

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

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	

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 проекту “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

Напрямок підготовки 121 Інженерія програмного забезпечення

ЗАВДАННЯ

НА КУРСОВУ РОБОТУ СТУДЕНТА

Михайловичу Олександрю 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 проєкту 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 задачі	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 IoT 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, SQLAlchemy, tSQLt, WSGI/ASGI/WSGI 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, SQLAlchemy, tSQLt, server WSGI/ASGI/WSGI 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
РОЗДІЛ 1. ОПИС ПРОЄКТУ	12
1.1 Функціонал додатку	12
1.2 Інтерфейс додатку	14
1.3 Використані програмні засоби.....	28
1.4 Вимоги до проєкту	30
РОЗДІЛ 2. ТЕСТ-ПЛАН ПРОЄКТУ	34
2.1 Ідентифікатор плану тестування.....	35
2.2 Цілі тестування	35
2.3 Сфера застосування	36
2.4 Елементи тестування	36
2.5 Функціонал, що тестується	37
2.5.1 Тестування хешування на сервері	37
2.5.2 Тестування коректності JSON Web Token + обгортки драйверу на сервері	37
2.5.3 Тестування коректності обгортки драйверу + хешування на сервері ..	37
2.5.4 Тестування класу обгортки драйверу на додавання й видалення користувача на сервері	38
2.5.5 Тестування класу обгортки драйверу на додавання, видалення, отримання гідропоніки на сервері	38
2.5.6 Тестування класу обгортки драйвера на додавання води до гідропоніки на сервері.....	38
2.5.7 Тестування класу обгортки драйвера на додавання мінералів до гідропоніки на сервері	38

2.5.8 Тестування класу обгортки драйвера на додавання кисню до гідропоніки на сервері.....	39
2.5.9 Тестування класу обгортки драйвера на збільшення температури в гідропоніці на сервері.....	39
2.5.10 Тестування класу обгортки драйвера на зменшення температури в гідропоніці на сервері.....	40
2.5.11 Тестування класу обгортки драйвера на збільшення кислотності в гідропоніці на сервері.....	40
2.5.12 Тестування класу обгортки драйвера на зменшення кислотності в гідропоніці на сервері.....	41
2.5.13 Тестування класу обгортки драйвера на перевантаження гідропоніки на сервері.....	42
2.5.14 Набір тестів для тестування API, й запитів до бекенду із заміною/підміною елементів (використання тестових двійників).....	42
2.5.15 Тестування веб-клієнта, та загальної системи - залогінення.....	43
2.5.16 Тестування веб-клієнта, та загальної системи - реєстрація	43
2.5.17 Тестування веб-клієнта, та загальної системи - переадресація за відсутності токена.....	43
2.5.18 Тестування веб-клієнта, та загальної системи - вихід з акаунту	44
2.5.19 Тестування веб-клієнта, та загальної системи - додавання та видалення	44
2.5.20 Тестування веб-клієнта, та загальної системи - керування гідропонікою, кнопка перезавантаження гідропоніки.....	44
2.5.21 Тестування веб-клієнта, та загальної системи - керування гідропонікою, додавання води	44
2.5.22 Тестування веб-клієнта, та загальної системи - керування гідропонікою, додавання мінералів	45

2.5.23 Тестування веб-клієнта, та загальної системи - керування гідропонікою, додавання кисню	45
2.5.24 Тестування веб-клієнта, та загальної системи - керування гідропонікою, керування температурою	45
2.5.25 Тестування веб-клієнта, та загальної системи - керування гідропонікою, керування кислотністю	46
2.6 Функціонал, що не тестується	46
2.7 Підхід	47
2.7.1 Інструменти тестування.....	47
2.7.2 Збір показників та метрики	48
2.7.3 Конфігурації що будуть тестуватися	49
2.7.4 Підхід до тестування додатку	49
2.8 Критерії проходження/провалу для плану тестування	50
2.9 Критерії призупинення та умови відновлення тестування.....	51
2.9.1 Критерії призупинення тестування.....	51
2.9.2 Критерії відновлення тестування	51
2.10 Результати тестування	51
2.11 Тестові завдання	52
2.11.1 Залежності між тестами.....	52
2.11.2 Рівень навичок	52
2.11.3 Ключові етапи.....	52
2.12 Потреби середовища	53
2.12.1 Джерела інформації	53
2.12.2 Безпека.....	53
2.12.3 Відповідальності.....	53
2.13 Ризики.....	53

2.13.1 Зміна строків виконання роботи	53
2.13.2 Неповний проєкт	53
2.14 Затвердження	54
РОЗДІЛ 3 РОЗРОБКА БАГ РЕПОРТУ	55
3.1 Список знайдених багів.....	56
3.2 Складений баг репорт - баг №1	56
3.2.1 Тема багу	56
3.2.2 Детальний опис багу.....	56
3.2.3 Кроки до відтворення	57
3.2.4 Результати	57
3.2.5 Інформація про баг	57
3.3 Розбір рішення багу.....	58
3.3.1 Розбір причини проблеми.....	58
3.3.2 Огляд можливих рішень	60
3.3.3 Результат виправлення	61
Розділ 4 ТЕСТУВАННЯ ПРОЄКТУ	62
4.1 Автоматизоване тестування бекенд серверу (unittest, Coverage.py, unittest.mock).....	62
4.1.1 Переваги та недоліки	62
4.1.2 Середовище написання.....	63
4.1.3 Написання тестів.....	64
4.1.4 Результат	67
4.2 Автоматизоване тестування фронтенд клієнту із використанням Selenium WebDriver.....	69
4.2.1 Переваги та недоліки	69
4.2.2 Середовище написання.....	70

4.2.3 Написання тестів.....	70
4.2.4 Результат	73
4.3 Інші засоби тестування.....	74
ВИСНОВОК	78
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	80
ДОДАТОК А	81
ДОДАТОК В.....	101
ДОДАТОК С	116
ДОДАТОК D	134

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 аерозолі. 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, рН, 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 система може бути встановлена на балконі або на внутрішньому дворіку, 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 обмежень 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 ноутбуки. 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 акаунта 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.

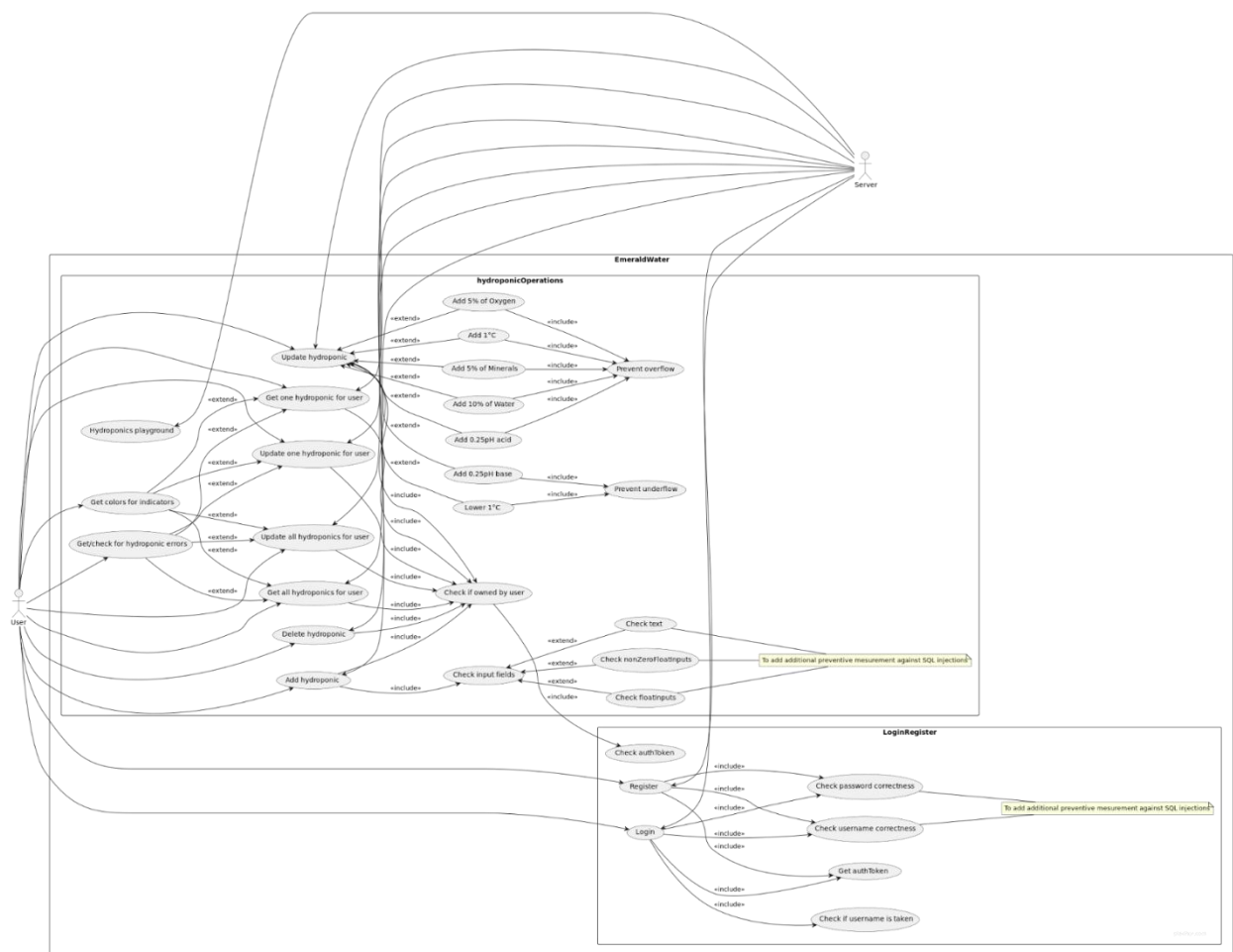


Рисунок 1.1 - use case діаграма написана на мові PlantUML

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 пароллю. Для нових користувачів додано позначку “квадратик, який розпадається”, для проходження реєстрації через його натискання.

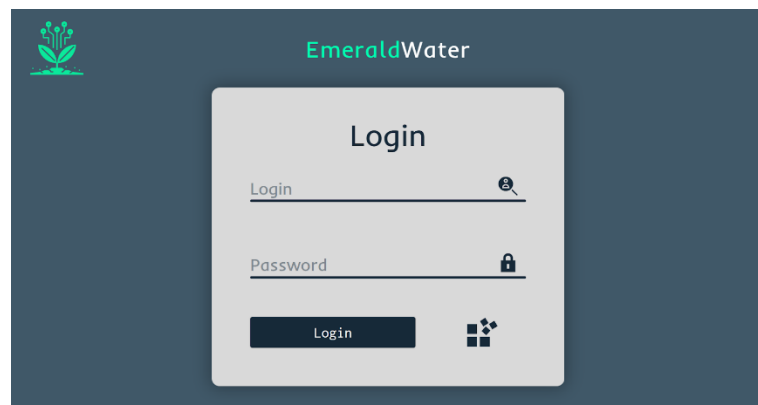


Рисунок 1.2 - основна сторінка Login (стан: порожній)

На рисунку 1.3 в блоці реєстрація (Sign Up) показано, відсутність у нового користувача права при реєстрації використати ім'я, яке вже закріплене за попередньо 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.

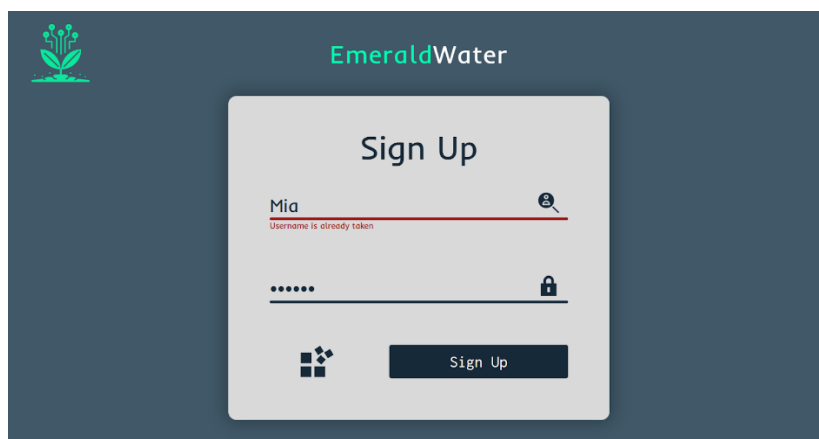


Рисунок 1.3. - основна сторінка Sign Up (стан: помилка при введенні ім'я для реєстрації, яке вже зайнято зареєстрованим користувачем)

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 натискання кнопки login здійснюється перехід на головну сторінку, з усією внесеною попередньо, гідропонікою зареєстрованого користувача.

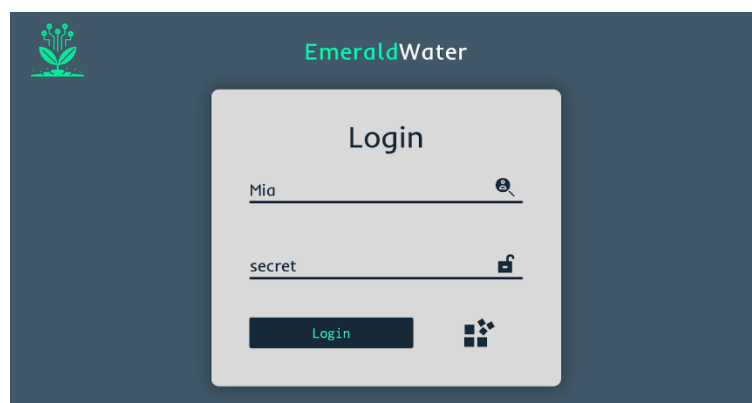


Рисунок 1.4 - основна сторінка Login (стан: перегляд попередньо зареєстрованого паролю та можливістю натискання кнопки login для переходу на головну сторінку)

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.

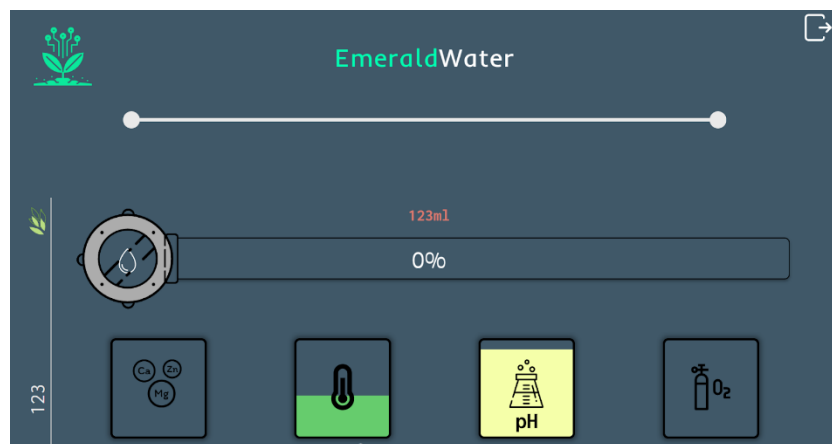


Рисунок 1.5 - основна сторінка, кнопка справа зверху надсилає до сторінки розлогінення, натискання на гідропоніку надсилає на сторінку керування нею

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.

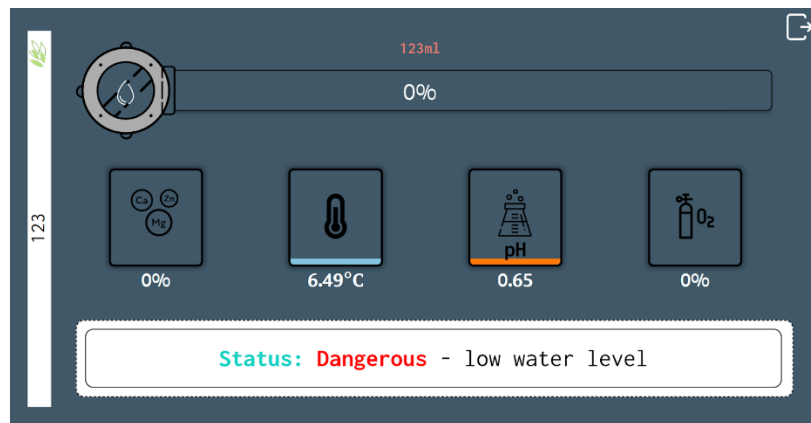


Рисунок 1.6 - критична гідропоніка - наведений компонент (біла заливка назви)
- небезпека в полі повідомлень

На рисунку 1.7 показаний стан гідропоніки у стані “ОК” 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.

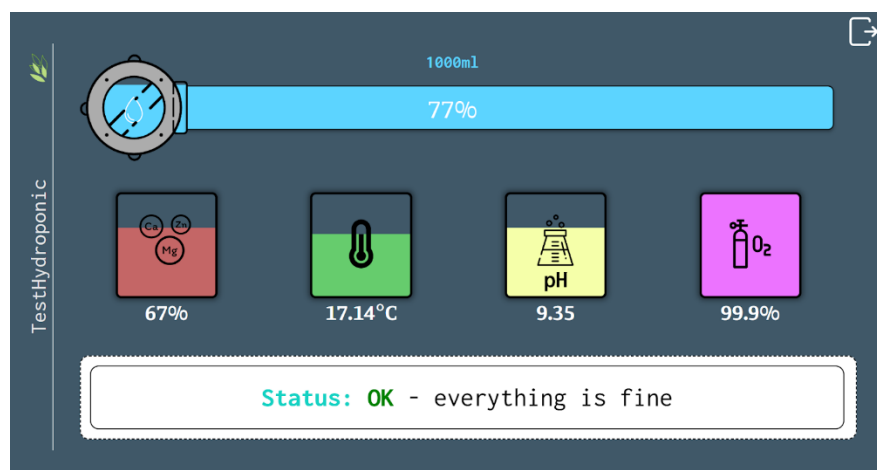


Рисунок 1.7 - гідропоніка в гарному стані

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, рН, 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.

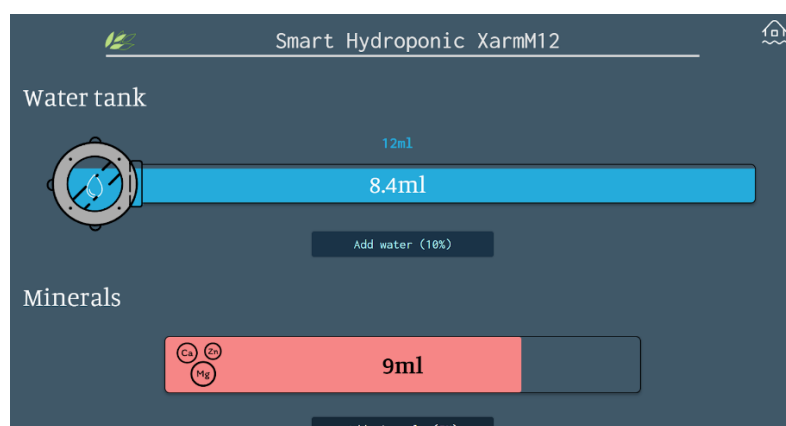


Рисунок 1.8 - керування гідропонікою перший екран, наведення на кнопку додавання води

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°C. 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 рН.

Рисунок 1.10 показує продовження сторінки керування компонентами (контейнерами) гідропоніки. Так компонент (контейнер) гідропоніки - кисень має одну клікабельну кнопку додати кисень “Add oxugen”. 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 наведенні/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.

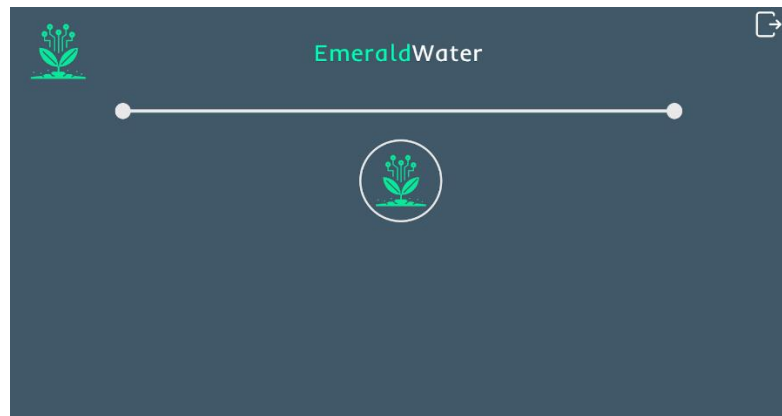


Рисунок 1.11 - порожня основна сторінка з кнопкою додавання нового елементу, яка має вигляд кола, в яке схематично вписано малюнок рослини

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.

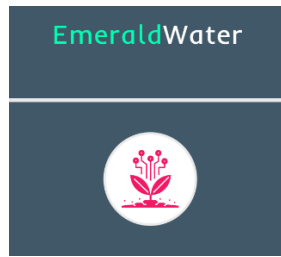


Рисунок 1.12 - зміна вигляду кнопки додавання гідропонік на наведення

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.

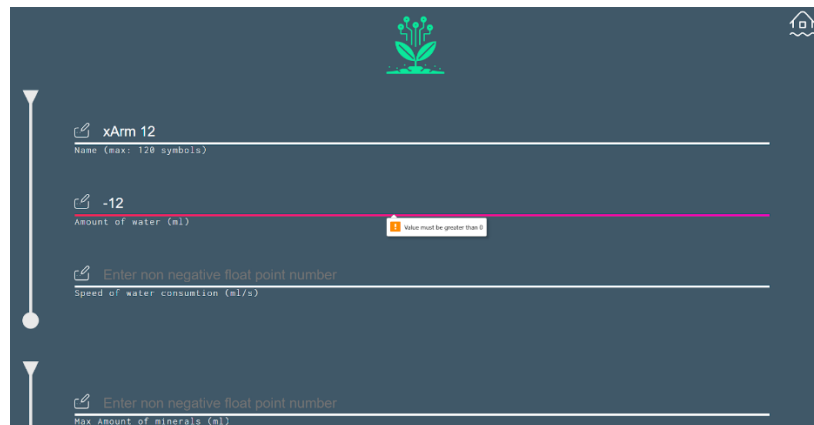


Рисунок 1.13 - помилка вводу- негативне значення

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.

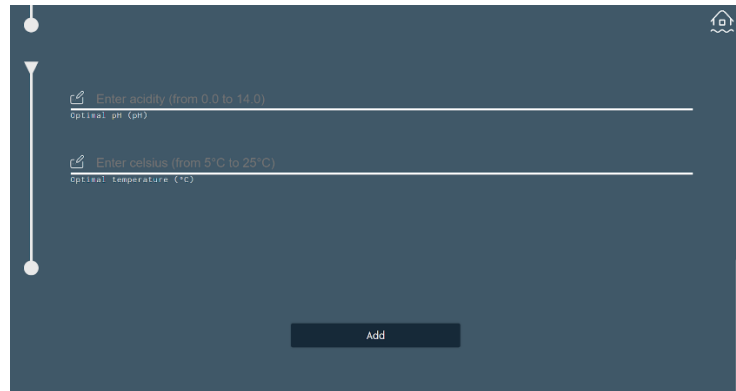
A screenshot of a web application interface. It features two input fields stacked vertically. The first field is labeled 'Enter acidity (from 0.0 to 14.0)' with a small icon of a clipboard and the text 'Optimal pH (pH)' below it. The second field is labeled 'Enter Celsius (from 5°C to 25°C)' with a similar icon and the text 'Optimal temperature (°C)' below it. At the bottom center, there is a dark rectangular button labeled 'Add'. The background is a dark blue-grey color.

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 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.

A screenshot of the same web application interface as in Figure 1.14, but with an error state. The 'Optimal pH (pH)' field is still empty. The 'Optimal temperature (°C)' field now contains the number '3' and has a red border. Below the input field, a white error message box with a red exclamation mark icon displays the text 'Value must be greater than 5'. The background remains dark blue-grey.

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 браузеру 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.

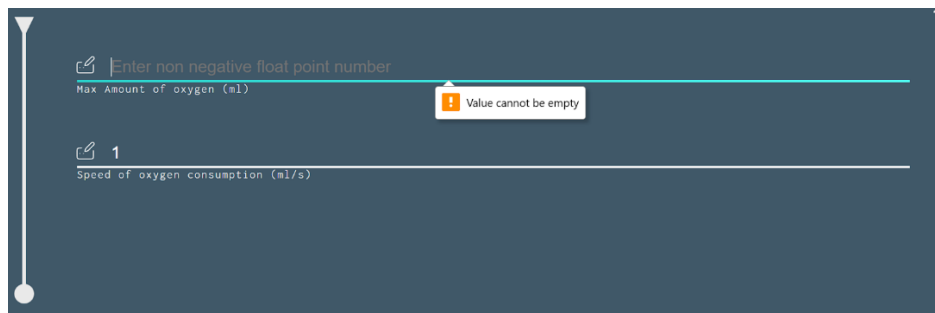


Рисунок 1.16 - випадіння вікна з попереджувальним написом та зміна кольору input у разі, пропуску користувачем заповнення поля вводу інформації для відповідних систем гідропоніки

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

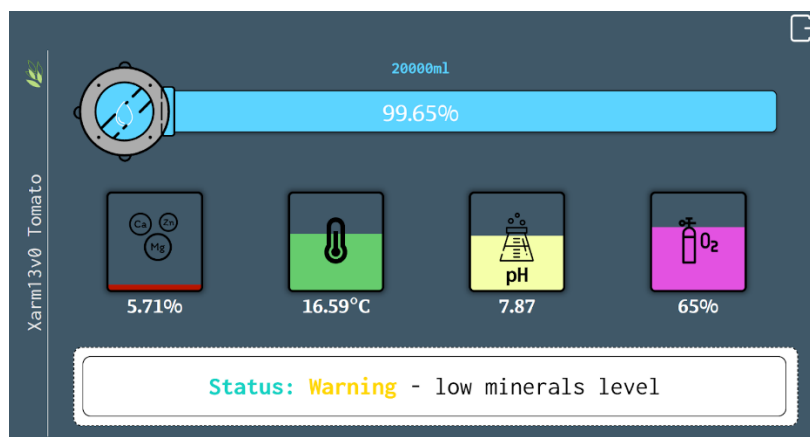


Рисунок 1.17 - в блоці-статус рівень небезпеки жовтий, оскільки зменшення суміші мінералів не так критично як, наприклад, кисень

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 - рН 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.

Oxygen	Temperature	
#9e336a	#66cc6d	
#c938c6	Less:	
#e352e1	#83c4de	
#ec73ff	#1c3eba	
	More:	
Acidness	#f1952b	
#f5ffa9	#dd1919	
Less:		Mineral
#91ff61	Water	#b71500
#55fe0c	#db7b6f	#be4141
More:	#70c6e5	#c46666
#fb9542	#25abdb	#f68686
#ff790b	#5cd4ff	

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.

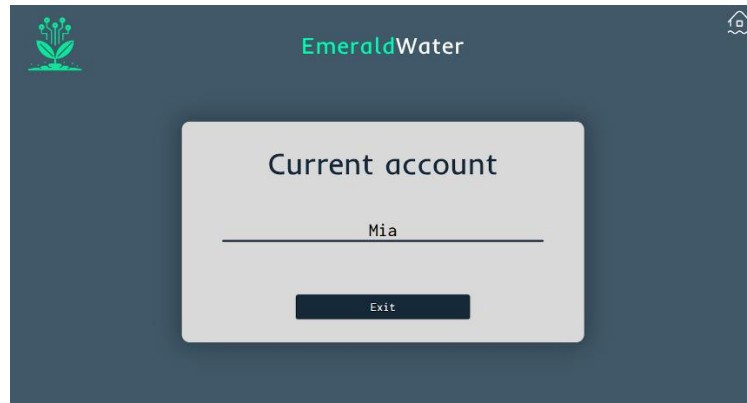


Рисунок 1.19 - сторінка виходу з акаунту

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

1.3 Використані програмні засоби

Проект 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/RSOI 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 став ефектом використання фреймворків, проте його використання є значним на серверній частині.

Для спілкування бекенд сервера з базою даних (Microsoft SQL server) було використано драйвер `pymsql`, проте його використання виконувалось через

фреймворк-обгортку SQLAlchemy (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 хешування 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 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 прт. 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:

а. Управління обліковими записами користувачів:

- i. Користувачі можуть реєструватися, входити та безпечно керувати своїми обліковими записами.
- ii. Паролі зберігаються в базі даних у вигляді хешованих і посолених значень.
- iii. Веб-токени JSON (JWT) використовуються для управління сесансами і втрачають чинність через певний час.

б. Управління гідропонними системами:

- i. 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).
- ii. Користувачі можуть видаляти гідропонні системи.
- iii. Додавання води, кисню та мінералів у резервуари.

- iv. Зміна температури та рівня кислотності.
- v. Внутрішній сервер емулює стан і відповідно оновлює базу даних.
- c. Перевірка даних:
 - i. На стороні клієнта: Забезпечує належне форматування вхідних даних та блокування надсилання некоректних значень.
 - ii. 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).
- d. Оновлення в режимі реального часу:
 - i. Веб-клієнт періодично запитує оновлення з сервера кожні 5 секунд.
 - ii. Емуляція фізичних дій на сервері відбувається кожні 60 секунд.
 - iii. 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).
- e. Обробка помилок та критичних ситуацій:
 - i. Після закінчення терміну дії JSON Web Token-у користувачі перенаправляються на сторінку входу в систему.
 - ii. Неправильні вхідні дані повертають описові повідомлення про помилки.
- f. Візуальні індикатори та повідомлення про помилки:
 - i. 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.

ii. Окремі гідропоніки на сторінці всіх гідропонік консолідують всі показники гідропонної системи і надають короткий огляд стану окремої системи.

iii. 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

a. Продуктивність:

i. Система ефективно обробляє запити API та оновлення бази даних, забезпечуючи швидке реагування в реальному часі.

ii. Веб-сторінки оновлюють дані безперебійно, не вимагаючи повного перезавантаження.

b. Безпека:

i. Паролі хешуються та “соляться” перед зберіганням. Відсутнє збереження “сирих” паролів на базі даних.

ii. Автентифікація на основі JWT забезпечує безпечний доступ до облікових записів і даних користувачів.

iii. Перевірка даних блокує введення зловмисних даних або такі, що призведуть до негативних наслідків для продукту.

c. Масштабованість:

i. Архітектура підтримує декілька клієнтів (наприклад, браузерів, мобільних пристроїв), що взаємодіють з одними і тими ж даними.

d. Надійність:

i. Система націлена на узгодженість даних під час взаємодії між клієнтом і сервером.

ii. Періодичні оновлення даних для оновлення станів гідропонних систем.

е. Зручність використання:

- і. Зрозумілі помилки про коректність вхідних даних на веб-інтерфейсі та на серверному API.
- ii. Автоматичне перенаправлення на сторінку входу після закінчення терміну дії токена.

ф. Придатність до змін:

- і. Чіткий розподіл обов'язків між фронтендом, бекендом та базою даних.
- ii. 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 часу.
3. Протестувати візуальні індикатори та пріоритетизацію 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:

- a. Реєстрація, логін, робота з JSON Web Tokens, переадресація на сторінку логіну за закінчення терміну валідності JSON Web Token-у.
- b. Зміна кольорів індикаторів, зміна статусу гідропонік, пріоритетизація помилок.
- c. Коректне оновлення даних внаслідок роботи з декількох клієнтів (наприклад дві різні сторінки в браузері), наявність оновлення даних, без перезавантаження сторінки.
- d. Формування запитів до API та відображення результатів.

2. Бекенд:

- a. Всі endpoint-и API.
- b. Логіка емуляції гідропонних систем.
- c. Правильна робота системи авторизації.
- d. Коректні виклики до бази даних відповідно до логіки API.
- e. Реалізація валідацій.

3. База даних:

- a. Загальна валідація даних.
- b. Валідація даних за допомогою тригер(-а/-ів).
- c. 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 функціональностей, 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 токена (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 акаунту, 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 (Mock), хешування було замінено/спрощено (Fake), замість JSON Web Token-а використана затишка (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 токєну).

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 токєну) - 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 токєна

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 токєн 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 $\geq 95\%$.

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 $\leq 50\%$. Далі кнопка для додавання мінералів натискається 20 разів (по 5% за натискання), поки очікуване значення рівня мінералів не досягне $\geq 95\%$.

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 $\leq 50\%$. Далі кнопка додавання кисню натискається 20 разів (по 5% за натискання), що націлене на підняття рівень кисню до очікуваного значення $\geq 95\%$.

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 стану, коли температура не стане $\leq 50\%$. Потім кнопка підвищення температури натискається 20 разів (по 1 градуса за натискання), поки температура не досягне $\geq 95\%$. Після цього тестуємо зниження температури, виконуючи аналогічні дії (перевантаження до $\geq 50\%$) досягнення значення $\leq 5\%$ 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 $\leq 50\%$. Потім кнопка підвищення лужності натискається 64 рази (по 0.25 рН за натискання), поки лужність не досягне $\geq 95\%$. Після цього тестуємо підвищення кислотності, виконуючи аналогічні дії (перевантаження до $\geq 50\%$) досягнення значення $\leq 5\%$ іншою кнопкою (підвищення кислотності).

2. Пріоритет: Високий - важливий функціонал веб-клієнту.

2.6 Функціонал, що не тестується

В рамках 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, чис 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 використовувався 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 (ЧЕСК та тригер(-и)), хоч певні перевірки можуть бути гіпотетично 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 зачасти) винятками, які хоча б і не в наявному кодї (хоч гіпотетично й такий код може мати помилки в такїй 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 запитів (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 “солєння” 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 обмеженнями 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.

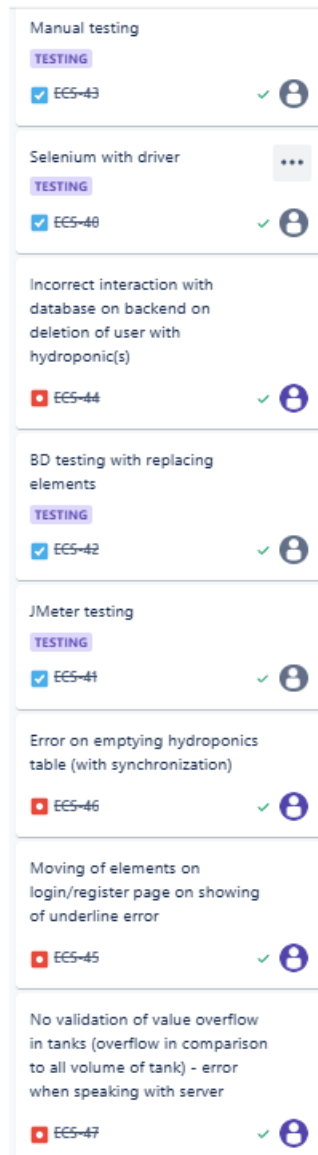


Рисунок 3.1 - червоним квадратом із білим колом в центрі позначені баги

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 зсування 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 елемента.
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).

```
File "src\pymssql\_pymssql.pyx", line 464, in pymssql._pymssql.Cursor.execute
sqlalchemy.exc.IntegrityError: (pymssql.exceptions.IntegrityError) (515, b"Cannot insert the value NULL into column 'user_id_owner', table 'Hydroponic.dbo.hydroponic'; column does not allow nulls. UPDATE fails.DB-Lib error message 20018, severity 16:
\nGeneral SQL Server error: Check messages from the SQL Server\n")
[SQL: UPDATE hydroponic SET user_id_owner=%(user_id_owner)s WHERE hydroponic.id = %(hydroponic_id)s]
[parameters: {'user_id_owner': None, 'hydroponic_id': 61}]
(Background on this error at: https://sqlalche.me/e/20/gkpi)
```

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 також 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 видалятися каскадно.

3.2.5 Інформація про баг

1. Пріоритетність - P1 (Висока).
2. Критичність - S1 (Блокуюча).
3. Статус – виправлений.
4. Автор - Михайлович Олександр Дмитрович.

5. Призначений до - Михайлович Олександр Дмитрович.

6. Оточення:

- a. Python 3.13.
- b. fastapi 0.115.6 (extras: standard).
- c. granian 1.7.4 (extras: reload).
- d. pyjwt 2.10.1.
- e. passlib 1.7.4 (extras: argon2).
- f. argon2-cffi 23.1.0.
- g. pymssql 2.3.2.
- h. sqlmodel 0.0.22.
- i. coverage 7.6.10.
- j. MS SQL 2022.
- k. 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).

```
File "src\pymssql\pymssql.pyx", line 464, in pymssql.pymssql.Cursor.execute
sqlalchemy.exc.IntegrityError: (pymssql.exceptions.IntegrityError) (515, b"Cannot insert the value NULL into column 'user_id_owner', table 'Hydroponic.dbo.hydroponic'; column does not allow nulls. UPDATE fails.DB-Lib error message 20018, severity 16:
\nGeneral SQL Server error: Check messages from the SQL Server\n")
[SQL: UPDATE hydroponic SET user_id_owner=%(user_id_owner)s WHERE hydroponic.id = %(hydroponic_id)s]
[parameters: {'user_id_owner': None, 'hydroponic_id': 64}]
(Background on this error at: https://sqlalche.me/e/20/gkpi)

-----
Ran 8 tests in 0.645s

FAILED (errors=1)
Finished running tests!
```

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.

```
280 @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 erroneous on delete"
```

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).

```
90 async def delete_user(user_id: int) -> None:
    with Session(__engine) as session:
        user = session.exec(select(User).where(User.id == user_id)).one_or_none()
        if user is None:
            raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="User not found")
        session.delete(user)
        session.commit()
```

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 загальному коректний код, навіть приймаючи до уваги код, який може здатись не коректним.

```
class Hydroponic(SQLModel, table=True):
    __table_args__ = {"implicit_returning": False} # due to https://docs.sqlalchemy.org/en/20/dialects/mssql.html#triggers

    id: Annotated[int | None, Field(default=None, primary_key=True)] = None
    user_id_owner: Annotated[int | None, Field(default=None, foreign_key="user.id", nullable=False, ondelete="CASCADE")] = None
    name: Annotated[str, Field(nullable=False, min_length=1, max_length=255)]

    20 water_amount: Annotated[float, Field(gt=0, nullable=False)]
    water_consumption: Annotated[float, Field(gt=0, nullable=False)]
```

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`.

```
class User(SQLModel, table=True):
    id: Annotated[int | None, Field(default=None, primary_key=True, nullable=False)] = None # default = None
    username: Annotated[str, Field(unique=True, nullable=False)]
    hash_salt: Annotated[str, Field()]

    70 hydroponics: List[Hydroponic] = Relationship(back_populates=None)
```

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_args__`) 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.

```
Received test ids from temp file.  
test_change_hydroponic_add_minerals (test_back.TestChangeHydroponic.test_change_hydroponic_add_minerals) ... ok  
-----  
Ran 1 test in 0.189s  
  
OK  
Finished running tests!
```

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].

```
class User(SQLModel, table=True):  
    id: Annotated[int | None, Field(default=None, primary_key=True, nullable=False)] = None # default = None  
    username: Annotated[str, Field(unique=True, nullable=False)]  
    hash_salt: Annotated[str, Field()]  
  
    hydroponics: List[Hydroponic] = Relationship(back_populates=None, passive_deletes=True)
```

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 серверу 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 endpoint-ів, тому тут з'являється

подвійна проблема з боку SQLAlchemy [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 десятків 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 обгортки 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 структури, 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.

```

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 = 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")

```

Рисунок 4.1 - тест перевірки C, 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 можливіфікація 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 endpoint-ів, важливо відмітити, що використовується два тестових двійники `Mock` для бази даних та тестовий двійник `Stub` для роботи з `JSON Web Token`-ом [2].

```
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")
```

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 SQLAlchemy).

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.

```
@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()
```

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

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

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 зазначений 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.

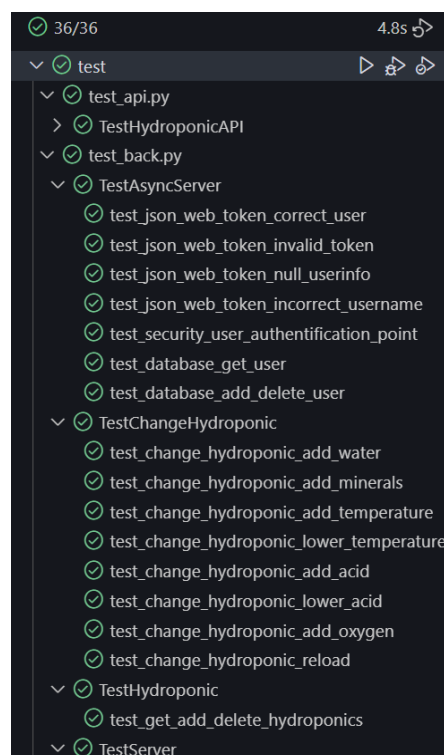


Рисунок 4.4 - проходження усіх 36 тестів

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

1. Покриття драйвера до бази даних: 78%, окремих компонент:
 - a. Безпосередньо обгортка драйвера 87%.
 - b. Драйвер 86%.

с. 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%.

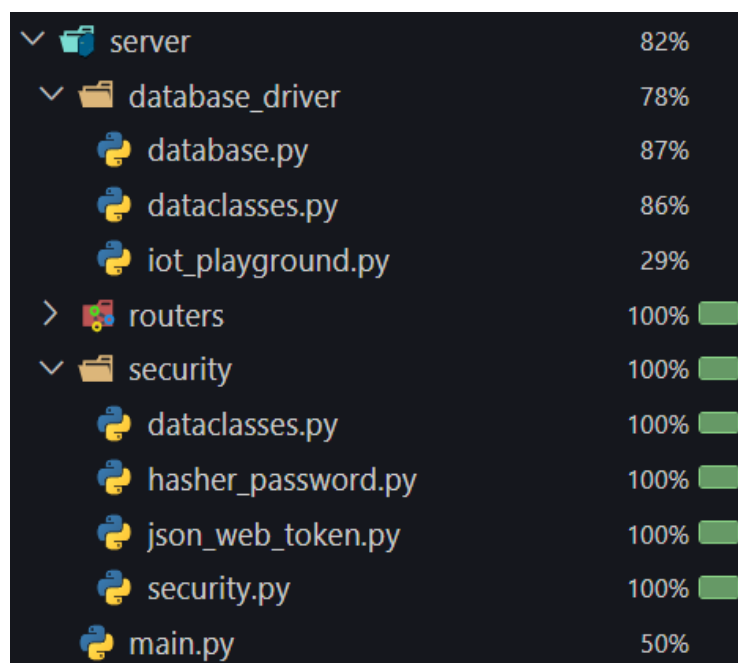


Рисунок 4.5 - рівень покриття коду тестами

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 Selenium 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 знайомим - 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 ефективно 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. Ресурсоемність: 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:

- a. `test_login` перевіряє коректність входу в систему.
- b. `test_signup` тестує функціональність реєстрації нового користувача.
- c. `test_unlogin` перевіряє коректність виходу з облікового запису.

2. Тести перенаправлення:

- a. `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:

- a. `test_add_delete_hydroponic` тестує додавання й видалення гідропоніки.

4. Тести загального контролю гідропонікою:

- a. `test_reload_button` перевіряє функцію оновлення усіх параметрів (отримання нових випадкових значень).

5. Тести керування гідропонікою

- a. `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).

```
@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"
```

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 токєну (порівнюється поле sub для username що було введено у поле логін).

4.2.4 Результат

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



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: $\pm 95\%$.
2. Додавання, видалення гідропонік: $\pm 90\%$ (окремі сценарії додаються вручну).
3. Контроль параметрів: $\pm 85\%$ (випадкові сценарії емуляції перевіряються частково).
4. Тестування кольорів заливок та тестування станів та помилок: $\pm 10\%$ (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).

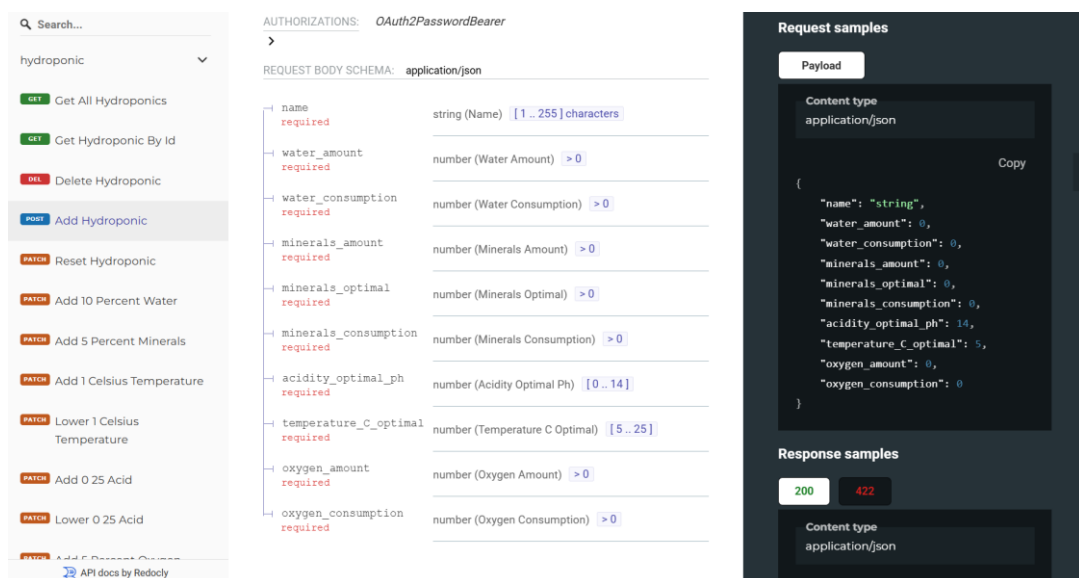


Рисунок 4.8 - документація до API сервера (redocly)

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.

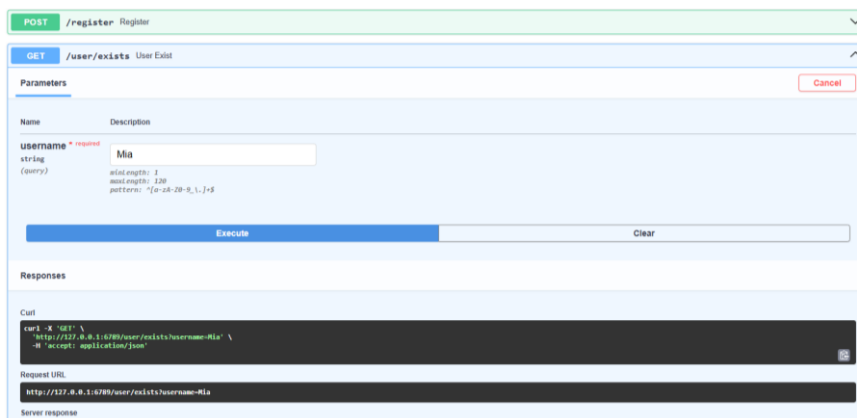


Рисунок 4.9 - Swagger UI

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.

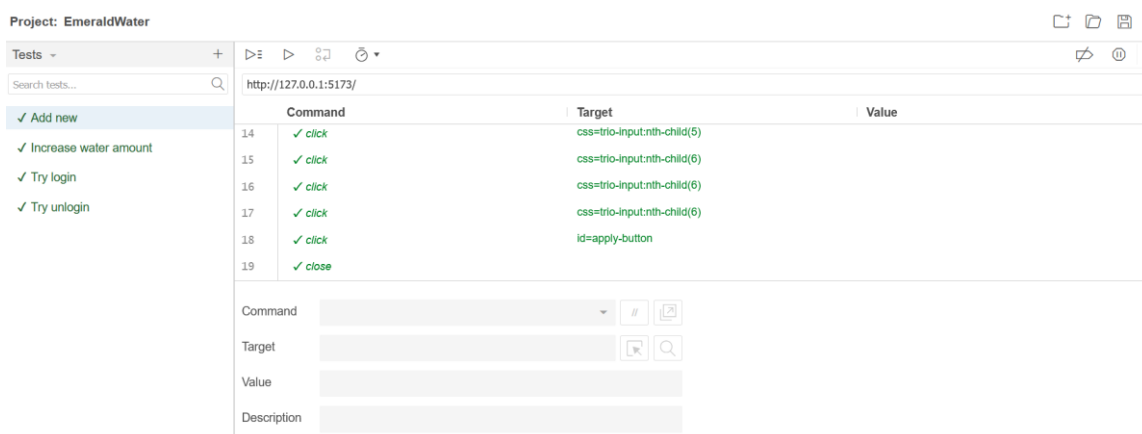


Рисунок 4.10 - Selenium IDE

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.

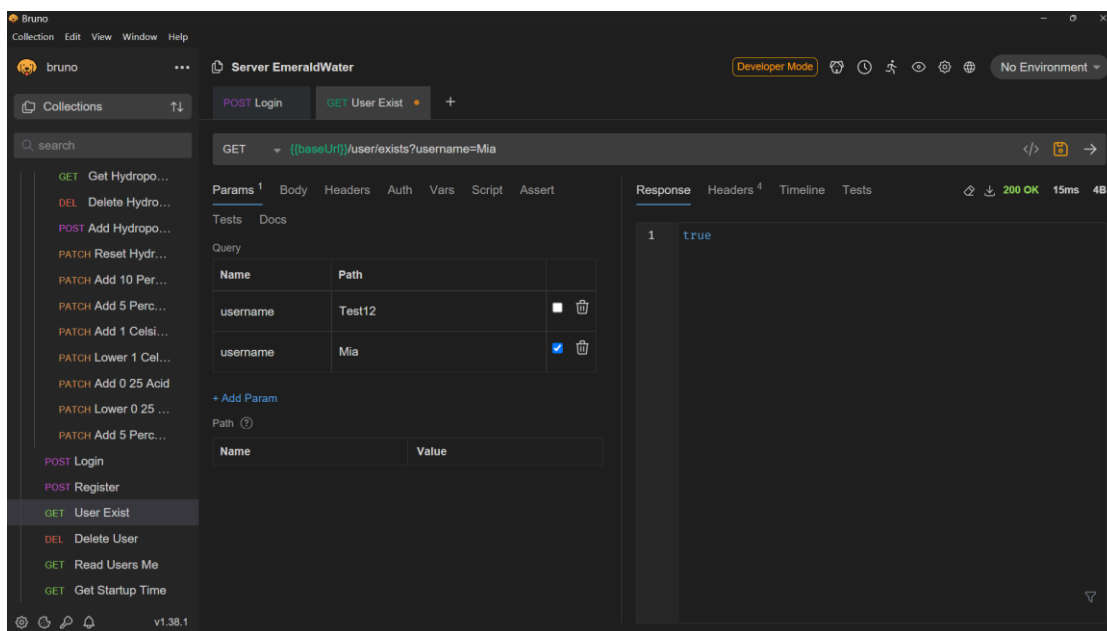


Рисунок 4.11 - Bruno

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

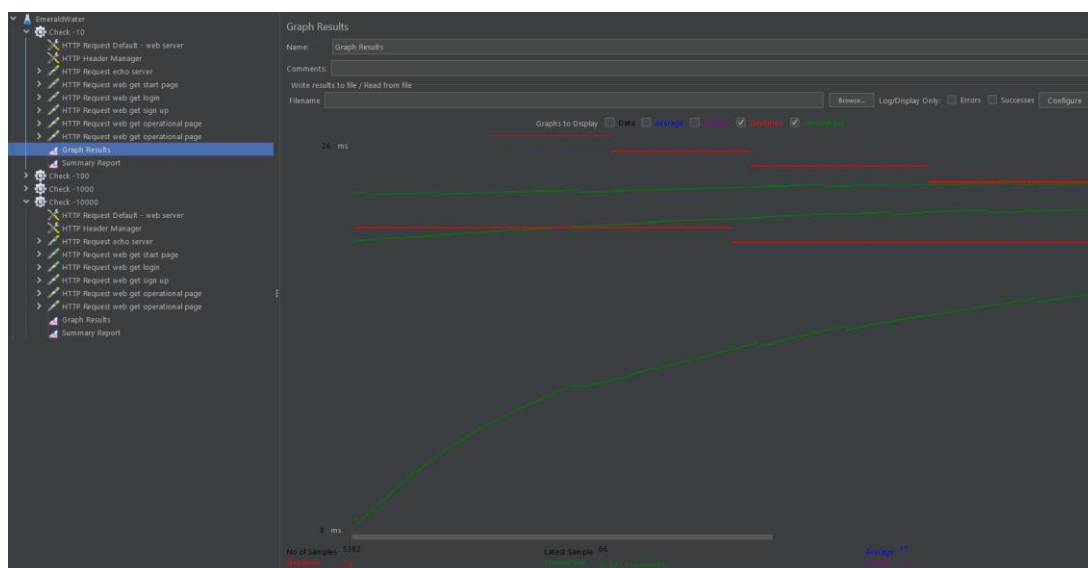


Рисунок 4.12 - інструмент перевірки навантаження Jmeter

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].

```
(18 rows affected)

+-----+
|Test Execution Summary|
+-----+

|No|Test Case Name                                     |Dur (ms)|Result |
+-----+-----+-----+-----+
|1 |[testHydroponic].[test_invalid_hydroponic_acidity_optimal_ph_out_of_range]|15|Success|
|2 |[testHydroponic].[test_invalid_hydroponic_acidity_ph_out_of_range]|14|Success|
|3 |[testHydroponic].[test_invalid_hydroponic_negative_water_amount]|12|Success|
|4 |[testHydroponic].[test_invalid_hydroponic_null_user_id_owner]|9|Success|
|5 |[testHydroponic].[test_invalid_hydroponic_oxygen_amount_non_positive]|16|Success|
|6 |[testHydroponic].[test_invalid_hydroponic_temperature_C_optimal_out_of_range]|29|Success|
|7 |[testHydroponic].[test_invalid_hydroponic_trigger_value_minerals_exceeds_limit]|13|Success|
|8 |[testHydroponic].[test_invalid_hydroponic_trigger_value_oxygen_exceeds_limit]|13|Success|
|9 |[testHydroponic].[test_invalid_hydroponic_trigger_value_water_exceeds_limit]|15|Success|
|10|[testHydroponic].[test_invalid_hydroponic_value_temperature_C_out_of_range]|15|Success|
|11|[testHydroponic].[test_invalid_user_name_collision]|16|Success|
|12|[testHydroponic].[test_invalid_user_null_hash_salt]|9|Success|
|13|[testHydroponic].[test_invalid_user_null_username]|4|Success|
|14|[testHydroponic].[test_valid_delete_cascade_hydroponic]|267|Success|
|15|[testHydroponic].[test_valid_insert_delete_hydroponic]|25|Success|
|16|[testHydroponic].[test_valid_insert_delete_user]|16|Success|
|17|[testHydroponic].[test_valid_update_hydroponic]|46|Success|
|18|[testHydroponic].[test_valid_update_user]|14|Success|
+-----+-----+-----+-----+

Test Case Summary: 18 test case(s) executed, 18 succeeded, 0 skipped, 0 failed, 0 errored.
```

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 ізолювали 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 даних, використовувалось оцінювання обширності тестування із застосування покриття коду, використовувались тестові двійники (Mock, Fake та Stub), налаштований та заповнений інструмент для оцінки навануження (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 зсування 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 методів 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. SQLAlchemy. SQLAlchemy. 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 А

Код тестування веб-клієнту Selenium WebDriver

НТУУ «КПІ» ІАТЕ ІІЗЕ

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_logged_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_logged_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_logged_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_logged_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 acidness 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 acidness 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 В

Код тестування unit тестами із застосування підміни тестовими
двійниками

НТУУ «КП» 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_USER
NAME)

# 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_USER
NAME)

        # 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 С

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

НТУУ «КПІ» ІАТЕ ІПЗЕ

Листів 17

```

# 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" # should be real user in database
TEST_PASSWORD = "secret" # should 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_authentication_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 ІІЗЕ

Листів 13

Київ - 2025

```
use MusicForTest;
```

```
GO
```

```
EXEC tSQLt.NewTestClass 'testMusic';
```

```
GO
```

```
-- Check incorrect 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 trigger
```

```

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 trigger
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'
```