирощування рослин, такі як назва сенсора та його прив'язка до конкретної рослини.

– Освітлювальні пристрої (Lamps) — включають характеристики джерел світла, серед яких назва, рівень яскравості, спектральні показники (ультрафіолетове, червоне, синє світло) та інформація про сенсор, до якого підключено відповідний пристрій.

– Зчитування даних (Readings) — зберігають результати вимірювань, отримані від сенсорів, із зазначенням значення, дати та часу зчитування, а також позначкою про наявність попереджень у разі виходу параметрів за межі допустимих норм.

Для побудови ефективної логічної моделі даних було визначено атрибути для кожної з основних сутностей програмної системи:

Користувачі (Users):

a. id — унікальний ідентифікатор облікового запису користувача;
b. login — унікальне ім'я користувача для входу в систему;

c. password — хешований пароль для забезпечення безпеки облікового запису;
d. email — електронна адреса користувача для відновлення доступу або сповіщень;
e. created_at — дата та час створення облікового запису;
f. last_login — дата та час останнього входу в систему.

Ролі (Roles):

a. #id — унікальний ідентифікатор ролі;
b. name — назва ролі (наприклад, адміністратор, технік, користувач);
c. description — короткий опис призначення ролі в системі;
d. permissions — перелік прав доступу, закріплених за відповідною роллю (наприклад, право редагувати рослини, додавати сенсори або переглядати звіти).

Призначення ролей користувачам (UserRoles):

a. #id — унікальний ідентифікатор запису призначення ролі;
b. user_id — ідентифікатор користувача, якому призначено роль;
c. role_id — ідентифікатор відповідної ролі;
d. assigned_at — дата та час призначення ролі користувачу.

Рослини (Plants):

a. #id — унікальний ідентифікатор рослини;
b. name — назва рослини;
c. min_light_level — мінімально допустимий рівень освітлення для рослини;
d. max_light_level — максимально допустимий рівень освітлення;
e. growth_percentage — показник стадії розвитку рослини у відсотках;
f. created_at — дата та час додавання рослини до системи;
g. description — додаткова текстова інформація про рослину.

Сенсори (Sensors):

a. #id — унікальний ідентифікатор сенсора;
b. plant_id — ідентифікатор рослини, до якої прикріплено сенсор;
c. name — найменування сенсора;
d. type — тип сенсора (наприклад, освітлення, температура, вологість);
e. location — опис розташування сенсора (за потреби для великих об'єктів);
f. status — статус сенсора (активний/неактивний).

Освітлювальні пристрої (Lamps):

a. #id — унікальний ідентифікатор лампи;
b. sensor_id — ідентифікатор сенсора, до якого прив'язаний освітлювальний пристрій;
c. name — назва джерела світла;
d. spectrum — тип спектра випромінювання (ультрафіолет, червоне світло тощо);
e. light_level — рівень яскравості випромінювання;
f. status — поточний стан лампи (увімкнено/вимкнено).

Зчитування даних (Readings):

a. #id — унікальний ідентифікатор зчитування;
b. sensor_id — ідентифікатор сенсора, що передав зчитування;
c. name — тип зчитування (наприклад, рівень освітлення, температура);
d. value — числове значення, отримане в результаті вимірювання;
e. date_time — дата та час отримання даних;
f. is_warning — позначка про наявність попередження (так/ні);
g. unit — одиниця виміру значення (наприклад, люкс, °C).

У структурі бази даних програмної системи передбачено такі основні зв'язки між сутностями:

a. Користувачі (Users) та Ролі (Roles) пов'язані через проміжну таблицю UserRoles відношенням типу "багато до багатьох", оскільки один користувач може мати кілька ролей, і одна роль може бути призначена кільком користувачам.
b. Сенсори (Sensors) та Рослини (Plants) перебувають у відношенні "один до багатьох": одна рослина може бути асоційована з кількома сенсорами, водночас кожен сенсор належить лише одній рослині.
c. Зчитування (Readings) та Сенсори (Sensors) також пов'язані зв'язком "один до багатьох", де кожен сенсор може мати багато зчитувань, проте кожне зчитування прив'язане до одного конкретного сенсора.
d. Освітлювальні пристрої (Lamps) та Сенсори (Sensors) мають відношення "один до багатьох", оскільки до одного сенсора може бути підключено кілька джерел освітлення, однак кожен освітлювальний пристрій прив'язаний тільки до одного сенсора.

На основі зазначених сутностей і зв'язків побудовано ER-діаграму бази даних (див. рис. 3.1).

Таким чином, розроблена ER-модель відображає основні сутності та їхні взаємозв'язки, що забезпечує структуроване зберігання даних і оптимізує процес обробки інформації в межах програмної системи.

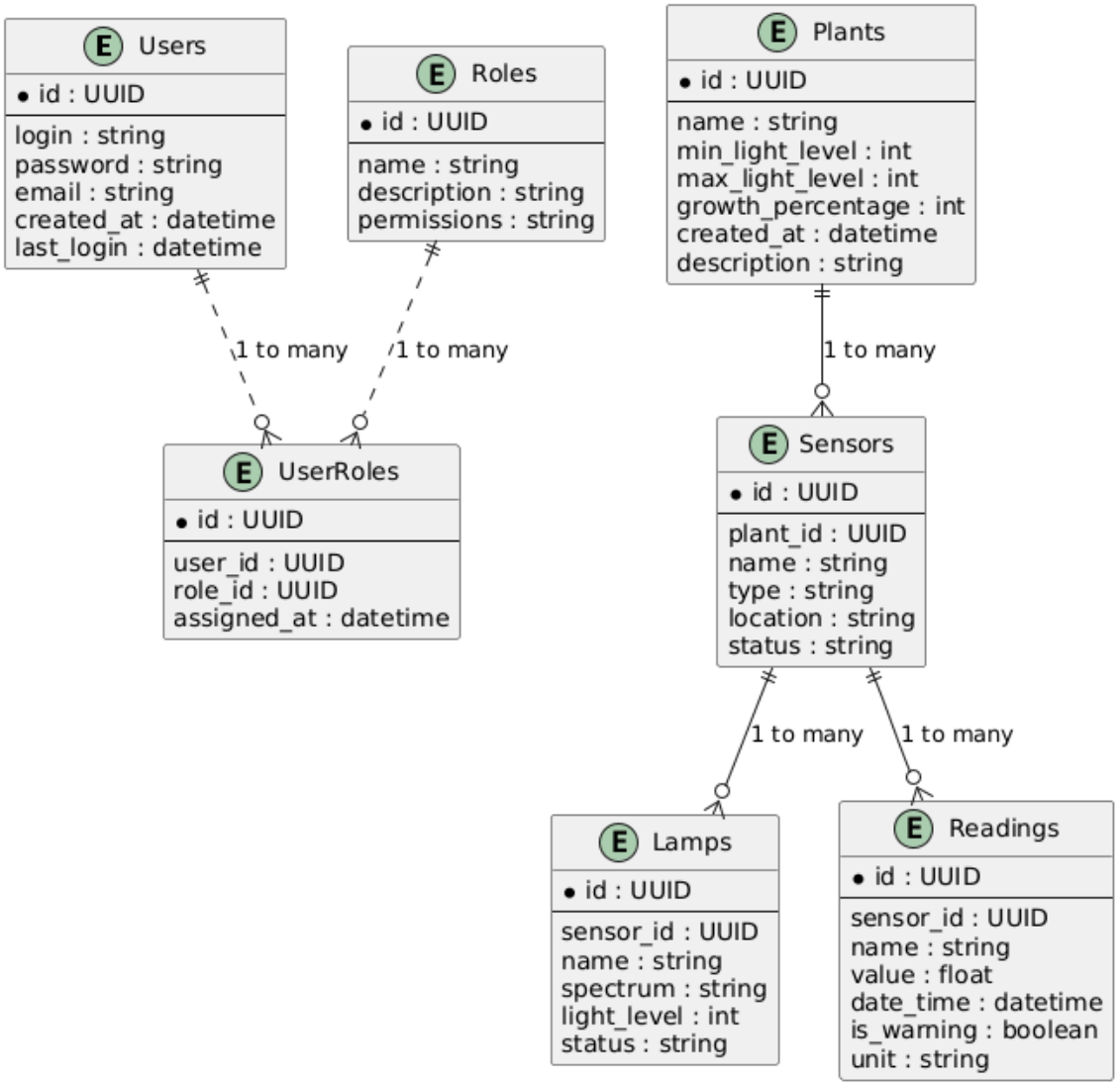

Рисунок 3.2 – Діаграма ER-модель даних (рисунок виконано самостійно)

Побудована структура дозволяє підтримувати цілісність даних, масштабованість та ефективність виконання запитів, що є необхідними умовами для подальшої реалізації та розвитку системи.

3.4 Найцікавіші алгоритми та методи

У системі для генерації документації REST API використано Swagger, що реалізовано через модуль @nestjs/swagger. Він дозволяє автоматично описати всі маршрути, параметри, запити та відповіді за допомогою зручних декораторів (@ApiTags, @ApiOperation, @ApiResponse тощо).

Документація доступна за адресою /api/docs і дозволяє тестувати запити прямо з браузера, що значно полегшує розробку та налагодження (див. рис. 3.5).

```
import { INestApplication } from "@nestjs/common";
import { DocumentBuilder, SwaggerModule } from "@nestjs/swagger";

export function swaggerConfig(app: INestApplication) {
    const swaggerConfig = new DocumentBuilder()
        .setTitle('Plants API')
        .setDescription('API for Plants Light Control')
        .setVersion('1.0')
        .build();

    const documentFactory = () =>
        SwaggerModule.createDocument(app, swaggerConfig);

    SwaggerModule.setup('api/docs', app, documentFactory);
};
```

Рисунок 3.3 – Код конфігурування Swagger (рисунок виконано самостійно)

Цей код налаштовує Swagger/OpenAPI для NestJS-додатку, використовуючи патерн "Builder" для ланцюжкового конфігурування об'єкта swaggerConfig та подальшої генерації документації API.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Використання сесій

У системі реалізовано автентифікацію за сесійним підходом. Після успішного входу користувача сервер зберігає сесію у Redis, а в браузер надсилає httpOnly cookie з ідентифікатором сесії (sid). Це забезпечує безпечну і стійку авторизацію без використання токенів на клієнтській стороні.

```
export function configureSession(app: INestApplication) {
    const redisClient = new Redis({ host: 'localhost', port: 6379 });

    app.use(
        session({
            name: 'sid',
            store: new RedisStore({ client: redisClient, disableTouch: true }),
            secret: 'mysecretkey',
            resave: false,
            saveUninitialized: false,
            cookie: {
                httpOnly: true,
                maxAge: 1000 * 60 * 60 * 24 * 7, // 7 дней
                sameSite: 'lax',
            },
        }),
    );
}
```

Рисунок 4.1 – Код конфігурування сесій з Redis у NestJS (рисунок виконано самостійно)

Під час завантаження додатку фронтенд викликає /auth/me для перевірки автентифікації. У разі невдалого запиту користувач автоматично перенаправляється на сторінку входу.

Доступ до сторінок обмежено.

– Ресурси /login, /signup, / – доступні всім;

– Ресурси /plants, /lamps, /sensors – лише для авторизованих користувачів;

– Ресурси /users – тільки для адміністратора (role: admin);

– Ресурси Профіль і налаштування — доступні після входу.

Клієнтська частина програмної системи реалізована як SPA (Single Page Application) з використанням сучасного стеку технологій:

– React — для побудови інтерфейсу;

– TypeScript — для типізації компонентів;

– Redux Toolkit Query (RTK Query) — для запитів до API та керування станом;

– React Router — для маршрутизації;

– Ant Design — для створення форм, таблиць, вікон.

4.2 Робота з API та керування станом через RTK Query

У клієнтській частині програмної системи для організації взаємодії з сервером та централізованого керування станом застосовується Redux Toolkit Query (RTK Query) — інструмент із екосистеми Redux, спеціально призначений для ефективної роботи з API.

RTK Query дозволяє значно спростити процес виконання HTTP-запитів, обробки помилок, кешування відповідей і повторного використання результатів запитів у різних компонентах. Він інтегрується з Redux Store, автоматично створюючи slice для запитів.

Усі запити до API оголошуються централізовано в окремому файлі api.ts, у якому описано базовий createApi, що містить інформацію про базовий URL сервера, тип заголовків, а також список endpoint'ів (наприклад: login, getMe, getPlants, createSensor тощо). Приклад визначення одного з endpoint'ів:

```
export const api = createApi({
  reducerPath: 'api',
  baseQuery: fetchBaseQuery({ baseUrl: '/api' }),
  endpoints: (builder) => ({
    getMe: builder.query<User, void>({
      query: () => '/auth/me',
```

```
    }),
  }),
});
```

Використання у компонентах:

Компоненти не містять жодної логіки HTTP-запитів — замість цього вони використовують хуки, автоматично згенеровані RTK Query на основі endpoint'ів. Наприклад:

```
const { data: user, isLoading, error } = useGetMeQuery();
```

Це дозволяє:

– спростити структуру компонентів;

– автоматично отримувати статуси завантаження (isLoading, isSuccess, isError);

– повторно використовувати відповіді з кешу, якщо вони вже були отримані;

– обробляти помилки централізовано через middleware.

RTK Query автоматично кешує дані, що зменшує кількість запитів до сервера при повторному рендерінгу компонентів або при поверненні до вже відкритих сторінок. Термін життя кешу можна налаштовувати окремо для кожного запиту.

Після виконання мутацій (наприклад, login, addPlant, updateLamp) RTK Query дозволяє вручну оновити кеш або автоматично перезапустити певні запити через invalidatesTags. Це гарантує, що дані на фронтенді завжди актуальні без необхідності перезавантаження сторінки.

Усі помилки, що виникають під час запитів, можна обробити як у компонентах, так і глобально — наприклад, для виведення повідомлень через Ant Design message.error.

4.3 Структура проєкту

Під час розробки клієнтської частини було застосовано сучасну структуру з чітким розділенням відповідальностей між API, інтерфейсом та маршрутизацією. Структуру директорій наведено на рисунках 4.2 та 4.3

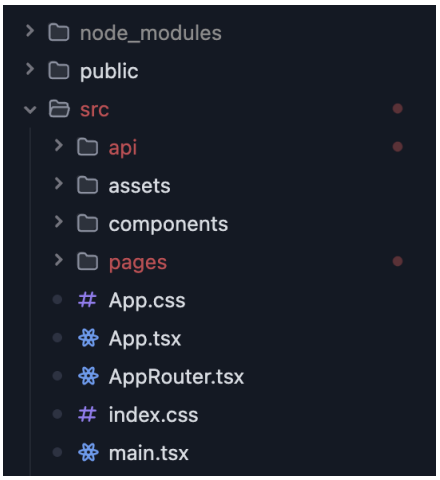

Рисунок 4.2 – Структура директорій клієнтської частини проєкту (частина 1) (рисунок виконано самостійно)

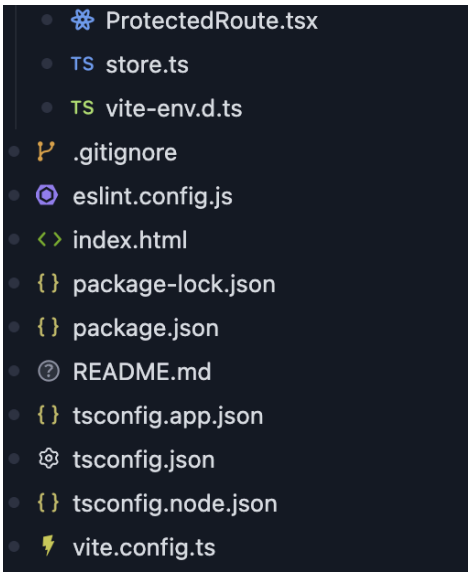

Рисунок 4.3 – Структура директорій клієнтської частини проєкту (частина 2) (рисунок виконано самостійно)

Опис директорій та файлів:

– api/ — директорія, яка містить централізовану логіку для взаємодії з серверною частиною. Тут оголошуються endpoint’и, що використовуються RTK Query для здійснення HTTP-запитів (login, getMe, getPlants тощо).

– assets/ — директорія для зберігання медіа-ресурсів, зображень, іконок або стилів, що використовуються в інтерфейсі.

– components/ — тут розміщені багаторазові UI-компоненти (наприклад, навігаційне меню, форми, заголовки), які використовуються в різних частинах інтерфейсу. Компоненти мають ізольовану логіку та власні стилі.

– pages/ — директорія з компонентами сторінок, кожна з яких відповідає окремому маршруту (наприклад, /login, /users, /plants тощо). Кожна сторінка об’єднує компоненти, які пов’язані з певною функціональністю.

– App.tsx — кореневий компонент застосунку, який відповідає за початкову ініціалізацію та обгортання всього інтерфейсу.

– AppRouter.tsx — файл, у якому реалізовано маршрутизацію за допомогою React Router. Тут визначаються усі маршрути, доступні у застосунку.

– ProtectedRoute.tsx — компонент, що реалізує перевірку прав доступу до захищених сторінок. Якщо користувач не автентифікований або не має необхідної ролі, його буде перенаправлено.

– store.ts — файл, у якому створюється Redux Store, підключається RTK Query middleware та налаштовуються редуктори.

– main.tsx — точка входу у застосунок, де ReactDOM рендерить кореневий компонент у DOM.

– vite-env.d.ts — декларації типів для Vite.

– index.html — базовий HTML-файл, у якому відбувається підключення React-програми.

– vite.config.ts — файл конфігурації для білдера Vite, який використовується для швидкої розробки та збірки фронтенду.

– tsconfig.json* — конфігураційні файли для TypeScript: основний (tsconfig.json), для app (tsconfig.app.json) та для node-скриптів (tsconfig.node.json).

– eslint.config.js — файл конфігурації ESLint, який забезпечує контроль якості коду та стандартизацію стилю.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Ручне тестування та тестування клієнтської частини

У межах перевірки якості розробленої програмної системи було проведено ручне тестування основного функціоналу для всіх типів користувачів: адміністратора, технічного персоналу та звичайного користувача. Тестування API здійснювалося за допомогою інтерактивної документації Swagger, що дозволило перевірити валідність запитів, обробку помилок та правильність маршрутизації.

Для контролю доступу до ресурсів реалізовано авторизацію через сесії. Кожен запит до захищених маршрутів супроводжувався автоматичною перевіркою сесії, збереженої на сервері в Redis. У разі відсутності сесії або невірної авторизації користувач отримував статус 401 Unauthorized, що свідчить про захищеність API.

– Під час тестування вручну перевірялися такі типові помилки:

– Запит до захищених маршрутів без активної сесії (без cookie);

– Неправильні дані для входу (невірний логін або пароль);

– Запити до ресурсів, які вимагають роль (наприклад, admin), з облікового запису без відповідних прав;

– Невалідні формати JSON-запитів;

– Спроба створити дублікати об’єктів (рослин, сенсорів, користувачів);

– Звернення до неіснуючих ресурсів за ID;

– Спроби видати або відкликати ролі, які вже призначені / не призначені.

Далі було перевірено основний функціонал:

Для адміністратора:

– Успішний вхід до системи (отримання сесії);

– Перегляд списку всіх користувачів;

– Перегляд деталей обраного користувача;

– Видалення користувача;

– Призначення та відкликання ролей;

– Експорт користувачів у формат JSON;

– Імпорт користувачів із JSON-файлу.

Для технічного персоналу:

– Отримання списку рослин;

– Створення, редагування, видалення рослин;

– Робота з сенсорами: додавання, редагування, зв’язок із рослинами, видалення;

– Додавання зчитувань (readings) для сенсорів;

– Перегляд джерел освітлення, їхнє додавання, редагування спектру та яскравості.

Для звичайного користувача:

– Перегляд показників окремих сенсорів;

– Перегляд системних попереджень, пов’язаних із рослинами;

– Отримання списку усіх сенсорів.

Фронтенд-застосунок тестувався через браузер у реальному середовищі, з перевіркою доступу до сторінок, коректності маршрутизації та взаємодії з API.

Основні кейси, що тестувались:

– Перевірка перенаправлення неавторизованого користувача на сторінку входу;

– Обробка сценарію з невалідними даними при вході (помилковий пароль);

– Доступність компонентів тільки для користувача з відповідною роллю (напр. /users — лише для адміністратора);

– Кешування, фетчинг і повторне отримання даних через RTK Query;

– Інтерфейс тестувався на коректність заповнення форм, коректність валідації, зручність навігації та наявність всіх очікуваних елементів.

5.2 Unit-тестування на стороні серверу (NestJS)

Для підвищення стабільності системи та виявлення потенційних помилок було реалізовано unit-тестування критичних сервісів NestJS.

Інструменти:

– @nestjs/testing — для створення тестових модулів;

– jest — як тест-ранер;

– supertest — для перевірки HTTP-взаємодії.

Приклади покриття:

– Тестування авторизаційного сервісу (успішний вхід, невірні облікові дані);

– Тестування ролей (додавання, перевірка наявності, видалення);

– Тестування логіки створення користувача;

– Тестування обробки винятків у сервісах.

5.3 E2E-тестування

Для перевірки взаємодії між різними частинами системи реалізовано end-to-end тестування з використанням:

– supertest — для виконання HTTP-запитів до розгорнутої NestJS-програми;

– Тестової бази даних (у режимі test.env);

– Попереднього створення тестових облікових записів і очищення після виконання тестів.

Основні сценарії:

– Повний цикл: реєстрація → вхід → отримання даних → вихід;

– Авторизація користувача з різними ролями;

– Додавання об’єктів (рослин, сенсорів, ламп) та перевірка їхнього збереження;

– Відмова в доступі при відсутності ролі або неавторизованому запиті.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було реалізовано серверну та клієнтську частини програмної системи для автоматизованого керування освітленням у середовищі вирощування рослин у режимі реального часу.

Сервер розроблено з використанням TypeScript, NestJS і TypeORM, що забезпечило гнучку структуру та надійну взаємодію з базою даних PostgreSQL. Обмін даними з IoT-пристроєм на базі Raspberry Pi відбувається через MQTT-протокол.

Клієнтська частина створена на основі React і TypeScript із використанням RTK Query та Redux Toolkit для обробки запитів, а також Ant Design для інтерфейсу.

У результаті створено систему з підтримкою рольового доступу (адміністратор, технік, користувач, підтримка), яка дозволяє управляти користувачами, рослинами, сенсорами, освітленням та зчитуваннями. Також передбачено імпорт і експорт даних у форматі JSON.

Отже, було розроблено сучасну інформаційну систему для моніторингу та керування освітленням у сільському господарстві.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Документація Node.js [Електронний ресурс] – URL: https://nodejs.org/en/docs/ (дата звернення: 28.05.2025)

2. Документація NestJS [Електронний ресурс] – URL: https://docs.nestjs.com/ (дата звернення: 30.05.2025)

3. Офіційна документація TypeORM [Електронний ресурс] – URL: https://typeorm.io/ (дата звернення: 31.05.2025)

4. Документація PostgreSQL [Електронний ресурс] – URL: https://www.postgresql.org/docs/ (дата звернення: 27.05.2025)

5. Документація MQTT Protocol [Електронний ресурс] – URL: https://mqtt.org/documentation/ (дата звернення: 29.05.2025)

6. Офіційна документація Redis [Електронний ресурс] – URL: https://redis.io/docs/ (дата звернення: 02.06.2025)

7. Посібник з використання React [Електронний ресурс] – URL: https://react.dev/learn (дата звернення: 26.05.2025)

8. Офіційна документація TypeScript [Електронний ресурс] – URL: https://www.typescriptlang.org/docs/ (дата звернення: 01.06.2025)

9. Офіційна документація RTK Query [Електронний ресурс] – URL: https://redux-toolkit.js.org/rtk-query/overview (дата звернення: 30.05.2025)

10. Документація Redux Toolkit [Електронний ресурс] – URL: https://redux-toolkit.js.org/ (дата звернення: 31.05.2025)

11. Документація React Router [Електронний ресурс] – URL: https://reactrouter.com/en/main (дата звернення: 27.05.2025)

12. Репозиторій з програмним кодом на GitHub [Електронний ресурс] – URL: https://github.com/NureBohdanIlienko/2025_B_PI_PZPI-21-10_Ilienko_B_A (дата звернення: 07.06.2025)