

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  
виращування рослин. Бек-енд \_\_\_\_\_  
(тема)

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

\_\_\_\_\_ Єгор ОСТРОВЕРХОВ \_\_\_\_\_  
(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 ім'я, 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 \_\_\_\_\_ Острроверхову Стору Андрійовичу \_\_\_\_\_  
 (прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для автоматизованого управління освітленням для вирощування рослин. Бек-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                     | <i>виконано</i> |
| 2  | Проектування ПЗ                    | 14.04.2025                     | <i>виконано</i> |
| 3  | Розробка ПЗ                        | 29.04.2025                     | <i>виконано</i> |
| 4  | Тестування ПЗ                      | 12.05.2025                     | <i>виконано</i> |
| 5  | Оформлення пояснювальної записки   | 20.05.2025                     | <i>виконано</i> |
| 6  | Підготовка презентації та доповіді | 27.05.2025                     | <i>виконано</i> |
| 7  | Нормоконтроль, рецензування        | 01.06.2025                     | <i>виконано</i> |
| 9  | Здача роботи у електронний архів   | 04.06.2025                     | <i>виконано</i> |
| 9  | Допуск до захисту у зав. кафедри   | 07.06.2025                     | <i>виконано</i> |
| 10 | Попередній захист                  | 08.06.2025                     | <i>виконано</i> |

Дата видачі завдання 9 квітня 2025 р.

Здобувач \_\_\_\_\_  
  
 (підпис)

Керівник роботи \_\_\_\_\_ к.т.н., доц. кафедри ІІ Дмитро КОЛЕСНИКОВ  
 (підпис) (посада, власне ім'я, прізвище)

## РЕФЕРАТ

Пояснювальна записка для передатестаційної практики: 53 сторінки, 24 рисунки, 13 джерел.

АВТОМАТИЗАЦІЯ ОСВІТЛЕННЯ, ВІДДАЛЕНИЙ КОНТРОЛЬ, МОНІТОРИНГ, ПЛАНТАЦІЇ, ПРОГРАМНА СИСТЕМА ФЕРМА, JAVA, MAVEN, POSTGRESQL, SPRING BOOT, SPRING WEB.

Об'єктом дослідження є сільське господарство, зокрема вирощування рослин, і вдосконалення автоматичних систем освітлення на фермах та плантаціях. Сучасні способи використання штучного освітлення часто не враховують потреби рослин, що може уповільнювати їхній розвиток і зменшувати врожай. Тому необхідно розробити нову систему, яка автоматично регулює освітлення залежно від фази росту рослин та рівня вологості, що допоможе зробити вирощування більш ефективним.

Метою роботи є проектування розробки програмної системи для ефективного контролю освітлення на фермах та плантаціях. Система є відповідальною за збір і збереження інформації про потреби рослин у світлі, їхні фази розвитку та оптимальні режими освітлення. Використовуючи ці дані, вона зможе автоматично регулювати освітлення, створюючи сприятливі умови для росту та розвитку рослин.

Методи розробки системи передбачають використання мови програмування Java 17 для створення серверної частини. Для зберігання даних про рослини та параметри освітлення застосовується СУБД PostgreSQL. Також розроблено смарт-пристрій у вигляді емулятора на Java, до якого можна підключати різні датчики для збору даних про стан рослин і визначення оптимальних умов освітлення.

У результаті було створено опис програмної системи, що забезпечить автоматизоване управління освітленням на фермах та плантаціях. До складу системи входить сервер для зберігання й обробки інформації, а також смарт-пристрій, що відстежуватиме параметри стану рослин.

## ABSTRACT

LIGHTING AUTOMATION, REMOTE CONTROL, MONITORING, PLANTATIONS, SOFTWARE SYSTEM FARM, JAVA, MAVEN, POSTGRESQL, SPRING BOOT, SPRING WEB.

The research object is agriculture, particularly plant cultivation, and the improvement of automatic lighting systems on farms and plantations. Modern methods of using artificial lighting often do not consider plants' needs, which can slow down their development and reduce yield. Therefore, it is necessary to develop a new system that automatically regulates lighting depending on the phase of plant growth and humidity level, which will help make cultivation more efficient. The work aims to design and develop a software system for effective lighting control on farms and plantations. The system will be responsible for collecting and storing information about the lighting needs of plants, their growth phases, and optimal lighting modes. Using this data, it will be able to automatically adjust the lighting, creating favorable conditions for plant growth and development. The methods of system development involve the use of Java 17 programming language for the creation of the server part. For storing data about plants and lighting parameters, the PostgreSQL database management system is used. A smart device in the form of an emulator developed in Java has also been created, to which various sensors can be dynamically connected for collecting data on plant conditions and determining optimal lighting conditions. As a result, a description of the software system will be created, which will provide automated lighting control on agricultural facilities such as farms and plantations. The system will include a server for storing and processing information, as well as a smart device that monitors plant condition parameters.

## ЗМІСТ

|   |    |
|---|----|
| Вступ   | 9  |
| 1. Аналіз предметної галузі                             | 10 |
| 1.1. Аналіз предметної галузі                           | 10 |
| 1.2. Виявлення та вирішення проблем                     | 10 |
| 1.3. Постановка задачі                                  | 15 |
| 2. Формування вимог для програмної системи              | 17 |
| 2.1. Огляд частин програмної системи                    | 17 |
| 2.2. Серверна частина програмної системи                | 17 |
| 2.3. IoT частина програмної системи                     | 17 |
| 2.4. Основні функції                                    | 18 |
| 3. Архітектура та проектування програмного забезпечення | 20 |
| 3.1. UML проектування ПЗ                                | 20 |
| 3.2. Проектування архітектури ПЗ                        | 21 |
| 3.3. Проектування структури зберігання даних            | 22 |
| 3.4. Найцікавіші алгоритми та методи                    | 26 |
| 4. Опис прийнятих програмних рішень                     | 28 |
| 4.1. Використання JWT                                   | 28 |
| 4.2. Обробка помилок                                    | 30 |
| 4.3. Структура проєкту                                  | 31 |
| 5. Тестування розробленого програмного забезпечення     | 34 |
| 5.1. Ручне тестування                                   | 34 |
| Висновки  | 36 |
| Перелік джерел посилань                                 | 37 |
| Додаток А   | 38 |
| Додаток Б   | 39 |
| Додаток В   | 45 |

## ВСТУП

Метою цієї кваліфікаційної роботи є створення програмної системи, яка дозволяє автоматизувати та оптимізувати керування освітленням на фермах і плантаціях. Для реалізації цієї мети були впроваджені такі функції:

- отримання інформації щодо рослин, освітлювальних пристроїв та сенсорів у програмній системі;
- функція додавання нових рослин, освітлювальних приладів та датчиків до програмної системи;
- налаштування рівня яскравості та кольорового спектру освітлення;
- автоматичне налаштування освітлювальних приладів згідно з даними, що були отримані з датчиків;
- створення резервних копій даних користувачів та їх відновлення;
- отримання всіх даних від сенсорів, щодо умов росту рослин та фільтрування цих даних;

При розробці програмної системи використовувалися наступні технології: Java, PostgreSQL[1], Maven [2], Spring Boot [3].

Система сприяє поліпшеному вирощуванню рослин через автоматизацію процесів освітлення та використання даних від сенсорів для моніторингу стану рослин.

## 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

### 1.1 Аналіз предметної галузі

У наш час зростає зацікавленість у впровадженні автоматизованих систем керування освітленням у сфері вирощування рослин. Традиційні підходи, які передбачають використання штучного освітлення без урахування реальних потреб рослин, часто виявляються малоефективними. Вони можуть не відповідати біологічним особливостям різних культур, що нерідко призводить до нестабільного зростання, зниження врожайності та завищені витрати енергоресурсів.

У зв'язку з цим виникає необхідність у впровадженні сучасного рішення, яке б забезпечувало автоматичне регулювання освітлення залежно від ряду ключових факторів – зокрема, стадії розвитку рослин, рівня вологості та інших параметрів навколишнього середовища. Програмна система, метою якої є не лише покращення умов для вирощування рослин, але й підвищення ефективності агропроцесів загалом, збільшення врожаю та оптимізація енергоспоживання. Це дозволить зробити процес вирощування керованішим, економічно доцільним і екологічно обґрунтованим.

### 1.2 Виявлення та вирішення проблем

Для того, щоб зрозуміти сучасний ринок та його проблеми, проаналізуємо конкурентів.

Один з найбільших конкурентів – Sollum Technologies (див. рис. 1.1) пропонує динамічні світлодіодні системи освітлення, які можуть автоматично змінювати спектр та інтенсивність світла в режимі реального часу, адаптуючись до потреб різних культур. Їхня платформа SUN as a Service дозволяє віддалено керувати освітленням, забезпечуючи оптимальні умови для

росту рослин та підвищення врожайності. Проте не дивлячись на сучасність та зручність їх рішень, вони мають великий недолік у вигляді високої вартості впровадження, що є такою через передові технології, що в свою чергу робить систему практично недоступною для малих або середніх господарств, орієнтуючись лише на великі.

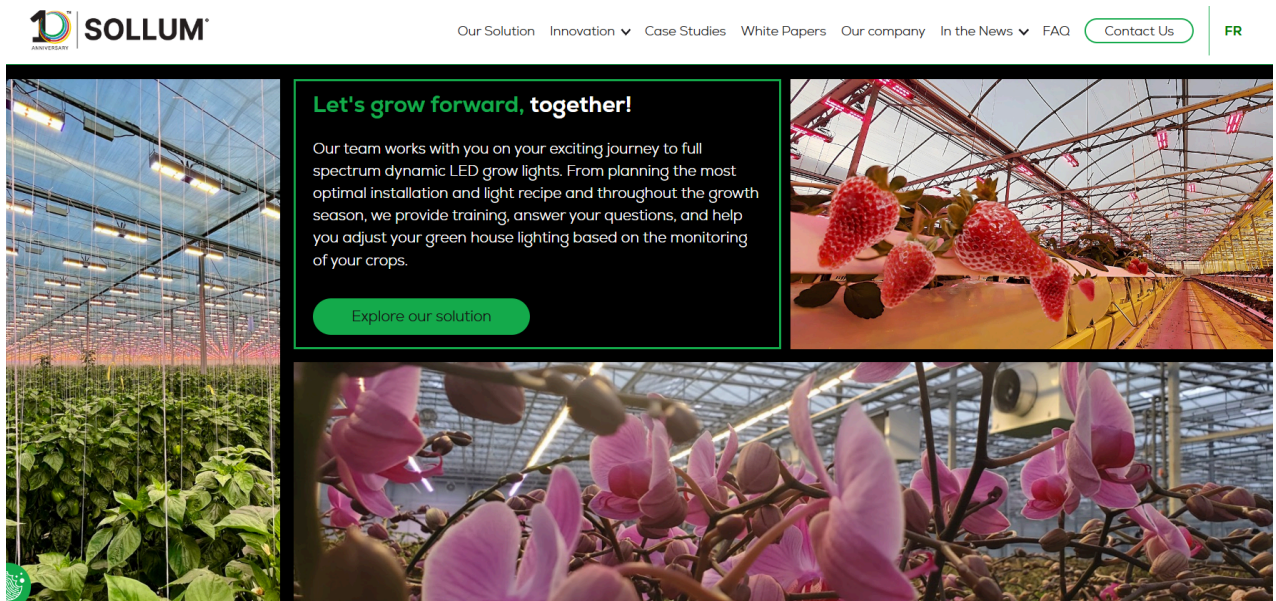


Рисунок 1.1 – Головна сторінка сайту Sollum Technologies [4]

Heliospectra (див. рис. 1.2) – спеціалізується на інтелектуальних світлодіодних рішеннях для теплиць. Їхня система helioCORE дозволяє автоматично регулювати освітлення на основі природнього світла, а також підтримує налаштування для кількох зон вирощування. Тут ми маємо таку саму проблему, основний фокус цієї компанії – великі промислові теплиці, що робить їх рішення малоадаптованим до невеликих ферм, а також складна інтеграція з різними типами датчиків, що не відносяться до Heliospectra.

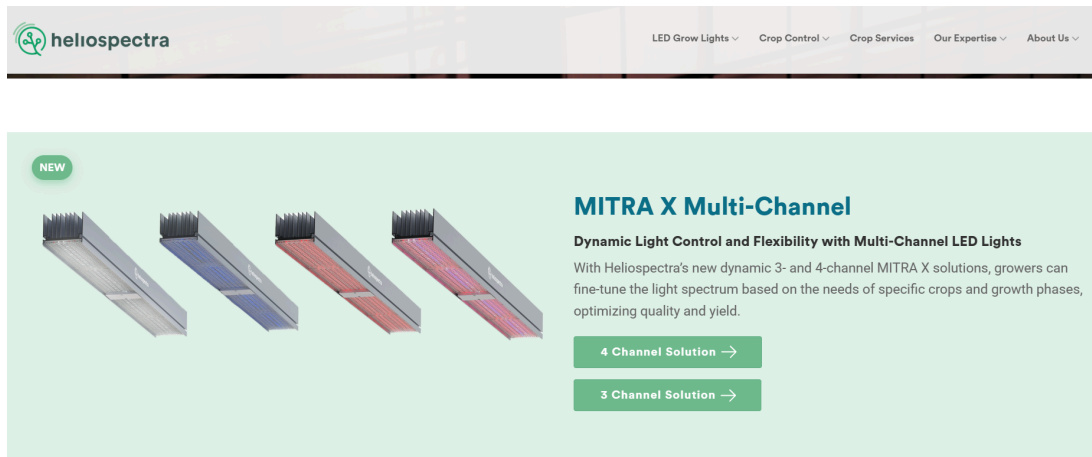


Рисунок 1.2 – Головна сторінка сайту Heliospectra [5]

Bioled (див. рис. 1.3) – спеціалізується на розробці інтелектуальних LED-систем освітлення для теплиць та закритих аграрних об'єктів. Їхні рішення включають IoT-керовані світильники з можливістю збору даних, аналізу та автоматичного налаштування освітлення на основі параметрів середовища, що сприяє енергоефективності та покращенню якості врожаю. Основна проблема цієї компанії в тому, що вона не має повноцінної екосистеми з мобільним чи веб-застосунком для кінцевого користувача, що унеможливорює роботу для недосвідченого користувача.

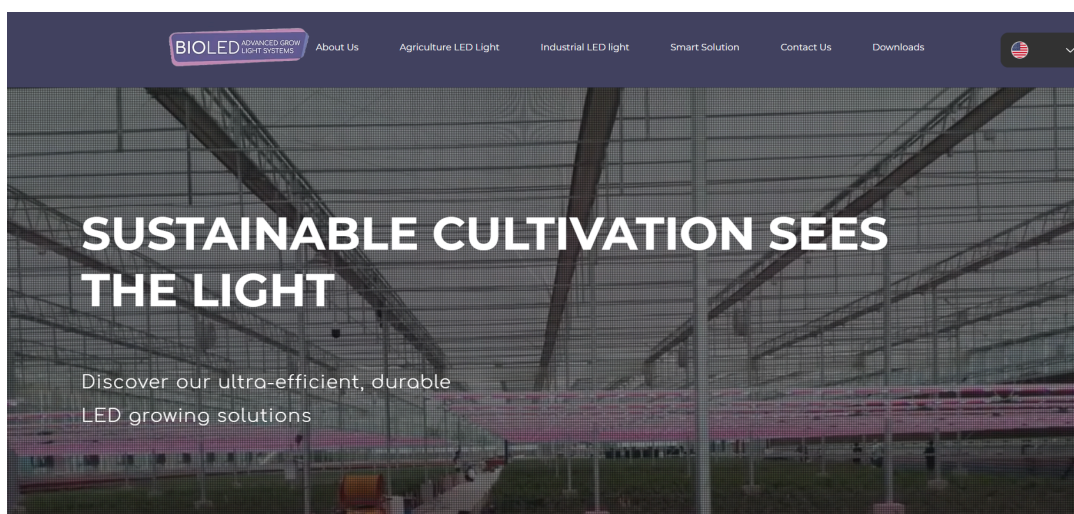


Рисунок 1.3 – Головна сторінка сайту Bioled [6]

Kroptek (див. рис. 1.4) – пропонує LED-світильники та рішення для контрольованого сільського господарства, включаючи автоматизоване управління кліматом та системи поливу. Їхні продукти спрямовані на зниження енергоспоживання та підвищення врожайності, забезпечуючи цілорічне вирощування різних культур. Kroptek зосереджені переважно на освітлювальних рішеннях, а не на повній автоматизації з контролем вологості, температури чи інших властивостей, а також в них виключно статичне налаштування, або ручне управління.



Рисунок 1.4 – Світлові рішення Kroptek [7]

Перш ніж розпочати розробку власної системи, було проаналізовано наявні на ринку рішення в галузі автоматизованих систем освітлення для вирощування рослин. Основна увага приділялася інноваційності, енергоефективності, рівню автоматизації, підтримці мобільних або веб-інтерфейсів, а також інтеграції з сенсорами.

Отже, порівняємо конкурентів виразивши їх переваги та недоліки (див. табл. 1.1).

Таблиця 1.1 – Порівняння аналогів (таблиця виконана самостійно)

| Назва компанії      | Переваги   | Недоліки   |
|---------------------|--|--|
| Sollum Technologies | Сучасні, інноваційні рішення; зручність використання; орієнтація на енергоефективність | Дуже висока вартість впровадження; недоступність для малих і середніх господарств  |
| Heliospectra        | Орієнтація на автоматизацію; потужні рішення для промислових теплиць                   | Погано адаптовані до невеликих ферм; складна інтеграція з небрендовими сенсорами   |
| Bioled              | Якісні освітлювальні системи   | Відсутність повноцінного мобільного або веб-інтерфейсу; складність у використанні для недосвідчених користувачів;  |
| Kroptek             | Якісні освітлювальні рішення; підтримка малих підприємств                              | Відсутність автоматизації зміни освітлення на основі кліматичних параметрів; (вологість, температура тощо); тільки ручне або статичне керування освітленням; |

Після порівняння існуючих аналогів, можна побачити з якими проблемами можемо зіштовхнутися ми під час розробки, отже, перейдемо до виявлення цих проблем та можливих рішень.

У процесі розробки програмної системи автоматизованого керування освітленням на фермах та плантаціях можуть виникнути кілька ключових проблем.

Однією з основних є інтеграція з різними типами сенсорів та обладнанням, що можуть мати несумісні протоколи або обмежену документацію. Це ускладнює налаштування системи в умовах реального господарства.

Ще однією проблемою є нестабільне або слабке інтернет з'єднання в сільській місцевості, що може впливати на своєчасне оновлення даних або

віддалене керування. Відповідно, потрібне рішення, що не буде орієнтуватися на такий тип з'єднання і зможе працювати офлайн.

Окремо слід зазначити, що в такій системі важливо забезпечити просте керування, уникнувши перевантаження функціями.

Також можливі ризики збереження та передавання даних, зокрема – втрата зчитувань сенсорів або помилки в обробці інформації, що може вплинути на якість прийняття рішень.

### 1.3 Постановка задачі

Отже, для створення цієї програмної системи необхідно вирішити низку задач.

Розробити серверну частину системи, яка відповідатиме за зберігання, обробку та логіку взаємодії між усіма компонентами. Для цього буде використано Java 17 у зв'язці з Spring Boot – сучасним фреймворком, що дозволяє швидко створювати надійні веб-сервіси.

Реалізувати REST API [8], що забезпечить взаємодію між клієнтськими застосунками, датчиками, смарт-пристроями та сервером.

Впровадити можливість авторизації користувачів за іменем користувача та паролем.

Здійснити зберігання структурованої інформації про рослини, фази зростання, типи освітлення, показники сенсорів та історію змін за допомогою системи управління базами даних PostgreSQL.

Реалізувати обробку вхідних даних від сенсорів у режимі реального часу та на основі цих даних керувати інтенсивністю й тривалістю роботи ламп.

Забезпечити можливість резервного копіювання та відновлення даних користувачів та налаштувань системи.

А також використати Maven як систему управління проектом для зручної роботи з залежностями, тестування та збірки застосунку.

## 2 ФОРМУВАННЯ ВИМОГ ДЛЯ ПРОГРАМНОЇ СИСТЕМИ

### 2.1 Огляд частин програмної системи

Програмна система складатиметься з двох основних компонентів:

- Бек-енд частина;
- модуля IoT (інтернету речей);

### 2.2 Серверна частина програмної системи

Серверна частина даної програмної системи створена з використанням мови програмування Java, фреймворку Spring Boot та ORM-бібліотеки Hibernate [9], що реалізує об'єктно-реляційне відображення. Для написання та тестування коду застосовується інтегроване середовище розробки IntelliJ IDEA, розроблене компанією JetBrains.

Управління залежностями та процесом збірки виконується за допомогою Maven — інструменту, який спрощує конфігурацію проекту.

Spring Boot надає зручні засоби для швидкої розробки веб-застосунків і сприяє створенню масштабованих, продуктивних та стабільних серверних рішень. Завдяки використанню Hibernate взаємодія з базою даних стає більш ефективною та автоматизованою, що дозволяє мінімізувати ручну обробку запитів і зберігання інформації.

Обраний перелік технологій забезпечує надійне, масштабоване та легко підтримуване серверне середовище для функціонування всієї системи.

### 2.3 IoT частина програмної системи

Для реалізації IoT-компонента було обрано мову програмування Java. Хоча система слугуватиме лише імітацією реального IoT-пристрою, вона

повністю виконуватиме необхідні функції — генеруватиме псевдовипадкові значення, які оброблюватимуться та надсилатимуться на сервер для подальшої обробки. Таким чином, пристрій відтворюватиме логіку справжнього "розумного" пристрою.

Обмін даними між IoT-сервісом і сервером здійснюватиметься за допомогою протоколу HTTP[10] із використанням архітектури REST. Сам сервіс буде реалізований з використанням фреймворку Spring Boot.

## 2.4 Основні функції

Програмна система включатиме такі функціональні можливості:

Функції, що має адміністратор:

- створення облікового запису користувача;
- вхід до системи з авторизацією;
- виведення повного списку зареєстрованих користувачів;
- перегляд детальної інформації про окремого користувача;
- видалення облікового запису окремого користувача;
- перегляд ролей, що закріплені за користувачем;
- видача нової ролі користувачеві;
- скасування наявної ролі користувача;
- експорт даних користувача у CSV-форматі;
- імпорт користувачів у систему з CSV-файлу;

Функції, що має технічний персонал:

- перегляд усіх зареєстрованих у системі рослин;
- реєстрація нової рослини в системі;
- оновлення наявної інформації про рослину;
- видалення рослини з системи;
- додавання нового сенсорного пристрою до системи;
- отримання списку сенсорів, що пов'язані із конкретною рослиною;

- видалення існуючого сенсору;
- додавання нових даних, що були отримані від сенсорів;
- отримання переліку всіх джерел світла;
- додавання нового джерела освітлення;
- видалення джерела світла;
- регулювання яскравості та спектру джерела світла;
- перегляд усіх джерел освітлення, що відносяться до певного сенсору;

Функції, що має звичайний користувач:

- перегляд інформації, зчитаної конкретним сенсором;
- перегляд попереджень, пов'язаних із конкретною рослиною;
- отримання всіх сенсорів, що наявні у системі;

## 3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 3.1 UML проєктування ПЗ

На наступній діаграмі прецедентів можна побачити як користувачі з різними ролями взаємодіють із майбутнім функціоналом (див. рис. 3.1).

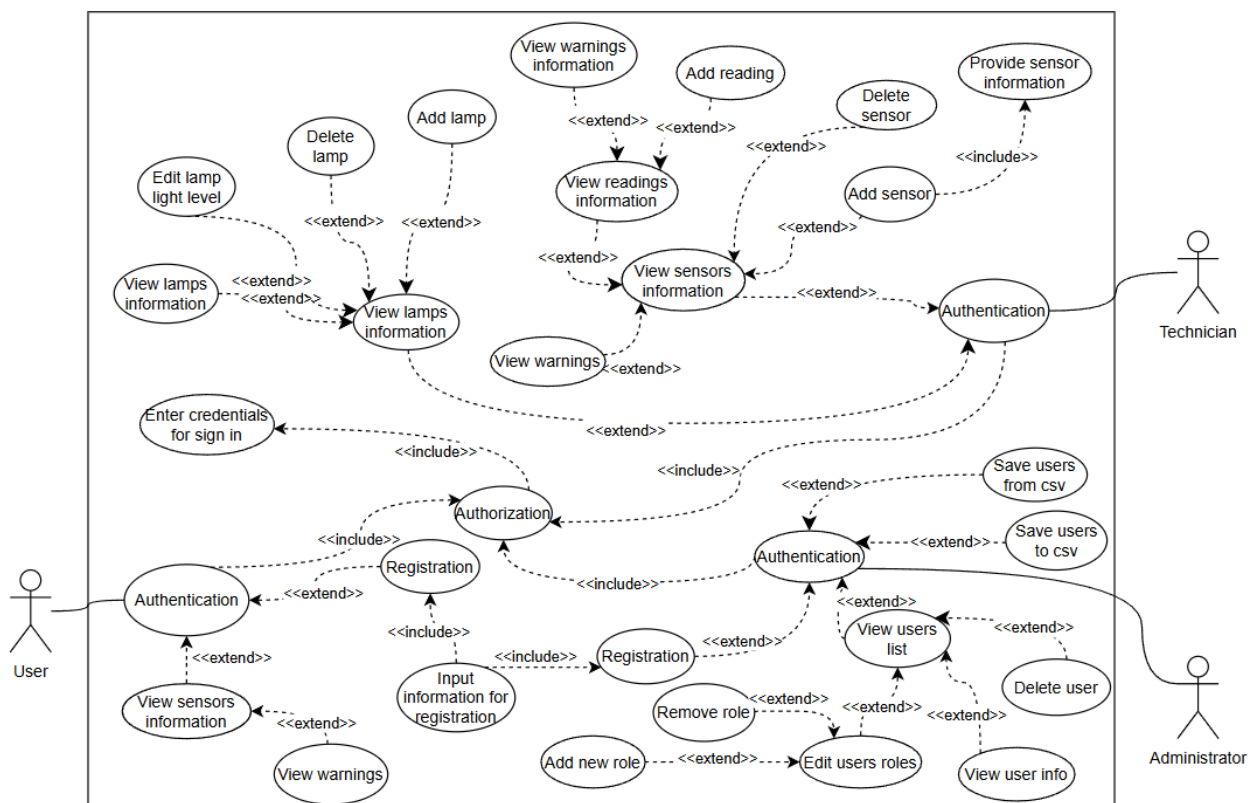


Рисунок 3.1 – Діаграма прецедентів (UML Use Case Diagram) (рисунок виконано самостійно)

На даній діаграмі відображено три основні типи користувачів програмної системи:

- користувач – особа, що має можливість створити обліковий запис, авторизуватися у системі та отримати доступ до списків усіх сенсорів і даних, що були отримані з сенсорів;

– адміністратор – користувач, що відповідає за управління обліковими записами: він може переглядати інформацію про інших користувачів, змінювати їхні ролі, або видаляти облікові записи за потреби;

– технічний працівник – користувач, що займається керуванням інформацією про рослини, сенсори та освітлювальні пристрої: він має можливість переглядати, редагувати, або видаляти відповідні елементи системи;

### 3.2 Проектування архітектури ПЗ

Для розробки програмної системи автоматизованого управління освітленням для вирощування рослин було використано інтегроване середовище розробки IntelliJ IDEA 2024.1.

Серверну частину системи було реалізовано за допомогою мови програмування Java, з активним використанням фреймворку Spring, а саме Spring Boot для швидкої розробки, а також Spring Data JPA для зручної роботи з базами даних.

До складу технологій, що використовуються, входять наступні бібліотеки:

- Hibernate – для реалізації об'єктно-реляційного відображення (ORM);
- OpenCsv – для роботи з CSV-файлами (експорт/імпорт даних);
- Lombok – для зручної генерації коду (до прикладу, getter/setter методів, або конструкторів);
- Spring Doc – для автоматичної генерації документації для API;
- JsonWebToken (JWT) – для безпечної аутентифікації та авторизації користувачів;
- PostgreSQL Driver – для взаємодії з базою даних PostgreSQL, що використовується для збереження та обробки даних.

Для розробки IoT емулятору буде використано мову програмування Java разом з використанням бібліотек Spring, Spring Web, Spring Boot, Lombok.

Серверна частина програмної системи взаємодіятиме з базою даних, PostgreSQL — сучасної реляційної системи управління базами даних, яка вирізняється надійністю, продуктивністю та широкими функціональними можливостями.

Інтеграція PostgreSQL з ORM-фреймворком Hibernate дає змогу спростити обробку даних, зменшуючи кількість рутинного коду й автоматизуючи ключові операції взаємодії з базою. Такий підхід підвищує ефективність системи, сприяє її масштабованості та гарантує стабільну роботу навіть при значному навантаженні.

У структурі бази даних передбачено можливість встановлення зв'язків між таблицями, що дає змогу систематизовано зберігати пов'язані між собою дані. Це сприяє збереженню цілісності інформації, підтримці логічної послідовності та спрощує обробку складних запитів.

Також, наявний емулятор IoT пристрою, що надсилатиме дані на серверну частину, що замінюють справжні смарт-пристрої.

### 3.3 Проектування структури зберігання даних

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

- користувачі (Users): зберігають дані про кожного окремого користувача системи, зокрема, логін, пароль і дату створення облікового запису. Кожен користувач може мати одну або кілька ролей, до прикладу, техніка чи адміністратора;

- ролі (Roles): містять перелік наявних ролей у системі, таких як адміністратор, технік або користувач;

- призначення ролей (UserRoles): представляє взаємозв'язок між користувачами та їхніми ролями в системі;
- рослини (Plants): зберігають відомості про кожну рослину, зокрема її назву, допустимі межі освітлення (мінімальне й максимальне значення) та відсоток росту;
- сенсори (Sensors): містять інформацію про сенсори, які встановлено для моніторингу рослин, зокрема їхню назву та ідентифікатор рослини, з якою вони пов'язані;
- освітлювальні прилади (Lamps): включають дані про джерела освітлення, зокрема назву, спектральні характеристики (ультрафіолетове, червоне та інші) та рівень яскравості, а також інформацію про сенсор, з яким пов'язана кожна лампа;
- зчитування (Readings): зберігають результати зчитування даних із сенсорів – значення, дату й час отримання, а також атрибут, що вказує на наявність попередження;

Визначимо атрибути, характерні для кожної з сутностей:

Користувачі:

- a. #id – унікальний ідентифікатор облікового запису;
- b. login – логін, що використовується для автентифікації у системі;
- c. password – хешований пароль для захисту облікового запису;
- d. created\_at – дата та час, коли був створений обліковий запис;

Ролі:

- a. #id – унікальний ідентифікатор кожної ролі в системі;
- b. name – найменування ролі (адміністратор, користувач, технік);

Ролі користувачів:

- a. #id – унікальний ідентифікатор облікового запису користувача;
- b. role\_id – ідентифікатор ролі, що призначена користувачеві;

Рослини:

- a. #id – унікальний ідентифікатор рослини;

- b. name – назва рослини;
- c. min\_light\_level – мінімальний допустимий рівень освітлення;
- d. max\_light\_level – максимальний допустимий рівень освітлення;
- e. growth\_percentage – відсоток дозрівання рослини;

Сенсори:

- a. #id – унікальний ідентифікатор сенсора;
- b. plant\_id – ідентифікатор рослини, до якої приєднано сенсор;
- c. name – найменування сенсора;

Освітлювальні прилади:

- a. #id – унікальний ідентифікатор джерела світла;
- b. sensor\_id – ідентифікатор сенсора, до якого приєднано освітлювальний прилад;
- c. name – назва освітлювального приладу;
- d. spectrum – тип світла, що випромінює освітлювальний прилад;
- e. light\_level – інтенсивність освітлення, яке видає лампа;

Зчитування:

- a. #id – унікальний ідентифікатор зчитування;
- b. sensor\_id – ідентифікатор сенсора, що здійснив зчитування;
- c. name – найменування зчитування;
- d. value – показник зчитування;
- e. date\_time – дата та час здійснення зчитування;
- f. is\_warning – вказує, чи є вимірювання попередженням;

У базі даних існують такі зв'язки між сутностями:

- a. користувачі та Ролі (UserRoles) мають зв'язок типу "багато до багатьох", оскільки кілька користувачів можуть мати однакові ролі, а також одна роль може бути призначена кільком користувачам;

- b. сенсори (Sensors) та Рослини (Plants) мають зв'язок типу "один до багатьох", оскільки одна рослина може мати декілька сенсорів, але кожен сенсор асоційований лише з однією рослиною;
- c. зчитування (Readings) та Сенсори (Sensors) мають зв'язок типу "один до багатьох", оскільки один сенсор може зібрати багато зчитувань, але кожне зчитування належить тільки до одного сенсора;
- d. освітлювальні прилади (Lamps) та Сенсори (Sensors) мають зв'язок типу "один до багатьох", оскільки кожен сенсор може бути підключений до кількох освітлювальних приладів, але кожен освітлювальний прилад пов'язаний тільки з одним сенсором;

Беручи за основу вищезгадані сутності та зв'язки було створено ER-діаграму (див. рис. 3.2).

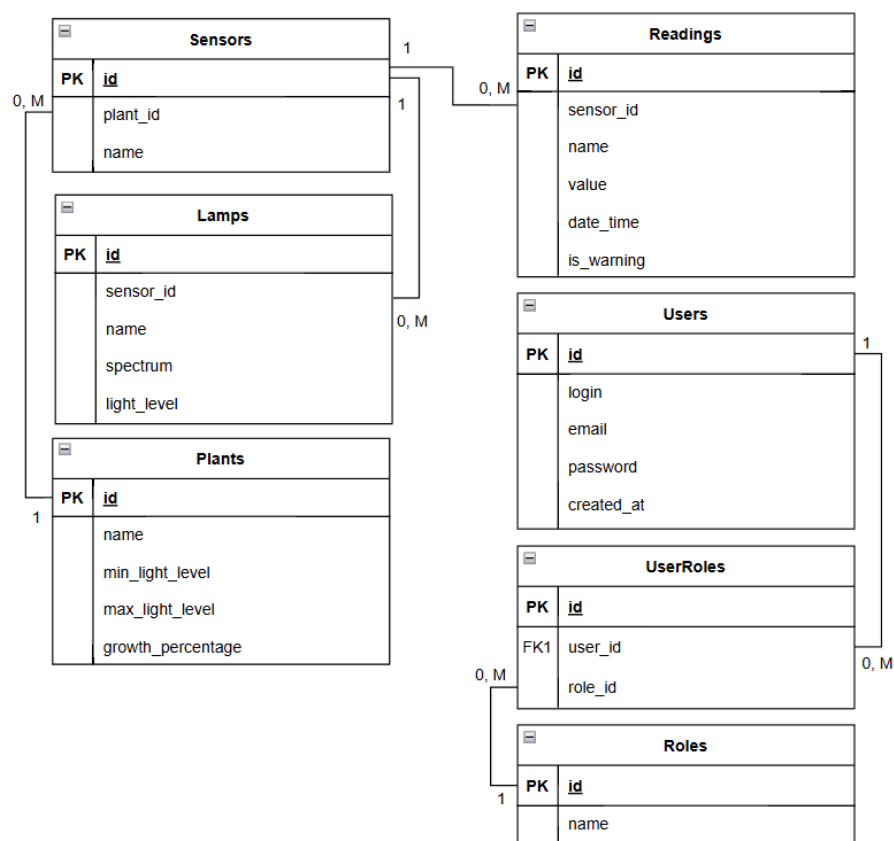


Рисунок 3.2 – Діаграма ER-модель даних (рисунок виконано самостійно)

### 3.4 Найцікавіші алгоритми та методи

Найцікавішими методами в цій програмній системі на мою думку є збереження та завантаження даних користувачів у csv файл (див. рис. 3.2), або з csv файлу (див. рис. 3.3).

```
public void saveUsersToCsv() throws IOException { 1usage
    List<UserEntity> users = userRepository.findAll();

    try (Writer writer = new FileWriter("users.csv");
        CSVWriter csvWriter = new CSVWriter(writer,
            DEFAULT_SEPARATOR,
            NO_QUOTE_CHARACTER,
            DEFAULT_ESCAPE_CHARACTER,
            DEFAULT_LINE_END)) {

        String[] headers = {"login", "password_hash"};
        csvWriter.writeNext(headers);

        // Going through all users in database and save them to csv file.
        for (UserEntity user : users) {
            String[] userData = {
                user.getLogin(),
                user.getPassword()
            };
            csvWriter.writeNext(userData);
        }
        csvWriter.flush();
    }
}
```

Рисунок 3.2 – Збереження користувачів у csv файл (рисунок виконано самостійно)

```
public void saveUsersFromCsv(MultipartFile file) throws IOException, CsvValidationException { 1 usage
    try (Reader reader = new InputStreamReader(file.getInputStream());
        CSVReader csvReader = new CSVReader(reader)) {

        String[] nextRecord;
        // Skip headers
        csvReader.readNext();
        while ((nextRecord = csvReader.readNext()) != null) {
            // Process each row and create UserEntity objects.
            // If user already exists, then skipping him
            if (userRepository.findByLogin(nextRecord[0]).isPresent()) {
                continue;
            }

            UserEntity user = new UserEntity(nextRecord[0], nextRecord[1]);

            userRepository.save(user);
        }
    }
}
```

Рисунок 3.3 – Відновлення користувачів з csv файлу (рисунок виконано самостійно)

В цьому коді можна побачити, що алгоритм було створено самостійно для імпорту та експорту даних користувачів.

## 4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

### 4.1 Використання JWT

У розробленій програмній системі для реалізації автентифікації користувачів застосовується технологія JWT (JSON Web Token).

JWT — це відкритий стандарт (RFC 7519) [11], який дозволяє безпечно передавати затверджену інформацію між сторонами як JSON-об'єкт.

Після успішного входу користувача в систему сервер формує та підписує спеціальний токен, що містить основну інформацію про користувача (ідентифікатор, ролі, дату створення та інші необхідні дані).

Цей токен відправляється клієнтові, який зберігає його, як правило, у локальному сховищі (LocalStorage або sessionStorage), і додає до кожного подальшого запиту в заголовок Authorization.

JWT складається з трьох частин:

- Header — заголовок, що вказує тип токена та алгоритм підпису (в нашому випадку, HS256);

- Payload — містяться дані (claims) користувача;

- Signature — криптографічний підпис, що забезпечує цілісність токена;

Застосування JWT дозволяє:

- зменшити навантаження на сервер, оскільки інформація користувача не зберігається на стороні сервера;

- забезпечити масштабованість системи;

- інтегрувати автентифікацію в RESTful API.

А у поєднанні з фреймворком Spring Security, JWT надає механізм керування доступом до ресурсів залежно від ролі користувача (див. рис. 4.1).

```

Configures the security filter chain defining authentication, authorization rules, exception handling,
and filters for HTTP requests.

Params: http – HttpSecurity object to configure security settings

Returns: SecurityFilterChain defining security configurations

Throws: Exception – if an error occurs during configuration

@Bean @Cheese *
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .csrf(AbstractHttpConfigurer::disable)
        .authorizeHttpRequests(authorization -> authorization
            .requestMatchers(Ⓢ "/swagger-ui/**", Ⓢ "/v3/api-docs/**",
                Ⓢ "/register", Ⓢ "/login").permitAll()
            .requestMatchers(Ⓢ "/users/**").hasAuthority("ADMIN")
            .requestMatchers(Ⓢ "/sensors/**", Ⓢ "/lamps/**", Ⓢ "/plants/**").hasAuthority("TECHNICIAN")
            .requestMatchers(Ⓢ "/readings/**", Ⓢ "/me/**").authenticated())
        .sessionManagement(management -> management
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authenticationProvider(authenticationProvider)
        .exceptionHandling(httpSecurityExceptionHandlerConfigurer -> httpSecurityExceptionHandlerConfigurer
            .authenticationEntryPoint(exceptionHandlerController)
            .accessDeniedHandler(exceptionHandlerController))
        .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}

```

Рисунок 4.1 – Конфігурація керування доступом до ресурсів у залежності від ролі користувача (рисунок виконано самостійно)

Тут ми бачимо, що певні ресурси доступні користувачам лише при умові, що вони мають необхідну роль, розпишемо детальніше, які необхідно мати ролі для доступу до певних ресурсів.

Ресурси `/swagger-ui`, `/v3/api-docs`, `/register`, `/login` – доступні усім користувачам.

Ресурси `/readings`, `/me` – доступні усім користувачам, що аутентифіковані у програмній системі.

Ресурси `/sensors`, `/lamps`, `/plants` – доступні аутентифікованим користувачам, що мають роль технічного персоналу.

Ресурс `/users` – доступний аутентифікованим користувачам, що мають роль адміністратора.

## 4.2 Обробка помилок

Для покращення надійності та зручності використання програмної системи, в проєкті реалізовано централізовану обробку помилок.

Основну роль у цьому відіграє клас `CustomExceptionHandlerController`, який відповідає за перехоплення винятків, що виникають під час виконання запитів, та повернення зрозумілих і локалізованих повідомлень користувачеві.

Клас `CustomExceptionHandlerController` реалізує централізовану обробку наступних типів помилок:

- автентифікація та авторизація: якщо користувач неавторизований або намагається отримати доступ до захищених ресурсів без відповідних прав, система повертає відповідні повідомлення з статус кодами 401 `Unauthorized` або 403 `Forbidden`.

- обробка бізнес-винятків: програмна система також оброблює помилки, що пов'язані з бізнес-логікою, до прикладу, якщо користувач намагається створити об'єкт, з ідентифікатором, що вже існує в базі даних (`EntityExistsException`, `RoleAlreadyExistsException`), або користувач намагається оновити об'єкт, що не був знайдений за вказаним ідентифікатором (`EntityNotFoundException`), користувач отримує відповідь з кодами 409 `Conflict` або 404 `Not Found`, відповідно.

- синтаксичні помилки у вхідних JSON-даних: у випадку, якщо запит містить некоректні або нечитасмі JSON-дані (`HttpMessageNotReadableException`), повертається помилка з кодом 422 `Unprocessable Entity`

Клас анотовано як `@RestController` і `@ControllerAdvice`, що дозволяє автоматично перехоплювати винятки по всій системі. Реалізовано інтерфейси `AuthenticationEntryPoint` та `AccessDeniedHandler`, щоб керувати помилками `Spring Security`. Відповіді на помилки отримуються з `MessageSource`, що забезпечує локалізацію відповідей відповідно до поточної мови користувача.

### 4.3 Структура проєкту

У процесі розробки програмної системи було дотримано принципів багаторівневої архітектури, що забезпечує зручну підтримку, масштабованість та розділення відповідальностей. Проєкт має наступну структуру директорій (див. рис. 4.2):

- `config`: ця директорія містить конфігураційні класи, які відповідають за налаштування безпеки, доступу до ресурсів, `cors`-політики, а також іншого;

- `controllers`: у цій директорії зберігаються REST-контролери, які відповідають за обробку HTTP-запитів. Вони є посередниками між клієнтською частиною (до прикладу, інтерфейсом користувача) та бізнес-логікою. Контролери викликають відповідні сервіси, оброблюють вхідні дані та повертають результати у вигляді HTTP-відповідей;

- `dtos`: ця директорія містить об'єкти для передачі даних між клієнтом і сервером. DTO використовуються для інкапсуляції необхідної інформації, виключаючи зайві або чутливі дані. Це допомагає захистити модель даних та контролювати формат обміну;

- `exceptions`: ця директорія містить в собі створені мною винятки, для особливих ситуацій, в яких не вистачає стандартного набору винятків, а саме `RoleAlreadyExistsException` – для випадку, коли адміністратор намагається додати користувачу роль, що в нього вже є, або `UserNotContainsRoleException` – для протилежного випадку, коли адміністратор намагається забрати роль у користувача, коли користувач її не має;

- `jwt`: ця директорія реалізує логіку роботи з JWT. Вона включає класи для генерації та валідації токенів, а також фільтрації запитів для перевірки аутентифікації;

- `models`: у цій директорії розташовані сутності (`Entity`), які відповідають таблицям бази даних. Вони описують структуру даних, які зберігаються та

обробляються системою (User (див. рис. 4.3), Sensor, Lamp, Reading тощо). Всі сутності позначені анотаціями JPA (@Entity, @Table, @Id тощо);

– services: у цій директорії зосереджена бізнес-логіка програми. Сервісні класи реалізують основні функції системи, а саме, взаємодію з репозиторіями, обробку даних, перевірку об'єктів тощо. Вони ізольовані від контролерів і забезпечують повторне використання логіки;

– repositories: тут містяться інтерфейси, які відповідають за доступ до бази даних. Вони розширюють JpaRepository і надають методи для стандартних CRUD-операцій, а також можуть містити власні запити (див. рис. 4.4);

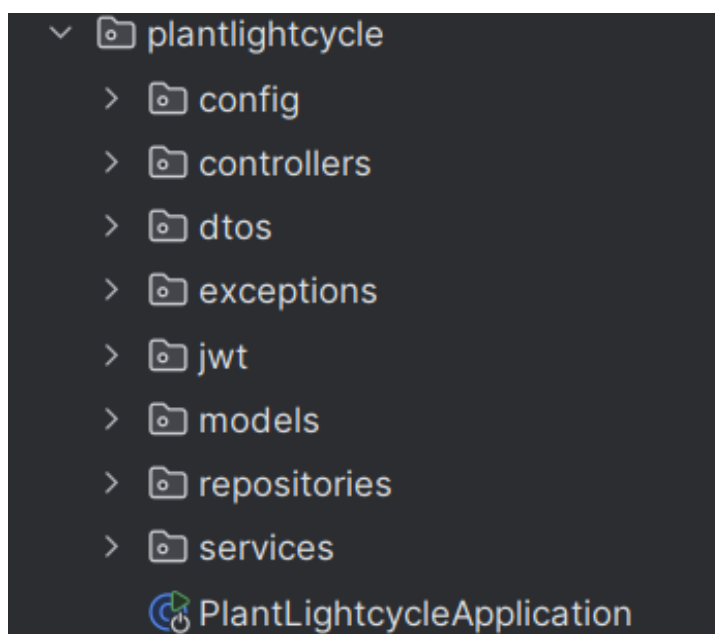


Рисунок 4.2 – Структура проекту (рисунок виконано самостійно)

```

Represents a user entity within the system. Implements UserDetails for Spring Security integration.

@Entity 32 usages  ▲ Cheese *
@Table(name = "user")
@NoArgsConstructor
@Getter
@Setter
public class UserEntity implements UserDetails {
    | Unique identifier for the user entity.
    |
    | @Id
    | @GeneratedValue
    | @JsonIgnore
    | private Long id;
    |
    | User's login name.
    |
    | private String login;
    |
    | User's password.
    |
    | @JsonIgnore
    | private String password;
    |
    | Date and time of user creation.
    |
    | private LocalDateTime createdAt;
    |
    | Set of roles associated with the user.

```

Рисунок 4.3 – Приклад описання сутності у вигляді класу (рисунок виконано самостійно)

```

Repository interface for LampEntity to perform CRUD operations.

@Repository 2 usages  ▲ Cheese
public interface LampRepository extends JpaRepository<LampEntity, Long> {
    | Retrieves a lamp by its name.
    |
    | Params: name - The name of the lamp to retrieve.
    | Returns: The LampEntity with the specified name.
    |
    | LampEntity findByName(String name); 1 usage  ▲ Cheese
    |
    | Retrieves all lamps associated with a specific sensor ID.
    |
    | Params: sensorId - The ID of the sensor to retrieve lamps for.
    | Returns: A list of LampEntity objects associated with the specified sensor ID.
    |
    | List<LampEntity> findAllBySensorId(Long sensorId); 1 usage  ▲ Cheese
}

```

Рисунок 4.4 – Приклад власних запитів findByName та findAllBySensorId (рисунок виконано самостійно)

На цьому прикладі можна побачити реалізацію лише для освітлювальних приладів, проте для інших частин системи код виглядає схожим чином.

## 5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 5.1 Ручне тестування

У процесі тестування реалізованої інформаційної системи було перевірено коректність виконання основних функціональних можливостей для кожного з типів користувачів: адміністратора, технічного персоналу та звичайного користувача.

Тестування проводилося вручну (manual testing) через swagger [12].

Для забезпечення безпеки доступу до функціоналу було реалізовано JWT-автентифікацію та авторизацію з розмежуванням ролей.

Правильність роботи фільтра JwtAuthenticationFilter перевірялась при кожному запиті до захищених ресурсів, а обробка помилок тестувалась за допомогою неправильних запитів (до прикладу, невалідні дані, спроба доступу без токена, проблеми з формуванням JSON тощо).

В першу чергу було протестовано обробку типових помилок:

- спроба доступу до захищених ресурсів без токена;
- неправильний логін чи пароль;
- відмова доступу до ресурсів без необхідної ролі;
- проблема з формуванням JSON;
- спроба створення дублікату об'єкту;
- спроба отримання або зміни неіснуючого об'єкту;
- спроба видати роль користувачу, яка в нього вже є, або забрати роль у користувача, якої у нього немає;

Після цього було протестовано основні функціональні можливості:

Для адміністратора:

- успішний вхід до системи із видачею JWT токена;
- отримання списку користувачів;
- перегляд детальної інформації про окремого користувача;

- видалення користувачів та перевірка їхньої відсутності після запиту;
- видача та скасування ролей;
- експорт даних у CSV;
- імпорт користувачів з CSV-файлу;

Для технічного персоналу:

- перегляд усіх рослин;
- додавання нової рослини, оновлення існуючої, видалення;
- робота з сенсорами: додавання, прив'язка до рослин, перегляд за ідентифікатором, видалення;

Збереження нових зчитувань, отриманих від сенсорів;

- перегляд джерел світла, додавання, редагування параметрів яскравості/спектру;

Для звичайного користувача:

- перегляд показників з окремого сенсора;
- перегляд попереджень, пов'язаних з рослиною;
- отримання переліку всіх сенсорів у системі.

## ВИСНОВКИ

У межах виконання кваліфікаційної роботи було реалізовано серверну частину програмної системи, призначеної для автоматизованого управління освітленням у середовищі вирощування рослин.

Реалізація програмного продукту здійснювалась з використанням мови програмування Java, фреймворку Spring Boot та бібліотеки Hibernate, що забезпечили масштабованість, надійність і зручність подальшого супроводу системи.

У якості системи управління базами даних було використано PostgreSQL, що дозволяє організувати збереження інформації з дотриманням принципів узгодженості, цілісності та ефективного доступу до даних.

Модель даних було побудовано з урахуванням реляційних зв'язків між сутностями предметної області.

У результаті реалізації було створено систему, що підтримує рольову модель доступу (адміністратор, технік, користувач), забезпечує обробку даних про користувачів, рослини, сенсори, освітлювальні пристрої та зчитування, а також передбачає можливість імпорту та експорту інформації у форматі CSV.

Отже, було створено функціональне серверне рішення, яке є основою для створення сучасної інформаційної системи автоматизованого моніторингу та регулювання освітлювання при вирощуванні рослин.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Документація PostgreSQL [Електронний ресурс] – URL: <https://www.postgresql.org/docs/> (дата звернення 30.04.2025)
2. Документація Maven [Електронний ресурс] – URL: <https://maven.apache.org/guides/index.html> (дата звернення 30.04.2025)
3. Посібник з практичного використання Spring Boot [Електронний ресурс] – URL: <https://www.geeksforgeeks.org/spring-boot/> (дата звернення 30.04.2025)
4. Офіційний сайт Sollum Technologies [Електронний ресурс] – URL: <https://www.sollumtechnologies.com/> (дата звернення 12.04.2025)
5. Офіційний сайт Heliospectra [Електронний ресурс] – URL: <https://heliospectra.com/> (дата звернення 12.04.2025)
6. Офіційний сайт Bioled [Електронний ресурс] – URL: <https://www.bioled.co.il/> (дата звернення 12.04.2025)
7. Офіційний сайт Kroptek [Електронний ресурс] – URL: <https://kroptek.com/> (дата звернення 12.04.2025)
8. Найкращі практики в дизайні REST API [Електронний ресурс] – URL: <https://swagger.io/resources/articles/best-practices-in-api-design/> (дата звернення 30.04.2025)
9. Бауэр К., Кінг Г., Грегорі Д. Java Persistence with Hibernate. 2-ге вид. – Greenwich: Manning Publications, 2015. – 608 с.
10. Посібник для початківців з розуміння протоколу HTTP [Електронний ресурс] – URL: [https://dev.to/carrie\\_luo1/the-beginners-guide-to-understanding-http-protocol-5e9](https://dev.to/carrie_luo1/the-beginners-guide-to-understanding-http-protocol-5e9) (дата звернення 30.04.2025)
11. Стандарт RFC 7519 [Електронний ресурс] – URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення 30.04.2025)

12. Інструкція по додаванню swagger в програмну систему [Електронний ресурс] – URL:

<https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api>

13.

Github.

NureOstroverkhovYehor/2025\_B\_PI\_PZPI-21-10\_Ostroverkhov\_Y\_A. URL:

[https://github.com/NureOstroverkhovYehor/2025\\_B\\_PI\\_PZPI-21-10\\_Ostroverkhov\\_Y\\_A](https://github.com/NureOstroverkhovYehor/2025_B_PI_PZPI-21-10_Ostroverkhov_Y_A)