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. Back-end
(тема)

Виконав:
здобувач 4 року навчання,
групи ПЗПІ-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 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 Островерхову Єгору Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для автоматизованого управління освітленням для вирощування рослин. Back-end

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	
9	Здача роботи у електронний архів	04.06.2025	
9	Допуск до захисту у зав. кафедри	07.06.2025	
10	Попередній захист	08.06.2025	

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 для передатестаційної практики: 56 сторінок, 24 рисунки, 12 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 проєктування 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 система управління базами даних PostgreSQL. Також розроблено смарт-пристрій у вигляді емулятора на Java, до якого можна динамічно 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 сільськогосподарських об’єктах, таких як ферми та плантації. До складу системи входитиме сервер для

зберігання й обробки інформації, а також смарт-пристрій, що відстежуватиме параметри стану рослин.

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

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 рослини, освітлювальних приладів та датчиків до програмної системи;

– налаштування рівня яскравості та кольорного 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 [4] (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.

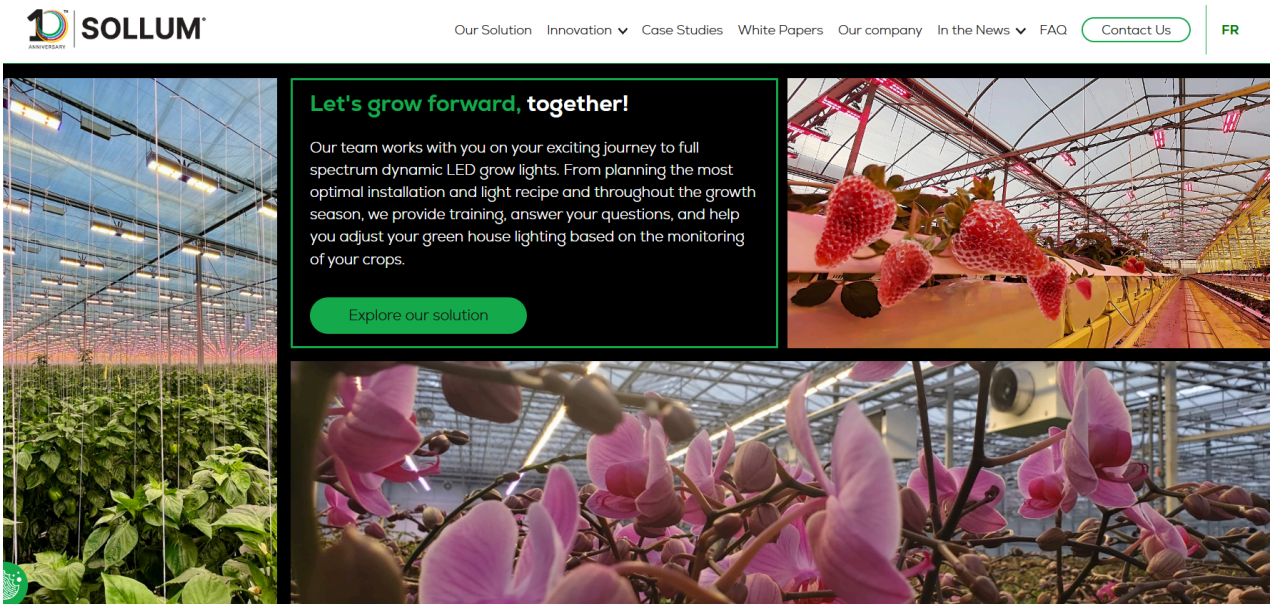


Рисунок 1.1 – Головна сторінка сайту Sollum Technologies

Heliospectra [5] (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 малоадаптоваим до невеликих ферм, а також складна інтеграція з різними типами датчиків, що не відносяться до Heliospectra.

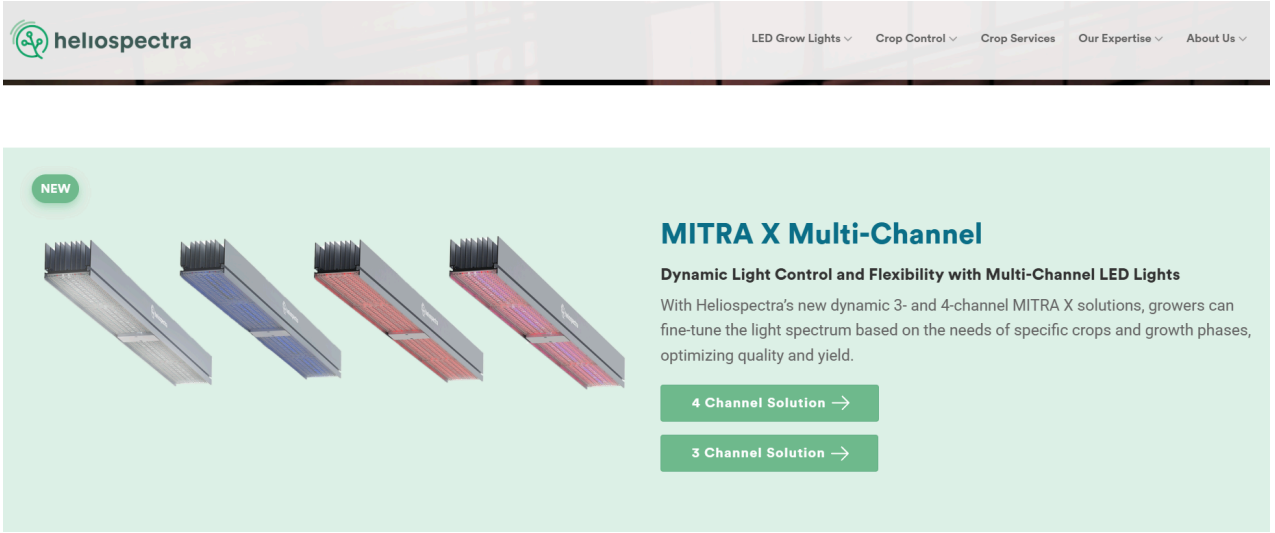


Рисунок 1.2 – Головна сторінка сайту Heliospectra

Bioled [6] (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 унеможливлює роботу для недосвідченного користувача.

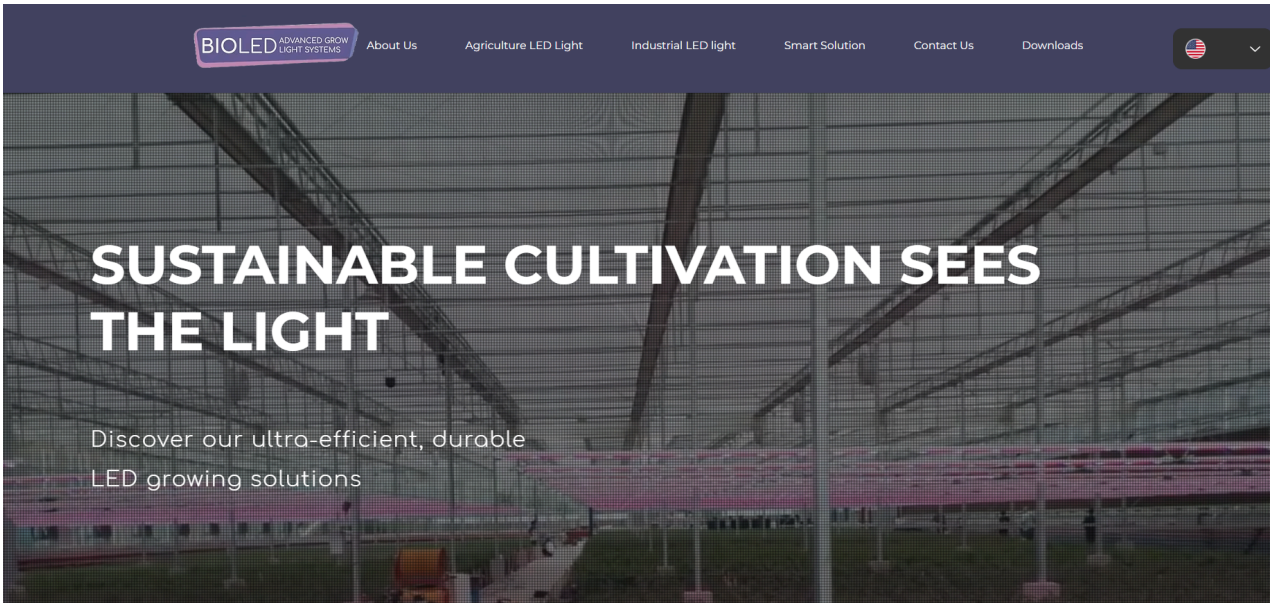


Рисунок 1.3 – Головна сторінка сайту Bioled

Kroptek [7] (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

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

Таблиця 1.1 – Порівняння аналогів

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 проєктом 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:

– Back-end 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 проєкту.

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).

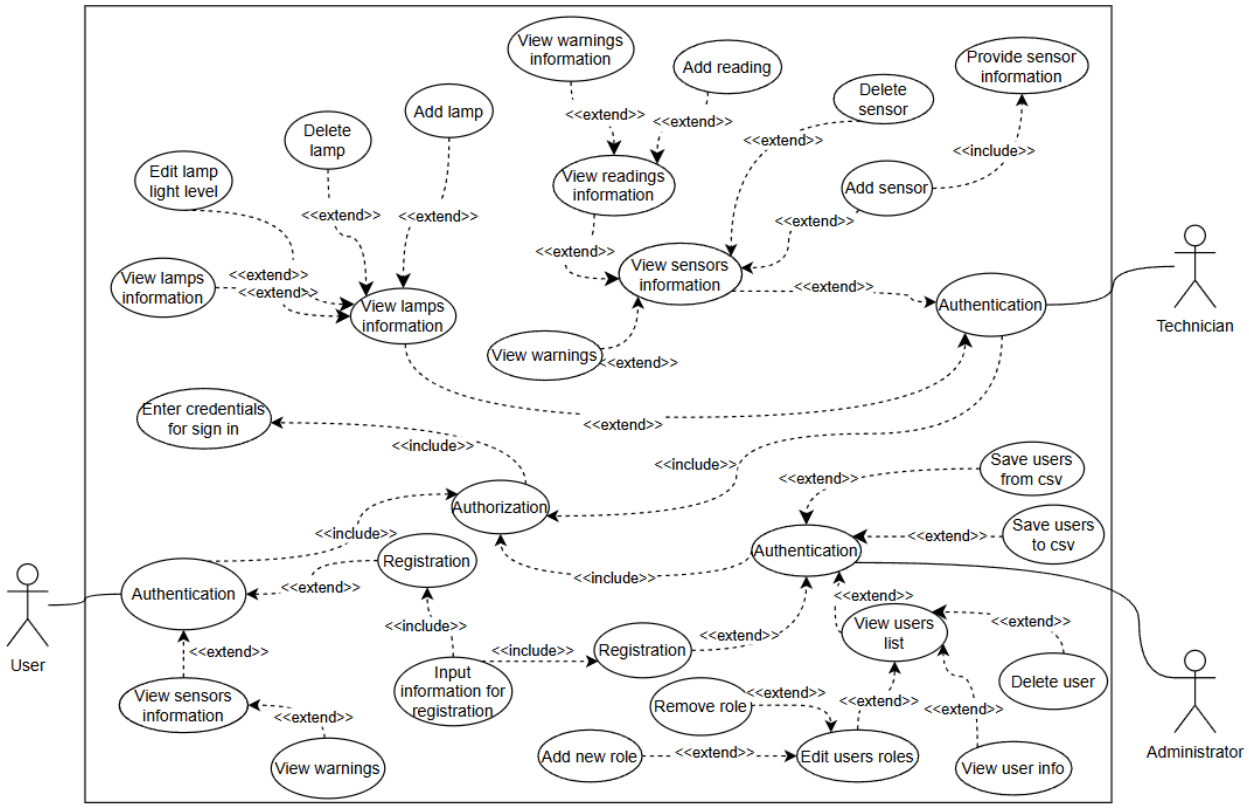


Рисунок 3.1 – Діаграма прецедентів (UML Use Case Diagram)

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 Проєктування 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 заміняють справжні смарт-пристрої.

3.3 Проєктування 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;

– освтілювальні 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).

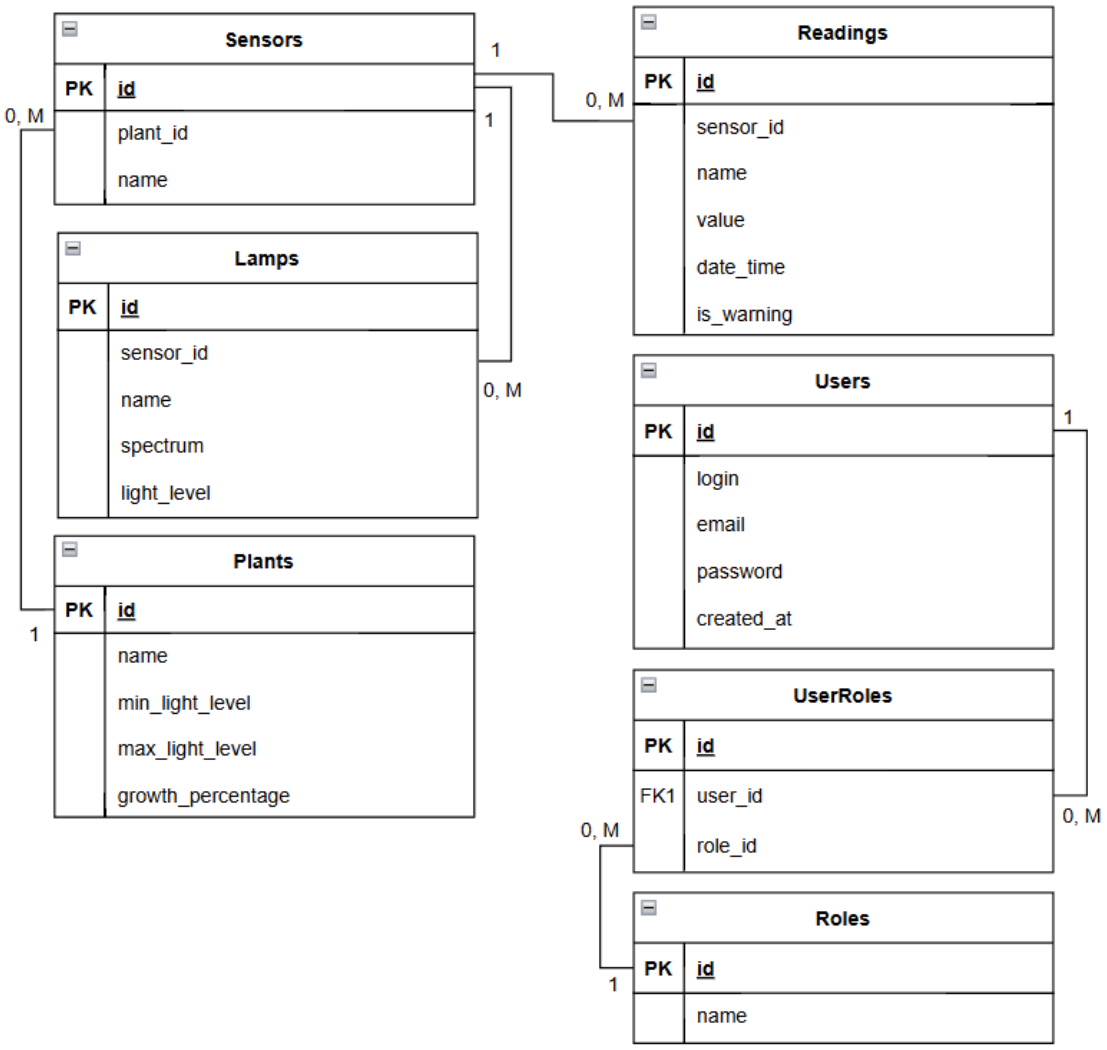


Рисунок 3.2 – Діаграма ER-модель даних

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 {  1 usage
    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

```
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

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 заголовку 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;

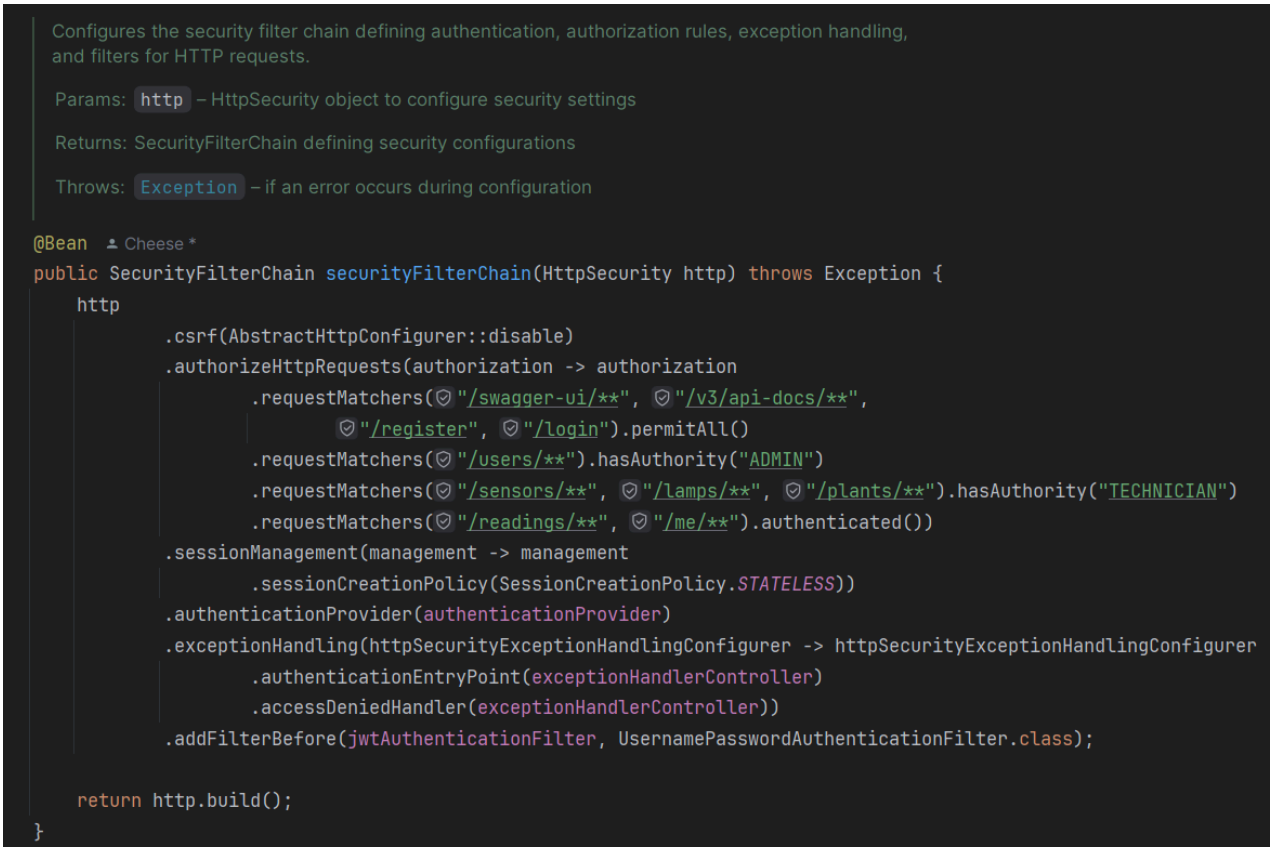
Застосування JWT дозволяє:

– 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(httpSecurityExceptionHandlingConfigurer -> httpSecurityExceptionHandlingConfigurer
                    .authenticationEntryPoint(exceptionHandlerController)
                    .accessDeniedHandler(exceptionHandlerController))
            .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}
```

Рисунок 4.1 – Конфігурація керування доступом до ресурсів у залежності від ролі користувача

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 нечитаємі 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);

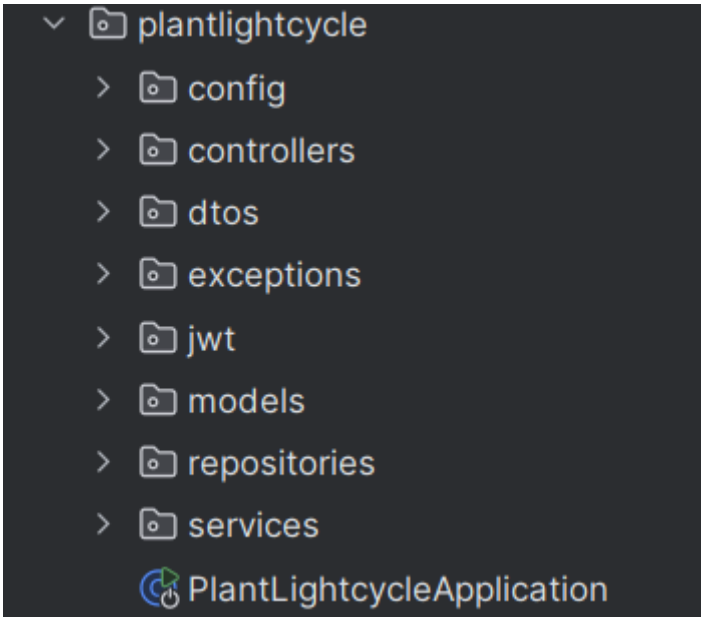


Рисунок 4.2 – Структура проєкту

```java
/** 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

```java
/** 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

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 токену;

– 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 URL: https://www.postgresql.org/docs/

2. Документація Maven URL: https://maven.apache.org/guides/index.html

3. Посібник з практичного використання Spring Boot URL: https://www.geeksforgeeks.org/spring-boot/

4. Офіційний сайт Sollum Technologies URL: https://www.sollumtechnologies.com/

5. Офіційний сайт Heliospectra URL: https://heliospectra.com/

6. Офіційний сайт Bioled URL: https://www.bioled.co.il/

7. Офіційний сайт Kroptek URL: https://kroptek.com/

8. Найкращі практики в дизайні REST API URL: https://swagger.io/resources/articles/best-practices-in-api-design/

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 URL: https://dev.to/carrie_luo1/the-beginners-guide-to-understanding-http-protocol-5e9

11. Стандарт RFC 7519 URL: https://datatracker.ietf.org/doc/html/rfc7519

12. Інструкція по додаванню swagger в програмну систему URL: https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту

StrikePlagiarism.com

Дата звіту 5/29/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_Б_ПІ_ПЗПІ-21-10_Островерхов_Є_А_скорочений
Автор Науковий керівник / Експерт
Островерхов Єгор Андрійович Євген Кардаш
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

1.14% 1.14% КП 1

0.38% 0.38% КЦ

25 Довжина фрази для коефіцієнта подібності 2
3425 Кількість слів
28148 Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	0
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	¶	0
Парафрази (SmartMarks)	a	3

Подібності за списком джерел

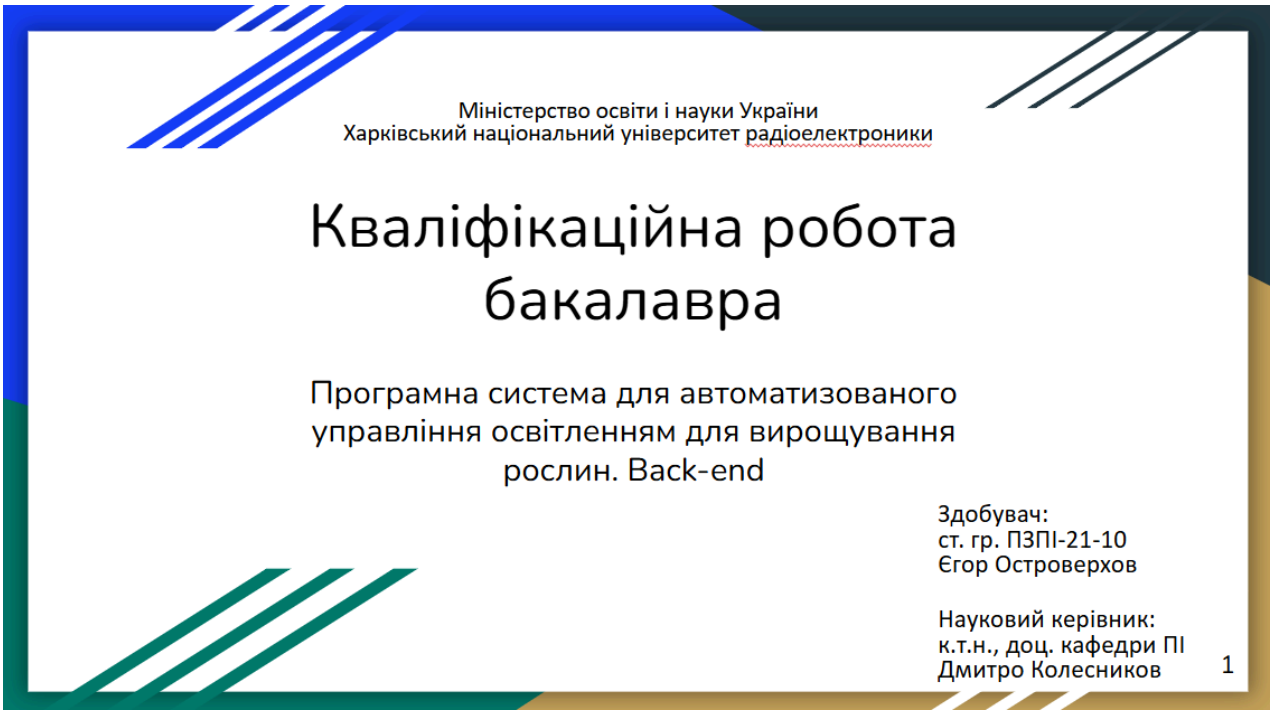
Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

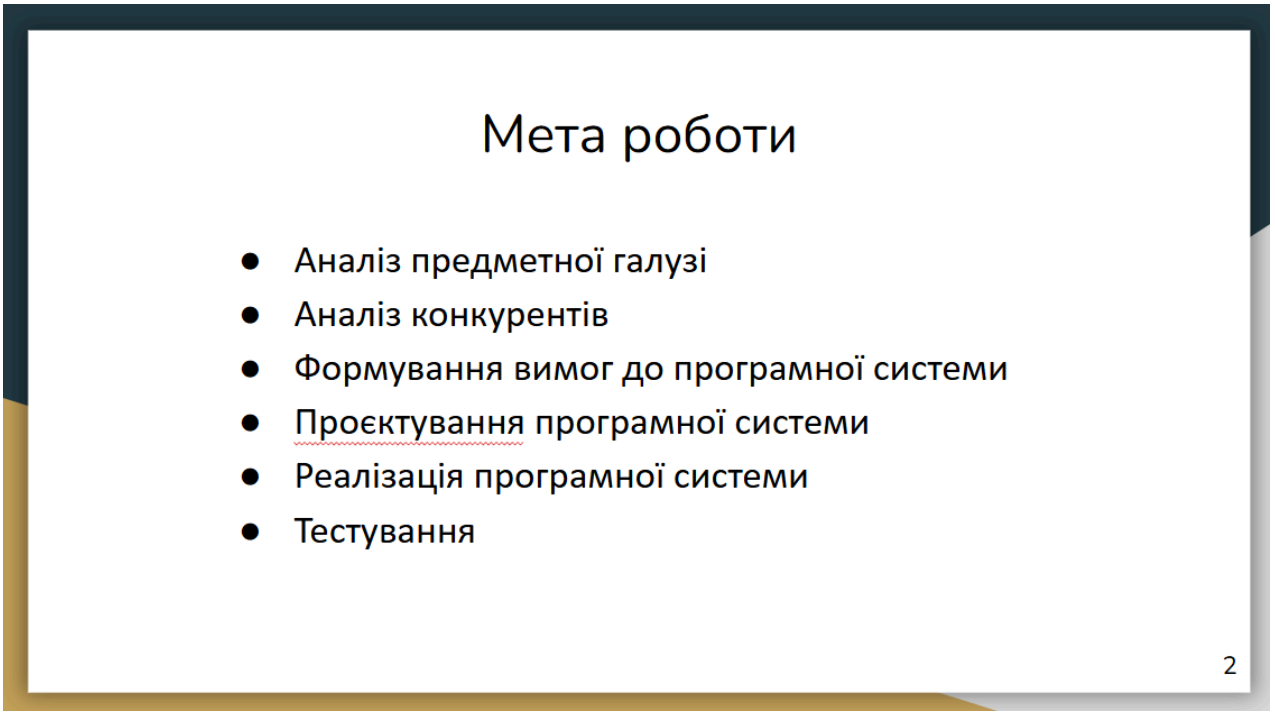
10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://openarchive.nure.ua/bitstreams/dcb865f6-766f-4683-a4bb-5cb937468862/download	11 0.32 %
2	https://ela.kpi.ua/bitstreams/5d896f18-05a7-4655-a184-eda579e3e2fe/download	10 0.29 %
3	ЗАХИСТ ІНТЕРНЕТ-МАГАЗИНУ ГАДЖЕТІВ ВІД КІБЕРАТАК [redacted] 5/29/2024 International University of Economics and Humanities named after Academician Stepan Demianchuk (International University of Economics and Humanities named after Academician Stepan Demianchuk)	10 0.29 %

ДОДАТОК Б

Слайди презентації





Аналіз предметної галузі

- Зростання інтересу до автоматизованих систем освітлення
- Переважно статичне керування освітленням
- Відсутність адаптації до змін у середовищі вирощування рослин

3

Існуючі рішення

Характеристика	Sollum Technologies	Heliospectra	Bioled	Kroptek
Інноваційні, сучасні рішення	+	+	–	+
Енергоефективність	–	+	+	+
Автоматизація освітлення	+	+	–	–
Інтеграція з сенсорами	+	–	–	–
Підтримка мобільного/веб-інтерфейсу	+	+	–	+
Динамічна зміна освітлення за кліматичними параметрами	+	+	–	–

4

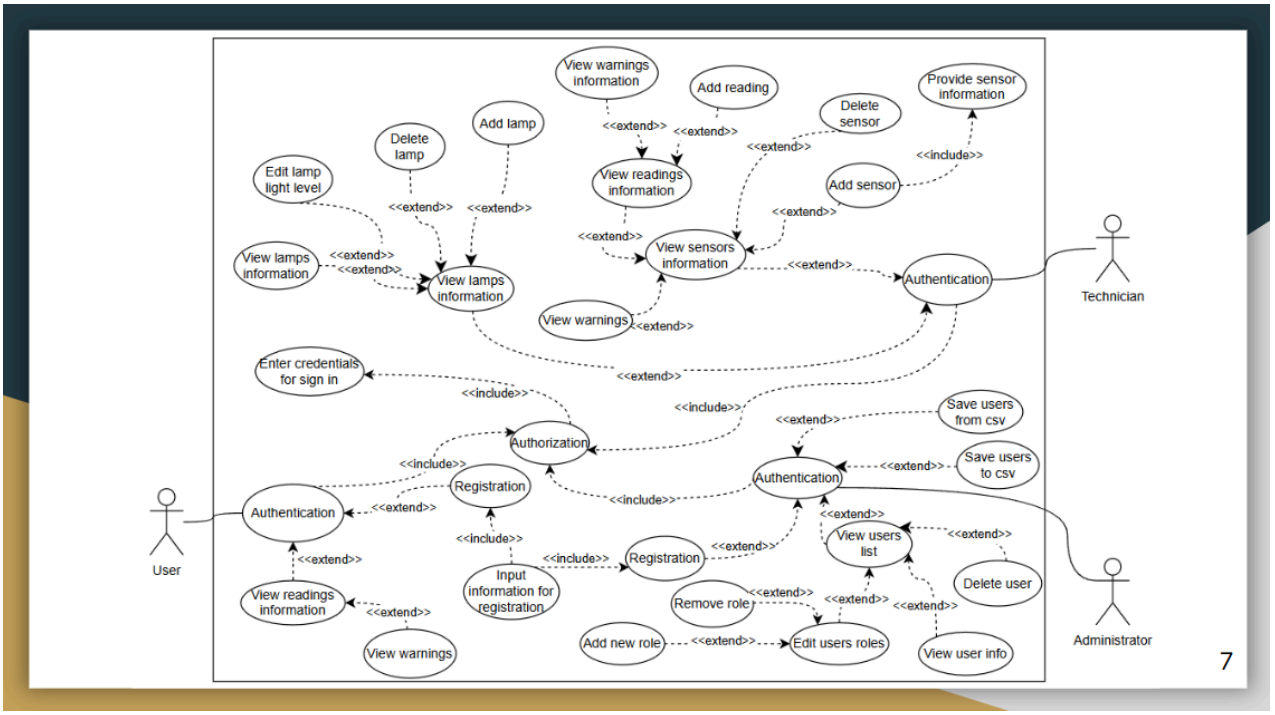
Постановка задачі

- Розробка серверної частини
- Реалізація REST API
- Авторизація користувачів
- Зберігання даних
- Обробка даних в залежності від показників
- Резервне копіювання

5

Інструменти розробки

6



7

Приклад коду контролера

```java
@RestController  Cheese
@RequestMapping("/lamps")
@RequiredArgsConstructor
public class LampController {
    Service handling lamp-related operations.

    private final LampService lampService;

    Endpoint to retrieve all lamps.
    Returns: ResponseEntity containing the list of all lamps

    @GetMapping  Cheese
    public ResponseEntity<List<LampEntity>> getAllLamps() {
        return ResponseEntity.ok(lampService.getAllLamps());
    }
```

8

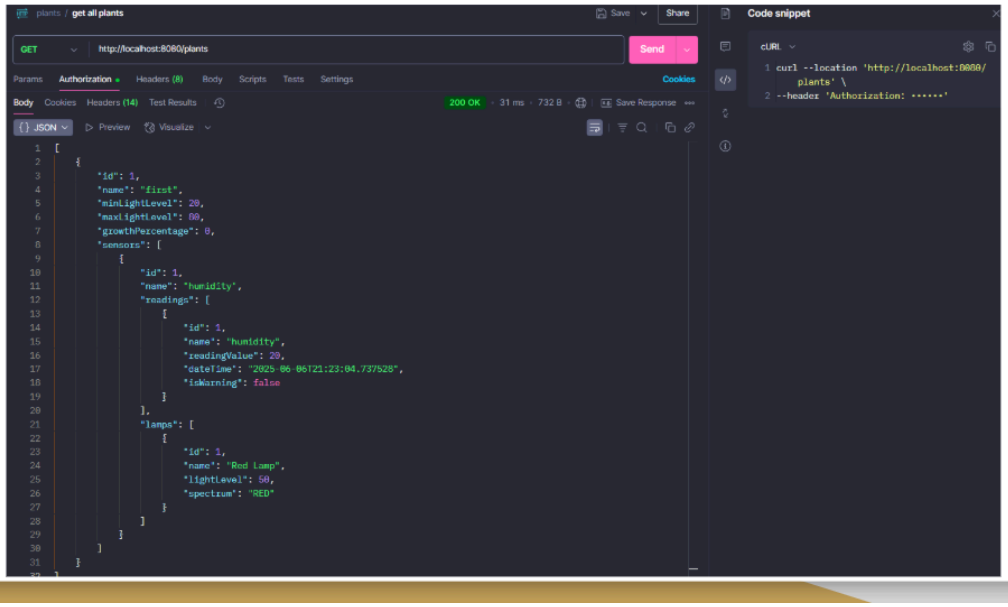
Приклад коду з сервісу

```
    Retrieves all lamps associated with a specific sensor.
    Params: sensorId – The ID of the sensor.
    Returns: A list of LampEntity objects associated with the specified sensor ID.
public List<LampEntity> getAllSensorLamps(Long sensorId) {  1 usage   Cheese
    return lampRepository.findAllBySensorId(sensorId);
}

    Saves a new lamp associated with a specific sensor.
    Params: lampDto – The LampDto object containing lamp details.
    Returns: A set of LampEntity objects associated with the sensor after saving the new lamp.
public Set<LampEntity> saveLamp(LampDto lampDto) {  1 usage   Cheese
    SensorEntity sensor = sensorService.getSensorById(lampDto.sensorId());

    LampEntity lamp = new LampEntity(lampDto.name(), lampDto.lightLevel(), lampDto.spectrum());

    lamp.setSensor(sensor);
    sensor.addLamp(lamp);

    return sensorService.saveSensor(sensor).getLamps();
}
```


Приклад репозиторію

```
    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
}
```

Тестування


plants / get all plants
Save
Share
GET
http://localhost:8080/plants
Send
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
Cookies
Body
Cookies
Headers (14)
Test Results
200 OK
31 ms
732 B
Save Response
JSON
Preview
Visualize
Code snippet
cURL
curl --location 'http://localhost:8080/plants' \
--header 'Authorization: ******'

ДОДАТОК В
Специфікація ПЗ

1 ВСТУП

1.1 Огляд продукту

Розроблений програмний продукт є серверною системою управління освітленням для вирощування рослин. Він забезпечує автоматичне регулювання освітлення на основі даних сенсорів, стану рослин та параметрів навколишнього середовища. Система включає функції авторизації, управління користувачами, пристроями та базою даних, з можливістю резервного копіювання і масштабування.

1.2 Мета

Метою даного продукту є автоматизація керування освітленням для оптимального вирощування рослин на основі даних з сенсорів.

1.3 Межі

Серверна частина:

– розроблена з використанням Java 17 та Spring Boot;

– реалізоване REST API для обміну даними між клієнтами, сенсорами та пристроями освітлення;

– використовується PostgreSQL для зберігання даних про рослини, сенсори, джерела світла та користувачів;

– реалізована авторизація користувачів із підтримкою ролей (адміністратор, технічний персонал, звичайний користувач);

– забезпечено обробку сенсорних даних у реальному часі для динамічного керування освітленням;

– впроваджено механізми резервного копіювання та відновлення даних;

– використано Maven для керування залежностями, збірки;

Інтеграція з пристроями:

– прийом та збереження показників від сенсорів (вологість, температура тощо);

– взаємодія із джерелами світла для зміни яскравості та спектру згідно з отриманими параметрами;

1.4 Посилання

В проєкті були використані такі посилання:

– https://www.geeksforgeeks.org/spring-boot/

– https://swagger.io/resources/articles/best-practices-in-api-design/

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

1.5 Означення та абревіатури

Користувач – особа, яка взаємодіє з програмною системою через веб-інтерфейс або пристрої введення даних (сенсори). Може мати одну з ролей: адміністратор, технічний персонал або звичайний користувач.

Сервер – програмна система, що обробляє запити, виконує бізнес-логіку системи та зберігає інформацію у базі даних.

Сенсор – пристрій, що зчитує параметри навколишнього середовища (вологість, температура, освітленість) і надсилає їх до серверної частини системи.

Джерело світла або Лампа – електронний пристрій, який забезпечує освітлення для рослин. Може керуватися системою автоматично на основі показників сенсорів.

API (Application Programming Interface) – інтерфейс програмування застосунків, через який інші системи та пристрої взаємодіють із сервером.

CRUD (Create, Read, Update, Delete) – базові операції для створення, зчитування, оновлення та видалення даних у системі.

CSV (Comma-Separated Values) – формат табличного зберігання та обміну даними, що використовується для імпорту та експорту користувачів у системі.

REST (Representational State Transfer) – архітектурний стиль для розробки веб-сервісів, використаний у створенні серверного API.

DB (Database) – база даних PostgreSQL, яка зберігає всю інформацію про користувачів, рослини, сенсори та зчитування.

Spring Boot – фреймворк Java для створення серверних застосунків, використаний у розробці серверної частини системи та IoT симулятору.

Maven – система управління проєктами, що використовується для автоматизації білдів, тестування й залежностей у проєкті.

2 ЗАГАЛЬНИЙ ОПИС

2.1 Перспективи продукту

Основні перспективи продукту:

– розширення функціональності системи шляхом покращення можливостей сенсорів;

– покращення алгоритмів обробки зчитувань та відповідної зміни освітлення;

– додавання інших пристроїв, що будуть покращувати зростання рослин, як, до прикладу, пристрої поливу;

– створення мобільного застосунку для зручного моніторингу й керування системою в реальному часі;

– забезпечення підтримки хмарної синхронізації для масштабування на великі агропромислові об’єкти;

2.2 Функції продукту

Користувачі матимуть такі функції:

– реєстрація;

– аутентифікація;

– перегляд усіх користувачів;

– перегляд усіх ролей;

– надавання, або скасування ролі користувачу;

– експорт та імпорт даних користувачів;

– управління рослинами, CRUD операції;

– управління сенсорами, CRUD операції;

– управління освітлювальними пристроями, CRUD операції;

– перегляд зчитувань та попереджень;

2.3 Характеристики користувачів

2.3.1 Адміністратори

– відповідають за управління користувачами та ролями;

– добре орієнтуються в налаштуваннях системи та безпеці;

– володіють базовими або розширеними технічними знаннями;

– працюють із CSV-експортом/імпортом даних;

2.3.2 Технічний персонал

– працюють із рослинами, сенсорами та освітленням;

– мають базові комп’ютерні навички;

– залучені до внесення даних, моніторингу та обслуговування обладнання;

– очікують простого та зрозумілого інтерфейсу.

2.3.3 Кінцеві користувачі (агровиробники або фермери)

– використовують систему для моніторингу стану рослин;

– цікавляться простими візуалізаціями та попередженнями;

– можуть не мати технічних вмінь, але зацікавлені в підвищенні ефективності виробництва;

– потребують інтуїтивно зрозумілого функціоналу та мобільності;

2.4 Загальні обмеження

На даному етапі присутні такі обмеження:

– присутній веб-інтерфейс, мобільного застосунку наразі не передбачено;

– доступ до різних ресурсів доступний користувачам лише при наявності необхідної ролі;

– присутні українська та англійська мови;

2.5 Припущення й залежності

На даному етапі припущення та залежності такі:

– система потребує локальної мережі, щоб передавати дані з сенсорів;

– серверна частина має бути розгорнута на системі із встановленою Java 17 та PostgreSQL;

– додаткові підключені пристрої мають передавати дані у форматі, сумісному з розробленим REST API;

– автоматичне керування освітленням залежить від наявності актуальних даних від сенсорів;

3 КОНКРЕТНІ ВИМОГИ

3.1 Вимоги до зовнішніх інтерфейсів

3.1.1 Інтерфейс користувача

Користувач має такий інтерфейс:

– взаємодія з користувачем: Swagger або Postman;

– доступні ресурси: реєстрація, автентифікація, рослини, сенсори, освітлювальні пристрої, зчитування, користувачі;

– простий інтерфейс, небагато вхідних параметрів;

3.1.2 Апаратний інтерфейс

Вимоги до апаратного інтерфейсу:

– підключення до локальної мережі;

– 1 ГБ оперативної пам’яті;

– 500 МБ вільного місця для зберігання даних;

3.1.3 Програмний інтерфейс

Серверна частина:

– технології: Java 17, Spring Boot 3, PostgreSQL;

– API: RESTful API для обміну даними між сервером, клієнтами та сенсорами;

3.1.4 Комунікаційний протокол

Для безпечної передачі даних між сенсорами, клієнтською та серверною частинами використовуватиметься протокол HTTPS.

Для управління сесіями користувачів та забезпечення контролю доступу використовується JWT. Це дозволяє реалізувати безпечну авторизацію з можливістю масштабування.

3.1.5 Обмеження пам’яті

Для коректної роботи програмного забезпечення діють такі обмеження:

– 500 МБ вільного місця на диску для встановлення програми та збереження даних;

– використовується реляційна база даних для зручної взаємодії різних об’єктів між собою;

3.1.6 Операції

Основні операції програмної системи:

– створення нового облікового запису користувача;

– вхід до системи з перевіркою даних (авторизація);

– можливість перегляду списку всіх зареєстрованих користувачів;

– доступ до інформації про всі наявні ролі в системі;

– призначення або видалення ролей у користувачів;

– можливість збереження інформації про користувачів у файл (експорт) або завантаження з файлу в систему (імпорт);

– додавання, редагування, перегляд і видалення інформації про рослини;

– повне керування даними сенсорів (додавання, оновлення, перегляд, видалення);

– налаштування параметрів освітлювальних пристроїв: їх створення, оновлення, видалення та перегляд;

– перегляд зчитувань, отриманих від сенсорів, та повідомлень про потенційні проблеми з рослинами;

3.1.7 Функції продукту

Програмне забезпечення має такі функції:

– регулювання освітлення;

– реєстрація та авторизація користувачів;

– управління користувачами з панелі адміністратора;

– моніторинг зчитувань з сенсорів;

– управління об’єктами програмної системи (сенсори, рослини, освітлювальні пристрої);

3.1.8 Припущення й залежності

Припущення:

– користувачі (фермери) мають базові навички роботи з веб-застосунками;

– усі сенсори та освітлювальні пристрої правильно під’єднані та знаходяться в робочому стані;

– дані, що надходять від сенсорів, є коректними та не містять суттєвих похибок;

Залежності:

– залежність від безперебійної роботи серверної частини, що відповідає за зберігання, обробку даних та логіку управління;

– залежність від точності показників, отриманих із сенсорів (вологість, температура, освітлення);

– залежність від енергопостачання для функціонування сенсорів, освітлення та серверної частини;

– залежність від зовнішніх бібліотек та фреймворків (Spring Boot, PostgreSQL тощо) для роботи системи;

– залежність від механізмів безпеки (JWT, HTTPS) для захисту конфіденційної інформації користувачів;

3.2 Властивості програмного продукту

Програмний продукт для автоматизованого вирощування рослин на плантаціях та фермах забезпечує автоматизований моніторинг і керування умов росту рослин, орієнтуючись на показники з сенсорів.

Основні властивості продукту:

– користувачі можуть додавати та редагувати інформацію про рослини, включаючи відсоток росту та вимоги до освітлення;

– система зчитує показники з сенсорів у реальному часі (температура, вологість, освітленість тощо) та реагує на зміни;

– автоматичне керування освітленням на основі отриманих кліматичних даних та встановлених параметрів для конкретних рослин;

– можливість перегляду зчитування параметрів середовища та попереджень про відхилення від норми;

– керування сенсорами та освітлювальними пристроями (додавання, редагування, видалення);

– підтримка багаторівневої рольової системи з можливістю призначення та скасування ролей користувачам;

– експорт та імпорт даних користувачів;

– захищений доступ до системи через реєстрацію та автентифікацію з використанням імені користувача і пароля;

3.3 Атрибути програмного продукту

3.3.1 Надійність

Ціль – система має стабільно працювати і видавати результати у режисі реального часу, обробляючи дані сенсорів і виконувати дії без збоїв.

Метрика – час простою не повинен перевищувати 0.1% на місяць при стандартному навантаженні.

3.3.2 Доступність

Ціль – програмна система повинна бути доступною користувачам в будь-який час доби, включно з обробкою даних з сенсорів та можливістю віддаленого управління пристроями.

Метрики:

– доступність системи має становити не менше 99.9% на місяць;

– система має бути доступною з будь-якого пристрою, що підключений до локальної мережі;

3.3.3 Безпека

Ціль – захистити дані програмної системи, такі як дані користувачів, параметри рослин, налаштування пристроїв, історію зчитувань.

Метрики:

– паролі мають бути захешовані;

– автентифікація відбувається з використанням JWT;

– всі попередження мають зберігатися у системі;

3.3.4 Супроводжуваність

Ціль – система повинна бути легкою для оновлення, тестування і масштабування.

Метрики:

– код має бути достатньо чистим та легким в розумінні;

– стандартизоване оформлення коду;

3.3.5 Переносимість

Ціль – можливість розгортання серверної частини на різних середовищах (локальних або хмарних).

Метрики:

– серверна частина побудована з використанням кросплатформеного стеку технологій;

– мінімальна кількість залежностей від специфічного середовища;

3.3.6 Продуктивність

Ціль – обробка вхідних даних та прийняття рішень щодо освітлення повинні виконуватись із мінімальною затримкою.

Метрики:

– час реакції системи на зміну сенсорних даних – не більше 1 секунди;

– підтримка обробки не менше 1000 запитів на хвилину у серверній частині;

3.4 Вимоги бази даних

Тип бази даних – реляційна база даних PostgreSQL.

Призначення – зберігання структурованих даних системи, зокрема:

– облікові записи користувачів і їхні ролі;

– інформація про рослини (стадії росту, потреби);

– дані сенсорів (вологість, температура, освітлення тощо);

– параметри освітлювальних пристроїв;

– зчитування;

3.5 Інші вимоги

Інші вимоги включають:

– підтримка локалізації української та англійської мов;