Національний технічний університет України

«Київський політехнічний інститут

імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

[Кафедра інженерії програмного забезпечення в енергетиці](https://kpi.ua/web_ipze)

***КУРСОВА РОБОТА***

з дисципліни **«Компоненти програмної інженерії-4»**

тема:**«Тестова документація проєкту “Застосунок для моніторингу та керування автоматичними гідропоніками”»**

|  |  |
| --- | --- |
| Керівник: доцент Варава І.А. | Виконав Михайлович О. Д. |
| Допущено до захисту | Студент 3 курсу |
| «21» січня 2025р. | Групи ТВ-22 |
| Захищено з оцінкою |  |
| \_\_\_\_\_\_\_ |  |

Київ - 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. ІГОРЯ СІКОРСЬКОГО»

Кафедра інженерії програмного забезпечення в енергетиці

**КУРСОВА РОБОТА**

з дисципліни: «Компоненти програмної інженерії 4»

на тему: «Тестова документація проєкту “Застосунок для моніторингу та керування автоматичними гідропоніками”»

Студента групи ТВ-22\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

напряму підготовки \_\_\_\_\_бакалавр\_\_\_\_\_\_\_\_\_\_\_

спеціальності \_\_\_121 Інженерія програмного забезпечення\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Михайловича О.Д\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник \_ доцент Варава І.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Національна оцінка \_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_ Оцінка: ECTS \_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_доцент Варава І.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

Київ - 2025

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

Інститут: Навчально-науковий атомної та теплової енергетики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра: Інженерія програмного забезпечення в енергетиці

Напрям підготовки 121 Інженерія програмного забезпечення\_\_\_\_\_\_\_\_\_\_

**ЗАВДАННЯ**

НА КУРСОВУ РОБОТУ СТУДЕНТА

\_\_\_\_\_Михайловичу Олександру Дмитровичу\_\_\_\_\_\_\_\_

(прізвище, ім’я, по-батькові)

1.Тема роботи: «Тестова документація проєкту “Застосунок для моніторингу та керування автоматичними гідропоніками”»

Керівник проєкту(роботи): Варава Іван Андрійович доцент

(прізвище, ім’я, по-батькові, науковий ступінь, вчене звання)

2.Срок подання студентом роботи: 24 січня 2025 р.

3.Вихідні дані до проєкту(роботи): надати користувачу можливість роботи та аналізу IoT гідропоніками, чия фізична частина імітується сервером, й чиї зміни відображаються на графічному інтерфейсі.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

опис проекту із включенням усіх вимог, складання traceability matrix, тест-план з усіма пунктами, баг-репорт та практичні тести

5.Дата видачі завдання: «8» жовтня 2024 р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назви етапів виконання курсової роботи | Строк виконання етапів проєкту(роботи) | Примітка |
| 1 | Затвердження теми роботи | 07.10.2024 |  |
| 2 | Вивчення та аналіз здачі | 12.10.2024 |  |
| 3 | Розробка тест плану | 27.10.2024 |  |
| 4 | Розробка баг репорту | 15.11.2024 |  |
| 5 | Опис задачі з усіма вимогами, traceability matrix | 24.11.2024 |  |
| 6 | Тестування програми | 20.12.2024 |  |
| 7 | Оформлення пояснювальної записки | 28.12.2024 |  |

**Студента**  \_\_\_\_\_ Михайлович О.Д.\_\_\_\_\_

**Керівник курсової роботи** \_\_\_\_\_\_\_\_\_ доцент Варава Іван Андрійович.

**АНОТАЦІЯ**

Мета курсової роботи розробити та протестувати додаток для роботи з ІоТ гідропоніками з емуляцією фізичних процесів на сервері із збереженням даних на SQL базі даних. Для розробки використовувалися: пакетні менеджери poetry та npm, фреймворки FastAPI, Vite, SQLModels, tSQLt, WSGI/ASGI/RSGI сервер Granian, Microsoft SQL сервер, VS Code, SQL Server Management Studio 20, Bruno, Swagger, мова програмування Python 3.13 та Vanilla Web Stack + WebComponents [8]. Було проведено тестування програмного забезпечення, для тестування було використано: unit-тести, інтеграційні тести, мануальне тестування, запис макросів, тестування бази даних. Результатом на додачу до основного ПЗ є розроблена супровідна документація з акцентом на тестування застосунку.

Обсяг пояснювальної записки - 79 аркушів, кількість ілюстрацій - 40, кількість додатків - 4.

**ANNOTATION**

The purpose of the course work is to develop and test an application for working with IoT hydroponics with emulation of physical processes on the server with data storage in a SQL database. For development was used: poetry and npm package managers, frameworks FastAPI, WebComponents, Vite, SQLModels, tSQLt, server WSGI/ASGI/RSGI Granian, Microsoft SQL server, VS Code, SQL Server Management Studio 20, Bruno, Swagger, Python 3.13 programming language and Vanilla Web Stack. The software was tested, and the following were used for it: unit tests, integration tests, manual testing, macro recording, database testing. The result, in addition to the main software, is the developed supporting documentation with a focus on application testing.

Volume of explanatory note sheets - 79, number of illustrations - 40, volume of appendices - 4.

**ЗМІСТ**

[ВСТУП 9](#_Toc188653519)

[РОЗДІЛ 1. ОПИС ПРОЄКТУ 12](#_Toc188653520)

[1.1 Функціонал додатку 12](#_Toc188653521)

[1.2 Інтерфейс додатку 14](#_Toc188653522)

[1.3 Використані програмні засоби 28](#_Toc188653523)

[1.4 Вимоги до проєкту 30](#_Toc188653524)

[РОЗДІЛ 2. ТЕСТ-ПЛАН ПРОЄКТУ 34](#_Toc188653525)

[2.1 Ідентифікатор плану тестування 35](#_Toc188653526)

[2.2 Цілі тестування 35](#_Toc188653527)

[2.3 Сфера застосування 36](#_Toc188653528)

[2.4 Елементи тестування 36](#_Toc188653529)

[2.5 Функціонал, що тестується 37](#_Toc188653530)

[2.5.1 Тестування хешування на сервері 37](#_Toc188653531)

[2.5.2 Тестування коректності JSON Web Token + обгортки драйверу на сервері 37](#_Toc188653532)

[2.5.3 Тестування коректності обгортки драйверу + хешування на сервері 37](#_Toc188653533)

[2.5.4 Тестування класу обгортки драйверу на додавання й видалення користувача на сервері 38](#_Toc188653534)

[2.5.5 Тестування класу обгортки драйверу на додавання, видалення, отримання гідропоніки на сервері 38](#_Toc188653535)

[2.5.6 Тестування класу обгортки драйвера на додавання води до гідропоніки на сервері 38](#_Toc188653536)

[2.5.7 Тестування класу обгортки драйвера на додавання мінералів до гідропоніки на сервері 38](#_Toc188653537)

[2.5.8 Тестування класу обгортки драйвера на додавання кисню до гідропоніки на сервері 39](#_Toc188653538)

[2.5.9 Тестування класу обгортки драйвера на збільшення температури в гідропоніці на сервері 39](#_Toc188653539)

[2.5.10 Тестування класу обгортки драйвера на зменшення температури в гідропоніці на сервері 40](#_Toc188653540)

[2.5.11 Тестування класу обгортки драйвера на збільшення кислотності в гідропоніці на сервері 40](#_Toc188653541)

[2.5.12 Тестування класу обгортки драйвера на зменшення кислотності в гідропоніці на сервері 41](#_Toc188653542)

[2.5.13 Тестування класу обгортки драйвера на перевантаження гідропоніки на сервері 42](#_Toc188653543)

[2.5.14 Набір тестів для тестування API, й запитів до бекенду із заміною/підміною елементів (використання тестових двійників) 42](#_Toc188653544)

[2.5.15 Тестування веб-клієнта, та загальної системи - залогінення 43](#_Toc188653545)

[2.5.16 Тестування веб-клієнта, та загальної системи - реєстрація 43](#_Toc188653546)

[2.5.17 Тестування веб-клієнта, та загальної системи - переадресація за відсутності токена 43](#_Toc188653547)

[2.5.18 Тестування веб-клієнта, та загальної системи - вихід з акаунту 44](#_Toc188653548)

[2.5.19 Тестування веб-клієнта, та загальної системи - додавання та видалення 44](#_Toc188653549)

[2.5.20 Тестування веб-клієнта, та загальної системи - керування гідропонікою, кнопка перезавантаження гідропоніки 44](#_Toc188653550)

[2.5.21 Тестування веб-клієнта, та загальної системи - керування гідропонікою, додавання води 44](#_Toc188653551)

[2.5.22 Тестування веб-клієнта, та загальної системи - керування гідропонікою, додавання мінералів 45](#_Toc188653552)

[2.5.23 Тестування веб-клієнта, та загальної системи - керування гідропонікою, додавання кисню 45](#_Toc188653553)

[2.5.24 Тестування веб-клієнта, та загальної системи - керування гідропонікою, керування температурою 45](#_Toc188653554)

[2.5.25 Тестування веб-клієнта, та загальної системи - керування гідропонікою, керування кислотністю 46](#_Toc188653555)

[2.6 Функціонал, що не тестується 46](#_Toc188653556)

[2.7 Підхід 47](#_Toc188653557)

[2.7.1 Інструменти тестування 47](#_Toc188653558)

[2.7.2 Збір показників та метрики 48](#_Toc188653559)

[2.7.3 Конфігурації що будуть тестуватися 49](#_Toc188653560)

[2.7.4 Підхід до тестування додатку 49](#_Toc188653561)

[2.8 Критерії проходження/провалу для плану тестування 50](#_Toc188653562)

[2.9 Критерії призупинення та умови відновлення тестування 51](#_Toc188653563)

[2.9.1 Критерії призупинення тестування 51](#_Toc188653564)

[2.9.2 Критерії відновлення тестування 51](#_Toc188653565)

[2.10 Результати тестування 51](#_Toc188653566)

[2.11 Тестові завдання 52](#_Toc188653567)

[2.11.1 Залежності між тестами 52](#_Toc188653568)

[2.11.2 Рівень навичок 52](#_Toc188653569)

[2.11.3 Ключові етапи 52](#_Toc188653570)

[2.12 Потреби середовища 53](#_Toc188653571)

[2.12.1 Джерела інформації 53](#_Toc188653572)

[2.12.2 Безпека 53](#_Toc188653573)

[2.12.3 Відповідальності 53](#_Toc188653574)

[2.13 Ризики 53](#_Toc188653575)

[2.13.1 Зміна строків виконання роботи 53](#_Toc188653576)

[2.13.2 Неповний проєкт 53](#_Toc188653577)

[2.14 Затвердження 54](#_Toc188653578)

[РОЗДІЛ 3 РОЗРОБКА БАГ РЕПОРТУ 55](#_Toc188653579)

[3.1 Список знайдених багів 56](#_Toc188653580)

[3.2 Складений баг репорт - баг №1 56](#_Toc188653581)

[3.2.1 Тема багу 56](#_Toc188653582)

[3.2.2 Детальний опис багу 56](#_Toc188653583)

[3.2.3 Кроки до відтворення 57](#_Toc188653584)

[3.2.4 Результати 57](#_Toc188653585)

[3.2.5 Інформація про баг 57](#_Toc188653586)

[3.3 Розбір рішення багу 58](#_Toc188653587)

[3.3.1 Розбір причини проблеми 58](#_Toc188653588)

[3.2.2 Огляд можливих рішень 60](#_Toc188653589)

[3.2.3 Результат виправлення 61](#_Toc188653590)

[Розділ 4 ТЕСТУВАННЯ ПРОЄКТУ 62](#_Toc188653591)

[4.1 Автоматизоване тестування бекенд серверу (unittest, Coverage.py, unittest.mock) 62](#_Toc188653592)

[4.1.1 Переваги та недоліки 62](#_Toc188653593)

[4.1.2 Середовище написання 63](#_Toc188653594)

[4.1.3 Написання тестів 64](#_Toc188653595)

[4.1.4 Результат 67](#_Toc188653596)

[4.2 Автоматизоване тестування фронтенд клієнту із використанням Silenium WebDriver 69](#_Toc188653597)

[4.2.1 Переваги та недоліки 69](#_Toc188653598)

[4.2.2 Середовище написання 70](#_Toc188653599)

[4.2.3 Написання тестів 70](#_Toc188653600)

[4.2.4 Результат 73](#_Toc188653601)

[4.3 Інші засоби тестування 74](#_Toc188653602)

[ВИСНОВОК 78](#_Toc188653603)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 80](#_Toc188653604)

[ДОДАТОК A 81](#_Toc188653605)

[ДОДАТОК B 101](#_Toc188653606)

[ДОДАТОК C 102](#_Toc188653607)

[ДОДАТОК D 120](#_Toc188653608)

# ВСТУП

Гідропоніка - це мистецтво вирощування рослин за допомогою занурення кореневої системи у воду. Це одна з систем вирощування рослин без використання ґрунту, до подібних систем також відносяться аеропоніки - себто такі системи де рослини отримують живлення від розпилення аерозолю. Отже, рослини можуть зростати у воді, але не за будь-яких умов. Існує помилкова думка про те, що якщо рослини мають достатнє водопостачання - тобто поливу, вони можуть бути залишені без нагляду. Фактично ж їх прискорений метаболізм вимагає більшої уваги.

Так для ефективного вирощування різних рослинних культур методом безпідставної культивації - гідропоніці потрібно зробити певні умови, які стимулюють зростання рослин за допомогою регулювання температури, pH, кількості води, мінеральних солей і, що найбільш важливо, розчиненого кисню.

Слід зазначити, що гідропоніка, як метод вирощування, застосовується на протязі більше ніж 50 років в багатьох дослідницьких центрах через його надійність, точність і широкий спектр його застосування. Завдяки гідропоніці були отримані дані, що дозволяють розуміти рослини та способи догляду за ними.

На даний час на рівні з промисловою гідропонікою стає популярним гідропонні системи малих розмірів для садівників-аматорів, які побудовані за тими ж принципами, що й промислової контрагенти. Ця системи може бути встановлена на балконі або на внутрішньому дворику, вітальні або кухні, офісі або зимовому саду.

Варто відзначити важливу характеристику, яка виявилась критичною для такого типу вирощування рослин - рівень кисню у воді. Кисень використовується як антисептик в цій системі. Без нього корені рослин подібно до деревини у воді (з певними виключеннями) були б піддані процесу гниття.

Враховуючи важливість контролю в гідропоніці за середовищем в якому знаходиться рослина, стає необхідним розробити додаток, через який виникає можливість контролювати стан кожного окремого насадження.

Таким чином розробка web-додатку з клієнт-сервісною архітектурою для контролю росту та забезпечення життєдіяльності як однієї рослини, так і справжнього міні городу, є доцільним з огляду на можливість регулювання гідропонік на відстані.

В цьому звіті наводиться текстова документація проєкту: застосунок для моніторингу та керування автоматичними гідропоніками, відносно якого було проведено тестування. Тестування для даного проєкту є важливим етапом, який в подальшому забезпечить стабільну, сталу і визначену роботу системи та стане підтвердженням коректності роботи всіх компонент.

Для повного тестування систем було використано низку способів та технологій тестування, починаючи з тестування бази даних, багато-векторного тестування серверної частини застосунку, автоматизованого тестування графічного інтерфейсу, що був виконаний на Vanilla web stack технологій із додатковим застосування WebComponents.

Базовою частиною тестування є тестування коректності та цілісності даних на базі даних у наслідку виконання CRUD операцій та додаткових обмежений накладеними тригерам(и). В наслідок цього було перевірено правильність та повноцінність накладених на дані обмежень та автоматизованих перевірок (тригер), відсутність конфліктів та ушкоджень даних у наслідок виконання CRUD операцій.

Було використано тестування з використанням тестових двійників (test doubles): Mock, Stub, Fake. Дане тестування з тестовими двійниками (test doubles) використано для виділення та тестування визначеного “ядра” серверної частини системи, в якому було потрібно відкинути крайові частини системи та імітувати ті, що є базовими, проте не складають частину “ядра”, що було виділено для цього тестування.

Завдяки використанню цих та інших методик вдалось збільшити test coverage unit та інтеграційних тестів, що були використані в серверній частини. Загальний test coverage серверної частини застосунку складає 82% (100% частини, що відповідає за безпеку, 100% за основні API endpoints, окрім окремого запиту на отримання часу запуску сервера, 78% за з’єднання з базою даних тощо).

Також було проведено тестування web-клієнту, та API endpoint-ів.

# РОЗДІЛ 1. ОПИС ПРОЄКТУ

Як результат курсової роботи було розроблено клієнт-серверний веб застосунок для контролю та аналізу автономних (IoT) гідропонік. Додаток характеризується використанням класичної архітектури/схеми база даних-бекенд-фронтенд. Завдяки використанню веб технологій в розробці клієнтської частини, наявна можливість легшого портування застосунку на інші платформи, наразі основними платформами є персональні комп’ютери та лептопи. Варто відзначити, що система емулює (значно спрощені) фізичні процеси гідропонік на бекенд сервері та зберігає дані на базі даних.

## 1.1 Функціонал додатку

Додаток виконує класичні операції додавання/видалення/зміни/відображення для гідропонік, а також ті ж самі операції (окрім операції видалення акаунту для веб клієнту, яка є ексклюзивною для серверною API) для аккаунта юзера. Виходячи з цього опису (та дизайну веб клієнту) функціонал додатку складається з відображення характеристик усіх гідропонік у вигляді списку, відображення окремої гідропоніки та можливість керування приладом на цій же сторінці. Керування приладом виконується наступним чином - наявна можливість зміни температури й кислотності розчину, а також імітація фізичної взаємодії з приладом у вигляді додавання кисню, мінералів та води у відповідні баки. Також на сторінці керування приладом наявно дві основні кнопки глобального управління автономним приладом, кнопка перевантаження - що дозволяє скинути значення приладу, а також кнопка видалення - яке виконує однойменну дію, яка полягає в формуванні та передачі запиту на видалення гідропоніки.

Для відображення фізичних станів приладу (наприклад температура, кислотність рівень кисню в баці, кількість мінералів у відповідному відділенні IoT приладу тощо), також додано відображення стану індикатору за допомогою уніфікованих (між однойменними показниками) кольорових позначень - себто кольорами, на додачу до текстових показників та геометричних показників рівня наповнення/стану певної характеристики.

Use case діаграма (рисунок 1.1), окрім зв’язків, також відображає важливу складову - компоненту авторизації. Важливо відзначити, що відповідно до практик використання паролів, їх було збережено у захешованому форматі (алгоритм хешування argon2) також було використано так звану сіль, додатковий випадковий рядок, що дозволяє підвищити захист системи від використання нападу із використанням словника. Для збереження автентифікації було використано Json Web Token та хешування HS256.

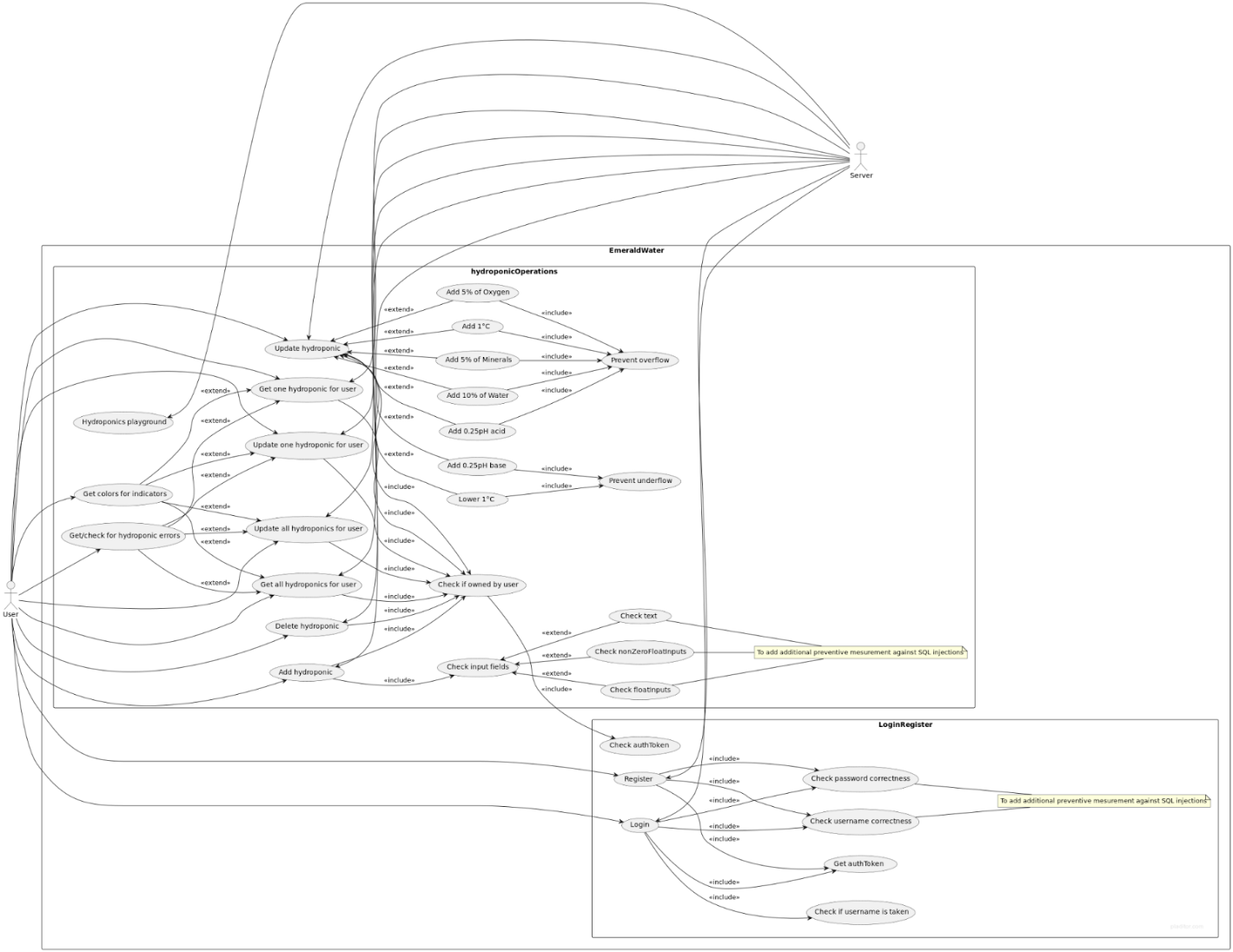


Рисунок 1.1 - use case діаграма написана на мові PlantUML

Логічно відзначити можливість додавання приладів, а також акаунту (реєстрації). За розробки цих компонентів на серверній частині було додано додаткові перевірки (regex), також перевірки було додано на клієнтській частині - таким чином ввід даних є уніфікованим та відображає помилки вводу (або блокує введення некоректних символів).

На додачу до цього, серверна API має згенеровану документацію у форматі (OpenAPI - їх дві) тому за бажанням користувачі можуть, як користуватися системою завдяки HTTP запитам (наприклад за використання застосунку cURL). Однак є обмеження на написання власної клієнтської частини накладеними CORS політиками.

## 1.2 Інтерфейс додатку

На рисунку 1.2 показано основну сторінку, де в компоненті логін (Login) забезпечено можливість входу зареєстрованим користувачам завдяки введення свого логіну та паролю. Для нових користувачів додано позначку “квадратик, який розпадається”, для проходження реєстрації через його натискання.

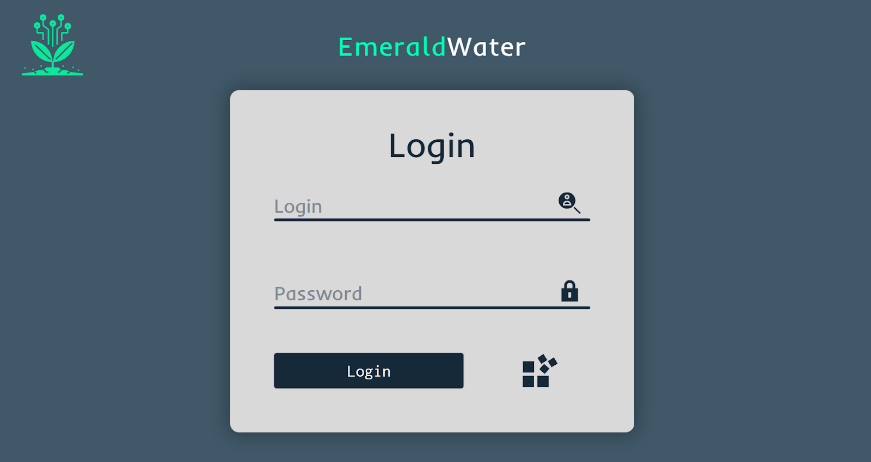


Рисунок 1.2 - основна сторінка Login (стан: порожній)

На рисунку 1.3 в блоці реєстрація (Sigin Up) показано, відсутність у нового користувача права при реєстрації використати ім’я, яке вже закріплене за попердньо зареєстрованим користувачем. При введенні ім’я, яке належить вже зареєстрованому користувачу гнучка частина компоненту input (риска знизу) забарвлюється у інший колір та з’являється попереджувальний напис “User is already taken”. В гнучкій частина компонентів input (риска знизу) також забезпечено наявність прихованого тексту, який за бажанням розробника може містити інформацію про помилку або довідкову інформацію.

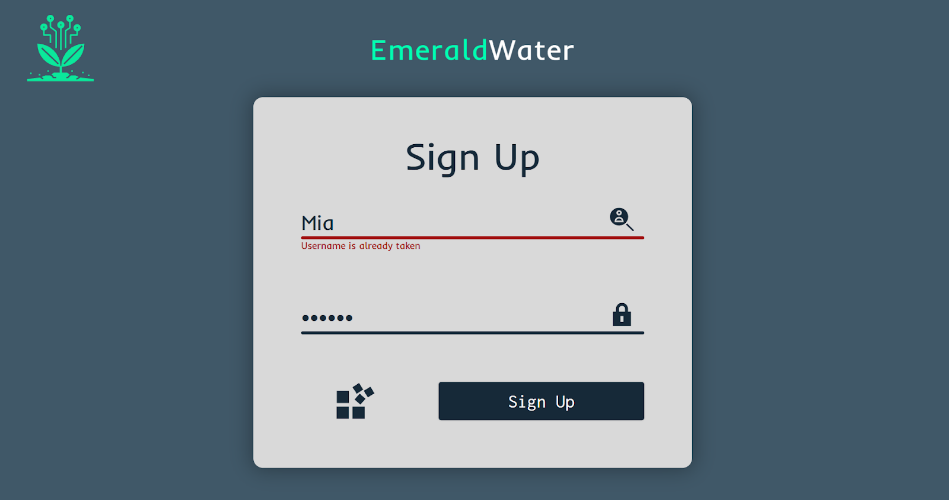


Рисунок 1.3. - основна сторінка Sign Up (стан: помилка при введенні ім’я для реєстрацій, яке вже зайнято зареєстрованим користувачем)

На рисунку 1.4 показано можливість в компоненті логін (Login), для зареєстрованого користувача, переглядати свій пароль шляхом натискання кнопки “замок”, де останній в первинному вигляді має образ закритого замку, а при натисканні має вигляд відкритого замку. Також в компоненті логін (Login) через натисканні кнопки login здійснюється перехід на головну сторінку, з усією внесеною попередньо, гідропонікою зареєстрованого користувача.

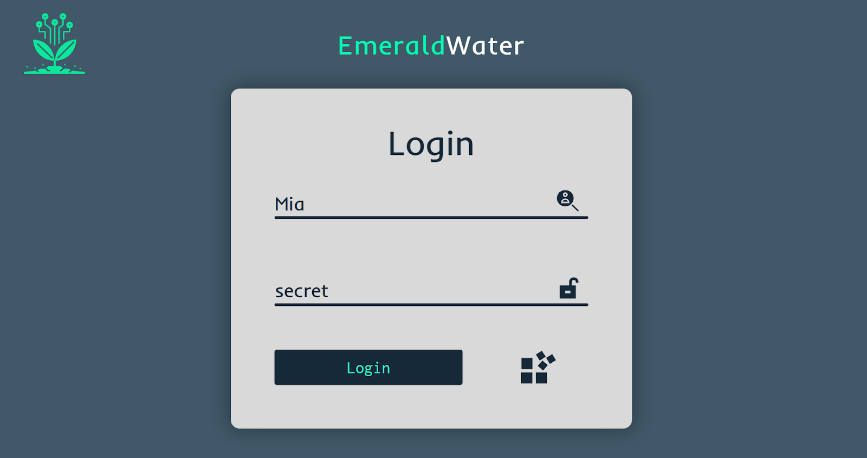


Рисунок 1.4 - основна сторінка Login (стан: перегляд попередньо зареєстрованого паролю та можливістю натискання кнопки login для переходу на головну сторінку)

Рисунок 1.5 відображає основну сторінку, яка містить в правому верхньому кутку кнопку переходу до сторінки розлогінення. При натисканні кнопки, остання змінює колір з блідо сірого на синій. Також на головній (основній) сторінці користувача розташовані контейнери гідропоніки різного призначення, які попередньо додані зареєстрованим користувачем. При натисканні на будь-який веб-компонент гідропоніки здійснюється перехід на її сторінку керування. Для більшої динамічності сторінки зліва додана плашка з тематичним значком та назвою кожної конкретної гідропоніки. Дана плашка при підведенні курсору до будь-якого компоненту гідропоніки рухається у бік розширення. Це ефект додана для нагадування користувачу у якій гідропоніці він саме зараз знаходиться.

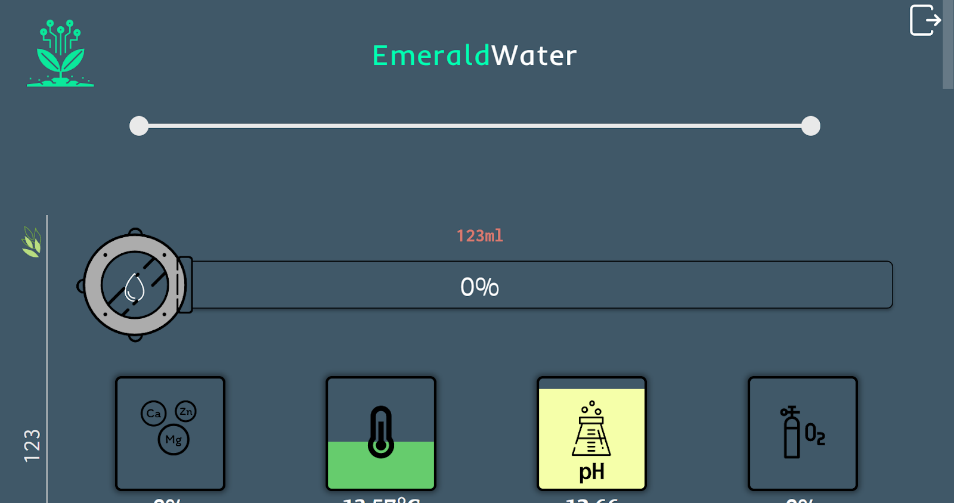


Рисунок 1.5 - основна сторінка, кнопка справа зверху надсилає до сторінки розлогінення, натискання на гідропоніку надсилає на сторінку керування нею

На рисунку 1.6 головна (основна) сторінка також містить інформаційний блок-статус, який відображає критичний стан контейнерів гідропоніки. Даний блок-статус має білу заливку та всередині містить інформацію, яка підпадає під категорію “Dangerous”. Блок-статус, який прив'язаний до системи гідропоніки, в момент заходу користувача на головну сторінку, має функціонал сповіщати про найбільш критичний стан систем гідропоніки. Окрім сигнального попередження “Dangerous” блок-статус має сповіщення про конкретну проблему критичного контейнеру гідропоніки.

Сповіщення змінюється відповідно до суті критичного контейнеру гідропоніки.

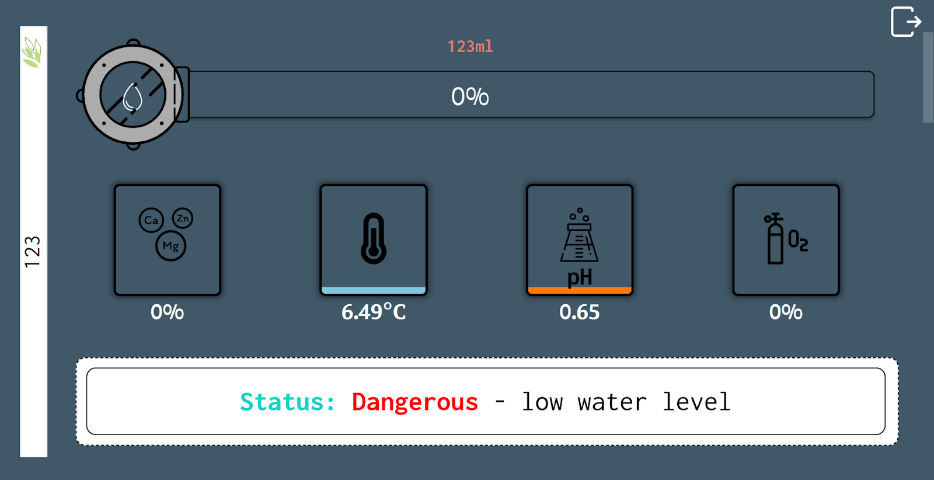


Рисунок 1.6 - критична гідропоніка - наведений компонент (біла заливка назви) - небезпека в полі повідомлень

На рисунку 1.7 показаний стан гідропоніки у стані “OK” шляхом надання про це інформації у блок-статус “everything is fine”

Контейнери гідропоніки є інтерактивними і мають зв’язок з віртуальною емуляцією фізичної системи гідропоніки.

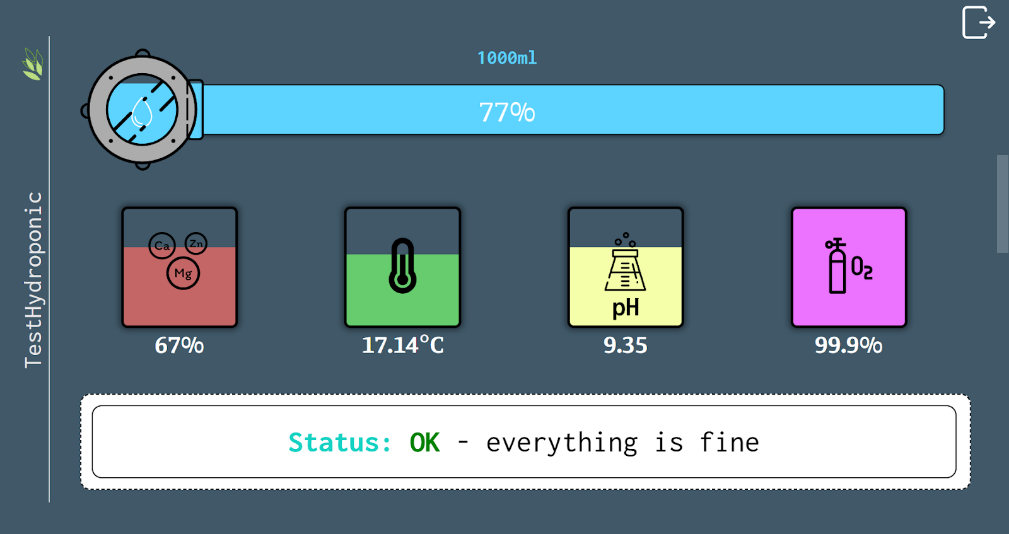


Рисунок 1.7 - гідропоніка в гарному стані

Зв’язок контейнеру гідропоніки з емуляцією віртуальної системи гідропоніки реалізується через зміну у контейнері гідропоніки значення/показань речовин необхідних для підтримання відповідного стану рослин.

Зміна значень/показань здійснюється шляхом зміни забарвлення,   
де наприклад для води червоний це є критична кількість, а світло-блакитний є її достатньою кількістю.

На рисунку 1.8 показано головну (основну) сторінку, на якій відображається результат керування/заповнення користувачем компонентів (контейнерів) гідропоніки, інформацію відносно яких було попередньо внесено на сторінку додавання під час додавання опису рослини та заповнення для неї її гідропоніки. Загальна кількість компонентів (контейнерів) гідропоніки для кожної рослини складається з п’яти необхідних елементів: вода, суміш мінералів, температура, pH, кисень. Також головна (основна) сторінка з компонентами (контейнерами) гідропоніки показує користувачу всі системи додавати ним для кожної конкретної рослини. Керування компонентами (контейнерами) гідропоніки здійснюється через клікабельні кнопки. Так клікабельна кнопка додавання води “Add water” заповнює контейнер з водою через кожне натискання на 10 % води. Клікабельна кнопка додавання суміші мінералів “Add minerals” додає до контейнеру з мінералами через кожне натискання 5 % суміші. Кожен із контейнерів має динамічну зміну показників, тобто за зміни шляхом дії фізичної емуляції чи вводу користувача показники змінюються.

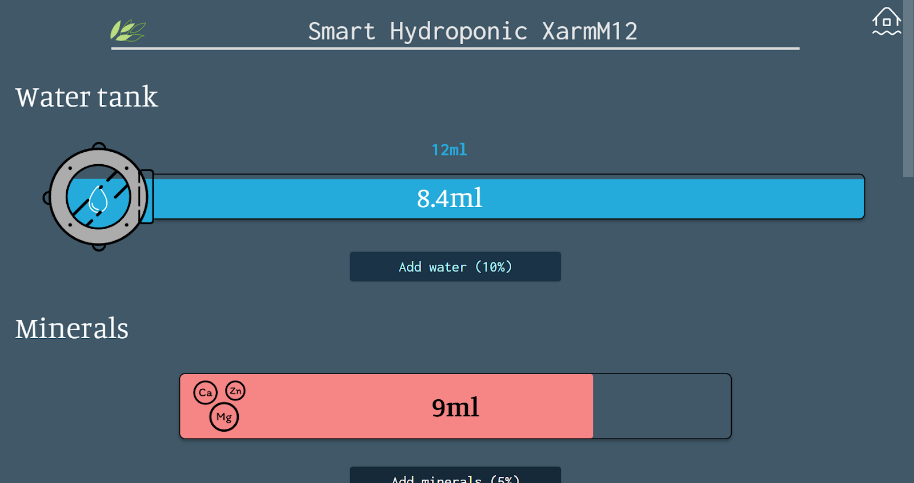


Рисунок 1.8 - керування гідропонікою перший екран, наведення на кнопку додавання води

Запит на отримання нової інформації виконується кожні 5 секунд, тобто зміна виконана іншим клієнтом (наприклад на іншому пристрої, або на іншій сторінці тощо) або зміни внаслідок емуляції відображається із зазначеною затримкою, проте зміни внесені користувачем в одному клієнті відображаються одразу (наприклад видалення гідропоніки, зміна показників на сторінці керування тощо). Кнопки під час натискання/наведення змінюють колір з блідо сірого на блакитний.

Рисунок 1.9 показує продовження сторінки керування компонентами (контейнерами) гідропоніки. Так компонент (контейнер) гідропоніки - температура має дві клікабельні кнопки, завдяки яким виникає змога збільшувати “Add” або зменшувати “Lower” показник на 1℃. Компонент (контейнер) гідропоніки - кислотність має дві клікабельні кнопки, завдяки яким виникає змога додавати луг “Add base” або навпаки підвищувати кислотність “Add acid”. Зміна pH відбувається з кроком 0,25. Контейнери (компоненти) гідропоніки - температура та кислота є динамічними.

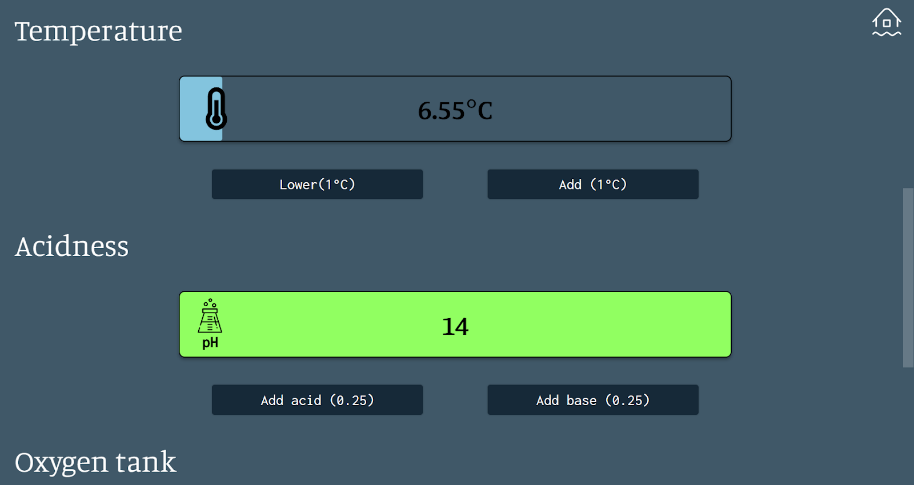


Рисунок 1.9 - керування гідропонікою перший екран, зміна температури у бік зниження зі зміною показника та збільшення кислотності із зміною забарвлення та зміною показника

Так, наприклад, при зменшенні температури показник рухається в лівий бік та змінює колір на блідо блакитний, при цьому в самому контейнері гідропоніки - температура також змінюється показник, який вказує яка саме температура води наразі у конкретної рослини.

При зміні показнику кислотності їде зміна кольору. Так у разі, якщо кислотність збільшується колір стає яскраво зеленим, при цьому також змінюється показник pH.

Рисунок 1.10 показує продовження сторінки керування компонентами (контейнерами) гідропоніки. Так компонент (контейнер) гідропоніки - кисень має одну клікабельну кнопку додати кисень “Add oxygen”. Кожне натискання клікабельної кнопки додає кисню на 5 %. Компонент (контейнер) гідропоніки - кисень є динамічним тобто змінює показники зі зміною кольору та рухливою лінією у разі збільшення в правий бік, а зменшення у лівий бік, при цьому їде зміна цифри вмісту кисню залежно від того, в який бік рухається лінія. Клікабельна кнопка при наведені/натисканні змінює колір з блідо сірого на блакитний.

Також на рисунку 1.10 показано кнопку зверху справа у вигляді будинку, яка є фіксованою і в статичному стані має блідо сірий колір, а під час наведення/натискання змінює забарвлення на яскраво блакитне. Дана кнопка при натисканні переносить користувача на головну (основну) сторінку з усіма гідропоніками.



Рисунок 1.10 - керування гідропонікою перший екран, зміна кисню у бік збільшення, наведення на кнопку, яка розташована зверху справа, для переходу на сторінку усієї гідропоніки

У нижній частині, яка показана на рисунку 1.10, розташовані дві клікабельні кнопки - скидання “Reload”, вона використовується для емуляції дії фізичного втручання в систему гідропоніки із, наприклад, переливанням води - внаслідок таких дій користувач зміною значення усіх показників й отримає випадкові значення від сервера та видалення “Delete” при натисканні на яку виконується видалення компонентів (контейнерів) зі сторінки та перекидає на головну (основну сторінку) гідропоніки.

Дані клікабельні кнопки скидання “Reload” та видалення “Delete” при наведенні/натисканні змінюють колір.

На рисунку 1.11 показано порожню сторінку, на якій здійснюється додавання нового елементу (нової гідропоніки у складі 5 елементів) шляхом натискання на кнопку у вигляді кола, в яке схематично вписано малюнок рослини.

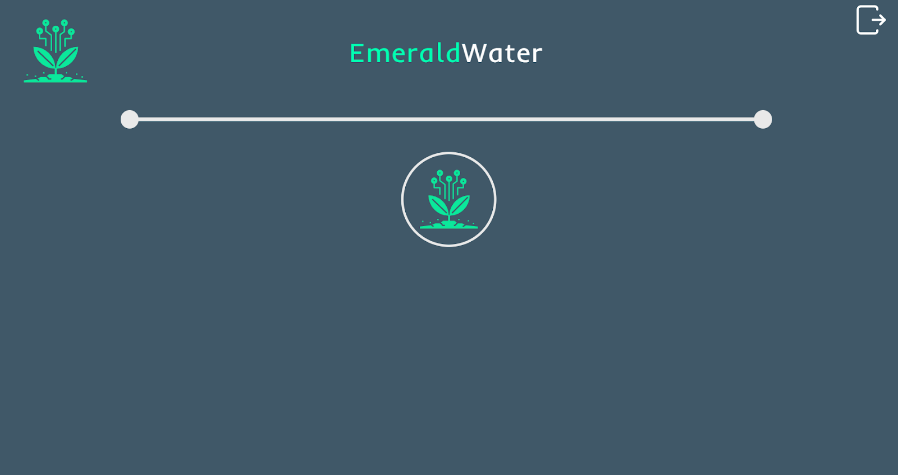


Рисунок 1.11 - порожня основна сторінка з кнопкою додавання нового елементу, яка має вигляд кола, в яке схематично вписано малюнок рослини

Також дана сторінка містить в правому верхньому кутку кнопку переходу до сторінки розлогінення, яка під час наведення/натискання змінює колір з блідо сірого на блакитний.

На рисунку 1.12 показано кнопку додавання нового елементу (нової гідропоніки у складі 5 елементів), яка має вигляд кола, в яке схематично вписано малюнок рослини.



Рисунок 1.12 - зміна вигляду кнопки додавання гідропонік на наведення

Кнопка за натискання змінює колір з біло-зеленого на біло-рожевий.

На рисунку 1.13 показано сторінку вводу інформації для формування компонентів (контейнерів) гідропоніки для кожної окремої рослини. Кількість полів для заповнення становить 10. В верхній частині сторінки додавання першим є поле для заповнення ім’я/назви рослини.

Під час заповнення даного поля/input (риска знизу) змінює колір з блідо сірого на яскраво блакитний.

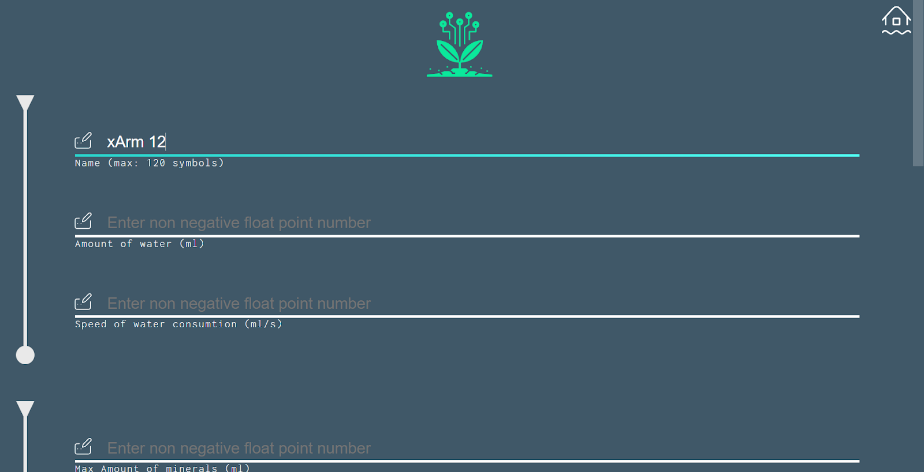


Рисунок 1.13 - верхня частина сторінки додавання - введення поля ім’я/назви рослини

На рисунку 1.13 показано сторінку вводу інформації для формування компонентів (контейнерів) гідропоніки для кожної окремої рослини, де наступним полем є внесення показників кількості води в мілілітрах - “Amount of water”. Враховуючи що одним із основних компонентів гідропоніки є вода, було заблоковано внесення показника, що є меншим або рівно 0. Подібні елементи вимагають введення числа із плаваючою комою, що є строго більшим нуля (окрім полів введення температури, кислотності та назви гідропоніки). Перевірки дублюються на бекенд сервері.

Також з'являється вікно з попереджувальний написом “Значення повинно бути більше 0” - “Value must be greater than 0”, яке випадає під час невірно введеного значення.

Також під час помилкового заповнення даного поля/input (риска знизу) змінює колір з блідо сірого на яскраво червоний.

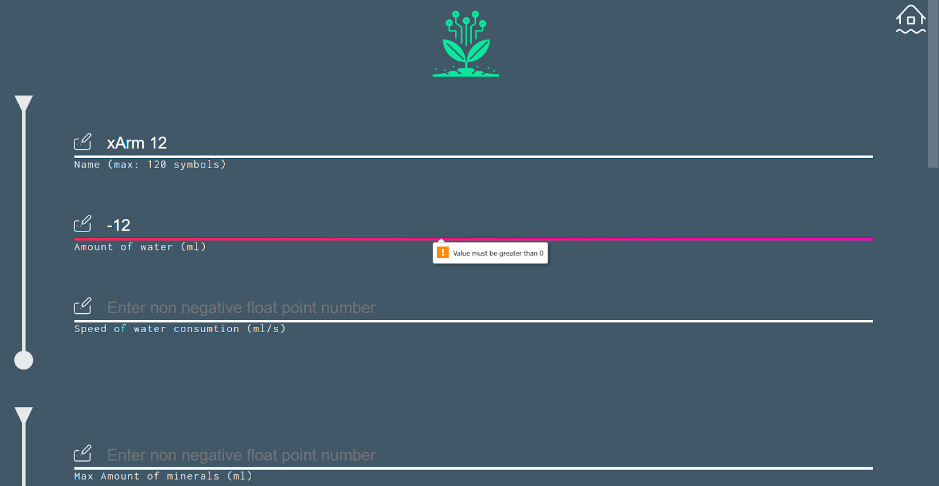


Рисунок 1.13 - помилка вводу- негативне значення

На рисунку 1.14 зображено ще один тип полів вводу - поле для введення інформації про кислотність (обмеження в цьому випадку полягає у відрізку де значення вважається коректним [0, 14]), та поле вводу інформації про температуру (проміжок [5, 25]).

Обидва поля очікують/приймають число із плаваючою комою.



Рисунок 1.14 - Два особливих поля вводу

На рисунку 1.15 сторінка вводу інформації для формування компонентів (контейнерів) гідропоніки для кожної окремої рослини, в полі внесення показників необхідної температури “Optimal temperature” заблоковано можливість заблоковано внесення показника температури, що є меншим або рівно 0.

Також завдяки браузеру надається оповішувальне вікно з попереджувальним написом - “Значення повинно бути більше 5” - “Value must be greater than 5”, яке випадає під час невірного введення значення. Під час помилкового заповнення даного поля input (риска знизу) змінює колір з блідо сірого на яскраво червоний.

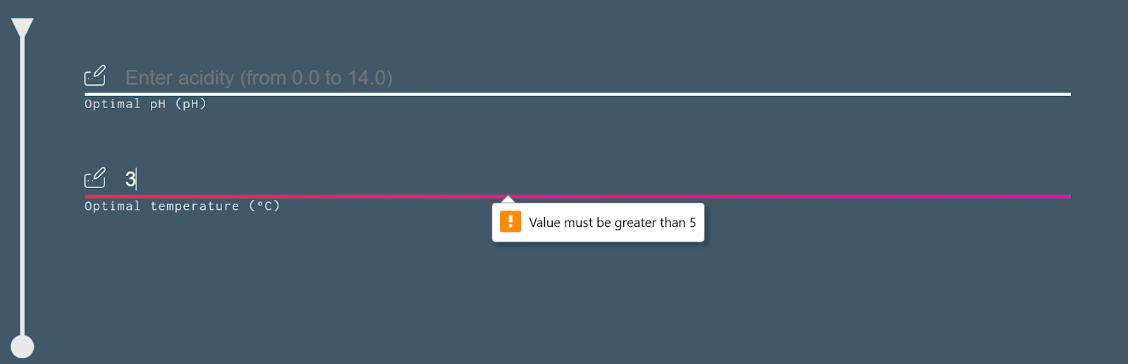


Рисунок 1.15 - введення значення за межею допустимого значення

На рисунку 1.16 сторінка вводу інформації для формування компонентів (контейнерів) гідропоніки для кожної окремої рослини, в полі внесення показників завдяки брайзеру з’являється вікно з попереджувальним написом “Значення не може бути пустим” - “Value cannot be empty” у разі, якщо користувач пропустив заповнення поля, наприклад, поле “Максимальне значення (сума) кисню” - “Max Amount of oxygen”. Також дане поле/input (риска знизу) змінює колір з блідо сірого на яскраво блакитний.

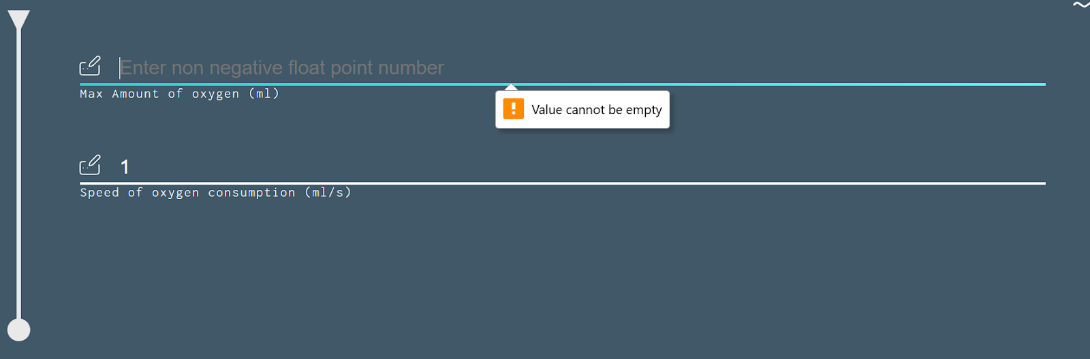


Рисунок 1.16 - випадіння вікна з попереджувальним написом та зміна кольору input у разі, пропуску користувачем заповнення поля вводу інформації для відповідних систем гідропоніки

На рисунку 1.17 показано головну (основну) сторінку, де у інформаційному блоці-статус попереджувальне слово забарвлено у жовтий колір. Даний колір є рівнем не критичного стану гідропоніки. Загалом прописано чотири рівня небезпеки у блоці - статус: зелений “OK” - все добре; жовтий “Warning”- увага; помаранчевий “Problem” - проблема; червоний “Dangerous”- небезпека. Під час програмування вище вказаних рівнів критичності кожен із компонентів (контейнерів) гідропоніки був виділений в окремий рівень критичності, в тому числі залежно від кількості показників у бік зменшення/збільшення. Для показників води, кисню зі значенням кількості речовини в баках  < 25  та показнику температури зі значенням  > 10 в блок-статусі попереджувальне слово завжди буде червоним  “Dangerous”- небезпека. Для показнику мінерали, зі значенням кількості речовини у баці < 25, в блок-статусі попереджувальне слово завжди буде жовтим “Warning”- увага.

Для показнику pH зі значенням > 5 в блок-статусі попереджувальне слово завжди буде помаранчевим “Problem” - проблема.

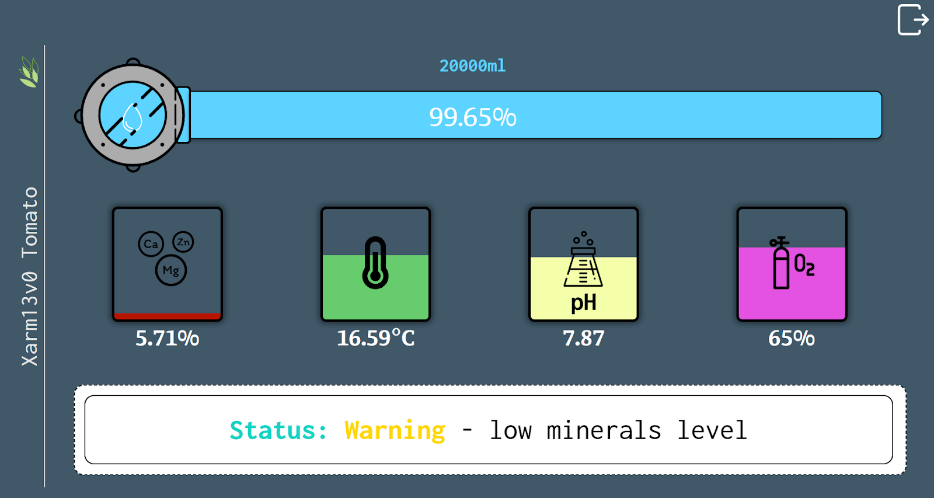


Рисунок 1.17 - в блоці-статус рівень небезпеки жовтий, оскільки зменшення суміші мінералів не так критично як, наприклад, кисень

На рисунку 1.18  показано ще один спосіб контролю/попередження стану гідропоніки шляхом зміни кольору для кожного із 5-ти елементів гідропоніки. Даний контроль/попередження є більш плавним завдяки ступеневої зміни кольору.

Так  компоненти (контейнерів) гідропоніки - вода, кисень та суміш мінералів мають чотири зміни кольору. Зміна кольору відбувається в інтервалах від 75-100 % (перша зміна колір), в інтервалах 50-75 % (друга зміна кольору); в інтервалах 50-25% (третя зміна кольору); в інтервалах 25-0 % (четверта зміна кольору).

Компонент (контейнер) гідропоніки - температура має п’ять змін кольору. За інтервалу абсолютної помилки (тобто помилка взята за модулем)  від 0 до 10 % відбувається перша зміна кольору; за інтервалу загальної помилки (тобто помилка взята за модулем) від 10 до 60 % відбувається зміна кольору у двох варіаціях; за інтервалу критичної помилки (тобто помилка взята за модулем) від 60 до 100 % відбувається зміна кольору у двох варіаціях.

Компонент (контейнер) гідропоніки - pH має п’ять змін кольору. За інтервалу абсолютної помилки (тобто помилка взята за модулем)  від 0 до 5 % відбувається перша зміна кольору; за інтервалу загальної помилки (тобто помилка взята за модулем) від 5 до 60 % відбувається зміна кольору у двох варіаціях; за інтервалу критичної помилки (тобто помилка взята за модулем) від 60 до 100 % відбувається зміна кольору у двох варіаціях.

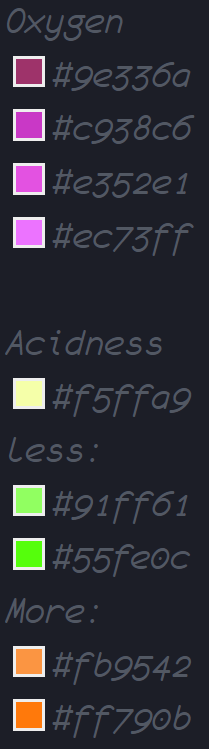


Рисунок 1.18 - кольори записані в форматі hex для різного стану 5-ти елементів гідропоніки

На рисунку 1.19 показана сторінка з блоком “Current account”, в якому здійснюється вхід користувача через введення логіну. Передбачено що логін працює через json web token, що в свою чергу дає можливість видаляти токен з кешу юзера під час виходу користувача із системи гідропоніки “EmeraldWater”. також дані токени мають певний час валідності після якої вони припиняють бути валідними.

Для таких випадків написано скрипт перекидання на сторінку логіну.

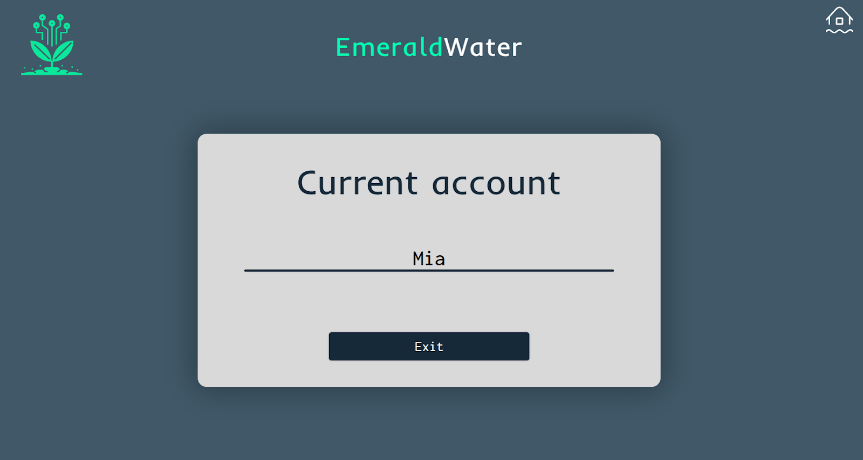


Рисунок 1.19 - сторінка виходу з акаунту

Кнопка виходу “Exit” є клікабельною і під час наведння/натискання на неї змінює колір з блідо сірого на блакитний.

## 1.3 Використані програмні засоби

Проєкт складається з трьох сервісів, бекенд сервера, фронтенд клієнта, та бази даних.

Мова написання бекенд-серверу: Python 3.13.

Як основу бекенд сервера було використано Rust ASGI/WSGI/RSGI сервер Granian, цей сервер було використано у стані ASGI сервера. Для отримання запитів було використано фреймворк FastAPI (він ґрунтується: Starlette для веб-частин; Pydantic для частин даних) [1], загалом варто зазначити, що Pydantic хоч і в рамках фреймворків використовувався доволі значно, наприклад в дата-класі Hydroponic було використано три “кастомних” валідаторів check\_value\_water, check\_value\_minerals, check\_value\_oxygen. Та й в принципі Pydantic семантика використовувалась і для визначення структур запитів, і для визначення структур таблиць в базі даних, тому хоч Pydantic стал ефектом використання фреймворків, проте його використання є значним на серверній частині.

Для спілкування бекенд сервера з базою даних (Microsoft SQL server) було використано драйвер pymssql, проте його використання виконувалось через фреймворк-обгортку SQLModels (що базується на SQLAlchemy і знову ж на Pydantic).

Оскільки додаток має можливість реєстрації та авторизації, відповідно до цього було використано засоби для:

Хешування (із додатковим використанням солі), та валідації паролю й хешу отриманого від джерела даних про паролі (база даних) - для цього використано бібліотеку passlib та алгоритм хешування argon2 (для цього також використано бібліотеку argon2-cffi).

Для отримання/генерації та валідації токенів (Json Web Token) було використано бібліотеку pyjwt, алгоритм хешуваня HS256

Для тестування серверу було використано стандартну бібліотеку unittest [7] та бібліотеку Coverage.py для отримання інформації про покриття коду.

Для тестування API було використано SwaggerUI, Redocly та Bruno (інструмент подібний до Postman).

Мова (мови) написання фронтенду HTML5/CSS3/JavaScript. Також для написання клієнтської частини було використано “набір різних технологій” (suite of different technologies allowing you to create reusable custom elements) [9] WebComponents.

Основним браузером для розробки був браузер Brave.

Для автоматизованого тестування було використано браузер Google Chrome.  Для тестування із записаними макросами було використано Mozilla Firefox.

Для тестування із записаними макросами було використано інструмент Selenium IDE [4], хоч варто зазначити, що під час використання цього інструменту були значні труднощі (деякі дії в автоматизованому тестуванні не вийшло виконувати автоматично, ймовірно через не підтримку WebComponents інструментом Selenium IDE), тому його використання не було основним, й не є рекомендованим для збільшення кількості тестів в застосунку.

Для автоматизованого тестування було використано Selenium WebDriver, мову програмування Python, бібліотеку selenium, та драйвер Chrome.

Для бази даних було використано Microsoft SQL database.

Для тестування бази даних даних було використано фреймворк tSQLt.

Для розробки також використовувались Visual Studio Code (додатково слід зазначити одне з розширень LiveServer) та SQL Server Management Studio 20.

Пакетними менеджерами були: бекенд сервер - пакетний менеджер Poetry [3], веб-клієнт npm. Для фронтенд збірки було використано Vite.

Для тестування навантаження було використано інструмент Jmeter.

## 1.4 Вимоги до проєкту

Для опису вимог буде використано стандартну структуру розділення на функціональні та нефункціональні вимоги (що оновлює надану раніше traceability matrix).

Для опису вимог буде використано стандартну структуру розділення на функціональні та нефункціональні вимоги.

1. Функціональні вимоги:
   1. Управління обліковими записами користувачів:
      1. Користувачі можуть реєструватися, входити та безпечно керувати своїми обліковими записами.
      2. Паролі зберігаються в базі даних у вигляді хешованих і посолених значень.
      3. Веб-токени JSON (JWT) використовуються для управління сеансами і втрачають чинність через певний час.
   2. Управління гідропонними системами:
      1. Користувачі можуть додавати гідропонні системи з наступними параметрами: Назва, максимальна ємність (вода, кисень, мінерали), швидкість споживання (вода, кисень, мінерали в секунду), оптимальні умови (температура, кислотність та кількість мінералів).
      2. Користувачі можуть видаляти гідропонні системи.
      3. Додавання води, кисню та мінералів у резервуари.
      4. Зміна температури та рівня кислотності.
      5. Внутрішній сервер емулює стан і відповідно оновлює базу даних.
   3. Перевірка даних:
      1. На стороні клієнта: Забезпечує належне форматування вхідних даних та блокування надсилання некоректних значень.
      2. На внутрішньому рівні: Забезпечує дотримання обмежень, таких як ємність резервуара та оптимальні діапазони та логічні обмеження (наприклад, кількість води в баці не може бути більшою за розмір баку).
   4. Оновлення в режимі реального часу:
      1. Веб-клієнт періодично запитує оновлення з сервера кожні 5 секунд.
      2. Емуляція фізичних дій на сервері відбувається кожні 60 секунд.
      3. Час змін є контрольованим і може бути зміненим (наприклад за використання сильнішого сервера може бути змінена значення в Python скрипті, також гіпотетично можна додати додаткові можливості для налаштування швидкості/частоти оновлення).
   5. Обробка помилок та критичних ситуацій:
      1. Після закінчення терміну дії JSON Web Token-у користувачі перенаправляються на сторінку входу в систему.
      2. Неправильні вхідні дані повертають описові повідомлення про помилки.
   6. Візуальні індикатори та повідомлення про помилки:
      1. Веб-клієнт використовує кольорові заповнювачі для індикаторів, щоб представити різні рівні для параметрів. Зелений для оптимальних рівнів. Жовтий, помаранчевий та червоний для попереджувального, проміжного та критичного рівнів відповідно.
      2. Окремі гідропоніки на сторінці всіх гідропонік консолідують всі показники гідропонної системи і надають короткий огляд стану окремої системи.
      3. Помилки класифікуються за категоріями та пріоритетами. Критичні помилки (наприклад, низький рівень води) перекривають інші помилки в тій же групі (наприклад, рівень кисню). Помилки з нижчим пріоритетом (наприклад, попередження про мінеральні речовини) придушуються, якщо існує помилка з вищим пріоритетом.
2. Нефункціональні вимоги
   1. Продуктивність:
      1. Система ефективно обробляє запити API та оновлення бази даних, забезпечуючи швидке реагування в реальному часі.
      2. Веб-сторінки оновлюють дані безперебійно, не вимагаючи повного перезавантаження.
   2. Безпека:
      1. Паролі хешуються та “соляться” перед зберіганням. Відсутнє збереження “сирих” паролів на базі даних.
      2. Автентифікація на основі JWT забезпечує безпечний доступ до облікових записів і даних користувачів.
      3. Перевірка даних блокує введення зловмисних даних або такі, що призведуть до негативних наслідків для продукту.
   3. Масштабованість:
      1. Архітектура підтримує декілька клієнтів (наприклад, браузерів, мобільних пристроїв), що взаємодіють з одними і тими ж даними.
   4. Надійність:
      1. Система націлена на узгодженість даних під час взаємодії між клієнтом і сервером.
      2. Періодичні оновлення даних для оновлення станів гідропонних систем.
   5. Зручність використання:
      1. Зрозумілі помилки про коректність вхідних даних на веб-інтерфейсі та на серверному API.
      2. Автоматичне перенаправлення на сторінку входу після закінчення терміну дії токена.
   6. Придатність до змін:
      1. Чіткий розподіл обов'язків між фронтендом, бекендом та базою даних.
      2. Модульна кодова база для полегшення майбутніх оновлень або додавання функцій.

# РОЗДІЛ 2. ТЕСТ-ПЛАН ПРОЄКТУ

**ПЛАН ТЕСТУВАННЯ ДЛЯ ПРОЄКТУ**

**«ЗАСТОСУНОК ДЛЯ МОНІТОРІНГУ ТА КЕРУВАННЯ АВТОМАТИЧНИМИ ГІДРОПОНІКАМИ»**

**EW01-01**

**Підготував**

**Михайлович Олександр Дмитрович**

## 2.1 Ідентифікатор плану тестування

EW01-01

## 2.2 Цілі тестування

Програмне забезпечення, що тестується: Застосунок для моніторингу та керування автоматичними гідропоніками.

Наступний план тестування націлений на тестування (перевірку) та валідацію компонент, цілої системи та окремих сервісів для досягнення наступних цілей:

1. Перевірити коректність обміну даними між клієнтом, бекендом та базою даних.
2. Перевірити коректність оновлення даних в режимі реального часі.
3. Протестувати візуальні індикатори та приоритетизацію помилок.
4. Перевірити коректність роботи системи авторизації (JSON Web Token) та хешування.
5. Перевірити наявність емуляції фізики гідропонних систем.
6. Перевірити загальну роботу системи.
7. Перевірити загальну роботу бази даних.
8. Перевірити коректність веб-клієнта.
9. Перевірити коректність валідацій даних на базі даних.
10. Перевірити коректність валідації даних на фронтенд клієнті.
11. Перевірити усі endpoint-и API.
12. Перевірити коректність обробки низки/ланцюжка запитів.
13. Перевірити наявність всіх компонентів.
14. Перевірити навантаження.

## 2.3 Сфера застосування

Тестовий план призначений для використання з ціллю тестування застосунку EmeraldWater (“Застосунок для моніторингу та керування автоматичними гідропоніками”). Це включає в себе тестування на рівні кожного окремого сервісу, а також на рівні усієї системи. Цей тест план розроблений на основі низки технологій та методів тестування.

## 2.4 Елементи тестування

В процесі тест-плану тестуються наступні компоненти:

1. Клієнтська частина:
   1. Реєстрація, логін, робота з JSON Web Tokens, переадресація на сторінку логіну за закінчення терміну валідності JSON Web Token-у.
   2. Зміна кольорів індикаторів, зміна статусу гідропонік, пріоритетизація помилок.
   3. Коректне оновлення даних внаслідок роботи з декількох клієнтів (наприклад дві різні сторінки в браузері), наявність оновлення даних, без перезавантаження сторінки.
   4. Формування запитів до API та відображення результатів.
2. Бекенд:
   1. Всі endpoint-и API.
   2. Логіка емуляції гідропонних систем.
   3. Правильна робота системи авторизації.
   4. Коректні виклики до бази даних відповідно до логіки API.
   5. Реалізація валідацій.
3. База даних:
   1. Загальна валідація даних.
   2. Валідація даних за допомогою тригер(-a/-ів).
   3. Наявність даних, що надсилає бекенд.

## 2.5 Функціонал, що тестується

### 2.5.1 Тестування хешування на сервері

1. Опис: Тестується хешування й валідація хешу функціями компоненти hasher\_password, перевіряється, що хешування одного й того самого пароля двічі дає значення хеш+сіль, також, що хешування різних паролів не дає однакових результатів та виконується перевірка, що верифікація коректного пароля завершується успішно, а некоректного - завершується помилкою.
2. Пріоритет: Високий - оскільки це є базовою компонентою для всієї системи аутентифікації, на рівні по важливості з коректною роботою бази даних.

### 2.5.2 Тестування коректності JSON Web Token + обгортки драйверу на сервері

1. Опис: перевіряється повна ітерація функцій (коректний користувач, з коректним токеном із коректним хешуванням, з коректними даними, отримує та валідує токен), також перевіряються помилкові випадки некоректний токен, нульові значення полів у токені, використання некоректного користувача (невідповідність id з ім’я акаунту користувача).
2. Пріоритет: Високий - стосується керування об’єктом.

### 2.5.3 Тестування коректності обгортки драйверу + хешування на сервері

1. Опис: перевіряється повна ітерація функцій автентифікації, використовується акаунт, що вже доданий до системи, перевіряється коректність роботи функції перевірки валідації користувача, та функцій отримання користувача за ім’ям акаунту та за ідентифікатором.
2. Пріоритет: Високий - стосується ланцюжка автентифікації.

### 2.5.4 Тестування класу обгортки драйверу на додавання й видалення користувача на сервері

1. Опис: перевіряється коректність додавання та видалення користувача.
2. Пріоритет: Високий, щодо додавання - стосується ланцюжка реєстрації, середній, щодо видалення - оскільки хоч і є частиною керування акаунтом, у цій реалізації веб інтерфейсу не надається користувачу (все одно існує як API endpoint).

### 2.5.5 Тестування класу обгортки драйверу на додавання, видалення, отримання гідропоніки на сервері

1. Опис: додається певна гідропоніка, після цього перевіряється чи вона була дійсно додано, її значення й чи коректно вона була видалена.
2. Пріоритет: Високий — стосується ланцюжка керування гідропонікою.

### 2.5.6 Тестування класу обгортки драйвера на додавання води до гідропоніки на сервері

1. Опис: Спочатку додається клієнт, потім до нього додається гідропоніка, після чого виконується дія для гідропоніки. Далі виконана дія перевіряється - це виражається в порівняльних діях з очікування та фактичного значення води в системі, враховуючи обмеження максимального об’єму води. Далі отримується нове значення гідропоніки, розраховується дельта (на скільки має змінитися значення за функції), і ця дельта порівнюється з певною похибкою (оскільки порівнюються значення з плаваючою комою). Наприкінці клієнт видаляється, а разом із ним каскадно видаляється і гідропоніка.
2. Пріоритет: Високий - тест забезпечує перевірку коректної зміни рівня води в гідропонній системі, яка є ключовою функціональністю системи.

### 2.5.7 Тестування класу обгортки драйвера на додавання мінералів до гідропоніки на сервері

1. Опис: Спочатку додається клієнт, потім до нього додається гідропоніка, після чого виконується дія для гідропоніки, для порівняння (себто валідації/тестування) виконується дії з очікування та порівняння фактичного значення мінералів в системі, враховуючи обмеження максимального об’єму мінералів. Далі отримується нове значення гідропоніки, розраховується дельта (на скільки має змінитися значення за функції), і ця дельта порівнюється з певною похибкою (оскільки порівнюються значення з плаваючою комою). Наприкінці клієнт видаляється, а разом із ним каскадно видаляється і гідропоніка.
2. Пріоритет: Високий - тест забезпечує перевірку коректної зміни рівня мінералів в гідропонній системі, що важливою функціональністю системи.

### 2.5.8 Тестування класу обгортки драйвера на додавання кисню до гідропоніки на сервері

1. Опис: Спочатку в базу даних через бекенд обгортку додається клієнт, потім до нього додається гідропоніка, яка фіксується за юзером, після чого виконується додавання кисню для новододаної гідропоніки. Після підготовчих дій здійснюється аналіз через порівняння значень очікуваного та фактичного значення кисню в системі, з урахуванням його максимального обмеження накладеного об’ємом контейнеру. Далі отримується нове значення гідропоніки, розраховується дельта (на скільки має змінитися значення за виконання функції функції), і ця дельта порівнюється з певною похибкою (оскільки порівнюються значення з плаваючою комою). Наприкінці клієнт видаляється, а разом із ним каскадно видаляється і гідропоніка.
2. Пріоритет: Високий - тест забезпечує перевірку коректної зміни рівня кисню в гідропонній системі.

### 2.5.9 Тестування класу обгортки драйвера на збільшення температури в гідропоніці на сервері

1. Опис: Першим етапом додається клієнт, наступним етапом до нього додається гідропоніка, після чого виконується дія для гідропоніки. Перевіряється виконана дія шляхом порівняння очікуваного та фактичного значення збільшення температури в системі, з урахуванням обмеження на максимальне збільшення температури. Далі отримується нове значення гідропоніки, розраховується дельта (на скільки має змінитися значення за функції), і ця дельта порівнюється з певною похибкою (оскільки порівнюються значення з плаваючою комою). Наприкінці клієнт видаляється, а разом із ним каскадно видаляється і гідропоніка.
2. Пріоритет: Високий - тест забезпечує перевірку коректної зміни у бік збільшення температури в гідропонній системі.

### 2.5.10 Тестування класу обгортки драйвера на зменшення температури в гідропоніці на сервері

1. Опис: Спочатку додається клієнт, потім до нього додається гідропоніка, після чого виконується дія для гідропоніки. Це виражається в порівняльних діях з очікування та фактичного значення зменшення температури в системі, враховуючи обмеження мінімального значення температури. Далі отримується нове значення гідропоніки, розраховується дельта (на скільки має змінитися значення за функції), і ця дельта порівнюється з певною похибкою (оскільки порівнюються значення з плаваючою комою). Наприкінці клієнт видаляється, а разом із ним каскадно видаляється і гідропоніка.
2. Пріоритет: Високий - тест забезпечує перевірку коректної зміни температури у бік зменшення в гідропонній системі, що виступає.

### 2.5.11 Тестування класу обгортки драйвера на збільшення кислотності в гідропоніці на сервері

1. Опис: Спочатку додається клієнт, після чого до нього додається гідропоніка, над якою виконується дія. Це полягає у перевірці відповідності очікуваного та фактичного значень рівня кислотності в системі з урахуванням обмежень на максимальне підвищення кислотності. Потім отримується нове значення гідропоніки, обчислюється дельта (зміна, що мала відбутися згідно з функцією) і порівнюється з допустимою похибкою (враховуючи природу чисел з плаваючою комою). Наприкінці клієнт видаляється, а разом із ним каскадно видаляється і гідропоніка.
2. Пріоритет: Високий — тест гарантує коректність підвищення рівня кислотності в гідропонній системі.

### 2.5.12 Тестування класу обгортки драйвера на зменшення кислотності в гідропоніці на сервері

1. Опис: Спочатку додається клієнт, потім до нього додається гідропоніка, після чого виконується дія для гідропоніки. Це виражається в порівняльних діях з очікування та фактичного значення зменшення температури в системі, враховуючи обмеження мінімального значення температури. Далі отримується нове значення гідропоніки, розраховується дельта (на скільки має змінитися значення за функції), і ця дельта порівнюється з певною похибкою (оскільки порівнюються значення з плаваючою комою). Наприкінці клієнт видаляється, а разом із ним каскадно видаляється і гідропоніка.
2. Пріоритет: Високий - тест забезпечує перевірку коректної зміни рівня води в гідропонній системі.

### 2.5.13 Тестування класу обгортки драйвера на перевантаження гідропоніки на сервері

1. Опис: Спочатку додається клієнт, потім до нього додається гідропоніка, після чого виконується дія для гідропоніки. Викликається функція перевантаження гідропоніки й порівнюється, щоб усі значення були відмінними до й після перевантаження гідропоніки (значення, що оновлюються) - оскільки ці значення є значно випадковими перевірка виконується 10 разів.
2. Пріоритет: Середній - тест забезпечує перевірку коректної дії однієї з основних функціональносте, проте вона є менш важливою ніж більш критичні компоненти/функціональності.

### 2.5.14 Набір тестів для тестування API, й запитів до бекенду із заміною/підміною елементів (використання тестових двійників)

1. Опис: Для тестування було написано функції для тестування: отримання всіх гідропонік, отримання токену (реєстрація), видалення гідропоніки, перевантаження гідропоніки, отримання гідропоніки - знайденої, отримання гідропоніки - не знайдено, додати гідропоніку, перевантаження гідропоніки, керування гідропонікою додавання води, керування гідропонікою додавання мінералів, керування гідропонікою збільшення температури, керування гідропонікою зменшення температури, керування гідропонікою збільшення кислотності, керування гідропонікою зменшення кислотності, реєстрація, запит чи існує користувач із заданим ім’ям аккаунту, видалення користувача, отримання інформації про користувача із токеном, некоректна реєстрація - користувач з таким ім’ям акаунту вже зареєстрований. Перевіряється коректний результат й коректність запитів до бази даних (сама база даних була підмінена (Moсk), хешування було замінено/спрощено (Fake), замість JSON Web Token-а використана затичка (Stub)).
2. Пріоритет: Високий - тест забезпечує перевірку коректної зміни рівня води в гідропонній системі, що виступає основною функціональністю системи.

### 2.5.15 Тестування веб-клієнта, та загальної системи - залогінення

1. Опис: Для тестування було написано автоматичний тест на Selenium driver, для цього вводяться значення в поля інпутів на сторінці логін, натискається кнопка надсилання значень, та перевіряється json web token (правильний юзер - наявність токену).
2. Пріоритет: Високий - тест забезпечує перевірку коректної роботи усієї системи на всьому ланцюжку авторизації.

### 2.5.16 Тестування веб-клієнта, та загальної системи - реєстрація

1. Опис: Для тестування було написано автоматичний тест на Selenium driver, для цього вводяться значення в поля інпутів на сторінці реєстрації, натискається кнопка надсилання значень, та перевіряється json web token (правильний юзер - наявність токену) - акаунт не видаляється оскільки такий функціонал відсутній в даній реалізації веб інтерфейсу.
2. Пріоритет: Високий - тест забезпечує перевірку коректної роботи усієї системи на всьому ланцюжку авторизації.

### 2.5.17 Тестування веб-клієнта, та загальної системи - переадресація за відсутності токена

1. Опис: Для тестування було написано автоматичний тест на Selenium driver, для цього надається запит драйверу на перехід на різні сторінки - для яких має існувати токен для їх роботи, далі перевіряється, що драйвер переадресовується на сторінку авторизації.
2. Пріоритет: Високий - значно важливий функціонал, оскільки без нього теоретично веб клієнт працює некоректно (як мінімум відкриття головної сторінки гіпотетично призводить до помилок).

### 2.5.18 Тестування веб-клієнта, та загальної системи - вихід з акаунту

1. Опис: Для тестування було написано автоматичний тест на Selenium driver (+ підготовчий запит авторизації), для цього надається запит драйверу на перехід до сторінки logout, натискається кнопка виходу з акаунту, та перевіряється видалення токенів.
2. Пріоритет: Високий - важливий функціонал веб-клієнту.

### 2.5.19 Тестування веб-клієнта, та загальної системи - додавання та видалення

1. Опис: Для тестування було написано автоматичний тест на Selenium driver (+ підготовчий запит авторизації), для цього на сторінці усіх гідропонік натискається кнопка додавання гідропоніки, заповнюються дані, додається гідропоніка, обирається гідропоніка на сторінці усіх гідропонік - використовується кнопка видалення - перевіряється, що елемент було видалено (за допомогою перевірки ім’я).
2. Пріоритет: Високий - важливий функціонал веб-клієнту.

### 2.5.20 Тестування веб-клієнта, та загальної системи - керування гідропонікою, кнопка перезавантаження гідропоніки

1. Опис: Для тестування було написано автоматичний тест на Selenium driver (+ підготовчий запит авторизації), після натискання на кнопку "Reset" перевіряється, те щоб як результат цієї дії значення, що підлягають цій зміні (рівень води, мінералів, кисню, температура, кислотність) були змінені на нові.
2. Пріоритет: Середній - важливий але не критичний функціонал веб-клієнту, його можна зімітувати видаленням й додаванням нової гідропоніки.

### 2.5.21 Тестування веб-клієнта, та загальної системи - керування гідропонікою, додавання води

1. Опис: Для тестування було написано автоматичний тест на Selenium driver (+ підготовчий запит авторизації) - тест підготовує гідропоніку для перевірки - він спочатку перевантажує гідропоніку до того моменту поки значення рівня води не буде меншою або рівною 50%, а потім 10 разів (де кожне натискання збільшує рівень на 10%) натискається кнопка для додавання 10% води - очікуване значення ≥95%.
2. Пріоритет: Високий - важливий функціонал веб-клієнту.

### 2.5.22 Тестування веб-клієнта, та загальної системи - керування гідропонікою, додавання мінералів

1. Опис: Для тестування було написано автоматичний тест на Selenium driver (+ підготовчий запит авторизації). Спочатку гідропоніка перевантажується до поки рівень мінералів не стане ≤50%. Далі кнопка для додавання мінералів натискається 20 разів (по 5% за натискання), поки очікуване значення рівня мінералів не досягне ≥95%.
2. Пріоритет: Високий - важливий функціонал веб-клієнту.

### 2.5.23 Тестування веб-клієнта, та загальної системи - керування гідропонікою, додавання кисню

1. Опис: для тестування було написано автоматичний тест на Selenium driver (+ підготовчий запит авторизації). На початку тесту гідропоніка перевантажується до стана, допоки рівень кисню не стане ≤50%. Далі кнопка додавання кисню натискається 20 разів (по 5% за натискання), що націлене на підняття рівень кисню до очікуваного значення ≥95%.
2. Пріоритет: Високий - важливий функціонал веб-клієнту.

### 2.5.24 Тестування веб-клієнта, та загальної системи - керування гідропонікою, керування температурою

1. Опис: для тестування було написано автоматичний тест на Selenium driver (+ підготовчий запит авторизації). Спочатку перевіряється підвищення температури. Гідропоніка перевантажується до стана, коли температура не стане ≤50%. Потім кнопка підвищення температури натискається 20 разів (по 1 градуса за натискання), поки температура не досягне ≥95%. Після цього тестуємо зниження температури, виконуючи аналогічні дії (перевантаження до ≥50%) досягнення значення ≤5% іншою кнопкою.
2. Пріоритет: Високий - важливий функціонал веб-клієнту.

### 2.5.25 Тестування веб-клієнта, та загальної системи - керування гідропонікою, керування кислотністю

1. Опис: для тестування було написано автоматичний тест на Selenium driver (+ підготовчий запит авторизації). Спочатку перевіряється пониження кислотності. Гідропоніка перевантажується до стану, коли кислотності не стане ≤50%. Потім кнопка підвищення лужності натискається 64 рази (по 0.25 pH за натискання), поки лужність не досягне ≥95%. Після цього тестуємо підвищення кислотності, виконуючи аналогічні дії (перевантаження до ≥50%) досягнення значення ≤5% іншою кнопкою (підвищення кислотності).
2. Пріоритет: Високий - важливий функціонал веб-клієнту.

## 2.6 Функціонал, що не тестується

В рамка тест-плану відсутнє тестування навантаження на базу даних, відсутнє тестування реальної взаємодії з контрагентами. Також відсутнє тестування коректності роботи клієнтської частини на екранах з тач-скрінами (оскільки в рамках цього етапу розробки відсутня вимога, щодо такого типу пристроїв). Також відсутнє тестування адаптивності веб-клієнта. Також відсутнє тестування в різних наборах клієнт/операційна система.

## 2.7 Підхід

Під час тестування застосовувались різні підходи, але наявні відмінності пов’язані специфікою того, які саме методи та технології були використані. Провідним у цих тестуваннях було те, що тестування проводились як на рівні декількох сервісів, так і на окремому ізольованому сервісі. Тобто це означає, що тестування певних компонентів було ізольованим для використання від/або початку від/або/і кінці цього сервісу, а тестування значної кількості інших компонентів було протестовано на певному проміжку, наприклад, воно починалось на бекенді і доходило до сервера та назад, або воно починалось на фронтенді і проходило повну ітерацію отримуючи кінцеві значення на фронтенді (себто повні інтеграційні тести системи).

Варто зазначити, загалом в тестуванні було використано (хоч і не усюди) декомпозицію функцій, тобто виділення певних частин функціоналу, код який дублюється у певних функціях, що призвело до зменшення кількості коду, чиє повторення було в зазначених тестах. Це наприклад яскраво показано в кодах написаних для веб-клієнту у Selenium WebDriver, оскільки там повторюються такі дії, як реєстрація клієнту, видалення клієнту тощо.

Також, хоч декомпозиція і виділення окремих частин не було використано в усіх тестах, але вони відіграли значну роль та використовуються в значній кількості функцій.

### 2.7.1 Інструменти тестування

Для тестування використовувалось: unittest - стандартна бібліотека для тестування в Python. Вона була застосована для того, щоб протестувати загальний функціонал бекенду, а також використовувались її піделементи для підміни (використання тестових двійників) такі, як: Fake, Mock та Stub.

Було також використано бібліотеку pytest для виклику тестів, які були зроблені за допомогою Selenium WebDriver (дозволяє робити тестування клієнтської частини себто веб-клієнта) із залученням веб-драйвера Chrome.

Для тестування також було застосовано Selenium IDE для запису макросів, для незначного тестування певних компонентів веб-клієнта.

Використано Apache JMeter для тестування навантаження.

Для тестування також застосовано бібліотеку Coverage.py для отримання показників покриття тестами.

Було використано tSQLt фреймворк для тестування коректності роботи бази даних [6] (було використано Microsoft SQL database) і виконанню всіх обмежень, які були накладені на неї для валідації та перевірки.

### 2.7.2 Збір показників та метрики

Головною метрикою тестування було проходження усіх тестів певного класу та певного файлу, що відображає набір функцій, які виконують конкретну логічну дію або логічно пов’язані у своєму виконанні. Кожен файл, компонента та функція не завжди відноситься до певного сервісу, тобто тестування бекенду є/може бути пов’язаним із тестуванням бази даних, і воно ґрунтується на тому, що тестування нижнього сервісу було виконано коректно, так само і у рівнях вище (себто в цьому випадку компоненти веб-клієнту), де використовувася погляд, що бекенд вже було протестовано.

Кожне тестування бекенду та фронтенду, воно може і часто застосовує окремі результати попередніх тестувань (хоч тести і написані з ціллю можливості тестування без порядку, більш глибоке тестування початкових етапів проводиться спочатку). Це особливо яскраво помітно за порівняння окремих різних класів. Оскільки в певних ситуаціях могла бути використана певна логіка тестування початкових класів, логіка яких принаймні частково може використовуватись (дублюватись) в спеціалізованих функціях підготовки певних інших наступних класів.

Тестування в базі даних є відокремленим, оскільки тестування кожного компоненту бази даних перевіряється наявністю конкретних перевірок (CHECK та трігер(-и)), хоч певні перевірки можуть бути гіпотитично пов’язані.

### 2.7.3 Конфігурації що будуть тестуватися

1. Для тестування бази даних було використано MS SQL 2022, на Windows 11 машині та tSQLt build-V1.0.5873.27393(2014).
2. Для тестування веб браузера за допомогою Selenium Web використано Python 3.13, Pytest 8.3.4, Selenium 4.28.0.
3. Для тестування бекенду використано Python 3.13, Coverage.py 7.6.10.
4. Для ручного тестування веб клієнта використано Brave version 1.74+.

### 2.7.4 Підхід до тестування додатку

Для тестування було використано низку підходів основними з яких можна виділити:

1. Тестування за допомогою коректності - перевірка коректного проходу, з отриманням правильних результатів.
2. Тестування за допомогою перевірки отримання помилок - спроба отримати певні помилки, що виражають правильність роботи системи.
3. Перевірка вводу - полягає в перевірці за допомогою введення коректного, некоректних значень, а також корисним може бути введення випадкових значень.
4. Перевірка за допомогою підмін (тестових двійників) вводить дві додаткові компоненти - тестування з перевіркою виклику певної функції (коректна функція додавання клієнта, має викликати додавання клієнта з певними параметрами), а також очікування значень від визначених функція - ця перевірка означає, що значення, що отримується або є статичним або генерується за певними визначеними й ймовірно простими (хоч і не обов’язково правилами).
5. Важливо зазначити корисність використання покриття тестів, оскільки воно допомагає знайти критичні кінці (кидання помилок тощо), що не є перевіреними, а саме критичні випадки є (принаймні зачасту) винятками, які хоча б і не в наявному коді (хоч гіпотетично й такий код може мати помилки в такій частині, які могли б бути знайдені за певного тесту), так можливо за змін (принаймні гіпотетично) можуть бути не обробленими або обробленими некоректно.

## 2.8 Критерії проходження/провалу для плану тестування

Поданий тест-план має на меті охарактеризувати значний обсяг тестування, тому проходження цього тест плану означає певний рівень коректної роботи всієї системи - таким чином будь яка помилка в цьому тест плані вважається критичною, навіть помилки у візуальній частині, означають некоректну роботу частини, що тест-план націлений гарантувати.

Оскільки помилки можуть бути каскадними, як за появою так і за способом вирішення, незалежно від загального пріоритету, помилки на бекенді мають вищий пріоритет за фронтенд, найвищий рівень помилок це помилки на базі даних та помилки в загальній розгорнутій структурі (неробочий інтерпретатор мови, відсутність коректного з’єднання з портом тощо).

За появи помилок в бібліотеках й залежностях потрібно спробувати перейти на останню стабільну версію, замінити бібліотеку та в крайньому випадку допомогти у виправленні обраної (можливо новообраної) бібліотеки або й почати писати власну (що може призвести до постановки питання про доцільність такої імплементації продукту).

## 2.9 Критерії призупинення та умови відновлення тестування

За отримання критичної помилки тестування має бути зупинене та передане на доопрацювання продукту (критичними помилками є ті, що не дозволяють провести подальше тестування). За умови проходження усіх тестів, окрім певних кінцевих (ті що не зупиняють тестування інших) продукт передається на доопрацювання для вирішення цих помилок.

### 2.9.1 Критерії призупинення тестування

1. Критичні помилки - ті що блокують подальше тестування.
2. Відсутність критичних помилок, проте наявність некритичних.

### 2.9.2 Критерії відновлення тестування

1. Виправлення критичної помилки.
2. Виправлення всіх або частини не критичних помилок.

## 2.10 Результати тестування

Результатом тестування є три стани - наявність критичних помилок, наявність лише не критичних помилок, та повне проходження тест-плану. Також для розуміння рівня проходження тест плану використовується показник кількості пройдених тестів, та як додатковий показник покриття коду коректно виконаними тестами, для приблизного розуміння обширності тест-плану.

## 2.11 Тестові завдання

### 2.11.1 Залежності між тестами

Важливою залежністю в тестуванні беку (назва тест-файлу), на бекенді за допомогою юніт тестів - є зазначена в цьому файлі вимога наявності коректного акаунту Mia із паролем secret. Також наявна залежність верхніх рівнів від нижнів - значна кількість тестів на бекенді пов’язана із коректною роботою та налаштуванням бази даних, подібна ситуація пов’язана і з фронтедом (проте для фронтенда це скоріш пов’язано з функціональними вимогами, що не пов’язані з наявністю певного відображення за наявності даних - себто пов’язаної саме із з’єднанням, оскільки візуальну складову можна тестувати та розробляти з імітацією даних).

### 2.11.2 Рівень навичок

Для написання тестів потрібне знання з Unittest [7], Pytest, Selenium Web driver для Python [4], (не обов’язкове) Selenium IDE, вміння роботи з консоллю розробника у веб-браузері, знання використання Swagger та клієнта для надсилання запросів (наприклад Postman, Bruno, cURL тощо).

### 2.11.3 Ключові етапи

1. Автоматичне тестування бази даних.
2. Автоматичне тестування бекенду.
3. Автоматичне тестування фронтенду.
4. Автоматичне інтеграційне тестування бекенд-база даних.
5. Автоматичне інтеграційне тестування фронтенд-бекенд-база даних.
6. Мануальне тестування API.
7. Мануальне тестування фронтенду.
8. Мануальне тестування загальної системи.

## 2.12 Потреби середовища

### 2.12.1 Джерела інформації

Коректна робота бази даних.

## 2.12.2 Безпека

Захищеність з’єднання (оскільки передаються паролі користувачів та JSON Web Token-и), захищеність бази даних (хоч і використовується хешування й “солення” паролів). Захист від втручання в безпосередній код та базу даних.

### 2.12.3 Відповідальності

Тестувальник та розробник Михайлович Олександр Дмитрович несе відповідальність за якість кінцевого продукту.

## 2.13 Ризики

### 2.13.1 Зміна строків виконання роботи

Через можливу ймовірність наявності критичних помилок, часозатратних багів, та малої кількості розробників може виникнути значний зсув кінцевої здачі роботи.

### 2.13.2 Неповний проєкт

Виходячи із обмежень пов’язаними із об’ємом роботи та обмеженями по часу певний функціонал може бути урізаним та/або зміненим.

## 2.14 Затвердження

Керівник курсової роботи Варава Іван Андрійович виступає в ролі замовника проєкта. Перевіряє стан та цілісність роботи. Визначає доцільність та коректність виконання окремих функцій та функціоналів.

# РОЗДІЛ 3 РОЗРОБКА БАГ РЕПОРТУ

Як результат написання програмного продукту було додано систему Jira, яка позиціонується, як система для відстеження помилок, проблем і гнучкого управління проєктами. Виходячи з того, що система націлена на відстеження помилок, себто багів, на рисунку 3.1 наводиться скріншот цієї системи із баг-трекером.

A screenshot of a test

Description automatically generated

Рисунок 3.1 - червоним квадратом із білим колом в центрі позначені баги

Як результат можна побачити чотири баги про які вестиметиметься мова далі. Хоч можна зазначити, що загальна кількість помилок (ці 4 та не зазначені) була більшою за 4.

## 3.1 Список знайдених багів

1. Помилка взаємодії бекенду з базою даних - за видалення клієнту із наявними гідропоніками, значення зовнішнього ключа гідропоніки занулюється, що призводить до відхилення некоректного запиту сервером.
2. За появи підлінійної помилки на сторінках логіну/реєстрації наявний небажаний ефект зсунення полів/груп вводу.
3. За видалення гідропонік на фронтенді у синхронному режимі (один клієнт видаляє, інший спостерігає за усіма гідропоніками на головній сторінці) помилка про видалення зайвого елементу.
4. Відсутність валідації об'єктів на сервері - помилка взаємодії із базою даних.

## 3.2 Складений баг репорт - баг №1

### 3.2.1 Тема багу

Помилка взаємодії бекенду з базою даних - за видалення клієнту із наявними гідропоніками.

### 3.2.2 Детальний опис багу

За видалення користувача, до якого прив’язані гідропоніки бекенд сервер отримує наступну помилку в консолі (скорочений вивід головної інформації подано на рисунку 3.2). Проблема полягає в тому, що за якоїсь причини рядок, що видаляє користувача призводить до занулення значення користувача пов’язаних в пов’язаних із ним гідропонік. Важливо зазначити, що ця помилка призводить до некоректної роботи бази даних - значення не зберігаються (в цьому випадку значення користувача не видаляються). На додачу до цього також не видаляється пов’язана гідропоніка (з огляду на характер помилки можна доцільно припустити, що зазначена помилка про відсутність видалення пов’язаної гідропоніка також розповсюджується на відсутність видалення декількох пов’язаних гідропонік).

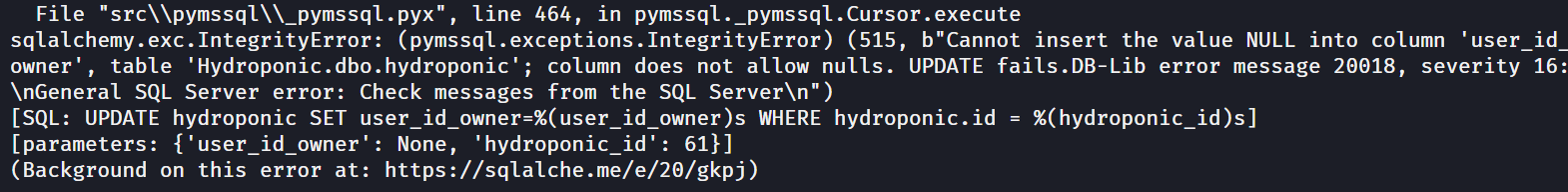


Рисунок 3.2 - помилка внаслідок наявності багу №1

Виходячи з того, що помилку надсилає саме pymssql driver можна припустити про локалізованість проблеми на бекенді.

### 3.2.3 Кроки до відтворення

Для відтворення достатньо запустити тестову функцію test\_change\_hydroponic\_add\_minerals із набору/файлу test\_back.py із пакету server, що знаходиться в директорії Backend.

### 3.2.4 Результати

Очікувалося видалення значень про клієнта, а такоє пов’язаних з ним гідропонік, оскільки, як мінімум на сервері зазначено, що гідропоніки пов’язані із користувачем таким чином, що за видалення користувача, пов’язані гідропоніки мають видалятись каскадно.

### 3.2.5 Інформація про баг

1. Пріорітетність - P1 (Висока).
2. Критичність - S1 (Блокуюча).
3. Статус – виправлений.
4. Автор - Михайлович Олександр Дмитрович.
5. Призначений до - Михайлович Олександр Дмитрович.
6. Оточення:
   1. Python 3.13.
   2. fastapi 0.115.6 (extras: standard).
   3. granian 1.7.4 (extras: reload).
   4. pyjwt 2.10.1.
   5. passlib 1.7.4 (extras: argon2).
   6. argon2-cffi 23.1.0.
   7. pymssql 2.3.2.
   8. sqlmodel 0.0.22.
   9. coverage 7.6.10.
   10. MS SQL 2022.
   11. Windows 11.

## 3.3 Розбір рішення багу

### 3.3.1 Розбір причини проблеми

Проблема виникає також за виклику всіх тестів (всі з яких проходять перевірку) із класу TestChangeHydroponic, до того ж слід звернути увагу, що ця проблема виникає один раз, в обох випадках і для одного тесту, і для багатьох (рисунок 3.3).

A screen shot of a computer code

Description automatically generated

Рисунок 3.3 Виклик усіх 8-ми тестів із класу TestChangeHydroponic

З цього можна зробити висновок, що помилку викликає класові функції підготовки й завершення, а з огляду на те, що всі 8 тестів проходять перевірку із коректним завершенням можна зробити висновок про наявність помилки в функції завершення роботи класу (рисунок 3.4).

В іншому випадку очікувалось би виконання блоку except.

A screenshot of a computer screen

Description automatically generated

Рисунок 3.4 - функція tearDownClass класу TestChangeHydroponic

Оскільки була кинута не визначена помилка за перехоплення HTTPException, можна зробити, що помилку надсилає функція delete\_user. Проте тест, що тестує цю функцію (test\_database\_add\_delete\_user із класу TestAsyncServer) завершується коректно. Та й сама функція не робить дій, що залежать від наявності чи відсутності гідропонік (рисунок 3.5).

A computer code on a black background

Description automatically generated

Рисунок 3.5 - функція delete\_user із модуля server.database\_driver

Проблема ймовірно не полягає у викликах функцій. Проте, якщо поглянути на визначення ORM моделей, варто зазначити, що визначення для таблиці гідропонік явно зазначає каскадне видалення за видалення користувача (рисунок 3.6).

Також варто зауважити про додатковий параметр nullable, що встановлений в позицію False. Проблему може викликати значення за замовчуванням, проте яким чином вона звільняє від виконання видалення за каскадом.

Та й до того ж це в загалом коректний код, навіть приймаючи до уваги код, який може здатись не коректним.

A computer screen shot of text

Description automatically generated

Рисунок 3.6 - ORM модель Hydroponic

Також можна звернути на другу сторону зв’язку, що й призводить до такої поведінки, конкретно в цьому випадку також наявний зв’язок Relationship, проте він не має впливати на наявність такої помилки, оскільки параметр back\_populate встановлений до None.

A computer code on a black background

Description automatically generated

Рисунок 3.7 - ORM модель таблиці user (тобто [user])

### 3.2.2 Огляд можливих рішень

Проблема не має явного вирішення, вона виникає внаслідок незрозумілих факторів. Ймовірно подібних до проблеми із тригером і автоматичним оновленням id за допомогою метода OUTPUT, (про що можна побачити коментар на рисунку 3.6 у визначенні \_\_table\_ars\_\_ ) вона полягала в ймовірно особливій поведінці MS SQL, що й призводила до помилки.

Також можливо, що параметри за замовчуванням ламають коректно налаштовані обмеження.

Або проблема не має явного вирішення, тому потрібно написати “workaround” для видалення користувача із явним видаленням належних йому гідропонік.

### 3.2.3 Результат виправлення

Якщо спробувати закоментувати доволі логічний рядок із визначенням відносин ORM таблиці користувача та таблиці гідропонік (рисунок 3.7, рядок 71) тест (test\_change\_hydroponic\_add\_minerals) починає виконуватись без після помилок (рисунок 3.8).

Тобто можливо помилка знайшлась.

A screen shot of a computer

Description automatically generated

Рисунок 3.8 - виконання тесту test\_change\_hydroponic\_add\_minerals із закоментованим рядком декларації стосунків

Проте просто видаляти не працюючі частини коду було б марним та можливо б ламало б той код, що залежить від нього, або було б легше й логічніше написати із такими стосунками, вирішенням виявилось додати додатковий параметр passive\_deletes, із значенням True (рисунок 3.2) [5].

A screen shot of a computer code

Description automatically generated

Рисунок 3.2 - Виправлення багу №1

Варто зазначити, що в такому випадку видалення гідропонік відбувається завдяки видаленню каскадом задекларованої на базі даних.

# Розділ 4 ТЕСТУВАННЯ ПРОЄКТУ

В цьому розділі подано дані про різні методи, засоби й частини використані за тестування додатку. В ньому зазначено саме автоматизовані тести написані кодом.

## 4.1 Автоматизоване тестування бекенд серверу (unittest, Coverage.py, unittest.mock)

У цьому підрозділі розглядається тестування бекенд сервреу написаного на Python, для програмного продукту EmeraldWater, із використанням unittest, Coverage.py, unittest.mock [2]. Перший й третій використані для відповідно тестування (третій використано тільки для тестування із заміною (тестовими двійниками) що відокремлено в окремий файл де відбувається тестування API). Другий використаний для перевірки покриття коду.

Загалом це тестування знаходиться в одному пакеті, що й увесь бекенд, й розділений на два файли (test\_api.py та test\_back.py) й на значну кількість класів. Загальна кількість тестів 36.

Тестування проводилось в рамках тестування unit тестами та інтеграційними тестами.

### 4.1.1 Переваги та недоліки

Загальною проблемою написання unit тесту, можна розмірковувати, є проблема не написання інтеграційного тесту, приклад, як можна написати unit тести в складному коді, наведено в файлі test\_api.py (там також використовуються тестові двійники) - складність цього тестування полягає в автоматичному додаванні ORM об’єктів, та необхідності знайти таке налаштування, що дозволить відтестувати систему без потреби змінювати системи для полегшення тестування, також другою проблемою були дії які потрібно виконати для налаштування API enpoint-ів, тому тут з’являється подвійна проблема з боку SQLModels [5] та FastAPI [1] проміж яких треба коректно налаштувати тестових двійників. Такі тести є доволі корисними оскільки вони дозволяють ізолювати окрему функцію й перевірити саме її, а не ще з десяток додаткових функцій й викликів, які гіпотетично вже могли бути перевіреними.

У файлі test\_back.py також можна знайти приклад unit тестування, проте ця частина скоріш про інтеграційні тести - в ній перевіряється коректна робота driver-а сервера, його обгорток та бази даних, на цьому етапі можна говорити про тестування версії продукту без інтерфейсів (ні API, ні звісно Web клієнтського інтерфейсу). Проте оскільки API ґрунтується на зазначеній системі із кінцевими точками в обгортках можна розглядати це тестування з точки зору тестування значної частини базового функціоналу. Основою проблемою написання таких тестів є їх далекі виклики, як тільки виклик покидає межі певного сервісу, можна розмірковувати про між сервісні інтеграційні тести, й вони можуть бути надмірними для тестування, яке могло бути проведено у меншому форматі (більша кількість unit-тестів й більша їх кластеризація). Також такі тести можуть бути доволі зв’язуючими, за певних змін струтури, такі тести можуть ламатися й їх потрібно переписувати. З іншого боку такі тести дові корисні, оскільки дозволяють зрозуміти роботи всієї системи за певного зрізу, а не тільки за певного проміжку (порівняння з unit тестами) [7].

Варто зауважити, в результаті тестування було перевірено всі API endpoint-и окрім одного запиту - запиту часу запуску сервера, що можна розглядати, як аналог echo, й він використовувався в розробці - тому можна вважати перевіреним іншими засобами.

### 4.1.2 Середовище написання

Для написання цих тестів використовувалось середовище Visual Studio Code, та наступне оточення:

1. Python 3.13.
2. fastapi 0.115.6 (extras: standard).
3. granian 1.7.4 (extras: reload).
4. pyjwt 2.10.1.
5. passlib 1.7.4 (extras: argon2).
6. argon2-cffi 23.1.0.
7. pymssql 2.3.2.
8. sqlmodel 0.0.22.
9. coverage 7.6.10.
10. MS SQL 2022.
11. Windows 11.

### 4.1.3 Написання тестів

Опис інформації про ці тести можна побачити в попередньому розділі (дивись підрозділ 2.5.1-2.5.14) далі наведено опис двох тестів (по одному з кожного файлу) для пояснення тестування.

На рисунку 4.1 показано інтеграційне тестування із файлу test\_back.py.

A screenshot of a computer screen

Description automatically generated

Рисунок 4.1 - тест перевірки С, R, D частин CRUD операції над ORM гідропонік

Варто зазначити деталь, в попередньому класі TestAsyncServer у цьому ж файлі проведено певне тестування роботи з користувачем, тому в цьому тесті, додавання користувача (яке потрібне для використання гідропоніки) та його видалення виконується в методах класу TestChangeHydroponic setUpClass та tearDownClass відповідно, також використовується функція setUp, в результаті цього тестовані частини досі перевіряються (що дозволяє виконувати тести паралельно) але не займають основну функцію (що дозволяє абстрагуватись під час тестування). Також в цій функції показано два важливих типів перевірки за допомогою self.assert\_ та структури try-except(fail) (також наявна можлифікація для перевірки отримання виключення try-except(pass)-else(fail), яке не наявна в цьому тесті). Також властиво доданий текст до перевірок, а не лише перевірки й аварійне завершення. Цей приклад є прикладом інтеграційного тестування.

На рисунку 4.2 показано unit функцію, що перевіряє один з API enpointit-ів, важливо відмітити, що викристовується два тестових двійники Mock для бази даних та тестовий двійник Stub для роботи з JSON Web Token-ом [2].

A computer screen shot of a program code

Description automatically generated

Рисунок 4.2 - unit тест із використанням тестових двійників

Пропущеним є використання Fake для хещування та валідації паролів. Таким чином ця та інші функції перевіряють коректність роботи викликів API (які є обгорткою) то викликів обгортки драйвера, тому в процесі тестування можна протестувати ті елементи, які гіпотетично могли викликати складність у тому, як саме їх тестувати, через наявність елемента з FastAPI, та наявність елемента, що використовує SQLModel).

На рисунку 4.3 Показано додавання й заміни зазначених в цих елементах (Mock, Stub, Fake). Варто зазначити використання асинхронного Mock, а не звичайного (в цій Python бібліотеці це клас MagicMock). Використання асинхронного мока пов’язане із використанням із тим, що він замінює асинхронні функції.

A screenshot of a computer program

Description automatically generated

Рисунок 4.3 - дві з трьох функцій налаштування/обгорток класу TestHydroponicAPI

Значна проблема виникала із налаштуванням системи, тому нинєшнє налаштування є результатом певної роботи над налаштування правильного порядку дій, та й сами дій.

### 4.1.4 Результат

Як результат цього тестування було написано 36 тестів із сумарним розміром двох файлів приблизно 750+ рядків. Під час тестування як і було зазначено протестовано усе API окрім виклику особливої, яка полягає в тому, щоб отримати час запуску сервера. Цей зазначейний endpoint є особливим, й використовувався він для спеціалізованого js скрипту ціль якого полягала в перевантаженні сторінки, якщо змінювався час запуску сервера, це було то, оскільки lifespan функція перезапускалась за оновлення сервера, що відбувалось завдяки серверу granian та властивості --reload.

A screenshot of a computer program

Description automatically generated

Рисунок 4.4 - проходження усіх 36 тестів

Далі на рисунку 4.5 зображено рівень покриття коду тестами. Показники наступні:

1. Покриття драйвера до бази даних: 78%, окремих компонент:
   1. Безпосередньо обгортка драйвера 87%.
   2. Драйвер 86%.
   3. Емуляція фізичних процесів 29% - перевірка емуляції відбувається завдяки мануального тестування додатку.
2. Роутери й API endpoints - 100%.
3. Безпека й внутрішні елементи - 100%.
4. Головний файл пакету - 50%.

A screenshot of a computer screen

Description automatically generated

Рисунок 4.5 - рівень покриття коду тестами

Варто зазначити, що покриття коду не несе визначальний характер правильності написання тестів, отримання високого рівня покриття показує лише наявність проходу тесту по великій частині функції. Також варто відмітити, що приблизно на відмітках 70%-90% може з’являтися ефект (показники, які просто виглядають доволі коректними), що для кожного додаткового відсотка необхідно зробити ітеративно більшу кількість дій ніж до попереднього (до того ж це скоріш в загальному ніж для кожної функції).

## 4.2 Автоматизоване тестування фронтенд клієнту із використанням Silenium WebDriver

У цьому підрозділі розглядається тестування веб-додатку EmeraldWater, використовуючи Selenium WebDriver та бібліотеку Pytest. Тестування реалізоване у єдиному файлі test\_main.py, що належить пакету seleniumdriver виходячи із суті цих тестів, їх можна назвати повністю інтеграційними - повністю оскільки вони охоплюють взагалі усі компоненти й увесь програмний продукт. Бібліотека Selenium WebDriver використовується за-для відповідно автоматизованої взаємодії з веб-сторінками, а pytest для запуску тестів. Для тестування розроблено 12 тестових функцій і кілька допоміжних класів та функцій, що забезпечують зручне тестування компонентів додатку. Прикладом допоміжного класу може бути клас Control\_page він отримує індикатори, кнопки зі сторінки керування гідропонікою, а також надає функції для отримання значень у відсотках. Варто відзначити, що другий клас New\_page\_inputs не є таким знаяущим - він лише розділяє масив на значення, що зручно використовувати, тому фактично, Control\_page є єдиним представником цієї ідеї з інкапсулюванням отримання компонент у конструкторі об’єкта й їх використання у цих тестах.

Тестування охоплює основні функціональні можливості системи, такими як вхід, реєстрацію, вихід з акаунту, управління гідропоніками (додавання, видалення), і до цього ж контроль на значеннями в гідропоніці (кількість води, мінералів та кисню, й рівні температури та кислотності). Загалом, це дозволяє перевірити коректність роботи усього додатку з точки зору кінцевого користувача.

### 4.2.1 Переваги та недоліки

До переваг відносяться наступні пункти:

1. Інтеграція з UI: Selenium дозволяє автоматизовано взаємодіяти з веб-інтерфейсом, симулюючи поведінку користувача.
2. Гнучкість тестів: Завдяки використанню pytest fixtures, забезпечується ефективне налаштування оточення для тестування, включаючи підготовку тестових даних та очищення після виконання тестів.
3. Перевірка інтеграцій: Завдяки покриттю взаємодії між компонентами (API, база даних, драйвери) забезпечується перевірка роботи системи як цілісної структури.
4. До недоліків відносяться наступні пункти:
5. Чутливість до змін інтерфейсу: Тести залежать від структури веб-сторінок, що робить їх уразливими до змін в HTML/CSS.
6. Ресурсоємність: Selenium WebDriver запускає реальний браузер для виконання тестів, що потребує додаткових ресурсів і часу.
7. Ручне налаштування середовища: Для роботи тестів необхідне встановлення драйвера браузера, що додає складності у налаштуванні CI/CD.

### 4.2.2 Середовище написання

Для написання цих тестів використовувалось середовище Visual Studio Code, та наступне середовище:

1. Python 3.13.
2. pytest 8.3.4.
3. selenium 4.28.0.
4. Windows 11.
5. Веб-драйвер: Google Chrome WebDriver.
6. Інструмент для керування залежностями Poetry [3].

### 4.2.3 Написання тестів

Опис інформації про ці тести можна побачити в попередньому розділі (дивись підрозділ 2.5.15-2.5.25).

Тести розділені на кілька груп:

1. Тести авторизації:
   1. test\_login перевіряє коректність входу в систему.
   2. test\_signup тестує функціональність реєстрації нового користувача.
   3. test\_unlogin перевіряє коректність виходу з облікового запису.
2. Тести перенаправлення:
   1. test\_readdress перевіряє, чи система повертає клієнта на сторінку входу при спробі доступу до захищених сторінок без наявності токена із авторизацією.
3. Тести управління гідропоніками:
   1. test\_add\_delete\_hydroponic тестує додавання й видалення гідропоніки.
4. Тести загального контролю гідропонікою:
   1. test\_reload\_button перевіряє функцію оновлення усіх параметрів (отримання нових випадкових значень).
5. Тести керування гідропонікою
   1. test\_water\_operation, test\_mineral\_operation,   
      test\_oxygen\_operation, test\_temperature\_operation,  
      test\_acidness\_operation перевіряють зміни параметрів гідропонної системи (збільшення/зменшення значень, додавання значень до баку, тощо).

Далі наведено опис одного з початкових (простіших) тестів для пояснення тестування.

Функція test\_login призначена для тестування авторизації користувача у веб-додатку EmeraldWater. У рамках тесту перевіряється коректність введення облікових даних, відображення відповідної сторінки після входу, а також отримання JSON Web Token для подальшої взаємодії з системою.

На рисунку 4.6 показано тест логіну, а також базову test-fixture, яка потім доповнювалась (на основі неї будувалися інші test-fixture).

Тест починається з перевірки того, що поточна сторінка має назву EmeraldWater - login.

Це дозволяє перевірити, що користувач перебуває на сторінці авторизації (це до того ж додаткова перевірка переадресації оскільки користувача мало би бути переадресовано до цієї сторінки із головної).

A screen shot of a computer program

Description automatically generated

Рисунок 4.6 - зображення прикладу тестування на Selenium WebDriver

Використовуючи WebDriverWait, функція очікує, допоки поля для введення логіна та пароля не стануть активними (загалом додавання очікування активності є стандартним на протязі усього тестування). У поле логіна і пароля вводяться значення доданого завчасу акаунту вводиться {“username”: “Mia”, “password”: “secret”}. Тест очікує, що після входу (за натискання відповідної кнопки) користувач буде перенаправлений на головну сторінку. Тест також перевіряє коректність JSON Web Token-а, чи збережений він у localStorage, також він декодується у функції get\_login\_from\_json\_web\_token та перевіряється коректність токену (порівнюється поле sub для username що було введено у поле логін).

### 4.2.4 Результат

Як результат цього тестувння було написано 11 тестів із розміром файлу приблизно 470 рядків. Під час тестування усі тести були пройдені протягом 176.5 секунд, всі тести завершились коректно (рисунок 4.7).

A screenshot of a computer

Description automatically generated

Рисунок 4.7 - результат тестування Selenium WebDriver

Слід зазначити, що налаштування драйвера і мало додаткові нюанси оскільки були використані Shadow DOM-и. Для тестування використано нізу різних способів адресації елементів.

Рівень покриття функціоналу тестами складно оцінити оскільки не використовували засоби покриття тесту веб клієнту, якщо такі існують, тому приблизно можна оцінити покриття тестування наступним чином:

1. Авторизація, реєстрація, вихід з акаунту: ±95%.
2. Додавання, видалення гідропонік: ±90% (окремі сценарії додаються вручну).
3. Контроль параметрів: ±85% (випадкові сценарії емуляції перевіряються частково).
4. Тестування кольорів заливок та тестування станів та помилок: ±10% (тестування відбувається за допомогою мануального тестування).

## 4.3 Інші засоби тестування

На рисунку 4.8 показано redocly одна з двох API, що генеруються сервером на основі схеми (на основі схеми OpenAPI).

A screenshot of a computer

Description automatically generated

Рисунок 4.8 - документація до API сервера (redocly)

За допомогою цієї документації можна дізнатися також про обмеження накладені на поля (що й показано на рисунку 4.8).

На рисунку 4.9 показано другу документацію на основі OpenAPI - SwaggerUI.

A screenshot of a computer

Description automatically generated

Рисунок 4.9 - Swagger UI

У цій документації також можна надсилати запити й отримати інформацію про API server-а. Саме ця документація значно використовувалась під час розробки застосунку.

На рисунку 4.10 показано Selenium IDE - засіб для тестування із записаними макросами.

A screenshot of a computer

Description automatically generated

Рисунок 4.10 - Selenium IDE

Цей інструмент був не зручним у використанні й мав значні проблеми з автоматизацією, які ймовірно виникнули внаслідок використання WebComponents (деякі дії потрібно робити вручну - в автоматизованому тестуванні).

Також наявний Bruno клієнт (програма подібна до Postman), скріншот екрану якого показано на рисунку 4.11.

A screenshot of a computer

Description automatically generated

Рисунок 4.11 - Bruno

Також на рисунку 4.12 можна побачити використання інструменту для контролю навантаження Jmeter - із налаштованим проєктом для отримання значень про навантаження по групах.

A screenshot of a computer

Description automatically generated

Рисунок 4.12 - інструмент перевірки навантаження Jmeter

Всього наявно 4 групи із 10, 100, 1000, 10000 одночасних користувачів.

Для тестування бази даних MS SQL було використано фреймворк tSQLt та написані тести для перевірки роботи бази даних (рисунок 4.13) [6].

A screenshot of a computer program

Description automatically generated

Рисунок 4.13 - виконання tSQLt тестів на базі даних

Всього було написано 18 тестів для бази даних, всі вони були виконані успішно, що видно на рисунку 4.13.

# ВИСНОВОК

Результатом виконання курсової роботи є розроблений клієнт-серверний застосунок для моніторингу та керування автоматичними гідропоніками, що емулює фізичні процеси та забезпечує збереження даних у базі даних MS SQL.

Також було розроблено низку тестів, тест план, діаграми, документації й налаштовані додаткові інструменти для перевірки нефункціональних вимог.

Значну увагу було приділено тестуванню, що було проведене на всіх рівнях проєкту, від клієнтської частини до серверної логіки та бази даних, до того ж різні тести по різному об’єднували ці частини. Одні тести ізольовули компоненти інші перевіряли глибоку інтеграцією з переходом між сервісами.

Для забезпечення якості роботи програмного продукту використовувалися різні підходи та інструменти: unit-тести, інтеграційні тести, тести із записуванням макросів, тести із автоматичною інтеграційною перевіркою веб клієнта, наявні різні клієнти для надсилання HTTP запитів (SwaggerUI, Redocly, Bruno, також наявний додатковий експорт до Postman), використовувалось тестування бази дани, використовувалось оцінювання обширності тестування із застосування покриття коду, використовувались тестові двійники (Mock, Fake та Stub), налаштований та заповнений інструмент для оцінки наваження (Jmeter), також наявний проєкт на Jira.

Під час тестування та розробки були виявлені та усунені проблеми, гарними прикладами яких є:

1. Помилка взаємодії бекенду з базою даних - за видалення клієнту із наявними гідропоніками, значення зовнішнього ключа гідропоніки занулюється, що призводить до відхилення некоректного запиту сервером.
2. За появи підлінійної помилки на сторінках логіну/реєстрації наявний небажаний ефект зсунення полів/груп вводу.
3. За видалення гідропонік на фронтенді у синхронному режимі (один клієнт видаляє, інший спостерігає за усіма гідропоніками на головній сторінці) помилка про видалення зайвого елементу.

Використання різних методи та інструментів допомогло протестувати систему із ціллю розробити стабільну та надійну систему, що відповідає викладеним функціональним та нефункціональним вимогам. Використані методи є методиками різних спектрів, тому тестування відбувалось на різних рівнях із різноманітними підходами.

Загалом можна вважати, що проведене тестування підтвердило коректність розробленого застосунку, його відповідність заданим технічним вимогам і готовність до подальшої експлуатації. Також було додано додаткові інструменти для легшого супроводження додатку.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. FastAPI. FastAPI. URL: https://fastapi.tiangolo.com/ (date of access: 10.01.2025).
2. Mock object library. Python documentation. URL: https://docs.python.org/3/library/unittest.mock.html (date of access: 04.01.2025).
3. Poetry - Python dependency management and packaging made easy. Poetry - Python dependency management and packaging made easy. URL: https://python-poetry.org/ (date of access: 14.01.2025).
4. Selenium with python – selenium python bindings 2 documentation. Selenium with Python – Selenium Python Bindings 2 documentation. URL: https://selenium-python.readthedocs.io/ (date of access: 03.01.2025).
5. SQLModel. SQLModel. URL: https://sqlmodel.tiangolo.com/ (date of access: 06.01.2025).
6. TSQLt - database unit testing for SQL server. tSQLt - Database Unit Testing for SQL Server. URL: https://tsqlt.org/ (date of access: 02.01.2025).
7. Unit testing framework. Python documentation. URL: https://docs.python.org/3/library/unittest.html (date of access: 11.01.2025).
8. Web Components - Web APIs | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web\_components (date of access: 11.01.2025).

# ДОДАТОК A

Код тестування веб-клієнту Selenium WebDriver

**НТУУ «КПІ» ІАТЕ ІПЗЕ**

Листів 19

Київ - 2025

import pytest

from selenium import webdriver

from selenium.webdriver.remote.webelement import WebElement

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.by import By

import base64

import json

import random

@pytest.fixture()

def test\_setup():

global driver

driver = webdriver.Chrome()

driver.get("http://127.0.0.1:5173/")

yield

driver.close()

driver.quit()

def get\_login\_from\_json\_web\_token(token: str):

try:

# Split the JWT into its components: header, payload, and signature

header, payload, signature = token.split('.')

# Decode the payload from base64url format

padded\_payload = payload + '=' \* (-len(payload) % 4) # Add padding if needed

decoded\_payload = base64.urlsafe\_b64decode(padded\_payload).decode('utf-8')

# Convert the JSON string to a Python dictionary

payload\_dict = json.loads(decoded\_payload)

return payload\_dict

except Exception as e:

pytest.fail(f"Error decoding JSON Web Token: {e}")

def test\_login(test\_setup):

title = driver.title

assert title == "EmeraldWater - login", "First page should be login page"

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.XPATH, "//input[@type='text'][@id='login']")))

input\_login: WebElement = driver.find\_element(By.XPATH, "//input[@type='text'][@id='login']")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.XPATH, "//input[@type='password'][@id='password']")))

input\_password: WebElement = driver.find\_element(By.XPATH, "//input[@type='password'][@id='password']")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.XPATH, "//button[contains(text(),'Login')]")))

submit\_button: WebElement = driver.find\_element(By.XPATH, "//button[contains(text(),'Login')]")

input\_login.send\_keys("Mia")

input\_password.send\_keys("secret")

submit\_button.click()

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - hydroponics list")

token = driver.execute\_script("return localStorage.getItem('authToken');")

assert token is not None, "Login failed, no token"

token\_info = get\_login\_from\_json\_web\_token(token)

assert token\_info['sub'] == 'Mia', "Login failed, wrong login in token"

def test\_signup(test\_setup):

title = driver.title

assert title == "EmeraldWater - login", "First page should be login page"

to\_sign\_up\_button: WebElement = driver.find\_element(By.XPATH, "//button[@id='to-sign-up']")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(to\_sign\_up\_button))

to\_sign\_up\_button.click()

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - sign up")

input\_login: WebElement = driver.find\_element(By.XPATH, "//input[@type='text'][@id='login']")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(input\_login))

input\_password: WebElement = driver.find\_element(By.XPATH, "//input[@type='password'][@id='password']")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(input\_password))

submit\_button: WebElement = driver.find\_element(By.XPATH, "//button[contains(text(),'Sign Up')]")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(submit\_button))

username = f"TestUser.{random.randint(1, 100000)}"

input\_login.send\_keys(username)

input\_password.send\_keys("secret")

submit\_button.click()

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - hydroponics list")

token = driver.execute\_script("return localStorage.getItem('authToken');")

assert token is not None, "Sign up failed, no token"

token\_info = get\_login\_from\_json\_web\_token(token)

assert token\_info['sub'] == username, "Sign up failed, wrong login in token"

def test\_readdress(test\_setup):

driver.get("http://127.0.0.1:5173/")

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - login")

driver.get("http://127.0.0.1:5173/src/pages/new/new.html")

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - login")

driver.get("http://127.0.0.1:5173/src/pages/logout/logout.html")

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - login")

driver.get("http://127.0.0.1:5173/src/pages/operating/operating.html?element=1034")

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - login")

@pytest.fixture()

def test\_logined\_setup(test\_setup):

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - login")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.ID, "login")))

input\_login: WebElement = driver.find\_element(By.ID, "login")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.ID, "password")))

input\_password: WebElement = driver.find\_element(By.ID, "password")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.ID, "apply-button")))

submit\_button: WebElement = driver.find\_elements(By.TAG\_NAME, "button")[1]

input\_login.send\_keys("Mia")

input\_password.send\_keys("secret")

submit\_button.click()

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - hydroponics list")

def test\_unlogin(test\_logined\_setup):

driver.get("http://127.0.0.1:5173/src/pages/logout/logout.html")

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - logout")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.CLASS\_NAME, "base-rectangle-button")))

button\_logout: WebElement = driver.find\_element(By.CLASS\_NAME, "base-rectangle-button")

button\_logout.click()

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - login")

assert driver.title == "EmeraldWater - login", "First page after logout should be login page"

token = driver.execute\_script("return localStorage.getItem('authToken');")

assert token is None, "Logout failed, token still exists"

class New\_page\_inputs:

def \_\_init\_\_(self, inputs: list[WebElement]):

self.name = inputs[0]

self.water\_amount = inputs[1]

self.water\_consumption\_speed = inputs[2]

self.minerals\_amount = inputs[3]

self.mineral\_optimal\_amount = inputs[4]

self.mineral\_consumption\_speed = inputs[5]

self.oxygen\_amount = inputs[6]

self.oxygen\_consumption\_speed = inputs[7]

self.acidness\_optimal\_amount = inputs[8]

self.temperature\_optimal\_amount = inputs[9]

def get\_inputs\_from\_new\_page() -> New\_page\_inputs:

trio\_inputs = driver.find\_elements(By.TAG\_NAME, "trio-input")

assert len(trio\_inputs) == 4, "Not 4 trio-inputs"

inputs = []

for i in range(len(trio\_inputs)):

inputs\_temp = trio\_inputs[i].shadow\_root.find\_elements(By.CSS\_SELECTOR, '.input-group:not([style\*="display: none"]) input')

for x in inputs\_temp:

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(x))

inputs.extend(inputs\_temp)

assert len(inputs) == 10, inputs

return New\_page\_inputs(inputs)

def add\_hydroponic\_sub(is\_empty\_add: bool = False) -> str:

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.CSS\_SELECTOR, "a.add-new-button")))

button\_add\_hydroponic: WebElement = driver.find\_element(By.CSS\_SELECTOR, "a.add-new-button")

button\_add\_hydroponic.click()

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - add new hydroponic")

test\_title = "Test hydroponic " + str(random.randint(1, 1000000))

if is\_empty\_add:

input\_values = [

test\_title, "1000", "0.001", "1000", "10",

"0.001", "1000", "0.001", "7.3", "16"

]

else:

input\_values = [

test\_title, "1000", "10", "1000", "10",

"10", "1000", "10", "7.3", "16"

]

inputs = get\_inputs\_from\_new\_page()

inputs.name.send\_keys(input\_values[0])

inputs.water\_amount.send\_keys(input\_values[1])

inputs.water\_consumption\_speed.send\_keys(input\_values[2])

inputs.minerals\_amount.send\_keys(input\_values[3])

inputs.mineral\_optimal\_amount.send\_keys(input\_values[4])

inputs.mineral\_consumption\_speed.send\_keys(input\_values[5])

inputs.oxygen\_amount.send\_keys(input\_values[6])

inputs.oxygen\_consumption\_speed.send\_keys(input\_values[7])

inputs.acidness\_optimal\_amount.send\_keys(input\_values[8])

inputs.temperature\_optimal\_amount.send\_keys(input\_values[9])

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.ID, "apply-button")))

button\_add\_hydroponic: WebElement = driver.find\_element(By.ID, "apply-button")

button\_add\_hydroponic.click()

return test\_title

def delete\_hydroponic\_sub():

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.ID, "button-delete")))

button\_delete: WebElement = driver.find\_element(By.ID, "button-delete")

button\_delete.click()

def test\_add\_delete\_hydroponic(test\_logined\_setup):

test\_title = add\_hydroponic\_sub()

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.XPATH, f"//hydroponic-iter[@name='{test\_title}']")))

hydroponic\_element: WebElement = driver.find\_element(By.XPATH, f"//hydroponic-iter[@name='{test\_title}']")

hydroponic\_element.click()

WebDriverWait(driver, 10).until(lambda driver: driver.title == f"EmeraldWater - operating {test\_title}")

delete\_hydroponic\_sub()

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - hydroponics list")

list\_hydroponic\_element: list[WebElement] = driver.find\_elements(By.XPATH, f"//hydroponic-iter[@name='{test\_title}']")

assert len(list\_hydroponic\_element) == 0, "Hydroponic was not deleted"

@pytest.fixture()

def test\_control\_page(test\_logined\_setup):

# Add and open

test\_title = add\_hydroponic\_sub(True)

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable((By.XPATH, f"//hydroponic-iter[@name='{test\_title}']")))

hydroponic\_element: WebElement = driver.find\_element(By.XPATH, f"//hydroponic-iter[@name='{test\_title}']")

hydroponic\_element.click()

yield

# Delete

delete\_hydroponic\_sub()

WebDriverWait(driver, 10).until(lambda driver: driver.title == "EmeraldWater - hydroponics list")

list\_hydroponic\_element: list[WebElement] = driver.find\_elements(By.XPATH, f"//hydroponic-iter[@name='{test\_title}']")

assert len(list\_hydroponic\_element) == 0, "Hydroponic was not deleted"

class Control\_page:

def \_\_init\_\_(self):

# water

self.water\_indicator = driver.find\_element(By.ID, "water-tank-progress-bar")

WebDriverWait(driver, 3).until(lambda driver: self.water\_indicator.is\_displayed())

self.water\_increase\_button = driver.find\_element(By.ID, "button-add-water")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(self.water\_increase\_button))

# minerals

self.mineral\_indicator = driver.find\_element(By.ID, "mineral-progress-bar")

WebDriverWait(driver, 3).until(lambda driver: self.mineral\_indicator.is\_displayed())

self.mineral\_increase\_button = driver.find\_element(By.ID, "button-add-mineral")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(self.mineral\_increase\_button))

# temperature

self.temperature\_indicator = driver.find\_element(By.ID, "temperature-progress-bar")

WebDriverWait(driver, 3).until(lambda driver: self.temperature\_indicator.is\_displayed())

self.temperature\_increase\_button = driver.find\_element(By.ID, "button-add-temperature")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(self.temperature\_increase\_button))

self.temperature\_decrease\_button = driver.find\_element(By.ID, "button-lower-temperature")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(self.temperature\_decrease\_button))

# acidndness

self.acidness\_indicator = driver.find\_element(By.ID, "acidity-progress-bar")

WebDriverWait(driver, 3).until(lambda driver: self.acidness\_indicator.is\_displayed())

self.acidness\_increase\_button = driver.find\_element(By.ID, "button-add-acidity")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(self.acidness\_increase\_button))

self.acidness\_decrease\_button = driver.find\_element(By.ID, "button-lower-acidity")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(self.acidness\_decrease\_button))

# oxygen

self.oxygen\_indicator = driver.find\_element(By.ID, "oxygen-progress-bar")

WebDriverWait(driver, 3).until(lambda driver: self.oxygen\_indicator.is\_displayed())

self.oxygen\_increase\_button = driver.find\_element(By.ID, "button-add-oxygen")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(self.oxygen\_increase\_button))

# controls

self.reload\_button = driver.find\_element(By.ID, "button-reload")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(self.reload\_button))

self.delete\_button = driver.find\_element(By.ID, "button-delete")

WebDriverWait(driver, 3).until(EC.element\_to\_be\_clickable(self.delete\_button))

def get\_water\_percentage(self) -> float:

water\_percentage\_str = self.water\_indicator.get\_attribute("percentage")

assert water\_percentage\_str is not None, "Water percentage attribute is None"

water\_percentage = float(water\_percentage\_str)

return water\_percentage

def get\_mineral\_percentage(self) -> float:

mineral\_percentage\_str = self.mineral\_indicator.get\_attribute("percentage")

assert mineral\_percentage\_str is not None, "Mineral percentage attribute is None"

mineral\_percentage = float(mineral\_percentage\_str)

return mineral\_percentage

def get\_temperature\_percentage(self) -> float:

temperature\_percentage\_str = self.temperature\_indicator.get\_attribute("percentage")

assert temperature\_percentage\_str is not None, "Temperature percentage attribute is None"

temperature\_percentage = float(temperature\_percentage\_str)

return temperature\_percentage

def get\_acidness\_percentage(self) -> float:

acidness\_percentage\_str = self.acidness\_indicator.get\_attribute("percentage")

assert acidness\_percentage\_str is not None, "Acidness percentage attribute is None"

acidness\_percentage = float(acidness\_percentage\_str)

return acidness\_percentage

def get\_oxygen\_percentage(self) -> float:

oxygen\_percentage\_str = self.oxygen\_indicator.get\_attribute("percentage")

assert oxygen\_percentage\_str is not None, "Oxygen percentage attribute is None"

oxygen\_percentage = float(oxygen\_percentage\_str)

return oxygen\_percentage

def test\_reload\_button(test\_control\_page):

control\_page = Control\_page()

water\_percentage = control\_page.get\_water\_percentage()

mineral\_percentage = control\_page.get\_mineral\_percentage()

temperature\_percentage = control\_page.get\_temperature\_percentage()

acidness\_percentage = control\_page.get\_acidness\_percentage()

oxygen\_percentage = control\_page.get\_oxygen\_percentage()

# RELOAD

control\_page.reload\_button.click()

WebDriverWait(driver, 3).until(lambda driver: water\_percentage != control\_page.get\_water\_percentage())

water\_percentage\_new = control\_page.get\_water\_percentage()

mineral\_percentage\_new = control\_page.get\_mineral\_percentage()

temperature\_percentage\_new = control\_page.get\_temperature\_percentage()

acidness\_percentage\_new = control\_page.get\_acidness\_percentage()

oxygen\_percentage\_new = control\_page.get\_oxygen\_percentage()

# Checks

assert abs(water\_percentage - water\_percentage\_new) > 0.01, "Water percentage is not changed, or bad random"

assert abs(mineral\_percentage - mineral\_percentage\_new) > 0.01, "Mineral percentage is not changed, or bad random"

assert abs(temperature\_percentage - temperature\_percentage\_new) > 0.01, "Temperature percentage is not changed, or bad random"

assert abs(acidness\_percentage - acidness\_percentage\_new) > 0.01, "Acidness percentage is not changed, or bad random"

assert abs(oxygen\_percentage - oxygen\_percentage\_new) > 0.01, "Oxygen percentage is not changed, or bad random"

def test\_water\_operation(test\_control\_page):

control\_page = Control\_page()

water\_percentage = control\_page.get\_water\_percentage()

while water\_percentage > 50:

control\_page.reload\_button.click()

WebDriverWait(driver, 3).until(lambda driver: water\_percentage != control\_page.get\_water\_percentage())

water\_percentage = control\_page.get\_water\_percentage()

for i in range(10):

control\_page.water\_increase\_button.click()

WebDriverWait(driver, 3).until(lambda driver: water\_percentage != control\_page.get\_water\_percentage())

water\_percentage = control\_page.get\_water\_percentage()

assert water\_percentage >= 95, "Water percentage is not increased"

def test\_mineral\_operation(test\_control\_page):

control\_page = Control\_page()

mineral\_percentage = control\_page.get\_mineral\_percentage()

while mineral\_percentage > 50:

control\_page.reload\_button.click()

WebDriverWait(driver, 3).until(lambda driver: mineral\_percentage != control\_page.get\_mineral\_percentage())

mineral\_percentage = control\_page.get\_mineral\_percentage()

for i in range(20):

control\_page.mineral\_increase\_button.click()

WebDriverWait(driver, 3).until(lambda driver: mineral\_percentage != control\_page.get\_mineral\_percentage())

mineral\_percentage = control\_page.get\_mineral\_percentage()

assert mineral\_percentage >= 95, "Mineral percentage is not increased"

def test\_oxygen\_operation(test\_control\_page):

control\_page = Control\_page()

oxygen\_percentage = control\_page.get\_oxygen\_percentage()

while oxygen\_percentage > 50:

control\_page.reload\_button.click()

WebDriverWait(driver, 3).until(lambda driver: oxygen\_percentage != control\_page.get\_oxygen\_percentage())

oxygen\_percentage = control\_page.get\_oxygen\_percentage()

for i in range(20):

control\_page.oxygen\_increase\_button.click()

WebDriverWait(driver, 3).until(lambda driver: oxygen\_percentage != control\_page.get\_oxygen\_percentage())

oxygen\_percentage = control\_page.get\_oxygen\_percentage()

def test\_temperature\_operation(test\_control\_page):

control\_page = Control\_page()

temperature\_percentage = control\_page.get\_temperature\_percentage()

# Test increase

while temperature\_percentage > 50:

control\_page.reload\_button.click()

WebDriverWait(driver, 3).until(lambda driver: temperature\_percentage != control\_page.get\_temperature\_percentage())

temperature\_percentage = control\_page.get\_temperature\_percentage()

for i in range(20):

control\_page.temperature\_increase\_button.click()

WebDriverWait(driver, 3).until(lambda driver: temperature\_percentage != control\_page.get\_temperature\_percentage())

temperature\_percentage = control\_page.get\_temperature\_percentage()

assert temperature\_percentage >= 95, "Temperature percentage is not increased"

# Test decrease

while temperature\_percentage < 50:

control\_page.reload\_button.click()

WebDriverWait(driver, 3).until(lambda driver: temperature\_percentage != control\_page.get\_temperature\_percentage())

temperature\_percentage = control\_page.get\_temperature\_percentage()

for i in range(20):

control\_page.temperature\_decrease\_button.click()

WebDriverWait(driver, 3).until(lambda driver: temperature\_percentage != control\_page.get\_temperature\_percentage())

temperature\_percentage = control\_page.get\_temperature\_percentage()

assert temperature\_percentage <= 5, "Temperature percentage is not decreased"

def test\_acidness\_operation(test\_control\_page):

control\_page = Control\_page()

acidness\_percentage = control\_page.get\_acidness\_percentage()

# Test increase - is reversed to other due to how accidness is represented

while acidness\_percentage < 50:

control\_page.reload\_button.click()

WebDriverWait(driver, 3).until(lambda driver: acidness\_percentage != control\_page.get\_acidness\_percentage())

acidness\_percentage = control\_page.get\_acidness\_percentage()

for i in range(64):

control\_page.acidness\_increase\_button.click()

WebDriverWait(driver, 3).until(lambda driver: acidness\_percentage != control\_page.get\_acidness\_percentage())

acidness\_percentage = control\_page.get\_acidness\_percentage()

assert acidness\_percentage <= 5, "Acidness percentage is not increased"

# Test decrease - is reversed to other due to how accidness is represented

while acidness\_percentage > 50:

control\_page.reload\_button.click()

WebDriverWait(driver, 3).until(lambda driver: acidness\_percentage != control\_page.get\_acidness\_percentage())

acidness\_percentage = control\_page.get\_acidness\_percentage()

for i in range(64):

control\_page.acidness\_decrease\_button.click()

WebDriverWait(driver, 3).until(lambda driver: acidness\_percentage != control\_page.get\_acidness\_percentage())

acidness\_percentage = control\_page.get\_acidness\_percentage()

assert acidness\_percentage >= 95, "Acidness percentage is not decreased"

# ДОДАТОК B

Код тестування unit тестами із застосування підміни тестовими двійниками

**НТУУ «КПІ» ІАТЕ ІПЗЕ**

Листів 14

Київ - 2025

import unittest

from unittest.mock import AsyncMock, patch

from server.security.dataclasses import User, UserInDB

from server.security.json\_web\_token import process\_access\_token

from server.security.security import add\_token\_endpoint, add\_test\_endpoint

from fastapi.testclient import TestClient

FAKE\_USER\_ID = 1

FAKE\_USER\_USERNAME = "test\_user"

class FakeHasher:

@staticmethod

def hash\_password(password: str) -> str:

return password # Just returns the same password (no real hashing).

@staticmethod

def verify\_password(password: str, hashed\_password: str) -> bool:

return password == hashed\_password # Compares directly without hashing.

class TestHydroponicAPI(unittest.IsolatedAsyncioTestCase):

@classmethod

def setUpClass(cls):

# Mock Database

cls.db\_mock = AsyncMock()

cls.database\_patcher = patch("server.database\_driver.database", cls.db\_mock)

cls.database\_patcher2 = patch("server.security.security.database\_manager", cls.db\_mock)

cls.database\_patcher.start()

cls.database\_patcher2.start()

from server.main import app # should be after db mock

# Fake Hasher

cls.hasher\_fake = patch("server.security.hasher\_password.hash\_password", FakeHasher.hash\_password)

cls.verifier\_fake = patch("server.security.hasher\_password.verify\_password", FakeHasher.verify\_password)

cls.hasher\_fake.start()

cls.verifier\_fake.start()

# Stub JWT

app.dependency\_overrides[process\_access\_token] = lambda: User(id=FAKE\_USER\_ID, username=FAKE\_USER\_USERNAME)

cls.stub\_jwt\_generate = patch(

"server.security.json\_web\_token.generate\_access\_token",

lambda \*args, \*\*kwargs: "fake\_token"

)

cls.stub\_jwt\_generate.start()

# Add token and test endpoints

add\_token\_endpoint(app)

add\_test\_endpoint(app)

# Test client

cls.client = TestClient(app)

def setUp(self):

# Reset mocks

self.db\_mock.reset\_mock()

@classmethod

def tearDownClass(cls):

cls.database\_patcher.stop()

cls.database\_patcher2.stop()

cls.hasher\_fake.stop()

cls.stub\_jwt\_generate.stop()

cls.verifier\_fake.stop()

async def test\_get\_all\_hydroponics(self):

# Simulate mocked database returning hydroponic data

from server.database\_driver.dataclasses import Hydroponic\_response

self.db\_mock.get\_all\_hydroponics.return\_value = [

Hydroponic\_response(

id=1, name="Arm1", water\_amount=1000, water\_consumption=1,

minerals\_amount=12, minerals\_optimal=12, minerals\_consumption=12,

acidity\_optimal\_ph=12, temperature\_C\_optimal=12, oxygen\_amount=12,

oxygen\_consumption=12, value\_water=12, value\_minerals=12,

value\_acidity\_ph=12, value\_temperature\_C=12, value\_oxygen=12

),

Hydroponic\_response(

id=2, name="Arm2", water\_amount=1000, water\_consumption=1,

minerals\_amount=12, minerals\_optimal=12, minerals\_consumption=12,

acidity\_optimal\_ph=12, temperature\_C\_optimal=12, oxygen\_amount=12,

oxygen\_consumption=12, value\_water=12, value\_minerals=12,

value\_acidity\_ph=12, value\_temperature\_C=12, value\_oxygen=12

)

]

# Make API call

response = self.client.get("/api/hydroponic/all", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.get\_all\_hydroponics.assert\_called\_once\_with(FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(len(response.json()), 2)

self.assertEqual(response.json()[0]["name"], "Arm1")

async def test\_generate\_token(self):

# Prepare a mocked UserInDB object

mocked\_user = UserInDB(

id=FAKE\_USER\_ID,

username=FAKE\_USER\_USERNAME,

hashed\_password=FakeHasher.hash\_password("password") # Match the expected value for password verification

)

# Mock the get\_user\_by\_username method to return the mocked\_user

self.db\_mock.get\_user\_by\_username.return\_value = mocked\_user

# Call the token endpoint with the correct data

response = self.client.post("/token", data={"username": FAKE\_USER\_USERNAME, "password": FakeHasher.hash\_password("password")})

self.db\_mock.get\_user\_by\_username.assert\_called\_once\_with(FAKE\_USER\_USERNAME)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertIn("access\_token", response.json())

self.assertEqual(response.json()["access\_token"], "fake\_token")

async def test\_delete\_hydroponic(self):

# Simulate mocked database deleting hydroponic data

self.db\_mock.delete\_hydroponic.return\_value = True

# Make API call

response = self.client.delete("/api/hydroponic/1", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.delete\_hydroponic.assert\_called\_once\_with(1, FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), None)

async def test\_get\_hydroponic\_found(self):

from server.database\_driver.dataclasses import Hydroponic\_response

# Simulate mocked database returning hydroponic data

self.db\_mock.get\_hydroponic\_by\_id.return\_value = Hydroponic\_response(

id=1, name="Arm1", water\_amount=1000, water\_consumption=1,

minerals\_amount=12, minerals\_optimal=12, minerals\_consumption=12,

acidity\_optimal\_ph=12, temperature\_C\_optimal=12, oxygen\_amount=12,

oxygen\_consumption=12, value\_water=12, value\_minerals=12,

value\_acidity\_ph=12, value\_temperature\_C=12, value\_oxygen=12

)

# Make API call

response = self.client.get("/api/hydroponic/1", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.get\_hydroponic\_by\_id.assert\_called\_once\_with(1, FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json()["name"], "Arm1")

self.assertEqual(response.json()["id"], 1)

self.assertEqual(response.json()["water\_amount"], 1000)

self.assertEqual(response.json()["water\_consumption"], 1)

self.assertEqual(response.json()["minerals\_amount"], 12)

self.assertEqual(response.json()["minerals\_optimal"], 12)

self.assertEqual(response.json()["minerals\_consumption"], 12)

self.assertEqual(response.json()["acidity\_optimal\_ph"], 12)

self.assertEqual(response.json()["temperature\_C\_optimal"], 12)

self.assertEqual(response.json()["oxygen\_amount"], 12)

self.assertEqual(response.json()["oxygen\_consumption"], 12)

self.assertEqual(response.json()["value\_water"], 12)

self.assertEqual(response.json()["value\_minerals"], 12)

self.assertEqual(response.json()["value\_acidity\_ph"], 12)

self.assertEqual(response.json()["value\_temperature\_C"], 12)

self.assertEqual(response.json()["value\_oxygen"], 12)

async def test\_get\_hydroponic\_not\_found(self):

# Simulate mocked database returning None

self.db\_mock.get\_hydroponic\_by\_id.return\_value = None

# Make API call

response = self.client.get("/api/hydroponic/1", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.get\_hydroponic\_by\_id.assert\_called\_once\_with(1, FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 404)

self.assertEqual(response.json(), {"detail": "Hydroponic not found"})

async def test\_add\_hydroponic(self):

# Simulate mocked database adding hydroponic data

from server.database\_driver.dataclasses import Hydroponic\_input

self.db\_mock.add\_hydroponic.return\_value = True

input\_obj = Hydroponic\_input(

name="Arm1", water\_amount=1000, water\_consumption=1,

minerals\_amount=12, minerals\_optimal=12, minerals\_consumption=12,

acidity\_optimal\_ph=12, temperature\_C\_optimal=12, oxygen\_consumption=12,

oxygen\_amount=12

)

# Make API callq

response = self.client.post("/api/hydroponic/add", json=input\_obj.model\_dump(), headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.add\_hydroponic.assert\_called\_once\_with(input\_obj, FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), None)

async def test\_reset\_hydroponic(self):

# Simulate mocked database resetting hydroponic data

self.db\_mock.reset.return\_value = True

# Make API call

response = self.client.patch("/api/hydroponic/reset/1", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.reset.assert\_called\_once\_with(1, FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), None)

async def test\_add\_10\_percent\_water(self):

# Simulate mocked database adding 10% water

self.db\_mock.add\_10\_percent\_water.return\_value = True

# Make API call

response = self.client.patch("/api/hydroponic/1/update/water/add\_10\_percent", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.add\_10\_percent\_water.assert\_called\_once\_with(1, FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), None)

async def test\_add\_5\_percent\_minerals(self):

# Simulate mocked database adding 5% minerals

self.db\_mock.add\_5\_percent\_minerals.return\_value = True

# Make API call

response = self.client.patch("/api/hydroponic/1/update/minerals/add\_5\_percent", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.add\_5\_percent\_minerals.assert\_called\_once\_with(1, FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), None)

async def test\_add\_1\_celsius\_temperature(self):

# Simulate mocked database adding 1 celsius temperature

self.db\_mock.add\_1\_celsius\_temperature.return\_value = True

# Make API call

response = self.client.patch("/api/hydroponic/1/update/temperature/add\_1\_celsius", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.add\_1\_celsius\_temperature.assert\_called\_once\_with(1, FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), None)

async def test\_lower\_1\_celsius\_temperature(self):

# Simulate mocked database adding 1 celsius temperature

self.db\_mock.lower\_1\_celsius\_temperature.return\_value = True

# Make API call

response = self.client.patch("/api/hydroponic/1/update/temperature/lower\_1\_celsius", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.lower\_1\_celsius\_temperature.assert\_called\_once\_with(1, FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), None)

async def test\_add\_0\_25\_acid(self):

# Simulate mocked database adding 0.25 acid

self.db\_mock.add\_0\_25\_acid.return\_value = True

# Make API call

response = self.client.patch("/api/hydroponic/1/update/acidity/add\_0\_25", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.add\_0\_25\_acid.assert\_called\_once\_with(1, FAKE\_USER\_ID)\

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), None)

async def test\_lower\_0\_25\_acid(self):

# Simulate mocked database adding 0.25 acid

self.db\_mock.lower\_0\_25\_acid.return\_value = True

# Make API call

response = self.client.patch("/api/hydroponic/1/update/acidity/lower\_0\_25", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.lower\_0\_25\_acid.assert\_called\_once\_with(1, FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), None)

async def test\_add\_5\_percent\_oxygen(self):

# Simulate mocked database adding 5% oxygen

self.db\_mock.add\_5\_percent\_oxygen.return\_value = True

# Make API call

response = self.client.patch("/api/hydroponic/1/update/oxygen/add\_5\_percent", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.add\_5\_percent\_oxygen.assert\_called\_once\_with(1, FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), None)

async def test\_register(self):

# Simulate mocked database adding user

self.db\_mock.add\_user.return\_value = True

self.db\_mock.get\_user\_by\_username.return\_value = UserInDB(id=1, username=FAKE\_USER\_USERNAME, hashed\_password=FakeHasher.hash\_password("password"))

# Make API call

response = self.client.post("/register", data={

"username": FAKE\_USER\_USERNAME,

"password": "password"

})

self.db\_mock.add\_user.assert\_called\_once\_with(FAKE\_USER\_USERNAME, "password")

self.db\_mock.get\_user\_by\_username.assert\_called\_once\_with(FAKE\_USER\_USERNAME)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), {'access\_token': 'fake\_token', 'token\_type': 'bearer'})

async def test\_user\_exist(self):

# Simulate mocked database checking username

self.db\_mock.check\_username\_exists.return\_value = True

# Make API call

response = self.client.get(f"/user/exists?username={FAKE\_USER\_USERNAME}")

self.db\_mock.check\_username\_exists.assert\_called\_once\_with(FAKE\_USER\_USERNAME)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), True)

async def test\_delete\_user(self):

# Simulate mocked database deleting user

self.db\_mock.delete\_user.return\_value = True

# Make API call

response = self.client.delete("/user", headers={"Authorization": "Bearer fake\_token"})

self.db\_mock.delete\_user.assert\_called\_once\_with(FAKE\_USER\_ID)

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), None)

async def test\_read\_users\_me(self):

# Make API call

response = self.client.get("/users/me", headers={"Authorization": "Bearer fake\_token"})

# Assertions

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), {"id": FAKE\_USER\_ID, "username": FAKE\_USER\_USERNAME})

async def test\_register\_user\_already\_exists(self):

# Simulate mocked database adding user

self.db\_mock.add\_user.return\_value = False

# Make API call

response = self.client.post("/register", data={

"username": FAKE\_USER\_USERNAME,

"password": "testpassword"

})

self.db\_mock.add\_user.assert\_called\_once\_with(FAKE\_USER\_USERNAME, "testpassword")

# Assertions

self.assertEqual(response.status\_code, 409)

self.assertEqual(response.json(), {"detail": "User already exists"})

# ДОДАТОК C

Код тестування інтеграційними тестами (та невеликою кількістю unit тестів) без застосування підміни тестовими двійниками

**НТУУ «КПІ» ІАТЕ ІПЗЕ**

Листів 17

Київ - 2025

# WARNING if test fails check database that is been tested

# USER - Username: Mia, password: secret | should exist

import unittest

from fastapi import HTTPException

import random

import asyncio

from server.database\_driver.dataclasses import Hydroponic, Hydroponic\_input

from server.security.dataclasses import UserInDB

import server.security.hasher\_password as hasher

import server.security.json\_web\_token as token\_provider

import server.security.security as security

import server.database\_driver.database as database\_manager

TEST\_USERNAME = "Mia" # shoud be real user in database

TEST\_PASSWORD = "secret" # shoud be real user in database

TEST\_HYDROPONIC = Hydroponic\_input(

name="test xHydro3456",

water\_amount=1000,

water\_consumption=1,

minerals\_amount=1000,

minerals\_optimal=1,

minerals\_consumption=1,

acidity\_optimal\_ph=7,

temperature\_C\_optimal=19,

oxygen\_amount=1000,

oxygen\_consumption=1

)

class TestServer(unittest.TestCase):

def test\_hash(self):

test1 = hasher.hash\_password("test")

test2 = hasher.hash\_password("test")

wrong = hasher.hash\_password("wrong")

self.assertNotEqual(test1, test2, "The same password hashed should not be the same, yet may")

self.assertNotEqual(test1, wrong, "Different passwords hashed should not be the same, yet may")

self.assertNotEqual(test2, wrong, "Different passwords hashed should not be the same, yet may")

result1 = hasher.verify\_password("test", test1)

result2 = hasher.verify\_password("test", test2)

result3 = hasher.verify\_password("wrong", wrong)

result3\_error = hasher.verify\_password("test", wrong)

self.assertTrue(result1, "The password should be verified")

self.assertTrue(result2, "The password should be verified")

self.assertTrue(result3, "TThe password should be verified")

self.assertFalse(result3\_error, "The password should not be verified")

class TestAsyncServer(unittest.IsolatedAsyncioTestCase):

# # Check if the user exists in the database

# database\_user = await database\_manager.get\_user\_by\_id(user\_id)

# if database\_user is None or database\_user.username != username:

# raise credentials\_exception

async def test\_json\_web\_token\_correct\_user(self):

database\_user = await database\_manager.get\_user\_by\_username(TEST\_USERNAME)

if database\_user is None:

self.fail(f"User {TEST\_USERNAME} is absent from Database")

correct\_user = {

"sub": TEST\_USERNAME,

"id": database\_user.id

}

token\_correct\_user = token\_provider.generate\_access\_token(correct\_user)

correct\_user\_result = await token\_provider.process\_access\_token(token\_correct\_user)

self.assertEqual(correct\_user\_result.id, database\_user.id, "Provided username should be equal to what ws returned")

self.assertEqual(correct\_user\_result.username, TEST\_USERNAME, "Provided id should be equal to what ws returned")

async def test\_json\_web\_token\_invalid\_token(self):

try:

await token\_provider.process\_access\_token("incorrect\_token")

except HTTPException:

pass

else:

self.fail("Incorrect token has passed")

async def test\_json\_web\_token\_null\_userinfo(self):

# values aren't important in this case - it should raise in any case

none\_user1 = {

"sub": None,

"id": 1

}

token1 = token\_provider.generate\_access\_token(none\_user1)

none\_user2 = {

"sub": "Some value",

"id": None

}

token2 = token\_provider.generate\_access\_token(none\_user2)

none\_user3 = {

"sub": None,

"id": None

}

token3 = token\_provider.generate\_access\_token(none\_user3)

try:

await token\_provider.process\_access\_token(token1)

except HTTPException:

pass

else:

self.fail("Incorrect token (sub: None) has passed")

try:

await token\_provider.process\_access\_token(token2)

except HTTPException:

pass

else:

self.fail("Incorrect token (id: None) has passed")

try:

await token\_provider.process\_access\_token(token3)

except HTTPException:

pass

else:

self.fail("Incorrect token (sub: None and id: None) has passed")

async def test\_json\_web\_token\_incorrect\_username(self):

# values aren't important in this case - it should raise in any case

database\_correct\_user = await database\_manager.get\_user\_by\_username(TEST\_USERNAME)

if database\_correct\_user is None:

self.fail(f"User {TEST\_USERNAME} is absent from Database")

bad\_user = {

"sub": TEST\_USERNAME + "wrong",

"id": database\_correct\_user.id

}

token = token\_provider.generate\_access\_token(bad\_user)

try:

await token\_provider.process\_access\_token(token)

except HTTPException:

pass

else:

self.fail("Incorrect token (username doesn't align with id) has passed")

async def test\_security\_user\_authentification\_point(self):

username = TEST\_USERNAME

password = TEST\_PASSWORD

await security.authenticate\_user(username, password)

fake\_password = "wrong" + password

with self.assertRaises(HTTPException):

await security.authenticate\_user(username, fake\_password)

fake\_username = "wrong" + username

with self.assertRaises(HTTPException):

await security.authenticate\_user(fake\_username, password)

async def test\_database\_get\_user(self):

username = TEST\_USERNAME

password = TEST\_PASSWORD

self.assertTrue(await database\_manager.check\_username\_exists(username), "Test user should exist")

self.assertFalse(await database\_manager.check\_username\_exists("wrong"), "Test user should not exist")

user: UserInDB | None = await database\_manager.get\_user\_by\_username(username)

if user is None:

self.fail("Test user should be found by username")

self.assertTrue(hasher.verify\_password(password, user.hashed\_password), "Test user should have the correct password")

user\_by\_id: UserInDB | None = await database\_manager.get\_user\_by\_id(user.id)

if user\_by\_id is None:

self.fail("Test user should be found by id")

self.assertEqual(user.id, user\_by\_id.id, "Test user should have the same id")

self.assertEqual(user.username, user\_by\_id.username, "Test user should have the same username")

self.assertEqual(user.hashed\_password, user\_by\_id.hashed\_password, "Test user should have the same hashed password")

async def test\_database\_add\_delete\_user(self):

username = f"testUser{random.randint(0, 10000)}"

password = TEST\_PASSWORD

self.assertTrue(await database\_manager.add\_user(username, password), "Test user should be added")

self.assertFalse(await database\_manager.add\_user(username, password), "Test user should not be added twice")

self.assertTrue(await database\_manager.check\_username\_exists(username), "Test user should exist")

user: UserInDB | None = await database\_manager.get\_user\_by\_username(username)

if user is None:

self.fail("Test user should be found by username")

await database\_manager.delete\_user(user.id)

self.assertFalse(await database\_manager.check\_username\_exists(username), "Test user should not exist")

with self.assertRaises(HTTPException):

await database\_manager.delete\_user(user.id)

self.assertFalse(await database\_manager.check\_username\_exists(username), "Test user should not exist (test of double delete)")

class TestHydroponic(unittest.IsolatedAsyncioTestCase):

@classmethod

def setUpClass(cls):

cls.\_\_username = f"testUser{random.randint(0, 10000)}"

cls.\_\_password = TEST\_PASSWORD

is\_added: bool = asyncio.run(database\_manager.add\_user(cls.\_\_username, cls.\_\_password))

assert is\_added, "Test user should be added"

cls.\_\_user: UserInDB | None = asyncio.run(database\_manager.get\_user\_by\_username(cls.\_\_username))

assert cls.\_\_user is not None, "Test user should be found by username"

@classmethod

def tearDownClass(cls):

assert cls.\_\_user is not None, "Test user should be found by username"

try:

asyncio.run(database\_manager.delete\_user(cls.\_\_user.id))

except HTTPException:

assert False, "Test user was errornous on delete"

async def test\_get\_add\_delete\_hydroponics(self):

if self.\_\_user is None:

self.fail("Test user should be found by username")

await database\_manager.add\_hydroponic(TEST\_HYDROPONIC, self.\_\_user.id)

list\_of\_hydroponics: list[Hydroponic] = await database\_manager.get\_all\_hydroponics(self.\_\_user.id)

self.assertGreater(len(list\_of\_hydroponics), 0, "Test user should have at least one hydroponic - added one")

max\_id = max(tuple(hydroponic.id for hydroponic in list\_of\_hydroponics if hydroponic.id is not None), default=-1)

if max\_id == -1:

self.fail("Test user should have at least one hydroponic - added one (get hydroponic id error)")

try:

hydroponic: Hydroponic | None = await database\_manager.get\_hydroponic\_by\_id(max\_id, self.\_\_user.id)

except HTTPException:

self.fail("Test user should have at least one hydroponic - added one (get hydroponic by id (1))")

if hydroponic is None:

self.fail("Test user should have at least one hydroponic - added one (get hydroponic by id (2))")

self.assertEqual(hydroponic.name, TEST\_HYDROPONIC.name, "Test hydroponic should have the same name as send one")

self.assertEqual(hydroponic.water\_amount, TEST\_HYDROPONIC.water\_amount, "Test hydroponic should have the same water amount as send one")

self.assertEqual(hydroponic.water\_consumption, TEST\_HYDROPONIC.water\_consumption, "Test hydroponic should have the same water consumption as send one")

self.assertEqual(hydroponic.minerals\_amount, TEST\_HYDROPONIC.minerals\_amount, "Test hydroponic should have the same minerals amount as send one")

self.assertEqual(hydroponic.minerals\_optimal, TEST\_HYDROPONIC.minerals\_optimal, "Test hydroponic should have the same minerals optimal as send one")

self.assertEqual(hydroponic.minerals\_consumption, TEST\_HYDROPONIC.minerals\_consumption, "Test hydroponic should have the same minerals consumption as send one")

self.assertEqual(hydroponic.acidity\_optimal\_ph, TEST\_HYDROPONIC.acidity\_optimal\_ph, "Test hydroponic should have the same acidity optimal ph as send one")

self.assertEqual(hydroponic.temperature\_C\_optimal, TEST\_HYDROPONIC.temperature\_C\_optimal, "Test hydroponic should have the same temperature optimal as send one")

self.assertEqual(hydroponic.oxygen\_amount, TEST\_HYDROPONIC.oxygen\_amount, "Test hydroponic should have the same oxygen amount as send one")

self.assertEqual(hydroponic.oxygen\_consumption, TEST\_HYDROPONIC.oxygen\_consumption, "Test hydroponic should have the same oxygen consumption as send one")

# ---

try:

await database\_manager.delete\_hydroponic(max\_id, self.\_\_user.id)

except HTTPException:

self.fail("Error on deletion of hydroponic")

self.assertIsNone(await database\_manager.get\_hydroponic\_by\_id(max\_id, self.\_\_user.id), "Test should not have hydroponic with id after it's deletion")

class TestChangeHydroponic(unittest.IsolatedAsyncioTestCase):

@classmethod

def setUpClass(cls):

cls.\_\_username = f"testUser{random.randint(0, 10000)}"

cls.\_\_password = TEST\_PASSWORD

is\_added: bool = asyncio.run(database\_manager.add\_user(cls.\_\_username, cls.\_\_password))

assert is\_added, "Test user should be added"

cls.\_\_user: UserInDB | None = asyncio.run(database\_manager.get\_user\_by\_username(cls.\_\_username))

assert cls.\_\_user is not None, "Test user should be found by username"

asyncio.run(database\_manager.add\_hydroponic(TEST\_HYDROPONIC, cls.\_\_user.id))

list\_of\_hydroponics = asyncio.run(database\_manager.get\_all\_hydroponics(cls.\_\_user.id))

assert len(list\_of\_hydroponics) > 0, "Test user should have at least one hydroponic - added one"

max\_id = max(tuple(hydroponic.id for hydroponic in list\_of\_hydroponics if hydroponic.id is not None), default=-1)

assert max\_id != -1, "Test user should have at least one hydroponic - added one (get hydroponic id error)"

cls.\_\_hydroponic\_id = max\_id

@classmethod

def tearDownClass(cls):

assert cls.\_\_user is not None, "Test user should be found by username"

try:

asyncio.run(database\_manager.delete\_user(cls.\_\_user.id))

except HTTPException:

assert False, "Test user was errornous on delete"

def setUp(self) -> None:

if self.\_\_user is None:

self.fail("Test user should be found by username")

self.user\_id: int = self.\_\_user.id

if self.\_\_hydroponic\_id is None:

self.fail("Test hydroponic should be found by id")

hydroponic\_temp = asyncio.run(database\_manager.get\_hydroponic\_by\_id(self.\_\_hydroponic\_id, self.user\_id))

if hydroponic\_temp is None:

self.fail("Test hydroponic should be found by id")

self.hydroponic: Hydroponic = hydroponic\_temp

async def test\_change\_hydroponic\_add\_water(self):

if self.hydroponic.id is None:

self.fail("Test hydroponic should be found by id")

await database\_manager.add\_10\_percent\_water(self.hydroponic.id, self.user\_id)

new\_hydroponic = await database\_manager.get\_hydroponic\_by\_id(self.hydroponic.id, self.user\_id)

if new\_hydroponic is None:

self.fail("Test hydroponic should be found by id")

add\_delta = min(TEST\_HYDROPONIC.water\_amount - self.hydroponic.value\_water, TEST\_HYDROPONIC.water\_amount \* 0.1)

if abs(new\_hydroponic.value\_water - self.hydroponic.value\_water - add\_delta) > 0.001:

self.fail("Test hydroponic should have the same water amount as send one")

async def test\_change\_hydroponic\_add\_minerals(self):

if self.hydroponic.id is None:

self.fail("Test hydroponic should be found by id")

await database\_manager.add\_5\_percent\_minerals(self.hydroponic.id, self.user\_id)

new\_hydroponic = await database\_manager.get\_hydroponic\_by\_id(self.hydroponic.id, self.user\_id)

if new\_hydroponic is None:

self.fail("Test hydroponic should be found by id")

add\_delta = min(TEST\_HYDROPONIC.minerals\_amount - self.hydroponic.minerals\_amount, TEST\_HYDROPONIC.minerals\_amount \* 0.05)

if abs(new\_hydroponic.minerals\_amount - self.hydroponic.minerals\_amount - add\_delta) > 0.001:

self.fail("Test hydroponic should have the same minerals amount as send one")

async def test\_change\_hydroponic\_add\_temperature(self):

if self.hydroponic.id is None:

self.fail("Test hydroponic should be found by id")

await database\_manager.add\_1\_celsius\_temperature(self.hydroponic.id, self.user\_id)

new\_hydroponic = await database\_manager.get\_hydroponic\_by\_id(self.hydroponic.id, self.user\_id)

if new\_hydroponic is None:

self.fail("Test hydroponic should be found by id")

add\_delta = min(database\_manager.HYDROPONIC\_MAX\_TEMPERATURE - self.hydroponic.value\_temperature\_C, 1)

if abs(new\_hydroponic.value\_temperature\_C - add\_delta - self.hydroponic.value\_temperature\_C) > 0.001:

self.fail("Test hydroponic should have the same temperature current as send one")

async def test\_change\_hydroponic\_lower\_temperature(self):

if self.hydroponic.id is None:

self.fail("Test hydroponic should be found by id")

await database\_manager.lower\_1\_celsius\_temperature(self.hydroponic.id, self.user\_id)

new\_hydroponic = await database\_manager.get\_hydroponic\_by\_id(self.hydroponic.id, self.user\_id)

if new\_hydroponic is None:

self.fail("Test hydroponic should be found by id")

lower\_delta = min(self.hydroponic.value\_temperature\_C - database\_manager.HYDROPONIC\_MIN\_TEMPERATURE, 1)

if abs(self.hydroponic.value\_temperature\_C - lower\_delta - new\_hydroponic.value\_temperature\_C) > 0.001:

self.fail("Test hydroponic should have the same temperature current as send one")

async def test\_change\_hydroponic\_add\_acid(self):

if self.hydroponic.id is None:

self.fail("Test hydroponic should be found by id")

await database\_manager.add\_0\_25\_acid(self.hydroponic.id, self.user\_id)

new\_hydroponic = await database\_manager.get\_hydroponic\_by\_id(self.hydroponic.id, self.user\_id)

if new\_hydroponic is None:

self.fail("Test hydroponic should be found by id")

add\_delta = min(self.hydroponic.value\_acidity\_ph - database\_manager.HYDROPONIC\_MIN\_PH, 0.25)

if abs(self.hydroponic.value\_acidity\_ph - add\_delta - new\_hydroponic.value\_acidity\_ph) > 0.001:

self.fail("Test hydroponic should have the same acid amount as send one")

async def test\_change\_hydroponic\_lower\_acid(self):

if self.hydroponic.id is None:

self.fail("Test hydroponic should be found by id")

await database\_manager.lower\_0\_25\_acid(self.hydroponic.id, self.user\_id)

new\_hydroponic = await database\_manager.get\_hydroponic\_by\_id(self.hydroponic.id, self.user\_id)

if new\_hydroponic is None:

self.fail("Test hydroponic should be found by id")

lower\_delta = min(database\_manager.HYDROPONIC\_MAX\_PH - self.hydroponic.value\_acidity\_ph, 0.25)

if abs(self.hydroponic.value\_acidity\_ph + lower\_delta - new\_hydroponic.value\_acidity\_ph) > 0.001:

self.fail("Test hydroponic should have the same acid amount as send one")

async def test\_change\_hydroponic\_add\_oxygen(self):

if self.hydroponic.id is None:

self.fail("Test hydroponic should be found by id")

await database\_manager.add\_5\_percent\_oxygen(self.hydroponic.id, self.user\_id)

new\_hydroponic = await database\_manager.get\_hydroponic\_by\_id(self.hydroponic.id, self.user\_id)

if new\_hydroponic is None:

self.fail("Test hydroponic should be found by id")

add\_delta = min(TEST\_HYDROPONIC.oxygen\_amount - self.hydroponic.oxygen\_amount, TEST\_HYDROPONIC.oxygen\_amount \* 0.05)

if abs(self.hydroponic.oxygen\_amount - add\_delta - new\_hydroponic.oxygen\_amount) > 0.001:

self.fail("Test hydroponic should have the same oxygen amount as send one")

async def test\_change\_hydroponic\_reload(self):

if self.hydroponic.id is None:

self.fail("Test hydroponic should be found by id")

is\_randomized = False

for \_ in range(10):

await database\_manager.reset(self.hydroponic.id, self.user\_id)

new\_hydroponic = await database\_manager.get\_hydroponic\_by\_id(self.hydroponic.id, self.user\_id)

if new\_hydroponic is None:

self.fail("Test hydroponic should be found by id")

if (

self.hydroponic.value\_temperature\_C != new\_hydroponic.value\_temperature\_C and

self.hydroponic.value\_acidity\_ph != new\_hydroponic.value\_acidity\_ph and

self.hydroponic.value\_minerals != new\_hydroponic.value\_minerals and

self.hydroponic.value\_oxygen != new\_hydroponic.value\_oxygen and

self.hydroponic.value\_water != new\_hydroponic.value\_water

):

is\_randomized = True

break

self.assertTrue(is\_randomized, "Test hydroponic should be randomized, yet wasn't after 10 tries")

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

# ДОДАТОК D

Код тестування бази даних MS SQL за допомогою фреймворка tSQLt

**НТУУ «КПІ» ІАТЕ ІПЗЕ**

Листів 13

Київ - 2025

use MusicForTest;

GO

EXEC tSQLt.NewTestClass 'testMusic';

GO

-- Check inccorect track\_count

CREATE or ALTER PROCEDURE testMusic.test\_assertEquals

AS

BEGIN

INSERT INTO music\_band(name) VALUES('Test Band');

EXEC tSQLt.ExpectException @ExpectedErrorNumber=547;

INSERT INTO music\_band(name, track\_count) VALUES('Invalid Band', -1);

END;

GO

-- Check correct insert into genre

CREATE PROCEDURE testMusic.test\_genre\_insert

AS

BEGIN

DECLARE @initial\_count INT;

SELECT @initial\_count = COUNT(\*) FROM band\_genre;

INSERT INTO band\_genre(name) VALUES ('Genre 1'), ('Genre 2');

DECLARE @new\_count INT;

SELECT @new\_count = COUNT(\*) FROM band\_genre;

IF @new\_count != @initial\_count + 2

EXEC tSQLt.Fail 'Expected 2 new rows in band\_genre, but count did not match';

END;

GO

-- Check correct input into music band

CREATE PROCEDURE TestMusic.test\_music\_band\_insert

AS

BEGIN

INSERT INTO band\_genre(name) VALUES('Genre for Test');

DECLARE @genre\_id INT = SCOPE\_IDENTITY();

INSERT INTO band\_genre(name) VALUES('Genre for Test2');

DECLARE @genre\_id2 INT = SCOPE\_IDENTITY();

INSERT INTO music\_band(name) VALUES('Band for Test');

DECLARE @band\_id INT = SCOPE\_IDENTITY();

DECLARE @initial\_count INT;

SELECT @initial\_count = COUNT(\*) FROM music\_band\_2\_genre;

-- Act

INSERT INTO music\_band\_2\_genre (band\_id, genre\_id) VALUES

(@band\_id, @genre\_id), (@band\_id, @genre\_id2);

-- Assert

DECLARE @new\_count INT;

SELECT @new\_count = COUNT(\*) FROM music\_band\_2\_genre;

SET @initial\_count = @initial\_count + 2;

EXEC tSQLt.assertEquals @new\_count, @initial\_count, 'Expected 2 new rows in music\_band\_2\_genre, but count did not match'

END;

GO

-- Check correct input into track

CREATE PROCEDURE TestMusic.test\_track\_insert

AS

BEGIN

INSERT INTO music\_band(name) VALUES('Band for Track Test');

DECLARE @band\_id INT = SCOPE\_IDENTITY();

DECLARE @initial\_count INT;

SELECT @initial\_count = COUNT(\*) FROM track;

INSERT INTO track(title, band\_id) VALUES

('Track 1', @band\_id),

('Track 2', @band\_id);

DECLARE @new\_count INT;

SELECT @new\_count = COUNT(\*) FROM track;

EXEC tSQLt.AssertNotEquals @new\_count, @initial\_count;

SET @initial\_count = @initial\_count + 2;

EXEC tSQLt.AssertEquals @new\_count, @initial\_count, 'Expected 2 new rows in track, but count did not match.';

END;

GO

-- Check correct amount of track by insert and update by triger

CREATE PROCEDURE TestMusic.test\_music\_band\_track\_count\_on\_insert

AS

BEGIN

INSERT INTO music\_band (name) VALUES ('Band for Track Insert Test');

DECLARE @band\_id INT = SCOPE\_IDENTITY();

DECLARE @initial\_track\_count INT;

SELECT @initial\_track\_count = track\_count FROM music\_band WHERE band\_id = @band\_id;

INSERT INTO track (title, band\_id) VALUES

('Track 1', @band\_id),

('Track 2', @band\_id);

DECLARE @expected\_track\_count INT = @initial\_track\_count + 2;

DECLARE @actual\_track\_count INT;

SELECT @actual\_track\_count = track\_count FROM music\_band WHERE band\_id = @band\_id;

EXEC tSQLt.AssertEquals @actual\_track\_count, @expected\_track\_count, 'Expected track\_count to be + 2 after inserting 2 tracks';

END;

GO

-- Check correct amount of track by delete and update by triger

CREATE PROCEDURE TestMusic.test\_music\_band\_track\_count\_on\_delete

AS

BEGIN

INSERT INTO music\_band (name) VALUES ('Band for Track Delete Test');

DECLARE @band\_id INT = SCOPE\_IDENTITY();

INSERT INTO track (title, band\_id) VALUES

('Track 1', @band\_id),

('Track 2', @band\_id);

DECLARE @initial\_track\_count INT;

SELECT @initial\_track\_count = track\_count FROM music\_band WHERE band\_id = @band\_id;

DELETE FROM track WHERE band\_id = @band\_id AND title = 'Track 1';

DECLARE @expected\_track\_count INT = @initial\_track\_count - 1;

DECLARE @actual\_track\_count INT;

SELECT @actual\_track\_count = track\_count FROM music\_band WHERE band\_id = @band\_id;

IF @actual\_track\_count != @expected\_track\_count

EXEC tSQLt.AssertEquals @actual\_track\_count, @expected\_track\_count, 'Expected track\_count to be - 1 after deleting 1 track';

END;

GO

-- Test total\_track in band\_genre on Insert

CREATE PROCEDURE TestMusic.test\_band\_genre\_total\_track\_on\_insert

AS

BEGIN

INSERT INTO band\_genre (name) VALUES ('Genre for Insert Test');

DECLARE @genre\_id INT = SCOPE\_IDENTITY();

INSERT INTO music\_band (name) VALUES ('Band for Genre Insert Test');

DECLARE @band\_id INT = SCOPE\_IDENTITY();

INSERT INTO music\_band\_2\_genre (band\_id, genre\_id) VALUES (@band\_id, @genre\_id);

DECLARE @initial\_total\_track INT;

SELECT @initial\_total\_track = total\_track FROM band\_genre WHERE genre\_id = @genre\_id;

INSERT INTO track (title, band\_id) VALUES

('Track 1', @band\_id),

('Track 2', @band\_id);

DECLARE @expected\_total\_track INT = @initial\_total\_track + 2;

DECLARE @actual\_total\_track INT;

SELECT @actual\_total\_track = total\_track FROM band\_genre WHERE genre\_id = @genre\_id;

EXEC tSQLt.AssertEquals @actual\_total\_track, @expected\_total\_track, 'Expected total\_track to be + 2 after insert';

END;

GO

-- Test local track in band\_genre on Delete

CREATE PROCEDURE TestMusic.test\_band\_genre\_total\_track\_on\_delete

AS

BEGIN

INSERT INTO band\_genre (name) VALUES ('Genre for Delete Test');

DECLARE @genre\_id INT = SCOPE\_IDENTITY();

INSERT INTO music\_band (name) VALUES ('Band for Genre Delete Test');

DECLARE @band\_id INT = SCOPE\_IDENTITY();

INSERT INTO music\_band\_2\_genre (band\_id, genre\_id) VALUES (@band\_id, @genre\_id);

INSERT INTO track (title, band\_id) VALUES

('Track 1', @band\_id),

('Track 2', @band\_id);

DECLARE @initial\_total\_track INT;

SELECT @initial\_total\_track = total\_track FROM band\_genre WHERE genre\_id = @genre\_id;

DELETE FROM track WHERE band\_id = @band\_id AND title = 'Track 1';

DECLARE @expected\_total\_track INT = @initial\_total\_track - 1;

DECLARE @actual\_total\_track INT;

SELECT @actual\_total\_track = total\_track FROM band\_genre WHERE genre\_id = @genre\_id;

EXEC tSQLt.AssertEquals @actual\_total\_track, @expected\_total\_track, 'Expected total\_track to be - 1';

END;

GO

--

CREATE PROCEDURE TestMusic.test\_GetSongsByGenre

AS

BEGIN

INSERT INTO band\_genre (name) VALUES ('Rock');

DECLARE @genre\_id INT = SCOPE\_IDENTITY();

INSERT INTO music\_band (name) VALUES ('Band 3');

DECLARE @band\_id INT = SCOPE\_IDENTITY();

INSERT INTO music\_band\_2\_genre (band\_id, genre\_id) VALUES (@band\_id, @genre\_id);

INSERT INTO track (title, band\_id) VALUES ('Song 1', @band\_id);

DECLARE @result TABLE (TrackTitle NVARCHAR(255), BandName NVARCHAR(255));

INSERT INTO @result

EXEC GetSongsByGenre @genre\_id;

DECLARE @count INT;

SELECT @count = COUNT(\*) FROM @result;

EXEC tSQLt.AssertEquals @count, 1, 'GetSongsByGenre did not return the expected number of tracks.';

END;

GO

-- Test band with multiple genres

CREATE PROCEDURE TestMusic.test\_band\_with\_multiple\_genres

AS

BEGIN

INSERT INTO band\_genre (name) VALUES ('Test\_genre\_Rock'), ('Test\_genre\_Pop');

DECLARE @genre\_id1 INT = (SELECT genre\_id FROM band\_genre WHERE name = 'Test\_genre\_Rock');

DECLARE @genre\_id2 INT = (SELECT genre\_id FROM band\_genre WHERE name = 'Test\_genre\_Pop');

INSERT INTO music\_band (name) VALUES ('Multi-Genre Band');

DECLARE @band\_id INT = SCOPE\_IDENTITY();

INSERT INTO music\_band\_2\_genre (band\_id, genre\_id) VALUES (@band\_id, @genre\_id1), (@band\_id, @genre\_id2);

DECLARE @initial\_total\_track\_genre1 INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id1);

DECLARE @initial\_total\_track\_genre2 INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id2);

INSERT INTO track (title, band\_id) VALUES ('Track 1', @band\_id), ('Track 2', @band\_id);

-- check

DECLARE @expected\_total\_track\_genre1 INT = @initial\_total\_track\_genre1 + 2;

DECLARE @expected\_total\_track\_genre2 INT = @initial\_total\_track\_genre2 + 2;

DECLARE @result\_total\_track\_genre1 INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id1);

DECLARE @result\_total\_track\_genre2 INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id2);

EXEC tSQLt.AssertEquals @result\_total\_track\_genre1, @expected\_total\_track\_genre1, 'Genre 1 total\_track mismatch';

EXEC tSQLt.AssertEquals @result\_total\_track\_genre2, @expected\_total\_track\_genre2, 'Genre 2 total\_track mismatch';

END;

GO

-- Test genre counters on band\_deletion

CREATE PROCEDURE TestMusic.test\_genre\_counters\_on\_band\_deletion

AS

BEGIN

INSERT INTO band\_genre(name) VALUES ('Genre A'), ('Genre B');

DECLARE @genre\_id1 INT = (SELECT genre\_id FROM band\_genre WHERE name = 'Genre A');

DECLARE @genre\_id2 INT = (SELECT genre\_id FROM band\_genre WHERE name = 'Genre B');

INSERT INTO music\_band(name) VALUES ('Band for Deletion');

DECLARE @band\_id INT = SCOPE\_IDENTITY();

INSERT INTO music\_band\_2\_genre (band\_id, genre\_id) VALUES (@band\_id, @genre\_id1), (@band\_id, @genre\_id2);

INSERT INTO track (title, band\_id) VALUES ('Track 1', @band\_id), ('Track 2', @band\_id);

DECLARE @initial\_total\_track\_genre1 INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id1);

DECLARE @initial\_total\_track\_genre2 INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id2);

DELETE FROM music\_band WHERE band\_id = @band\_id;

-- check

DECLARE @expected\_total\_track\_genre1 INT = @initial\_total\_track\_genre1 - 2;

DECLARE @expected\_total\_track\_genre2 INT = @initial\_total\_track\_genre2 - 2;

DECLARE @result\_total\_track\_genre1 INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id1);

DECLARE @result\_total\_track\_genre2 INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id2);

EXEC tSQLt.AssertEquals @result\_total\_track\_genre1, @expected\_total\_track\_genre1, 'Genre 1 total\_track mismatch';

EXEC tSQLt.AssertEquals @result\_total\_track\_genre2, @expected\_total\_track\_genre2, 'Genre 2 total\_track mismatch';

END;

GO

-- Test Duplicate Entries in band genre

CREATE PROCEDURE TestMusic.test\_duplicate\_entries\_in\_band\_genre

AS

BEGIN

INSERT INTO band\_genre (name) VALUES ('Genre Duplicate Check');

DECLARE @genre\_id INT = (SELECT genre\_id FROM band\_genre WHERE name = 'Genre Duplicate Check');

INSERT INTO music\_band (name) VALUES ('Band Duplicate Check');

DECLARE @band\_id INT = SCOPE\_IDENTITY();

INSERT INTO music\_band\_2\_genre (band\_id, genre\_id) VALUES (@band\_id, @genre\_id);

EXEC tSQLt.ExpectException @ExpectedErrorNumber=2627;

INSERT INTO music\_band\_2\_genre (band\_id, genre\_id) VALUES (@band\_id, @genre\_id);

END;

GO

-- test band tracks with multiple genres

CREATE PROCEDURE TestMusic.test\_band\_tracks\_with\_multiple\_genres

AS

BEGIN

INSERT INTO band\_genre (name) VALUES ('Genre 1'), ('Genre 2');

DECLARE @genre\_id1 INT = (SELECT genre\_id FROM band\_genre WHERE name = 'Genre 1');

DECLARE @genre\_id2 INT = (SELECT genre\_id FROM band\_genre WHERE name = 'Genre 2');

INSERT INTO music\_band (name) VALUES ('Band Multi-Genre Tracks');

DECLARE @band\_id INT = SCOPE\_IDENTITY();

INSERT INTO music\_band\_2\_genre (band\_id, genre\_id) VALUES (@band\_id, @genre\_id1), (@band\_id, @genre\_id2);

DECLARE @initial\_track\_count INT = (SELECT track\_count FROM music\_band WHERE band\_id = @band\_id);

DECLARE @initial\_total\_track\_genre1 INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id1);

DECLARE @initial\_total\_track\_genre2 INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id2);

INSERT INTO track (title, band\_id) VALUES ('Track 1', @band\_id), ('Track 2', @band\_id);

DECLARE @expected\_track\_count INT = @initial\_track\_count + 2;

DECLARE @track\_count\_result INT = (SELECT track\_count FROM music\_band WHERE band\_id = @band\_id);

EXEC tSQLt.AssertEquals @expected\_track\_count, @track\_count\_result, 'Band track\_count not updated correctly';

DECLARE @expected\_total\_track\_genre1 INT = @initial\_total\_track\_genre1 + 2;

DECLARE @expected\_total\_track\_genre2 INT = @initial\_total\_track\_genre2 + 2;

DECLARE @total\_track\_genre1\_result INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id1);

DECLARE @total\_track\_genre2\_result INT = (SELECT total\_track FROM band\_genre WHERE genre\_id = @genre\_id2);

EXEC tSQLt.AssertEquals @expected\_total\_track\_genre1, @total\_track\_genre1\_result, 'Genre 1 total\_track not updated correctly';

EXEC tSQLt.AssertEquals @expected\_total\_track\_genre1, @total\_track\_genre1\_result, 'Genre 2 total\_track not updated correctly';

END;

GO

EXEC tSQLt.Run 'testMusic'