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 «Система автоматизованого управління вентиляцією в приміщеннях»»**

Керівник: доцент Варава І.А.	Виконав Оніщук М.І
Допущено до захисту	Студент 3 курсу
«25» грудня 2024р.	Групи ТВ-22
Захищено з оцінкою	
_____	

Київ – 2024

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. ІГОРЯ СІКОРСЬКОГО»

Кафедра інженерії програмного забезпечення в енергетиці

КУРСОВА РОБОТА

з дисципліни: «Компоненти програмної інженерії 4»

на тему: «Тестова документація проєкту «Система автоматизованого управління вентиляцією в приміщеннях»»

Студента _____групи ТВ-22
напряму підготовки _____бакалавр_____
спеціальності ___121 Інженерія програмного забезпечення_____
_____Оніщук М. І._____
(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 - 2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут: Навчально-науковий атомної та теплової енергетики _____

Кафедра: Інженерія програмного забезпечення в енергетиці
Напрям підготовки 121 Інженерія програмного забезпечення_____

ЗАВДАННЯ

НА КУРСОВУ РОБОТУ СТУДЕНТА

_____Оніщуку Максимові Івановичу_____
(прізвище, ім'я, по-батькові)

1.Тема роботи: _«Тестова документація проєкту «Система автоматизованого управління вентиляцією в приміщеннях»»_____

Керівник проєкту(роботи):___Варава Іван Андрійович_доцент_____
(прізвище, ім'я, по-батькові, науковий ступінь, вчене звання)

2.Срок подання студентом роботи: 28 грудня 2024 р.

3.Вихідні дані до проєкту(роботи): розробити систему, яка отримує дані з датчиків вологості, забруднення та вологості повітря та регулює ці речі використовуючи різноманітні пристрої.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): опис проекту із включенням усіх вимог, складання traceability matrix, тест-план з усіма пунктами, баг-репорт та практичні тести.

5.Дата видачі завдання: «28» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

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

Студент _____ Оніщук М.І._____

Керівник курсової роботи _____ доцент Варава Іван Андрійович.

АНОТАЦІЯ

В рамках нашої курсової роботи було розроблено систему автоматизованого управління вентиляцією у приміщені. Дана система зчитує такі фізичні властивості як температура, вологість та забруднення(а саме кількість PM 2.5) повітря. Отримавши дані, система вирішує які пристрої потрібно ввімкнути і до якого моменту. Але оскільки наша система, не має ніяких фізичних сенсорів та пристроїв, вона лише їх симулює.

У процесі розробки було виконано кілька етапів тестування, зокрема: модульне тестування для перевірки коректності роботи окремих компонентів додатка, використання JMeter для оцінки навантаження та продуктивності, а також автоматизація тестів веб-інтерфейсу за допомогою Selenium.

Обсяг пояснювальної записки 40 аркушів, кількість ілюстрацій – 32, кількість додатків - 2.

ANNOTATION

As part of our course work, we developed a system for automated control of indoor ventilation. This system reads such physical properties as temperature, humidity, and pollution (namely, the amount of PM 2.5) of the air. After receiving the data, the system decides which devices should be turned on and when. But since our system does not have any physical sensors and devices, it only simulates them.

During the development process, several stages of testing were performed, including: unit testing to check the correctness of individual application components, using JMeter to evaluate load and performance, and automating web interface tests using Selenium.

The explanatory note is 40 pages long, the number of illustrations is 32, and the number of applications is 2.

Зміст

ВСТУП

У сучасних умовах управління технічними системами значну роль відіграє автоматизація процесів, яка дозволяє значно підвищити ефективність і зручність експлуатації різноманітних інженерних систем. Однією з таких систем є автоматизоване управління вентиляцією в приміщеннях, що має на меті забезпечення комфортного мікроклімату та поліпшення якості повітря в будівлях. Вентиляційні системи є невід'ємною частиною сучасної інфраструктури і знаходять застосування в житлових, комерційних, адміністративних та промислових будівлях. Забезпечення їх ефективної роботи є важливим завданням для забезпечення належних умов для людей.

Актуальність теми курсової роботи зумовлена необхідністю розробки та тестування ефективних методів автоматизації системи вентиляції, що дозволяє не лише забезпечити високий рівень комфорту в приміщеннях, а й підвищити енергоефективність за рахунок точного управління процесами вентиляції. Впровадження автоматизованої системи управління вентиляцією дозволяє досягти значних економічних та екологічних переваг, таких як зниження витрат на енергоспоживання, зменшення впливу на навколишнє середовище та підвищення рівня комфорту для мешканців чи працівників.

Одним із ключових етапів розробки програмного забезпечення для автоматизованої системи управління вентиляцією є тестування. Воно дозволяє перевірити, чи відповідає система встановленим вимогам, виявити помилки та недоліки в її роботі, а також оцінити її надійність, продуктивність і стійкість до навантажень. Тестування є необхідним інструментом для забезпечення високої якості системи та гарантування її коректної роботи в умовах реальної експлуатації.

В процесі тестування автоматизованої системи управління вентиляцією важливим аспектом є проведення кількох типів тестів, серед яких виділяються модульне та навантажувальне тестування. Кожен із цих етапів має свою специфіку і завдання, але всі вони сприяють забезпеченню стабільної та надійної роботи системи в умовах реального використання.

Модульне тестування є основою для перевірки роботи окремих компонентів програмного забезпечення, що складають систему. Завдяки модульному тестуванню можна виявити помилки на ранніх етапах розробки, коли система ще не інтегрована в повну структуру. В рамках цього етапу перевіряються функції, алгоритми і методи, що реалізуються в кожному окремому модулі системи. Це дозволяє забезпечити високу якість коду та уникнути майбутніх проблем при інтеграції компонентів. Модульне тестування також дозволяє забезпечити стабільність і правильну взаємодію компонентів, що особливо важливо в контексті автоматизованої системи управління, де навіть найменша помилка може призвести до серйозних наслідків.

Навантажувальне тестування є ще одним важливим етапом, який дозволяє оцінити, як система поводиться в умовах високих навантажень, що можуть виникнути під час її експлуатації. У випадку системи автоматизованого управління вентиляцією це тестування важливе для перевірки її здатності працювати в умовах великих обсягів даних та інтенсивного використання. Тестування проводиться з різними рівнями навантаження, і його основною метою є виявлення точок, у яких система може зламатися або втратити ефективність. Наприклад, перевіряється, як система реагує на значне збільшення кількості одночасних запитів або змін у параметрах вентиляції, що можуть виникнути при великих змінах температури або вологості. Результати навантажувального тестування дозволяють зрозуміти межі можливостей системи та оцінити, чи здатна вона справлятися з реальними умовами експлуатації.

Інтеграційне тестування є наступним етапом, на якому перевіряється, як окремі модулі взаємодіють один з одним. Важливо перевірити, чи правильно передаються дані між компонентами системи, чи не виникають проблеми з інтеграцією сторонніх сервісів або компонентів, таких як датчики або виконавчі механізми. Тестування на цьому етапі дозволяє переконатися, що всі частини системи функціонують як єдине ціле.

Завдяки таким типам тестування система автоматизованого управління вентиляцією може бути ретельно перевірена на всіх етапах її розробки та

впровадження, що гарантує її високу якість, надійність і безпеку. Пропонована тестова документація є основою для забезпечення коректної та ефективної роботи системи в реальних умовах, а також дозволяє виявити й усунути всі потенційні недоліки до запуску системи в експлуатацію.

Таким чином, виконання курсової роботи з розробки тестової документації для системи автоматизованого управління вентиляцією в приміщеннях є важливим етапом у процесі створення високоякісного програмного забезпечення, яке відповідатиме вимогам сучасних стандартів і забезпечить безперебійну та ефективну роботу автоматизованих систем вентиляції.

РОЗДІЛ 1. ОПИС ПРОЄКТУ

Метою цієї курсової роботи є розробка проєкту "Система автоматизованого управління вентиляцією в приміщеннях" та тестової документації для нього, що включає модульне тестування, тестування інтеграції компонентів системи, а також тестування на різноманітні навантаження. Завдяки цій документації буде забезпечено належний рівень перевірки програмного забезпечення, що дозволить мінімізувати ризики виникнення помилок у процесі експлуатації системи.

Основними функціями нашої системи є:

- Вимірювання фізичних даних з наступних датчиків:
  - Термометр(температура повітря, °C)
  - Датчик вологості(вимірювання вологості повітря, %)
  - Датчик твердих частинок PM2.5(μg/m³)
- На основі отриманих даних з датчиків увімкнення необхідних пристроїв для покращення стану мікроклімату та повітря у приміщенні. Всього наша система містить п’ять наступних пристроїв для догляду:
  - Кондиціонер
  - Система обігріву
  - Зволожувач повітря
  - Осушувач повітря
  - Вентиляція
- Оскільки наша система не містить справжніх приладів, а лише їх імітує, то система також надає можливість генерувати нові дані для обробки.

Перш за все, було створено наступні UML-діаграми для детальнішого опису нашої системи:

- Діаграма використання варіантів(Use-Case Diagram)
- Діаграма класів(Class Diagram)
- Діаграма компонентів(Component Diagram)

Діаграма варіантів використання (Use-Case Diagram) демонструє нам сценарій взаємодії користувача з системою. В нашому випадку користувач має

лише два сценарії для роботи з нашим застосунком, а саме: генерація даних для імітування зчитування з датчиків, ввімкнення аналізу даних та подальшого управління на їх основі. На рисунку 1.1 зображено нашу діаграму:

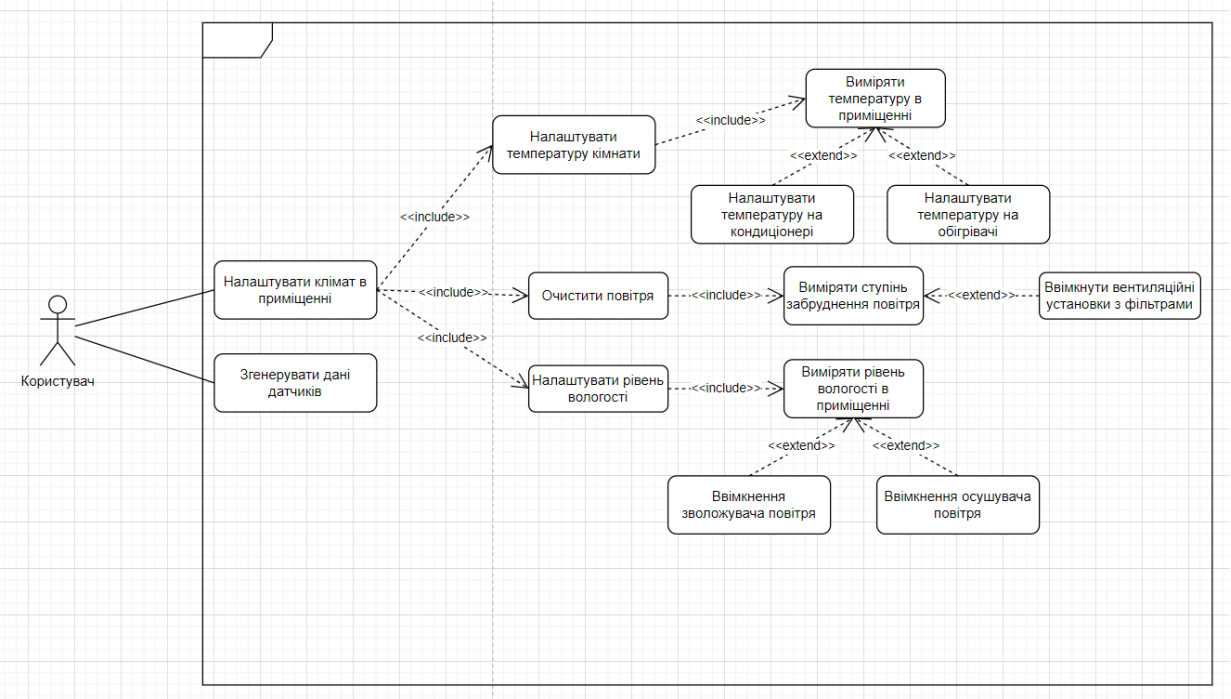


Рисунок 1.1. – Діаграма варіантів використання

Діаграма класів (Class Diagram) демонструє нам структуру системи з точки зору класів, їх властивостей (атрибутів), методів (операцій) та взаємозв'язків між ними. Це один із ключових артефактів об'єктно-орієнтованого проєктування, який дозволяє зрозуміти, як окремі компоненти системи взаємодіють один з одним. На рисунку 1.2 зображено нашу діаграму класів:

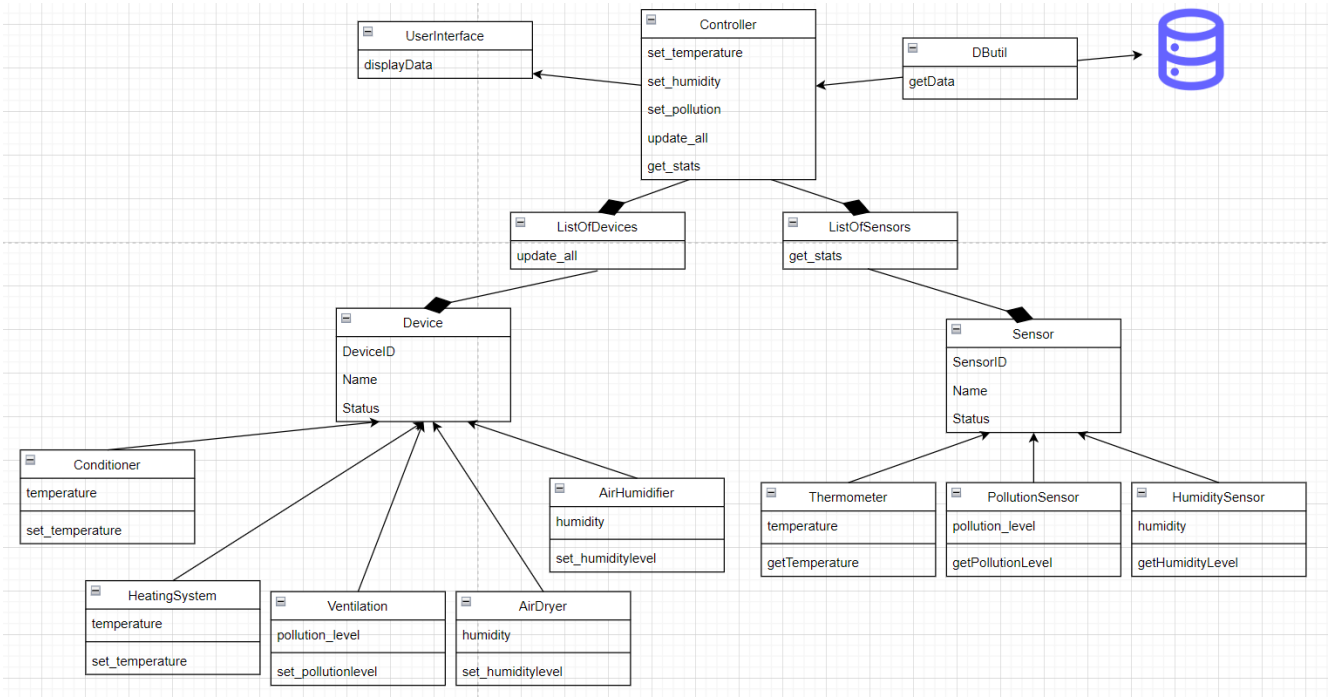


Рисунок 1.2. – Діаграма класів

Діаграма компонентів (Component Diagram) демонструє нам архітектурну структуру системи, зокрема, як різні програмні компоненти взаємодіють один з одним. Вона відображає компоненти системи, їх інтерфейси, залежності та зв'язки між ними. Ця діаграма дозволяє отримати уявлення про те, як організовані модулі системи, та як вони взаємодіють для досягнення функціональності. На рисунку 1.3 зображено нашу діаграму компонентів:

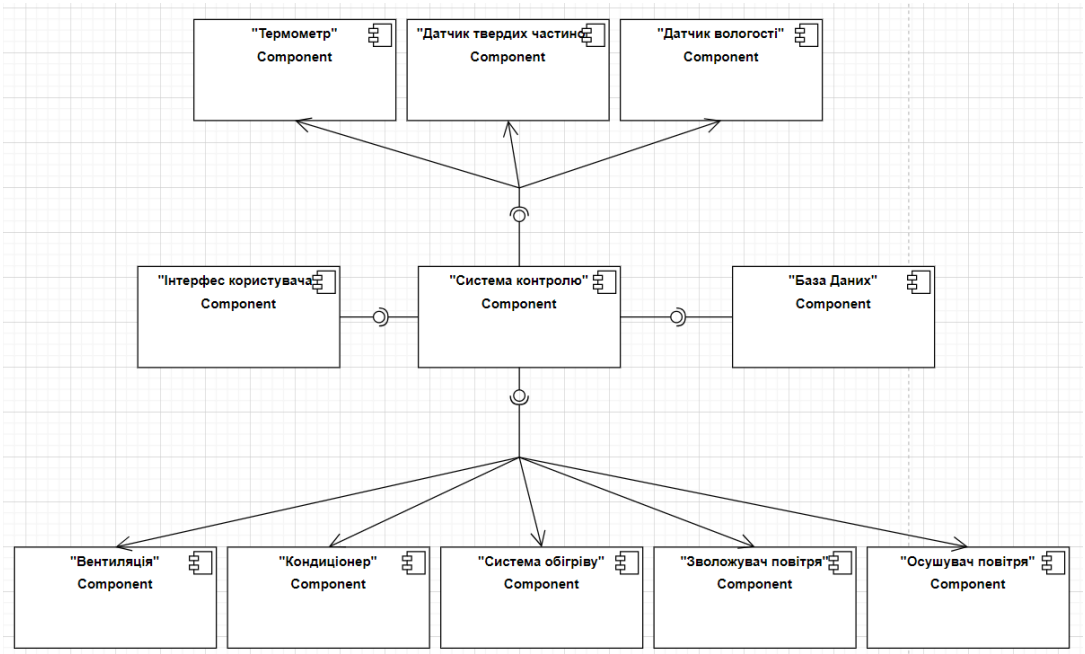


Рисунок 1.3. – Діаграма компонентів

1.1. Requirements Traceability Matrix

Requirements Traceability Matrix (RTM) – це документ, який відображає зв'язок між вимогами до системи та їх реалізацією у вигляді тест-кейсів, програмного коду або інших артефактів розробки. Він використовується для відстеження виконання вимог на всіх етапах життєвого циклу проєкту: від визначення вимог до їх реалізації, тестування та впровадження. Основною метою RTM є забезпечення того, щоб усі встановлені вимоги були належним чином враховані, реалізовані та протестовані.

RTM дозволяє контролювати відповідність розробленого продукту початковим вимогам, відстежувати прогрес виконання завдань та аналізувати покриття тестування. Це допомагає переконатися, що всі вимоги задоволені, і жодна з них не залишилася поза увагою. Документ зазвичай містить інформацію про ідентифікатор вимоги, її опис, пріоритетність, тест-кейси, які її перевіряють, стан реалізації та додаткові примітки.

RTM полегшує процес управління вимогами, забезпечуючи прозорість і точність під час роботи над проєктом. Він дозволяє зменшити ризики пропуску критичних вимог, спрощує комунікацію між учасниками команди та є ключовим інструментом для аудитів або аналізу готовності проєкту. RTM допомагає гарантувати, що кінцевий продукт відповідає початковим очікуванням і вимогам замовника. RTM нашої системи зображена на таблиці 1.1:

Таблиця 1.1. - Requirements Traceability Matrix веб-додатку

ID вимоги	Опис вимоги	Функціональність	Пріоритет	Відповідальний
F-1	Вимірювання температури та вологості	Система моніторингу повітря, що використовує датчики	Високий	Розробник
F-2	Вимірювання PM2.5	Моніторинг часток забруднень у реальному часі	Високий	Розробник
F-3	Запуск вентиляції користувачем для автоматизації	Алгоритми прийняття рішень	Високий	Розробник

NF-1	Безперервний моніторинг і передача даних	Підтримка роботи системи 24/7	Високий	Розробник
NF-2	Зручність інтерфейсу для кінцевих користувачів	Інтуїтивно зрозумілий інтерфейс	Середній	Розробник
ID вимоги	Опис вимоги	Функціональність	Пріоритет	Відповідальний

F-# - функціональні вимоги

NF-# - нефункціональні вимоги

РОЗДІЛ 2. ТЕСТ-ПЛАН

2.1. Загальні відомості про тест-план

Тест-план - це документ, який описує всі роботи, які виконуватиме команда тестування на проекті. Він містить ризики, перелік необхідних ресурсів, порядок, опис різних процесів тестування. Тест-план створюють на початковій стадії проекту, коли йде збирання вимог, формується технічне завдання, стає зрозумілим обсяг роботи та перелік завдань.[1]

Тестовий план виконує наступні функції:

- Тест-план забезпечує чіткий план дій для тестувальників і команди.
- Встановлює межі тестування та перелік функцій, які будуть перевірені.
- Визначає необхідні інструменти, середовище, тестові дані та людські ресурси.
- Визначає, які результати вважатимуться прийнятними для успішного завершення тестування.
- Сприяє підтриманню стандартів тестування та моніторингу прогресу.

Тест-план є критично важливим документом для забезпечення ефективності та узгодженості тестування.

2.2. Тест-план розробленого продукту

Тест-план для системи автоматизованого управління вентиляцією включає перевірку функціональності, надійності, продуктивності та відповідності вимогам. Охоплює тестування ключових функцій, інтеграції компонентів (сенсори, виконавчі пристрої, інтерфейс) та зручності використання. Передбачає модульне, інтеграційне й навантажувальне

тестування для оцінки роботи окремих частин і всієї системи під навантаженням. Основна мета – гарантувати ефективну та надійну роботу системи у реальних умовах.

2.2.1. Елементи для тестування

Основними елементами тестування в системі будуть:

- Сенсори – елементи, що вимірюють температуру, вологість та кількість твердих частинок повітря в приміщенні. Тестування їх здатності до передачі даних.
- Виконавчі пристрої – механізми, які реалізують зміни в системі (а саме, вентилятор, зволожувач, осушувач, кондиціонер та обігрівач). Перевірка їх реакції на зміни параметрів та управління через інтерфейс.
- Програмне забезпечення – алгоритми, які обробляють дані від сенсорів і керують виконавчими пристроями. Тестування правильності обробки даних та реалізації управління.
- Інтерфейс користувача – програмна частина, яка дозволяє користувачеві взаємодіяти з системою. Тестування зручності та функціональності інтерфейсу для налаштування та моніторингу.
- База даних – зберігання інформації про пристрої. Тестування коректності запису, збереження та відновлення даних.

Кожен з цих елементів тестування відповідає важливим функціональним можливостям системи, що дозволяє забезпечити її стабільну роботу.

2.2.2. Функції що підлягають тестуванню

Функціональне тестування — це тип тестування програмного забезпечення, під час якого система перевіряється на відповідність вимогам і специфікаціям. Функціональне тестування гарантує, що вимоги або специфікації належним чином задовольняються продуктом. Цей тип тестування, у першу чергу, стосується результату обробки. Він зосереджений на моделюванні фактичного використання системи, але не розробляє жодних припущень про структуру системи. [2]

Функції які бути протестовані:

- Налаштування температури – перевірка коректності роботи механізму зміни температури в приміщеннях та забезпечення заданих значень.
- Контроль вологості – тестування функціональності з контролю та регулювання рівня вологості в приміщенні.
- Очищення повітря – перевірка роботи системи очищення повітря від твердих частинок.
- Управління зволожувачем та осушувачем повітря – тестування увімкнення та вимкнення зволожувача та осушувача, а також їх налаштування відповідно до параметрів.
- Інтерфейс користувача – перевірка зручності і коректності роботи інтерфейсу для взаємодії з користувачем, включаючи налаштування та відображення стану системи.
- Інтеграція компонентів системи – перевірка взаємодії між різними частинами системи, такими як сенсори, виконавчі пристрої та програмне забезпечення.

Дані тести перевіряють різні функціональні частини вашої системи для забезпечення коректної роботи всіх основних функцій автоматизованого управління вентиляцією в приміщеннях.

2.2.3. Підхід

У цьому проєкті був застосований підхід до тестування, орієнтований на забезпечення стабільності, функціональності та продуктивності системи автоматизованого управління вентиляцією в приміщеннях. Підхід включає модульне тестування, а також тестування продуктивності. Для автоматизації тестування використовувалися інструменти, такі як Selenium та JMeter, які дозволяють охопити різні аспекти тестування.

• Модульне тестування

Основний акцент був зроблений на модульному тестуванні, яке перевіряє окремі компоненти системи (наприклад, налаштування

температури, контролювання вологості, управління вентиляцією та виконавчими пристроями). Модульне тестування дозволяє ізолювати кожен компонент і перевірити його функціональність, що допомагає швидко виявити помилки на ранніх етапах розробки.

- Тести Selenium

Selenium використовувався для автоматизації функціональних тестів веб-інтерфейсу системи. Тестування за допомогою Selenium дозволяє перевірити поведінку веб-сторінок, симулюючи реальні дії користувача, такі як налаштування параметрів вентиляції, введення даних у форми, навігація між сторінками та перевірка інтерфейсу управління. Це дає змогу перевірити, чи відповідає інтерфейс користувача вимогам та чи правильно працює взаємодія з усіма елементами.

- Тести JMeter

JMeter використовувався для тестування продуктивності системи, щоб оцінити її здатність обробляти великий потік даних від сенсорів і запитів на управління вентиляцією. Інструмент JMeter дозволяє симулювати багатокористувацькі сценарії та навантаження на сервери, що дає змогу перевірити, як система поводиться при високому навантаженні, а також оцінити швидкість реакції на зміни параметрів.

Цей комплексний підхід до тестування забезпечує високу якість та надійність системи, що дозволяє гарантувати ефективне та стабільне управління вентиляцією в приміщеннях.

2.2.4. Екологічні потреби

Для нашої системи автоматизованого управління вентиляцією в приміщеннях екологічні потреби полягають у забезпеченні ефективного використання ресурсів, зниженні енергоспоживання та мінімізації впливу на навколишнє середовище. Основні екологічні потреби вашої системи можуть бути наступними:

1. Енергетична ефективність

Система повинна забезпечувати оптимальне використання енергії для підтримки комфортного мікроклімату в приміщеннях. Вентиляція має працювати тільки в разі потреби, що дозволяє знизити енергоспоживання. Використання енергоефективних компонентів, таких як інтелектуальні сенсори для вимірювання температури та вологості, дозволяє системі автоматично адаптуватися до змін у приміщенні і таким чином зменшити витрати енергії.

2. Зниження викидів CO2

Завдяки оптимізації роботи вентиляції та покращенню енергоефективності система може допомогти зменшити викиди вуглекислого газу. Наприклад, у разі правильного регулювання параметрів температури та вологості, система дозволяє знизити необхідність в енергоємних методах охолодження та обігріву приміщень, що сприяє зниженню викидів CO2.

3. Поліпшення якості повітря

Автоматизовані системи вентиляції можуть активно контролювати рівень забруднення повітря в приміщеннях, забезпечуючи постійний обмін повітрям і очищення його від шкідливих часток, таких як пил або токсичні гази. Це допомагає підтримувати здоров’я мешканців або працівників, зменшуючи потребу в механічному очищенні повітря або використанні шкідливих для екології продуктів.

4. Використання сталих матеріалів та компонентів

При розробці та впровадженні системи важливо використовувати екологічно чисті матеріали та компоненти, які можуть бути перероблені або мають мінімальний вплив на довкілля. Це включає вибір компонентів, які мають низький рівень енергоспоживання, тривалий термін служби і не шкодять природі.

5. Зниження шумового забруднення

Автоматизоване управління вентиляцією може допомогти знизити рівень шуму, що є важливим для забезпечення комфортних умов в будівлях.

Оптимізація роботи вентиляційних систем дозволяє уникати надмірного шуму, що створюється за рахунок неефективного функціонування технічних засобів.

Таким чином, екологічні потреби вашої системи полягають у створенні енергоефективного, екологічно чистого та безпечного середовища для користувачів, що відповідає сучасним вимогам сталого розвитку.

2.2.5. Графік

Жовтень:

1. Створення базової архітектури програми:
    - Збір вимог до проекту
    - Розробка архітектури програмного забезпечення

Листопад:

- Розробка сторінки для перегляду даних з датчиків (температура, вологість, тверді частинки)
- Розробка системи
- Можливість ввімкнення автоматичного режиму роботи системи
- Інтеграція бази даних

Грудень:

1. Тестування продуктивності:
    - Використання JMeter для оцінки ефективності роботи системи при великому навантаженні
    - Виявлення потенційних проблем у часі відповіді на запити та швидкості обробки даних
    - Оптимізація запитів до сервера та обробки даних для забезпечення стабільної роботи системи при високому навантаженні
2. Автоматизоване тестування інтерфейсу користувача:
    - Використання Selenium для автоматизованого тестування інтерфейсу користувача
    - Перевірка коректності відображення даних з датчиків

- Перевірка роботи системи в різних сценаріях експлуатації для забезпечення надійності та стабільності.

2.2.5. Ризики та непередбачені обставини

У процесі розробки та експлуатації автоматизованої системи управління вентиляцією в приміщеннях можуть виникати різні ризики, які можуть вплинути на ефективність роботи системи або її безпеку. Основні ризики, які можуть бути пов’язані з розробкою та впровадженням цієї системи, включають:

1. Технічні ризики:
    - Несправність або низька точність датчиків: Некоректні дані, що надходять від датчиків (температури, вологості, тверді частинки), можуть призвести до неправильного регулювання системи вентиляції та вплинути на ефективність роботи.
    - Проблеми з інтеграцією компонентів: Якщо різні частини системи (датчики, сервери, програмне забезпечення) не будуть належним чином інтегровані, це може викликати збої у роботі або зниження ефективності системи.
2. Програмні ризики:
    - Помилки в коді: Виявлені баги або невірно реалізовані функції можуть призвести до некоректної роботи системи або її відмови
    - Низька продуктивність: Низька швидкість обробки запитів або недостатня продуктивність при високих навантаженнях може призвести до затримок у роботі системи або її відмови при великій кількості підключених датчиків або користувачів.
3. Ризики безпеки:
    - Невірна обробка даних: Якщо система не забезпечує належний рівень захисту даних користувачів або чутливих відомостей про

приміщення, це може призвести до витоку конфіденційної інформації.

- Кіберзагрози: Уразливості в програмному забезпеченні або мережевих протоколах можуть стати точками входу для зловмисників, які можуть отримати доступ до системи та маніпулювати її роботою.

2.3. Розробка тестів для системи

Тестування є невід'ємною частиною процесу розробки програмного забезпечення, оскільки воно забезпечує стабільну та безпечну роботу системи. Для досягнення високої якості програмного продукту необхідно розробити всебічний тестовий план, який охоплює всі аспекти функціональності системи, зокрема взаємодію з користувачем, обробку даних, інтеграцію з іншими компонентами, а також виконання критичних процесів. Це дозволяє не тільки забезпечити правильність роботи програмного продукту, але й виявити можливі дефекти на ранніх етапах, що суттєво підвищує надійність і ефективність роботи системи в реальних умовах.

2.3.1. Планування тестів

Планування тестів є важливою складовою процесу забезпечення якості системи для управління автоматизованим контролем вентиляції в приміщеннях. Це етап, на якому визначаються основні напрямки тестування, вибираються відповідні інструменти та методи тестування, а також розробляються тестові сценарії для перевірки критичних функціональних та нефункціональних аспектів системи.

Ключові етапи планування тестів:

1. Аналіз вимог до системи:
    - Першим кроком є ретельне вивчення функціональних і нефункціональних вимог до системи автоматизованого управління вентиляцією. Функціональні вимоги визначають, що

система повинна виконувати, зокрема автоматичне регулювання вентиляції та моніторинг параметрів (температури, вологості, кількість твердих частинок). Нефункціональні вимоги зосереджуються на характеристиках, таких як швидкість обробки даних, ефективність роботи при високих навантаженнях і безпека даних.

2. Типи тестування, які будуть реалізовані:
    - Модульне тестування: перевірка окремих компонентів системи, таких як алгоритми регулювання вентиляції, обробка даних з датчиків. Для цього використовуються автоматизовані тести, наприклад, за допомогою фреймворку pytest.
    - Інтеграційне тестування: перевірка взаємодії між різними компонентами системи, такими як датчики, сервери та програмне забезпечення, а також перевірка інтеграції з іншими зовнішніми системами (наприклад, базами даних або іншими інженерними системами будівлі).
    - Тестування інтерфейсу користувача (UI): перевірка інтерфейсу користувача для налаштування параметрів вентиляції та перегляду даних, за допомогою автоматизованих інструментів, таких як Selenium. Це включає перевірку зручності використання, коректності відображення даних та функціональності елементів інтерфейсу.
    - Тестування навантаження: оцінка здатності системи витримувати високі навантаження при одночасному підключенні багатьох датчиків і користувачів. Це тестування за допомогою таких інструментів, як JMeter, дає змогу симулювати високий потік запитів і перевірити, як система обробляє навантаження на сервери і швидкість реагування на запити.

2.3.2. Реалізація тестів

Для нашої системи було реалізовано модульні тести та тести за допомогою Selenium, що дозволяє перевіряти функціональність та стабільність системи.

У модульних тестах використовувався фреймворк pytest для тестування різних аспектів функціонування системи. Ось деякі з основних тестів, що були реалізовані:

- Тестування входу на сторінку. Перевірка коректності функціонування входу користувача на веб-сторінку. Приклад: test_home_page перевіряє, чи правильно вивелась сторінка
- Тест для імітування зчитування даних з датчиків. У цьому випадку test_api_generate перевіряє правильність роботи генерації випадкових даних з датчиків
- Тест для отримання інформації про прилади з бази даних. Тест test_initialize_devices перевіряє чи створені всі об’єкти які відповідають класам приладів. Аналогічно для датчиків
- Тест для отримання результатів з сенсорів. Наприклад test_termometer_get_temperature створює тестовий об’єкт який описує термометр та отримує з нього дані та перевіряє їх коректність.
- Тест для контролю роботи сенсорів. Наприклад test_termometer_generate генерує випадкове значення температури та перевіряє чи воно знаходиться у потрібному діапазоні
- Тестування вимкнення приладів для обслуговування стану повітря у приміщенні. Наприклад test_ventilation_off змінити значення змінної status на 0(тобто вимкнути)
- Тест для перевірки отримання даних від сенсорів до сторінки. Маємо: test_api_get_data отримує дані від сенсорів та перевіряє чи корректно вони виводяться на веб-сторінці

На рисунках нижче наведені результати тестування наших модулів, а DButil(для роботи з базою даних), Sensors(для роботою з сенсорами), Devices(для роботи з приладами), Controller(для опрацювання даних), app(для виводу даних на веб-сторінку):

```
============================= test session starts ==============================
collecting ... collected 3 items

db_tests.py::test_initialize_devices PASSED                              [ 33%]
db_tests.py::test_initialize_sensors PASSED                              [ 66%]
db_tests.py::test_get_devices_and_sensors PASSED                         [100%]

============================== 3 passed in 3.00s ===============================
```

Рисунок 2.1 – Тести для бази даних

```
============================= test session starts ==============================
collecting ... collected 12 items

sensors_tests.py::test_pollution_sensor_generate PASSED                  [  8%]
sensors_tests.py::test_pollution_sensor_get_pollutionlevel PASSED        [ 16%]
sensors_tests.py::test_set_pollution PASSED                              [ 25%]
sensors_tests.py::test_humidity_sensor_generate PASSED                   [ 33%]
sensors_tests.py::test_humidity_sensor_get_humiditylevel PASSED          [ 41%]
sensors_tests.py::test_set_humidity PASSED                               [ 50%]
sensors_tests.py::test_set_humidity_decrease PASSED                      [ 58%]
sensors_tests.py::test_termometer_generate PASSED                        [ 66%]
sensors_tests.py::test_termometer_get_temperature PASSED                 [ 75%]
sensors_tests.py::test_set_temperature PASSED                            [ 83%]
sensors_tests.py::test_set_temperature_decrease PASSED                   [ 91%]
sensors_tests.py::test_turn_on_and_off_sensor PASSED                     [100%]

============================= 12 passed in 32.33s ==============================
```

Рисунок 2.2 – Тести для модуля роботи з сенсорами

```
============================= test session starts ==============================
collecting ... collected 11 items

devices_tests.py::test_ventilation_set_pollution PASSED                  [  9%]
devices_tests.py::test_conditioner_set_temperature PASSED                [ 18%]
devices_tests.py::test_air_humidifier_set_humiditylevel PASSED           [ 27%]
devices_tests.py::test_air_dryer_set_humiditylevel PASSED                [ 36%]
devices_tests.py::test_heating_set_temperature PASSED                    [ 45%]
devices_tests.py::test_device_on_off PASSED                              [ 54%]
devices_tests.py::test_ventilation_off PASSED                            [ 63%]
devices_tests.py::test_conditioner_off PASSED                            [ 72%]
devices_tests.py::test_air_humidifier_off PASSED                         [ 81%]
devices_tests.py::test_air_dryer_off PASSED                              [ 90%]
devices_tests.py::test_heating_off PASSED                                [100%]

============================== 11 passed in 0.06s ==============================
```

Рисунок 2.3 – Тести для модуля роботи з девайсами

```
============================= test session starts ==============================
collecting ... collected 3 items

controller_tests.py::test_controller_initialization PASSED               [ 33%]
controller_tests.py::test_generate_data PASSED                           [ 66%]
controller_tests.py::test_get_stats PASSED                               [100%]

============================== 3 passed in 1.04s ===============================
```

Рисунок 2.4 – Тести для модуля опрацювання даних

```
============================= test session starts ==============================
collecting ... collected 3 items

server_tests.py::test_home_page PASSED                                   [ 33%]
server_tests.py::test_api_generate PASSED                                [ 66%]
server_tests.py::test_api_get_data PASSED                                [100%]

============================== 3 passed in 0.92s ===============================
```

Рисунок 2.5 – Тести для модуля застосунку

В тестах Selenium перевіряються основні функціональності веб-додатку, зокрема чи можна отримати елемент зі сторінки, перевірка генерації даних, тестування коректності виводу даних та перевірка оновлювальності даних.

- Тест для сторінки входу (test_page_load). Перевірено наявність елементу на сторінці за допомогою TAG_NAME. Тест перевіряє чи є на сторінці заголовок.
- Тест для перевірки отриманих даних (test_get_data). Перевіряється тип значень отриманих від сенсорів для виводу на сторінці.
- Тест для отримання інформації про типи даних отриманих зі сенсорів(test_info_data)
- Тест для перевірки оновлення даних на сторінці(test_auto_update). Даний тест отримує за допомогою CSS_SELECTOR дані сенсорів зі сторінки та через три секунди(оскільки дані повинні змінюватись кожної секунди) ще раз отримує дані та порівнює чи хоча б дані одного типу відрізняються

Результати Selenium тестів зображені на рисунку 2.1:

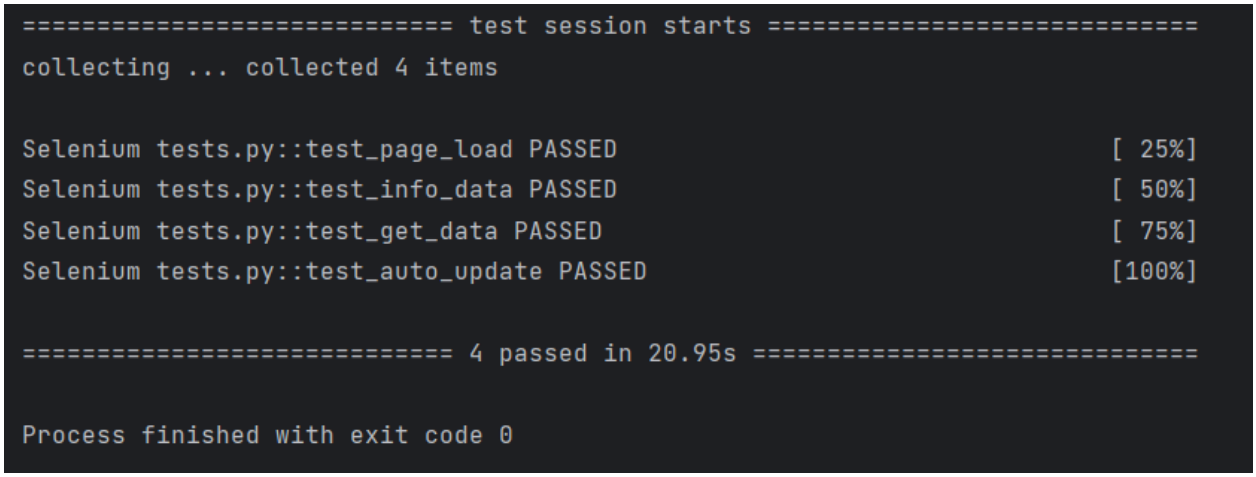


Рисунок 2.6. – Результат тестів Selenium

2.3.3. Аналіз тестів

Аналіз тесту у тестуванні програмного забезпечення – це процес перевірки та аналізу тестових артефактів для того, щоб заснувати умови тестування або тестові випадки. Метою аналізу тесту є збір вимог і визначення цілей тесту для створення основи умов тестування. Тому його також називають тестовою основою.[3]

Після аналізу тестів можна визначити, які аспекти системи потребують доопрацювання або оптимізації, а також зробити висновки про готовність додатку до подальших етапів тестування або запуску в експлуатацію. Під час тестування можуть бути виявлені помилки та дефекти, які необхідно проаналізувати.

Загальний аналіз тестів:

- Тести повинні охоплювати не лише стандартні сценарії, але й потенційно несприятливі ситуації, наприклад, перевірка на некоректні дані, надмірне навантаження або помилки в мережі.
- Важливо проводити ретельний аналіз помилок, щоб визначити причину їх виникнення, наприклад, чи проблема в програмному коді, чи на рівні інфраструктури, або ж у неправильному форматі даних.
- Тестування має бути повторюваним, щоб забезпечити стабільність і надійність системи після внесення змін у код. Тому аналіз результатів тестів допомагає підтримувати високий рівень якості системи на кожному етапі розробки.

У нашому випадку всі тести виконуються правильно.

РОЗДІЛ 3. БАГ РЕПОРТ

3.1. Поняття багу

Якщо коротко, то баг — це сленгове позначення програмної помилки. У свою чергу програмна помилка — це помилка у програмі або системі, через яку програма виявляє неочікувану поведінку і, як наслідок, видає результат, який не відповідає плану. Тобто, bug — це розбіжність між очікуваним і фактичним результатом.[4]

Баги – це нормальне та неминуче явище у будь-якому складному програмному забезпеченні. Масовий користувач познайомився з цим феноменом переважно через відеоігри та офісні програми. Навіть корпорації на кшталт Microsoft або Google, що містять штат із тисяч QA-фахівців, не можуть повністю гарантувати їхню відсутність у своїх продуктах.

Як правило, проблема виявляється в ході налагодження програми або в період бета-тестування. Іноді баг проявляє себе і після запуску товару ринку. І тут готуються оновлення.

Варіантів вияву помилок досить багато, серед них:

- з'являється повідомлення про помилку, але при цьому програма продовжує свою роботу;
- додаток просто зависає чи вилітає, жодних попереджень немає;
- з'являється попередження та додаток вилітає;
- одночасно з помилкою надсилається повідомлення розробникам.
- програмне забезпечення просто працює некоректно, у нього "відвалюється" важливий функціонал чи елементи інтерфейсу.[5]

Баги в додатках можуть бути викликані різними факторами і їх наявність не завжди очевидна. Вони можуть не проявлятися зовсім або виникати постійно. Ось деякі з основних факторів, що впливають на їх появу:

• Досвід розробників – рівень кваліфікації команди розробників, їх здатність розуміти код та ефективно вирішувати складні проблеми.

• Технічна складність – чим більше складних функцій і можливостей у проекті, тим більша ймовірність наявності помилок, що можуть бути пропущені.

• Кількість внутрішніх процесів – велика кількість операцій всередині програми може збільшити шанси на виникнення багів.

• Взаємодія з іншими учасниками – під час розробки програмісти не завжди можуть передбачити, як саме користувачі будуть взаємодіяти з додатком.

• Ефективність тестування – якість і охоплення тестів безпосередньо впливають на виявлення багів.

3.2. Поняття баг репорту

Баг-репорт — це технічний документ, який містить звіт про будь-які дефекти програмного забезпечення. Він включає детальний опис природи бага, умови його виникнення та іншу важливу інформацію. Стандартна структура баг-репорта виглядає так[6]:

1. Заголовок (Summary) – короткий і інформативний опис помилки, що вказує на її причину та тип.

2. Проект (Project) – повна назва проекту, в рамках якого була виявлена помилка.

3. Компонент програми (Component) – частина програми або функціональність, де виникнула помилка.

4. Номер версії (Version) – версія програми, в якій виявлено помилку.

5. Критичність (Severity) – рівень серйозності помилки за 5-рівневою шкалою: від S1 до S5.

6. Пріоритет (Priority) – важливість швидкості вирішення проблеми: P1, P2 або P3.

7. Статус (Status) – поточний стан помилки в процесі життєвого циклу, наприклад: новий, відкритий чи закритий.

8. Автор (Author) – ім’я особи, яка створила баг-репорт.

9. Призначений на (Assigned To) – ім’я особи, яка відповідає за виправлення помилки.

10. Опис (Description) – детальний опис обставин, в яких була виявлена помилка: ОС, сервіс-пак, версія бібліотеки, назва функції тощо. Також містить кроки для відтворення помилки, результат, який був отриманий (некоректний), та очікуваний результат після виправлення.

11. Прикріплений файл (Attachment) – скріншоти, відео або інші документи, які допомагають продемонструвати причину помилки чи пояснюють, чому результат є неправильним.

3.3. Проєкт в Jira

3.3.1. Про Jira

Jira – це популярна система управління проєктами та завданнями від компанії Atlassian. Вона необхідна для ефективного планування, оптимізації процесів, відстеження прогресу, злагодженої роботи та аналітики.[7]

JIRA вирізняється своєю адаптивністю, дозволяючи налаштувати систему під індивідуальні потреби команд і організацій. Вона забезпечує ефективне управління завданнями, створення спринтів, пріоритизацію задач, розподіл ресурсів та контроль строків виконання.

Основні принципи роботи в JIRA побудовані на чотирьох ключових елементах:

- Задача (Issue): Це базова одиниця роботи в JIRA, яка відстежується на всіх етапах — від постановки до завершення. Задачі можуть включати баги, користувацькі історії, епіки, завдання для інших відділів або навіть документи, які потрібно підготувати.
- Проєкт (Project): Об’єднує всі задачі в межах одного контексту, дозволяючи управляти інформацією, пов’язаною з певним продуктом або напрямком діяльності. Зазвичай кожен продукт або великий процес має свій окремий проєкт.

- Дошка (Board): Візуальний інструмент для відображення поточного прогресу завдань у межах проєкту. JIRA підтримує різні формати дошок, як-от Kanban для безперервного потоку задач або Sprint для управління ітераціями.
- Робочий процес (Workflow): Це набір етапів, через які проходять задачі, від їх створення до завершення. Робочий процес може включати стани, як-от "Очікує виконання", "У роботі", "Готово", і налаштовувати переходи між ними для точного відображення процесів команди.

Серед ключових особливостей Jira можна виділити:

- Інтуїтивно зрозумілий інтерфейс.
- Зручне управління правами доступу та версіями.
- Можливість налаштувати Jira під унікальні вимоги проєкту або команди.
- Просунутий трекінг задач. Jira забезпечує прозорість кожного аспекту проєкту, щоб моніторити прогрес, пріоритети, терміни та інші показники.

3.3.2. Веб-додаток в Jira

Епіки в Jira — це великі завдання або історії, які об’єднують менші, взаємопов’язані задачі. Вони слугують для організації роботи, яка потребує більше часу і зусиль, ніж стандартне завдання, і допомагають групувати задачі, що належать до однієї великої мети чи функціоналу.

Епіки для веб-додатку для управління особистим кабінетом студента були поділені на три основні групи, відповідно до завдань, визначених у тест-плані. Згідно з тест-планом, завдання, які повинні бути виконані в жовтні, включають:

- Збір вимог до проекту
- Розробка архітектури програмного забезпечення

У листопаді:

- Розробка сторінки для перегляду даних з датчиків (температура, вологість, тверді частинки)
- Можливість ввімкнення автоматичного режиму роботи системи
- Інтеграція бази даних
- Розробка системи

Ці епіки зосереджені на розробці основних функцій веб-додатку, які забезпечать інтерактивність та зручність використання для студентів і викладачів.

В грудні основна увага буде зосереджена на тестуванні та оптимізації функціоналу:

- Selenium тести
- Модульні тести
- Jmeter тестування

Ці епіки включають важливі етапи перевірки працездатності системи, що допоможуть забезпечити стабільність і ефективність роботи додатку перед його запуском. Частина створеного проєкту в Jira представлено на рисунку 3.1.

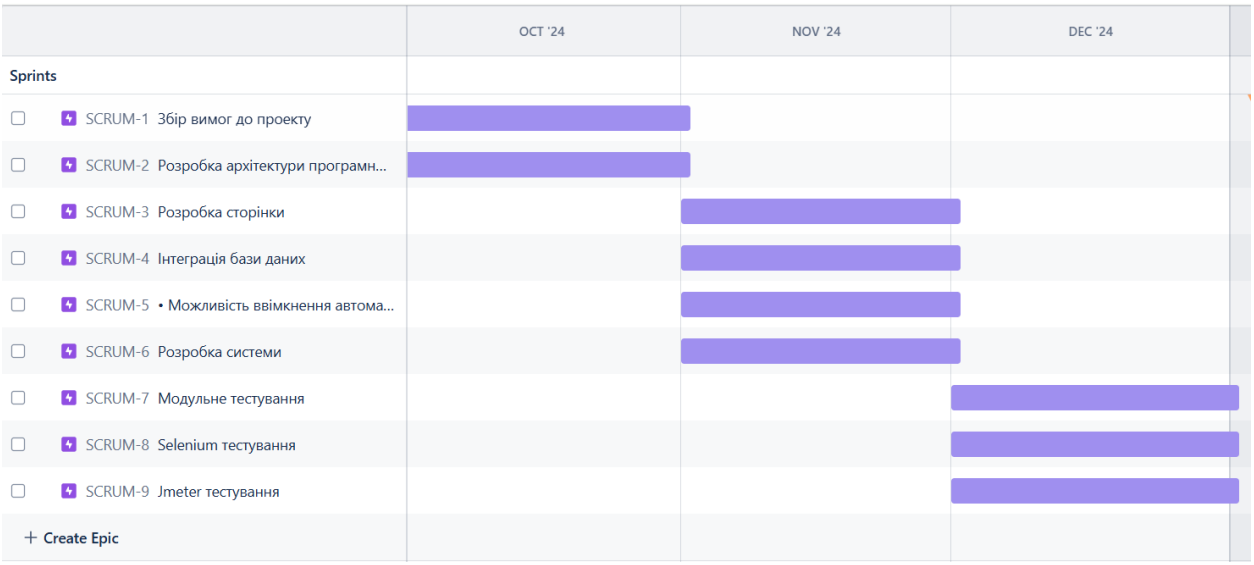


Рисунок 3.1. – Timeline в Jira для проєкту

На листопад і грудень було визначено ключові напрями роботи, орієнтовані на завершення базового функціоналу системи перед фінальним тестуванням. Основні завдання, спрямовані на розробку критичних елементів,

отримали найвищий пріоритет і були розпочаті ще на початкових етапах, щоб забезпечити стабільну основу для подальших доопрацювань та перевірок.

3.3. Баг репорт

У процесі виконання розробки системи автоматизованого управління вентиляцією було виявлено кілька багів. Один з них стосувався генерування нового значення вологості. Кожного разу при генерації нових даних значення вологості ніколи не перевищувало 20. На наступних рисунках зображено результати генерування:

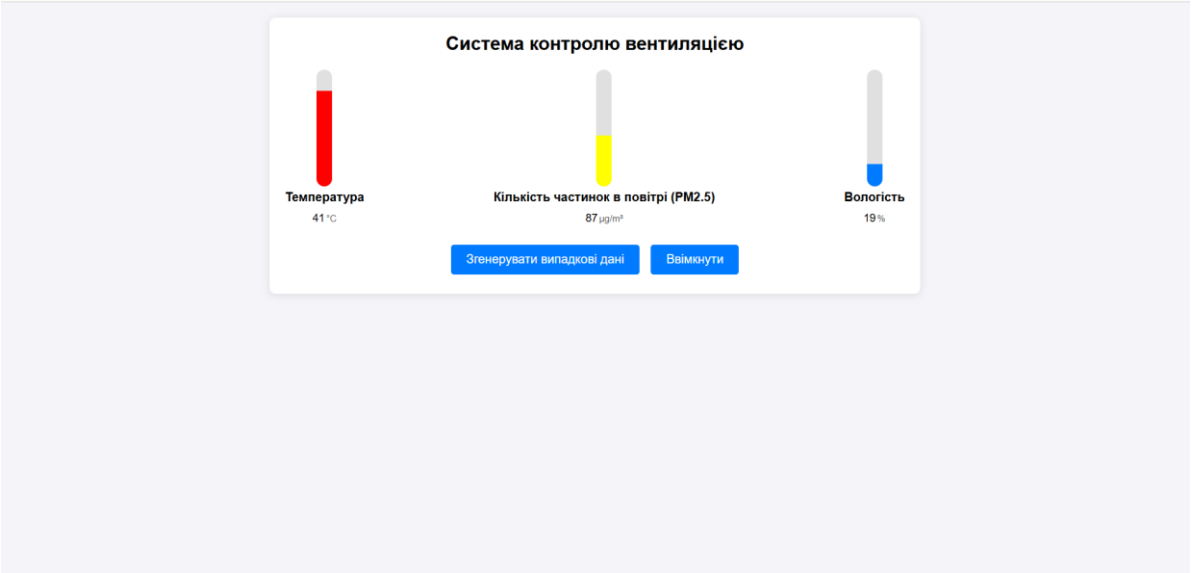


Рисунок 3.2. – Генерування нових даних

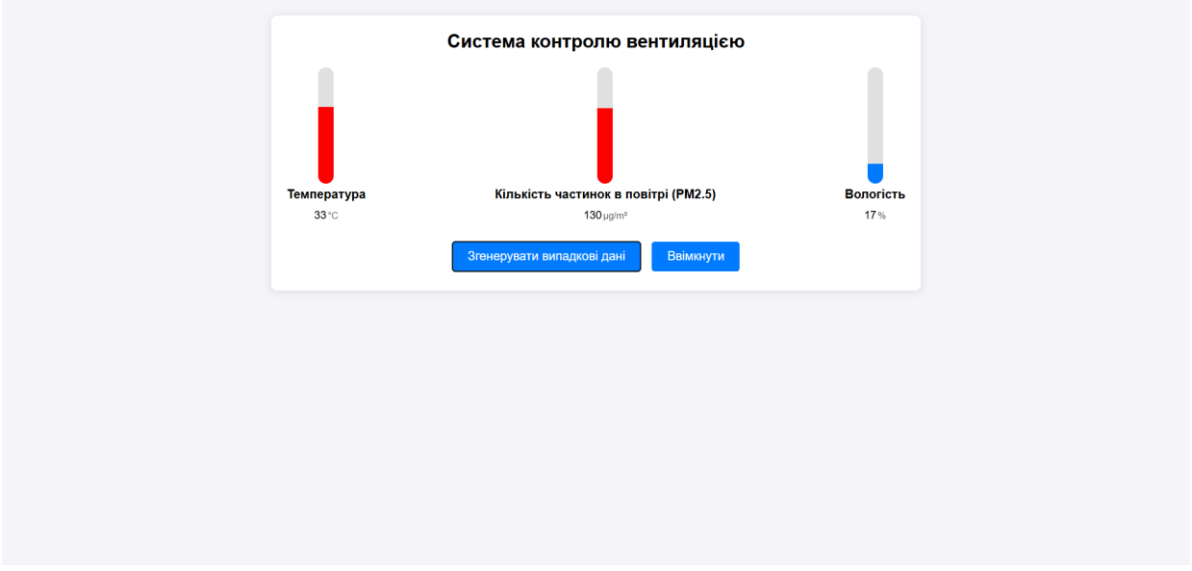


Рисунок 3.3. – Генерування нових даних

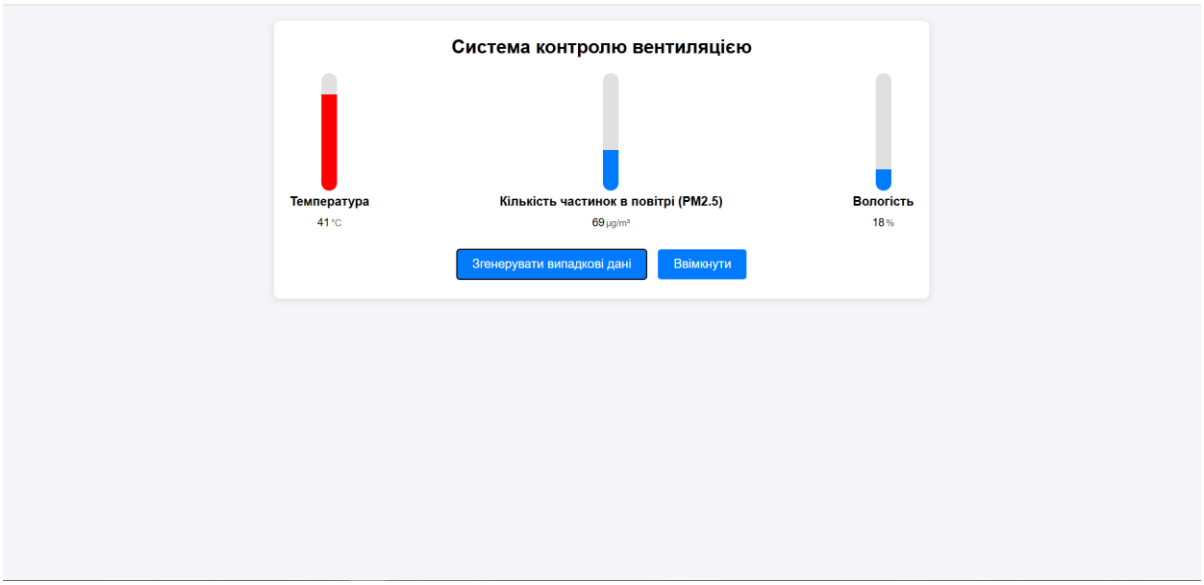


Рисунок 3.4. – Генерування нових даних

Баг: у маршруті /api/generate показник вологості завжди генерувався менше 20 %.

У процесі генерації нового значення вологості студенту була виявлена проблема, що у класі що відповідає за роботу Датчику вологості, у функції generate() верхньою межею генерації чисел було число 20. Код представлений на рисунку 3.5.

```
class HumiditySensor(Sensor):
    def __init__(self, devid, name):
        super().__init__(devid, name)
        self.humidity = 0
        self.generate()

    10 usages (3 dynamic)
    def generate(self):
        self.humidity = random.randrange( start: 15,  stop: 20)
```

Рисунок 3.5. – Функція генерації вологості

Змінюємо межу на 80, як на рисунку 3.6:

```
    10 usages (3 dynamic)
    def generate(self):
        self.humidity = random.randrange( start: 15,  stop: 80)
```

Рисунок 3.6. – Виправлена генерація

Після виправлення даної функції спробуємо згенерувати дані заново, як на рисунку 3.7:

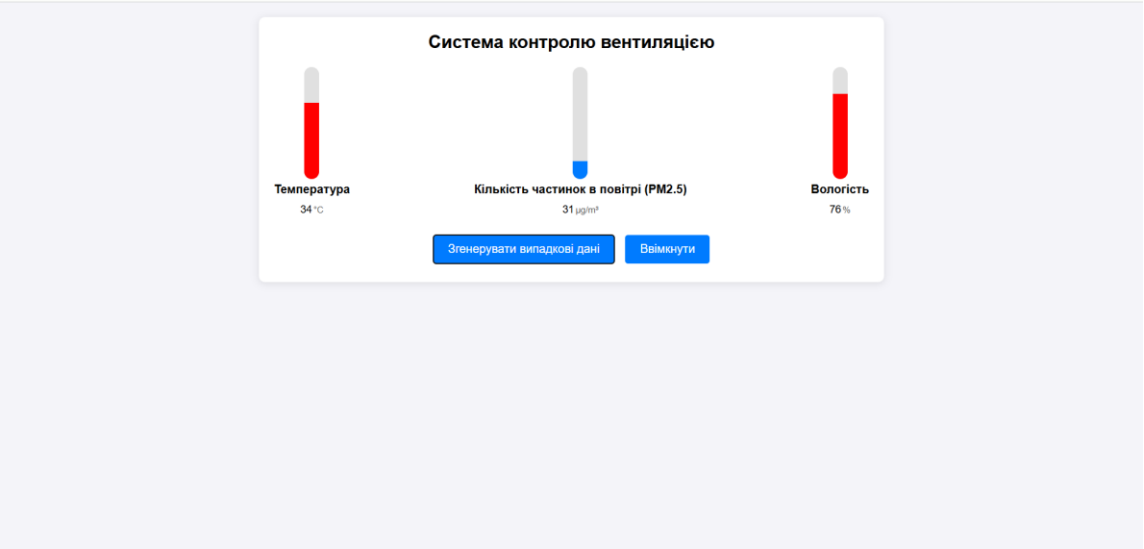


Рисунок 3.7. – Перевірка генерації

Помилку задокументували у Jira шляхом створення спеціального тикета для її аналізу та вирішення. Після внесення потрібних змін проблему було усунено, а тикет закрили зі статусом "виконано". У результаті генерація показників вологості працює належним чином, а дані успішно зберігаються в базі даних, що продемонстровано на рисунку 3.9.

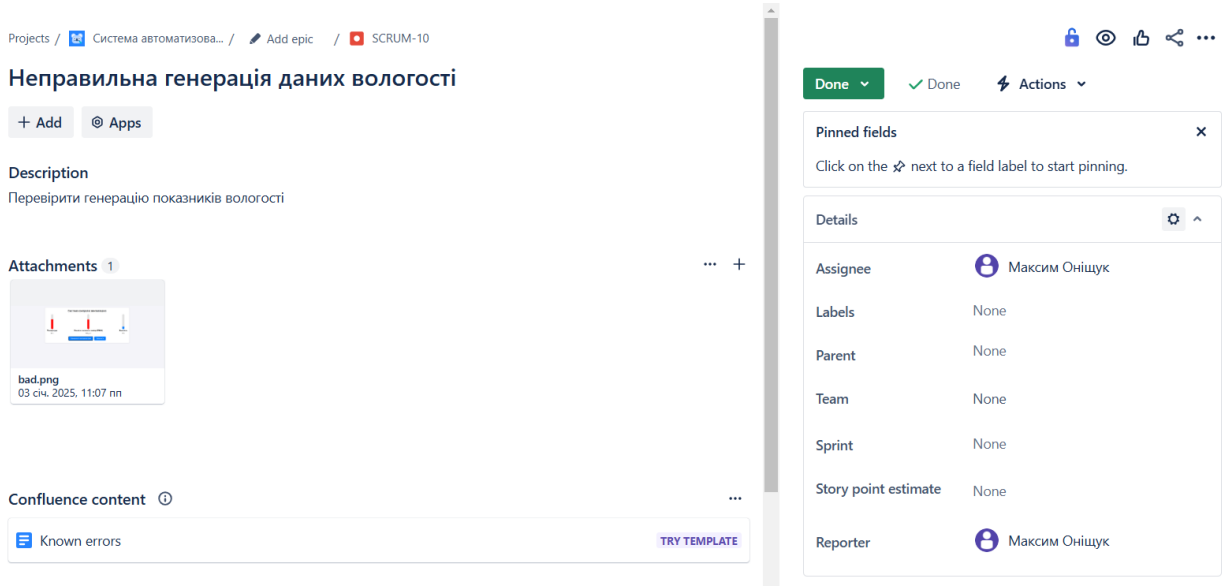


Рисунок 3.8. – Закритий баг репорт

Розглянемо інший баг, який виник під час роботи нашої системи. Після запуску автоматизованої вентиляції, температура, менша за 20, не

підвищувалась. Але інші прилади працювали правильно. На рисунку 3.9 зображено скріншот датчиків:

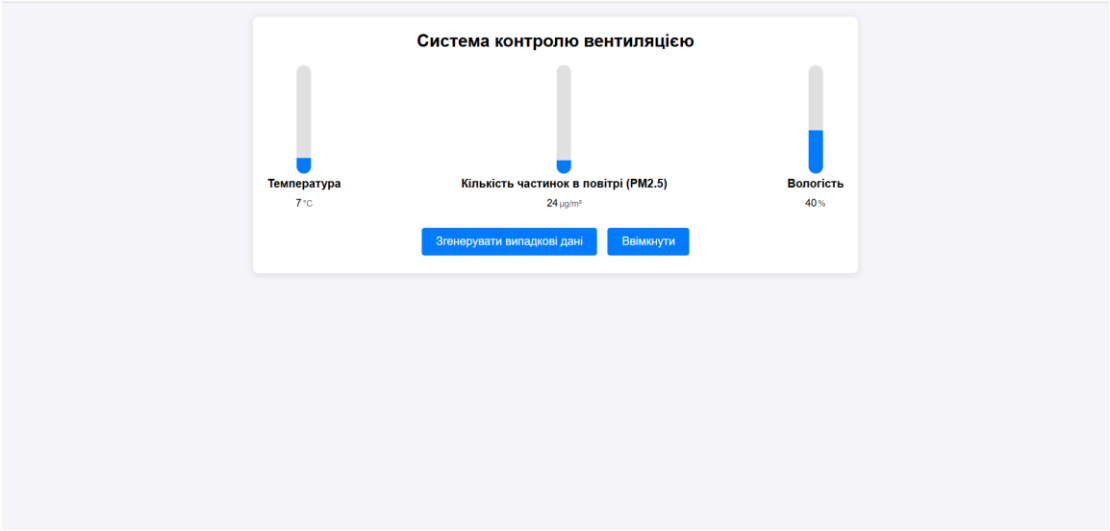


Рисунок 3.9. – Проблема з нагріванням приміщення

Передивившись код в модулі Controller, було помічено у функції зміни температури(set_temperature()) логічну помилку, через яку функція нагрівання повітря ніколи не викликається. Даний код наведено на рисунку 3.10:

```
async def set_temperature(self, temperature=20):
    if self.temperature > temperature:
        await self.conditioner.set_temperature(self.termometer, temperature)
    elif self.temperature > temperature:
        await self.heating_system.set_temperature(self.termometer, temperature)
```

Рисунок 3.10. – Неправильний код

Змінимо знак “>” у другій умові на знак “<”. Після чого перезапустимо нашу систему. Результат наведено на рисунку 3.11:

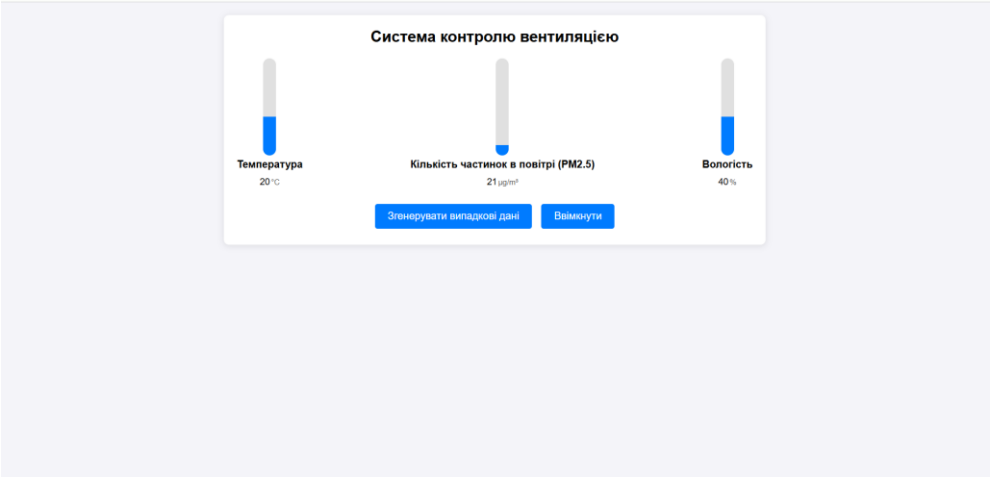


Рисунок 3.11. – Виправлений обігрів

Виявлений баг із обігрівом розкладу був доданий до Jira для подальшого відстеження та аналізу. Завдання отримало відповідний статус та опис проблеми, включаючи кроки для її відтворення.

Після внесення змін у код та успішного тестування виправлення завдання в Jira було позначено як виконане, що представлено на рисунку 3.12.

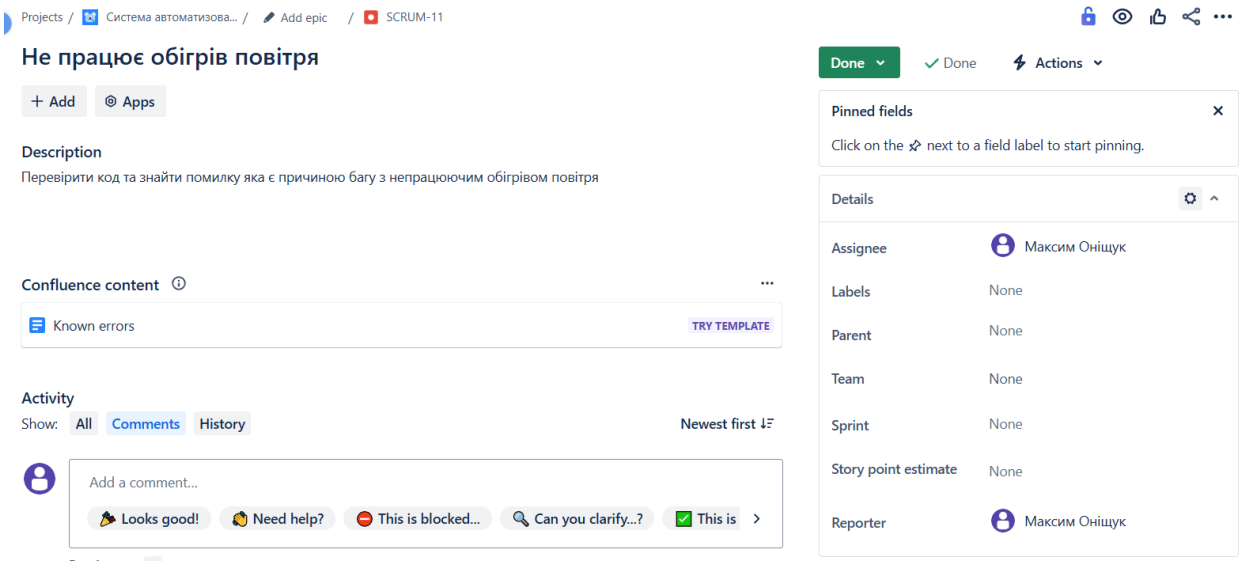


Рисунок 3.12. – Закритий баг репорт

Ще одною проблемою є вивід температури та вологості. Ці значення завжди мають однакові значення. Для того щоб відтворити даний баг потрібно просто на веб-сторінку. На рисунку 3.13 зображено скріншот сторінки:

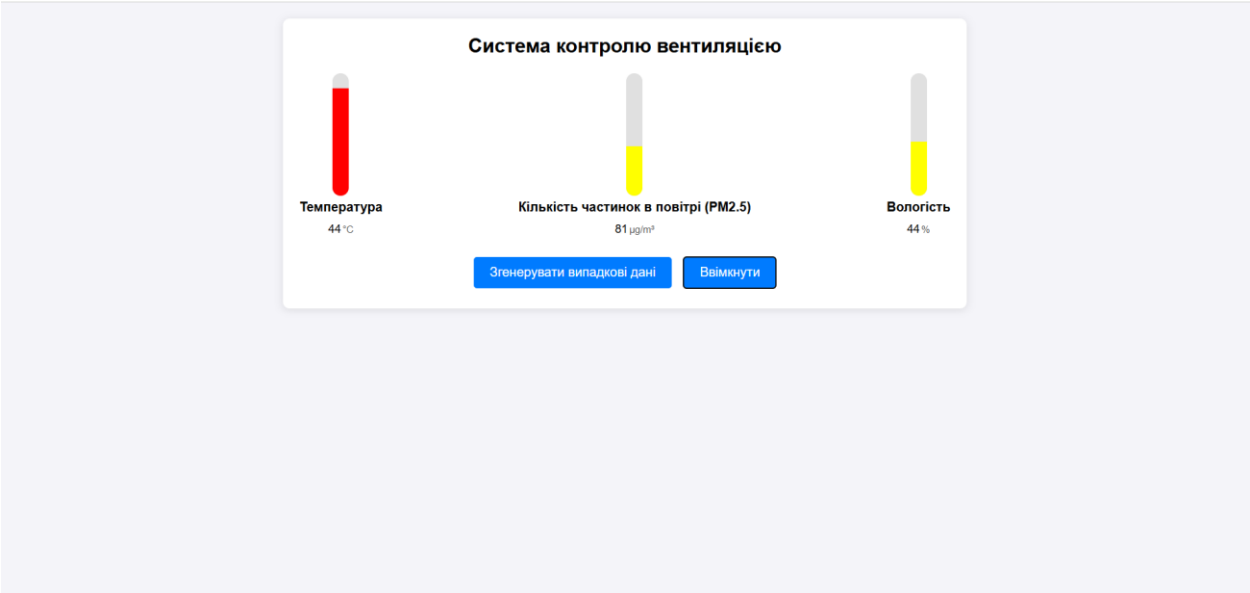


Рисунок 3.13. – Баг зі значеннями температури та вологості

Очікуваний результат: показники температури та вологості не залежать одне від одного

Фактичний результат: показники однакові.

У маршруті /api/get_data, у словнику response ключі “temperature” та “humidity” отримують однакові значення data[0]

Виправимо код

Виправлений код зображено на рисунку 3.14:

```python
@app.route( rule: '/api/get_data', methods=['GET'])
def get_sensor_data():
    data = control.get_stats()
    response = {
        "pollution": data[2],
        "humidity": data[1],
        "temperature": data[0],
    }
    return jsonify(response)
```

Рисунок 3.14. – Виправлений баг з розкладом

Результат виправлення коду наведено на рисунку 3.15:

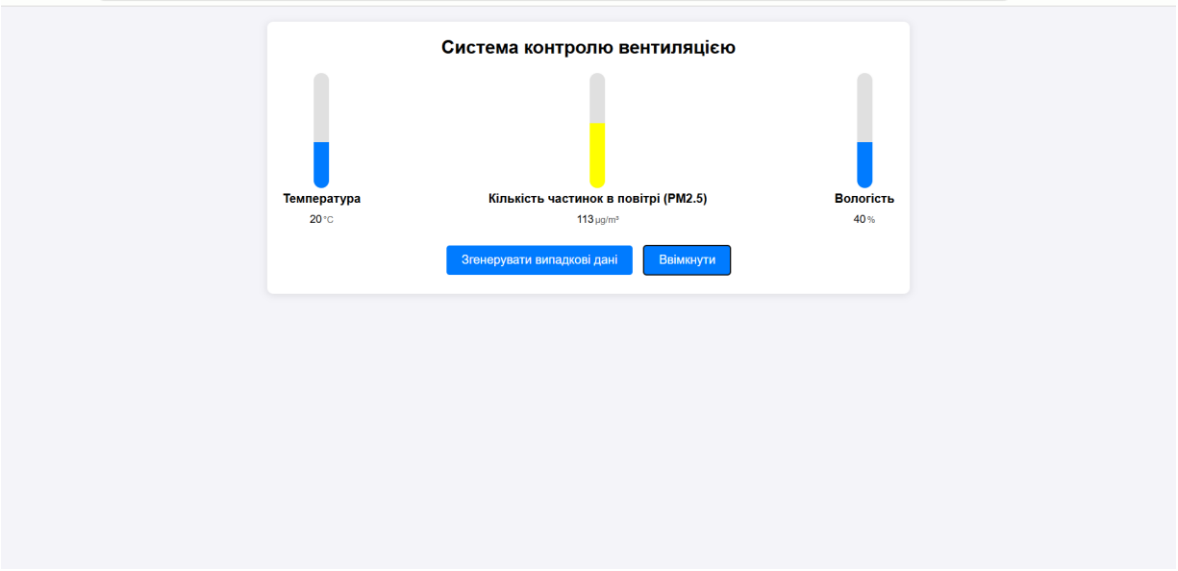


Рисунок 3.15. – Виправлена система

Після цього створимо баг-репорт нашого багу. На рисунку 3.16 зображено створення баг-репорту:

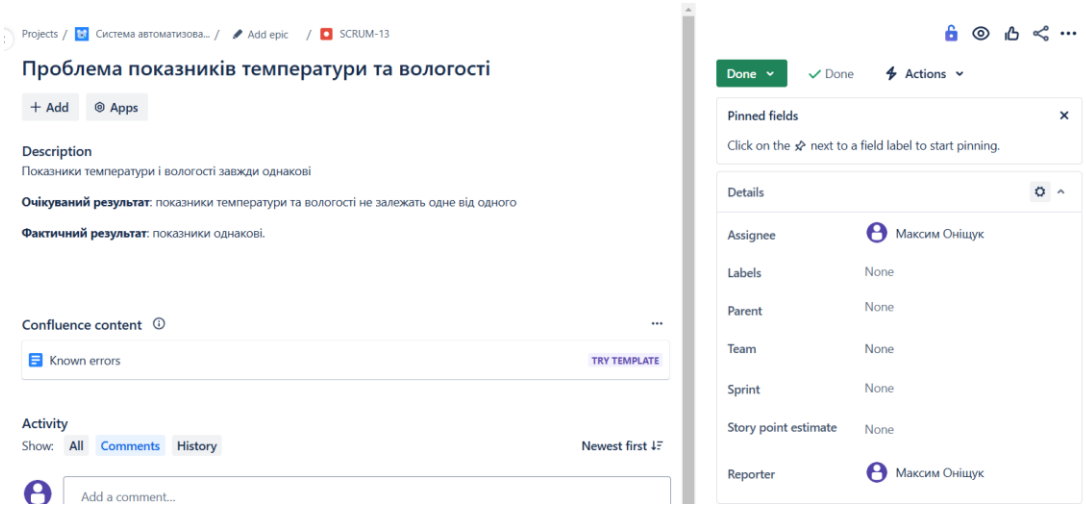


Рисунок 3.16. – Закритий баг-репорт

Усі баги були виправлені та відмічені як «Done», що представлено на рисунку 3.17.

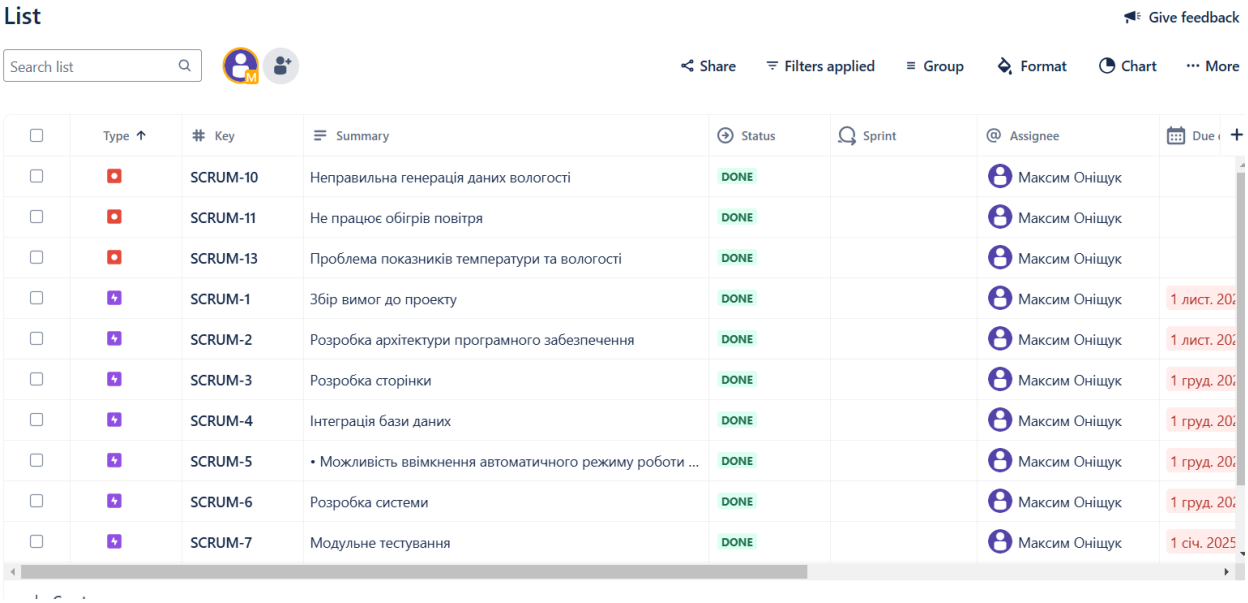


Рисунок 3.17. – Баг-репорти

ВИСНОВКИ

Під час виконання курсової роботи було розроблено систему автоматизованого управління вентиляцією у приміщеннях та тестову документації до її. Дана система реалізує інтуїтивно зручний інтерфейс для користувача.

Основними інструментами розробки є мова програмування Python, фреймворк Flask, система управління базою даних MongoDB та середовище розробки PyCharm.

Було розроблено наступний функціонал :

- Отримання даних(імітація) з сенсорів температури, вологи та кількості твердих частинок PM2.5
- Опрацювання системою отриманих фізичних показників з подальшим
- Автоматичне ввімкнення наступних приладів:
    - Вентиляція
    - Кондиціонер
    - Обігрів
    - Зволожувач повітря
    - Осушувач повітря
- Генерація нових показників сенсорів

Було проведено наступне тестування:

- Selenium-тести використовувались для автоматизації перевірки функціональності, зокрема тестування завантаження сторінки, генерації даних, виводу даних та їх оновлення.
- Модульні тести було створено для перевірки функцій системи, включаючи отримання інформації з бази даних, зміна температури та зчитування.

- JMeter-тести. Проведено навантажувальне тестування модулів виводу та генерації даних, щоб оцінити продуктивність та стабільність системи під час високих навантажень.

Досягнуті результати:

1. Забезпечення коректної роботи системи: У результаті проведених тестувань вдалося виявити та виправити декілька критичних помилок, пов'язаних з обробкою даних та взаємодією між модулями. Система стабільно працює при різних рівнях навантаження.

2. Оптимізація процесів: Вдалося оптимізувати алгоритми обробки даних, що дозволило значно знизити час відгуку системи та підвищити ефективність роботи приладів.

3. Покращення взаємодії з користувачем: Завдяки інтуїтивно зрозумілому інтерфейсу, користувачам було легко взаємодіяти з системою. Реалізація графічного відображення показників та стану приладів сприяла зручності роботи.

4. Виявлення та усунення багів: Завдяки використанню системи відстеження помилок, такі баги, як неправильне збереження даних або некоректне відображення показників, були швидко знайдені та виправлені.

5. Підвищення продуктивності: Завдяки проведеному навантажувальному тестуванню було вдосконалено систему управління даними, що дозволило системі стабільно працювати при високих навантаженнях, зберігаючи свою ефективність і точність.

6. Інтеграція з базою даних: Інтеграція з MongoDB забезпечила зберігання та швидкий доступ до історії даних, що дозволяє проводити подальший аналіз та удосконалення алгоритмів.

Загалом, система автоматизованого управління вентиляцією була розроблена та протестована успішно, що забезпечує її готовність до використання в реальних умовах для ефективного контролю за мікрокліматом у приміщеннях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тест-план не для галочки, або 8 запитань до замовника на старті проекту. | DOU. URL: https://dou.ua/lenta/columns/creating-quality-test-plan/
2. Різниця між функціональним і нефункціональним тестуванням. | QATestLab URL: https://training.qatestlab.com/blog/technical-articles/difference-between-functional-and-non-functional-testing/
3. Що таке аналіз тестів (основа тестування) у тестуванні програмного забезпечення? | GURU99. URL: https://www.guru99.com/uk/test-analysis-basis.html
4. Що таке Bug і Bug Report в тестуванні? | EPAM Campus. URL: https://campus.epam.ua/ua/blog/460
5. Баги: як у додатках виникають помилки і чому їх не треба боятися | Wezom. URL: https://wezom.com.ua/ua/blog/bagi-kak-v-prilozhenijah-voznikajut-oshibki-i-pochemu-ih-ne-nuzhno-bojatsja
6. Як правильно оформити баг-репорт | SpaceLab. URL: https://spacelab.ua/articles/yak-pravilno-oformiti-bag-report/
7. Що таке Jira і для чого вона потрібна | GoIT. URL: https://goit.global/ua/articles/shcho-take-jira-i-dlia-choho-vona-potribna/

ДОДАТОК 1

J-meter навантажувальне тестування

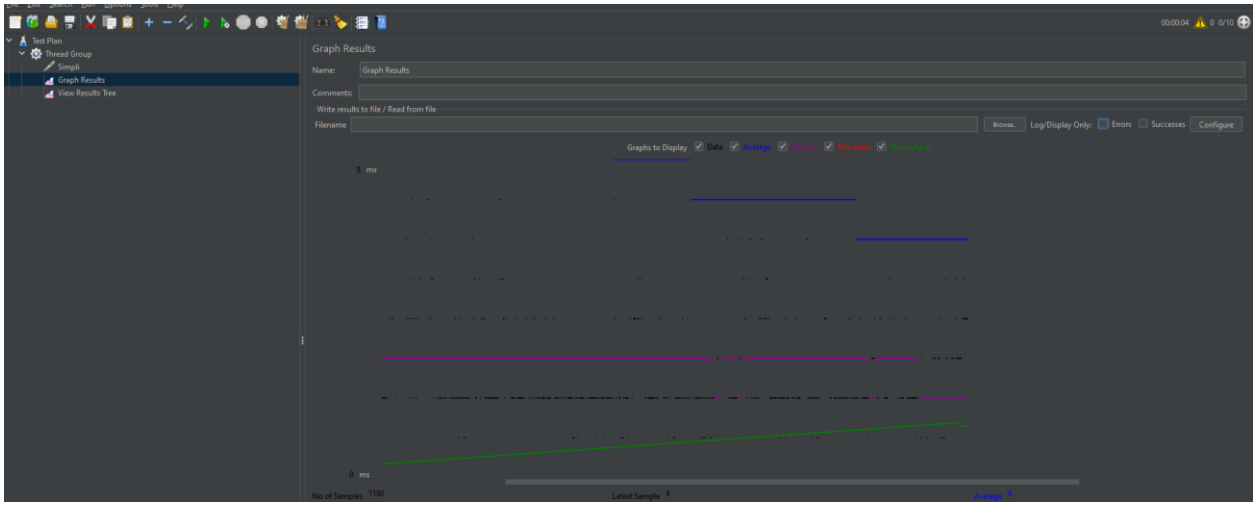
НТУУ «КПІ» ІАТЕ ІПЗЕ

Листів 3

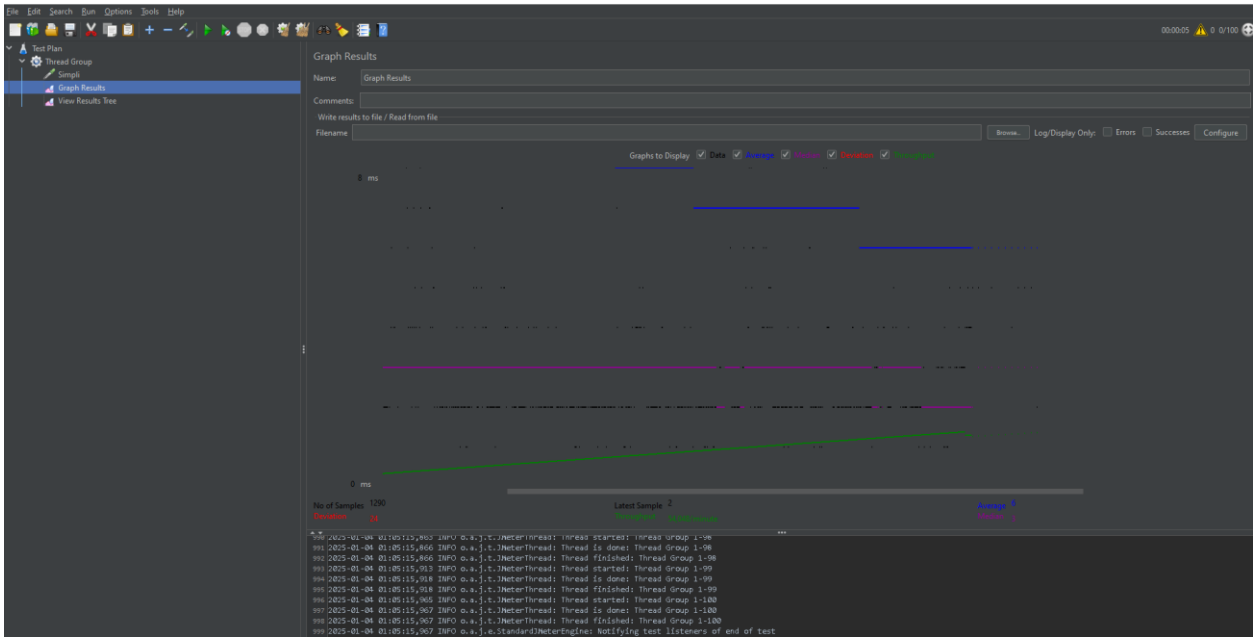
Київ – 2024

/api/get_data

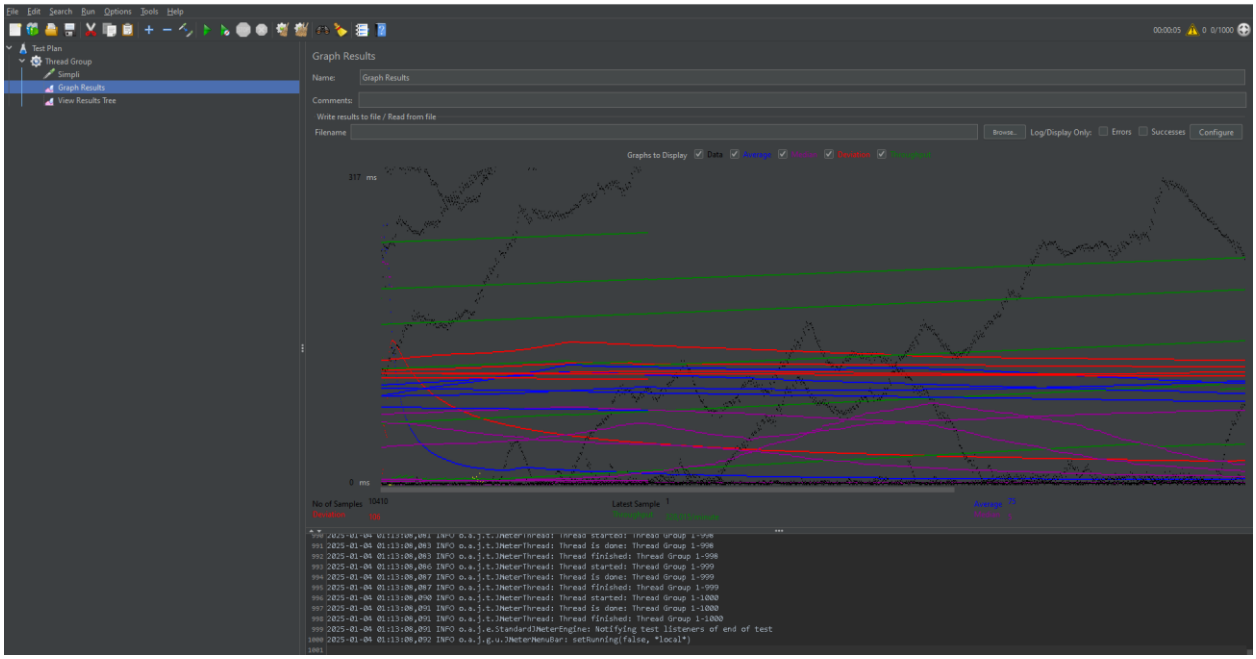
10 користувачів



100 користувачів

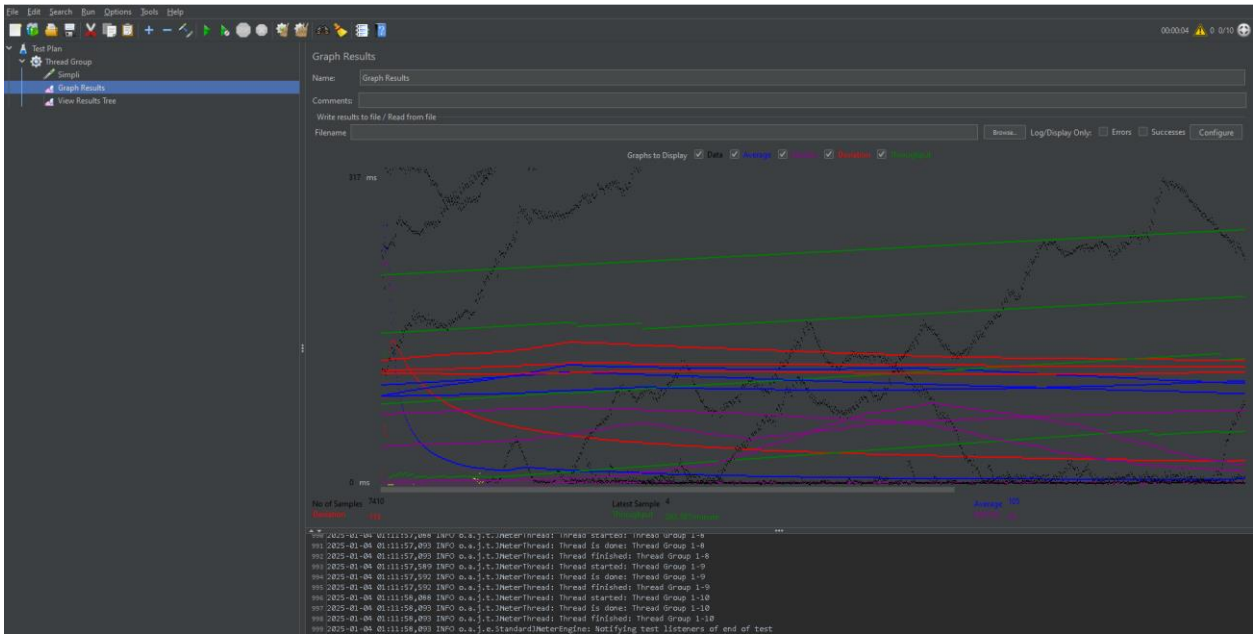


1000 користувачів

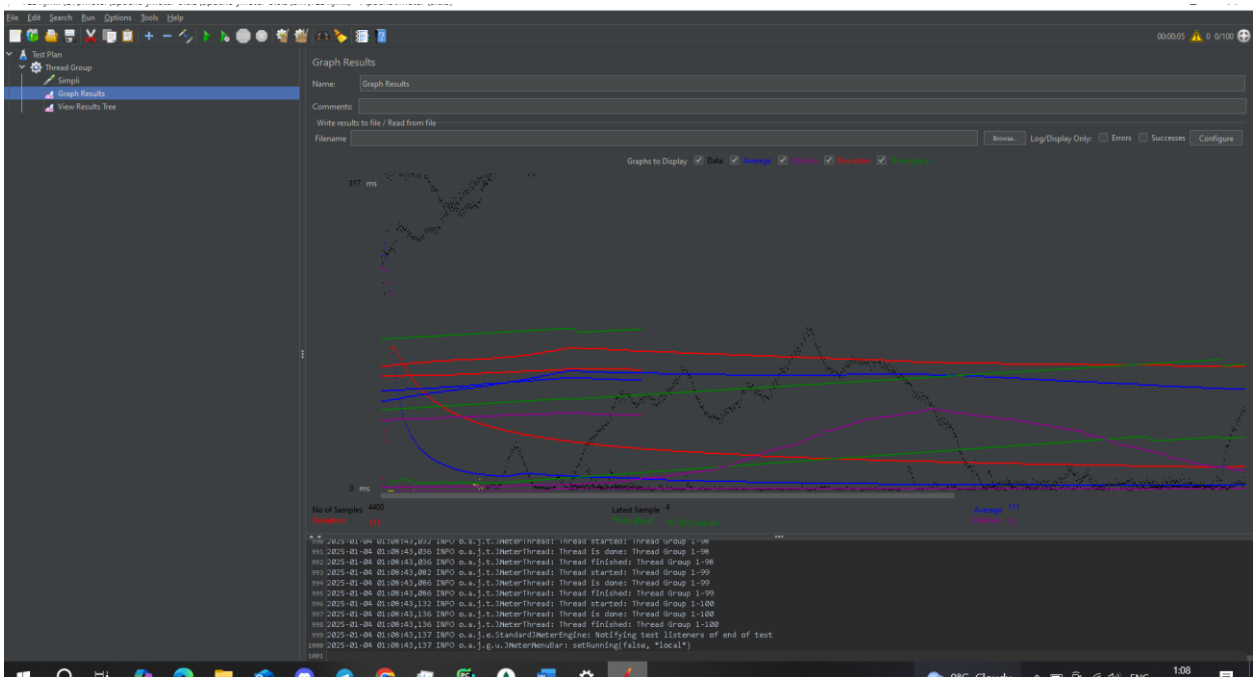


/api/generate

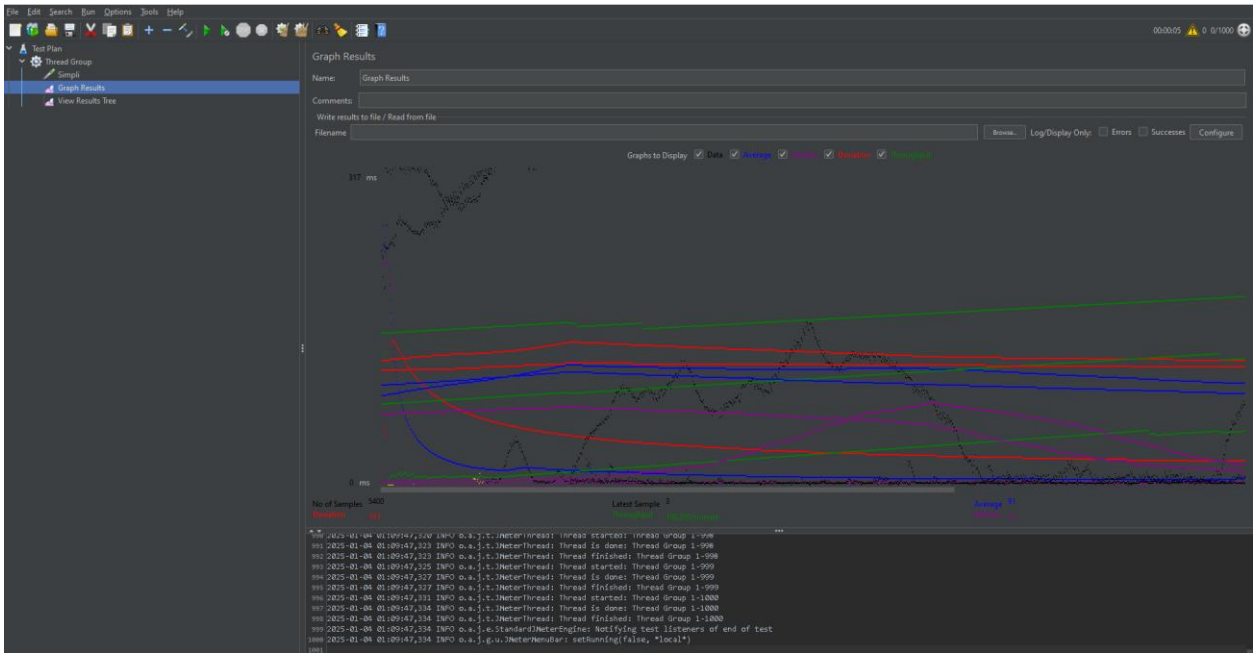
10 користувачів



100 користувачів



1000 користувачів



ДОДАТОК 2

Модульне та Selenium тестування

НТУУ «КПІ» ІАТЕ ІПЗЕ

Листів 18

Київ – 2024

Selenium_tests.py:

```
import time
import pytest
import re
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service as ChromeService
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as ec


# Фікстура для створення драйвера
@pytest.fixture
def driver():
    driver = webdriver.Chrome(service=ChromeService())
    yield driver
    driver.quit()


# Тест для перевірки завантаження сторінки
def test_page_load(driver):
    driver.get("http://127.0.0.1:5000")  # Замініть на URL вашого додатку
    assert "Система контролю вентиляцією" in driver.title
    header = driver.find_element(By.TAG_NAME, "h1")
    assert header.text == "Система контролю вентиляцією"


# Тест для перевірки генерації даних
def test_info_data(driver):
    driver.get("http://127.0.0.1:5000")

    # Очікуємо оновлення сенсорних даних
    temperature_value = driver.find_element(By.ID, "temperature-value").text
    assert "°C" in temperature_value

    pollution_value = driver.find_element(By.ID, "pollution-value").text
    assert "µg/m³" in pollution_value

    humidity_value = driver.find_element(By.ID, "humidity-value").text
```

```
    assert "%" in humidity_value


def test_get_data(driver):
    driver.get("http://127.0.0.1:5000")
    button = driver.find_element(By.XPATH,
"//button[contains(text(),'Ввімкнути')]")
    button.click()

    # Очікуємо оновлення сенсорних даних
    wait = WebDriverWait(driver, 1)

    # Перевірка для температури
    temperature_value = wait.until(
        ec.presence_of_element_located((By.CSS_SELECTOR, "#temperature-value
.number"))
    ).text
    assert re.match(r"^-?\d+(\.\d+)?$", temperature_value), f"Temperature value is
not a number: {temperature_value}"

    # Перевірка для забруднення
    pollution_value = driver.find_element(By.CSS_SELECTOR, "#pollution-value
.number").text
    assert re.match(r"^-?\d+(\.\d+)?$", pollution_value), f"Pollution value is not a
number: {pollution_value}"

    # Перевірка для вологості
    humidity_value = driver.find_element(By.CSS_SELECTOR, "#humidity-value
.number").text
    assert re.match(r"^-?\d+(\.\d+)?$", humidity_value), f"Humidity value is not a
number: {humidity_value}"


# Тест для автоматичного оновлення
def test_auto_update(driver):
    driver.get("http://127.0.0.1:5000")

    # Отримуємо початкові значення
    initial_temperature_value = driver.find_element(By.CSS_SELECTOR,
```

```
"#temperature-value .number").text
    initial_pollution_value = driver.find_element(By.CSS_SELECTOR,
"#pollution-value .number").text
    initial_humidity_value = driver.find_element(By.CSS_SELECTOR,
"#humidity-value .number").text

    # Чекаємо автоматичного оновлення
    time.sleep(3)

    # Отримуємо нові значення
    updated_temperature_value = driver.find_element(By.CSS_SELECTOR,
"#temperature-value .number").text
    updated_pollution_value = driver.find_element(By.CSS_SELECTOR,
"#pollution-value .number").text
    updated_humidity_value = driver.find_element(By.CSS_SELECTOR,
"#humidity-value .number").text

    # Перевіряємо, чи значення змінилися
    assert initial_temperature_value != updated_temperature_value or
initial_pollution_value != updated_pollution_value or initial_humidity_value !=
updated_humidity_value
```

Модульне тестування:
controller_tests.py:

```
import pytest
import asyncio
from unittest.mock import MagicMock
import Controller


# Моки для DButil
@pytest.fixture
def mock_dbutil():
    mock_dbutil = MagicMock()
    mock_dbutil.get_ventilation.return_value = MagicMock()
    mock_dbutil.get_humidifier.return_value = MagicMock()
    mock_dbutil.get_air_dryer.return_value = MagicMock()
    mock_dbutil.get_conditioner.return_value = MagicMock()
```

```
    mock_dbutil.get_heating.return_value = MagicMock()
    mock_dbutil.get_termometer.return_value = MagicMock()
    mock_dbutil.get_humidity_sensor.return_value = MagicMock()
    mock_dbutil.get_pollution_sensor.return_value = MagicMock()
    return mock_dbutil


# Тест 1: Ініціалізація класу Controller
def test_controller_initialization(mock_dbutil):
    controller = Controller.Controller()

    assert controller.ventilation is not None
    assert controller.humidifier is not None
    assert controller.conditioner is not None
    assert controller.heating_system is not None
    assert controller.termometer is not None
    assert controller.humidity_sensor is not None
    assert controller.pollution_sensor is not None


# Тест 2: Перевірка методу generate_data
def test_generate_data(mock_dbutil):
    controller = Controller.Controller()

    # Створення моків для функцій, які викликаються в generate_data
    controller.termometer.generate = MagicMock()
    controller.pollution_sensor.generate = MagicMock()
    controller.humidity_sensor.generate = MagicMock()

    controller.generate_data()

    # Перевірка, що метод generate був викликаний для кожного сенсора
    controller.termometer.generate.assert_called_once()
    controller.pollution_sensor.generate.assert_called_once()
    controller.humidity_sensor.generate.assert_called_once()


# Тест 3: Перевірка методу get_stats
def test_get_stats(mock_dbutil):
```

```
    controller = Controller.Controller()

    # Моки для методів сенсорів
    controller.termometer.get_temperature = MagicMock(return_value=25)
    controller.humidity_sensor.get_humiditylevel = MagicMock(return_value=50)
    controller.pollution_sensor.get_pollutionlevel = MagicMock(return_value=10)

    stats = controller.get_stats()

    assert stats == [25, 50, 10]
```

server_tests.py:

```
import pytest
from app import app
import asyncio


pytest_plugins = ('pytest_asyncio',)


@pytest.fixture
def client():
    # Тестовий клієнт Flask-додатка
    app.config['TESTING'] = True
    with app.test_client() as client:
        yield client


def test_home_page(client):
    """Тест для домашньої сторінки."""
    response = client.get('/')
    assert response.status_code == 200
    assert "Система контролю вентиляцією" in response.get_data(as_text=True)


def test_api_generate(client):
    """Тест для ендпоінта /api/generate."""
    response = client.get('/api/generate')
```

```
    assert response.status_code == 200
    data = response.get_json()
    assert "temperature" in data
    assert "humidity" in data
    assert "pollution" in data
    assert isinstance(data["temperature"], (int, float))
    assert isinstance(data["humidity"], (int, float))
    assert isinstance(data["pollution"], (int, float))


def test_api_get_data(client):
    """Тест для ендпоінта /api/get_data."""
    response = client.get('/api/get_data')
    assert response.status_code == 200
    data = response.get_json()
    assert "temperature" in data
    assert "humidity" in data
    assert "pollution" in data
    assert isinstance(data["temperature"], (int, float))
    assert isinstance(data["humidity"], (int, float))
    assert isinstance(data["pollution"], (int, float))
```

sensor_tests.py:

```
import pytest
import asyncio
from unittest.mock import MagicMock
from random import randint

from Sensors import PollutionSensor, HumiditySensor, Termometer  # замініть на правильний шлях


@pytest.mark.asyncio
async def test_pollution_sensor_generate():
    sensor = PollutionSensor(devid=1, name="Pollution Sensor")

    # Перевірка, чи було правильно згенеровано значення забруднення
    assert 20 <= sensor.pollution <= 200
```

```
@pytest.mark.asyncio
async def test_pollution_sensor_get_pollutionlevel():
    sensor = PollutionSensor(devid=1, name="Pollution Sensor")

    # Виклик методу get_pollutionlevel
    pollution_level = sensor.get_pollutionlevel()

    # Перевірка, що значення забруднення було правильно повернуто
    assert pollution_level == sensor.pollution


@pytest.mark.asyncio
async def test_set_pollution():
    sensor = PollutionSensor(devid=1, name="Pollution Sensor")

    initial_pollution = sensor.pollution
    target_pollution = initial_pollution - 10  # Зниження рівня забруднення

    # Виклик асинхронного методу для зниження рівня забруднення
    await sensor.set_pollution(target_pollution)

    # Перевірка, що рівень забруднення зменшився
    assert sensor.pollution <= target_pollution


@pytest.mark.asyncio
async def test_humidity_sensor_generate():
    sensor = HumiditySensor(devid=1, name="Humidity Sensor")

    # Перевірка, чи було правильно згенеровано значення вологості
    assert 15 <= sensor.humidity <= 80


@pytest.mark.asyncio
async def test_humidity_sensor_get_humiditylevel():
    sensor = HumiditySensor(devid=1, name="Humidity Sensor")
```

```
    # Виклик методу get_humiditylevel
    humidity_level = sensor.get_humiditylevel()

    # Перевірка, що значення вологості було правильно повернуто
    assert humidity_level == sensor.humidity


@pytest.mark.asyncio
async def test_set_humidity():
    sensor = HumiditySensor(devid=1, name="Humidity Sensor")

    initial_humidity = sensor.humidity
    target_humidity = initial_humidity + 10  # Збільшення вологості

    # Виклик асинхронного методу для збільшення рівня вологості
    await sensor.set_humidity(target_humidity, '+')

    # Перевірка, що рівень вологості збільшився
    assert sensor.humidity >= target_humidity


@pytest.mark.asyncio
async def test_set_humidity_decrease():
    sensor = HumiditySensor(devid=1, name="Humidity Sensor")

    initial_humidity = sensor.humidity
    target_humidity = initial_humidity - 10  # Зменшення вологості

    # Виклик асинхронного методу для зменшення рівня вологості
    await sensor.set_humidity(target_humidity, '-')

    # Перевірка, що рівень вологості зменшився
    assert sensor.humidity <= target_humidity


@pytest.mark.asyncio
async def test_termometer_generate():
    sensor = Termometer(devid=1, name="Termometer")
```

```
    # Перевірка, чи було правильно згенеровано значення температури
    assert 5 <= sensor.temperature <= 50


@pytest.mark.asyncio
async def test_termometer_get_temperature():
    sensor = Termometer(devid=1, name="Termometer")

    # Виклик методу get_temperature
    temperature = sensor.get_temperature()

    # Перевірка, що значення температури було правильно повернуто
    assert temperature == sensor.temperature


@pytest.mark.asyncio
async def test_set_temperature():
    sensor = Termometer(devid=1, name="Termometer")

    initial_temperature = sensor.temperature
    target_temperature = initial_temperature + 5  # Підвищення температури

    # Виклик асинхронного методу для підвищення температури
    await sensor.set_temperature(target_temperature, '+')

    # Перевірка, що температура збільшилась
    assert sensor.temperature >= target_temperature


@pytest.mark.asyncio
async def test_set_temperature_decrease():
    sensor = Termometer(devid=1, name="Termometer")

    initial_temperature = sensor.temperature
    target_temperature = initial_temperature - 5  # Зниження температури

    # Виклик асинхронного методу для зниження температури
    await sensor.set_temperature(target_temperature, '-')
```

```
    # Перевірка, що температура зменшилась
    assert sensor.temperature <= target_temperature


@pytest.mark.asyncio
async def test_turn_on_and_off_sensor():
    sensor = Termometer(devid=1, name="Termometer")

    # Перевірка стану при включенні
    sensor.on()
    assert sensor.status == 0

    # Перевірка стану при вимиканні
    sensor.off()
    assert sensor.status == 1
```

devices_tests.py:

```
import pytest
import asyncio
from unittest.mock import AsyncMock  # Замість MagicMock
from Devices import Ventilation, Conditioner, AirHumidifier, AirDryer, Heating,
Device  # замініть на правильний шлях
from Sensors import PollutionSensor, Termometer, HumiditySensor  # також
замінити на правильний шлях


@pytest.mark.asyncio
async def test_ventilation_set_pollution():
    ventilation = Ventilation(devid=1, name="Ventilation")
    pollution_sensor = AsyncMock(spec=PollutionSensor)  # Використовуємо
AsyncMock

    # Викликаємо асинхронний метод set_pollution
    await ventilation.set_pollution(pollution_sensor, pollution=30)

    # Перевірка, чи був викликаний метод set_pollution в сенсорі
    pollution_sensor.set_pollution.assert_called_once_with(30)
```

```
    # Перевірка, чи змінилося значення pollution в вентиляції
    assert ventilation.pollution == 30


@pytest.mark.asyncio
async def test_conditioner_set_temperature():
    conditioner = Conditioner(devid=1, name="Conditioner")
    termometer = AsyncMock(spec=Termometer)  # Використовуємо AsyncMock

    # Викликаємо асинхронний метод set_temperature
    await conditioner.set_temperature(termometer, temperature=22)

    # Перевірка, чи був викликаний метод set_temperature в термометрі
    termometer.set_temperature.assert_called_once_with(22, '-')

    # Перевірка, чи змінилося значення temperature в кондиціонері
    assert conditioner.temperature == 22


@pytest.mark.asyncio
async def test_air_humidifier_set_humiditylevel():
    humidifier = AirHumidifier(devid=1, name="Air Humidifier")
    humidity_sensor = AsyncMock(spec=HumiditySensor)  # Використовуємо
AsyncMock

    # Викликаємо асинхронний метод set_humiditylevel
    await humidifier.set_humiditylevel(humidity_sensor, humidity=50)

    # Перевірка, чи був викликаний метод set_humidity в сенсорі
    humidity_sensor.set_humidity.assert_called_once_with(50, '+')

    # Перевірка, чи змінилося значення humidity в зволожувачі
    assert humidifier.humidity == 50


@pytest.mark.asyncio
async def test_air_dryer_set_humiditylevel():
    air_dryer = AirDryer(devid=1, name="Air Dryer")
    humidity_sensor = AsyncMock(spec=HumiditySensor)  # Використовуємо
```

AsyncMock

```
    # Викликаємо асинхронний метод set_humiditylevel
    await air_dryer.set_humiditylevel(humidity_sensor, humidity=30)

    # Перевірка, чи був викликаний метод set_humidity в сенсорі
    humidity_sensor.set_humidity.assert_called_once_with(30, '-')

    # Перевірка, чи змінилося значення humidity в осушувачі
    assert air_dryer.humidity == 30


@pytest.mark.asyncio
async def test_heating_set_temperature():
    heating = Heating(devid=1, name="Heating")
    termometer = AsyncMock(spec=Termometer)  # Використовуємо AsyncMock

    # Викликаємо асинхронний метод set_temperature
    await heating.set_temperature(termometer, temperature=25)

    # Перевірка, чи був викликаний метод set_temperature в термометрі
    termometer.set_temperature.assert_called_once_with(25, '+')

    # Перевірка, чи змінилося значення temperature в опаленні
    assert heating.temperature == 25


@pytest.mark.asyncio
async def test_device_on_off():
    device = Device(devid=1, name="Device")

    # Перевірка, чи правильний статус при включенні
    device.on()
    assert device.status == 0

    # Перевірка, чи правильний статус при вимкненні
    device.off()
    assert device.status == 1
```

```
@pytest.mark.asyncio
async def test_ventilation_off():
    ventilation = Ventilation(devid=1, name="Ventilation")

    # Включення вентиляції
    await ventilation.set_pollution(AsyncMock(), pollution=50)

    # Перевірка, що вентиляція включена
    assert ventilation.status == 0

    # Виклик методу off
    ventilation.off()

    # Перевірка, чи статус вентиляції змінився на вимкнений
    assert ventilation.status == 1
    assert ventilation.pollution == 0


@pytest.mark.asyncio
async def test_conditioner_off():
    conditioner = Conditioner(devid=1, name="Conditioner")

    # Включення кондиціонера
    await conditioner.set_temperature(AsyncMock(), temperature=20)

    # Перевірка, що кондиціонер включений
    assert conditioner.status == 0

    # Виклик методу off
    conditioner.off()

    # Перевірка, чи статус кондиціонера змінився на вимкнений
    assert conditioner.status == 1
    assert conditioner.temperature is None


@pytest.mark.asyncio
async def test_air_humidifier_off():
```

```
    humidifier = AirHumidifier(devid=1, name="Air Humidifier")

    # Включення зволожувача
    await humidifier.set_humiditylevel(AsyncMock(), humidity=50)

    # Перевірка, що зволожувач включений
    assert humidifier.status == 0

    # Виклик методу off
    humidifier.off()

    # Перевірка, чи статус зволожувача змінився на вимкнений
    assert humidifier.status == 1
    assert humidifier.humidity == 0


@pytest.mark.asyncio
async def test_air_dryer_off():
    air_dryer = AirDryer(devid=1, name="Air Dryer")

    # Включення осушувача
    await air_dryer.set_humiditylevel(AsyncMock(), humidity=30)

    # Перевірка, що осушувач включений
    assert air_dryer.status == 0

    # Виклик методу off
    air_dryer.off()

    # Перевірка, чи статус осушувача змінився на вимкнений
    assert air_dryer.status == 1
    assert air_dryer.humidity == 0


@pytest.mark.asyncio
async def test_heating_off():
    heating = Heating(devid=1, name="Heating")

    # Включення обігрівача
```

```
    await heating.set_temperature(AsyncMock(), temperature=22)

    # Перевірка, що обігрівач включений
    assert heating.status == 0

    # Виклик методу off
    heating.off()

    # Перевірка, чи статус обігрівача змінився на вимкнений
    assert heating.status == 1
    assert heating.temperature is None
```

db_tests.py:

```
import pytest
from unittest.mock import MagicMock, patch
import Devices
import Sensors
import DButil


@pytest.fixture
def mock_mongo():
    with patch("DButil.pymongo.MongoClient") as mock_client:
        mock_db = MagicMock()
        mock_client.return_value = mock_db
        yield mock_db


@pytest.fixture
def mock_devices(mock_mongo):
    mock_collection = mock_mongo["Ventilation"]["Devices"]
    mock_collection.find.return_value = [
        {'_id': 1, 'name': 'Device1', 'type': 'Conditioner'},
        {'_id': 2, 'name': 'Device2', 'type': 'HeatingSystem'},
        {'_id': 3, 'name': 'Device3', 'type': 'Humidifier'},
        {'_id': 4, 'name': 'Device4', 'type': 'Ventilation'},
        {'_id': 5, 'name': 'Device5', 'type': 'AirDryer'},
    ]
```

```
    return mock_collection


@pytest.fixture
def mock_sensors(mock_mongo):
    mock_collection = mock_mongo["Ventilation"]["Sensors"]
    mock_collection.find.return_value = [
        {'_id': 101, 'name': 'Sensor1', 'type': 'Termometer'},
        {'_id': 102, 'name': 'Sensor2', 'type': 'HumiditySensor'},
        {'_id': 103, 'name': 'Sensor3', 'type': 'PollutionSensor'},
    ]
    return mock_collection


def test_initialize_devices(mock_devices):
    DButil.devices = mock_devices
    DButil.conditioner = None
    DButil.heating = None
    DButil.humidifier = None
    DButil.ventilation = None
    DButil.dryer = None

    for a in mock_devices.find():
        match a['type']:
            case 'Conditioner':
                assert a['name'] == 'Device1'
                DButil.conditioner = Devices.Conditioner(a['_id'], a['name'])
            case 'HeatingSystem':
                assert a['name'] == 'Device2'
                DButil.heating = Devices.Heating(a['_id'], a['name'])
            case 'Humidifier':
                assert a['name'] == 'Device3'
                DButil.humidifier = Devices.AirHumidifier(a['_id'], a['name'])
            case 'Ventilation':
                assert a['name'] == 'Device4'
                DButil.ventilation = Devices.Ventilation(a['_id'], a['name'])
            case 'AirDryer':
                assert a['name'] == 'Device5'
                DButil.dryer = Devices.AirDryer(a['_id'], a['name'])
```

```
    assert DButil.conditioner is not None
    assert DButil.heating is not None
    assert DButil.humidifier is not None
    assert DButil.ventilation is not None
    assert DButil.dryer is not None


def test_initialize_sensors(mock_sensors):
    DButil.sensors = mock_sensors
    DButil.termometer = None
    DButil.humidity_sensor = None
    DButil.pollution_sensor = None

    for a in mock_sensors.find():
        match a['type']:
            case 'Termometer':
                assert a['name'] == 'Sensor1'
                DButil.termometer = Sensors.Termometer(a['_id'], a['name'])
            case 'HumiditySensor':
                assert a['name'] == 'Sensor2'
                DButil.humidity_sensor = Sensors.HumiditySensor(a['_id'], a['name'])
            case 'PollutionSensor':
                assert a['name'] == 'Sensor3'
                DButil.pollution_sensor = Sensors.PollutionSensor(a['_id'], a['name'])

    assert DButil.termometer is not None
    assert DButil.humidity_sensor is not None
    assert DButil.pollution_sensor is not None


def test_get_devices_and_sensors():
    assert DButil.get_conditioner() is DButil.conditioner
    assert DButil.get_heating() is DButil.heating
    assert DButil.get_humidifier() is DButil.humidifier
    assert DButil.get_ventilation() is DButil.ventilation
    assert DButil.get_air_dryer() is DButil.dryer
    assert DButil.get_termometer() is DButil.termometer
```

```
assert DButil.get_humidity_sensor() is DButil.humidity_sensor
assert DButil.get_pollution_sensor() is DButil.pollution_sensor
```