ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»
Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики та інформатики

«До захисту допущено»
Завідувач кафедри
Наталія МАСЛОВА
(підпис) (Ім'я та ПРІЗВИЩЕ)

«_____»_____2024 р.

Випускна кваліфікаційна робота

бакалавра
(освітній ступінь)

на тему: «Системи візуалізації та аналізу даних Smart Home»

зі спецчастиною: «Розробка додатку управління та аналізу даних системи Smart Home з використанням нейромереж за допомогою Python, Node.js, SQL»

Виконав: студент 4 курсу, групи ІПЗ-20
(шифр групи)

спеціальності 121 Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

освітньої програми 121 Інженерія програмного забезпечення

Мартинцов Олексій Ігорович
(прізвище та ініціали) (підпис)

Керівник доц. каф. ПМІ, к.т.н., доц. Ірина НАЗАРОВА _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Рецензент доц. каф. ЕТіКІ, к.т.н., доц. Віталій ШАМАЄВ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Луцьк – 2024р.

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики та інформатики
Освітній ступінь бакалавр
Напрям підготовки (спеціальність) 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри ПМІ
_____ Наталія МАСЛОВА
"____" _____2024 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Мартинцову Олексію Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Системи візуалізації та аналізу даних Smart Home
зі спецчастиною Розробка додатку управління та аналізу даних системи Smart Home з використанням нейромереж за допомогою Python, Node.js, SQL
Керівник роботи Назарова І. А., к.т.н., доц ., доц. каф. ПМІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом від " 25 " квітня 2024 року № 164
2. Строк подання студентом роботи "10" червня 2024 року
3. Вихідні дані до роботи результати науково-дослідної роботи, матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
1) Аналіз предметної області та постановка задачі;
2) Аналіз інструментів розробки;
3) Розробка архітектури та програмного застосунку;
4) Тестування розробки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) робота містить 15 рисунків, 4 таблиці, інший матеріал, у кількості, достатній для демонстрації результатів кваліфікаційної роботи бакалавра

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	к.т.н., доц. ПМІ Назарова І.А.		

7. Дата видачі завдання 25.04.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	25.04.2024 – 5.05.2024	
2	Постановка завдання	6.05.2024 – 8.05.2024	
3	Огляд тематики роботи, актуальність роботи, формулювання мети на задач дослідження	9.05.2024 – 20.05.2024	
4	Опис та обґрунтування основних ідей роботи	12.05.2024 – 24.05. 2024	
5	Вибір та розробка програмного забезпечення	25.05. 2024 – 26.05.2024	
6	Дослідження запропонованих ідей, аналіз результатів дослідження, формулювання висновків та тенденцій	27.05. 2024 – 03.06. 2024	
7	Оформлення пояснювальної записки	04.06. 2024 – 10.06. 2024	
8	Нормоконтроль пояснювальної записки та додаткових матеріалів	11.06. 2024 – 14.06. 2024	
9	Рецензування роботи	15.06. 2024 – 17.06. 2024	
10	Захист роботи	24.06.2024	

Студент _____ Мартинцов О.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Назарова І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Мартинцов О. І. Розробка системи візуалізації та аналізу даних Smart Home / Випускна кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення». ДВНЗ ДонНТУ, Луцьк, 2024.

Об’єктом дослідження є процес розробки системи від аналізу існуючих рішень, визначення основної ідеї до реалізації та впровадження програмного забезпечення.

Предметом дослідження є етапи розробки програмного забезпечення для системи візуалізації та аналізу даних Smart Home, з використанням веб технологій для створення ефективного та зручного для користувача додатку.

Наукова новизна: розробка підходів виявлення тенденцій, закономірностей та аномалій у подібних системах, на практиці, що потенційно дозволить зрозуміти тенденції споживання енергії користувачем, рівень температури, вологості в приміщенні, сформувати рекомендації для економії та оптимізації, отримати уявлення про безпеку помешкання та потенційні вразливі місця, відстежувати та аналізувати стан навколишнього середовища.

Практична цінність: розроблений застосунок може бути використаний у різних приміщеннях, підвищити комфорт, надати можливості для аналізу та відстеження даних користувачів та подальших рекомендацій щодо корекції конкретних параметрів розумного будинку.

Для розробки програмного забезпечення було використано засоби веб розробки HTML, CSS, мови програмування JavaScript та Python, фреймворк Node.js, статистичні пакети statistics, numpy, matplotlib. Використана мова структурованих запитів для взаємодії користувача з базами даних SQL.

Ключові слова: SMART HOME, АНАЛІЗ ДАНИХ, ВІЗУАЛІЗАЦІЯ ДАНИХ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВЕБ ТЕХНОЛОГІЇ, ЕФЕКТИВНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

ЗМІСТ

ВСТУП 7
1 ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ СИСТЕМ ВІЗУАЛІЗАЦІЇ ТА АНАЛІЗУ ДАНИХ 9
1.1 Концепція Smart Home та її особливості 9
1.2 Типи датчиків та їх характеристики 15
1.3 Методи збору та обробки даних з датчиків 17
1.4 Висновки за розділом 1 19
2 ВИБІР АПАРАТНОЇ ТА ПРОГРАМНОЇ ПЛАТФОРМИ ДЛЯ РОЗРОБКИ СИСТЕМИ 22
2.1 Основа апаратної платформи 22
2.2 Обрані інструменти для реалізації програмної частини 29
2.2.1 Частина Frontend (клієнтська) для ПЗ 30
2.2.2 Частина Backend (серверна) для ПЗ 32
2.3 Типи баз даних й їх інтеграція 33
2.4 Висновки за розділом 2 35
3 РОЗРОБКА ДОДАТКУ ВІЗУАЛІЗАЦІЇ ТА АНАЛІЗУ ДАНИХ SMART HOME 37
3.1 Архітектура й опис розробленої системи 37
3.2 Проектування бази даних 39
3.3 Розробка модуля збору та обробки даних 42
3.4 Розробка вебдодатку для візуалізації та аналізу даних 46
3.5 Розробка структури графічного інтерфейсу 49
3.6 Висновки за розділом 3 51
4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКА 53
4.1 Існуючі методи тестування 53
4.2 Тестування додатка 54
4.3 Висновки за розділом 4 60

ВИСНОВКИ 62
ПЕРЕЛІК ПОСИЛАНЬ..... 64
ДОДАТОК А Зауваження нормоконтролера 67
ДОДАТОК Б Текст програмного застосунку……....67

ВСТУП

У наш час більшість щоденних завдань стають простішими або автоматизованими, і цей тренд лише посилюється з роками. Сучасний стиль життя складно уявити без використання електроніки та технологій аналізу отриманих даних. Взагалі, з результатами спостережень, вимірювань, випробувань, з їх аналізом мають справу фахівці у всіх галузях практичної діяльності, майже в усіх областях теоретичних і практичних досліджень.

Зростаючий інтерес до автоматизованих систем, таких як Smart Home, обумовлений бажанням забезпечити собі комфорт і зручність. Окрім цього, SHS приваблюють безпекою, наявністю необхідних користувачу інструментів, якими можна користуватися віддалено.

Системи Smart Home відіграють важливу роль у підвищенні якості життя, автоматизуючи багато процесів та надаючи можливість дистанційного керування. Вони можуть виступати у ролі передових засобів взаємодії людини з її будинком, квартирою тощо, оскільки дозволяють налаштовувати та аналізувати умови за допомогою передачі інструкцій автоматичній системі, яка керує різними електричними та інженерними системами згідно з попередньо заданим алгоритмом.

Загальна тенденція суспільного розвитку, зокрема технічних систем, підкреслює необхідність впровадження додатку візуалізації та аналізу даних житла для покращення якості життя та підвищення ефективності роботи інженерних систем. Ці системи використовують сучасні електронні пристрої, програмне забезпечення та допоміжне обладнання. При цьому важливо враховувати як позитивні, так і негативні аспекти існуючих та розроблюваних систем.

У зв’язку з цим, розробка програмного додатку для візуалізації та аналізу даних є актуальним та важливим рішення для сучасної сфери програмування.

У випускній кваліфікаційній роботі розглядається розробка системи візуалізації та аналізу даних Smart Home. Основним завданням додатку є візуалізація, представлення складних наборів даних у чітких та інтуїтивно зрозумілих форматах, на різних типах графіків. Розробка алгоритму виявлення тенденцій, закономірностей та аномалій, що потенційно дозволить зрозуміти звички споживання енергії користувачем, сформувати рекомендації для економії та оптимізації, отримати уявлення про безпеку помешкання та потенційні вразливі місця, відстежувати та аналізувати стан навколишнього середовища.

Результатом дослідження є розробка програмного додатку, який буде використовуватися для аналізу та візуалізації даних, отриманих від SHS.

Отже, робота має актуальність з точки зору розкриття потенціалу розробки більш поглиблених рішень для систем розумного будинку, що зрештою призведе до більших можливостей аналізу даних отриманих з системи.

1 ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ СИСТЕМ ВІЗУАЛІЗАЦІЇ ТА АНАЛІЗУ ДАНИХ

1.1 Концепція Smart Home та її особливості

За останнє десятиліття сфера систем домашньої автоматизації (SHS) досягла великого успіху, а дослідники продовжують вдосконалювати цю технологію. За допомогою Internet of Things (IoT) стало легше підключати домашні пристрої і забезпечувати зв'язок між ними та користувачами [1].

Розумні технології, орієнтовані на задоволення потреб людини, стали рушійною силою для багатьох інноваційних ідей, таких як система «Розумний будинок» (SHS). Завдяки розвитку «розумних» продуктів і послуг відбулось зростання взаємозв'язку між пристроями та обміном інформацією, що призвело до стрімкого зростання популярності технології «розумного будинку» [2].

SHS на основі IoT стала популярним вибором серед споживачів протягом останніх років, оскільки дозволяє зручно керувати всіма домашніми пристроями одночасно через Інтернет. Усі аспекти будинку можна контролювати з будь-якої точки світу за допомогою пристроїв з підтримкою IoT.

Системи розумного будинку будуються за базовою структурою, яка включає збір інформації за допомогою датчиків, збереження та обробку цієї інформації головним процесором і генерацію результатів та надання послуг на основі обробленої інформації (рис. 1.1).

Першим кроком система SHS використовує датчики, такі як датчики руху, температури, вологості, полум'я, газу та освітленості (LDR) для збору атмосферних даних, інші пристрої (камера та мікрофон), для запису відео та аудіо мешканців будинку. Система також використовує підключені через IoT домашні пристрої для збору інформації про їх стан.

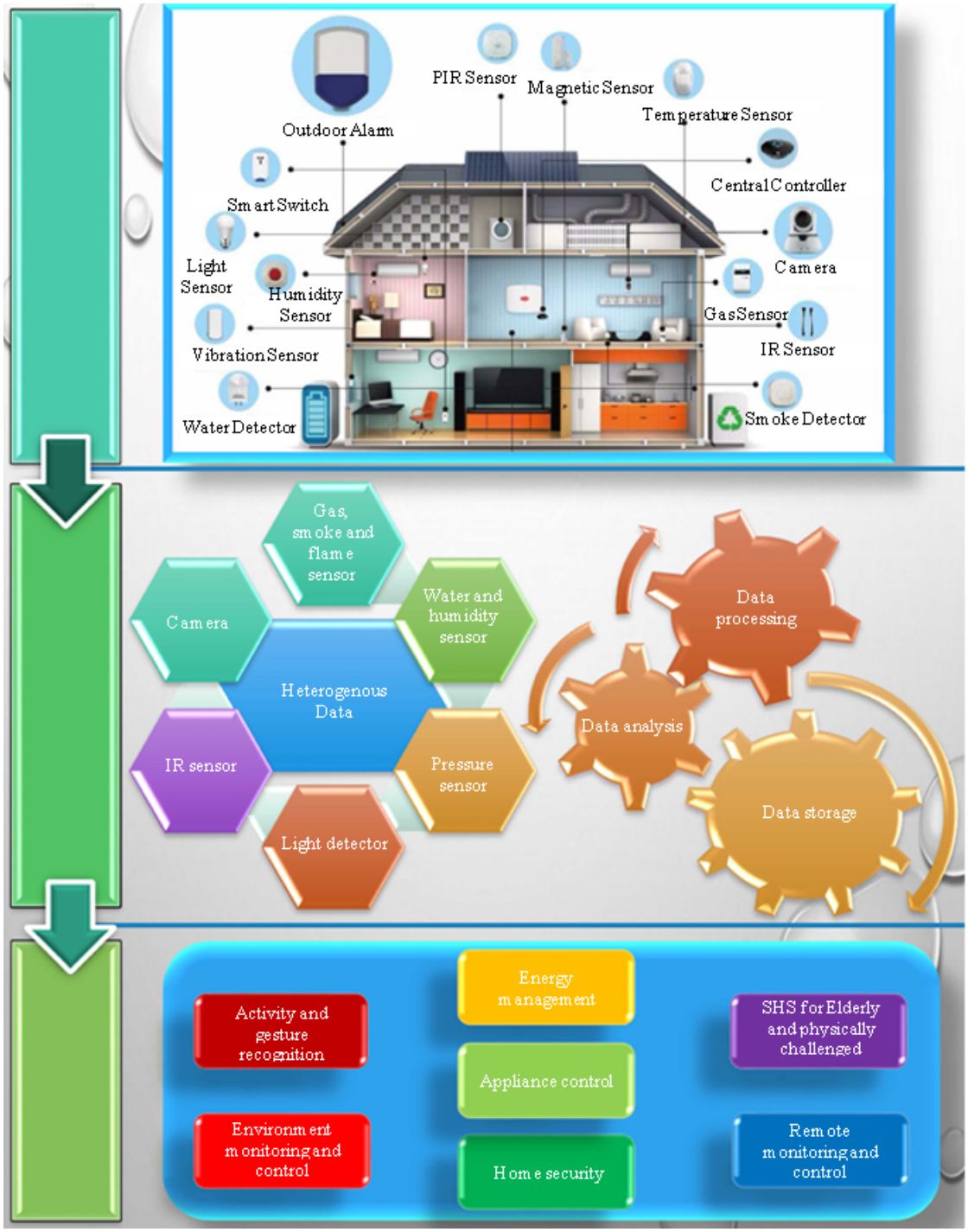

Рисунок 1.1 – Система Smart Home за джерелом [3]

Після збору інформація передається до головного блоку обробки даних дротовим або бездротовим способом. Процесор зберігає та аналізує дані, а також визначає наступну дію на основі цієї інформації. Наприклад, температура та вологість в приміщенні порівнюються із заздалегідь встановленими пороговими значеннями, і якщо поточна величина перевищує

його або навпаки, власнику надсилається повідомлення для подальших дій. Аналогічно, про кожне виявлене вторгнення (зафіксоване камерою відеоспостереження) на територію будинку/помешкання негайно оповіщаються приватні служби охорони/поліція (залежно від вибору споживача).

На останньому етапі зібрана та оброблена інформація використовується для надання різноманітних послуг, таких як комфорт в будинку, оповіщення про вторгнення, догляд за літніми людьми та керування побутовими приладами. Користувачі можуть дистанційно регулювати температуру в приміщенні та опалювати або охолоджувати будинок перед приїздом.

Аналогічно, за допомогою датчиків полум'я та газу будь-яке займання або витік газу можна миттєво виявити та вжити необхідних заходів. В сучасних системах також є можливості виконання голосових команд, жестів що дозволяють керувати приладами, надає можливостей розпізнання злочинців, за допомогою датчиків і правильного аналізу інформації з них.

Отже, система Smart Home характеризується наступними особливостями:

– автоматично виконує рутинні завдання (увімкнення/вимкнення світла, термостата, побутової техніки тощо);

– користувачі віддалено керують пристроями та даними, передивляються звіти про стан необхідних параметрів, що знаходяться вдома за допомогою смартфону, планшету або комп’ютеру;

– датчики збирають дані про температуру, освітлення, вологість, безпеку та інші параметри, після чого надсилають користувачам загальний звіт про стан будинку;

– оптимізує використання енергії завдяки рекомендаціям для зменшення споживання;

– налаштовується згідно з індивідуальними потребами та уподобаннями користувачів.

Користувацький досвід (під час взаємодії з додатком) покращується шляхом візуалізації даних у зрозумілій та привабливій формі, використовуючи різні типи інструментів. До методів візуалізації відносяться:

– інтерактивні панелі інструментів, які надають користувачам доступ до даних у режимі реального часу та дозволяють сортувати, змінювати, налаштовувати візуалізацію даних;

– графіки та діаграми для візуалізації зміни показників з часом;

– відображення геопросторових даних (розташування датчиків або енергоспоживання в різних зонах будинку) на картах;

– інфографіка відображає інформацію у стислому та доступному форматі.

Відзначимо, що існує декілька методів аналізу даних для SHS, серед них:

– аналіз даних про економію енергії;

– аналіз даних про рівень безпеки;

– прогнозування майбутніх подій (споживання енергії або необхідність техобслуговування) за допомогою предиктивного аналізу.

Розглянемо центральні процесори, контролери, що використовуються в SH. У рамках систем Smart Home процесор або контролер виконує роль мініатюрного комп'ютера, інтегрованого на одній мікросхемі.

Типова структура мікроконтролера містить один або декілька центральних процесорів (CPU), модулі пам'яті та периферійні пристрої вводу-виводу. В даному випадку мікроконтролери використовуються для наступних операцій:

– зчитування даних з датчиків;

– обробка та аналіз зібраних даних;

– виконання команд користувача;

– зберігання інформації, або відправка її на сервер чи локальне сховище.

Arduino належать до сімейства мікроконтролерів з відкритим кодом, які мають доступні ціни, гнучкі та прості у використанні. У SHS зазвичай використовуються наступні дві плати Arduino.

Arduino Uno – плата базується на мікросхемі ATmega328P та має достатню потужність для керування декількома датчиками та пристроями, завдяки цьому підходить для простих систем SHS [4-5].

Arduino Mega – плата використовує мікросхему ATmega2560, яка пропонує більшу кількість вхідних портів та кращу обробку даних порівняно з Arduino Uno [5]. Arduino Mega оптимальний вибір для складних систем SHS, що оброблять більший обсяг даних.

Основна перевага платформи Arduino – низьке енергоспоживання, а отже економічність та високий ступінь автономності при роботі від акумулятора. Окрім вищезазначеного, Arduino пропонує й інші переваги, які роблять їх цінними інструментами для розробників SHS [6]:

– програмуються за допомогою мови C++, яка на сьогодні є доволі поширеною для вивчення та використання у рамках проектів;

– доступний широкий вибір датчиків та модулів, які дозволяють з часом розширювати функціональність додатку та всієї системи;

– у Arduino протягом років сформувалась велика та активна спільнота, яка регулярно надає допомогу та підтримку молодим розробникам, наявна велика база готових проектів та рішень.

Raspberry Pi – компактний одноплатний комп'ютер, який володіє широким спектром можливостей, що робить його цінним інструментом для розробки SHS. Розглядаючи Raspberry Pi з погляду використання у SHS, зауважимо, що він збирає дані з датчиків, камер та інших пристроїв, зберігає їх у локальній пам'яті або обробляє у хмарному середовищі. Таким чином, Raspberry Pi використовується для аналітики даних та прийняття рішень на основі даних, закладених у SHS [7].

Також можна розглянути Raspberry Pi, вона керує електронними пристроями, наприклад активаторами, реле та освітленням, паралельно відстежуючи стан пристроїв. Завдяки цьому досягається автоматизація завдань. Перевагами Raspberry Pi в контексті SHS є:

– доступність порівняно з традиційними комп'ютерами;

– має модульну конструкцію, яка дозволяє розширювати функціональність за допомогою додаткових плат, модулів;

– система працює на основі платформи з відкритим кодом, що надає розробникам можливості модифікації та налаштування відповідно до потреб замовника чи потреб проекту [8].

Наступними можуть бути ESP32 та ESP8266 – мікроконтролери сімейства ESP, мають інтегровані модулі Wi-Fi та Bluetooth. Характеризуються доступністю з погляду ціни та енергоефективністю [9].

До ключових переваг ESP32/ESP8266 в даному контексті можна віднести:

– ESP32 має двоядерний процесор з тактовою частотою до 240 МГц, завдяки чому відбувається швидша обробка даних з датчиків, прийняття рішень у реальному часу всередині пристрою та інтеграція з хмарними платформами;

– ESP8266 має одне ядро Tensilica Xtensa LX100 з тактовою частотою 80 МГц, але його достатньо для використання простих завдань, зчитування даних і відправка їх на центральний вузол.

– ESP32 та ESP8266 забезпечені підтримкою сплячих режимів з мінімальним рівнем енергоспоживання. Менший рівень енергоспоживання дозволяє подовжити термін безперервної роботи системи.

Обидва пристрої мають вбудований WI-FI з підтримкою таких стандартів як 802.11 b/g/n.

ESP32 та ESP8266 підтримують різні протоколи безпеки (WPA/WPA2), для безпечного зв'язку через Wi-FI, таким чином SHS залишається захищеною

від несанкціонованого отримання доступу та інших порушень безпеки системи.

Класифікацію існуючих мікроконтролерів та мікрокомп'ютерів для подібних систем за технологічними підходами наведено у табл. 1.1.

Таблиця 1.1 – Класифікація мікроконтролерів та мікрокомп'ютерів для SHS за технологічними підходами.

Характеристика	Arduino Uno	Arduino Mega	ESP 32	ESP8266	Raspberry Pi4
Мікроконтролер	ATmega328P	ATmega2560	Tensilica Xtensa LX106	Tensilica Xtensa LX106	Broadcom BCM2837B0
Тактова частота	16 МГц	16 МГц	240 МГц	80/160 МГц	1.5 ГГц
Пам’ять	2 КБ SRAM, 32 КБ Flash	256 КБ SRAM, 256 КБ Flash	520 КБ SRAM, 8 МБ SPI Flash	32 КБ SRAM, 4 МБ SPI Flash	2 ГБ LPDDR4 SDRAM
Бездротові інтерфейси	-	-	Wi-Fi 802.11 b/g/n, Bluetooth 4.2 BLE	Wi-Fi 802.11 b/g/n	Wi-Fi 802.11 b/g/n, Bluetooth 5.0 BLE
Інші інтерфейси	SPI, I2C, UART, PWM	SPI, I2C, UART, PWM	SPI, I2C, UART, PWM	SPI, I2C, UART, PWM	HDMI, USB 2.0, Gigabit Ethernet
Споживання енергії	30 мА (в режимі очікування)	50 мА (в режимі очікування)	3.6 мА (в режимі глибокого сну)	30 мА (в режимі глибокого сну)	500 мА (в режимі очікування)
Розмір	68.6 x 53.3 x 25.4 мм	83 x 53 x 15.2 мм	28 x 25.6 x 3.2 мм	25.5 x 25.5 x 3.2 мм	85 x 56 x 17.5 мм
Вартість	Від $10	Від $20	Від $5	Від $2	Від $35
Модульність	-	-	-	-	+
Програмування	Arduino IDE, C/C++	Arduino IDE, C/C++	Arduino IDE, C/C++	Arduino IDE, C/C++	Python, C/C++, та ін.
Інтегрований Bluetooth	-	-	+	-	-
Інтегрований Wi-fi	-	-	+	+	-

1.2 Типи датчиків та їх характеристики

Датчик – це пристрій, що сприймає сигнали і зовнішні впливи і реагує на них, або сукупність вимірювальних перетворювачів, об’єднаних в один

конструктивний вузол, що виноситься на об’єкт вимірювання, або чутливий елемент або конструктивно об’єднана група чутливих елементів [10].

Наразі існують наступні типи датчиків.

Датчики температури. Пристрій збирає інформацію про температуру від джерела і перетворює її у форму, яка зрозуміла для іншого пристрою або людини. Найкращою ілюстрацією датчика температури є ртутний скляний термометр. Ртуть у склі розширюється і стискається залежно від зміни температури [11]. Це дозволяє контролювати та відстежувати параметри навколишнього середовища.

Інфрачервоні датчики. Пристрій випромінює або детектує інфрачервоне випромінювання, щоб відчувати фази навколишнього середовища. Як правило тепло випромінюється всіма об'єктами в інфрачервоному спектрі. Інфрачервоний датчик виявляє той тип випромінювання, який невидимий для людського ока. Перевагами інфрачервоного датчика є простота підключення, доступність на ринку. Недоліками вважаються зайві шуми у навколишньому середовищі, такі як навколишнє світло, випромінювання тощо [10]. Наприклад, вони можуть вмикати світло при вході людини в кімнату і вимикати його при виході.

УФ-датчики, вимірюють інтенсивність або потужність падаючого ультрафіолетового випромінювання. Форма електромагнітного випромінювання має довшу довжину хвилі, ніж рентгенівське випромінювання, але все ще коротшу, ніж видиме випромінювання. Для надійного ультрафіолетового зондування використовується активний матеріал, відомий як полікристалічний алмаз. УФ-датчики можуть виявити вплив ультрафіолетового випромінювання на навколишнє середовище [11].

Датчики присутності, вони визначають присутність людей у приміщенні і відповідно до цього регулюють освітлення, температуру або систему безпеки. Зазвичай використовуються у розумних будинках для економії енергії, вимикаючи світло або регулюючи температуру, коли в приміщенні нікого немає. Наприклад, датчик руху може використовуватися для

автоматичного ввімкнення світла, коли хтось заходить у кімнату, і вимкнення, коли він виходить.

Датчики руху – виявляють руху приміщенні та вмикають сповіщення. Використовуються в системах безпеки для виявлення зловмисників, але їх також можна використовувати і для інших цілей [12].

Датчики диму, вони використовуються для виявлення диму і запускають відповідні інструкції, щоб попередити власників будинків про потенційну пожежну небезпеку. Інтегруються у SH, щоб автоматично вимикають системи опалення і подачі газу, щоб дозволити власникам будинку безпечно евакуюватися. Наприклад, Nest Protect Smoke + Carbon Monoxide Detector може виявляти дим і чадний газ і надсилати сповіщення на смартфони власників будинків [13].

Таким чином, датчики стали невід'ємною частиною сучасних будинків та систем розумного будинку, надаючи домовласникам широкий спектр можливостей таких як контроль показників їх аналіз, удосконалення систем безпеки, автоматизація освітлення та клімат-контролю, а також забезпечення енергоефективності та комфорту в оселі.

1.3 Методи збору та обробки даних з датчиків

Збір та обробка отриманих даних з датчиків відбувається за допомогою підключення датчику до системи розумного будинку, існує як дротове (RS-485, Modbus) або бездротове (Wi-Fi, Zigbee, Bluetooth) підключення інтерфейсів. Для передачі даних з датчиків до системи можуть використовуватися різні бездротові протоколи такі як:

1. MQTT (Message Queuing Telemetry Transport) використовує модель публікування/підписки, де пристрої публікують повідомлення на теми, а інші пристрої можуть підписуватися на ці теми, щоб отримувати повідомлення. Переваги MQTT для пристроїв з обмеженими ресурсами це гарантія доставки повідомлень з мінімальними затримками, використовується шифрування та

аутентифікація. MQTT використовується у SHS для збору даних з датчиків, керування пристроями, обміном повідомленнями між пристроями, моніторингом систем.

2. HTTP (Hypertext Transfer Protocol) – універсальний протокол, який використовується для передачі даних в Інтернеті. Використовує алгоритм на запиті/відповіді, де клієнт надсилає запит серверу, а сервер надсилає відповідь клієнту. HTTP у SH застосовується для завантаження веб-сторінок, передачі файлів, доступу до веб-API, обміну даними між веб-застосунками.

3. CoAP (Constrained Application Protocol) – протокол для пристроїв, що володіють обмеженими ресурсами. Заснований на UDP (User Datagram Protocol) для передачі даних, є ефективним та безпечним і схожим на вищеописаний протокол MQTT. Використовується зазвичай для збору даних з датчиків, керування пристроями, обміну повідомленнями між пристроями та моніторингом систем (схожі сфери використання до протоколу MQTT).

Станом на сьогодні розглядається багато варіантів для обробки даних з датчиків що містять в собі статистичний аналіз, прогнозування, візуалізація на графіках діаграмах та картах. Розглянемо наступний алгоритм обробки даних [12].

Перший етап – збір даних. Інфраструктура IoT використовується для збору даних з датчиків, вбудованих у пристрої або навколишнє середовище. Датчики збирають інформацію в реальному часі, таку як температура, вологість, атмосферний тиск, рух, вібрація або будь-який інший вимірюваний параметр, залежно від системи. Дані зазвичай передаються через мережі і зберігаються для подальшого аналізу.

Другий етап – попередня обробка. Перед аналізом даних часто виконується попередня обробка. Відбувається очищення даних для видалення будь-якого шуму або викидів, які впливають на точність подальшого аналізу. Під час попередньої обробки застосовується нормалізація або масштабування даних, щоб забезпечити узгодженість і порівнянність між різними типами датчиків або одиницями вимірювання.

Третій етап – виділення ознак. Після попередньої обробки даних виділяються відповідні ознаки або закономірності з даних датчиків. Виділення ознак передбачає визначення конкретних атрибутів або характеристик, які сприяють розумінню проблеми, що розглядається. До таких належать статистичні показники (середнє значення, дисперсія тощо), або складніші, отримані за допомогою таких методів, як перетворення Фур'є або wavelet-аналіз.

Четвертий етап – статистичний аналіз. Алгоритми статистичного аналізу використовуються для вивчення закономірностей і взаємозв'язків у вилучених даних. Статистичні методи, такі як регресійний аналіз, кореляційний аналіз, факторний аналіз тощо, допомагають визначити важливі тенденції і закономірності в даних. Це дозволяє робити прогнози або класифікації на основі отриманих результатів.

П'ятий етап – обробка моделі. Обраний статистичний метод застосовується до набору даних. Під час обробки моделі налаштовуються параметри для оптимізації здатності розпізнавати закономірності і робити точні прогнози або класифікації. Процес передбачає ітеративне представлення моделі зразків даних та оновлення її параметрів на основі розрахованих помилок або різниці між прогнозованими та фактичними значеннями.

Шостий етап – висновки та прийняття рішень. Після того, як всі дані збережені та впорядковані і визначені методи аналізу, їх можна використовувати для аналізу нових, небачених даних з датчиків у режимі реального часу. Модель обробляє вхідні дані і надає результат обчислень на основі яких можна робити певні висновки.

1.4 Висновки за розділом 1

Отже можна зробити висновок що концепція Smart Home активно розвивається і розробка більш поглиблених рішень для аналізу та візуалізації

має перспективи. Таким чином потрібно врахувати деякі особливості розвитку та створення систем розумного будинку:

– протягом останнього десятиліття ця сфера досягла значного прогресу завдяки впровадженню Internet of Things (IoT);

– взаємозв'язок між пристроями та обмін інформацією значно покращився, що сприяло зростанню популярності системи Smart Home;

– система Smart Home складається з датчиків, головного процесора та різноманітних послуг, що базуються на обробленій інформації;

– використовуються різного роду датчики такі як датчики руху, температури, вологості, атмосферного тиску, напруги мережі, полум'я, газу, освітленості, енергоспоживання для збору даних;

– обробка даних здійснюється головним процесором, який аналізує дані та генерує результати для надання послуг;

– система надає можливості автоматизувати певні рутинні завдання;

– зазвичай користувачі можуть віддалено керувати пристроями та переглядати звіти про стан будинку за допомогою смартфона, планшета або комп'ютера;

– використовуються різні платформи та контролери,для збору необхідних даних такі як Arduino Uno, Arduino Mega, Raspberry Pi, ESP32 та ESP8266,вони мають відмінності за тактовою частотою, пам'яттю, бездротовими інтерфейсами та іншими характеристиками, і вибір певної платформи залежить від потреб користувача;

– збір даних з датчиків здійснюється через дротові або бездротові інтерфейси. Використовуються протоколи MQTT, HTTP, CoAP для передачі даних;

– дані проходять етапи попередньої обробки, перед тим як передати їх додатку для виділення ознак, статистичного аналізу та обробки моделі для прийняття рішень на основі отриманих результатів.

Таким чином, система Smart Home забезпечує автоматизацію та контроль різноманітних аспектів житла, що підвищує комфорт, безпеку та енергоефективність. Користувачі отримують можливість керувати всіма пристроями в будинку дистанційно, що забезпечує зручність та оптимізацію ресурсів. Використання сучасних технологій та датчиків дозволяє створювати інтелектуальні рішення, що адаптуються до потреб користувачів, забезпечуючи високу ступінь персоналізації та автономності системи.

2 ВИБІР АПАРАТНОЇ ТА ПРОГРАМНОЇ ПЛАТФОРМИ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1 Основа апаратної платформи

Апаратна платформа в сучасних системах Smart Home може бути реалізована за допомогою мікроконтролера ESP32 який є хорошим вибором для побудови системи розумного будинку з кількох причин.

ESP32 має двоядерний процесор, що дозволяє виконувати кілька завдань одночасно. Це особливо важливо для масштабних систем розумного будинку, де потрібно обробляти великі обсяги даних з різних датчиків. Можна відмітити і високу тактову частоту, процесор працює на частоті до 240 МГц, що може забезпечити високу продуктивність для різних завдань.

Мікроконтролер має вбудовані модулі Wi-Fi та Bluetooth, що робить його ідеальним для підключення до мережі інтернет та комунікації з іншими пристроями без необхідності у додаткових модулях. Це надає можливості для розгортання на його базі API, що дозволяє здійснювати віддалене управління та моніторинг пристроїв, передавати дані в реальному часі, інтегруватися з хмарними сервісами для зберігання та аналізу даних, а також створювати масштабовані системи, які легко оновлюються та налаштовуються через мережу.

Завдяки енергоефективним режимам роботи, ESP32 може працювати тривалий час від батареї, що важливо для автономних пристроїв розумного будинку.

ESP32 підтримує різноманітні інтерфейси такі як GPIO, I2C, SPI, UART для підключення датчиків і виконавчих механізмів. Це дозволяє легко інтегрувати його з різноманітними компонентами, такими як датчики температури, вологості, руху, освітлення тощо.

ESP32 підтримує розробку на платформі Arduino, що значно полегшує процес програмування і налагодження. Велика спільнота розробників створює безліч бібліотек та прикладів коду, що спрощує розробку.

Компанія Espressif надає власний набір інструментів для розробки (SDK), що забезпечує розширений доступ до можливостей ESP32.

ESP32 підтримує апаратне шифрування (AES, RSA, SHA), що забезпечує захист даних та комунікацій.

Відносно низька вартість ESP32 робить його економічно вигідним рішенням для масового використання в системах розумного будинку.

В роботі не буде приділено більшість уваги на апаратну частину, тому фізично додаток буде отримувати від мікроконтролера дані температури, вологості, атмосферного тиску (датчик BME280), а дані напруги мережі, енергоспоживання та рівня диму чи вмісту газу в повітрі будуть згенеровані на апаратній стороні задля отримання набору тестових даних та їх подальшої обробки. При необхідності до мікроконтролера можна буде підключити необхідний користувачу набір датчиків і адаптувати розроблені підходи аналізу та візуалізації отриманих даних. На рис. 2.1 зображено мікроконтролер з підключеним до нього датчиком BME280 (барометр, термометр, датчик вологості).

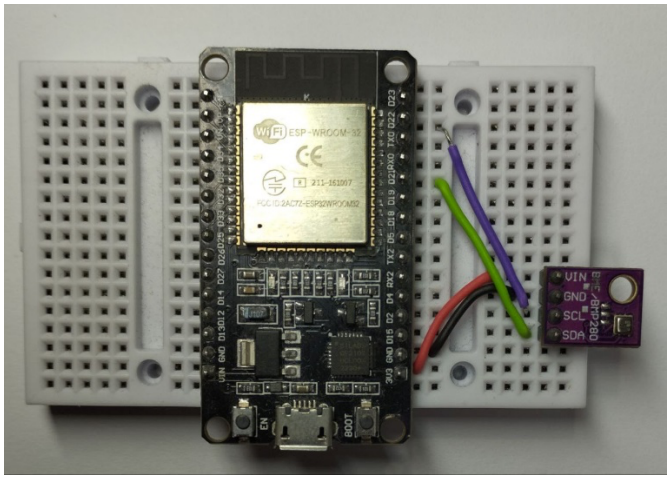

Рисунок 2.1 – Використаний мікроконтролер ESP32

Використано середовище програмування Arduino IDE – інтегрована середа розробки для Windows, macOS и GNU/Linux, розроблена на C і C++, призначена для створення та завантаження програм на Arduino-сумісні плати, а також на плати інших виробників і в тому числі мікроконтролери серії ESP [13].

Також можна продемонструвати алгоритми отримання та генерації даних з допомогою блок-схем. Сюди входять алгоритм, що генерує тестові псевдовипадкові значення для напруги (рис. 2.2), алгоритм що генерує тестові псевдовипадкові значення за експоненційним законом розподілу для рівня диму (рис. 2.3) та алгоритм що генерує тестові псевдовипадкові значення за нормальним законом розподілу використовуючи перетворення Бокса-Мюллера для енергоспоживання на рисунку(рис. 2.4).

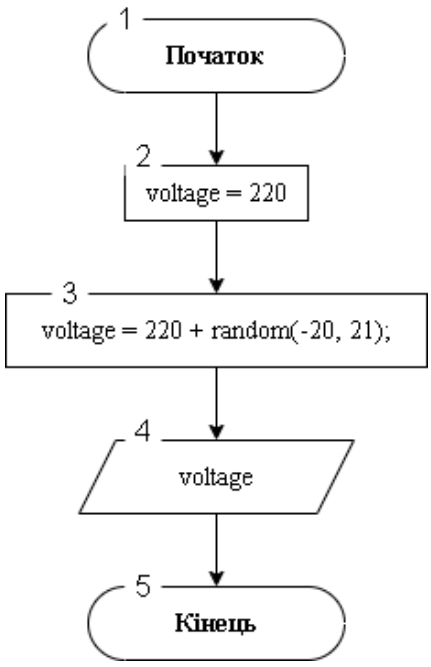

Рисунок 2.2 – Блок-схема алгоритму генерування значень напруги мережі

Для кращого розуміння роботи вище приведеного алгоритму, роздивимось його детальніше:

1) початок;

2) ініціалізація змінної напруги (voltage), присвоєння початкового значення змінній voltage рівного 220;

3) до початкового значення напруги додається псевдо випадкове значення в діапазоні від -20 до 20;

4) виведення значення напруги (voltage);

5) кінець.

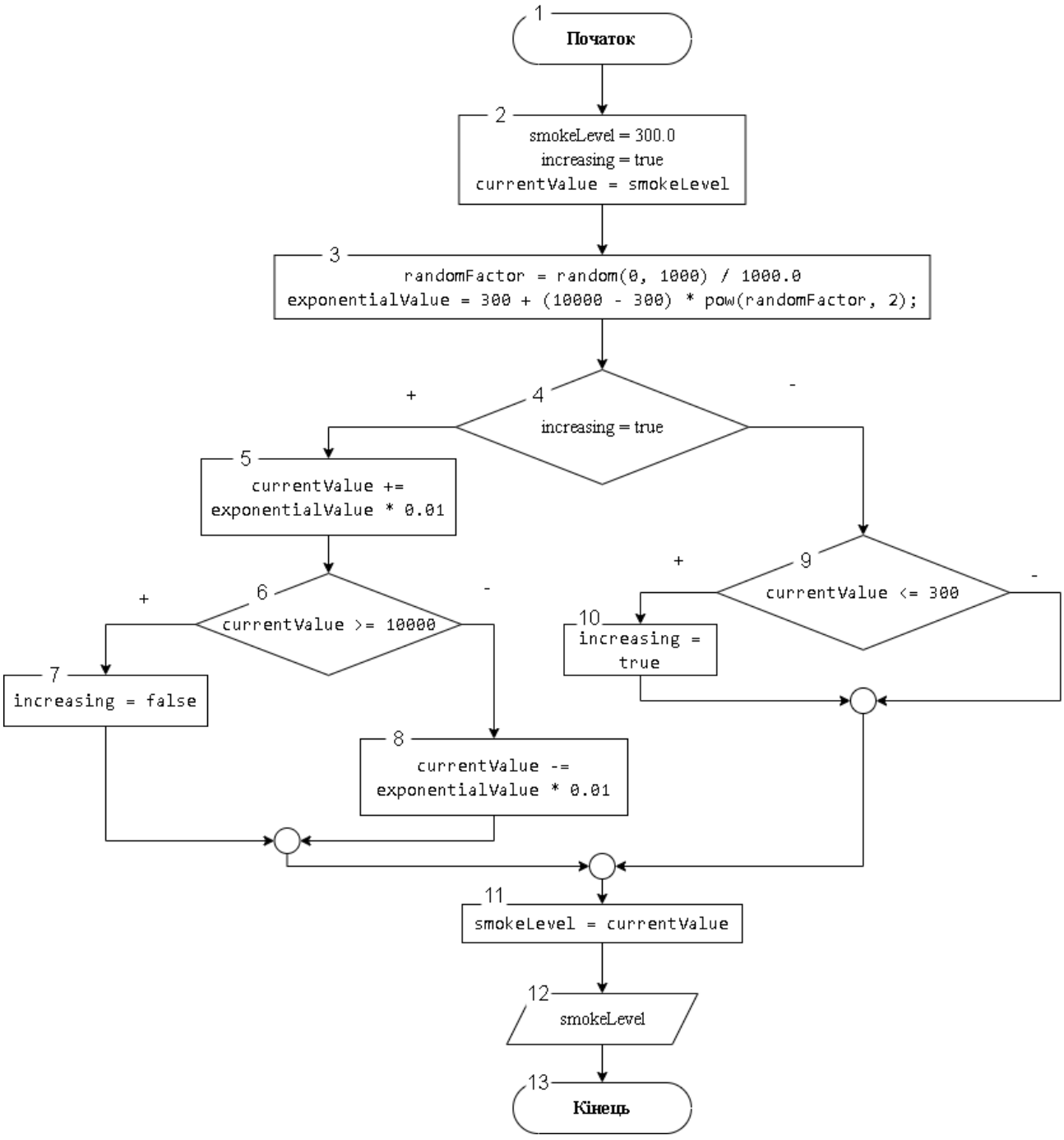

Рисунок 2.3 – Блок-схема алгоритму генерування значень рівня диму

Можна також роздивитись вище приведений алгоритм детальніше:

1) початок;

2) ініціалізація змінних та встановлення початкових значень для змінних: smokelevel = 300.0; increasing = true; currentvalue = smokelevel;

3) визначення випадкового фактора і експоненціального значення: randomfactor = random(0, 1000) / 1000.0; обчислюється експоненціальне значення: exponentialvalue = 300 + (10000 - 300) * pow(randomfactor, 2);

4) перевірка стану increasing: якщо increasing = true, переходимо до кроку 5, якщо increasing = false, переходимо до кроку 9;

5) збільшення поточного значення диму та currentvalue, currentvalue += exponentialvalue * 0.01;

6) перевірка, чи досягнуто максимального значення. якщо currentvalue >= 10000, переходимо до кроку 7, якщо currentvalue < 10000, повертаємося до кроку 3;

7) зміна напрямку на зменшення, встановлюємо increasing = false та зменшуємо currentvalue;

8) зменшення поточного значення диму, currentvalue -= exponentialvalue * 0.01;

9) перевірка, чи досягнуто мінімального значення: якщо currentvalue <= 300, переходимо до кроку 10;

10) змінюємо напрям на збільшення та встановлюємо increasing = true;

11) присвоєння поточного значення змінної smokelevel = currentvalue;

12) оновлення значення smokelevel;

13) кінець, завершення виконання алгоритму.

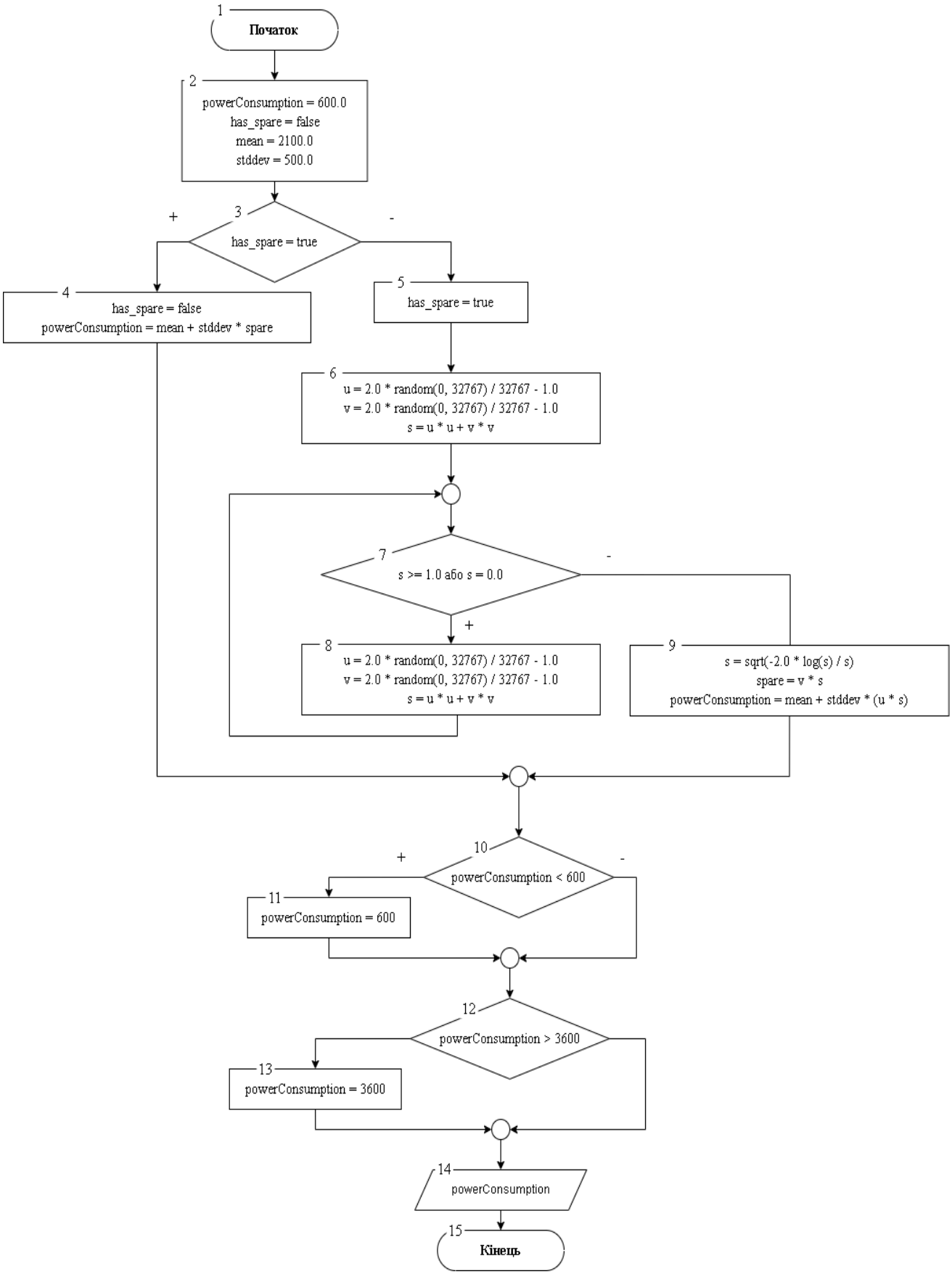

Рисунок 2.4 – Блок-схема алгоритму генерування значень енергоспоживання

Можна також роздивитись вище приведений алгоритм детальніше:

1) початок;

2) ініціалізація змінних та встановлення початкових значень для змінних: powerConsumption = 600.0; has_spare = false; mean = 2100.0; stddev = 500.0;

3) перевірка наявності запасного значення (spare);

4) якщо has_spare = true, переходимо до кроку 4, якщо has_spare = false, переходимо до кроку 5;

5) визначення нового значення powerConsumption за допомогою середнього (mean) та стандартного відхилення (stddev): powerConsumption = mean + stddev * spare, встановлення флагу has_spare = false;

6) встановлення запасного значення has_spare = true;

7) генерація двох випадкових чисел: u = 2.0 * random(0, 32767) / 32767 - 1.0; v = 2.0 * random(0, 32767) / 32767 - 1.0; s = u * u + v * v;

8) якщо s < 1.0, переходимо до кроку 9, якщо s >= 1.0 або s = 0.0, переходимо до кроку 8 генеруємо нові значення та повертаємося назад;

9) генерація нових випадкових чисел: u = 2.0 * random(0, 32767) / 32767 - 1.0; v = 2.0 * random(0, 32767) / 32767 - 1.0; s = u * u + v * v (повторний крок), якщо умова s >= 1.0 або s = 0.0 була виконана;

10) обчислення запасного значення: s = sqrt(-2.0 * log(s) / s); spare = v * s; powerConsumption = mean + stddev * (u * s);

11) перевірка, якщо powerConsumption < 600, встановлюємо powerConsumption = 600;

12) встановлення мінімального значення: powerConsumption = 600;

13) перевірка умови, якщо powerConsumption > 3600, встановлюємо powerConsumption = 3600;

14) встановлення максимального значення: powerConsumption = 3600;

15) виведення значення powerConsumption;

16) кінець, завершення виконання алгоритму.

2.2 Обрані інструменти для реалізації програмної частини

Для реалізації програмної частини, створення вебдодатка для візуалізації та аналізу даних Smart Home важливим чинником є вибір необхідних інструментів так як від цього залежить якість, ефективність та швидкість розробки.

Був використаний редактор VSCodium який є безкоштовним та відкритим редактором коду, що пропонує численні розширення та плагіни, які дозволяють налаштувати функціональність під конкретні потреби розробника. Цей редактор надає розробникам потужні інструменти для редагування коду, включаючи підсвічування синтаксису, автодоповнення, вбудовані інструменти для роботи з Git та багато інших. Також даний редактор підтримує інтеграції з різноманітними сервісами розробки, такими як GitHab, Docker та інші, що надає більші можливості для розробника та потенційно надає великі можливості до масштабованості розроблюваного програмного додатку, в цьому також допоможе і велика спільнота як розробників самого редактору так і користувачів що забезпечує оновлення, підтримку та стабільність функціоналу [14].

Програмна реалізація додатку для візуалізації та аналізу даних Smart Home буде містити фреймворк (фреймворк від англ. framework це програмна платформа, яка надає розробникам набір інструментів, бібліотек та практик для спрощення та пришвидшення розробки програмного забезпечення. Він здебільшого включає готові компоненти, структурні елементи та шаблони для вирішення типових завдань, що дозволяє розробникам зосередитись та приділити більше уваги на специфічних аспектах проекту) Node.js, Flask та Jinja2 для створення веб інтерфейсу для додатку.

Для платформи IoT яка зчитує дані та надає їх при запросі в вигляді рядків json було використано Arduino IDE. Візуалізація даних реалізовано з допомогою chart.js, а аналіз даних за допомогою пакетів для різного роду математичних обчислень Pandas, NumPy, statistics, matplotlib, scipy.

Необхідно зазначити і те, що для розробки були використані мови програмування Python та JavaSkript.

Наразі мова програмування Python доволі популярна і це обумовлено тим, що вона здатна вирішувати широке коло завдань і її застосування можливе на великій кількості популярних на сьогоднішній день платформах. Також Python здобуває все більше поширення в проектах за рахунок ефективності в Machine Learning та Data Science завдяки своїй розширюваності та гнучкості, що так необхідно в даному сегменті програмного забезпечення. У той же час Python широко застосовується в WEB, так як вирішує багато завдань на серверній стороні додатків і розробляються велика кількість фреймворків та розширень для цих завдань [15].

2.2.1 Частина Frontend (клієнтська) для ПЗ

Фронтенд (від англ. Frontend) – це частина додатку, яка відповідає за взаємодію з користувачем, відображення інформації та обробку подій на стороні клієнта. Для розробки клієнтської частини додатка було використано: HTML, CSS та JS для створення веб-інтерфейсу додатку. Розберемо їх детальніше.

В першу чергу HTML (від англ. HyperText Markup Language) – стандартна мова для створення вебсторінок і вебдодатків. Він описує структуру сторінки засобами серії елементів і тегів. HTML визначає структуру та макет сторінки за допомогою різних елементів для представлення заголовків, абзаців, зображень та посилання. Він забезпечує основу, на якій далі надбудовуються CSS і JavaScript для покращення зовнішнього вигляду та функціональності програмного забезпечення.

Також можна розглянути CSS – це мова таблиць стилів, яка створена для опису представлення документа HTML. Вона контролює макет, кольори, шрифти та загальний візуальний вигляд сторінки. CSS стилізує вміст HTML, керуючи представленням, включаючи макет, кольори, шрифти та інтервали.

Це надає можливості створювати вебдодатки візуально привабливі та адаптовані до різних розмірів екрана та пристроїв, покращуючи взаємодію з користувачем.

І нарешті JavaScript – це інтерпретована мова програмування високого рівня, яка забезпечує інтерактивність для сторінки і не тільки. Це дозволяє створювати динамічний вміст, керувати медіа контентом та файлами й оновлювати певні елементи в реальному часі. Це дозволяє оновлювати дані в режимі реального часу, перевіряти введені користувачем дані, змінювати вміст без перезавантаження сторінки, створювати складні функції, такі як анімація, перевірка форм та інтерактивні карти, виконувати певні завдання, які дозволяють активувати або викликати деякі елементи за потреби [16].

Були також використані наступні бібліотеки що спростили розробку та додали нового функціоналу в додаток:

jQuery – це швидка, невелика та багатофункціональна бібліотека JavaScript. Це значно спрощує такі речі, як обхід HTML-документів і маніпуляції з ними, обробка подій, анімація та Ajax завдяки простому у використанні API, який працює в багатьох браузерах. Завдяки поєднанню універсальності та розширюваності jQuery змінив спосіб написання JavaScript мільйонами людей [17]. В додатку вона використана для спрощення маніпуляцій з DOM, обробки подій користувача, реалізації AJAX-запитів для взаємодії з сервером без перезавантаження сторінки.

Leaflet – це провідна бібліотека JavaScript з відкритим кодом для інтерактивних карт, зручних для мобільних пристроїв. Він важить лише 42 Кбайт JS і має всі функції відображення, які коли-небудь знадобляться більшості розробників. Він розроблений з урахуванням простоти, продуктивності та зручності використання, ефективно працює на всіх основних настільних і мобільних платформах, може бути розширений за допомогою багатьох плагінів, має гарний, простий у використанні та добре задокументований API, а також простий, читабельний вихідний код [18].

В додатку бібліотека використана для інтеграції інтерактивних карт, що дозволяє візуально відображати географічні дані, для вибору конкретного місця, визначення його координат та передачі їх в подальшому в сервіс отримання даних про погоду.

Chart.js – це проста у використанні бібліотека JavaScript для створення різноманітних інтерактивних графіків і діаграм. Вона підтримує різні типи графіків, включаючи лінійні, стовпчасті, кругові, радіальні тощо [19]. В додатку вона використовується для візуалізації даних датчиків отриманих з бази даних у вигляді графіків і діаграм, що дозволяє користувачам бачити різні параметри у зрозумілому вигляді задовольняючи задачі візуалізації.

Отже, разом вище перераховані засоби утворюють основу веб-розробки, дозволяючи розробникам створювати структуровані, візуально привабливі та інтерактивні веб-сторінки та програмні додатки.

2.2.2 Частина Backend (серверна) для ПЗ

Бекенд (від англ. Backend) – це розробка бізнес-логіки вебдодатку, він відповідає за взаємодію користувача з даними, які потім відображається на клієнтській стороні. Ця частина реалізована за допомогою мови Python, котра має широкий спектр використання, але в останній час вона популярна завдяки своїм можливостям створення програм для аналітики і аналізу даних та середовищ для власних проектів на базі віртуальних машин. Node.js та Flask буде використано для розробки та налагодження серверної частини додатку.

Сервіс за допомогою якого реалізовано отримання даних про погоду це OpenWeather, він надає доступ до даних про погоду через API. Ці дані включають поточні погодні умови, прогнози, історичні дані, кліматичні зміни та іншу інформацію, пов'язану з погодою [20]. В додатку використаний для відображення погодних умов в окремій вкладці на інформаційній панелі, щоб користувачі могли легко бачити поточні погодні умови та необхідні параметри. OpenWeather API в свою чергу це вебсервіс, який надає доступ до

даних про погоду через REST API. Для отримання даних потрібно виконувати HTTP-запити до відповідних кінцевих точок API з використанням особистого ключа API для автентифікації.

Як було сказано вище, в роботі також будуть використані бібліотеки Statistics та SciPy. Модуль Statistics полегшує розрахунки таких величин як середнє арифметичне, медіана, середнє квадратичне відхилення тощо. SciPy ж спрямований на полегшення роботи з розподілами ймовірностей, дозволяє однією функцією розрахувати безперервну випадкову величину Стьюдента t, критичне значення хі-квадрат тощо. Такі бібліотеки значно полегшують роботу із статистичними функціями та роблять програмний код продукту «чистішим», тобто легшим для написання та читання, що важливо в умовах роботи з іншими людьми, публікації або демонстрації такого коду.

Важливим є також правильна і зрозуміла графічна ілюстрація тих чи інших обчислень. Оскільки в цій роботі будуть потрібні графіки та гістограми, буде використано бібліотеку Matplotlib. Вона має об'єктно-орієнтований інтерфейс, тобто користувач безпосередньо взаємодіє з кожним об'єктом. За допомогою коду можна задавати будь-який елемент діаграми, у тому числі ярлики та позначки на осях, що є дуже важливим аспектом для достовірних та точних ілюстрацій.

2.3 Типи баз даних й їх інтеграція

Вибір типу бази даних для додатку візуалізації та аналізу даних є важливим етапом, який значно впливає на ефективність та масштабованість проекту. Станом на сьогодні найбільш поширеними є реляційні бази даних (RDBMS), стандартизований тип бази даних, який зберігає дані у вигляді таблиць з взаємопов'язаними рядками та стовпцями. RDBMS підходять для структурованих даних, таких як дані про датчики, історію подій та інформацію про користувачів. Є і альтернатива, NoSQL бази даних, вони в свою чергу не мають жорсткої схеми та систематизації даних і зберігають неструктуровані

або напівструктуровані дані, такі як текстові звіти, зображення та відео. Такі бази даних підходять для гнучких даних, яким властива змінність з часом. Часовими базами даних є спеціально оптимізовані для зберігання та аналізу часові ряди даних.

Системи Smart Home генерують великий обсяг даних з датчиків, пристроїв та користувачів. До таких даних належать:

1) дані з датчиків (температура, вологість, освітлення, рух, CO2, якість повітря, параметри конкретних пристроїв типу відкриття/закриття дверей, рівень води тощо);

2) дані про пристрої (статус пристроїв, наприклад, увімкнено чи вимкнено, рівень заряду батареї, налаштування пристроїв у реальному часі);

3) дані користувачів (збережені налаштування системи, статус користувача, наприклад, власник тощо).

Бази даних для зберігання даних Smart Home повинні відповідати таким вимогам, як здатність зберігати великі обсяги даних, які генеруються SHS, тобто масштабованість. Ефективність для швидкого записування, зчитування та оновлення даних. Гнучкість для зберігання різних типів даних. Надійність необхідна для забезпечення безпеки та цілісності даних. І головне, безпека для захисту даних від несанкціонованого доступу.

Існує кілька варіантів зберігання даних з датчиків в системі IoT, зокрема.

Реляційні бази даних – традиційні бази даних, які зберігають дані в таблицях з рядками і стовпчиками, підходять для структурованих даних. Наприклад: MySQL, PostgreSQL, Oracle та інші.

Бази даних NoSQL – нереляційні бази даних, які можуть обробляти великі обсяги неструктурованих даних. Наприклад: MongoDB, Cassandra, RavenDB та інші.

Бази даних часових рядів – спеціалізовані бази даних, розроблені спеціально для обробки даних з позначкою часу, наприклад, даних з датчиків. Приклади включають InfluxDB і TimescaleDB.

Багато систем IoT використовують для зберігання даних хмарні сховища, такі як Amazon S3, Microsoft Azure Blob Storage або Google Cloud Storage.

Граничні бази даних – новий тип баз даних, які зберігають дані на периферійному пристрої. Наприклад, SQLite, RocksDB тощо [21].

Зрештою, вибір бази даних залежить від конкретних вимог системи IoT, таких як тип і обсяг даних, що збираються, необхідна продуктивність запитів, а також потреба в масштабованості і відмовостійкості.

2.4 Висновки за розділом 2

Таким чином, розділ був присвячений вибору апаратної та програмної платформи для реалізації вебдодатку. В якості середовища розробки був використаний VSCodium, потужний та універсальний редактор коду, який забезпечує розробникам ефективні та продуктивні умови для роботи над їхніми проектами, пропонуючи велику кількість функцій та можливостей. Основою апаратної платформи було обрано мікроконтролер ESP32, який завдяки своїй поширеності, вбудованим модулям Wi-Fi та Bluetooth, енергоефективності та гнучкості у підключенні різних датчиків і виконавчих механізмів є ідеальним рішенням для розробки подібних систем.

Для програмної реалізації було використано сучасні інструменти та мови програмування, такі як Python та JavaScript. Для розробки вебдодатка, який забезпечує візуалізацію та аналіз даних, було також обрано фреймворки Node.js, Flask та Jinja2. Візуалізація даних реалізована з використанням бібліотеки chart.js, а аналіз даних за допомогою Pandas, NumPy, statistics, matplotlib та scipy.

Важливим аспектом було також обрання відповідної бази даних. Було розглянуто кілька варіантів, включаючи як реляційні бази даних так і не реляційні, NoSQL бази даних. Вибір конкретної бази даних залежить від типу і обсягу даних, які потрібно зберігати, необхідної продуктивності запитів,

масштабованості та відмовостійкості. Вибір зупинився на базі даних SQLite так як це оптимальний варіант для розроблюваного додатку на даному етапі. За рахунок обраних засобів розробки додаток візуалізації та аналізу даних потенційно може розширити можливості систем розумного будинку шляхом більш поглибленого аналізу даних та представленню їх користувачу.

3 РОЗРОБКА ДОДАТКУ ВІЗУАЛІЗАЦІЇ ТА АНАЛІЗУ ДАНИХ SMART HOME

3.1 Архітектура й опис розробленої системи

Правильний та обґрунтований вибір архітектури розроблюваного додатка є важливим чинником для існування та роботи ПЗ з декількох основних причин.

Оптимізована архітектура дозволяє додатку рости та обробляти збільшення навантаження без значних змін у кодовій базі. Це особливо важливо для додатків, які можуть зіткнутися зі швидким зростанням кількості користувачів або обсягу даних.

Ефективна архітектура допомагає забезпечити високу продуктивність додатка, знижуючи час відгуку та забезпечуючи швидку обробку даних.

Добре продумана архітектура сприяє створенню надійних додатків, які можуть працювати без збоїв протягом тривалого часу. Це включає управління помилками та можливість швидкого відновлення після збоїв. Також сюди можна віднести те, що правильна архітектура додатка дозволяє легко модифікувати та впроваджувати нові функції або зміни в існуючий код. Це дозволяє швидко адаптуватися до змін у вимогах бізнесу або технологій.

Не менш важливим чинником є безпека, правильна архітектура включає механізми забезпечення безпеки, які допомагають захистити дані та функціональність додатка від різних видів атак та вразливостей.

Також завдяки чіткій структурі та використанню повторно використовуваних компонентів, розробка нового функціоналу може здійснюватися швидше та з меншими зусиллями [22].

Таким чином, правильний вибір архітектури забезпечує основу для створення ефективного, надійного та зручного у підтримці програмного

забезпечення, що може адаптуватися до майбутніх змін та вимог. На рис. 3.1 зображено архітектуру вебдодатка.

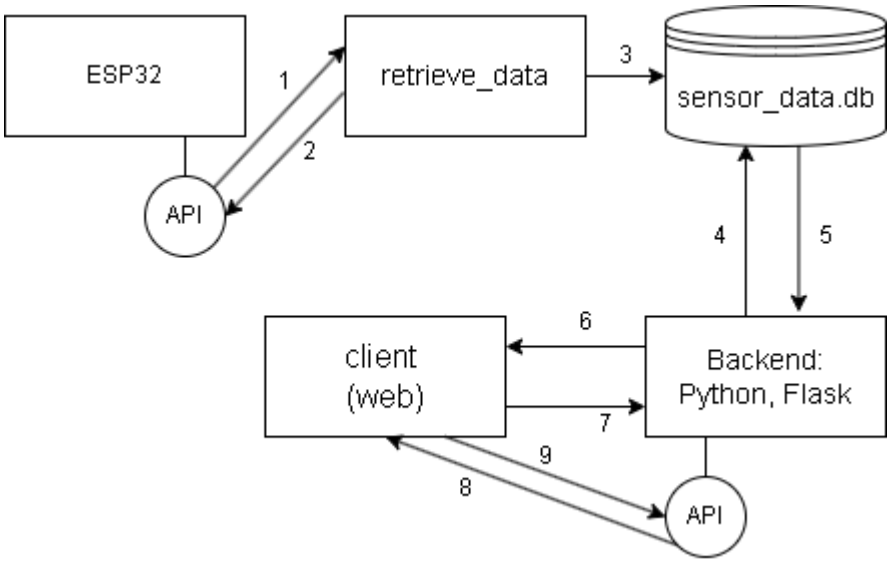

Рисунок 3.1 – Архітектура створеного вебдодатка

Як видно з вище наведеного рисунку, додаток має більше мікросервісну архітектуру ніж монолітну.

А мікросервісна архітектура (Microservices Architecture) в свою чергу це підхід до розробки програмного забезпечення, в якому додаток складається з невеликих незалежних компонентів, які називаються мікросервісами. Кожен мікросервіс обробляє один або декілька пов’язаних запитів або функцій, і може бути розроблений, випробуваний та розгорнутий незалежно від інших мікросервісів [23].

Отже зображена діаграма ілюструє архітектуру системи, що складається з мікроконтролера ESP32 (виконує задачі збору даних), веб-клієнта, бекенду на Python з використанням Flask, бази даних для збереження даних сенсорів та API для взаємодії між компонентами. Розглянемо ці компоненти та їх ролі окремо.

ESP32 – мікроконтролер ESP32 збирає дані з сенсорів та генерує їх і надсилає до API у json форматі при відповідному запиті retrieve_data з певними часовими проміжками.

retrieve_data – цей компонент відправляє запит на вказану IP адресу для отримання та збереження даних від ESP32 за допомогою SQL запросу в базу даних sensor_data.db.

sensor_data.db – база даних для збереження даних з сенсорів.

Бекенд (Python, Flask) – отримує дані з бази даних для обробки, аналізу та відправки клієнту. Бекенд на основі Flask відповідає за обробку запитів від клієнтів та взаємодію з базою даних. Вебклієнт надсилає запити до бекенду для отримання або відправки даних на серверну частину і вона при необхідності надсилає відповіді на запити клієнта, надаючи необхідну структуровану інформацію.

client (web) – вебклієнт є інтерфейсом користувача, через який здійснюється взаємодія з системою він може безпосередньо звертатися до API для отримання або відправки даних. API надсилає відповіді на запити вебклієнта, забезпечуючи необхідну інформацію.

API – слугує інтерфейсом для взаємодію між компонентами ESP32, веб-клієнтом та бекендом.

3.2 Проектування бази даних

Від вибору бази даних і інструментів для роботи з нею також залежить успіх програмного продукту. На даний момент у світі налічується багато варіантів баз даних, як реляційних, так і нереляційних. База даних визначає й спосіб доступу до даних, а не тільки як вони будуть зберігатися та структуруватися, що робить її вибір важливим аспектом розробки програмного забезпечення.

Як термін проектування бази даних можна визначити як процес створення її структури для підтримки функціонування економічного об'єкта та сприяння досягненню його цілей. Це трудомісткий процес, що потребує зусиль аналітиків, проектувальників та користувачів [24]. На початку проектування БД, зазвичай, створюється модель предметної області, для якої

вона розробляється. На цьому етапі вказуються типи об'єктів, що будуть міститися в базі даних, і зв'язки між ними.

При роботі з БД можна зіштовхнутися з багатьма труднощами та необхідно враховувати та дотримуватися кількох важливих вимог:

– вимога дотримання цілісності бд та її записів що в подальшому гарантує те, що дані не будуть втрачені чи пошкоджені;

– забезпечення вимоги багаторазового використання даних створюючи можливість використовувати дані в різних додатках і процесах;

– дотримання оптимальної архітектури та взаємодії з БД для швидкого пошуку, оновлення, видалення даних і отримання інформації за запитами користувачів що є важливим чинником для забезпечення високої продуктивності вебдодатка;

– дотримання кібербезпеки забезпечуючи конфіденційності та безпеку користування даними.

Не менш важливими є і поняття таблиці, поля, первинного ключа, нормалізації, запису таблиці, сутності. Розглянемо їх детальніше:

Таблиця – основна структура для зберігання даних у реляційній базі даних, складається з записів і полів. Кожна таблиця має унікальну назву і представляє певну сутність, наприклад, користувачів або замовлення.

Поле – окремий стовпець у таблиці, що зберігає певний тип даних. Наприклад, поле "Ім'я" у таблиці "Користувачі".

Первинний ключ – унікальний ідентифікатор для кожного запису в таблиці, не може містити null значень і забезпечує унікальність кожного запису.

Нормалізація – процес організації даних у базі даних для мінімізації надлишковості і запобігання аномаліям оновлення.

Запис – сукупність полів, що представляє одиничний екземпляр сутності в таблиці. Наприклад, один користувач у таблиці "Користувачі".

Зовнішній ключ – поле або набір полів, що посилається на первинний ключ в іншій таблиці, використовується для створення зв'язків між таблицями.

Сутність – об'єкт, що має значення для предметної області, інформація про який підлягає зберіганню, з унікальним ідентифікатором. Кожен екземпляр сутності повинен однозначно ідентифікуватися і відрізнятися від інших екземплярів даного типу сутності [24].

Всі ці поняття знадобилися при розробці БД. А досвід і вміння їх застосовувати допоможуть краще зрозуміти та ефективно працювати з базами даних, забезпечуючи їхню надійність, продуктивність і безпеку.

Для реалізації в розроблюваному вебдодатку було використана реляційна система керування базами даних SQLite. Але при цьому SQLite має свої особливості. Зокрема, вона не є окремим сервером баз даних, як MySQL або PostgreSQL, а є бібліотекою, яку можна вбудувати безпосередньо в додаток. Це означає, що для роботи з SQLite не потрібно окремо встановлювати та налаштовувати сервер баз даних. Основні характеристики SQLite включають низькі системні вимоги та мінімальне використання ресурсів, що робить його відмінним вибором для мобільних застосунків та інших середовищ.

SQLite підтримує більшу частину стандарту SQL, що дозволяє виконувати запити та керувати даними. Оскільки SQLite не має окремого сервера, доступ до бази даних здійснюється через бібліотеки API різними мовами програмування, що спрощує процес взаємодії з базою даних у додатку. SQLite підтримує транзакції, що забезпечує цілісність даних та безпеку при виконанні багатьох операцій одночасно.

Завдяки своїм характеристикам, також він широко використовується в різних типах програм, включаючи мобільні застосунки, браузери та багато іншого. Архітектура SQLite включає кілька ключових компонентів, забезпечуючи просту та ефективну роботу що доцільно для даного додатку на даному етапі [25]. Для отримання розуміння структури БД розглянемо дві її таблиці, табл. 3.1 та 3.2.

Таблиця 3.1 – Таблиця sensor_data бази даних

Поле	Тип	Пояснення
id	INT	Автоматично створюване поле, первинний ключ
temperature	REAL	Поле для значень температури
pressure	REAL	Поле для значень атмосферного тиску
humidity	REAL	Поле для значень вологості
voltage	REAL	Поле для значень напруги мережі
power_consumption	REAL	Поле для значень енергоспоживання
Smoke_level	REAL	Поле для значень рівня диму
created_at	DATETIME	Поле для запису дати отримання даних

Таблиця 3.2 – Таблиця user бази даних

Поле	Тип	Пояснення
id	INT	Автоматично створюване поле, первинний ключ
username	VARCHAR(15)	Поле для даних ім’я користувача
email	VARCHAR(319)	Поле для даних електронної пошти користувача
password	TEXT	Поле для даних паролю користувача, при чому сам пароль не зберігається в БД а лише його хеш для забезпечення безпеки

Таким чином в вище наведених таблицях були приведені існуючі таблиці їх поля та типи даних що зберігаються, перша таблиця зберігає дані датчиків а друга інформацію про користувача.

3.3 Розробка модуля збору та обробки даних

Як було сказано вище система має архітектуру яка включає такі компоненти як ESP32, retrieve_data, sensor_data.db, серверну частину (бекенд), client (web), та API інтерфейси що надають можливості взаємодії різних компонентів між собою.

І саме retrieve_data виконує завдання збору даних. Цей компонент відправляє запит на вказану IP адресу для отримання та збереження даних від ESP32 за допомогою SQL запросу в базу даних sensor_data.db.

За обробку даних відповідає серверна частина виконана засобами Python та Flask. Шляхом отримання даних з бази даних для обробки, аналізу та відправки клієнту. Бекенд на основі Flask відповідає за обробку запитів від клієнтів та взаємодію з базою даних. Вебклієнт надсилає запити до бекенду для отримання або відправки даних на серверну частину і вона при необхідності надсилає відповіді на запити клієнта, надаючи необхідну структуровану інформацію.

Для побудови варіаційних рядів і зручної подальшої роботи с даними буде створено п’ять масивів:

– початкові вхідні дані;

– відсортовані дані в порядку зростання;

– відсортовані дані без дублікатів (варіанти);

– кількість спостережень кожного значення (частоти);

– відносини частот до обсягу вибірки (ймовірність).

У вікні додатку можна буде продивитись весь обсяг вибірки з відповідними характеристиками.

Для побудови інтервального ряду були розраховані x_{min} та x_{max} можна вбудованими функціями мови Python «min()» та «max()» відповідно.

Далі знайдено R – амплітуду варіювання вибірки, k – кількість часткових інтервалів та h – крок інтервального ряду.

Розрахунки проводилися за наступними формулами (3.1):

$$R = x_{max} - x_{min},$$

$$k = [1 + 3.32 \cdot \log_{10} n],$$

$$h = \frac{R}{k}. \qquad (3.1)$$

Побудова інтервального ряду відбувається за правилами, наведеними в таблиці (3.3).

Таблиця 3.3 – Правила побудови інтервального ряду

1	2	…	k
$x_{min}, x_{min} + h$	$x_{min} + h, x_{min} + 2h$	…	$x_{min} + (k-1)h, x_{max}$
m_1	m_2	…	m_k
$p_1^* = \frac{m_1}{n}$	$p_2^* = \frac{m_2}{n}$	…	$p_k^* = \frac{m_k}{n}$

В таблиці 3.3 m_i – частота попадання до і-го інтервалу, а p_i^* – відносна частота попадання до і-го інтервалу. Також слід додати стовпець середини інтервалів, накопичених частот та величини $H_i = \frac{m_i}{h*n}$.

Гістограма була побудована наступним чином: по осі OX інтервальний ряд, по осі OY величина Hi.

А полігон частот був побудований за наступним правилами: по OX відкладаються інтервали, за OУ статистична ймовірність.

Також була побудована емпірична функція розподілу і за її даними графік.

Оцінки варіаційного ряду. Оцінку значення вибіркового математичного очікування було розраховано за формулою (3.2):

$$m^* = \frac{\sum_{i=1}^{n} x_i}{n}. \quad (3.2)$$

Оцінки значень вибіркової дисперсії та виправленої вибіркової дисперсії наведено у формулах (3.3-3.4):

$$D^* = \frac{\sum_{i=1}^{n} (x_i - m^*)^2}{n} = 794.89, \quad (3.3)$$

$$\widetilde{D} = \frac{\sum_{i=1}^{n} (x_i - m^*)^2}{n-1} = 798.89. \quad (3.4)$$

Оцінки значень вибіркового середнього квадратичного відхилення та виправленого вибіркового середнього квадратичного відхилення наведено у формулах (3.5-3.6):

$$\sigma^* = \sqrt{D^*}, \tag{3.5}$$

$$\tilde{\sigma} = \sqrt{\tilde{D}}. \tag{3.6}$$

Оцінки моментів 3-го та 4-го порядків наведені у формулах (3.7-3.8):

$$\mu^3 = \frac{\sum_{i=1}^{n}(x_i - m^*)^3}{n}, \tag{3.7}$$

$$\mu^4 = \frac{\sum_{i=1}^{n}(x_i - m^*)^4}{n}. \tag{3.8}$$

Оцінку коефіцієнту асиметрії можна знайти по формулі (3.9):

$$A = \frac{\mu^3}{D^{*3}}. \tag{3.9}$$

Оцінку ексцесу наведено у формулі (3.10):

$$E = \frac{\mu^4}{D^{*2}} - 3. \tag{3.10}$$

Медіану було знайдено за допомогою бібліотеки «statistics» вбудованою функцією «median()». Оцінки вибірки можна відобразити натиснувши на відповідний пункт головного меню та вибравши необхідні дані.

Також була додана гістограма за інтервальним рядом на ній відображена теоретична функція щільності із знайденими параметрами та математичне очікування.

Для перевірки однорідності вибіркової сукупності застосовується критерій серій.

Нульова гіпотеза H_0: елементи вибірки отримані випадковим чином і незалежні. Варіаційний ряд був побудований раніше, медіана також . Кожному елементу вихідної вибірки ставиться у відповідність «+», якщо елемент більший за медіану та «-», якщо менший.

Для отриманого ряду знаків вводяться наступні позначення:

– кількість « + » $= n_1$;

– кількість « − » $= n_2$;

– число серій $=$ N .

Розрахунки критерію z наведено у формулі (3.11):

$$z = \frac{\left(N - \frac{2n_1n_2}{n_1+n_2} - 1\right)}{\sqrt{\frac{2n_1n_2(2n_1n_2 - (n_1 + n_2))}{(n_1+n_2)^2(n_1 + n_2 - 1)}}}. \quad (3.11)$$

Рівень значущості α = 0.01. Для того, щоб порахувати u для двостороннього критерію можна скористуватись функцією «norm.ppf(1-0.01/2)» бібліотеки scipy.

Тоді нульова гіпотеза H_0 може задовільнити нерівність $|z^*| < u[1 - \frac{\alpha}{2}]$. Це означає, що вибірка отримана випадковим чином і є однорідною.

Щоб перейти до вікна «Критерій серій» в додатку, в головному меню треба натиснути на відповідний пункт.

3.4 Розробка вебдодатку для візуалізації та аналізу даних

Для демонстрації всього функціоналу додатка було побудовано діаграму варіантів використання (рис. 3.2).

Діаграма варіантів використання відображає функціональну поведінку системи, зображуючи взаємодії між користувачами та системою, яку розробляють. Актори на діаграмі представляють користувачів системи, а

варіанти використання описують поведінку системи стосовно акторів. Випадок використання визначає, з точки зору акторів (користувачів), групу дій у системі, які призводять до конкретного видимого результату. На діаграмі UML вони зображуються овалами.

Відношення розширення extend використовується для показу залежності між двома прецедентами, де один прецедент може розширювати функціональність іншого прецеденту. Позначається у формі пунктирної лінії зі стрілкою, яка виходить з основного прецеденту та вказує на прецедент, що розширюється. Відношення узагальнення визначає зв'язок, коли два й більше актори мають загальні властивості, тобто взаємодіють з тією самою множиною варіантів використання однаковим чином.

Відношення включення встановлюється тільки між двома варіантами використання й вказує на те, що задана поведінка для одного варіанта використання включається як складений фрагмент у послідовність поведінки іншого варіанта використання [26]. Графічно дане відношення позначається за допомогою пунктирної стрілки, яка виходить з основного прецеденту та вказує на включений прецедент. Дана лінія позначається стереотипом include.

Інакше кажучи, діаграма варіантів використання допомагає визначити функціональні вимоги до системи та зрозуміти, які вимоги повинні бути виконані. Діаграма наочно показує, які дії доступні кожному типу користувачів.

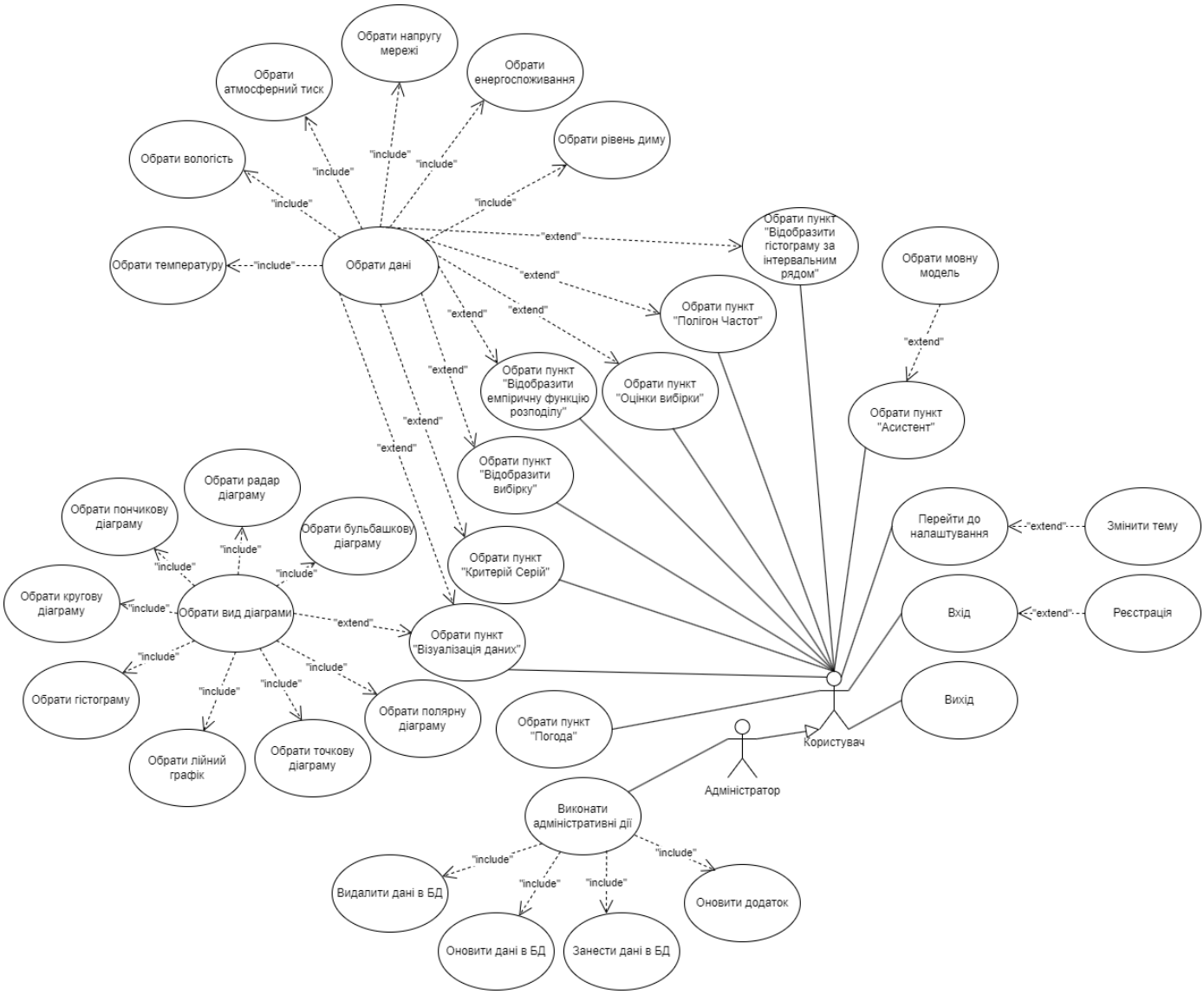

Рисунок 3.2 – Діаграма варіантів використання додатку

Таким чином користувач має можливість увійти в додаток перед цим зареєструвавши обліковий запис, в такому випадку він отримає повний функціонал додатку, за замовчуванням користувач може тільки змінити налаштування або переглянути дані про погоду певного обраного регіону.

Увійшовши в обліковий запис користувач має можливості обрати пункти меню що пов’язані з статистичним аналізом даних, такі як відобразити вибірку зі всіма даними що наявні в базі даних, відобразити оцінки вибірки, відобразити гістограму за інтервальним рядом та таблицю відповідно з цими інтервалами, відобразити полігон частот, відобразити емпіричну функцію розподілу та відобразити критерій серій.

При цьому є можливість на кожній вкладці обрати за якими даними проводити обчислення (температура, вологість, атмосферний тиск, напруга мережу, енергоспоживання та рівень диму).

Окремим пунктом меню є візуалізація даних з використанням вище описаного chartjs, дана бібліотека надає можливості будувати графіки діаграми та гістограми різних типів за даними з бази даних при цьому визначаючи шкали графіків автоматично за наявними даними. Використані лінійна діаграма, гістограма, кругова діаграма, стовпчикова діаграма, радарна діаграма, полярна також бульбашкова та точкова діаграми.

Також в якості асистента (довідника чи додаткового інструменту для аналізу даних) був інтегрований API нейромережі ChatGPT з можливістю обирати необхідні користувачу мовні моделі. Потенційно його використання може надати можливості автоматизації відповідей на запити клієнтів, створення розмовних чатів, інтерактивних освітніх платформ, генерації контенту, проведення аналітики та досліджень, розробки ігор, підтримки креативних індустрій, а також для особистого асистування і планування завдань та пошуку інформації, що підвищує ефективність, продуктивність та якість різноманітних процесів у бізнесі та розробленому додатку зокрема [27].

Також можна зазначити що адміністратор має можливості видалити, оновити та занести дані в БД потенційно оновлювати вебдодаток.

3.5 Розробка структури графічного інтерфейсу

Графічний інтерфейс користувача (від англ. Graphical User Interface) – це система, яка дозволяє користувачам взаємодіяти з комп'ютерами або іншими електронними пристроями за допомогою графічних елементів, таких як вікна, іконки, кнопки та меню. Інтерфейс робить взаємодію з пристроями інтуїтивно зрозумілою та зручною, що дозволяє користувачу виконувати складні завдання простими діями, такими як натискання кнопок або перетягуванням об'єктів.

Також, він забезпечує миттєвий зворотний зв'язок, підтримує багатозадачність і персоналізацію, знижує ймовірність помилок і робить технології доступними для ширшого кола користувачів. Таким чином, відіграє важливу роль у підвищенні продуктивності та ефективності роботи користувача з комп’ютером. Компонентами графічного інтерфейсу користувача можуть бути діалогові вікна, іконки, кнопки, меню, панелі інструментів та поля введення. Продемонструвати графічний інтерфейс додатку можна за допомогою структурних схем інтерфейсу, вони зображені на рис. 3.3 та 3.4.

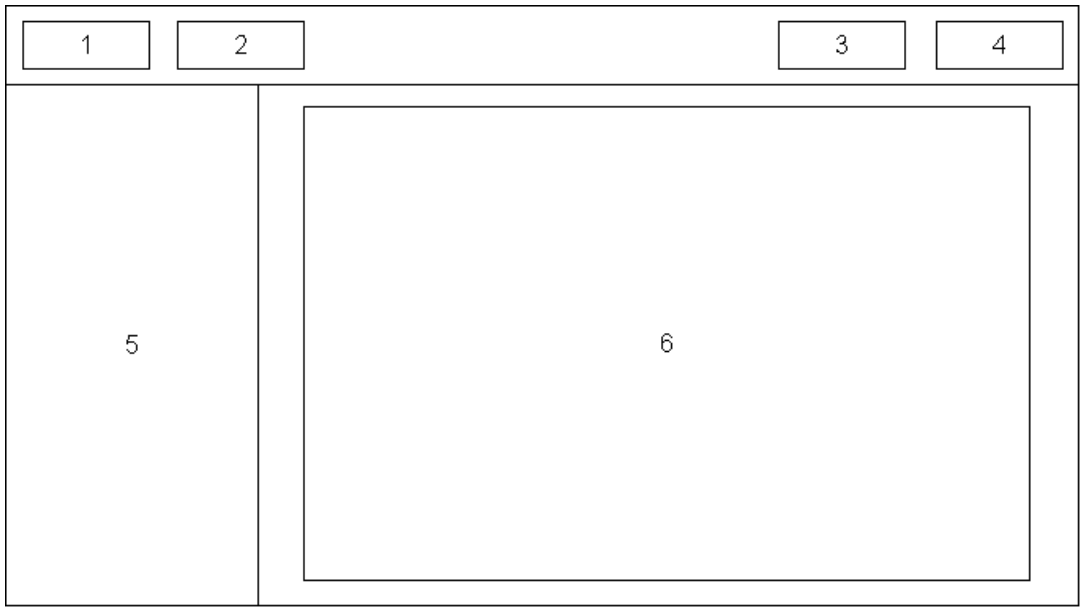

Рисунок 3.3 – Структурна схема інтерфейсу вебдодатку

Як видно з рисунку до основних елементів інтерфейсу вебдодатка належать:

1) кнопка для переходу на домашню сторінку;
2) кнопка для налаштувань додатку;
3) кнопка для реєстрації користувача;
4) кнопка для авторизації користувача;
5) бокове меню вебдодатка;

6) головний вміст вебдодатка на якому відображається основна інформація та розрахунки.

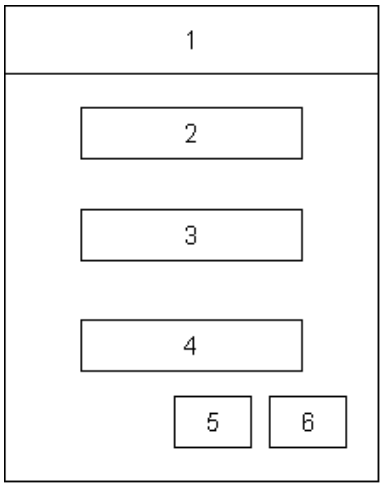

Рисунок 3.4 – Структурна схема інтерфейсу реєстрації вебдодатку

Як видно з рисунку до основних елементів інтерфейсу реєстрації вебдодатка належать:

1) заголовок сторінки реєстрації в додатку;
2) поле для вводу ім’я користувача;
3) поле для вводу пошти користувача;
4) поле для вводу паролю користувача;
5) кнопка що підтверджує реєстрацію;
6) кнопка скасувати.

Необхідно зазначити що вхід до облікового запису користувача в додаток виконується в такому самому вікні, за відмінності пошти, таким чином якщо користувач ввів пароль і ім’я правильно і має обліковий запис збережений в базі даних тоді вхід буде здійснено успішно.

3.6 Висновки за розділом 3

Отже третій розділ записки присвячений опису розробки вебдодатку. Була визначена архітектура додатка, спроектована база даних, розроблені

модулі збору та обробки даних, а також розглянутий графічний інтерфейс користувача.

Таким чином, додаток має мікросервісну архітектуру, що дозволяє легко масштабувати систему та ефективно обробляти збільшення навантаження. Архітектура включає компоненти ESP32 для збору даних, API для взаємодії між компонентами, базу даних для збереження даних сенсорів, бекенд на Python з використанням Flask та статистичних пакетів, та веб-клієнт для користувача.

Для реалізації додатку була використана реляційна база даних SQLite, яка має низькі системні вимоги та мінімальне використання ресурсів. Структура бази даних включає дві основні таблиці: sensor_data для збереження даних сенсорів та user для інформації про користувачів.

Компонент retrieve_data відповідає за збір даних з ESP32 та їх збереження у базу даних.

Серверна частина на основі Flask обробляє запити від клієнтів та взаємодіє з базою даних, забезпечуючи обробку та аналіз даних.

Вебклієнт надає користувачам можливість реєстрації, авторизації та доступу до функціоналу додатку.

Додаток дозволяє користувачам виконувати статистичний аналіз даних, візуалізувати дані за допомогою різних типів діаграм та графіків, використовувати критерії серій та інші методи аналізу.

Інтерфейс користувача включає основні елементи навігації, такі як кнопки для переходу на домашню сторінку, налаштувань, перегляду даних та реєстрації й авторизації.

Розроблена система базується на мікросервісній архітектурі, яка забезпечує високу продуктивність, надійність, безпеку та масштабованість. Використання реляційної бази даних SQLite спрощує роботу з даними та знижує системні вимоги. Веб-додаток надає користувачам інтуїтивно зрозумілий інтерфейс для взаємодії з системою та проведення статистичного аналізу даних.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКА

4.1 Існуючі методи тестування

Методи тестування програмного забезпечення включають різноманітні підходи та техніки, які використовуються для забезпечення його якості. Роздивимося основні методи тестування програмних продуктів.

Інтеграційне тестування перевіряє взаємодію між різними модулями або компонентами програми. Його мета полягає в тому, щоб виявити помилки, які можуть виникнути при інтеграції окремих частин системи.

Модульне тестування зосереджується на перевірці окремих модулів або компонентів коду. При цьому кожен модуль тестується поодинці. Цей вид тестування може допомогти виявити помилки на ранніх етапах розробки.

Тестування прийняття, проводиться для визначення, чи відповідає програмне забезпечення вимогам кінцевих користувачів та замовників. Це може включати альфа-тестування що проводиться розробниками, чи бета-тестування яке зазвичай проводиться кінцевими користувачами.

Тестування продуктивності перевіряє, як система працює під різного рівня навантаженнями. Воно включає тестування швидкості, масштабованості, стабільності додатка.

Регресійне тестування спрямоване на виявлення помилок у вже протестованих функціях після внесення змін до коду. Це допомагає переконатися, що нові зміни не вплинули негативно на існуючий функціонал.

Тестування безпеки ж виявляє вразливості в програмі, які можуть бути використані для несанкціонованого та незаконного доступу або втрати даних. Цей вид тестування включає аналіз вразливостей та інші методи забезпечення безпеки.

Функціональне тестування перевіряє функціональність системи відповідно до заданого користувачем або замовником функціоналу. Це тестування виконується з метою перевірки, чи всі функції додатку працюють як очікується.

Тестування сумісності перевіряє, правильно працює система чи ні на різних платформах, пристроях, браузерах та операційних системах.

Тестування чорного ящика, це метод тестування програмного забезпечення, при якому внутрішня структура або робота програми не відомі тестувальнику. Основна мета цього виду тестування перевірити функціональність програмного забезпечення, ґрунтуючись на специфікаціях і вимогах до системи. Тестувальники зосереджуються на введенні даних в систему і перевірці вихідних даних на коректність [28].

Ці методи тестування дозволяють утримувати високу якість програмного забезпечення, зменшити кількість помилок та задовільнити потреби користувачів.

4.2 Тестування додатка

Отже для розроблюваного додатку доцільним є використання методу тестування чорного ящика, оскільки воно забезпечує оцінку функціональності з точки зору кінцевого користувача, незалежно від внутрішньої реалізації коду. Цей метод дозволяє перевірити взаємодію між різними компонентами додатку, включаючи серверну частину, базу даних, API та фронтенд, на відповідність очікуваним результатам.

Також були використані підходи ручного тестування вебдодатка так як він має не великий обсяг та весь його функціонал можна перевірити вручну. Роздивимося даний метод тестування більш детально на конкретних прикладах.

В якості першого тесту розглянемо сценарій «реєстрація нового користувача» для цього необхідно:

1) відкрити додаток;
2) натиснути «Реєстрація»;
3) заповнення полів ім’я нового користувача, його електронна пошта, пароль;
4) натиснути кнопку «Зареєструватися».

Успішним результатом буде реєстрація користувача в додатку без попереджень про неправильно введені дані. Результат виконання тесту наведено на рис. 4.1.

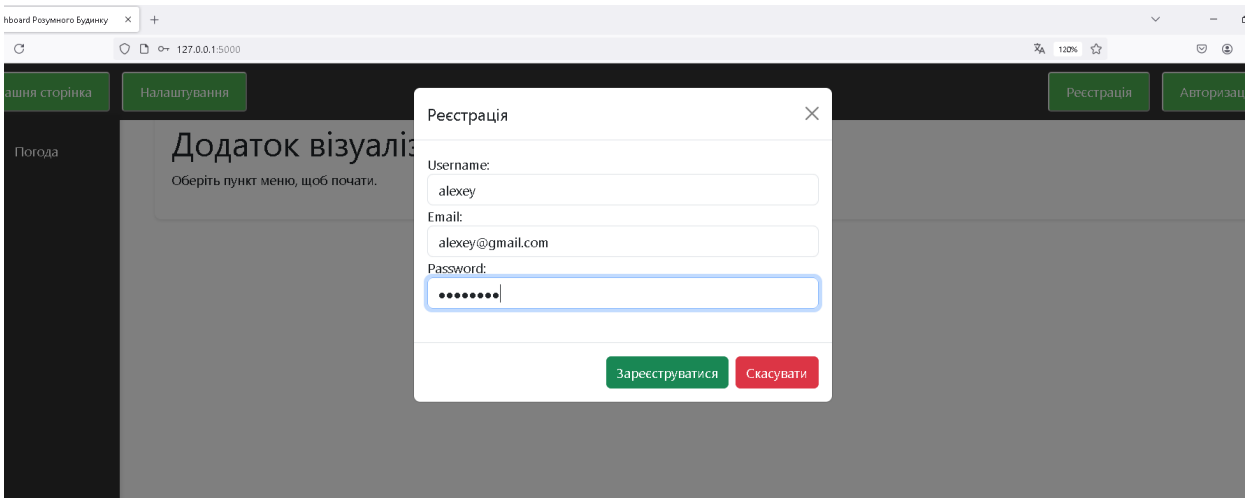

Рисунок 4.1 – Перший тест «реєстрація нового користувача»

Для того щоб переконатися в успішності попереднього кроку здійснимо вхід до додатку. Другий тест «авторизація користувача»:

1) відкрити додаток;
2) обрати пункт «авторизація»;
3) ввести дані в поля;
4) натиснути кнопку «Авторизуватися».

Очікуваним результатом є авторизація користувача в додатку, вивід на місці кнопки реєстрації «Hello, alexey "aleksei@gmail.com"», та відкриття

доступу до вкладок візуалізація та аналіз даних. Результат виконання тесту наведено на рис. 4.2.

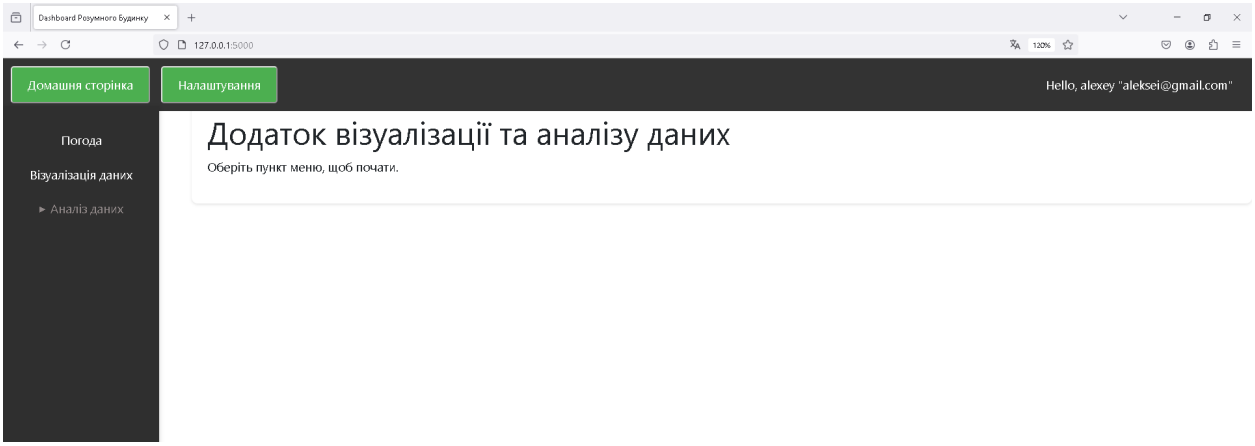

Рисунок 4.2 – Другий тест «авторизація користувача»

Наступним тестом буде обрання вкладки «Налаштування» для того щоб змінити тему сторінки:

1) відкрити додаток;
2) обрати пункт «Налаштування»;
3) перевести чек бокс в положення «Dark Theme»;
4) натиснути кнопку «Зберегти зміни».

Очікуваним результатом є зміна теми сторінки на темну, що може бути корисним в темний час доби. Результат виконання тесту наведено на рис. 4.3.

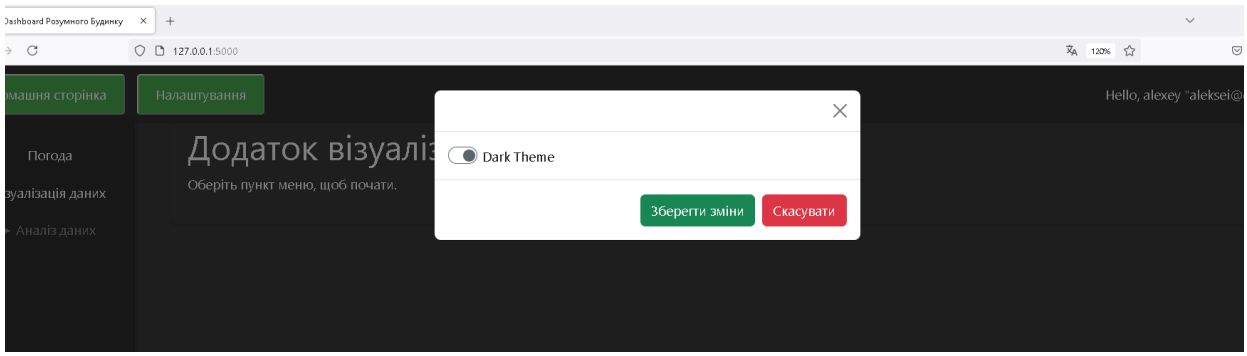

Рисунок 4.3 – Третій тест «Dark Theme»

Наступним тестом буде обрання вкладки «Погода» для отримання даних про навколишнє середовище:

1) відкрити додаток;

2) обрати пункт «Погода»;

3) ввести назву міста в відповідне поле або натиснути на обране місце на карті для отримання даних про погоду через координати.

Очікуваним результатом є вивід відомостей про погодні умови на сторінку, що може бути корисним користувачу для розуміння стану навколишнього середовища. Результат виконання тесту наведено на рис. 4.4.

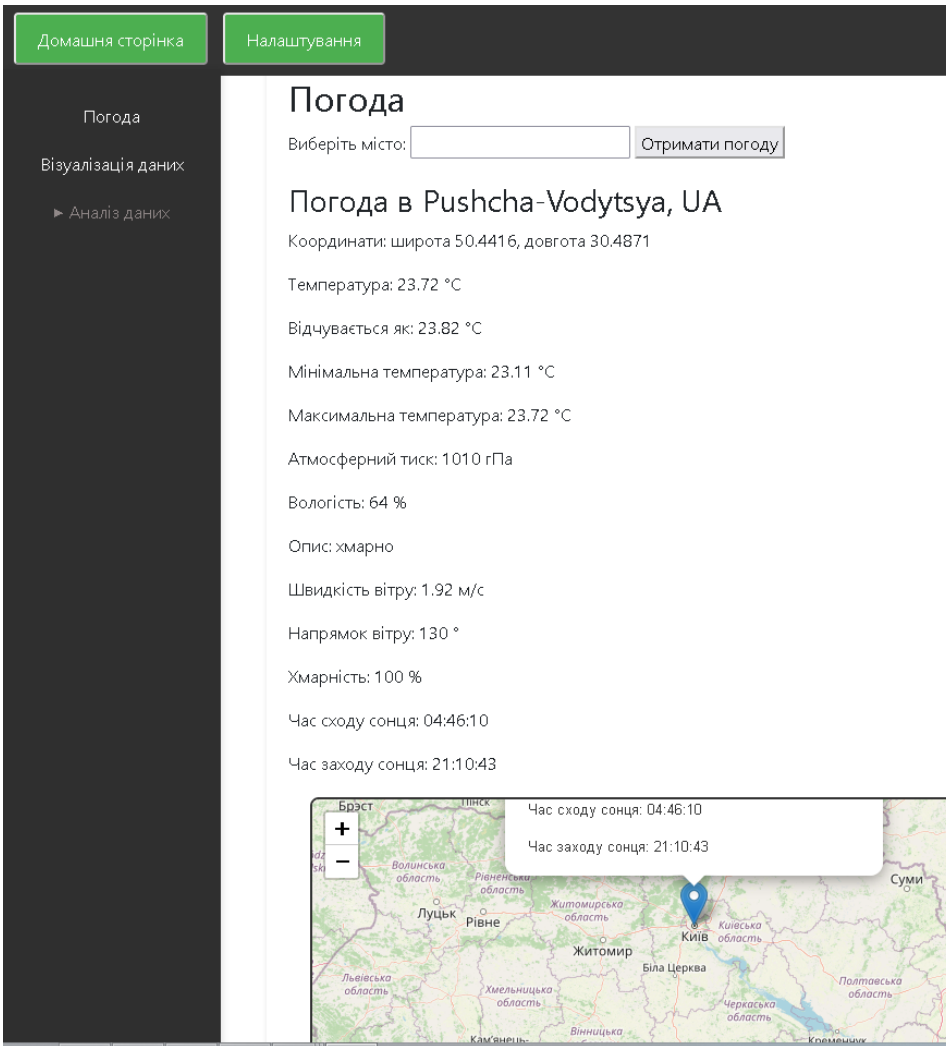

Рисунок 4.4 – Четвертий тест «відомості про погоду»

Наступним тестом буде обрання вкладки «Візуалізація даних» для візуального представлення даних отриманих з БД:

1) відкрити додаток;

2) обрати пункт «Візуалізація даних»;

3) обрати дані які будуть відображені на графіку, наприклад «вологість»;

4) обрати вид діаграми, наприклад «Лінійна».

Очікуваним результатом є візуальне представлення даних з БД, що може бути корисним користувачу для візуального сприйняття показників стану приміщення. Результат виконання тесту наведено на рис. 4.5.

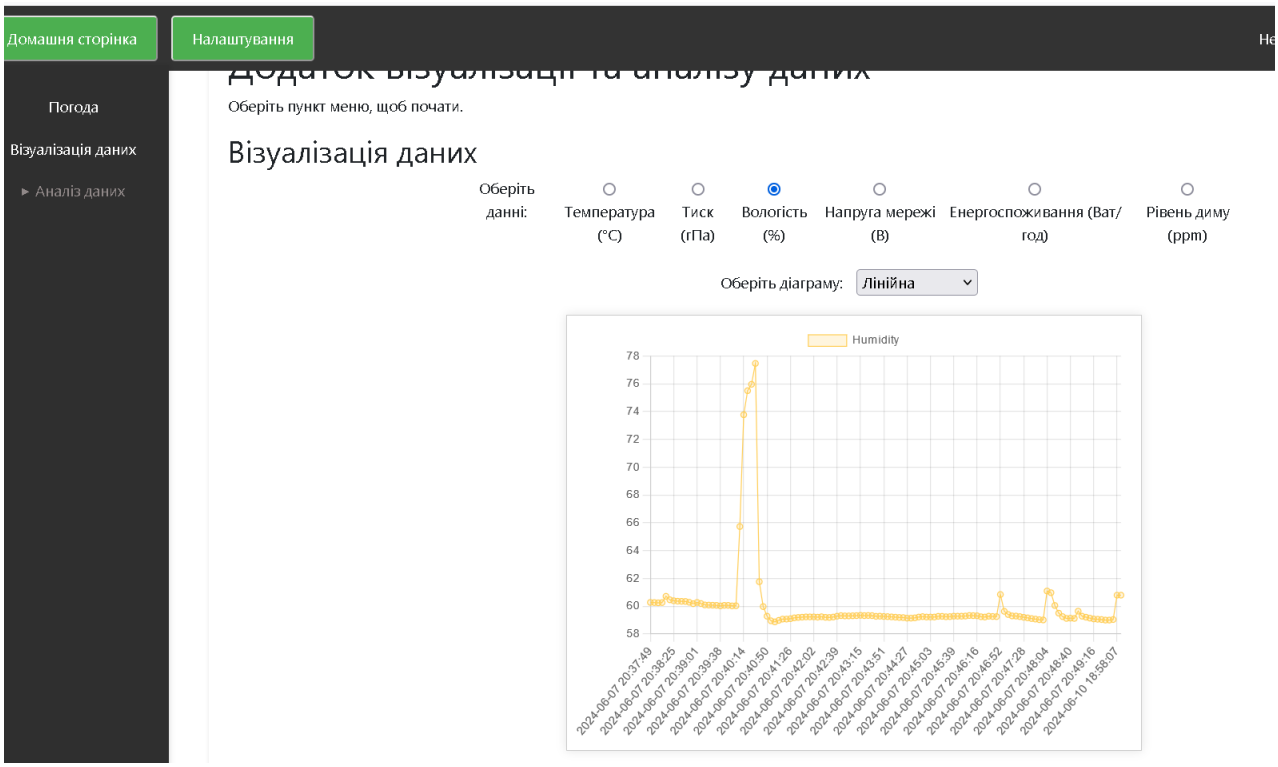

Рисунок 4.5 – П’ятий тест «візуалізація даних»

Наступним тестом буде обрання вкладки «Аналіз даних» для аналізу даних отриманих з БД:

1) відкрити додаток;

2) обрати пункт «Аналіз даних»;

3) обрати необхідний пункт статистичних розрахунків, наприклад точкові та інтервальні «оцінки вибірки»;

4) обрати дані за якими будуть проведені розрахунки, наприклад «температура».

Очікуваним результатом є представлення розрахованих даних по обраному параметру, що може бути корисним користувачу для більш поглибленого аналізу отриманих даних. Результат виконання тесту наведено на рис. 4.6.

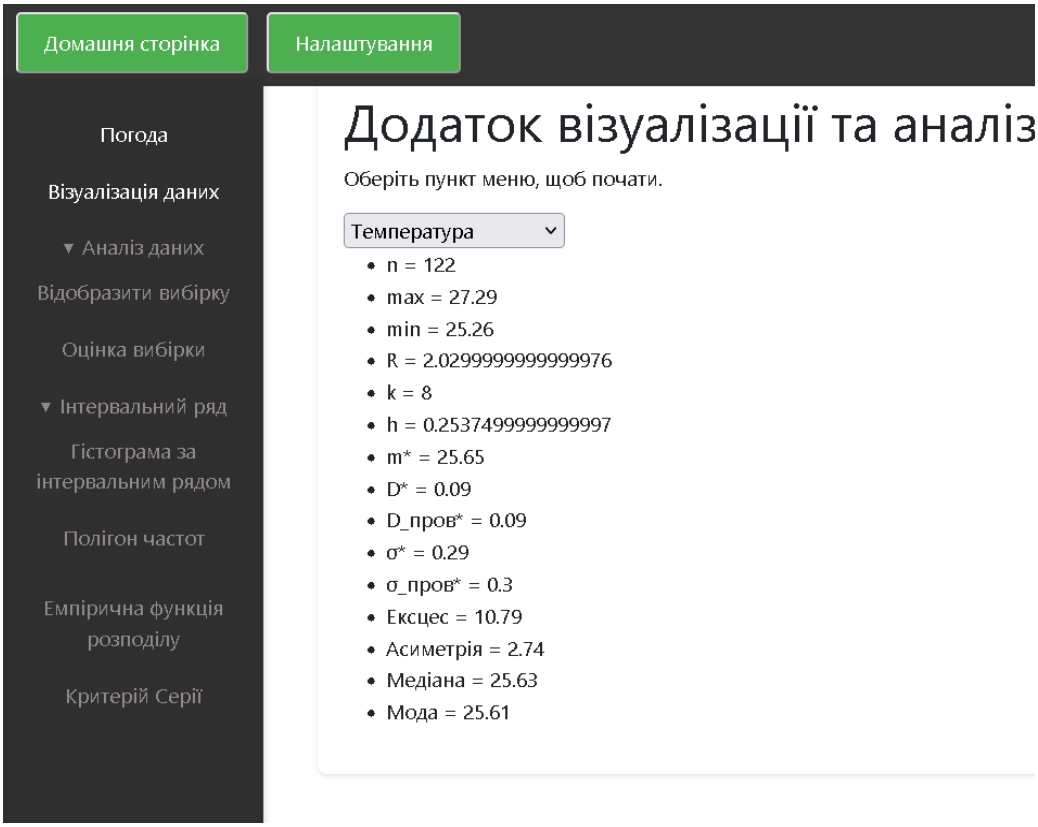

Рисунок 4.6 – Шостий тест «аналіз даних»

Наступним тестом буде обрання вкладки «Асистент» для отримання доступу до нейромережі що підключена за допомогою API:

1) відкрити додаток;

2) обрати пункт «Асистент»;

3) обрати необхідну модель нейромережі, наприклад «gpt-3.5-turbo»;

4) ввести необхідне питання, без привязки до додатку, наприклад «Що ти вмієш?».

Очікуваним результатом є вивід відповіді що отримана від API сервісу, без помилок, що може бути корисним користувачу також для аналізу даних, пояснення якихось тем, навігації по додатку, пошуку інформації, та інших задач. Результат виконання тесту наведено на рис. 4.7.

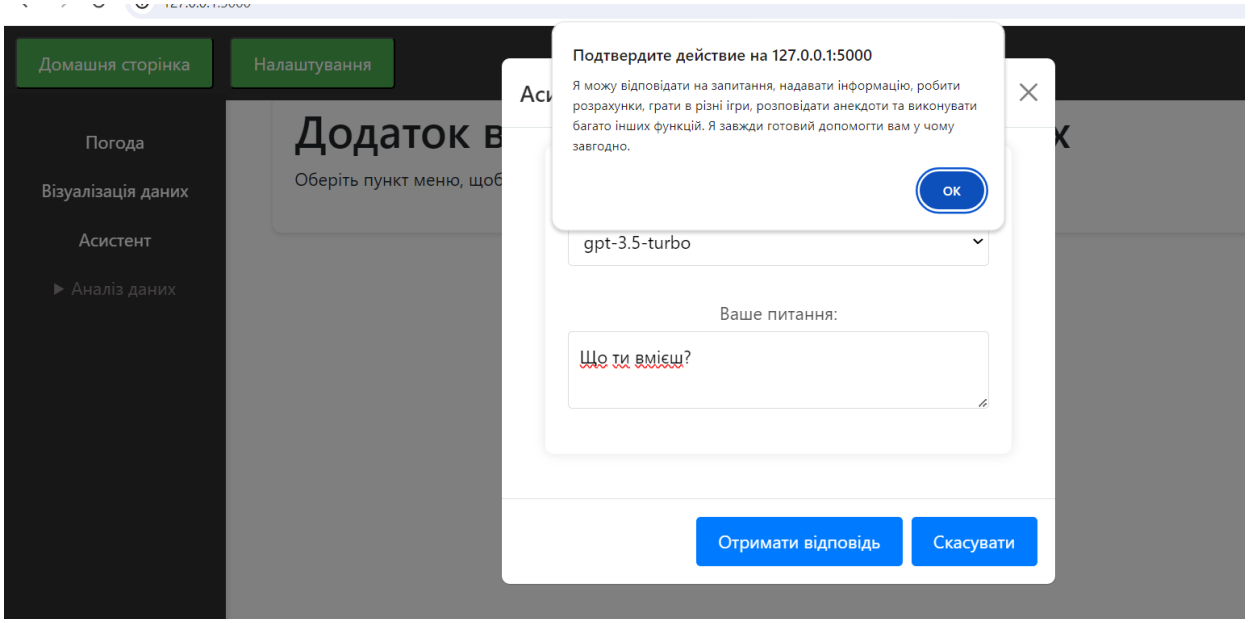

Рисунок 4.7 – Сьомий тест «асистент»

Отже, в цілому, виконання представлених тестів та їх результати відповідають очікуваним результатам, таким чином ручні тести за методом «чорного ящика» надали можливість протестувати та оцінити функціонал додатку.

4.3 Висновки за розділом 4

У даному розділі було проведено комплексне тестування розробленого вебдодатка для аналізу вибіркових даних з використанням методу «чорного ящика» та підходами ручного тестування. Виконання тестів продемонструвало

коректність роботи основних функцій додатка, так як тести пройшли успішно та відповідали очікуваним результатам. Тестування було проведено вручну, що дозволило детально перевірити кожен аспект функціонала додатка. Метод «чорного ящика» виявився ефективним для перевірки додатка такого обсягу.

Таким чином, аналіз вибіркових даних є важливою складовою розробленого вебдодатка, що забезпечує користувачам можливість отримувати цінну інформацію на основі даних з сенсорів.

Основна мета аналізу вибіркових даних полягає в отриманні інформації про популяцію на основі аналізу даних, зібраних з сенсорів у різних точках будинку, що дозволяє екстраполювати ці дані на всю систему. Це потенційно допомагає оцінити ключові параметри навколишнього середовища, такі як середня температура, вологість, рівень диму тощо.

Аналіз також дозволяє виявити зміни в отриманих даних з часом так як наявні часові мітки в базі даних, що є хорошим підґрунтям для подальшої автоматизації системи та її вдосконалення. Взаємозв’язки між різними параметрами, наприклад, вологості та температури, можна перевірити за допомогою графіків та порівняння розрахованих значень, в подальшому можна використати кореляційний та регресійний аналіз.

Також обробка даних з сенсорів допомагає приймати рішення щодо управління системою розумного дому, а також ідентифікувати можливості для покращення ефективності використання ресурсів. Моніторинг даних сенсорів дозволяє виявляти відхилення від нормальних умов та швидко реагувати на них, забезпечуючи контроль якості та стабільність роботи системи.

На основі проведеного тестування можна зробити висновок, що додаток готовий до використання кінцевими користувачами, оскільки основні його функції працюють належним чином. Подальше тестування може бути сфокусовано на перевірці додаткових сценаріїв використання та навантажувальному тестуванні для впевненості у стабільній роботі додатка в різних умовах.

ВИСНОВКИ

В бакалаврській дипломній роботі було проведено аналіз теми, існуючого програмного забезпечення та методів збору та аналізу даних, було визначено об’єкт і предмет дослідження також мету роботи.

Основним напрямком стало розроблення додатку для візуалізації (представлення складних наборів даних у чітких та інтуїтивно зрозумілих форматах з використанням веб технологій) та аналізу стану навколишнього середовища. Робота має актуальність з точки зору розкриття потенціалу розробки більш поглиблених рішень для систем розумного будинку, що зрештою призведе до підвищення їх можливостей та зручності використання.

Для розробки додатка були вибрані інструменти, засоби та технології, такі як платформа ESP, засоби веб розробки HTML, CSS, мова програмування Python, JavaScript, фреймворк Flak, та різного роду статистичні пакети. Використана декларативна мова програмування SQL, зокрема реляційна система керування базами даних SQLite та численні API що дозволили розширити функціонал додатку, та надали доступ до нейромереж. Була розроблена архітектура що включає ефективну інтеграцію між різними компонентами через API, забезпечуючи надійну взаємодію та передачу даних. Проведено тестування з використанням методу «чорного ящика», що дозволило перевірити кожен аспект функціоналу додатка, виявити та усунути можливі недоліки. Створений інтуїтивно зрозумілий графічний інтерфейс.

Таким чином, вебдодаток для візуалізації та аналізу вибіркових даних успішно пройшов основні етапи розробки та тестування. Його архітектура, функціональність та методика тестування забезпечують надійну роботу та готовність до використання. Аналіз вибіркових даних забезпечує додаткову цінність для користувачів, надаючи можливість моніторингу та оптимізації системи розумного дому.

Виконання бакалаврської дипломної роботи безумовно надало досвід в розробках подібного типу і сприяло розвитку аналітичних навичок та здатності до систематичного аналізу й проектування ПЗ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Hasan M., Anik M. H., Islam S. Microcontroller Based Smart Home System with Enhanced Appliance Switching Capacity. URL: https://www.researchgate.net/profile/Mehedi-Hasan

2. Hasan M., Biswas P., Bilash T. I., Dipto M. A. Z. Smart Home Systems: Overview And Comparative Analysis. Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN). 2018. 716 p.

3. Chakraborty A., Islam M., Shahriyar F., Islam S., Zaman H. U., Hasan M. Smart Home System: a comprehensive review. Journal of electrical and computer engineering. 2023. №2. P. 1-30.

4. Sharhan Q. I. Arduino based Smart Home warning system. IEEE 6th International Conference on Control Science and Systems Engineering (China). 2020. 210 p.

5. Ibrahim J. M., Karami A., Jafari F. A Secure Smart home Using Internet-Of-Tings. The 9th International Conference. 2017. P. 69 - 74.

6. Kalaivanan S., Manoharan S. Monitoring and controlling of smart homes using IoT and low power wireless technology. Indian Journal of Science and Technology. 2016. 31 p.

7. Choudekar P. IoT-based smart home surveillance system. Intelligent Systems and Computing. 2021. Vol. 1354.

8. Teja P. A., Joe A. A. F., Kalist V. Home security system using raspberry PI with IOT. International a conference on advance computing and innovative technologies in engineering, ICACITE. 2021. 453 p.

9. Shaheed S. M., Ilyas M. S. B., Ahamed J. Effective smart home system based on flexible cost in Pakistan. Fourth HCT Information Technology Trends (ITT). 2017. 35-38 p.

10. Невлюдов І.Ш. Технічні засоби автоматизації: Підручник / І.Ш. Невлюдов, А.О. Андрусевич, О.І. Филипенко, Н.П. Демська, С.П. Новоселов. Кривий Ріг Криворізький коледж НАУ, 2019. 366 с.

11. Kumari S., Rathi G., Attri P., Kumar M. Types of sensor and their application. International Journal of engineering research and development, 2014. № 10 (4). P. 72-85.

12. Smart sensors: Sensing the Future: The Role of Smart Sensors in Homes. URL: https://fastercapital.com/content/Smart-sensors--Sensing-the-Future--The-Role-of-Smart-Sensors-in-Homes.html

13. Official site Arduino. URL: https://www.arduino.cc/en/software

14. Official site vscodium. URL: https://vscodium.com/#why

15. Джейсон Р. Бріггс. PYTHON. Вступ до програмування [Текст] Джейсон Р Бріггс; перекл. З англ. Олександри Гордінчук. Львів: Видавництво Старого Лева, 2023. 400с.

16. Web-технології та Web-дизайн: застосування мови HTML для створення електронних ресурсів : навч. посіб. / І. Л. Бородкіна, Г. О. Бородкін. Київ: Видавництво Ліра-К, 2020. 212 с.

17. Official site jquery. URL: https://jquery.com

18. Official site leafletjs. URL: https://leafletjs.com

19. Official site chartjs. URL: https://www.chartjs.org

20. Official site openweather [Електронний ресурс]. – Режим доступу: https://openweathermap.org/guide

21. What is the best database for storing data from sensors and devices connected to an Internet of Things (IoT) system. URL: https://www.quora.com/What-is-the-best-database-for-storing-data-from-sensors-and-devices-connected-to-an-Internet-of-Things-IoT-system\

22. Роберт Мартін: Чиста архітектура: мистецтво розробки програмного забезпечення: Фабула, 2019. 416с.

23. Paul Clements, Rick Kazman: Software Architecture in Practice: Addison- Wesley Professional, 2012. 322p.

24. Володимир Гайдаржи, Ігор Ізварін: Бази даних в інформаційних системах: Університет "Україна", 2018 . 418 с.

25. Основні характеристики SQLite. URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/

26. Шаров С. В., Шамардак О. А. Концептуальне моделювання інформаційної системи за допомогою мови UML. Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення. 2014. 14 с.

27. Official site openai. URL: https://platform.openai.com/docs/guides/text-generation

28. Panagiotis Leloudas: Introduction to Software Testing: A Practical Guide to Testing, Design, Automation, and Execution 1st ed. Edition, 2023. 232 с.

Додаток А

Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

Позначення документа	Сторінка	Зміст зауваження
ПЗ		Нещільне заповнення листів
С.17-18		Невірний перелік

Додаток Б

Текст програмного застосунку

Файл “SMART-HOME/calculation / statistical_calculation.py”

```
from typing import Tuple
from io import BytesIO
from base64 import b64encode
from math import log10, sqrt
from statistics import mode, mean, median as get_median

from itertools import groupby
from scipy.stats import norm
from numpy import arange, linspace, histogram
import matplotlib.pyplot as plt
import matplotlib

from data.classes.series import Series


def calculate_parameters(gs: list) -> tuple:

    gs_n = len(gs)
    gs_max = max(gs)
    gs_min = min(gs)
    gs_r = gs_max - gs_min
    gs_k = round(1 + 3.32 * log10(gs_n))
    gs_h = gs_r / gs_k

    m_x = mean(gs)
    d_x = sum((x - m_x) ** 2 for x in gs) / len(gs)
    d_x_prav = sum((x - m_x) ** 2 for x in gs) / (len(gs) - 1)
    st_dev = sqrt(d_x)
    st_dev_prav = sqrt(gs_n / (gs_n - 1) * d_x)
    tor = sum((gs.count(i) / len(gs)) * (i - m_x) ** 4 for i in set(gs))
    excess = tor / d_x**2 - 3
    asimetr = sum((gs.count(i) / len(gs)) * (i - m_x) ** 3 for i in set(gs)) / st_dev**3
    median = get_median(gs)
    mode_val = mode(gs)

    return (
        gs_n,
        gs_max,
        gs_min,
        gs_r,
        gs_k,
        gs_h,
        m_x,
        d_x,
        d_x_prav,
        st_dev,
        st_dev_prav,
        excess,
        asimetr,
```

```
        median,
        mode_val,
    )


def frequency_distribution(gs: list, not_zip: bool = False) -> str:
    gs_n = len(gs)
    gs_r = max(gs) - min(gs)
    gs_k = round(1 + 3.32 * log10(gs_n))
    gs_h = gs_r / gs_k
    gs_sort = sorted(gs)
    gs_unique = sorted(set(gs))
    freqs_arr = [gs.count(x) for x in gs_unique]
    stat_prob = [freq / len(gs) for freq in freqs_arr]

    int_1 = [gs_sort[0] + i * gs_h for i in range(gs_k)]
    int_2 = [gs_sort[0] + (i + 1) * gs_h for i in range(gs_k)]
    int_srd = [(int_1[i] + int_2[i]) / 2 for i in range(gs_k)]

    int_m = [sum(1 for x in gs_sort if int_1[i] <= x < int_2[i]) for i in range(gs_k)]
    int_m[-1] += sum(1 for x in gs_sort if x == int_2[-1])  # Include the rightmost edge

    int_h = [int_m[i] / (gs_h * len(gs)) for i in range(gs_k)]
    int_p = [m / len(gs) for m in int_m]
    int_freq = [sum(int_p[: i + 1]) for i in range(gs_k)]

    if not_zip:
        return (
            gs_sort,
            gs_unique,
            freqs_arr,
            stat_prob,
            int_1,
            int_2,
            int_srd,
            int_m,
            int_h,
            int_p,
            int_freq,
        )
    else:
        return list(zip(int_1, int_2, int_srd, int_m, int_h, int_p, int_freq))


def build_histogram(gs: list) -> str:

    parameters = calculate_parameters(gs)
    (
        gs_sort,
        gs_unique,
        freqs_arr,
        stat_prob,
        int_1,
        int_2,
        int_srd,
        int_m,
```

```
        int_h,
        int_p,
        int_freq,
    ) = frequency_distribution(gs, not_zip=True)

    # Создаем график по новому описанию
    img = BytesIO()
    plt.figure(1, figsize=(10, 5))

    # Нормализованная густота частоты для интервального ряда
    total_count = sum(int_m)
    int_density = [m / (total_count * parameters[5]) for m in int_m]

    # Построение гистограммы
    plt.bar(
        int_srd,
        int_density,
        width=parameters[5],
        linewidth=1.2,
        edgecolor="#4472c4",
        color="none",
        label="Інтервальний ряд",
    )
    plt.title("Гістограма за інтервальним рядом")
    plt.xticks(arange(int_1[0], int_2[-1] + parameters[5], parameters[5]), rotation=45)

    # Построение функции плотности
    x = linspace(int_1[0], int_2[-1], 700)
    y = norm.pdf(x, loc=parameters[6], scale=parameters[9])
    plt.plot(x, y, color="orange", linewidth=2.2, label="Функція щільності")

    # Добавление вертикальной линии для m*
    temp_s = f"m* = {parameters[6]:.4f}"
    plt.axvline(x=parameters[6], color="green", linestyle="dashed", label=temp_s)

    plt.legend(loc="upper right")
    # plt.xlabel("Інтервали")
    # plt.ylabel("Густина частоти")
    plt.grid(True)
    plt.savefig(img, format="png")
    plt.close()
    img.seek(0)
    img_b64 = b64encode(img.getvalue()).decode("utf-8")

    return img_b64


def build_polygon(gs: list) -> str:
    parameters = calculate_parameters(gs)
    _, _, _, _, int_1, int_2, int_srd, int_m, _, _, _ = frequency_distribution(
        gs, not_zip=True
    )

    img = BytesIO()
    plt.figure()
```

```
    plt.plot(
        int_srd, int_m, marker="o", linestyle="-", color="b", label="Полігон частот"
    )
    plt.title("Полігон частот")
    plt.xlabel("Інтервали")
    plt.ylabel("Частота")
    plt.grid(True)
    plt.savefig(img, format="png")
    plt.close()

    img.seek(0)
    img_b64 = b64encode(img.getvalue()).decode("utf-8")

    return img_b64


def build_empirical_function(gs: list) -> Tuple[Tuple[int, int], str]:

    parameters = calculate_parameters(gs)
    (
        gs_sort,
        gs_unique,
        freqs_arr,
        stat_prob,
        int_1,
        int_2,
        int_srd,
        int_m,
        int_h,
        int_p,
        int_freq,
    ) = frequency_distribution(gs, not_zip=True)

    emp_func_y = [0]  # Первый элемент 0
    total = 0
    for i in range(len(int_freq)):
        total += int_p[i]  # Используем накопленные частоты для Y
        emp_func_y.append(total)
    emp_func_y[-1] = 1  # Последний элемент 1, чтобы корректно завершить функцию

    img = BytesIO()
    plt.figure()
    plt.title("Емпірична функція розподілу")
    hist, edges = histogram(int_2, bins=len(int_2))
    for i in range(len(emp_func_y) - 1):
        plt.plot(
            [edges[i], edges[i + 1]], [emp_func_y[i + 1], emp_func_y[i + 1]], c="blue"
        )
    plt.grid()
    plt.savefig(img, format="png")
    plt.close()

    img.seek(0)
    img_b64 = b64encode(img.getvalue()).decode("utf-8")

    int_data = list(
```

```
        zip(int_1, emp_func_y[:-1])
    )  # Исключаем последний элемент для таблицы
    int_data.append((int_2[-1], 1))  # Добавляем последний элемент отдельно

    return (int_data, img_b64)


def series_test(gs, alfa_ser=0.01) -> Series:

    median = get_median(gs)

    duo_arr = ["+" if x > median else "-" for x in gs]
    n1_ser = duo_arr.count("+")
    n2_ser = duo_arr.count("-")

    if n1_ser == 0 or n2_ser == 0:
        return float("inf"), float("inf"), 0, n1_ser, n2_ser

    n_ser = sum(1 for _ in groupby(duo_arr))

    z_ser = (n_ser - 2 * n1_ser * n2_ser / (n1_ser + n2_ser) - 1) / sqrt(
        (2 * n1_ser * n2_ser * (2 * n1_ser * n2_ser - (n1_ser + n2_ser)))
        / ((n1_ser + n2_ser) ** 2 * (n1_ser + n2_ser - 1))
    )
    kvant_ser = norm.ppf(1 - alfa_ser / 2)

    series = Series(
        z_ser=z_ser,
        kvant_ser=kvant_ser,
        n_ser=n_ser,
        n1_ser=n1_ser,
        n2_ser=n2_ser,
        median=median,
    )

    return series
```

Файл “SMART-HOME/data / execute_sql_query.py”

```
from os import getenv

from typing import Iterable, Any, Optional, Tuple
from sqlite3 import connect, Cursor, Connection


def execute_sql_query(sql_filename: str, parameters: Iterable[Any] = None, close_connection:bool = True)
-> Optional[Tuple[Cursor, Connection]]:

    connection = connect(getenv("DB_URI"))
    cursor = connection.cursor()

    file = open(sql_filename, "r")
    data = file.read()
```

```
    if parameters:
        cursor.execute(data, parameters)
    else:
        cursor.execute(data)

    connection.commit()

    if close_connection:
        cursor.close()
        connection.close()
    else:
        return (cursor, connection)
```

Файл “SMART-HOME/data / retrieve_data.py”

```
from time import sleep

from requests import get

from data.execute_sql_query import execute_sql_query

from dotenv import load_dotenv

load_dotenv()

execute_sql_query("./data/sql/create_table.sql")
execute_sql_query("./data/sql/create_user_table.sql")

while True:
    response = get("http://192.168.0.103/")
    data = response.json()

    # data = {
    #     "temperature": 25.26,
    #     "pressure": 992.54,
    #     "humidity": 60.79,
    #     "voltage": 234,
    #     "powerConsumption": 3600,
    #     "smokeLevel": 6379.93
    # }

    temperature = data.get("temperature")
    pressure = data.get("pressure")
    humidity = data.get("humidity")
    voltage = data.get("voltage")
    power_consumption = data.get("powerConsumption")
    smoke_level = data.get("smokeLevel")

    execute_sql_query(
        "./data/sql/insert_data.sql",
        (temperature, pressure, humidity, voltage, power_consumption, smoke_level),
    )

    sleep(5)
    # час в секундах 216, 3 хвилини 36 сек (400 измерений в сутки)
```

Файл “SMART-HOME/data /classes/ sensor_data.py”

```
from dataclasses import dataclass


@dataclass
class SensorData:

    _id: list
    temperatures: list
    pressures: list
    humidities: list
    voltages: list
    power_consumptions: list
    smoke_levels: list
    created_at: list
```

Файл “SMART-HOME/data /classes/ series.py”

```
from typing import List

from dataclasses import dataclass


@dataclass
class Series:
    z_ser: float
    kvant_ser: float
    n_ser: float
    median: float
    n1_ser: List[float]
    n2_ser: List[float]
```

Файл “SMART-HOME/data /classes/ user.py”

```
from dataclasses import dataclass

@dataclass
class User:
    _id: int
    username: str
    email: str
    password_hash: str
```

Файл “SMART-HOME/data /sql/ create_table.py”

```
CREATE TABLE IF NOT EXISTS sensor_data(
    id INTEGER NOT NULL primary key AUTOINCREMENT,
```

```
    temperature REAL NOT NULL,
    pressure REAL NOT NULL,
    humidity REAL NOT NULL,
    voltage REAL NOT NULL,
    power_consumption REAL NOT NULL,
    smoke_level REAL NOT NULL,
    created_at DATETIME NOT NULL DEFAULT (DATETIME(CURRENT_TIMESTAMP, '+3 hours'))
);
```

Файл “SMART-HOME/data /sql/ create_user_table.py”

```
CREATE TABLE IF NOT EXISTS users(
    id INTEGER NOT NULL primary key AUTOINCREMENT,
    username VARCHAR(15) NOT NULL UNIQUE,
    email VARCHAR(319) NOT NULL UNIQUE,
    password_hash TEXT NOT NULL
);
```

Файл “SMART-HOME/data /sql/ get_data.py”

```
SELECT id, temperature, pressure, humidity, voltage, power_consumption, smoke_level, created_at
FROM sensor_data
```

Файл “SMART-HOME/data /sql/ get_user.py”

```
SELECT * FROM users WHERE username = ?;
```

Файл “SMART-HOME/data /sql/ insert_data.py”

```
INSERT INTO sensor_data (temperature,pressure,humidity,voltage,power_consumption,smoke_level)
VALUES
    (?,?,?,?,?,?);
```

Файл “SMART-HOME/data /sql/ insert_user.py”

```
INSERT INTO users(username, email, password_hash)
    VALUES (?,?,?);
```

Файл “SMART-HOME/services / sensor_data_service.py”

```
from typing import List, Any, List

from data.execute_sql_query import execute_sql_query
from data.classes.sensor_data import SensorData


def get_sensor_data() -> SensorData:
    cursor, connection = execute_sql_query(
        "./data/sql/get_data.sql", close_connection=False
```

```
    )
    data = cursor.fetchall()
    cursor.close()
    connection.close()

    sensor_data = SensorData(
        _id=[row[0] for row in data],
        temperatures=[row[1] for row in data],
        pressures=[row[2] for row in data],
        humidities=[row[3] for row in data],
        voltages=[row[4] for row in data],
        power_consumptions=[row[5] for row in data],
        smoke_levels=[row[6] for row in data],
        created_at=[row[7] for row in data],
    )

    return sensor_data

def get_sensor_matrix_data() -> List[List[str]]:

    sensor_data: list = get_sensor_data()

    sensor_matrix_data = [
        sensor_data.temperatures,
        sensor_data.pressures,
        sensor_data.humidities,
        sensor_data.voltages,
        sensor_data.power_consumptions,
        sensor_data.smoke_levels
    ]

    return sensor_matrix_data
```

Файл “SMART-HOME/services / user_service.py”

```
from typing import Optional, Tuple

from data.execute_sql_query import execute_sql_query
from data.classes.user import User


def create_user(username, email, password) -> None:
    from app import bcrypt

    password_hash = bcrypt.generate_password_hash(password).decode()

    execute_sql_query("./data/sql/insert_user.sql", (username, email, password_hash))


def get_user(username, password) -> Tuple[Optional[User], Optional[str]]:
    from app import bcrypt

    cursor, connection = execute_sql_query(
        "data/sql/get_user.sql", (username,), close_connection=False
    )
```

```
    user_tuple = cursor.fetchone()

    cursor.close()
    connection.close()

    if user_tuple:
        user = User(
            _id=user_tuple[0],
            username=user_tuple[1],
            email=user_tuple[2],
            password_hash=user_tuple[3],
        )

        if bcrypt.check_password_hash(user.password_hash, password):
            return user, None
        else:
            return None, "Невірний пароль!"
    else:
        return None, "Невірний пароль та невірне ім'я!"
```

Файл “SMART-HOME/ sketch_esp_webserver / sketch_esp_webserver.ino”

```
#include <WiFi.h>
#include <WebServer.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include "networkConfig.h"
#include "jsonHelper.h" // Підключення файлу з функцією генерації JSON

// Адреса BME280 на I2C шині
#define BME280_ADDRESS 0x76

// Створення об'єкта веб-сервера на порту 80
WebServer server(80);

// Створення об'єкта датчика BME280
Adafruit_BME280 bme;

// Дані з датчика
float temperature = 0;
float humidity = 0;
float pressure = 0;
float voltage = 220.0;
float powerConsumption = 600.0;
float smokeLevel = 300.0; // Початкове значення рівня диму

void setup() {
  Serial.begin(57600); // Ініціалізація серійного зв'язку для дебагу

  // Ініціалізація BME280
  if (!bme.begin(BME280_ADDRESS)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1); // Залишатися тут, якщо датчик не знайдений
  }
```

```
  setupWiFi(); // Виклик функції для налаштування Wi-Fi

  // Налаштування веб-сервера
  server.on("/", handleRoot); // Вказуємо, яка функція буде обробляти запит до кореня ("/")
  server.begin(); // Запускаємо веб-сервер
  Serial.println("HTTP server started");
}

void loop() {
  server.handleClient(); // Обробка клієнтських запитів
  getSensorData(); // Оновлення даних з датчика
  updateVoltage(); // Оновлення значення напруги
  updatePowerConsumption(); // Оновлення значення енергоспоживання
  updateSmokeLevel(); // Оновлення значення рівня диму
  delay(1000); // Затримка в 1 секунду перед наступним циклом
}

void handleRoot() {
  String json = getJSON(temperature, pressure, humidity, voltage, powerConsumption, smokeLevel); // Отримуємо JSON з даними датчика
  server.send(200, "application/json", json); // Відправляємо JSON у відповідь на запит
}

void getSensorData() {
  temperature = bme.readTemperature(); // Зчитування температури з датчика
  humidity = bme.readHumidity(); // Зчитування вологості з датчика
  pressure = bme.readPressure() / 100.0F; // Зчитування тиску з датчика та конвертація з Па в гПа
}

// Функция для генерации числа по нормальному распределению методом Бокса-Мюллера и обновления значения энергопотребления
void updatePowerConsumption() {
    static bool has_spare = false; // Флаг для хранения запаса
    static double spare; // Запасное значение

    const double mean = 2100.0; // Среднее значение
    const double stddev = 500.0; // Стандартное отклонение

    // Проверка наличия запасного значения
    if (has_spare) {
        has_spare = false; // Используем запасное значение
        powerConsumption = mean + stddev * spare; // Вычисляем энергопотребление с использованием запасного значения
    } else {
        has_spare = true; // Устанавливаем флаг наличия запасного значения
        double u, v, s;
        do {
            // Генерация двух случайных чисел в диапазоне [-1, 1]
            u = 2.0 * random(0, 32767) / 32767 - 1.0;
            v = 2.0 * random(0, 32767) / 32767 - 1.0;
            s = u * u + v * v;
        } while (s >= 1.0 || s == 0.0); // Повторяем генерацию, пока s не будет в пределах (0, 1)

        s = sqrt(-2.0 * log(s) / s); // Вычисляем значение s по формуле Бокса-Мюллера
        spare = v * s; // Сохраняем запасное значение
```

```
        powerConsumption = mean + stddev * (u * s); // Вычисляем энергопотребление
    }

    // Ограничиваем значение энергопотребления в пределах [600, 3600]
    if (powerConsumption < 600) {
        powerConsumption = 600;
    }
    if (powerConsumption > 3600) {
        powerConsumption = 3600;
    }
}

// Функція для оновлення значення напруги
void updateVoltage() {
    // коливання напруги в межах 220 ± 20
    voltage = 220 + random(-20, 21);
}

// Функція для оновлення значення рівня диму

bool increasing = true; // Флаг для відстеження напрямку зміни
float currentValue = smokeLevel; // Поточне значення

void updateSmokeLevel() {
  // Генерація випадкового значення з експоненціальним розподілом
  float randomFactor = random(0, 1000) / 1000.0;
  float exponentialValue = 300 + (10000 - 300) * pow(randomFactor, 2); // Використовуємо квадратичну
функцію для плавного зміни

  // Оновлюємо поточне значення з урахуванням напрямку зміни
  if (increasing) {
    currentValue += exponentialValue * 0.01; // Плавно збільшуємо значення
    if (currentValue >= 10000) {
      increasing = false; // Міняємо напрямок на зменшення
    }
  } else {
    currentValue -= exponentialValue * 0.01; // Плавно зменшуємо значення
    if (currentValue <= 300) {
      increasing = true; // Міняємо напрямок на збільшення
    }
  }

  // Обмежуємо значення в межах 300 та 10000
  currentValue = constrain(currentValue, 300, 10000);

  smokeLevel = currentValue;
}
```

Файл “SMART-HOME/ sketch_esp_webserver / networkConfig.h”

```
#ifndef NETWORKCONFIG_H
#define NETWORKCONFIG_H

#include <WiFi.h>
```

```
// Функція для налаштування Wi-Fi з статичною IP-адресою
void setupWiFi();

#endif // NETWORKCONFIG_H
```

Файл “SMART-HOME/ sketch_esp_webserver / networkConfig.cpp”

```
#include "networkConfig.h"

#include <WiFi.h>

// Параметри Wi-Fi мережі
const char* ssid = "TP-Link_72CA";
const char* password = "39374233";

// Статична IP-адреса конфігурації
IPAddress local_IP(192, 168, 0, 103);  // Вкажіть бажану IP-адресу
IPAddress gateway(192, 168, 1, 1);     // Вкажіть IP-адресу шлюзу
IPAddress subnet(255, 255, 255, 0);    // Вкажіть маску підмережі
//IPAddress primaryDNS(8, 8, 8, 8);      // Вкажіть бажаний DNS сервер
//IPAddress secondaryDNS(8, 8, 4, 4);    // Вкажіть додатковий DNS сервер (необов'язково)

void setupWiFi(){

   WiFi.config(local_IP, gateway, subnet);
   // Підключення до Wi-Fi
   WiFi.begin(ssid, password);
   while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print("."); // Вивід точки поки з'єднання не встановлено
   }
   Serial.println("\nWiFi connected");
   Serial.println("IP address: ");
   Serial.println(WiFi.localIP()); // Виведення IP адреси
}
```

Файл “SMART-HOME/ sketch_esp_webserver / jsonHelper.h”

```
#ifndef JSONHELPER_H
#define JSONHELPER_H

#include <Arduino.h>

// Функція для формування JSON з даними датчика
String getJSON(float temperature, float pressure, float humidity, float voltage, float powerConsumption,
float smokeLevel);

#endif // JSONHELPER_H
```

Файл “SMART-HOME/ sketch_esp_webserver / jsonHelper.cpp”

```
#include "jsonHelper.h"
//#include <string>
```

```
String getJSON(float temperature, float pressure, float humidity, float voltage, float powerConsumption,
float smokeLevel) {
  // Формування JSON з даними датчика
  String json = "{ \"temperature\": " + String(temperature) +
                ", \"pressure\": " + String(pressure) +
                ", \"humidity\": " + String(humidity) +
                ", \"voltage\": " + String(voltage) +
                ", \"powerConsumption\": " + String(powerConsumption) +
                ", \"smokeLevel\": " + String(smokeLevel) + "}";

  return json; // Повертаємо сформований JSON
}
```

Файл “SMART-HOME/ static /js/ dark-theme.js”

```
(function() {
    document.querySelector('#dark-theme-checkbox').checked = false
})();

function changeTheme(checkbox) {
    const body = document.querySelector('body')

    if (checkbox.checked)
        body.classList.add('dark-theme')
    else
        body.classList.remove('dark-theme')
}
```

Файл “SMART-HOME/ static /js/ empirical-function.js”

```
const dataViewsEmpiricalFunction = document.querySelectorAll('.data-view-empirical-function')
const dataViewsEmpiricalFunctionDiagram = document.querySelectorAll('.data-view-empirical-function-
diagram')
let firstLookEmpiricalFunction = true

function hideAllDataViewsEmpiricalFunction() {
    dataViewsEmpiricalFunction.forEach((dataView) => {
        dataView.style.display = 'none'
    })
    dataViewsEmpiricalFunctionDiagram.forEach((diagram) => {
        diagram.style.display = 'none'
    })
}

function showEmpiricalFunction() {
    document.getElementById('weather').style.display = 'none';
    document.getElementById('select').style.display = 'none';
    document.getElementById('visualization').style.display = 'none';
    document.getElementById('polygon').style.display = 'none';
    document.getElementById('sample-estimation').style.display = 'none';
    document.getElementById('empirical-function').style.display = 'block';
    document.getElementById('interval-row').style.display = 'none';
    document.getElementById('series-criteria').style.display = 'none';
```

```
    if (firstLookEmpiricalFunction === true) {
        selectDataViewsEmpiricalFunction(0)
        firstLookEmpiricalFunction = false
    }
}

function selectDataViewsEmpiricalFunction(value) {
    hideAllDataViewsEmpiricalFunction()
    dataViewsEmpiricalFunction[value].style.display = 'inline'
    dataViewsEmpiricalFunctionDiagram[value].style.display = 'inline'
}

function selectDataEmpiricalFunction(element) {
    value = element.value
    selectDataViewsEmpiricalFunction(value)
}
```

Файл “SMART-HOME/ static /js/ interval-row.js”

```
const dataViewsIntervalRow = document.querySelectorAll('.data-view-interval-row')
const dataViewsIntervalRowHistogram = document.querySelectorAll('.data-view-interval-row-
histogram')
let firstLookIntervalRow = true

function hideAllDataViewsIntervalRow() {
    dataViewsIntervalRow.forEach((dataView) => {
        dataView.style.display = 'none'
    })
    dataViewsIntervalRowHistogram.forEach((histogram) => {
        histogram.style.display = 'none'
    })
}

function showIntervalRow() {
    document.getElementById('weather').style.display = 'none';
    document.getElementById('select').style.display = 'none';
    document.getElementById('visualization').style.display = 'none';
    document.getElementById('polygon').style.display = 'none';
    document.getElementById('sample-estimation').style.display = 'none';
    document.getElementById('empirical-function').style.display = 'none';
    document.getElementById('interval-row').style.display = 'block';
    document.getElementById('series-criteria').style.display = 'none';
    if (firstLookIntervalRow === true) {
        selectDataViewsIntervalRow(0)
        firstLookIntervalRow = false
    }
}

function selectDataViewsIntervalRow(value) {
    hideAllDataViewsIntervalRow()
    dataViewsIntervalRow[value].style.display = 'inline'
    dataViewsIntervalRowHistogram[value].style.display = 'inline'
}
```

```
function selectDataIntervalRow(element) {
   value = element.value
   selectDataViewsIntervalRow(value)
}
```

Файл “SMART-HOME/ static /js/polygon.js”

```
const dataViewsPolygon = document.querySelectorAll('.data-view-polygon')
let firstLookPolygon = true

function hideAllDataViewsPolygon() {
   dataViewsPolygon.forEach((dataView) => {
      dataView.style.display = 'none'
   })
}

function showPolygon() {
   document.getElementById('weather').style.display = 'none';
   document.getElementById('select').style.display = 'none';
   document.getElementById('visualization').style.display = 'none';
   document.getElementById('sample-estimation').style.display = 'none';
   document.getElementById('polygon').style.display = 'block';
   document.getElementById('empirical-function').style.display = 'none';
   document.getElementById('interval-row').style.display = 'none';
   document.getElementById('series-criteria').style.display = 'none';
   if (firstLookPolygon === true) {
      selectDataViewsPolygon(0)
      firstLookPolygon = false
   }
}

function selectDataViewsPolygon(value) {
   hideAllDataViewsPolygon()
   dataViewsPolygon[value].style.display = 'block'
}

function selectDataPolygon(element) {
   value = element.value
   selectDataViewsPolygon(value)
}
```

Файл “SMART-HOME/ static /js/ sample-estimation.js”

```
const dataViewsSampleEstimation = document.querySelectorAll('.data-view-sample-estimation')
let firstLookSampleEstimation = true

function hideAllDataViewsSampleEstimation() {
   dataViewsSampleEstimation.forEach((dataView) => {
      dataView.style.display = 'none'
   })
}

function showSampleEstimation() {
   document.getElementById('weather').style.display = 'none';
```

```
    document.getElementById('select').style.display = 'none';
    document.getElementById('visualization').style.display = 'none';
    document.getElementById('polygon').style.display = 'none';
    document.getElementById('empirical-function').style.display = 'none';
    document.getElementById('sample-estimation').style.display = 'block';
    document.getElementById('interval-row').style.display = 'none';
    document.getElementById('series-criteria').style.display = 'none';
    if (firstLookSampleEstimation === true) {
        selectDataViewsSampleEstimation(0)
        firstLookSampleEstimation = false
    }
}

function selectDataViewsSampleEstimation(value) {
    hideAllDataViewsSampleEstimation()
    dataViewsSampleEstimation[value].style.display = 'inline'
}

function selectDataSampleEstimation(element) {
    value = element.value
    selectDataViewsSampleEstimation(value)
}
```

Файл “SMART-HOME/ static /js/ select.js”

```
const dataViews = document.querySelectorAll('.data-view')
let firstLook = true

function hideAllDataViews() {
    dataViews.forEach((dataView) => {
        dataView.style.display = 'none'
    })
}

function showSelect() {
    document.getElementById('weather').style.display = 'none';
    document.getElementById('select').style.display = 'block';
    document.getElementById('visualization').style.display = 'none';
    document.getElementById('polygon').style.display = 'none';
    document.getElementById('sample-estimation').style.display = 'none';
    document.getElementById('empirical-function').style.display = 'none';
    document.getElementById('interval-row').style.display = 'none';
    document.getElementById('series-criteria').style.display = 'none';
    if (firstLook === true) {
        selectDataView(0)
        firstLook = false
    }
}

function selectDataView(value) {
    hideAllDataViews()
    dataViews[value].style.display = 'inline'
}

function selectData(element) {
    value = element.value
```

```
    selectDataView(value)
}
```

Файл “SMART-HOME/ static /js/ series-criteria.js”

```
const dataViewsSeriesCriteria = document.querySelectorAll('.data-view-series-criteria')
const dataViewsSeriesCriteriaTable = document.querySelectorAll('.data-view-series-criteria-table')
let firstLookSeriesCriteria = true

function hideAllDataViewsSeriesCriteria() {
    dataViewsSeriesCriteria.forEach((dataView) => {
        dataView.style.display = 'none'
    })
    dataViewsSeriesCriteriaTable.forEach((table) => {
        table.style.display = 'none'
    })
}

function showSeriesCriteria() {
    document.getElementById('weather').style.display = 'none';
    document.getElementById('select').style.display = 'none';
    document.getElementById('visualization').style.display = 'none';
    document.getElementById('polygon').style.display = 'none';
    document.getElementById('sample-estimation').style.display = 'none';
    document.getElementById('series-criteria').style.display = 'block';
    document.getElementById('empirical-function').style.display = 'none';
    document.getElementById('interval-row').style.display = 'none';
    if (firstLookSeriesCriteria === true) {
        selectDataViewsSeriesCriteria(0)
        firstLookSeriesCriteria = false
    }
}

function selectDataViewsSeriesCriteria(value) {
    hideAllDataViewsSeriesCriteria()
    dataViewsSeriesCriteria[value].style.display = 'inline'
    dataViewsSeriesCriteriaTable[value].style.display = 'inline'
}

function selectDataSeriesCriteria(element) {
    value = element.value
    selectDataViewsSeriesCriteria(value)
}
```

Файл “SMART-HOME/ static /js/visualization.js”

```
function showVisualization() {
    document.getElementById('weather').style.display = 'none';
    document.getElementById('select').style.display = 'none';
    document.getElementById('polygon').style.display = 'none';
    document.getElementById('visualization').style.display = 'block';
    document.getElementById('sample-estimation').style.display = 'none';
    document.getElementById('interval-row').style.display = 'none';
    document.getElementById('empirical-function').style.display = 'none';
    document.getElementById('series-criteria').style.display = 'none';
}
```

```
async function fetchData() {
    const response = await fetch("/data");
    const data = await response.json();
    return data;
}

const colors = {
    temperature: "rgba(255, 99, 132, 0.2)",
    pressure: "rgba(54, 162, 235, 0.2)",
    humidity: "rgba(255, 206, 86, 0.2)",
    voltage: "rgba(75, 192, 192, 0.2)",
    powerConsumption: "rgba(153, 102, 255, 0.2)",
    smokeLevel: "rgba(255, 159, 64, 0.2)",
};

const borderColors = {
    temperature: "rgba(255, 99, 132, 1)",
    pressure: "rgba(54, 162, 235, 1)",
    humidity: "rgba(255, 206, 86, 1)",
    voltage: "rgba(75, 192, 192, 1)",
    powerConsumption: "rgba(153, 102, 255, 1)",
    smokeLevel: "rgba(255, 159, 64, 1)",
};

async function createChart(fields, chartType) {
    const data = await fetchData();
    data.labels = data.createdAt
    console.log(data);
    const context = document.getElementById("chart").getContext("2d");
    context.canvas.width = 800
    context.canvas.height = 600

    const datasets = fields.map((field) => ({
        label: field.charAt(0).toUpperCase() + field.slice(1),
        data: data[field],
        backgroundColor: colors[field],
        borderColor: borderColors[field],
        borderWidth: 1,
    }));

    const chartData = {
        labels: data.labels,
        datasets: datasets,
    };

    const chartOptions = {
        scales:
            chartType === "bar" ||
            chartType === "line" ||
            chartType === "horizontalBar"
                ? {
                      y: {
                          beginAtZero: true,
                      },
                  }
```

```
            : {},
    };

    if (currentChart) {
        currentChart.destroy();
    }

    currentChart = new Chart(context, {
        type: chartType,
        data: chartData,
        options: chartOptions,
    });
}

let currentChart;

async function updateChart() {
    const fieldCheckboxes = document.querySelectorAll(".fieldCheckbox");
    const chartTypeSelect = document.getElementById("chartTypeSelect");
    const selectedFields = Array.from(fieldCheckboxes)
        .filter((checkbox) => checkbox.checked)
        .map((checkbox) => checkbox.value);
    const selectedChartType = chartTypeSelect.value;

    await createChart(selectedFields, selectedChartType);
}

document.querySelectorAll(".fieldCheckbox").forEach((checkbox) => {
    checkbox.addEventListener("change", updateChart);
});
document
    .getElementById("chartTypeSelect")
    .addEventListener("change", updateChart);

window.onload = updateChart;
```

Файл “SMART-HOME/ static /js/weather.js”

```
let map;
let marker;

function initializeMap() {
    if (map) {
        return; // Карта вже ініціалізована
    }
    map = L.map('map').setView([48.3794, 31.1656], 6); // Центр України

    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
    }).addTo(map);

    map.on('click', onMapClick);
}

function onMapClick(e) {
```

```
    if (marker) {
        map.removeLayer(marker);
    }
    marker = L.marker([e.latlng.lat, e.latlng.lng]).addTo(map);
    console.log(`Latitude: ${e.latlng.lat}, Longitude: ${e.latlng.lng}`);
    getWeatherByCoordinates(e.latlng.lat, e.latlng.lng);
}

async function getWeather(city) {
    try {
        const response = await fetch(`/weather?city=${city}`);
        const data = await response.json();
        if (data.cod === 200) {
            displayWeather(data);
        } else {
            document.getElementById('weatherInfo').innerHTML = `<p>Місто не знайдено. Будь ласка, спробуйте ще раз.</p>`;
        }
    } catch (error) {
        console.error('Помилка:', error);
        document.getElementById('weatherInfo').innerHTML = `<p>Сталася помилка. Будь ласка, спробуйте пізніше.</p>`;
    }
}

async function getWeatherByCoordinates(lat, lon) {
    try {
        const response = await fetch(`/weather?lat=${lat}&lon=${lon}`);
        const data = await response.json();
        console.log('Weather data:', data);
        if (data.cod === 200) {
            displayWeather(data);
        } else {
            document.getElementById('weatherInfo').innerHTML = `<p>Місце не знайдено. Будь ласка, спробуйте ще раз.</p>`;
        }
    } catch (error) {
        console.error('Помилка:', error);
        document.getElementById('weatherInfo').innerHTML = `<p>Сталася помилка. Будь ласка, спробуйте пізніше.</p>`;
    }
}

function displayWeather(data) {
    const weatherInfo = `
        <h3>Погода в ${data.name}, ${data.sys.country}</h3>
        <p>Координати: широта ${data.coord.lat}, довгота ${data.coord.lon}</p>
        <p>Температура: ${data.main.temp} °C</p>
        <p>Відчувається як: ${data.main.feels_like} °C</p>
        <p>Мінімальна температура: ${data.main.temp_min} °C</p>
        <p>Максимальна температура: ${data.main.temp_max} °C</p>
        <p>Атмосферний тиск: ${data.main.pressure} гПа</p>
        <p>Вологість: ${data.main.humidity} %</p>
        <p>Опис: ${data.weather[0].description}</p>
        <p>Швидкість вітру: ${data.wind.speed} м/с</p>
        <p>Напрямок вітру: ${data.wind.deg} °</p>
```

```
        <p>Хмарність: ${data.clouds.all} %</p>
        ${data.rain ? `<p>Дощ (остання година): ${data.rain['1h']} мм</p>` : ''}
        ${data.snow ? `<p>Сніг (остання година): ${data.snow['1h']} мм</p>` : ''}
        <p>Час сходу сонця: ${new Date(data.sys.sunrise * 1000).toLocaleTimeString()}</p>
        <p>Час заходу сонця: ${new Date(data.sys.sunset * 1000).toLocaleTimeString()}</p>
    `;
    document.getElementById('weatherInfo').innerHTML = weatherInfo;

    if (marker) {
        marker.bindPopup(weatherInfo).openPopup();
    }
}

document.getElementById('cityForm').addEventListener('submit', (event) => {
    event.preventDefault();
    const city = document.getElementById('city').value;
    getWeather(city);
});

function showWeather() {
    document.getElementById('weather').style.display = 'block';
    try {
        document.getElementById('select').style.display = 'none';
        document.getElementById('visualization').style.display = 'none';
        document.getElementById('polygon').style.display = 'none';
        document.getElementById('sample-estimation').style.display = 'none';
        document.getElementById('interval-row').style.display = 'none';
        document.getElementById('empirical-function').style.display = 'none';
        document.getElementById('series-criteria').style.display = 'none';
        initializeMap();
    } catch(error) {
    }
}
```

Файл “SMART-HOME/ templates /index.html”

```
<!DOCTYPE html>
<html lang="uk">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dashboard Розумного Будинку</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles/styles.css') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='styles/lib/jquery-ui.css') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='styles/lib/pretty-checkbox.css') }}">
    <link rel="stylesheet" href="https://unpkg.com/leaflet/dist/leaflet.css" />
    <link href="{{ url_for('static', filename='styles/lib/bootstrap.min.css') }}" rel="stylesheet">
    <script src="{{ url_for('static', filename='js/lib/jquery-3.6.0.min.js') }}"></script>
    <script src="{{ url_for('static', filename='js/lib/jquery-ui.min.js') }}"></script>
    <script src="{{ url_for('static', filename='js/lib/leaflet.js') }}"></script>
    <script src="{{ url_for('static', filename='js/lib/chart.js') }}"></script>
</head>

<body>
    <div class="top-bar">
```

```
      <div class="menu-left">
        <button class="button menu-item" onclick="window.location.href='/'">Домашня сторінка</button>
        <button class="button menu-item" data-bs-toggle="modal"
          data-bs-target="#settingsModal">Налаштування</button>
      </div>
      <div class="menu-right">
        {% if user %}
        <span>Hello, {{ user.username }} "{{ user.email }}"</span>
        {% else %}
        <button class="button menu-item" data-bs-toggle="modal" data-bs-target="#registerModal">Реєстрація</button>
        <button class="button menu-item" data-bs-toggle="modal" data-bs-target="#loginModal">Авторизація</button>
        {% endif %}
      </div>
    </div>
    <div class="side-menu">
      <button type="button" onclick="showWeather()">Погода</button>
      {% if user %}
      <button type="button" onclick="showVisualization()">Візуалізація даних</button>
      <!-- <a onclick="fetch('/open-window')" class="clickable">Аналіз даних</a> -->
      <details>
        <summary>Аналіз даних</summary>
        <button type="button" onclick="showSelect()">Відобразити вибірку</button>
        <button type="button" onclick="showSampleEstimation()">Оцінка вибірки</button>
        <details>
          <summary>Інтервальний ряд</summary>
          <button type="button" onclick="showIntervalRow()">Гістограма за інтервальним рядом</button>
          <button type="button" onclick="showPolygon()">Полігон частот</button>
        </details>
        <button type="button" onclick="showEmpiricalFunction()">Емпірична функція розподілу</button>
        <button type="button" onclick="showSeriesCriteria()">Критерій Серії</button>
      </details>
      {% endif %}
    </div>
    <div class="content">

      {% if error_message %}
      <div class="alert alert-danger">
        <span>{{ error_message }}</span>
      </div>
      {% endif %}

      <h1>Додаток візуалізації та аналізу даних</h1>
      <p>Оберіть пункт меню, щоб почати.</p>

      <div id="weather" class="weather-container" style="display: none;">
        <h2>Погода</h2>
        <form id="cityForm">
          <label for="city">Виберіть місто:</label>
          <input type="text" id="city" name="city" required>
          <button type="submit">Отримати погоду</button>
        </form>
```

```
        <div id="weatherInfo" class="weather-info"></div>
        {% if user %}
        <div id="map"></div>
        {% endif %}
      </div>

      {% if user %}
      <div id="visualization" class="visualization-container" style="display: none;">
        <h2>Візуалізація даних</h2>
        <div id="controls">
          <center>
            <div class="selected-field">
              <label>Оберіть данні:</label>
              <label><input type="radio" name="selected-field" class="fieldCheckbox"
                  value="temperature"><br>Температура (°C)</label>
              <label><input type="radio" name="selected-field" class="fieldCheckbox"
value="pressure"><br>Тиск
                  (гПа)</label>
              <label><input type="radio" name="selected-field" class="fieldCheckbox"
                  value="humidity"><br>Вологість (%)</label>
              <label><input type="radio" name="selected-field" class="fieldCheckbox"
                  value="voltage"><br>Напруга мережі (В)</label>
              <label><input type="radio" name="selected-field" class="fieldCheckbox"
                  value="powerConsumption"><br>Енергоспоживання (Ват/год)</label>
              <label><input type="radio" name="selected-field" class="fieldCheckbox"
                  value="smokeLevel"><br>Рівень диму (ppm)</label>
            </div>
          </center>
          <div>
            <label for="chartTypeSelect">Оберіть діаграму:</label>
            <select id="chartTypeSelect">
              <option value="line">Лінійна</option>
              <option value="bar">Гістограма</option>
              <option value="pie">Кругова</option>
              <option value="doughnut">Пончикова</option>
              <option value="radar">Радар</option>
              <option value="polarArea">Полярна</option>
              <option value="bubble">Бульбашкова</option>
              <option value="scatter">Точкова</option>
            </select>
          </div>
        </div>
        <div id="chartContainer">
          <canvas id="chart"></canvas>
        </div>
      </div>
      <div id="select" class="select-container" style="display: none;">
        <h2>Вибірка та її обробка</h2>
        <select name="select-data" id="select-data" onchange="selectData(this)">
          <option value="0" selected>Температура</option>
          <option value="1">Тиск</option>
          <option value="2">Вологість</option>
          <option value="3">Напруга мережі</option>
          <option value="4">Енергоспоживання</option>
          <option value="5">Рівень диму</option>
        </select>
```

```
        <br><br>
        {% for i in range(0,6) %}
        <div class="data-view">
          <table>
            <thead>
              <tr>
                <th>#</th>
                <th>Простий ряд</th>
                <th>Варіаційний ряд</th>
                <th>Частотно-варіаційний ряд</th>
                <th>Частотна</th>
                <th>Ймовірність</th>
              </tr>
            </thead>
            <tbody>
              {% for j in range(length) %}
              <tr>
                <td>{{ j + 1 }}</td>
                <td>{{ data[i][j] }}</td>
                <td>{{ data_sorted[i][j] }}</td>
                <td>{{ matrix_unique[i][j] }}</td>
                <td>{{ matrix_quantity[i][j] }}</td>
                <td>{{ matrix_probability[i][j] }}</td>
              </tr>
              {% endfor %}
            </tbody>
          </table>
        </div>
        {% endfor %}
      </div>
      <div id="sample-estimation" class="sample-estimation-container" style="display: none;">
        <select name="select-data" id="select-data-sample-estimation"
onchange="selectDataSampleEstimation(this)">
          <option value="0" selected>Температура</option>
          <option value="1">Тиск</option>
          <option value="2">Вологість</option>
          <option value="3">Напруга мережі</option>
          <option value="4">Енергоспоживання</option>
          <option value="5">Рівень диму</option>
        </select>
        {% for parameters in matrix_parameters %}
        <div class="data-view-sample-estimation" style="display: none;">
          <ul>
            <li>n = {{ parameters[0] }}</li>
            <li>max = {{ parameters[1] }}</li>
            <li>min = {{ parameters[2] }}</li>
            <li>R = {{ parameters[3] }}</li>
            <li>k = {{ parameters[4] }}</li>
            <li>h = {{ parameters[5] }}</li>
            <li>m* = {{ parameters[6]|round(2) }}</li>
            <li>D* = {{ parameters[7]|round(2) }}</li>
            <li>D_пров* = {{ parameters[8]|round(2) }}</li>
            <li>σ* = {{ parameters[9]|round(2) }}</li>
            <li>σ_пров* = {{ parameters[10]|round(2) }}</li>
            <li>Ексцес = {{ parameters[11]|round(2) }}</li>
            <li>Асиметрія = {{ parameters[12]|round(2) }}</li>
```

```
                <li>Медіана = {{ parameters[13]|round(2) }}</li>
                <li>Мода = {{ parameters[14]|round(2) }}</li>
              </ul>
            </div>
            {% endfor %}
          </div>
          <div id="interval-row" class="interval-row-container" style="display: none;">
            <select name="select-data" id="select-data-interval-row"
onchange="selectDataIntervalRow(this)">
              <option value="0" selected>Температура</option>
              <option value="1">Тиск</option>
              <option value="2">Вологість</option>
              <option value="3">Напруга мережі</option>
              <option value="4">Енергоспоживання</option>
              <option value="5">Рівень диму</option>
            </select>
            {% for histogram in histograms %}
            <div class="data-view-interval-row-histogram" style="display: none;">
              <img src="{{ histogram }}" alt="Гістограма">
            </div>
            {% endfor %}
            {% for intervals in matrix_frequency_distribution %}
            <div class="data-view-interval-row" style="display: none;">
              <table>
                <thead>
                  <tr>
                    <th>Інтервали</th>
                    <th>Середина</th>
                    <th>m[i]</th>
                    <th>H[i]</th>
                    <th>p*[i]</th>
                    <th>Накоп. частоти</th>
                  </tr>
                </thead>
                <tbody>
                  {% for interval in intervals %}
                  <tr>
                    <td>{{ interval[0]|round(2) }} - {{ interval[1]|round(2) }}</td>
                    <td>{{ interval[2]|round(2) }}</td>
                    <td>{{ interval[3] }}</td>
                    <td>{{ interval[4]|round(4) }}</td>
                    <td>{{ interval[5]|round(2) }}</td>
                    <td>{{ interval[6]|round(2) }}</td>
                  </tr>
                  {% endfor %}
                </tbody>
              </table>
            </div>
            {% endfor %}
          </div>
          <div id="polygon" class="polygon-container" style="display: none;">
            <select name="select-data" id="select-data-polygon" onchange="selectDataPolygon(this)">
              <option value="0" selected>Температура</option>
              <option value="1">Тиск</option>
              <option value="2">Вологість</option>
              <option value="3">Напруга мережі</option>
```

```
            <option value="4">Енергоспоживання</option>
            <option value="5">Рівень диму</option>
          </select>
          {% for polygon in polygons %}
          <div class="data-view-polygon" style="display: none;">
            <img src="{{ polygon }}" alt="Полігон">
          </div>
          {% endfor %}
        </div>
        <div id="empirical-function" class="empirical-function-container" style="display: none;">
          <select name="select-data" id="select-data-empirical-function"
onchange="selectDataEmpiricalFunction(this)">
            <option value="0" selected>Температура</option>
            <option value="1">Тиск</option>
            <option value="2">Вологість</option>
            <option value="3">Напруга мережі</option>
            <option value="4">Енергоспоживання</option>
            <option value="5">Рівень диму</option>
          </select>
          <br>
          {% for empirical_function in empirical_functions %}
          <div class="data-view-empirical-function-diagram" style="display: none;">
            <img src="{{ empirical_function }}" alt="Функція розподілу">
          </div>
          {% endfor %}
          {% for x_and_y in x_and_y_list %}
          <div class="data-view-empirical-function" style="display: none;">
            <table>
              <thead>
                <tr>
                  <th>x</th>
                  <th>y</th>
                </tr>
              </thead>
              <tbody>
                {% for x, y in x_and_y %}
                <tr>
                  <td>{{ round(x, 5) }}</td>
                  <td>{{ round(y, 5) }}</td>
                </tr>
                {% endfor %}
              </tbody>
            </table>
          </div>
          {% endfor %}
        </div>
        <div id="series-criteria" class="series-criteria-container" style="display: none;">
          <select name="select-data" id="select-data-series-criteria"
onchange="selectDataSeriesCriteria(this)">
            <option value="0" selected>Температура</option>
            <option value="1">Тиск</option>
            <option value="2">Вологість</option>
            <option value="3">Напруга мережі</option>
            <option value="4">Енергоспоживання</option>
            <option value="5">Рівень диму</option>
          </select>
```

```
        <br>
        {% for series in series_list %}
        {% for specific_data in data %}
        <div class="data-view-series-criteria-table" style="display: none;">
          <h2>Дані</h2>
          <table>
            <thead>
              <tr>
                <th>x[i]</th>
                <th>+/-</th>
              </tr>
            </thead>
            <tbody>
              {% for x in specific_data %}
              <tr>
                <td>{{ round(x, 5) }}</td>
                {% if x > series.median %}
                <td>+</td>
                {% else %}
                <td>-</td>
                {% endif %}
              </tr>
              {% endfor %}
            </tbody>
          </table>
        </div>
        {% endfor %}
        <div class="data-view-series-criteria" style="display: none;">
          <div>
            <h2>Результати</h2>
            <p>h*x = {{ series.median }}</p>
            <p>n1 = {{ series.n1_ser }}</p>
            <p>n2 = {{ series.n2_ser }}</p>
            <p>N = {{ series.n_ser }}</p>
            <p>α = 0.01</p>
            <p>z = {{ round(series.z_ser, 5) }}</p>
            <p>u[1-α/2] = {{ round(series.kvant_ser, 5) }}</p>
            <div>
              {% if abs(series.z_ser) < series.kvant_ser %} <p>Нульова гіпотеза задовільняє
нерівності
                |z*|&#60;u[1-α/2]</p>
                <p>Це означає, що вибірка отримана випадковим чином і є однорідною.</p>
                {% else %}
                <p>Нульова гіпотеза не задовільняє нерівності |z*|&#60;u[1-α/2]</p>
                <p>Це означає, що вибірка не отримана випадковим чином і не є однорідною.</p>
                {% endif %}
            </div>
          </div>
        </div>
        {% endfor %}
      </div>
      {% endif %}
    </div>

    <!-- Modal -->
```

```
  <div class="modal fade" id="settingsModal" tabindex="-1" aria-labelledby="settingsModalLabel" aria-
hidden="true">
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-header">
          <h1 class="modal-title fs-5" id="settingsModalLabel"></h1>
          <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
        </div>
        <div class="modal-body">
          <div class="pretty p-switch">
            <input type="checkbox" id="dark-theme-checkbox" onchange="changeTheme(this)" />
            <div class="state">
              <label>Dark Theme</label>
            </div>
          </div>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-success" data-bs-dismiss="modal">Зберегти
зміни</button>
          <button type="button" class="btn btn-danger" data-bs-dismiss="modal">Скасувати</button>
        </div>
      </div>
    </div>
  </div>

  <!-- Modal -->
  <form method="POST" action="/register">
    <div class="modal fade" id="registerModal" tabindex="-1" aria-labelledby="registerModalLabel"
      aria-hidden="true">
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <h1 class="modal-title fs-5" id="registerModalLabel">Реєстрація</h1>
            <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
          </div>
          <div class="modal-body">
            <div class="form-group">
              <label for="username">Username: </label>
              <input class="form-control" type="text" name="username" id="username"
minlength="3"
                maxlength="15" required>
            </div>
            <div class="form-group">
              <label for="email">Email: </label>
              <input class="form-control" type="email" name="email" id="email" minlength="3"
                maxlength="319" required>
            </div>
            <div class="form-group">
              <label for="password">Password: </label>
              <input class="form-control" type="password" name="password" id="password"
minlength="8"
                maxlength="64" required>
            </div>
            <br>
```

```
            </div>
            <div class="modal-footer">
              <button type="submit" class="btn btn-success">Зареєструватися</button>
              <button type="button" class="btn btn-danger" data-bs-
dismiss="modal">Скасувати</button>
            </div>
          </div>
        </div>
      </div>
    </form>

    <!-- Modal -->
    <form method="POST" action="/">
      <div class="modal fade" id="loginModal" tabindex="-1" aria-labelledby="loginModalLabel" aria-
hidden="true">
        <div class="modal-dialog">
          <div class="modal-content">
            <div class="modal-header">
              <h1 class="modal-title fs-5" id="loginModalLabel">Авторизація</h1>
              <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
            </div>
            <div class="modal-body">
              <div class="form-group">
                <label for="username_login">Username: </label>
                <input class="form-control" type="text" name="username" id="username_login"
minlength="3"
                  maxlength="15" required>
              </div>
              <div class="form-group">
                <label for="password_login">Password: </label>
                <input class="form-control" type="password" name="password" id="password_login"
                  minlength="8" maxlength="64" required>
              </div>
              <br>
            </div>
            <div class="modal-footer">
              <button type="submit" class="btn btn-success">Авторизуватися</button>
              <button type="button" class="btn btn-danger" data-bs-
dismiss="modal">Скасувати</button>
            </div>
          </div>
        </div>
      </div>
    </form>


    <script src="{{ url_for('static', filename='js/lib/bootstrap.min.js') }}"></script>
    <script src="{{ url_for('static', filename='js/weather.js') }}"></script>
    <script src="{{ url_for('static', filename='js/visualization.js') }}"></script>
    <script src="{{ url_for('static', filename='js/select.js') }}"></script>
    <script src="{{ url_for('static', filename='js/sample-estimation.js') }}"></script>
    <script src="{{ url_for('static', filename='js/interval-row.js') }}"></script>
    <script src="{{ url_for('static', filename='js/polygon.js') }}"></script>
    <script src="{{ url_for('static', filename='js/empirical-function.js') }}"></script>
    <script src="{{ url_for('static', filename='js/series-criteria.js') }}"></script>
```

```
    <script src="{{ url_for('static', filename='js/dark-theme.js') }}"></script>
</body>

</html>
```

Файл “SMART-HOME/ utils / list_utils.py”

```
def sort(lst: list) -> list:
    lst_sorted = lst.copy()
    lst_sorted.sort()
    return lst_sorted

def remove_same_elements(lst: list) -> list:
    iterable = lst.copy()
    return list(set(iterable))
```

Файл “SMART-HOME/ views / get_data.py”

```
from flask import jsonify

from services.sensor_data_service import get_sensor_data


def get_data():
    try:

        sensor_data: list = get_sensor_data()

        if sensor_data:
            result = {
                "_id": sensor_data._id,
                "temperature": sensor_data.temperatures,
                "pressure": sensor_data.pressures,
                "humidity": sensor_data.humidities,
                "voltage": sensor_data.voltages,
                "powerConsumption": sensor_data.power_consumptions,
                "smokeLevel": sensor_data.smoke_levels,
                "createdAt": sensor_data.created_at,
            }
            return jsonify(result)
        else:
            return jsonify({"error": "Failed to fetch data"}), 500

    except Exception as exception:
        print(f"Exception has occurred while retrieve data from database: {exception}")
```

Файл “SMART-HOME/ views / get_weather.py”

```
from os import getenv

from requests import get
from flask import (
    request,
    jsonify,
    render_template,
```

```
)

API_KEY = getenv("API_KEY")


def get_weather():
    city = request.args.get("city")
    lat = request.args.get("lat")
    lon = request.args.get("lon")
    if city:
        url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}&units=metric&lang=uk"
    elif lat and lon:
        url = f"http://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API_KEY}&units=metric&lang=uk"
    else:
        return jsonify({"code": "400", "message": "City or coordinates required"}), 400

    print(f"Making request to URL: {url}")  # Додаємо діагностичне повідомлення
    response = get(url)
    print(f"Response from API: {response.json()}")  # Додаємо діагностичне повідомлення
    return jsonify(response.json())
```

Файл “SMART-HOME/ views / index.py”

```
from typing import List, Tuple

from flask import render_template, redirect, request

from data.classes.user import User
from data.classes.series import Series
from services.user_service import get_user
from services.sensor_data_service import get_sensor_matrix_data
from utils.list_utils import (
    sort,
    remove_same_elements,
)

from calculation.statistical_calculation import (
    calculate_parameters,
    frequency_distribution,
    build_histogram,
    build_polygon,
    build_empirical_function,
    series_test,
)


def index():

    user: User = None

    error_message: str = None
```

```
sensor_matrix_data = get_sensor_matrix_data()

data_sorted = sensor_matrix_data.copy()

matrix_parameters: List[tuple] = []

matrix_unique: List[list] = []

matrix_quantity: List[list] = []  # freqs_arr

matrix_probability: List[list] = []  # stat_prob

matrix_frequency_distribution: List[list] = []

histograms: List[str] = []

polygons: List[str] = []

x_and_y_list: List[Tuple[List[float], List[float]]] = []

empirical_functions: List[str] = []

series_list: List[Series] = []

for index, specific_data in enumerate(data_sorted):
    data_sorted[index] = sort(specific_data)
    matrix_parameters.append(calculate_parameters(specific_data))
    matrix_unique.append(remove_same_elements(specific_data))
    matrix_quantity.append(
        [specific_data.count(x) for x in matrix_unique[index]]
    )
    matrix_probability.append(
        [
            round(((quantity) / len(specific_data)), 5)
            for quantity in matrix_quantity[index]
        ]
    )
    matrix_frequency_distribution.append(frequency_distribution(specific_data))

    histograms.append(build_histogram(specific_data))
    polygons.append(build_polygon(specific_data))
    int_data, img_b64 = build_empirical_function(specific_data)
    x_and_y_list.append(int_data)
    empirical_functions.append(img_b64)
    series_list.append(series_test(specific_data))

    length = len(sensor_matrix_data[0])

if request.method == "POST":

    username = request.form.get("username")
    password = request.form.get("password")

    user, error_message = get_user(
        username=username,
        password=password,
```

```
        )

    return render_template(
        template_name_or_list="index.html",
        length=length,
        data=sensor_matrix_data,
        data_sorted=data_sorted,
        matrix_unique=matrix_unique,
        matrix_parameters=matrix_parameters,
        matrix_quantity=matrix_quantity,
        matrix_probability=matrix_probability,
        matrix_frequency_distribution=matrix_frequency_distribution,
        histograms=histograms,
        polygons=polygons,
        x_and_y_list=x_and_y_list,
        empirical_functions=empirical_functions,
        series_list=series_list,
        user=user,
        error_message=error_message,
        round=round,
        abs=abs,
    )
```

Файл “SMART-HOME/ app.py”

```
from flask import Flask
from flask_bcrypt import Bcrypt

from views.index import index
from views.get_weather import get_weather
from views.get_data import get_data
from views.register import register
from dotenv import load_dotenv

from matplotlib import use

load_dotenv()

use('Agg')

app = Flask(
    import_name=__name__,
    template_folder="templates",
    static_folder="static"
)
bcrypt = Bcrypt(app=app)

app.add_url_rule("/", view_func=index, methods=["GET", "POST"])
app.add_url_rule("/register", view_func=register, methods=["POST"])
app.add_url_rule("/weather", view_func=get_weather)
app.add_url_rule("/data", view_func=get_data)

if __name__ == "__main__":
    app.run(debug=True)
```

Файл “SMART-HOME/ getting_ready.py”

```
import data.retrieve_data
```