КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра інформаційних систем та технологій

Спеціальність 126 – Інформаційні системи та технології
Освітня програма «Програмні технології інтернет речей»

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему:

«Модель IoT-рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів»

Студента 2-го курсу групи ІРма-21

Валентин ТВАРДОВСЬКИЙ
(прізвище, ім'я, по батькові)

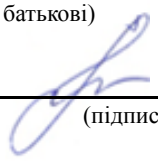

(підпис студента)

Науковий керівник:

к.т.н., доцент
(науковий ступінь, вчене звання)

Ольга КРАВЧЕНКО
(прізвище, ім'я, по батькові)

23.05.2022
(дата)


(підпис)

Попередній захист:

(Висновок: "До захисту в Екзаменаційній комісії")

Завідувач кафедри
Інформаційні системи та технології

Олександр КУЧАНСЬКИЙ
(підпис) (прізвище, ініціали) (дата)

Київ 2022

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра Інформаційні системи та технології
Освітній рівень Магістр
Спеціальність 126 Інформаційні системи та технології
Освітня програма Програмні технології інтернет речей

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., професор кафедри ІСТ
Олександр КУЧАНСЬКИЙ
_____
«_03__» __12____ 2021 року

ЗАВДАННЯ
НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Студент: Валентин ТВАРДОВСЬКИЙ

Група: **ІРма-21**

1. **Тема дипломної роботи**: «Модель IoT-рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів».
Затверджена протоколом засідання кафедри ІСТ 05/21_22 від «03» грудня 2021 р.

2. **Строк подання студентом готової роботи −** «18» травня 2022 р.

3. **Цільова установка та вихідні дані до роботи**: дослідження існуючих IoT рішень та пристроїв для моніторингу та аналізу системи сміттєзберігання твердих відходів та проектування власної моделі.

4. **Зміст роботи**: аналіз датчиків та пристроїв, засоби зв’язку і програмні технології IoT-рішення, архітектура IoT-рішення та налаштування, програмна реалізація.

5. **Перелік графічного матеріалу**: рисунки Google Trends графіків популярності та поширення, таблички специфікацій, рисунки діаграм, рисунки пристроїв та датчиків, рисунок інтерфейсу користувача (веб-сторінка).

6. **Календарний план виконання роботи:**

Етапи виконання дипломних робіт	**Термін виконання**
1. Аналіз літератури та вивчення аналогів	20.01.22-29.01.22
2. Розробка структурної схеми	29.01.22-03.02.22
3. Вибір елементної бази	03.02.22-06.02.22
4. Проектування прототипу системи	06.02.22-14.02.22
5.Програмування керівного пристрою	14.02.22-23.02.22
6. Програмування серверу	02.04.22-18.04.22
7. Створення програмного забезпечення	18.04.22-26.04.22
8. Тестування	26.04.22-05.05.22
9. Оформлення пояснювальної записки	01.04.22-08.05.2022
10. Захист роботи	26.05.22

Дата видачі завдання «_03___» __грудня____ 2021 р.

Керівник роботи: к.т.н., доц. Ольга КРАВЧЕНКО _____
(підпис)

Завдання прийняв до виконання:

студент групи **ІРма-21** Валентин ТВАРДОВСЬКИЙ _____(підпис)

АНОТАЦІЯ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра Інформаційних систем та технологій

Освітня програма «Програмні технології інтернет речей»

Кваліфікаційна робота магістра Валентина ТВАРДОВСЬКОГО

Тема роботи: «Модель IoT-рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів»

Об'єкт дослідження: IoT пристрої для контролю та моніторингу рівня наповненості ємностей для твердих відходів.

Предмет дослідження: апаратні та програмні засоби для розробки IoT рішень в сфері моніторингу та аналізу системи сміттєзберігання твердих відходів.

Мета кваліфікаційної роботи магістра: проаналізувати існуючі IoT рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів та спроектувати власний пристрій.

В роботі проведено аналіз інформації щодо існуючих IoT пристроїв для контролю та моніторингу рівня наповненості ємностей для твердих відходів, а також апаратного та програмного забезпечення, що активно застосовується в IoT проектах.

Розроблено: Модель IoT-рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів, а також повну схему взаємодії апаратного та програмного забезпечення.

Практичне значення одержаних результатів. Результати здійснених у проекті досліджень можуть бути використані при проектуванні IoT пристроїв з подібним функціоналом, а спроектоване та розроблене рішення може бути завершене і доповнене для практичного використання і в комерційних цілях.

Апробація результатів. Основні положення і результати досліджень, викладені у проекті, пройшли апробацію на IV міжнародній науково-практичній конференції «Сучасні тенденції розвитку інформаційних систем і телекомунікаційних технологій», 1–2 лютого 2022 р. (Київ, Україна) [1] та опубліковані у журналі Technology Audit and Production Reserves, Vol. 2 No. 2(64) (2022) Information and control systems. Information technologies. Pp. 6-10 . [2]

КЛЮЧОВІ СЛОВА: МОДЕЛЬ, ІНТЕРНЕТ РЕЧЕЙ, МОНІТОРИНГ, RASPBERRY PI, ДАТЧИК, СЕРВЕР, JSON

Власні публікації:

1. Твардовський В. Г., Кравченко О. В. "IoT-рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів" / IoT Solution for Monitoring and Analysis of Solid Waste Save System, IV Міжнародна науково-практична конференція, 1–2 лютого 2022 р., Pp. 180.
2. Valentyn Tvardovskyi, Olha Kravchenko, Svetlana Besedina. Design of IoT-solution for monitoring and analysis of the solid waste storage system. Technology Audit and Production Reserves, Vol. 2 No. 2(64) (2022) Information and control systems. Information technologies. Pp. 6-10 .

ABSTRACT

TARAS SHEVCHENKO NATIONAL UNIVERSITY OF KYIV

Faculty of Information Technologies

Department of Information Systems and Technologies

Educational Program "Software Technologies of the Internet of Things"

Qualification work of Master Valentyn TVARDOVSKY

Work topic: "Design of IoT-solution for monitoring and analysis of the solid waste storage system".

Object of research: IoT devices for control and monitoring of the level of filling of solid waste containers.

Subject of research: hardware and software for the development of IoT solutions in the field of monitoring and analysis of solid waste disposal system.

The purpose of the master's qualification work: to analyze existing IoT solutions for monitoring and analysis of solid waste disposal system and to design your own device.

The paper analyzes the information on existing IoT devices for control and monitoring the level of solid waste containers, as well as hardware and software that is actively used in IoT projects.

Developed: IoT solution model for monitoring and analysis of solid waste disposal system, as well as a complete scheme of hardware and software interaction.

The practical significance of the results obtained. The results of research conducted in the project can be used in the design of IoT devices with similar functionality, and the designed and developed solution can be completed and supplemented for practical use and for commercial purposes.

Approbation of results. The main provisions and results of the research presented in the project were tested at the IV International Scientific and Practical Conference "Modern Trends in Information Systems and Telecommunications Technologies", February 1-2, 2022 (Kyiv, Ukraine) [1] and published in the journal

Technology Audit and Production Reserves, Vol. 2 No. 2 (64) (2022) Information and control systems. Information technologies. Pp. 6-10. [2]

KEY WORDS: MODEL, INTERNET OF THINGS, MONITORING, RASPBERRY PI, SENSOR, SERVER, JSON

Own publications:

1. Tvardovsky VG, Kravchenko OV "IoT-Solution for Monitoring and Analysis of Solid Waste Save System", IV International Scientific and Practical Conference, February 1-2, 2022 ., Pp. 180.
2. Valentyn Tvardovskyi, Olha Kravchenko, Svetlana Besedina. Design of IoT-solution for monitoring and analysis of the solid waste storage system. Technology Audit and Production Reserves, Vol. 2 No. 2 (64) (2022) Information and control systems. Information technologies. Pp. 6-10.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTTP – HyperText Transfer Protocol

CSS – Cascading Style Sheets

API – Application Programming Interface

URL – Uniform Resource Locator

REST – Representational State Transfer

ОС – Операційна система

UUID – Universally Unique Identifier

MQTT – Message Queue Telemetry Transport

ВСТУП

Актуальність теми. В епоху Інтернету речей багато пристроїв розробляються з нагоди покращити або ж оптимізувати загальні проблеми міст та селищ. До таких актуальних проблем відноситься й питання регулярності очищення сміттєвих контейнерів біля будинків:

- відсутність роздільного збору відходів, призводить до переповнення полігонів. Висока частота виїздів сміттєвозів, веде до підвищеного викиду CO2;
- збір статистики не автоматизований, а найчастіше відсутній зовсім, внаслідок цього оптимізація процесу збору відходів неможлива;
- відсутність даних рівня заповнення контейнерів, призводить до зайвих виїздів сміттєвозів і підвищення собівартості вивезення відходів для регіональних операторів і не вигідних тарифів для клієнтів;
- переповнення контейнерів призводить: до зростання антисанітарії, створення локальних звалищ, доступу тварин і птахів до відходів.

Які реалізації подібних IoT систем вже є? Серед перших у пошуках:

- CleanFLEX від компанії ECUBE LABS;
- SmartCity WSens від компанії Binology;
- BrighterBins від компанії з такою ж назвою.

Проаналізувавши дані щодо зазначених датчиків можна зробити висновок, що в цілому вони однакові в тому, що стосується безпосередньо самого датчика, а не всієї системи в цілому. Саме тому задача розробки IoT рішення є **актуальною**. В усіх трьох прикладах для моніторингу рівня наповненості ємностей використовується ультразвуковий датчик відстані, всі три використовують незмінні батарейки, а також слідкують за тим, чи не загорівся контейнер. В CleanFLEX також встановлений GPS для відслідковування розташування пристрою. Основна різниця між ними в тому функціоналі, що надає хмара, в конкретних моделях датчиків, що використовувались в розробках, а також в ціні.

Мета і завдання дослідження. Метою кваліфікаційної роботи магістра є розробка методу забезпечення процесу моніторингу та аналізу системи сміттєзберігання твердих відходів засобами IoT та проектування і програмна реалізація на його основі високоефективної системи IoT рішень. Для цього необхідно проаналізувати існуючі IoT рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів та спроектувати власний пристрій.

Основні завдання, що потрібно виконати для досягнення мети:

1. Аналіз датчиків та пристроїв, що можна використати для проектування пристрою.
2. Аналіз існуючих систем IoT рішень, завдяки яким можна отримати оцінку рівня заповненості ємності для твердих відходів.
3. Дослідження комунікаційних систем та технологій.
4. Проектування логічних зв’язків вузлів системи.
5. Розроблено структурну схему системи IoT пристрою.
6. Створено апаратне та програмне забезпечення системи IoT пристрою.
7. За матеріалами роботи опубліковано статтю та тези.

Об’єкт дослідження є IoT пристрої для контролю та моніторингу рівня наповненості ємностей для твердих відходів.

Предмет дослідження - апаратні та програмні засоби для розробки IoT рішень в сфері моніторингу та аналізу системи сміттєзберігання твердих відходів.

Методи дослідження. Під час вирішення основних завдань при розробці системи IoT рішень для моніторингу та аналізу системи сміттєзберігання твердих відходів використано теоретичні та емпіричні методи дослідження: аналіз та узагальнення наукової літератури, розробка програмного коду, проектування програмного забезпечення та графічного інтерфейсу користувача.

Практичне значення одержаних результатів. Результати здійснених у проекті досліджень можуть бути використані при проектуванні IoT пристроїв з подібним функціоналом, а спроектоване та розроблене рішення може бути завершене і доповнене для практичного використання і в комерційних цілях.

Апробація результатів. Основні положення і результати досліджень, викладені у проекті, пройшли апробацію на IV міжнародній науково-практичній конференції «Сучасні тенденції розвитку інформаційних систем і телекомунікаційних технологій», 1–2 лютого 2022 р. (Київ, Україна) [1] та опубліковані у журналі Technology Audit and Production Reserves, Vol. 2 No. 2(64) (2022) Information and control systems. Information technologies. Pp. 6-10 [2].

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДАТЧИКІВ ТА ПРИСТРОЇВ

1.1 Основа IoT пристрою

В епоху Інтернету речей багато платформ на базі мікроконтролерів або мікрокомп’ютерів розроблені для обслуговування програм та систем IoT. Мікрокомп’ютер має інтерфейс, до якого можна отримати доступ, підключивши його до монітора. Поширеною практикою є наявність у мікрокомп’ютера операційної системи (ОС). Мікроконтролер не має інтерфейсу, тому програма повинна бути записана на комп'ютері і завантажена на плату. Він має можливість зберігати і запускати лише одну програму за раз, але може бути перепрограмований декілька разів. Відповідно до аналізу сервісу Google Trends найчастіше користувачі в пошуках інформації по наступним трьом технологіям: Arduino, Raspberry Pi та Microbit.

Далі наведено (рис 1.1) динаміку популярності веб-пошуку по вищезазначеним критеріям за останні 5 років у всьому світі.

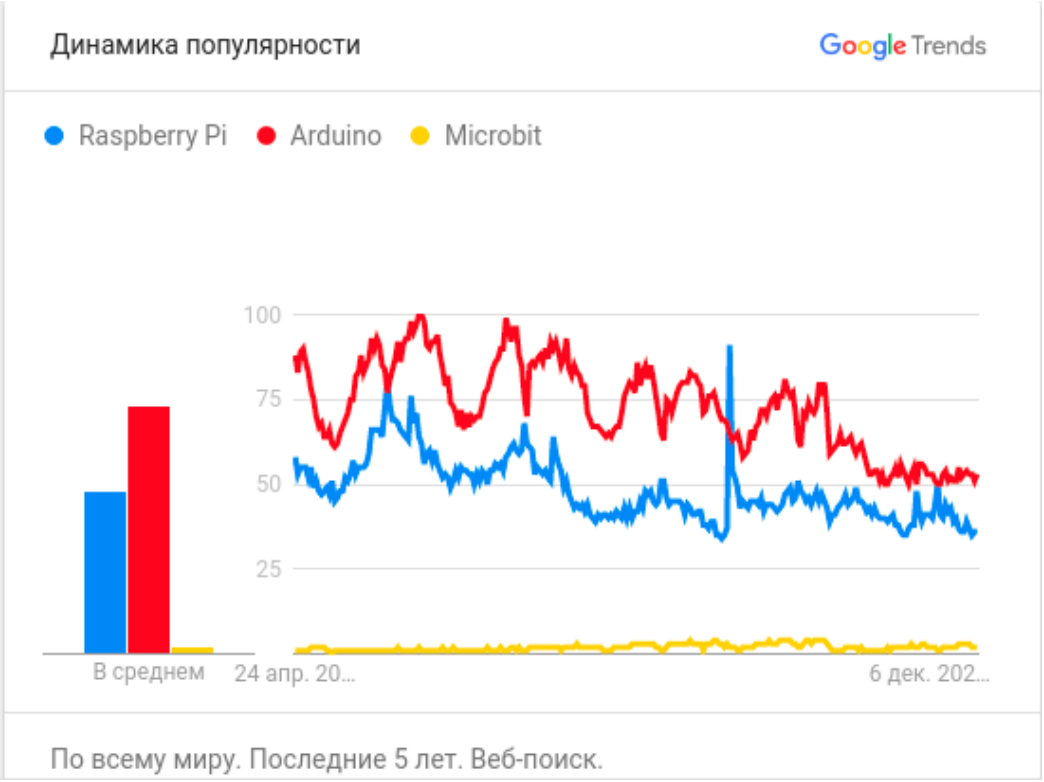


Рисунок 1.1. Динаміка популярності веб-пошуку в світі

Взявши дані з графіку за основу можна зробити висновок, що беззаперечним лідером популярності є Arduino, за ним йде Raspberry Pi і в порівнянні з попередніми двома варіантами Microbit абсолютно програє в популярності пошуку інформації щодо нього.

Не дивлячись на суцільне домінування Arduino в пошукових запитах - в період за 23-29 червня відбувся різких вибух популярності пошуку інформації про Raspberry Pi і також можна побачити, що з кожним роком різниця популярності двох основних конкурентів зменшується.

Нижче наведено теж саме порівняння, але по регіонам (рис 1.2). Звісно є й інші варіанти, крім зазначених трьох. Наприклад мікрокомп'ютери Orange Pi та Banana Pi.

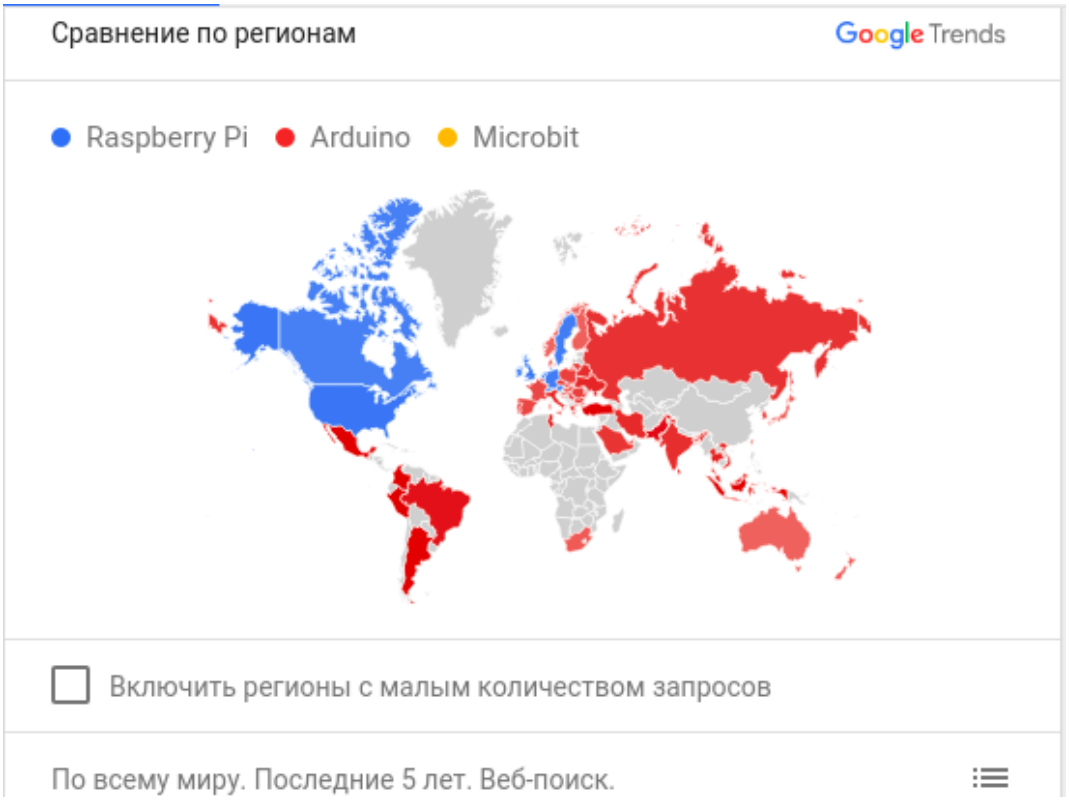


Рисунок 1.2. Порівняння популярності по регіонам веб-пошуку в світі

Далі наведено динаміку популярності пошукових запитів щодо інформації про ці мікрокомп'ютери в порівнянні з аутсайдером попередньої вибірки - Microbit (рисунок 1.3). Відповідно до графіку Microbit.

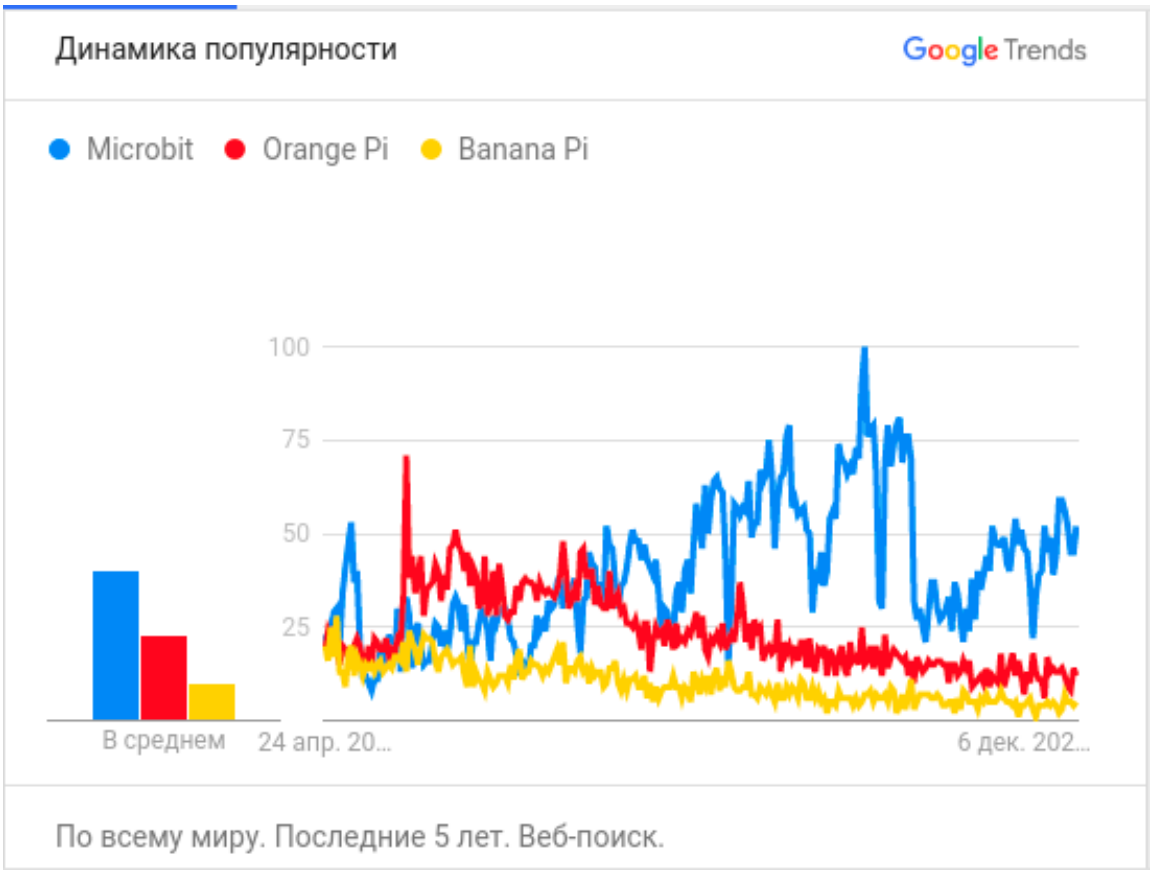


Рисунок 1.3. Динаміка популярності веб-пошуку в світі між менш відомими схемами

1.2 Arduino

Мікроконтролер Arduino призначений для розробки обладнання, оскільки він має багаті бібліотеки для різноманітних інтерфейсів датчиків. Для обсягу цієї статті порівняння специфікацій між платами Arduino обмежено лише платами на основі USB. Існує більше 15 плат на базі Arduino USB.

Причина розгляду лише плати на базі USB для порівняння специфікацій - більшість сучасних плат і платформ використовують USB для програмування, де вони працюють на змінних мікропрограмах через USB-зв'язок, без необхідності перезавантаження або підключення клавіатури для вибору програми для її запуску.

У таблиці 1.1 представлено більшість специфікації на базі Arduino USB.

Таблиця 1.1 – Специфікації на базі Arduino

Version	F1	F2 V	F3 MHZ	F4 Ana	F5 Dig	F6 Mem	F7 UART
101	Reg	3.3	32	6	14	*	-
Mega2560		5	16	4	54	**	4
Uno		5	16	6	14	**	1
Ethernet		5	16	6	14	**	-
Mega ADK		5	16	16	54	**	4
Nano	Mini	5	16	8	14	**	1
Gemma	Micro	3.3	8	1	3	**	-
Micro		5	16	12	20	**	1
MKR1000		3.3	48	8	8	*	1
Esplora		5	16	-	-	**	-
Leonardo		5	16	12	20	**	1
Yun		5	16	12	20	**	1
MKRZero		3.3	48	8	22	*	1
Zero		3.3	48	7	14	*	2
Due		3.3	84	14	54	*	4

Специфікація варіюється залежно від різних версій мікроконтролерів Arduino. Для порівняння специфікацій мікроконтролерів Arduino використовується сім функцій, а саме тип USB (позначений як F1), робоча напруга (F2), швидкість обробки (F3), аналогові входи / виводи (F4), цифрові входи / виводи (F5), пам'ять (F6) та порт UART (F7). З таблиці видно, що версії мікроконтролерів Arduino оснащені як звичайними, так і міні або мікро USB типами. Їх робочі напруги складають або 3,3В, або 5В. Швидкість обробки для версій мікроконтролерів Arduino знаходиться в діапазоні від 8 МГц до 84 МГц. Як правило, більшість з них використовує ATMEL AVR RiSC як MCU. Однак є

деякі з них, які використовують різні MCU, такі як ARM Cortex M для MKR1000, MKRZero, Due та Zero. Аналогові входи / виходи - це, як правило, порти ADC та DAC. Номери аналогових входів / виходів складають від 4 до 14, тоді як цифри входів / виходів починаються як мінімум від 3 до максимум до 54. Що стосується пам'яті, то "*" вказує, що версії мікроконтролерів Arduino мають SRAM та FLASH, тоді як '**' означає включення EEPROM в їх списки пам'яті. Більшість версій мікроконтролерів Arduino мають принаймні 1 порт UART, але Zero має 2 порти, а Mega2560, MegaADK та Due мають 4 порти.

1.3 Raspberry Pi

Що стосується Raspberry Pi, це мікрокомп’ютер, призначений для розробки програмного забезпечення. Потужні можливості обробки та мережі роблять його придатним для обробки відео / аудіо та як сервера. Для справедливого порівняння з усіма платформами в цій роботі, порівняння специфікацій між моделями Raspberry Pi адаптує ту саму структуру, що і порівняння специфікацій від мікроконтролера Arduino. Однак такі функції, як аналогові вводи-виводи (F4) та порт UART (F5), опущені та замінені SoC та мережею. Аналогові вводи-виводи опускаються, оскільки Raspberry Pi не потребує портів ADC / DAC, оскільки ці два блоки доступні всередині SoC. Що стосується порту UART, цифрові вводи-виводи для Raspberry Pi мають інтерфейс нижчого рівня для UART, який доступний через порт USB. У таблиці 1.2 представлені специфікації для всіх моделей Raspberry Pi, окрім 4-того покоління.

Специфікація варіюється залежно від різних версій моделей Raspberry Pi. Для порівняння специфікацій мікроконтролерів Arduino використовується сім функцій, а саме тип USB (позначений як F1), робоча напруга (F2), швидкість обробки (F3), SoC (F4), цифрові входи / виводи (F5), пам’ять (F6) та створення мереж (F7). З таблиці видно, що моделі Raspberry Pi оснащені мікро-USB різної кількості. Їх робочі напруги фіксовані на рівні 5В. Швидкість обробки моделей Raspberry Pi надзвичайно висока в порівнянні з мікроконтролерами Arduino, де їх значення коливаються від 700 МГц до 1,2 ГГц. Всі моделі A і B поколінь 1 та

1+, Compute і Zero використовують ARM11 як свій процесор, тоді як моделі A і B поколінь 2 і 3 використовують ARM Cortex A як свій процесор. SoC для всіх моделей Raspberry Pi - це SoC для обробки відео, які виробляє Broadcom. Кількість цифрових входів / виходів починається від 8 до 46.

Таблиця 1.2 – Специфікації моделей Raspberry Pi

Version	F1 USB	F2 V	F3 MHz	F4 SoC	F5 Di I/O	F6 Mem	F7 UART
Model A1	1	5	700	Video	8	256	-
Model A1+	1	5	700	Video	17	512	-
Model B1	2	5	700	Video	8	512	&
Model B1+	4	5	700	Video	17	512	&
Model B2	4	5	900	Video	17	1G	&
Model B2(1.2)	4	5	900	Video	17	1G	&
Model B3	4	5	1200	Video	17	1G	&&
Compute Module	1	5	700	Video	46	512	-
Zero	1	5	1000	Video	40	512	-

Вони мають SDRAM як зовнішню пам’ять розміром 256 МБ, 512 МБ або 1G. Що стосується мережі, то "&" вказує, що моделі Raspberry Pi мають швидкість обміну даними до 100 Мбіт/с Ethernet, тоді як "&&" означає, що вони мають до 1000 Мбіт/с Ethernet, бездротову мережу 802.11n та Bluetooth 4.1.

1.4 Microbit

Плата Microbit призначена для розробки обладнання. Він невеликий, але досить потужний, оскільки має мало аналогових входів / виходів і досить велику кількість цифрових входів / виходів, які можна використовувати для інтерфейсів датчиків; він також використовує ARM Cortex M0 як їх мікроконтролер і може бути запрограмований через веб платформу

TouchDevelop, розроблену Microsoft. Функції, що використовуються в специфікації Microbit, такі ж, як функції моделей Raspberry Pi.

З таблиці 1.3 видно, що моделі Microbit оснащені 1 мікро USB. Їх робочі напруги фіксовані на рівні 3В. Швидкість обробки для моделей становить 48 МГц. SoC для Microbit - це RF SoC, які виробляє Nordic. Цифрових входів / виходів 20. Що стосується пам’яті, то «*» вказує, що Microbit має SRAM та Flash, які вбудовані всередині RF SoC. Microbit включає Bluetooth як мережеву функцію. Samsung створила додаток для Android, яке дозволяє програмувати Microbit бездротово за допомогою смартфона з можливістю Bluetooth через Nordic RF SoC.

Таблиця 1.3 – Специфікація Microbit

Version	F1 USB	F2 V	F3 MHz	F4 SoC	F5 Di I/O	F6 Mem	F7 Nwk
Microbit	1	3	48	RF	20	*	Bluerooth

1.5 Датчики

Для вимірювання відстані від поверхні баку до сміття по відношенню до відстані від поверхні баку до максимальної глибини необхідно підібрати датчики відстані. Для перевірки положення кришки або ж самого бака потрібен гіроскоп. І врешті-решт для перевірки стану бака на предмет займання - датчик температури та газу або ж диму чи просто вогню.

Ультразвуковий датчик відстані HC-SR04 - це простий далекомір з великим діапазоном вимірюваних відстаней. Ультразвуковий далекомір генерує звукові імпульси на частоті 40 кГц і слухає відлуння. За часом поширення звукової хвилі туди і назад можна однозначно визначити відстань до об'єкта. Цей далекомір може служити датчиком для робота, завдяки якому він зможе визначати відстані до об'єктів або об'їжджати перешкоди. Його можна також використовувати в якості датчика для сигналізації, що спрацьовує при наближенні об'єктів.

Характеристики:

- напруга харчування: 5 В;
- споживання в режимі тиші: 2 мА;
- споживання при роботі: 15 мА;
- діапазон відстаней: 2-400 см;
- ефективний кут спостереження: 15 °;
- робочий кут спостереження: 30 °.

Нижче зображено діаграму направленості сигналу датчика (рис 1.4) та безпосередньо зовнішній вигляд датчика (рис 1.5).

Рисунок 1.4. Діаграма направленості HC-SR04

Рисунок 1.5. HC-SR04

Модуль лазерного датчика (Laser Sensor) від Waveshare призначений для виявлення предметів і різних перешкод на відстані до 1,5 м. Простий у використанні і підключенні (рис 1.6).

Особливості:

- ефективна дальність спрацьовування: до 1,5 м;
- напруга: 2,5 - 5V;
- включає в себе схему посилення;
- індикація прийнятого сигналу;
- є монтажні отвори 2мм;
- розміри: 47,7 x 17,9мм.

Рисунок 1.6. Waveshare Laser Sensor

Інфрачервоний далекомір дозволить визначити відстань до об'єкта в межах 10 - 150 см. У датчиках Sharp встановлений інфрачервоний (IR) світлодіод (LED) з лінзою, який випромінює вузький світловий промінь. Відбитий від об'єкта промінь прямує через іншу лінзу на позиційно-чутливий фотоелемент (Position-Sensitive Detector, PSD). Від місця розташування падаючого на PSD променя залежить його провідність. Провідність перетворюється в напругу і, наприклад, оцифровивая його аналого-цифровим перетворювачем мікроконтролера, можна обчислити відстань.

Вихід інфрачервоного сенсора відстані Sharp обернено пропорційний - із збільшенням відстані його значення повільно зменшується. Вид графіка залежності між відстанню і напругою. Датчики, в залежності від їх типу, мають кордони виміру, в межах яких їх вихід може бути визнаний надійним. Вимірювання максимального реального відстані обмежують два фактори:

зменшення інтенсивності відбитого світла і неможливість PSD реєструвати незначні зміни місця розташування відображеного променя.

Технічні характеристики інфрачервоного далекоміра Sharp (рис 1.7):

- робоча напруга: 4,5 - 5,5 В;
- максимальний споживаний струм: 40 мА (типовий - 30 мА);
- тип вихідного сигналу: аналоговий;
- диференціальне напруга, більша діапазону розпізнавання відстані: 2,0 В;
- час відгуку: 38 ± 10 мс.

Діапазон роботи:

- датчик GP2Y0A41SK0F: 4 - 30 см;
- датчик GP2Y0A021YK0F: 10 см - 80 см;
- датчик GP2Y0A02YK0F: 20 см - 150 см.

Гіроскоп акселерометр GY-521 використовується для вимірювання прискорення предмета, яке воно набуває при зміщенні щодо свого нульового положення, також для визначення положення в просторі і для вимірювання температури навколишнього середовища.

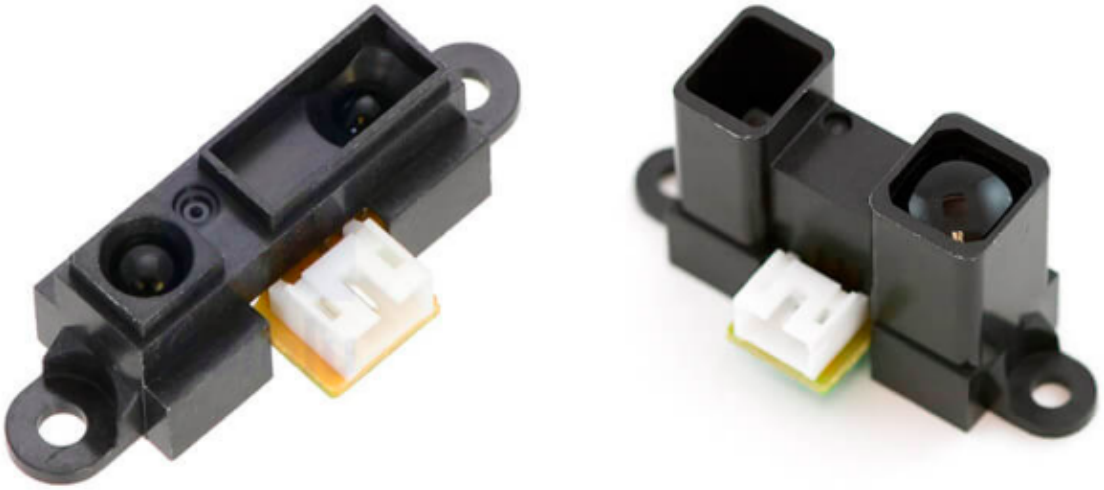


Рисунок 1.7. GP2Y0A41SK0F та GP2Y0A02YK0F

Практичне застосування – вимірювання вібрації, руху, ударів, кутів крену, кутів повороту. Гіроскоп акселерометр GY-521 може використовуватися в

авіамоделях на мікроконтролерах. Гіроскоп акселерометр GY-521 використовується для вимірювання прискорення предмета, яке воно набуває при зміщенні щодо свого нульового положення, також для визначення положення в просторі і для вимірювання температури навколишнього середовища.

Гіроскоп акселерометр GY-521, за замовчуванням, не йде з припаяними штировими контактами. У комплекті з датчиком йдуть: прямий перехідник 2,54 мм, кутовий перехідник 2,54 мм. Перед початком роботи один з перехідників (який більш зручний) потрібно припаяти до клем датчика. На платі датчика є два отвори для закріплення на плоскій поверхні за допомогою болтів або шурупів.

Для використання датчика потрібно підключити його до Arduino контролера або іншому мікропроцесорному керуючого пристрою з інтерфейсом IIC. Потім підключити до датчика харчування. Для нормальної роботи датчика з Arduino контролером знадобиться спеціальна бібліотека. Бібліотеку потрібно самостійно в інтернеті.

Гіроскоп акселерометр GY-521 (рис 1.8) має вісім контактів для підключення до контролера, харчування, периферійних пристроїв, адресації:

- харчування: VCC (напруга живлення), GND (загальний контакт);
- інтерфейс IIC для з'єднання з контролером: SCL, SDA;
- інтерфейс IIC для з'єднання з периферійними пристроями (наприклад, магнетометром): XDA, XCL. Керуючим пристроєм, по цій шині IIC буде сам GY-521;
- ADO: пін адресації. Використовується, коли підключається до контролера більше одного GY-521;
- INT: зовнішнє переривання.

Характеристики:

- модель: GY-521;
- датчик зібраний на мікросхемі: MCU-6050;
- режими для акселерометра: ±2g, ±4g, ±6g, ±8g, ±16g;
- режими для гіроскопа: ±250°, ±500°, ±1000°, ±2000°;

- ширина шини IIC: 16 біт;
- напруга живлення: 3,3 – 5 В;
- розміри: 20 х 15 х 3 мм.

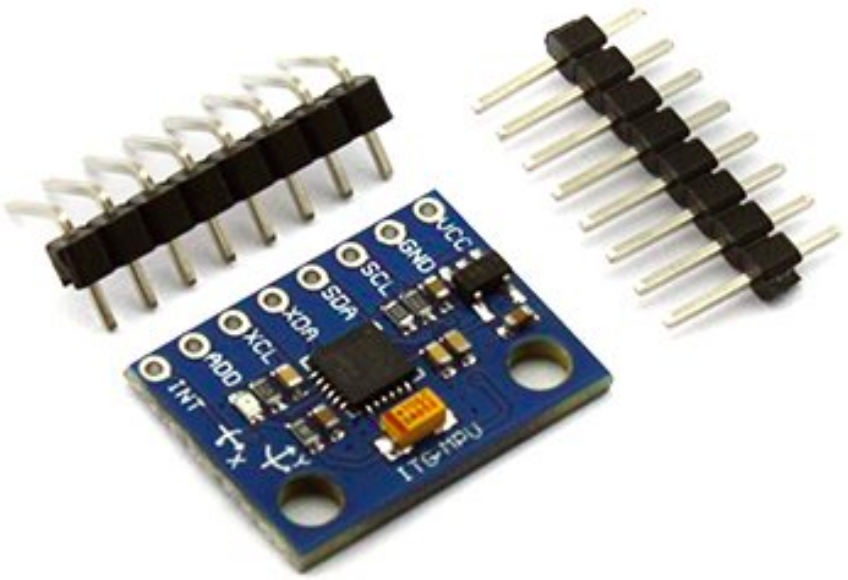


Рисунок 1.8. GY-521

Гіроскоп з лінійки Troyka-модулів (рис 1.9) виміряє кутову швидкість навколо власних осей X, Y і Z. Це властивість нагоді для стабілізації літального апарату по літаковим кутах: тангаж, крен і рискання. Гіроскоп спільно з акселерометром застосовується для відстеження кута повороту в сучасному смартфоні. Гіроскоп заснований на чіпі L3G4200D і являє собою мініатюрний датчик переміщень в 3D просторі, виконаний за технологією MEMS компанії STMicroelectronics. Мікросхема має вбудований датчик температури для точної роботи сенсора навіть в екстремальних умовах.

На модулі встановлено регулятор напруги і I²C-буфер для узгодження рівнів логіки. Тому можна використовувати гіроскоп з керуючою електронікою напругою 3,3 ... 5 В. Гіроскоп підключається до керуючої електроніці через дві пари Troyka-контактів: одна для подачі напруги і друга для підключення до шини I²C.

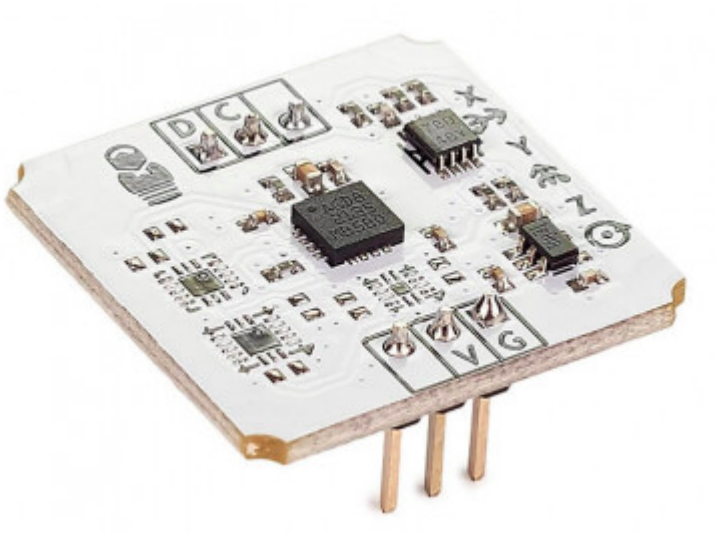


Рисунок 1.9. L3G4200D

Модуль датчика вологості і температури DHT-11 (рис 1.10). Датчик DHT-11 є цифровим датчиком температури і вологості. Працює датчик DHT11 тільки при позитивних температурах від 0 до 50 ° С. Відносну вологість повітря вимірює від 20% до 90%.

Характеристики:

- визначення вологості: 20 - 90% з точністю 5%;
- визначення температури: 0 - 50 ° С з точністю 2%;
- частота опитування датчика: не більше 1 Hz (не більше одного разу в 1 секунду);
- 3-контактний штирьовий інтерфейс для підключення до мікроконтролеру;
- АЦП (аналогово-цифровий перетворювач);
- напруга живлення: 3 - 5.5V;
- габарити: 32 x 14mm;
- вага: 8g;
- індикатор живлення: червоний;
- вихід: цифровий.

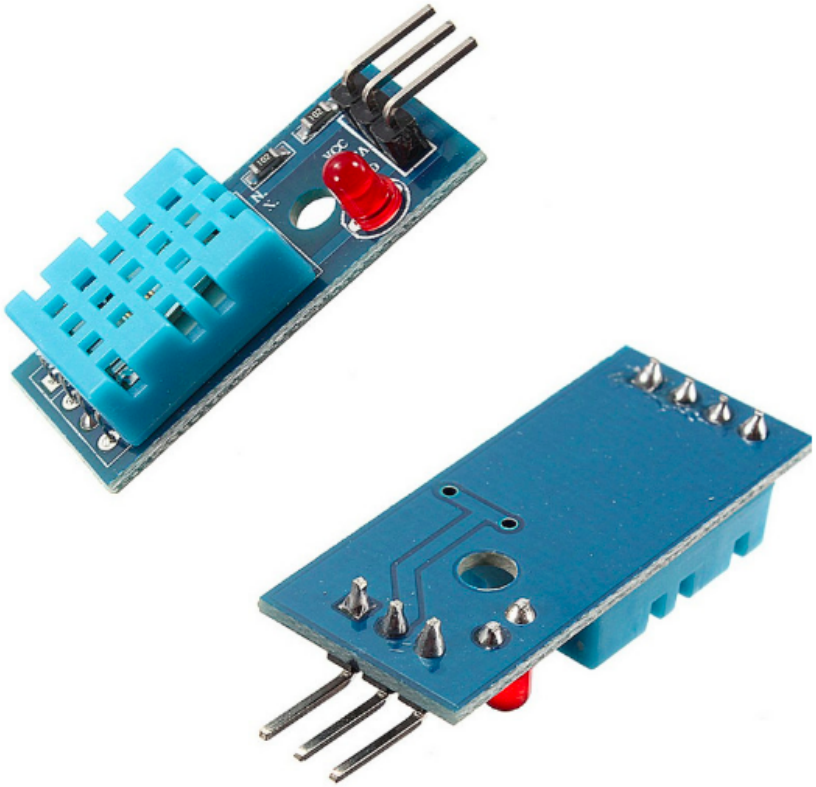


Рисунок 1.10. DHT-11

DHT22 Temperature-Humidity Sensor - якісний модуль від Waveshare, заснований на ємнісному датчику DHT22 і термісторі, призначений для вимірювання температури і вологості навколишнього середовища (рис 1.11). Містить в собі АЦП для перетворення аналогових значень в цифрові. Крім вбудованого АЦП основна відмінність від попередника DHT-11 - поліпшена точність, а так само можливість роботи при негативних температурах. Для зручного підключення модуля в комплект входить 3-піновий джампер.

Характеристики:

- робоча напруга: від 3.3V до 5.5V;
- діапазон вимірюваних температур: від -40 ° C до 80 ° C;
- точність вимірювання температури: ± 0.5 ° C;
- крок вимірювання температури: 0.1 ° C;
- діапазон вимірювальної вологості: від 0% RH до 99.9% RH;
- точність вимірювання вологості: ± 2% RH (при 25 ° C);
- крок вимірювання вологості: 0.1%;

- рекомендовані умови зберігання: t ° від 10 ° C до 40 ° C, вологість - 60% або нижче.

Рисунок 1.11. DHT22

AM2320 цифровий датчик температури і вологості , в якому температура і вологість вимірюються з використанням спеціальної технології захоплення для забезпечення високої надійності і відмінною довгостроковій стабільності.

Датчик включає в себе ємнісний елемент вимірювання вологості і вбудовані високоточні пристрої вимірювання температури, і з'єднаний з високопродуктивними процесорами. Продукт має відмінну якість, швидку реакцію, захист від перешкод, невисоку вартість і інші переваги.

Специфікація:

- інтерфейс: I2C;
- харчування: 3,1 ... 5,5 (номінально 5В);
- струм: у режимі очікування - 8 ... 10 мкА, а у режимі вимірювань - 350 мкА, піковий до 950 мкА;
- вимірювання температури:
    - допустимий діапазон: -40 ° C .. 80 ° C;
    - максимальна похибка: ± 0,5 ° C;
    - дозвіл шкали: 0,1 ° C.
- вимірювання відносної вологості:

- допустимий діапазон: 0 ... 99,9%;
- максимальна похибка: ± 3% при t = 25 ° C;
- дозвіл шкали: 0.1%;
- догляд показань: 0,5% / рік;
- мінімальний час між зчитування показань: 2 сек.

- габарити: 30x30x6,5 мм;
- вага: 4 гр.

Далі наведено зображення датчика (рис 1.12).

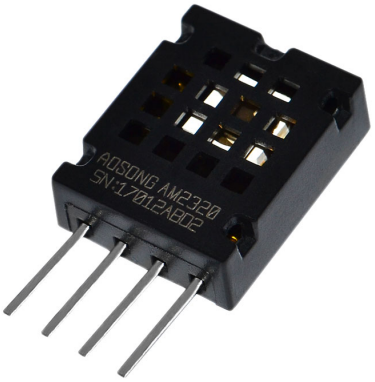


Рисунок 1.12. AM2320

Модуль датчика MQ-8 призначений для виявлення водню (H2) і коксового газу (рис 1.13). Діапазон чутливості: від 100 до 10000 ppm, який регулюється за допомогою потенціометра, розміщеного на платі.

Характеристики:

- гази, що відслідковуються: водень і коксовий газ;
- напруга: 5V;
- діапазон чутливості: від 100 до 10000 ppm;
- час відгуку: до 20 секунд;
- робоча температура: від -10 ° C до + 50 ° C, вологість: ≤ 95% RH;
- датчик конфігурації "A".

Датчик газу MQ-2 визначає наявність парів зріджених вуглеводневих газів, пропану, водню (рис 1.14). Вихідна напруга підвищується разом зі

збільшенням концентрації вимірюваних газів. Датчик характеризується швидким відгуком та відновленням. Має регульовану чутливість і індикатор вихідного сигналу.

Характеристики:

- напруга: 2,5 В ~ 5,0 В;
- розмір: 40,0 мм * 21,0 мм;
- діаметр монтажних отворів: 2,0 мм.

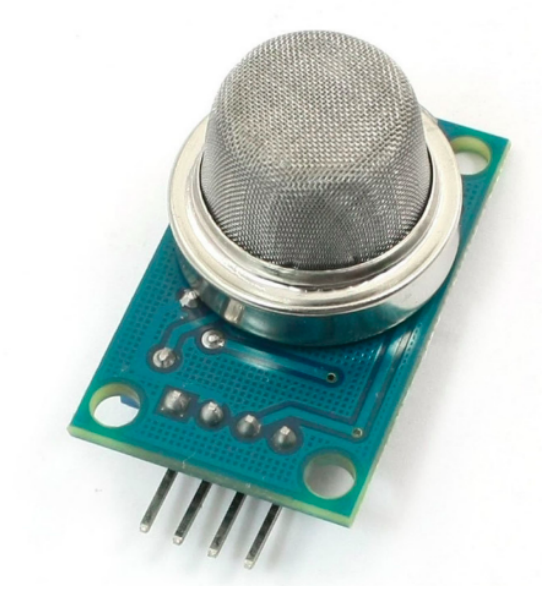


Рисунок 1.13. MQ-8

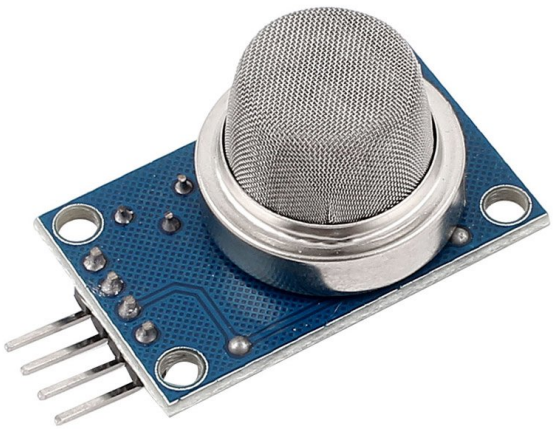


Рисунок 1.14. MQ-2

"KY-026" - датчик полум'я реагує на інфрачервоне випромінювання (відкритий вогонь) і найбільш чутливий до довжин хвиль від 760 нм до 1100 нм (рис 1.15). Цей детектор вогню має два виходи - цифровий та аналоговий. На платі є 2 світлодіода - індикації харчування і індикації виходу з компаратора при виявленні вогню. Модуль виконаний на мікросхемі LM393. При відсутності полум'я на аналоговому виході є напруга 4,2 В, а при появі вогню на відстані 1 метр, на аналоговому виході - 0,2 В (при напрузі живлення 5 В).

Характеристики:

- кут виявлення полум'я, град: 60;
- дальність виявлення вогню, м: 1;
- напруга живлення, В: 3 5.5;
- розміри (довжина х ширина), мм: 36 х 16.

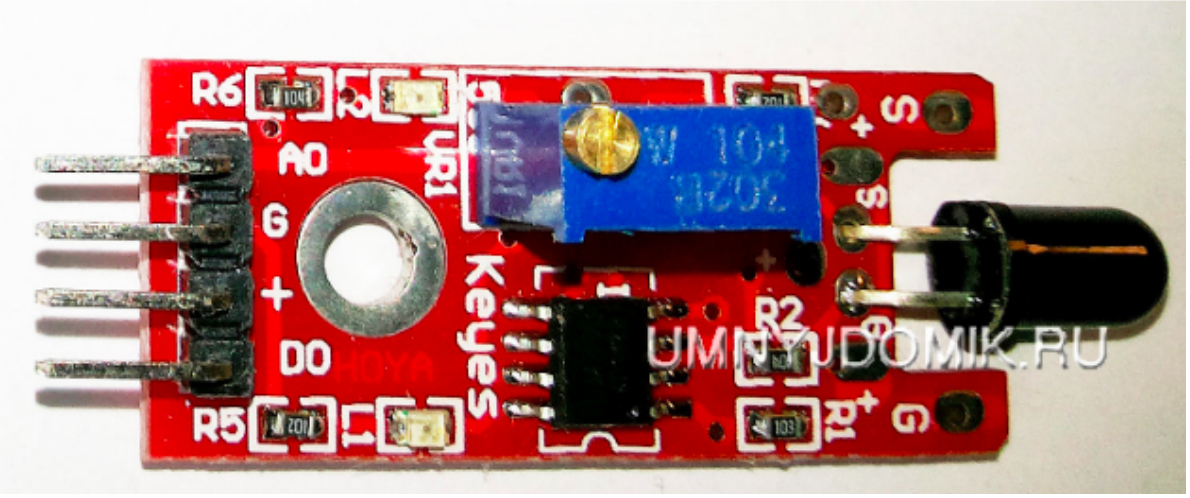


Рисунок 1.15. KY-026

1.6. Метод для моніторингу та аналізу системи сміттєзберігання твердих відходів

Проаналізувавши основні існуючі компоненти для розробки IoT проектів, необхідно розробити пристрій для аналізу заповненості сміттєвих контейнерів з можливістю передачі даних про стан заповненості контейнера, положення кришки або ж самого контейнера для випадків, коли контейнер буде перевернутий або ж кришка контейнера буде піднята, оскільки пристрій буде розташований на кришці із внутрішньої сторони контейнера. Також пристрій

має бути оснащений можливістю перевірки контейнера на предмет підпалу для запобігання потенційних нещасних випадків та викиду в атмосферу шкідливих відходів. Робота пристрою описується методом для моніторингу та аналізу системи сміттєзберігання твердих відходів. Він складається з наступних кроків:

Крок 1. Перевірка стартових значень. Відповідно до стартових показників датчиків в подальшому визначається відходження від норми стану контейнера.

Крок 2. Перевірка стану контейнера. Для забезпечення точності перевірки заповненості контейнера зчитуються показники датчиків на предмет відпалу чи виділення небезпечних речовин, а також положення контейнера відносно початкового.

Крок 3. Аналіз показників датчика MQ-2. Якщо значення датчика вказують на наявність диму, то надсилається повідомлення про небезпеку.

Крок 4. Аналіз показників датчика GY-521. Якщо значення датчика вказують на невідповідність стартовому положення, то надсилається повідомлення про некоректне положення контейнера, через що неможливо правильно виміряти стан заповненості контейнера.

Крок 5. Перевірка заповненості контейнера. Якщо датчик MQ-2 не сигналізує про небезпеку, а датчик GY-521 вказує на коректне положення контейнера, тоді відбувається зчитування значень з датчика HC-SR04 і на основі даних робиться висновок щодо заповненості контейнера.

Далі наведено блок схему роботи програмного забезпечення приладу (рис 1.16).

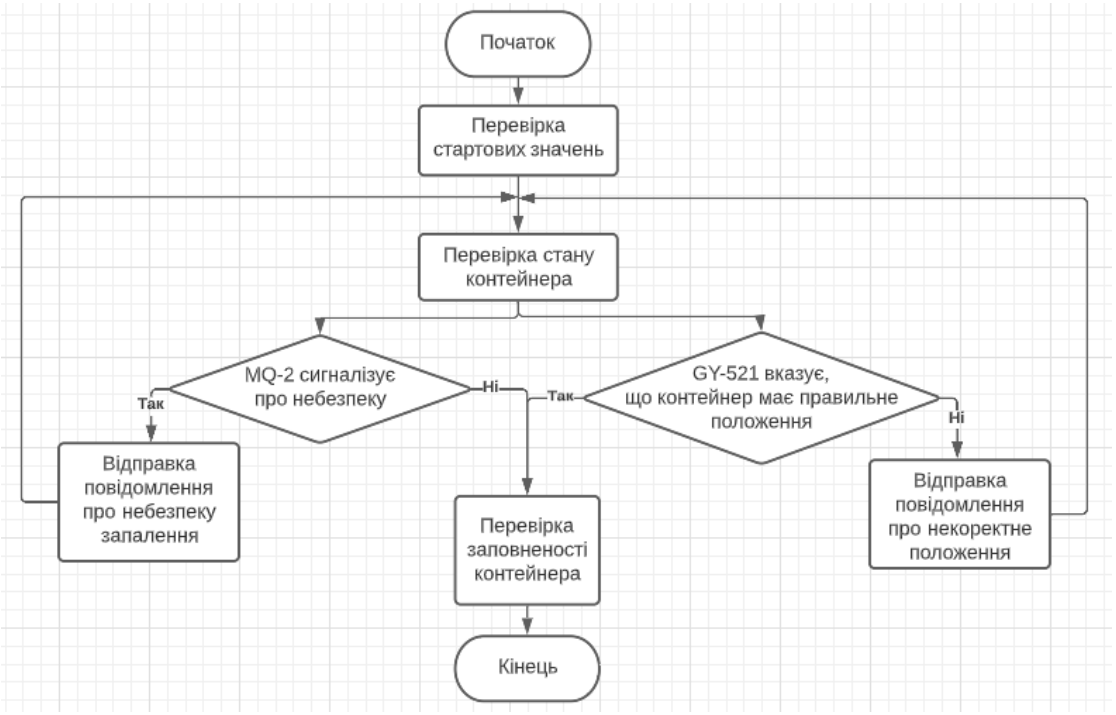


Рисунок 1.16. Блок схема роботи програмного зебезпечення у відповідності до методу для моніторингу та аналізу системи сміттєзберігання твердих відходів

Висновок до першого розділу

У першому розділі виконано аналіз датчиків та пристроїв, що використовуються або можуть бути використані для розробки проектів Інтернету речей і зокрема для розробки проекту для моніторингу та аналізу системи сміттєзберігання твердих відходів.

Проаналізовано динаміку популярності використання або зацікавленості в мікрокомп’ютерах та мікроконтролерах. А також розподілення по регіонам. Найпопулярнішими виявились, відповідно до Google Trends, мікроконтролер Arduino та мікрокомп’ютер Raspberry Pi. Проведено аналіз специфікацій на базі Arduino, Raspberry Pi та Microbit. В таблицях наведено відмінності характеристик в різних моделях в межах специфікацій кожної бази окремо.

Проведено аналіз існуючих датчиків для вимірювання відстані на основі ультразвуку, інфрачервоного світла та лазера, а також аналіз датчиків полум’я, газу, температури, вологості, гіроскопів та акселерометрів. Розроблено метод для моніторингу та аналізу системи сміттєзберігання твердих відходів.

РОЗДІЛ 2. ЗАСОБИ ЗВ’ЯЗКУ І ПРОГРАМНІ ТЕХНОЛОГІЇ IOT РІШЕННЯ ДЛЯ МОНІТОРИНГУ ТА АНАЛІЗУ СИСТЕМИ СМІТТЄЗБЕРІГАННЯ ТВЕРДИХ ВІДХОДІВ

2.1 Засоби зв’язку для IoT проектів

Хоча "i" в Інтернеті речей означає Інтернет, у нас є різні види мереж, доступних для зв'язку між пристроями та з платформою. Вибір правильної мережевої технології залежить від характеристик та вимог проекту. У проекті IoT зазвичай використовується більше однієї технології. Основними факторами, які слід враховувати при виборі мережевої технології для проектів IoT, є:

- поширення пристроїв: Якщо пристрої знаходяться в одній і тій же зоні, як будівля, промисловий завод або навіть мікрорайон, такі технології короткого діапазону, як Wi-Fi, ZigBee буде хорошим вибором. В іншому випадку, якщо пристрої широко розповсюджені в місті чи країні, довгий діапазон технологій, таких як LoRa або Sigfox, може бути правильним вибором. Прикладом спілкування LoraWan у містах є розміщення 20 000 розумних лічильників води у місті Кастельон компанією IoTsens;
- необхідна швидкість передачі даних: Деякі мережеві протоколи не підходять залежно від обсягу даних, які надсилають пристрої. Наприклад, Sigfox та LoRa не забезпечують достатню пропускну здатність, коли датчик повинен щохвилини передавати температуру приміщення або стан автостоянки;
- покриття мережі: Ви можете розгорнути власну мережу для проекту LoRa, включаючи використання сторонніх мереж LoRa (TTN) або використовувати доступну мережу, надану сторонніми сторонами (2G, Sigfox, NB-IoT тощо).

LoRaWAN (Long Range wide-area networks, глобальна мережа великого радіусу дії) - найбільш відомий апаратний протокол LoRa, який призначений для управління зв'язком між LPWAN-шлюзами і кінцевими пристроями. LoRaWAN базується на топології «зірка». Безліч пристроїв по бездротовому

з'єднанню передають дані не на один шлюз, а відразу на кілька. Підключення між пристроями і шлюзами здійснюється на двосторонній основі. Зв'язок між шлюзами здійснюється через бездротові рішення, які використовують широкосмуговий модуляцію LoRa або FSK.

Потім шлюзи, які отримали інформацію, перенаправляють отримані пакети від кінцевого вузла до хмарного мережному серверу, підключеного через мобільний або супутниковий зв'язок, провідний або бездротової ШПД. Звідти дані надходять на сервери додатків.

Використання декількох шлюзів зручно тим, що кінцеві вузли не мають прив'язки. Це дозволяє гарантувати передачу інформації і контролювати пристрої, що знаходяться в русі. Наприклад, бездротові маячки, прикріплені до вантажних контейнерів, що переміщуються на тривалі відстані, зможуть без проблем обмінюватися даними так як не мають прив'язки до одного шлюзу. Швидкість передачі даних між кінцевими пристроями і шлюзами варіюється від 0,3 до 50 кбіт / сек і може регулюватися самими шлюзами в залежності від сценаріїв використання.

NB-IoT (Narrow Band Internet of Things) - це стандарт стільникового зв'язку для пристроїв телеметрії, заснований на LTE і передбачає передачу невеликих обсягів даних. Стандарт розроблений консорціумом 3GPP в рамках робіт над мережами нового покоління. Першу робочу версію специфікації представили в червні 2016 року.

NB-IoT відноситься до стандарту LPWA (Low Power Wide Area), призначеному для M2M (Machine-to-Machine) додатків, які вимагають низкоскоростной передачі даних і роботи в автоматичному режимі протягом тривалого періоду часу, в тому числі у віддалених або важкодоступних місцях. NB-IoT - це окремо існуюча «гілка» на базі всім відомої технології LTE. Це саме частина ієрархії LTE-мереж, але зі своїми особливостями. Технологія NB-IoT багато успадкувала від LTE, починаючи з фізичної структури радіосигналу і закінчуючи архітектурою самої мережі. Вона створювалася з прицілом на

застосування в умовах більш низького рівня сигналу і більш високого рівня шумів з урахуванням економії ресурсу батареї.

Особливість NB-IoT в тому, що вона здатна передавати невеликі повідомлення від різних датчиків і приладів. Передача важкого контенту на зразок відео або аудіо в цьому випадку не застосовується.

Можливості:

- NB-IoT обмежений загальною смугою одного RB шириною в 180кГц;
- родючості пристрою, що використовує NB-IoT, має в своєму розпорядженні тільки одного антеною;
- передача і прийом рознесені за часом (полудуплекс);
- є можливість передавати дані по напрямку Uplink на одній піднесе;
- обмежені типи модуляцій, використовуються BPSK і QPSK;
- можуть застосовуватися переповтори переданого сигналу.

Sigfox - це глобальна мережа для Інтернету речей (IoT), створена у Франції. Ця технологія дозволяє пристроям спілкуватися недорого, безпечно і на великих відстанях з мінімальним енергоспоживанням. Sigfox використовує смугу частот 868 МГц і забезпечує двосторонній зв'язок з обмеженою кількістю зворотних повідомлень. У Чеській Республіці вона має великі зони покриття, тому добре підходить для дистанційного керування і моніторингу.

2.2 Технології серверної частини

Технологій для розробки серверної частини є безліч. Немає жодної причини, чому ми не можемо вибрати мову так само, як це робимо для настільного проекту. Якщо Raspberry Pi працює під управлінням Linux, його поведінка не так сильно відрізняється від робочого столу.

На той час, коли дебати про мови пробиваються до серверів, там теж немає різниці. Вони розмовляють з концентраторами та датчиками - як правило, з якоюсь архітектурою мікросервісу - тоді дані передаються у стандартні бази даних.

Хоча звичні підозрілі в популярних мовах вже домінують у просторі IoT, опитування Eclipse (https://iot.eclipse.org/) виявило щонайменше 14 різних мов,

про які згадували 5% і більше розробників. Робота з відкритим кодом IoT у проекті Eclipse розширюється, і зараз існує більше 20 різних проектів, що охоплюють мільйони рядків коду. Багато є в Java, але в Python, JavaScript, C та інших важливих фреймворках є багато.

Найкращим вибором опитування Eclipse та іншого опитування embedded-computing.com стала Java - результат, що не дивно для мови, котра все ще відома як "пишемо раз, використовуємо де завгодно". Оригінальний проект був спрямований на телевізійні приставки, один з перших доменів для настільних обчислень.

Переваги Java добре відомі. Розробники можуть створювати та налагоджувати код на своєму робочому столі, а потім переміщати його на будь-який чіп за допомогою віртуальної машини Java. Це означає, що код може працювати не тільки в місцях, де поширені JVM (сервери та смартфони), але і на найменших машинах. Java ME, або мікровипуск, був доступний на невеликих телефонних апаратах та інших вбудованих пристроях з моменту затвердження специфікації у 2000 році. Він заощадив простір завдяки дуже обмеженому набору бібліотек класів та інших інструментів. Сьогодні основна увага приділяється Java SE Embedded, яка набагато ближче за своїми можливостями до Standard Edition. Розробники можуть використовувати найновіші функції платформи Java 8, а потім перемістити свій код на менший вбудований пристрій.

Більша частина економії обчислювальних ресурсів за допомогою Java SE Embedded відбувається за рахунок вилучення класів, необхідних для відображення інформації, коли машини можна налаштувати на безголову роботу без монітора або клавіатури. Весь зв’язок йде через мережу. Існує безліч проектів з відкритим кодом, таких як Pi4J та BlueJ, які показують, як вбудована версія Java працює добре, навіть на чіпах, які здаються обмеженими. Окрім того, Java має низку бібліотек, за допомогою яких можна обробляти інформацію з датчиків і легко використовувати дані у зв’язці з сучасними технологіями розробки серверів на Java. Наприклад у поєднанні зі Spring Boot.

Що ж стосовно всім відомого C ? Синтаксис заповнений розділовими знаками, і можна зробити мільйон різних маленьких помилок, але мова все ще є першим вибором для багатьох програмістів, які пишуть для найнижчого рівня програмного забезпечення, найближчого до апаратного. Мова нічого не приховує від розробника, і це означає, що можна розібратись детально з кожною частиною коду, щоб витягнути найкращу продуктивність із недостатньо потужних пристроїв. Кожен біт можна перегорнути. Доступне кожне значення в стеку. Більш просунуті або більші пристрої з повноцінними операційними системами все ще використовують велику кількість коду C, додав він, але сказав, що інші мови, такі як Java, починають використовуватися так само часто. Коли смартфон походить від Apple, більша частина програмування все ще виконується в Objective C, але це, ймовірно, буде поступово замінено Swift.

Коли маленькі пристрої мають достатньо пам’яті та обчислювальної потужності, розробники можуть вільно вибирати мову, яка полегшує їхнє життя і яка все частіше виявляється Python. Python - це обрана мова для одного з найпопулярніших мікрокомп’ютерів на ринку, Raspberry Pi.
Є також версії, розроблені на ще менші розміри. Плата та програмний пакет MicroPython - це невеликий мікроконтролер, оптимізований для роботи Python на невеликій платі площею лише кілька квадратних дюймів.

Хоча багато хто досі думає про JavaScript як про мову, яка з’являється на веб-сторінках із полями сповіщень, порівняно нова популярність мови на сервері робить її напрочуд популярним вибором для додатків IoT. Повні 41,8 відсотка розробників в опитуванні Eclipse обрали JavaScript, а 31,5 відсотка зазначили, що вони використовують Node.js у своїх проектах.Значна частина цієї роботи зосереджена на серверах та шлюзах або концентраторах, які збирають інформацію, а потім зберігають її. Менші розумні концентратори та датчики, що працюють під Linux, зазвичай можуть запускати Node.js.

Але навіть якщо більша частина коду Node.js працює на більших машинах, є певні зусилля, спрямовані на те, щоб довести його до менших. Espruino та Tessel - два приклади мікроконтролерів, які запускають JavaScript з

самого початку. Наприклад, Tessel побудований навколо Node.js, що полегшує веб-розробникам перехід до IoT без вивчення нової мови.

Хоча Swift все ще в основному використовується для створення додатків для пристроїв Apple від iOS та macOS, переважання цих машин означає, що він часто є частиною стеку IoT. Якщо потрібно, щоб якісь речі взаємодіяли з iPhone або iPad, то можна написати програму на мові Swift (або, можливо, її попередницю, Objective C). Є й інші вагомі причини працювати в цьому просторі. Apple хоче зробити свої пристрої iOS центрами домашньої мережі датчиків, тому вона створює бібліотеки та інфраструктуру, які виконують значну частину роботи. Ці бібліотеки є основою платформи HomeKit, яка забезпечує підтримку інтеграції каналів даних із мережі сумісних пристроїв. Це означає, що можна зосередитися на деталях свого завдання і залишити більшу частину накладних витрат на інтеграцію на HomeKit.

PHP - мова може бути першим вибором для блогів та прототипів веб-сайтів, але вона також напрочуд популярна в IoT. Це та мова, яку розробники найчастіше згадують в опитуванні Eclipse; 11,2 відсотка заявляють, що включають PHP-код у свій стек. Ряд розробників Raspberry Pi говорять про запуск повного стеку LAMP з Apache, MySQL та PHP, що працюють поверх Linux. Вони ефективно перетворюють парадигму і перетворюють найнижче в Інтернеті на повноцінний веб-сервер. Якщо Raspberry Pi має достатньо запасних циклів - і це часто буває - тоді розміщення стека LAMP на чіпі спрощує розробку. Весь розроблений за останні 20 років серверний код може знайти дім у невеликому датчику. Це може здатися іграшкою поруч із повною силою C, але якщо це швидко й легко робить необхідно задачу, чому б їй не використовувати?

Крім основної мови програмування - необхідно обрати комунікаційні протоколи, наприклад MQTT, REST (HTTP), XMPP чи WebSockets, тощо.
MQTT (англ. Message Queue Telemetry Transport) — спрощений мережевий протокол, що працює на TCP/IP. Використовується для обміну повідомленнями між пристроями за принципом видавець-підписник:

- простий у використанні. Протокол є програмним блоком без зайвої функціональності, що може бути легко вбудований в будь-яку складну систему;
- зручний для більшості рішень з датчиками. Дає можливість пристроям виходити на зв'язок і публікувати повідомлення, які не були заздалегідь відомі або визначені;
- легкий у адмініструванні;
- низьке навантаження на канал зв'язку;
- робота в умовах постійної втрати зв'язку або інших проблем на лінії;
- немає обмежень на формат переданого контенту.

REST (скор. англ. Representational State Transfer, «передача репрезентативного стану») — підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP. В основі REST закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0.

Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабованість системи і дозволяє їй еволюціонувати з новими вимогами.

Extensible Messaging and Presence Protocol (XMPP) раніше Jabber — відкритий мережевий протокол для швидкого обміну повідомленнями та інформацією про присутність між користувачами мережі Інтернет. Основою XMPP є мова XML. Головною перевагою цієї технології є розподіленість, як в SMTP, та підтримка шлюзів в інші мережі обміну повідомленнями.

WebSocket — це протокол, що призначений для обміну інформацією між браузером і веб-сервером в режимі реального часу. Він забезпечує

двонаправлений повнодуплексний канал зв'язку через один TCP-сокет. WebSocket спроектовано для втілення у веб-браузерах та веб-серверах, але може також використовуватись будь-яким клієнт-серверним застосунком. Прикладний програмний інтерфейс WebSocket був стандартизований W3C, крім того протокол WebSocket стандартизований IETF як RFC 6455. У веб-застосунках доцільно використовувати протокол за необхідності відображення інформації в real-time.

2.3 Вибір компонентів IoT рішення

За основу буде взято Raspberry Pi, оскільки на кожному пристрої має бути запущений локальний сервер, який буде збирати інформацію з датчиків, обробляти цю інформацію для подальшої передачі в спрощеному однозначному вигляді. А також запуск локального сервера забезпечує можливість гнучкого налаштування часу обробку інформації з датчиків, оскільки в нас немає необхідності постійно передавати інформацію про стан заповненості контейнера і ми можемо просто очікувати запит від клієнта, але необхідно постійно перевіряти стан контейнера щодо диму чи вогню, а також чи піднята кришка контейнера або ж чи він взагалі не перевернутий. Для забезпечення енергоспоживання Raspberry Pi до нього буде інтегровано батарейку, яку буде легко замінювати.

Серед датчиків вимірювання відстані для оцінки заповненості контейнера буде використано ультразвуковий датчик відстані HC-SR04, оскільки він має ряд переваг по відношенню до інших варіантів. Цей датчик подає сигнал у певному діапазоні, що суттєво впливає на точність перевірки заповненості якоїсь області. До того ж він має низьку ціну.

Для перевірки розташування кришки або ж самого контейнера по відношенню до звичайного стану буде використано датчик GY-521, оскільки він має низьку ціну та легко підключається по інтерфейсу, що є в Raspberry Pi. Перевіряти температуру пристрій буде також за допомогою датчику GY-521. Це допоможе як в якості перевірки контейнера відносно його горіння, а також коригувати показники ультразвукового датчика відстані, оскільки, в залежності

від температури, показники цього датчика відстані можуть змінюватись в межах кількох сантиметрів.

Для перевірки задимленості на предмет займання буде використано датчик газу MQ-2. Його діапазону вимірюваних газів буде достатньо для завчасної перевірки контейнера ще до того, як він повністю загориться і дозволить запобігати, шляхом надсилання сервером відповідного повідомлення, викиду в атмосферу небезпечних і шкідливих газів.

Дані потрібно буде передавати по мережі, тому для цього обрано технологію NB-IoT, оскільки ця технологія швидко розвивається і її легко і дешево буде інтегрувати для кожного пристрою, навіть якщо це буде один пристрій на цілий квартал (тобто один контейнер), оскільки це не потребує встановлення базової станції і більшість українських провайдерів надають свої послуги з використанням NB-IoT.

Щодо налаштування серверу за допомогою мов програмування - буде використано мову програмування Java, а саме фреймворк Spring Boot, оскільки це допоможе легко, швидко і ефективно підняти локальний сервер Tomcat і мову програмування Python, а саме фреймворк Flask, для підняття ще одного локального сервера, що буде зчитувати дані з датчиків і передавати на Tomcat сервер.

Також потрібно буде підготувати проміжний сервер, для чого теж буде використаний фреймворк Spring Boot. Проміжний сервер допоможе оптимізувати інформацію, що приходить з великою кількості пристроїв, оскільки в одному місці може знаходитись декілька сміттєвих контейнерів, і надсилати в зкомпонованому вигляді. Для проектування кінцевих точок взаємодії локального і проміжного серверів буде використаний архітектурний стиль REST з передачею інформації у вигляді JSON. Для можливості адміністрування проміжного серверу буде написана клієнтська частина з використанням HTML та JavaScript.

Висновок до другого розділу

У другому розділі проведено аналіз засобів зв’язку, що використовуються для розробки пристроїв Інтернету речей. Більш детально проаналізовано саме LoRaWAN, NB-IoT та Sigfox. Основні фактори, що враховуються при виборі мережі це поширення пристроїв, необхідна швидкість передачі даних та покриття мережі.

Проаналізовано технології серверної частини для розробки бізнес-логіки роботи пристроїв. Взято до уваги популярність використання цих технологій, їх надійність, стабільність та розповсюдженість у виробництві як IoT рішень, так і проектів у світі в цілому.

Відповідно до отриманих результатів аналізу засобів зв’язку та технологій серверної частини розробки проекту, а також відповідно до особистих навичок і досвіду розробки програмного забезпечення, обрано компоненти, що будуть використані для імплементації власної моделі IoT-рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів.

РОЗДІЛ 3. АРХІТЕКТУРА ІОТ РІШЕННЯ ТА НАЛАШТУВАННЯ

3.1 Підготовка схеми взаємодії компонентів

Відповідно до схеми (рис 3.1) взаємодія компонентів відбувається наступним чином:

1. З мережі надходить запит на пристрій що до стану заповненості контейнера.
2. Локальний Tomcat сервер, що піднятий на Raspberry Pi опрацьовує запит мережі і робить запит на локальний сервер Flask.
3. В свою чергу локальний сервер Flask, що відповідає безпосередньо за зчитування даних з датчиків - ініціює сигнал в HC-SR04.
4. Ультразвуковий датчик відстані HC-SR04 зчитує сигнали, що повертаються від об’єктів, опрацьовує ці сигнали і передає інформацію до Flask сервера.
5. Flask сервер повертає інформацію до Tomcat сервера.
6. Tomcat сервер формує відповідь на запит з мережі.

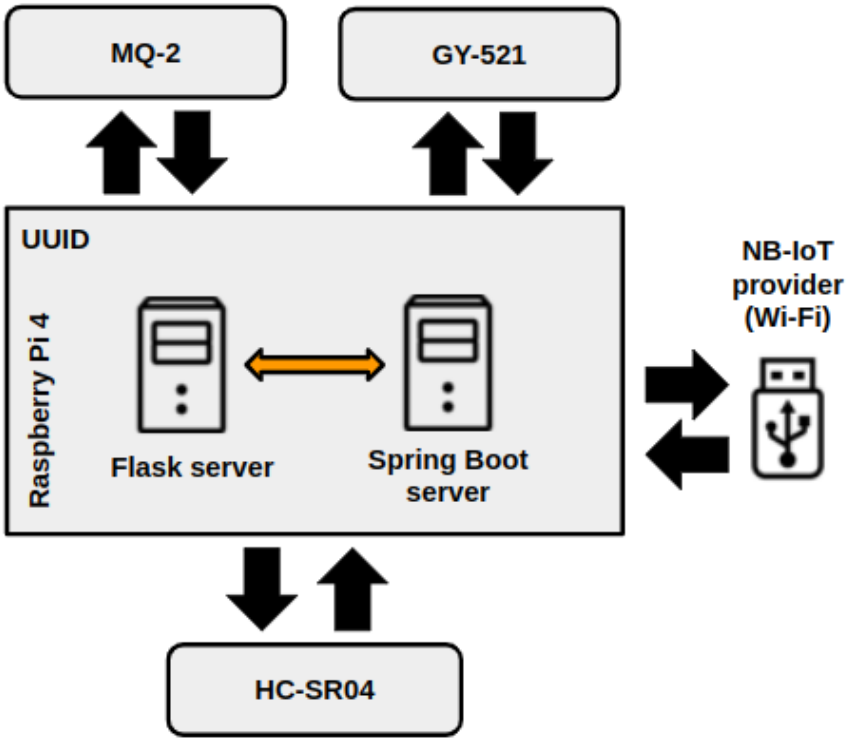


Рисунок 3.1. Структурна схема пристрою

Також, для забезпечення надійності, контейнер перевіряється ще двома датчиками - датчиком газу та гіроскопом-температури. Логіка їх взаємодії наступна:

1. Кожні 10 секунд Tomcat сервер виконує запит до Flask сервера по черзі щодо кожного з зазначених вище датчиків.
2. Локальний сервер Flask ініціює зчитування даних з датчиків.
3. Датчики опрацьовують інформацію і повертають її до Flask сервера.
4. Flask сервер передає отриману інформацію на Tomcat сервер.
5. Tomcat сервер аналізує отримані з датчиків дані та в разі необхідності (наприклад, коли інформація з гіроскопа вказує на те, що контейнер був перевернутий) надсилає інформацію про надзвичайний стан в мережу до проміжного сервера.

Проміжний сервер, в свою чергу, відображає повідомлення про стан контейнера на сайті, а також надсилає електронний лист на пошту або ж повідомлення в месенджері відповідальній особі.

Наведено структурну схему проміжного сервера (рис 3.2). Відповідно до схеми взаємодія користувача з пристроєм відбувається наступним чином:

1. Користувач реєструється через клієнт в інтернет-ресурсі, що представляє собою проміжний сервер.
2. Користувач робить через веб-клієнт пошук контейнера по UUID або за адресою.
3. Веб-клієнт надсилає HTTP GET запит на сервер.
4. Сервер здійснює пошук запису про контейнер в своїй базі даних за вказаними даними.
5. Сервер повертає на веб-клієнт знайдену кількість контейнерів (може бути і пустий список)
6. Користувач обирає контейнер (чи контейнери), який хоче відслідковувати на своїй головній сторінці.

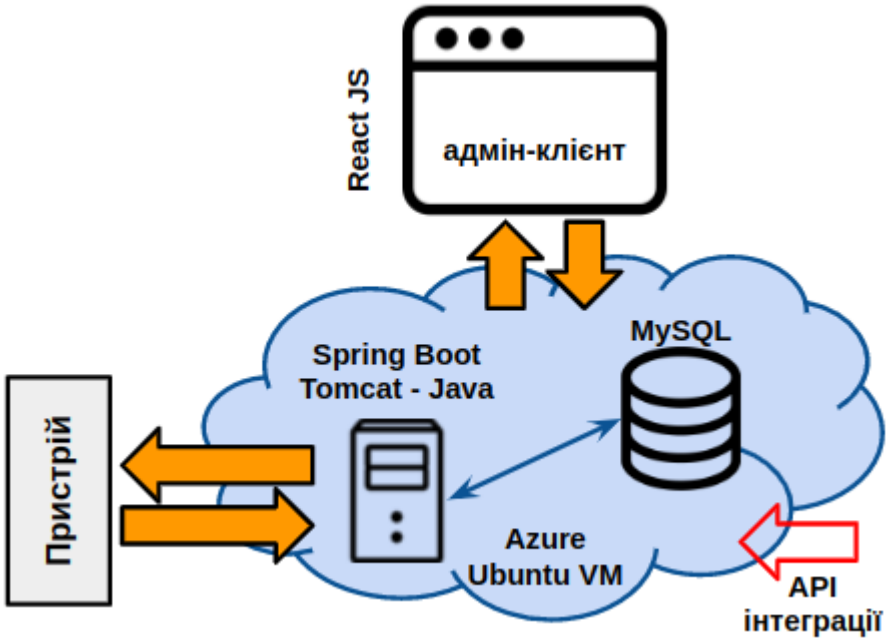


Рисунок 3.2. Структурна схема проміжного сервера

Перегляд стану контейнера для користувача відбувається наступним чином:

1. Користувач відкриває головну сторінку, де відображається список контейнерів, що відслідковуються користувачем.
2. Веб-клієнт надсилає HTTP GET запит на сервер.
3. Сервер знаходить в базі даних список контейнерів, що відслідковує користувач.
4. Сервер надсилає HTTP GET запит на пристрій для актуалізації інформації.
5. Пристрій повертає відповідь на запит.
6. Сервер зберігає оновлені дані про стан контейнера і повертає ці дані на веб-клієнт.
7. Користувач дивиться на актуальний стан контейнера на своїй сторінці.

Також користувач може підписатись на екстрені повідомлення про стан контейнера, щоб в разі підпалу чи перевертання контейнера - надійшло повідомлення. Так само користувач може через веб-клієнт завантажити csv-файл зі станом приєднаних для відслідковування контейнерів. Приклад вигляду csv-файлу можна побачити нижче (рис 3.3).

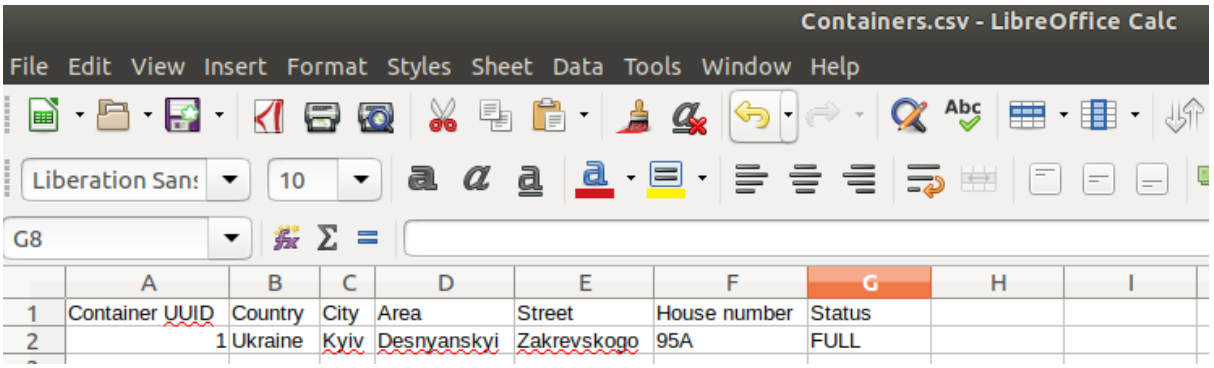


Рисунок 3.3. Csv-файл з інформацією про контейнер

Також передбачена інтеграція через API сервера. Перегляд стану контейнера через інтеграцію відбувається наступним чином:

1. Через API сервера виконується HTTP GET запит на сервер по UUID або за адресою.
2. Сервер знаходить в базі даних список контейнерів за зазначеними даними.
3. Сервер надсилає HTTP GET запит на пристрій для актуалізації інформації.
4. Пристрій повертає відповідь на запит.
5. Сервер зберігає оновлені дані про стан контейнера і формує відповідь на запит.

Далі наведено приклад відповіді на запит через API (рис 3.4), а також схему взаємодії між усіма компонентами проекту (рис 3.5).

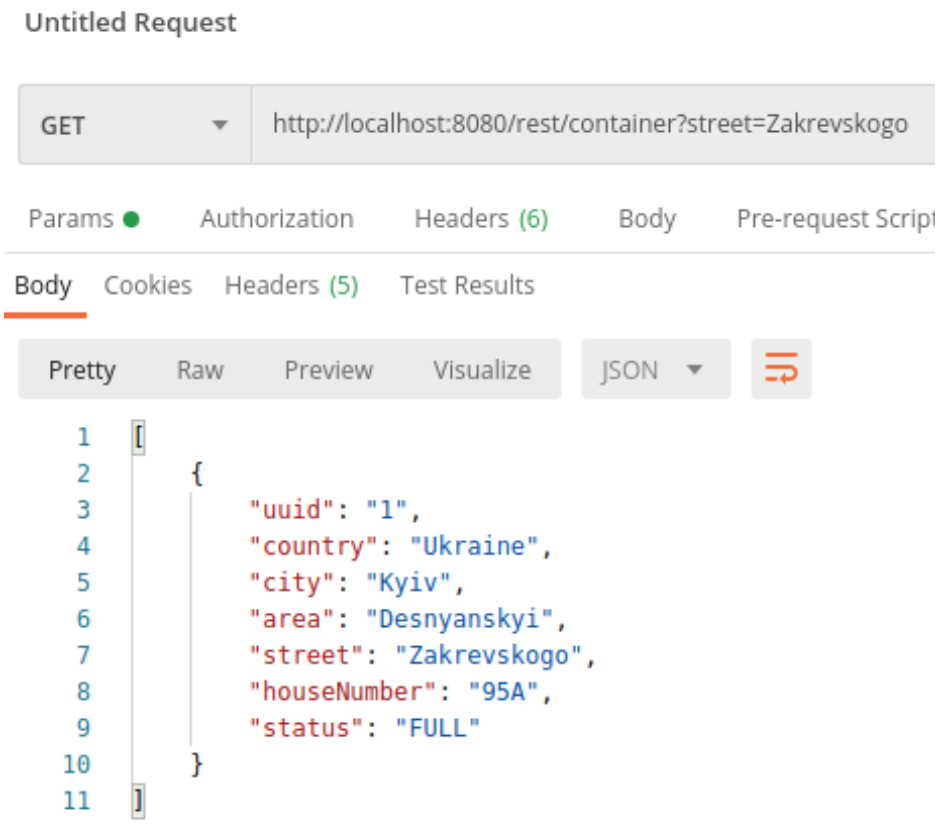


Рисунок 3.4. Відповідь на запит через API проміжного сервера

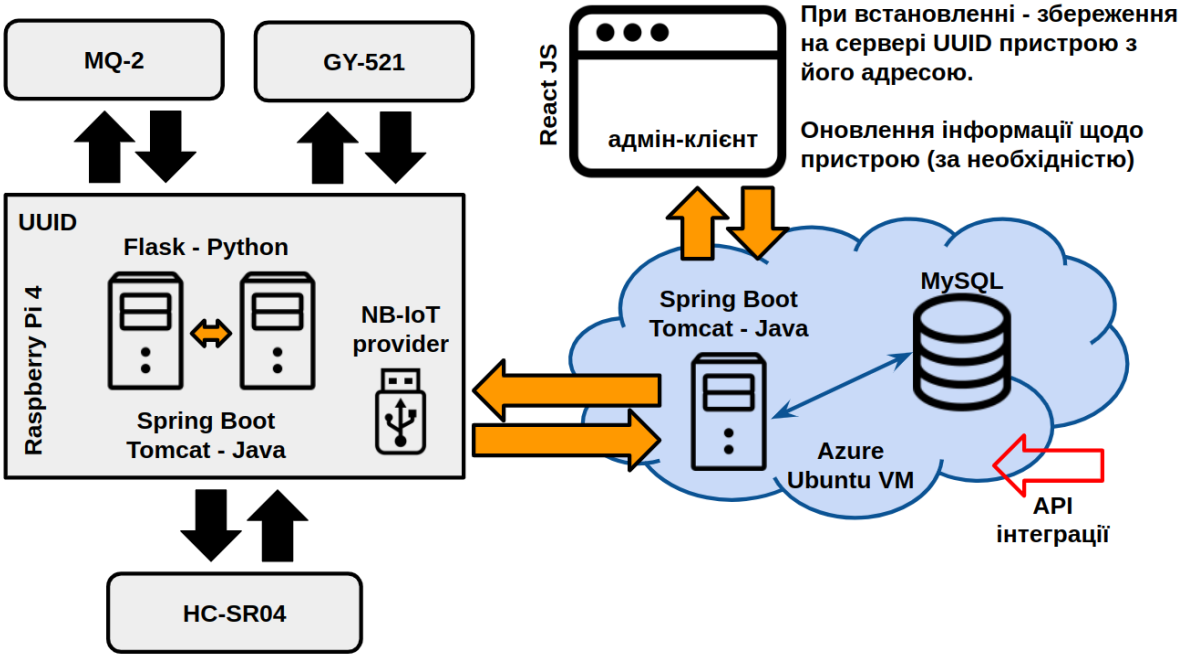


Рисунок 3.5. Взаємодія всіх компонентів

3.2 Налаштування апаратного забезпечення

Основою IoT пристрою слугує Raspberry Pi 4 Model B, який в моєму випадку, має 8 ГБ оперативної пам’яті. Для забезпечення більш ефективного охолодження системи використовується набір мідних радіаторів (рис 3.6). В якості карти пам’яті використовується MicroSD 32GB 10 class UHS.

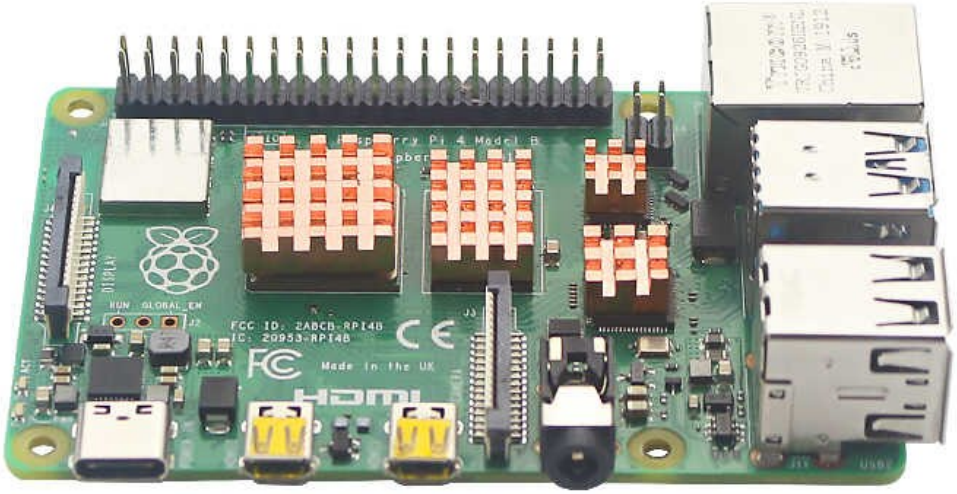


Рисунок 3.6. Raspberry Pi 4 Model B із мідними радіаторами

Ультразвуковий датчик відстані HC-SR04 має чотири контакти (рис 3.7). Контакти VCC та GND слугують для підключення живлення, а Trig і Echo - для відправки і прийому сигналів далекоміра. Наведено перелік контактів самого мікрокомп'ютера Raspberry Pi 4 Model B (рис 3.8). Напруга живлення далекоміра 5В, то ж його контакт VCC підключено до контакту 2 на Raspberry Pi. А контакт GND, відповідно, підключено до контакту 6 на Raspberry Pi. Далі підключено контакт Trig до контакту 16 на Raspberry Pi для ініціювання сигналу, а от з контактом Echo трохи складніше. Справа в тому, що контакти Raspberry Pi приймають сигнал напругою 3.3В, а от контакт Echo далекоміра віддає

Рисунок 3.7. Датчик HC-SR04

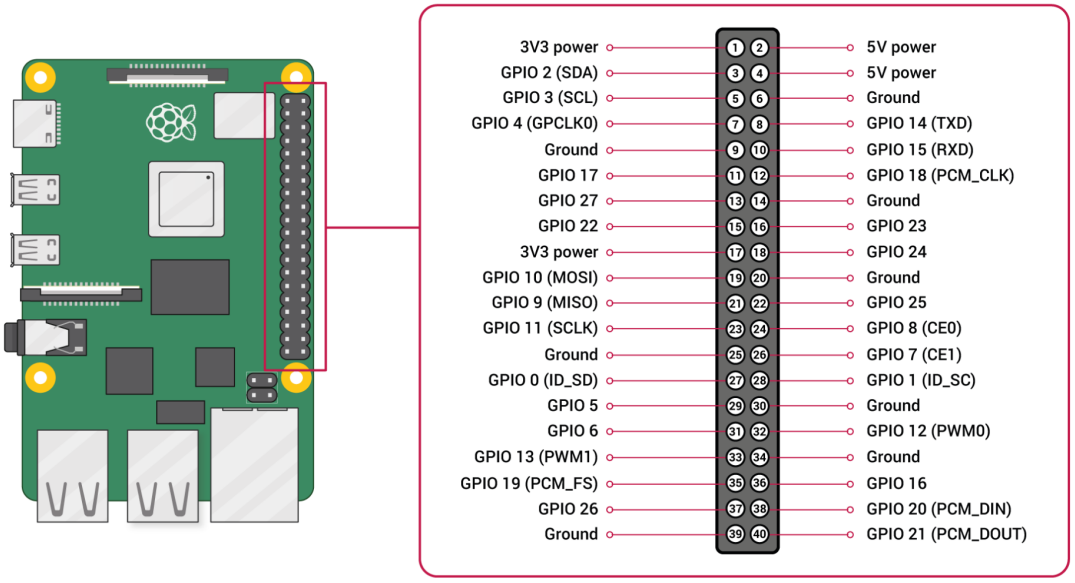


Рисунок 3.8. Контакти Raspberry Pi 4 Model B

сигнал з напругою 5В. То ж за допомогою резисторів 2кОм та 1кОм напругу було зменшено.

Для отримання необхідних даних від датчика GY-521 було під’єднано 4 його контакти: VCC, GND, SCL, SDA (рис 3.9). Контакт VCC приєднано до контакта 1 на платі, для отримання живлення 3.3В, а контакт GND, відповідно, до контакту 9 на платі Raspberry Pi. Далі контакт SCL датчика під’єднано до контакту 5 на платі, а контакт SDA датчика до контакту 3 на платі.

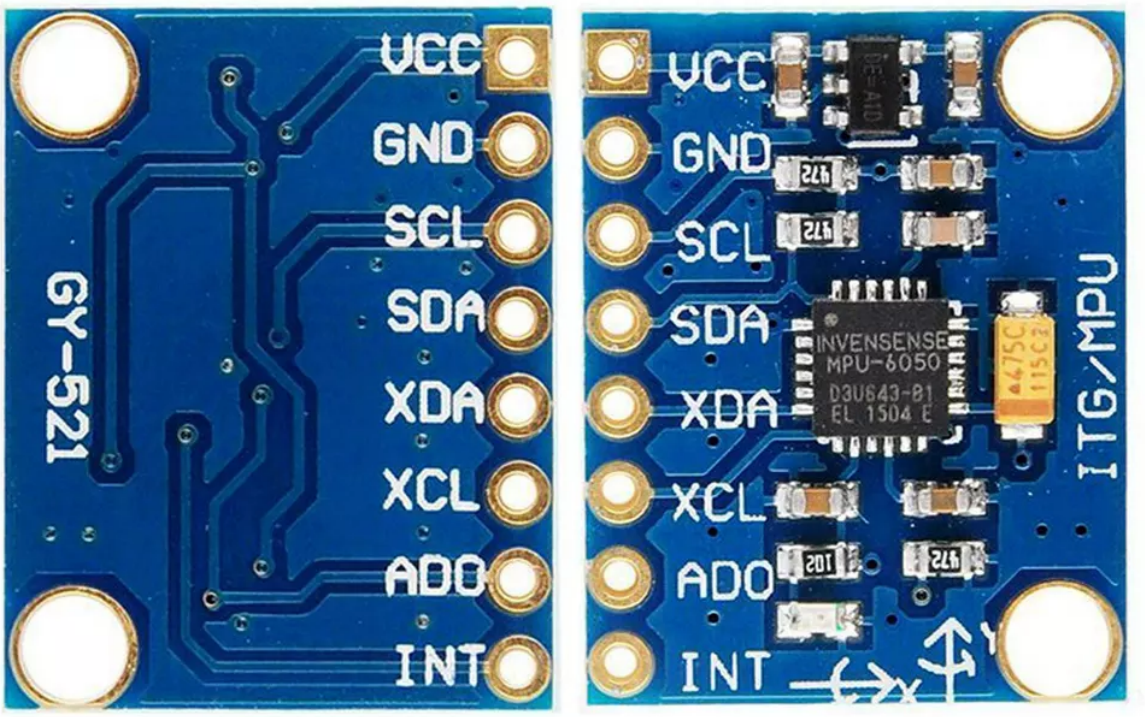


Рисунок 3.9. Датчик GY-521

Останнім було підключено датчик газу MQ-2. Оскільки Raspberry Pi може обробляти лише цифровий сигнал, нам потрібно додати аналого-цифровий перетворювач (АЦП) для обробки аналогового сигналу від датчика MQ-2. Він може виявляти дим у повітрі відповідно до значення напруги. MCP3008 (рис 3.10) як чіп АЦП дуже поширений і рекомендований. Датчика MQ-2 через MCP3008 підключено до плати Raspberry Pi (рис 3.11).

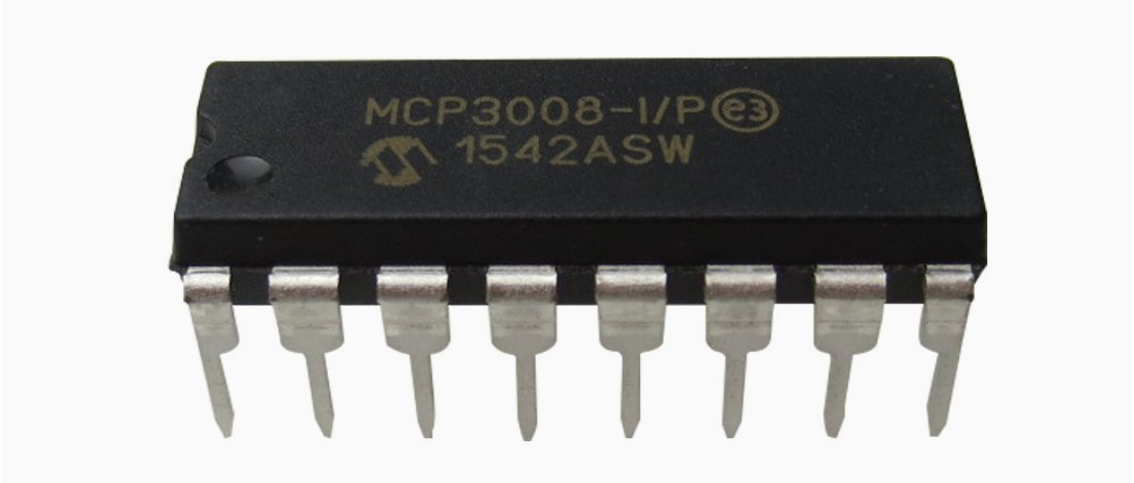


Рисунок 3.10. MCP3008

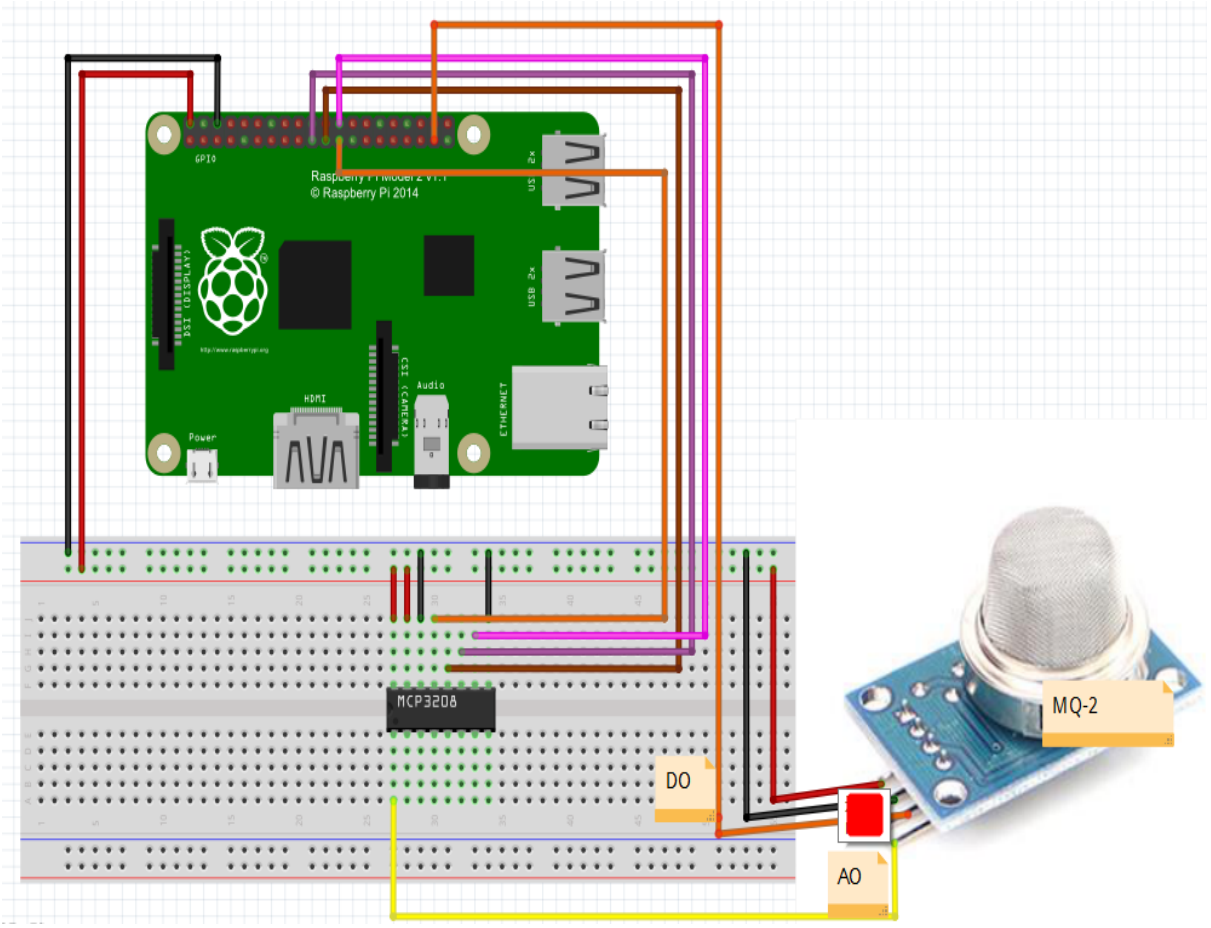


Рисунок 3.11. Підключення MQ-2

Фінальний вид підключення усіх компонентів з використанням резисторів на платі прототипування зображено нижче (рис 3.12).

Також Raspberry Pi 4 Model B буде поміщено в корпус із інтегрованим кулером для активного охолодження всього мікрокомп’ютера. Компоненти підключені і готові до подальшої розробки програмного забезпечення.

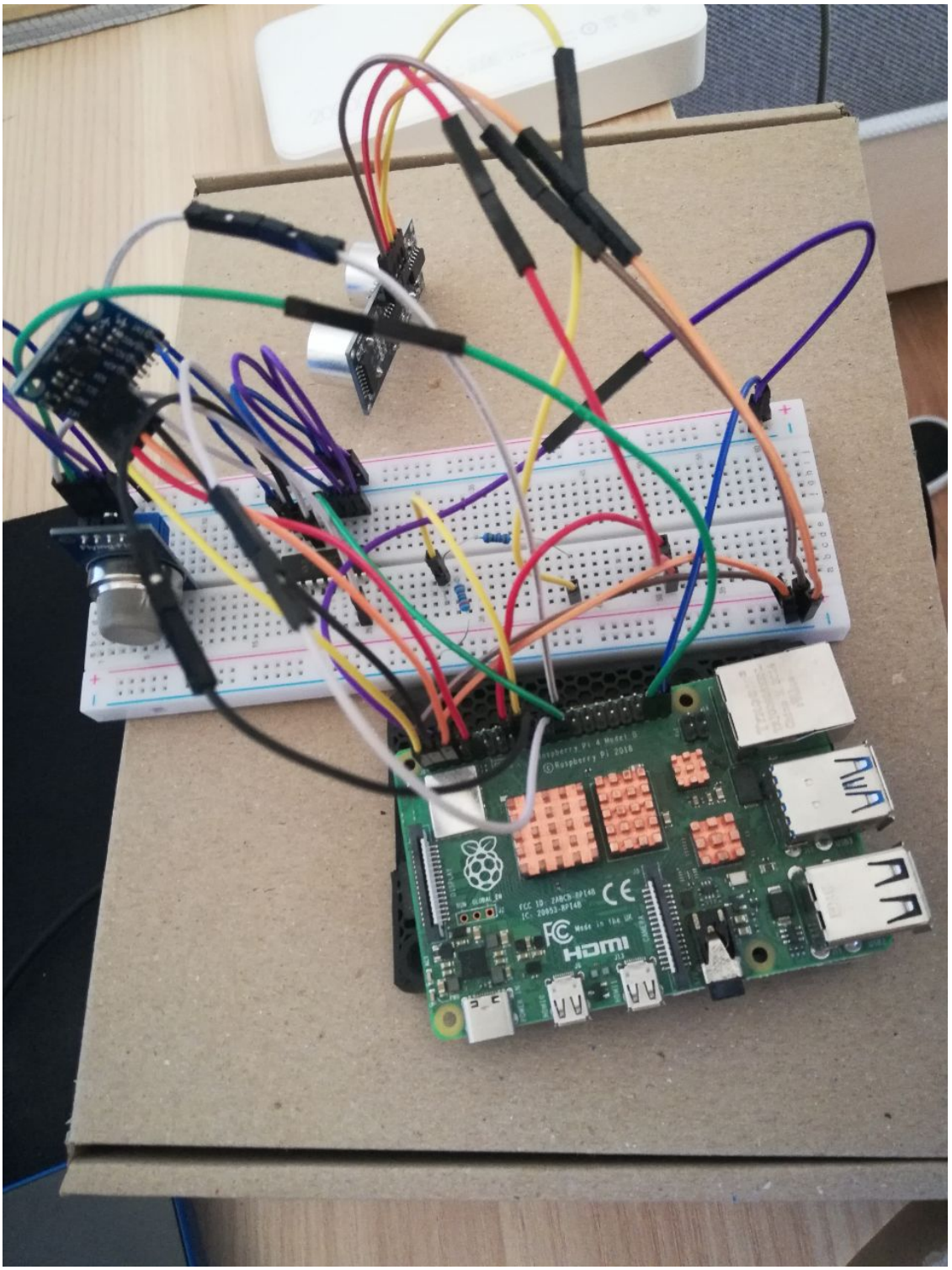


Рисунок 3.12. Підключення компонентів

Висновок до третього розділу

У третьому розділі розроблено схеми взаємодії компонентів. Розроблено структурну схему пристрою та описано взаємодію компонентів пристрою, а саме процес зчитування даних з ультразвукового датчика відстані для перевірки

заповненості контейнера, а також постійна перевірка стану контейнера за рахунок зчитування даних з датчика газу MQ-2 та акселерометра-гіроскопа GY-521.

Також розроблено структурну схему проміжного сервера та описано взаємодію користувача з користувацьким інтерфейсом для отримання даних щодо заповненості контейнера, а також отримання цих даних через інтеграцію з API проміжного сервера. Наведено варіанти отримання необхідної інформації для користувачів або інтеграторів.

Також налаштовано Raspberry Pi 4 Model B, що являється основним та найдорожчим компонентом розробки, і під’єднано до нього всі необхідні для IoT рішення датчики. Для отримання даних від ультразвукового датчику приєднано його контакти VCC, GND,Trig і Echo. Для отримання необхідних даних від датчика GY-521 було під’єднано 4 його контакти: VCC, GND, SCL, SDA. Для можливості обробки даних MQ-2 додатково приєднано аналого-цифровий перетворювач MCP3008.

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ ІОТ РІШЕННЯ ДЛЯ МОНІТОРИНГУ ТА АНАЛІЗУ СИСТЕМИ СМІТТЄЗБЕРІГАННЯ ТВЕРДИХ ВІДХОДІВ

Для розробки програмної частини проекту на Raspberry Pi було встановлено Java 8 OpenJDK, а Python вже був встановлений. Додатково було ще завантажено Python пакети для Flask сервера. В якості операційної системи самого мікрокомп'ютера використовуються Raspberry Pi OS.

4.1 Розробка компонентів та серверної частини Raspberry Pi

В першу чергу має бути налаштовано зчитування даних з датчиків. Для цього в проекті використовується Python. Створено файл distance.py, в якому імпортуються необхідні бібліотеки для взаємодії з контактами мікрокомп’ютера та для підняття локального Flask сервера і зчитування даних з HC-SR04 контролера. Код з логікою взаємодії наведено в додатку А.

Як можна побачити з лістингу - локальний сервер має в якості кінцевої точки для зчитування даних з ультразвукового датчика шлях “http://localhost:5000/ultrasonic”. Для Flask, якщо не вказувати напряму - доменне ім’я серверу матиме назву “http://localhost:5000”. Наведений код опрацьовує логіку роботи ультразвукового датчика відстані HC-SR04.

Розроблений код для отримання необхідних даних від датчика газу MQ-2 наведено в додатку Б.

Наступним наведено код для чіпу GY-521, який має повертати дані про положення та температуру контейнера:

```
from mpu6050 import mpu6050
import time
mpu = mpu6050(0x68)


@api.route('/gyroTemp, methods=['GET'])
def get_gyro_and_temp_data():
   gyro_data = mpu.get_gyro_data()
   return jsonify(
               {
```

```
                    temp: str(mpu.get_temp()),
                    x: str(gyro_data['x']),
                    y: str(gyro_data['y']),
                    z: str(gyro_data['z'])
                }
            )
```

Після того, як Flask сервер розроблений і готовий до використання - необхідно розробити Tomcat сервер, що виступає в якості головного сервера взаємодії між всім пристроєм та проміжним сервером, а також, в майбутньому, міститиме додаткову бізнес-логіку щодо інших датчиків.

Для швидкого старту всіх необхідних компонентів використано Spring Boot. Для простого зібрання залежностей і компонентів Spring Boot проекту використовується Maven. Повну структуру Maven pom.xml локального сервера можна побачити в додатку В. Для запуску локального Tomcat сервера із застосуванням Spring Boot вказано наступний код:

```
@SpringBootApplication
public class ContainercontroliotsensorApplication {
    public static void main(String[] args) {
        SpringApplication.run(ContainercontroliotsensorApplication.class, args);
    }
}
```

Після цього залишилось лише створити контролер та кінцеву точку, за яку цей контролер буде відповідати. Проміжний сервер робить запит за цією кінцевою точкою, а з неї вже запит направляється локально на Flask сервер. У кожного пристрою є свій унікальний UUID для ідентифікації пристрою і він також використовується як частина шляху кінцевої точки. Нижче наведено код контролера з кінцевою точкою для ультразвукового датчика:

```
@RestController
public class UltrasonicSensorController {
```

```
        @GetMapping("/" + DEVICE_ID + "/ultrasonicsensordata")
        public UltrasonicData getData() {
            String url = "http://127.0.0.1:5000" + ULTRASONIC_PATH;
            ResponseEntity<UltrasonicData> result = new
RestTemplate().getForEntity(url, UltrasonicData.class);
            System.out.println(result.getBody());
            return result.getBody();
        }
    }
```

Відповідно змінна DEVICE_ID і зберігає в собі значення UUID пристрою, а ULTRASONIC_PATH містить в собі значення-шлях до кінцевої точки в Flask сервері - “/ultrasonic”. Аналогічним чином працює взаємодія і з іншими датчика. Єдина відмінність - методи для датчиків газу та гіроскопа стартують не від виклику з проміжного сервера, а самим цим сервером і дані передаються на проміжний сервер лише у випадку відходження від норми показників, що зберігаються у внутрішній пам’яті пристрою.

4.2 Розробка серверної частини проміжного сервера

Серверна частина Raspberry Pi готова до роботи, а отже необхідно розробити проміжний сервер - хмару, яка відповідає за взаємодію з клієнтами проекту, а також за бізнес-логіку роботи компонентів проекту.

В якості сервера в хмарі також використовується Spring Boot, Tomcat, Maven і, відповідно, Java. Повну структуру Maven pom.xml проміжного сервера можна побачити в додатку Г. Для збереження інформації щодо стану контейнерів та іншого - використовується база даних MySQL.

Для запуску проміжного Tomcat сервера із застосуванням Spring Boot використовується аналогічний код, що й в локального Tomcat сервера. Відрізняється лише назва головного класу.

Для підключення до бази даних використані наступні налаштування:

```
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/iotcontroller
spring.datasource.username=valik
spring.datasource.password=16021999
spring.jpa.hibernate.ddl-auto=update
```

```
spring.jpa.show-sql=true
```

Залишилось додати безпосередньо кінцеві точки, з якими будуть взаємодіяти користувачі проекту. Для інтеграції з API проміжного серверу компаніям необхідно використовувати GET “https://domain/rest/container”, передавши в якості параметра “street” вулицю, за якою необхідна інформація про контейнери. Код описаної кінцевої точки має наступний вигляд:

```
@GetMapping("/rest/container")
    @ResponseBody
    public List<ContainerDTO> getRestData(@RequestParam String street) {
        List<Address> addresses = addressRepository.findAllByStreet(street);
        List<Container> containers =
containerRepository.findAllByAddressIn(addresses);

        for (Container container : containers) {
            String url = "http://192.168.1.103:8080/" + container.getUuid() +
"/ultrasonicsensordata";
            ResponseEntity<UltrasonicData> result = new
RestTemplate().getForEntity(url, UltrasonicData.class);
            if (result.getStatusCode().is2xxSuccessful() && result.getBody() != null) {
                if (container.getDefaultDistance() == 0) {

container.setDefaultDistance(Float.parseFloat(result.getBody().getDistance()));
                } else {

container.setLastDistance(Float.parseFloat(result.getBody().getDistance()));
                }
            }
        }
        List<ContainerDTO> containersDTOList = new ArrayList<>();
        List<Container> savedContainers = containerRepository.saveAll(containers);
        savedContainers.forEach(c -> containersDTOList.add(ContainerDTO.of(c)));
        return containersDTOList;
    }
```

Повний код контролера для отримання стану заповненості контейнера наведено в додатку Д.

Для отримання інформації також можна використовувати сайт проекту (рис 4.1 - 4.2).

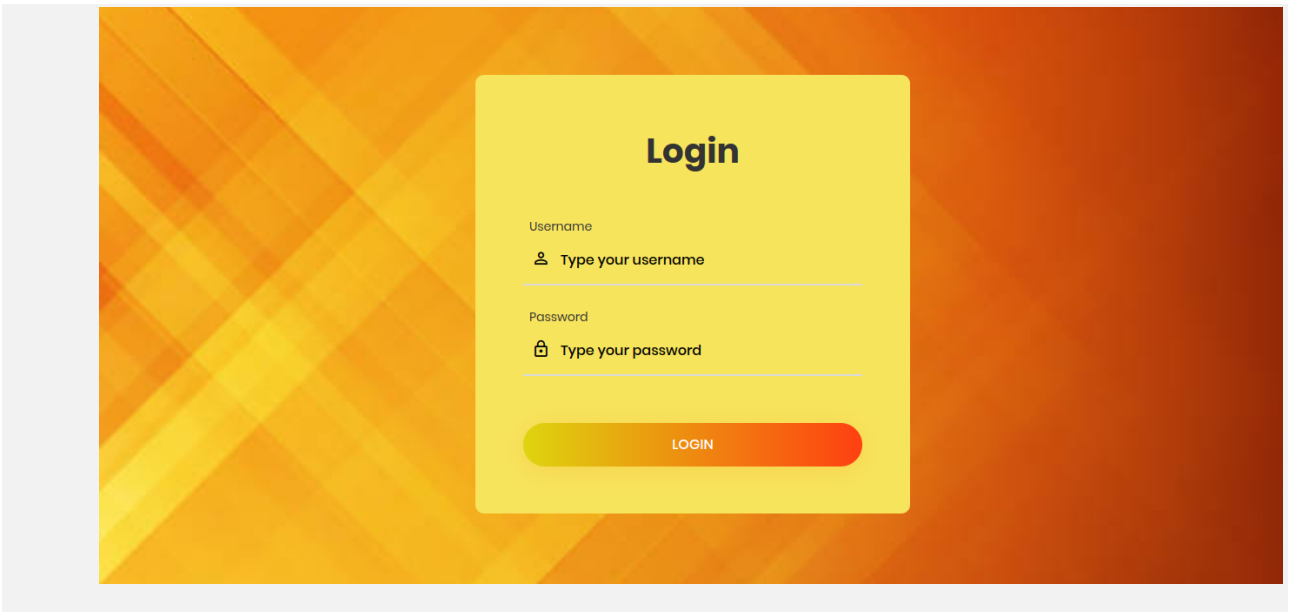


Рисунок 4.1. Сторінка логіну веб-сайту проекту

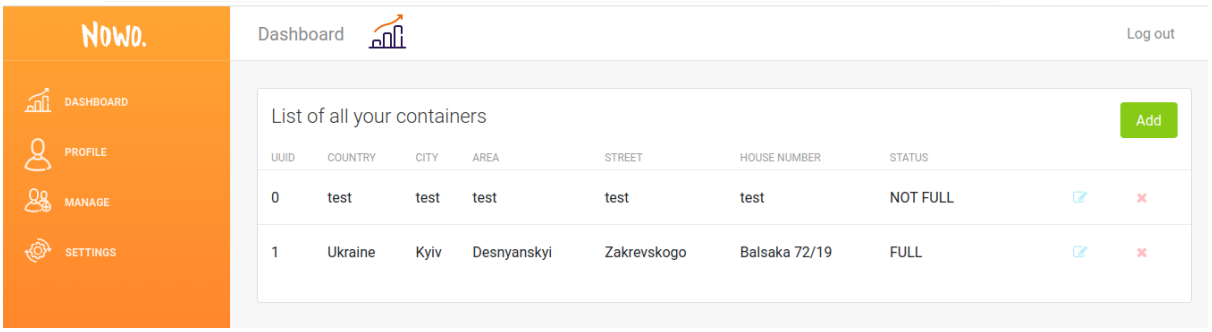


Рисунок 4.2. Головна сторінка веб-сайту проекту з погляду користувача

4.3 Розробка клієнтської частини

Дружній інтерфейс для зв'язку з користувачем було обрано у вигляді веб-сайту проекту було спрощено до однієї сторінки з базовим функціоналом:

1. Перелік збережених контейнерів у вигляді таблиці.
2. Форма додавання нового контейнера.
3. Поле та кнопка оновлення інформації щодо збережених контейнерів за вказаною вулицею.

4. Поле та кнопка завантаження csv файлу щодо збережених контейнерів за вказаною вулицею.

Для додавання нового контейнера використовується наступна форма, що наведена в додатку Е.

Форма для оновлення інформації щодо збережених контейнерів за вказаною вулицею має наступний вигляд:

```
<form method="post" action="/container" class="form-inline m-3
justify-content-center">
<input class="form-control" type="text" name="street" placeholder="Update
by street">
<button class="btn btn-primary ml-2" type="submit">Update</button>
</form>
```

Форма для завантаження csv файлу щодо збережених контейнерів за вказаною вулицею має наступний вигляд:

```
<form method="get" action="/csv/container" class="form-inline m-3
justify-content-center">
<input class="form-control" type="text" name="street"
placeholder="Download by street">
<button class="btn btn-primary ml-2" type="submit">Download</button>
</form>
```

Висновок до четвертого розділу

В четвертому розділі розроблено бізнес-логіку роботи ультразвукового датчика відстані HC-SR04, датчика газу MQ-2 та гіроскопу-акселерометру GY-521. Код цих датчиків працює на піднятому на пристрої веб-сервері Flask. Також імплементовано логіку другого внутрішнього веб-серверу з використанням Spring Boot, що слугує тунелем між проміжним сервером в хмарі та датчиками, що прослуховується Flask сервером. В ньому реалізовано функціонал збереження початкових даних зазначених датчиків в пам’яті пристрою, а також логіка повторюваної автоматичної перевірки даних датчиків на предмет підпалу чи некоректне положення у просторі.

Також, розроблено бізнес-логіку роботи проміжного серверу для взаємодії з датчиками, а також написано код обробки запитів з веб-інтерфейсу користувача та підготовлено API для можливості інтеграції стороннім ресурсам.

Імплементовано клієнтську частину функціоналу проекту. Створено сторінки для реєстрації користувача, логіну, а також загальна сторінка, де користувач може переглядати стан збереженого для відслідковування контейнера. Додано форми для додавання нового контейнера, оновлення інформації по існуючим контейнерам, а також кнопку для завантаження інформації по контейнерам у форматі csv.

ВИСНОВКИ

У даній кваліфікаційній роботі магістра запропоновано метод моніторингу та аналізу системи сміттєзберігання твердих відходів. На основі даного метода спроектовано та програмно реалізовано високоефективну систему IoT рішень для моніторингу та аналізу стану контейнера з мінімальним використанням енергоресурсів мікрокомп'ютера.

Виконано:

1. Аналіз датчиків та пристроїв, що можна використати для проектування пристрою.
2. Аналіз існуючих систем IoT рішень, завдяки яким можна отримати оцінку рівня заповненості ємності для твердих відходів.
3. Дослідження комунікаційних систем та технологій.
4. Проектування логічних зв'язків вузлів системи.
5. Розроблено структурну схему системи IoT пристрою.
6. Створено апаратне та програмне забезпечення системи IoT пристрою.
7. За матеріалами роботи опубліковано статтю та тези.

Отримані результати свідчать про виконання кваліфікаційної роботи магістра у повному обсязі. Досягнуто всіх поставлених у меті завдань. Під час створення даної системи отримано наступні наукові та практичні результати:

1. Виконано аналіз датчиків та пристроїв, що використовуються або можуть бути використані для розробки проектів Інтернету речей і зокрема для розробки проекту для моніторингу та аналізу системи сміттєзберігання твердих відходів.
2. Проаналізовано динаміку популярності використання або зацікавленості в мікрокомп'ютерах та мікроконтролерах. А також розподілення по регіонам. Найпопулярнішими виявились, відповідно до Google Trends, мікроконтролер Arduino та мікрокомп'ютер Raspberry Pi. Проведено аналіз специфікацій на базі Arduino, Raspberry Pi та Microbit. В таблицях наведено відмінності характеристик в різних моделях в межах специфікацій кожної бази окремо.

3. Проведено аналіз існуючих датчиків для вимірювання відстані на основі ультразвуку, інфрачервоного світла та лазера, а також аналіз датчиків полум'я, газу, температури, вологості, гіроскопів та акселерометрів. Розроблено метод для моніторингу та аналізу системи сміттєзберігання твердих відходів.
4. Проведено аналіз засобів зв'язку, що використовуються для розробки пристроїв Інтернету речей. Більш детально проаналізовано саме LoRaWAN, NB-IoT та Sigfox. Основні фактори, що враховуються при виборі мережі це поширення пристроїв, необхідна швидкість передачі даних та покриття мережі. Проаналізовано технології серверної частини для розробки бізнес-логіки роботи пристроїв. Взято до уваги популярність використання цих технологій, їх надійність, стабільність та розповсюдженість у виробництві як IoT рішень, так і проектів у світі в цілому.
5. Обрано компоненти, що будуть використані для імплементації власної моделі IoT-рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів.
6. Розроблено схеми взаємодії компонентів. Розроблено структурну схему пристрою та описано взаємодію компонентів пристрою, а саме процес зчитування даних з ультразвукового датчика відстані для перевірки заповненості контейнера, а також постійна перевірка стану контейнера за рахунок зчитування даних з датчика газу MQ-2 та акселерометра-гіроскопа GY-521. Розроблено структурну схему проміжного сервера та описано взаємодію користувача з користувацьким інтерфейсом для отримання даних щодо заповненості контейнера, а також отримання цих даних через інтеграцію з API проміжного сервера. Наведено варіанти отримання необхідної інформації для користувачів або інтеграторів.
7. Налаштовано Raspberry Pi 4 Model B, що являється основним та найдорожчим компонентом розробки, і під'єднано до нього всі необхідні

для IoT рішення датчики. Для отримання даних від ультразвукового датчику приєднано його контакти VCC, GND,Trig і Echo. Для отримання необхідних даних від датчика GY-521 було під’єднано 4 його контакти: VCC, GND, SCL, SDA. Для можливості обробки даних MQ-2 додатково приєднано аналого-цифровий перетворювач MCP3008.

8. Розроблено бізнес-логіку роботи ультразвукового датчика відстані HC-SR04, датчика газу MQ-2 та гіроскопу-акселерометру GY-521. Код цих датчиків працює на піднятому на пристрої веб-сервері Flask. Також імплементовано логіку другого внутрішнього веб-серверу з використанням Spring Boot, що слугує тунелем між проміжним сервером в хмарі та датчиками, що прослуховується Flask сервером. В ньому реалізовано функціонал збереження початкових даних зазначених датчиків в пам’яті пристрою, а також логіка повторюваної автоматичної перевірки даних датчиків на предмет підпалу чи некоректне положення у просторі. Також розроблено бізнес-логіку роботи проміжного серверу для взаємодії з датчиками, а також написано код обробки запитів з веб-інтерфейсу користувача та підготовлено API для можливості інтеграції стороннім ресурсам.
9. Імплементовано клієнтську частину функціоналу проекту. Створено сторінки для реєстрації користувача, логіну, а також загальна сторінка, де користувач може переглядати стан збереженого для відслідковування контейнера. Додано форми для додавання нового контейнера, оновлення інформації по існуючим контейнерам, а також кнопку для завантаження інформації по контейнерам у форматі csv.

Результати здійснених у проекті досліджень можуть бути використані при проектуванні IoT пристроїв з подібним функціоналом, а спроектоване та розроблене рішення може бути завершене і доповнене для практичного використання і в комерційних цілях. Завдання до кваліфікаційної роботи магістра виконано у повному обсязі.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Твардовський В. Г., Кравченко О. В. "IoT-рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів" / IoT Solution for Monitoring and Analysis of Solid Waste Save System, IV Міжнародна науково-практична конференція, 1–2 лютого 2022 р., Pp. 180.
2. Valentyn Tvardovskyi, Olha Kravchenko, Svetlana Besedina. Design of IoT-solution for monitoring and analysis of the solid waste storage system. Technology Audit and Production Reserves, Vol. 2 No. 2(64) (2022) Information and control systems. Information technologies. Pp. 6-10 .
3. Arduino vs Raspberry Pi vs Micro Bit: Platforms for Fast IoT Systems Prototyping: [навчальний посібник] / Nur Qamarina, Azi Azwady Jamaludin, Azizul Azizan та Hafiza Abas — Malaysia-Japan International Institute Of Technolog, 2018. — 12 с.
4. Архитектура интернета вещей: [навчальний посібник] / пер. с англ. М. А. Райтмана. – М.: ДМК Пресс, 2019. – 454 с. — ISBN 978-5-97060-672-8
5. Ecube Labs, CleanFLEX, the fill-level sensor [Електронний ресурс] — Режим доступу: https://www.ecubelabs.com/ultrasonic-fill-level-sensor/ (дата звертання: 28.01.2022)
6. Binology, Датчик уровня заполнения Smart City Waste Sens [Електронний ресурс] — Режим доступу: https://www.binology.ru/sensor (дата звертання: 28.01.2022)
7. BrighterBins, Our solution [Електронний ресурс] — Режим доступу: https://www.brighterbins.com/the-solution (дата звертання: 29.01.2022)
8. Raspberrypi.org, GPIO [Електронний ресурс] — Режим доступу: https://www.raspberrypi.org/documentation/usage/gpio/ (дата звертання: 02.02.2022)
9. Arduino Documentation [Електронний ресурс] — Режим доступу: https://docs.arduino.cc/ (дата звертання: 04.02.2022)

10. BBC micro:bit MicroPython documentation [Електронний ресурс] — Режим доступу: https://microbit-micropython.readthedocs.io/en/v2-docs/ (дата звертання: 08.02.2022)
11. Alldatasheet.com, HC-SR04 Datasheet (PDF) - List of Unclassifed Manufacturers [Електронний ресурс] — Режим доступу: https://www.alldatasheet.com/datasheet-pdf/pdf/1132203/ETC2/HC-SR04.html?gclid=CjwKCAjw4ayUBhA4EiwATWyBrrkS08krKzdfxCEDELtYXayIvIotFHR4eSfStDJGrUlAPO7PPBVgsBoCuy4QAvD_BwE (дата звертання: 10.02.2022)
12. Arduino.ua, Ультразвуковий датчик відстані HC-SR04 [Електронний ресурс] — Режим доступу: https://arduino.ua/prod182-yltrazvykovoi-datchik-rasstoyaniya-hc-sr04 (дата звертання: 10.02.2022)
13. Waveshare, Laser Sensor User Manual SR04 [Електронний ресурс] — Режим доступу: https://www.waveshare.com/w/upload/9/93/Laser-Sensor-UserManual.pdf (дата звертання: 11.02.2022)
14. Sharp, IR датчик расстояния Sharp GP2Y0A21YK0F [Електронний ресурс] — Режим доступу: https://www.robostore.com.ua/moduli-i-datchiki/sensory-datchiki-i-moduli/ir-datchik-rasstoyaniya-sharp-gp2y0a21yk0f-10-80-sm/ (дата звертання: 11.02.2022)
15. RobotChip, Обзор модуля GY-521 (MPU-6050) [Електронний ресурс] — Режим доступу: https://robotchip.ru/obzor-modulya-gy-521/ (дата звертання: 13.02.2022)
16. Arduino.ua, Датчик вологості і температури AM2320 [Електронний ресурс] — Режим доступу: https://arduino.ua/prod3563-datchik-vlajnosti-i-temperatyri-am2320 (дата звертання: 15.02.2022)
17. Arduino.ua, Датчик диму MQ-2 [Електронний ресурс] — Режим доступу: https://arduino.ua/prod188-datchik-dima-mq-2 (дата звертання: 16.02.2022)

18. Arduino.ua, Датчик вологості та температури DHT11 [Електронний ресурс] — Режим доступу: https://arduino.ua/prod185-datchik-vlajnosti-i-temperatyri-dht11 (дата звертання: 16.02.2022)
19. Arduino.ua, Датчик вологості та температури DHT22 [Електронний ресурс] — Режим доступу: https://arduino.ua/prod301-datchik-vlajnosti-i-temperatyri-dht22 (дата звертання: 19.02.2022)
20. Arduino.ua, Модуль датчика газу MQ-8 [Електронний ресурс] — Режим доступу: https://arduino.ua/prod1387-modyl-datchik-vodoroda-mq-8 (дата звертання: 21.02.2022)
21. Robostore.com.ua, Модуль датчика полум'я KY-026 для Arduino [Електронний ресурс] — Режим доступу: https://www.robostore.com.ua/ua/modul-datchika-plameni-ky-026-dlya-arduino/ (дата звертання: 29.03.2022)
22. eTechnophiles, Raspberry Pi 4 GPIO Pinout, Specs, Schematic (Detailed board layout) [Електронний ресурс] — Режим доступу: https://www.etechnophiles.com/raspberry-pi-4-gpio-pinout-specifications-and-schematic/ (дата звертання: 01.04.2022)
23. Alldatasheet.com, MPU-6050 Datasheet [Електронний ресурс] — Режим доступу: https://www.alldatasheet.com/view.jsp?Searchword=Mpu-6050&gclid=CjwKCAjw4ayUBhA4EiwATWyBrrxx3UPOecQJAz-ILBvf8yRuFPkdGp2Mn4D146FYbFOgK5CA-2daQRoChK0QAvD_BwE (дата звертання: 06.04.2022)
24. Kookye, Design a smoke detector through a raspberry pi board and MQ-2 smoke sensor [Електронний ресурс] — Режим доступу: https://kookye.com/2017/06/01/design-a-smoke-detector-through-a-raspberry-pi-board-and-mq-2-smoke-sensor/ (дата звертання: 06.04.2022)
25. DatasheetsPDF.com, MQ-2 Datasheet, Equivalent, Gas Sensor [Електронний ресурс] — Режим доступу:

https://datasheetspdf.com/pdf/622943/Hanwei/MQ-2/1 (дата звертання: 08.04.2022)

26. Arduino.ua, 10-розрядний АЦП MCP3008 для Raspberry PI та Arduino [Електронний ресурс] — Режим доступу: https://arduino.ua/prod1931-10-razryadnii-acp-mcp3008-dlya-raspberry-pi-i-arduino (дата звертання: 09.04.2022)
27. Alldatasheet.com, MCP3008 Datasheet [Електронний ресурс] — Режим доступу: https://www.alldatasheet.com/view.jsp?Searchword=Mcp3008%20datasheet&gclid=CjwKCAjw4ayUBhA4EiwATWyBrmvpFsIngvLRfUmiilP_jG6IGZoLG4QEAaM27doxQ1JI7sbbY5RQuRoCzwgQAvD_BwE (дата звертання: 11.04.2022)
28. Scand.com, How to Choose the Right Language for an IoT Project? [Електронний ресурс] — Режим доступу: https://scand.com/company/blog/top-languages-for-iot-development/ (дата звертання: 12.04.2022)
29. Docs.python.org, Python 3.10.4 documentation [Електронний ресурс] — Режим доступу: https://docs.python.org/3/ (дата звертання: 14.04.2022)
30. Readthedocs.org, Flask [Електронний ресурс] — Режим доступу: https://readthedocs.org/projects/flask/ (дата звертання: 15.04.2022)
31. Oracle, Java Documentation [Електронний ресурс] — Режим доступу: https://docs.oracle.com/en/java/ (дата звертання: 15.04.2022)
32. Spring, Spring Boot Reference Documentation [Електронний ресурс] — Режим доступу: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ (дата звертання: 19.04.2022)
33. Spring IO. Spring Data JPA [Електронний ресурс] — Режим доступу: https://spring.io/projects/spring-data-jpa (дата звертання: 21.04.2022)
34. Spring IO. Spring Data REST [Електронний ресурс] — Режим доступу: https://spring.io/projects/spring-data-rest (дата звертання: 24.04.2022)
35. Spring IO. Spring Security [Електронний ресурс] — Режим доступу: https://spring.io/projects/spring-security (дата звертання: 24.04.2022)

ДОДАТОК А

Код з логікою роботи HC-SR04

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)

TRIG = 23
ECHO = 24

from flask import Flask, jsonify

api = Flask(__name__)

@api.route('/ultrasonic', methods=['GET'])
def get_data():
    print"distance measurement in progress"
    GPIO.setup(TRIG,GPIO.OUT)
    GPIO.setup(ECHO,GPIO.IN)
    GPIO.output(TRIG,False)
    print"waiting for sensor to settle"
    time.sleep(0.2)
    GPIO.output(TRIG,True)
    time.sleep(0.00001)
    GPIO.output(TRIG,False)
    while GPIO.input(ECHO)==0:
        pulse_start=time.time()
    while GPIO.input(ECHO)==1:
        pulse_end=time.time()
    pulse_duration=pulse_end-pulse_start
    distance=pulse_duration*17150
    result=round(distance,2)
    print"distance:",distance,"cm"
    return jsonify(
        distance=result
    )

if __name__ == '__main__':
    api.run()
```

ДОДАТОК Б

Код з логікою роботи MQ-2

```
import RPi.GPIO as GPIO
import time

SPICLK = 11
SPIMISO = 9
SPIMOSI = 10
SPICS = 8
mq2_dpin = 26
mq2_apin = 0

def readadc(adcnum, clockpin, mosipin, misopin, cspin):
        if ((adcnum > 7) or (adcnum < 0)):
                return -1
        GPIO.output(cspin, True)

        GPIO.output(clockpin, False)
        GPIO.output(cspin, False)

        commandout = adcnum
        commandout |= 0x18
        commandout <<= 3
        for i in range(5):
                if (commandout & 0x80):
                        GPIO.output(mosipin, True)
                else:
                        GPIO.output(mosipin, False)
                commandout <<= 1
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)
        adcout = 0
        for i in range(12):
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)
                adcout <<= 1
                if (GPIO.input(misopin)):
                        adcout |= 0x1

        GPIO.output(cspin, True)

        adcout >>= 1
```

```
        return adcout

@api.route('/gas', methods=['GET'])
def get_gas_data():
        GPIO.setup(SPIMOSI, GPIO.OUT)
        GPIO.setup(SPIMISO, GPIO.IN)
        GPIO.setup(SPICLK, GPIO.OUT)
        GPIO.setup(SPICS, GPIO.OUT)
        GPIO.setup(mq2_dpin,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)

        COlevel=readadc(mq2_apin, SPICLK, SPIMOSI, SPIMISO, SPICS)
                return jsonify(
                  distance=str("%.2f"%((COlevel/1024.)*3.3))
                )
```

ДОДАТОК В

Структуру Maven pom.xml локального сервера

```
<?xml version="1.0" encoding="UTF-8"?>
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
tance"

xsi:schemaLocation="http://maven.apache.org/POM/4.
0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>
        <version>2.4.5</version>
        <relativePath/> <!-- lookup parent from
repository -->
    </parent>
    <groupId>ua.vatva</groupId>

<artifactId>containercontroliotsensor</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>containercontroliotsensor</name>
    <description>Demo project for Spring
Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId
>
        </dependency>
        <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-rest</artifactI
d>
        </dependency>
        <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

ДОДАТОК Г

Структуру Maven pom.xml проміжного сервера

```
<?xml version="1.0" encoding="UTF-8"?>
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
tance"

xsi:schemaLocation="http://maven.apache.org/POM/4.
0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.0</version>
        <relativePath/> <!-- lookup parent from
repository -->
    </parent>
    <groupId>ua.vatva</groupId>

<artifactId>containercontrolleriotserver</artifactId
>
    <version>0.0.1-SNAPSHOT</version>
    <name>containercontrolleriotserver</name>
    <description>Demo project for Spring
Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId
>
        </dependency>
        <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-rest</artifactI
d>
        </dependency>
        <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactI
d>
        </dependency>
        <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <!--
https://mvnrepository.com/artifact/commons-io/commo
ns-io -->
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.6</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>

<groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>
</projec
```

ДОДАТОК Д

Код контролера для отримання стану заповненості контейнера

```
@Controller
public class ContainerController {

    private final ContainerRepository containerRepository;
    private final AddressRepository addressRepository;

    public ContainerController(ContainerRepository containerRepository,
                               AddressRepository addressRepository) {
        this.containerRepository = containerRepository;
        this.addressRepository = addressRepository;
    }

    @GetMapping("/")
    public String containerList(Model model) {
        prepareContainerModel(model);
        return "container";
    }

    @GetMapping("/rest/container")
    @ResponseBody
    public List<ContainerDTO> getRestData(@RequestParam String street) {
        List<Address> addresses = addressRepository.findAllByStreet(street);
        List<Container> containers = containerRepository.findAllByAddressIn(addresses);

        for (Container container : containers) {
            String url = "http://192.168.1.103:8080/" + container.getUuid() + "/ultrasonicsensordata";
            ResponseEntity<UltrasonicData> result = new RestTemplate().getForEntity(url, UltrasonicData.class);
            if (result.getStatusCode().is2xxSuccessful() && result.getBody() != null) {
                if (container.getDefaultDistance() == 0) {
                    container.setDefaultDistance(Float.parseFloat(result.getBody().getDistance()));
                } else {
                    container.setLastDistance(Float.parseFloat(result.getBody().getDistance()));
                }
            }
        }
        List<ContainerDTO> containersDTOList = new ArrayList<>();
        List<Container> savedContainers = containerRepository.saveAll(containers);
        savedContainers.forEach(c -> containersDTOList.add(ContainerDTO.of(c)));
        return containersDTOList;
    }

    @GetMapping("/csv/container")
    public ResponseEntity<byte[]> getCsvData(@RequestParam String street) {
        List<Address> addresses = addressRepository.findAllByStreet(street);
        List<Container> containers = containerRepository.findAllByAddressIn(addresses);

        for (Container container : containers) {
            String url = "http://192.168.1.103:8080/" + container.getUuid() + "/ultrasonicsensordata";
            ResponseEntity<UltrasonicData> result = new RestTemplate().getForEntity(url, UltrasonicData.class);
            if (result.getStatusCode().is2xxSuccessful() && result.getBody() != null) {
                if (container.getDefaultDistance() == 0) {
                    container.setDefaultDistance(Float.parseFloat(result.getBody().getDistance()));
                } else {
                    container.setLastDistance(Float.parseFloat(result.getBody().getDistance()));
                }
            }
        }
```

```
    List<ContainerDTO> containersDTOList = new ArrayList<>();
    List<Container> savedContainers = containerRepository.saveAll(containers);
    savedContainers.forEach(c -> containersDTOList.add(ContainerDTO.of(c)));

    HttpHeaders csvHeaders = new HttpHeaders();
    csvHeaders.setLastModified(System.currentTimeMillis());
    csvHeaders.setContentType(MediaType.TEXT_PLAIN);
    csvHeaders.setContentDispositionFormData("data", "Containers.csv");

    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    exportContainersData(outputStream, containersDTOList);

    return new ResponseEntity<>(outputStream.toByteArray(), csvHeaders, HttpStatus.OK);
}

@PostMapping("/container")
public RedirectView getData(@RequestParam String street, Model model) {

    List<Address> addresses = addressRepository.findAllByStreet(street);
    List<Container> containers = containerRepository.findAllByAddressIn(addresses);

    for (Container container : containers) {
        String url = "http://192.168.1.103:8080/" + container.getUuid() + "/ultrasonicsensordata";
        ResponseEntity<UltrasonicData> result = new RestTemplate().getForEntity(url, UltrasonicData.class);
        if (result.getStatusCode().is2xxSuccessful() && result.getBody() != null) {
            if (container.getDefaultDistance() == 0) {
                container.setDefaultDistance(Float.parseFloat(result.getBody().getDistance()));
            } else {
                container.setLastDistance(Float.parseFloat(result.getBody().getDistance()));
            }
        }
        containerRepository.save(container);
    }
    prepareContainerModel(model);
    return new RedirectView("/");
}

@PostMapping("/")
public RedirectView addContainer(@ModelAttribute ContainerDTO containerDTO, Model model) {

    Address newAddress = new Address();
    newAddress.setCountry(containerDTO.getCountry());
    newAddress.setCity(containerDTO.getCity());
    newAddress.setArea(containerDTO.getArea());
    newAddress.setStreet(containerDTO.getStreet());
    newAddress.setHouseNumber(containerDTO.getHouseNumber());
    Address address = addressRepository.save(newAddress);

    Container container = new Container();
    container.setUuid(containerDTO.getUuid());
    container.setAddress(address);
    Container savedContainer = containerRepository.save(container);

    String url = "http://192.168.1.103:8080/" + container.getUuid() + "/ultrasonicsensordata";
    ResponseEntity<UltrasonicData> result = new RestTemplate().getForEntity(url, UltrasonicData.class);
    if (result.getStatusCode().is2xxSuccessful() && result.getBody() != null) {
        savedContainer.setDefaultDistance(Float.parseFloat(result.getBody().getDistance()));
        savedContainer.setLastDistance(savedContainer.getDefaultDistance());
        containerRepository.save(savedContainer);
    }

    prepareContainerModel(model);
    return new RedirectView("/");
}

private void prepareContainerModel(Model model) {
    List<ContainerDTO> containers = new ArrayList<>();
    containerRepository.findAll().forEach(c -> containers.add(ContainerDTO.of(c)));
```

```
        model.addAttribute("containers", containers);
        model.addAttribute("containerDTO", new ContainerDTO());
    }

    private void exportContainersData(ByteArrayOutputStream outputStream, List<ContainerDTO> containers) {
        attachContainerHeader(outputStream);

        for (ContainerDTO container : containers) {
            attachContainerDetails(outputStream, container);
        }
    }

    private void attachContainerHeader(OutputStream outputStream) {
        StringBuilder header = new StringBuilder();
        header.append("Container UUID").append(";");
        header.append("Country").append(";");
        header.append("City").append(";");
        header.append("Area").append(";");
        header.append("Street").append(";");
        header.append("House number").append(";");
        header.append("Status").append(";");
        header.append("\n");

        writeSilently(outputStream, header);
    }

    private void attachContainerDetails(OutputStream outputStream, ContainerDTO container) {

        StringBuilder line = new StringBuilder();
        line.append(container.getUuid()).append(";");
        line.append(container.getCountry()).append(";");
        line.append(container.getCity()).append(";");
        line.append(container.getArea()).append(";");
        line.append(container.getStreet()).append(";");
        line.append(container.getHouseNumber()).append(";");
        line.append(container.getStatus()).append(";");
        line.append("\n");

        writeSilently(outputStream, line);
    }

    private void writeSilently(OutputStream outputStream, StringBuilder buffer) {
        try {
            IOUtils.write(buffer.toString().getBytes(StandardCharsets.UTF_8), outputStream);
        }
        catch (IOException e) {
            //
        }
    }
}
```

ДОДАТОК Е

Веб-форма для додавання нового контейнера

```
<form method="post" action="#" th:action="@{/}" th:object="${containerDTO}"
class="m-3">
      <div class="form-inline justify-content-center">
         <div class="form-group">
            <input type="text" class="form-control" th:field="*{uuid}" id="uuid"
placeholder="Enter uuid">
         </div>
         <div class="form-group">
            <input type="text" class="form-control" th:field="*{country}" id="country"
placeholder="Enter country">
         </div>
         <div class="form-group">
            <input type="text" class="form-control" th:field="*{city}" id="city"
placeholder="Enter city">
         </div>
      </div>
      <div class="form-inline justify-content-center">
         <div class="form-group">
            <input type="text" class="form-control" th:field="*{area}" id="area"
placeholder="Enter area">
         </div>
         <div class="form-group">
            <input type="text" class="form-control" th:field="*{street}" id="street"
placeholder="Enter street">
         </div>
         <div class="form-group">
            <input type="text" class="form-control" th:field="*{houseNumber}"
id="houseNumber" placeholder="Enter house number">
         </div>
      </div>
```

```
    <div class="form-inline justify-content-center">
        <button type="submit" class="btn btn-primary">Save</button>
    </div>
</form>
```

ДОДАТОК Ж

Слайди презентації

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

ДИПЛОМНА РОБОТА
НА ТЕМУ «Модель IoT-рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів»

Виконав студент групи
ІРма-21:
Твардовський Валентин
Керівник: Кравченко О.В.

Слайд 1

Об’єкт дослідження: IoT пристрої для контролю та моніторингу рівня наповненості ємностей для твердих відходів

Мета роботи (проекту): проаналізувати існуючі IoT рішення для моніторингу та аналізу системи сміттєзберігання твердих відходів та спроектувати власний пристрій

Практичне значення роботи: результати здійснених у проекті досліджень можуть бути використані при проектуванні IoT пристроїв з подібним функціоналом, а спроектоване та розроблене рішення може бути завершене і доповнене для практичного використання і в комерційних цілях

Слайд 2

Актуальність роботи

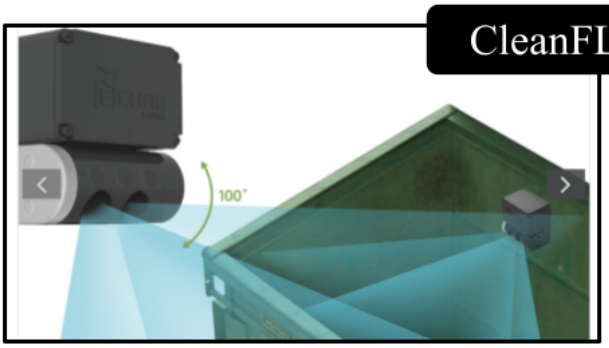
Збір статистики не автоматизований, а найчастіше відсутній зовсім, внаслідок цього оптимізація процесу збору відходів неможлива.

Відсутність даних рівня заповнення контейнерів, призводить до зайвих виїздів сміттєвозів і підвищення собівартості вивезення відходів для регіональних операторів і не вигідних тарифів для клієнтів

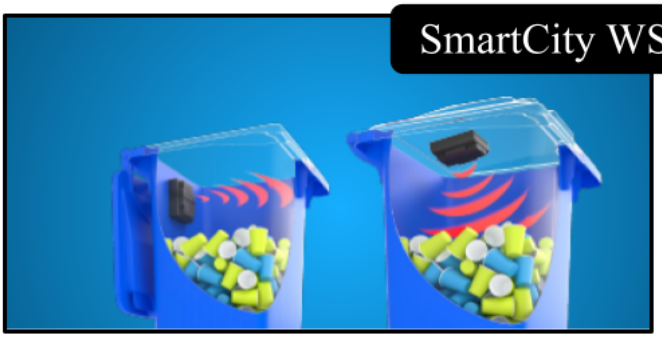
Переповнення контейнерів призводить: до зростання антисанітарії, створення локальних звалищ, доступу тварин і птахів до відходів.

Слайд 3

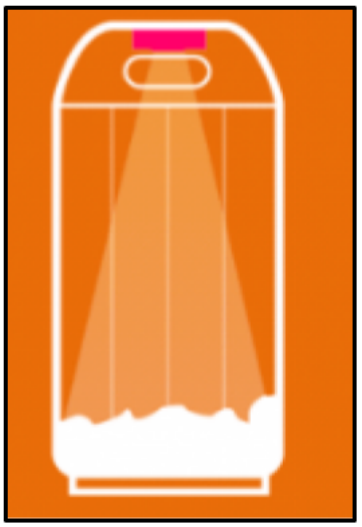
CleanFLEX



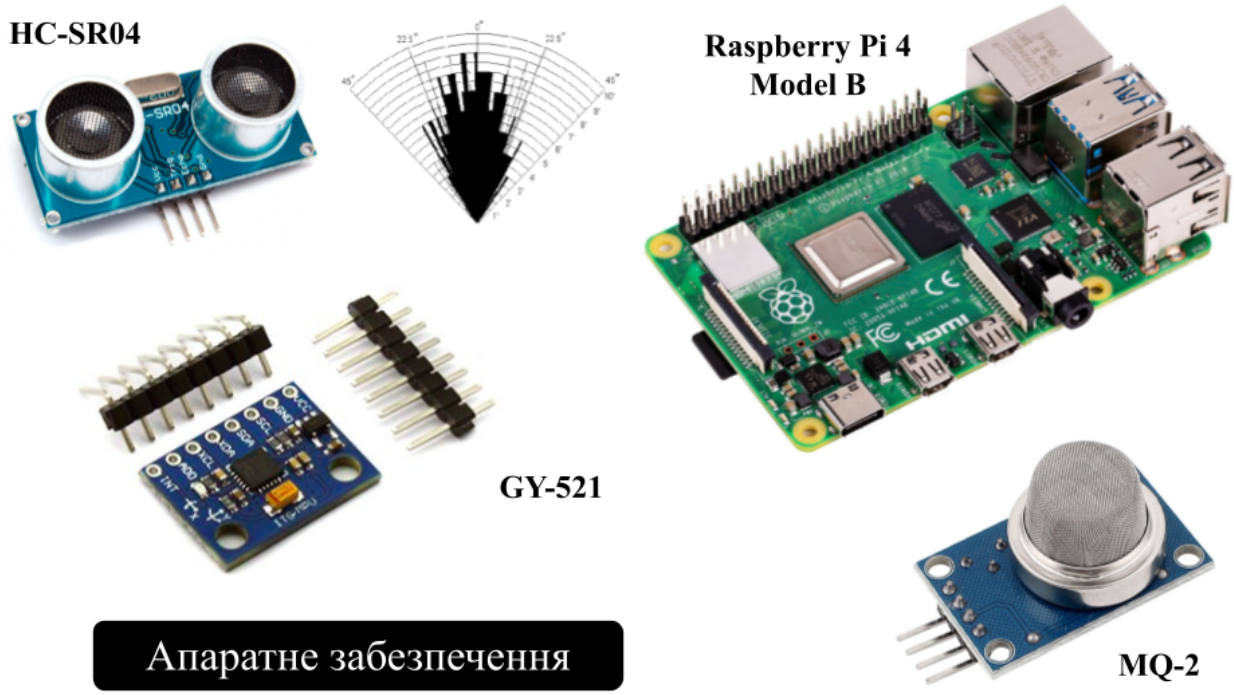
SmartCity WSens



BrighterBins

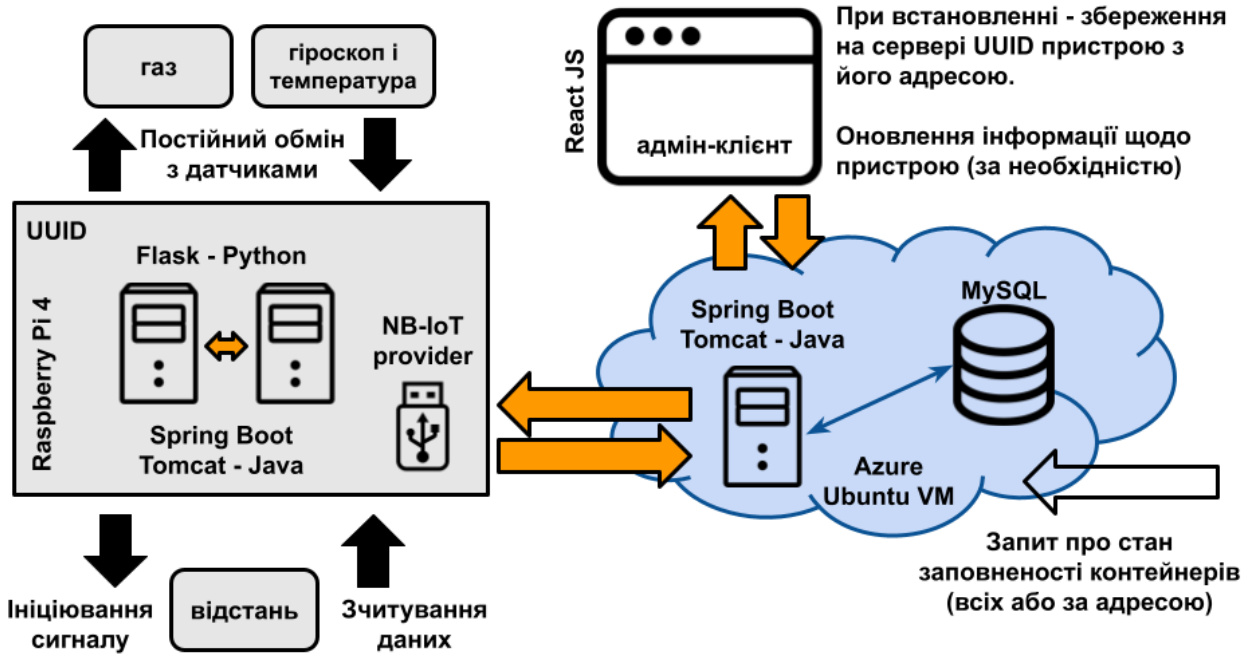


Слайд 4

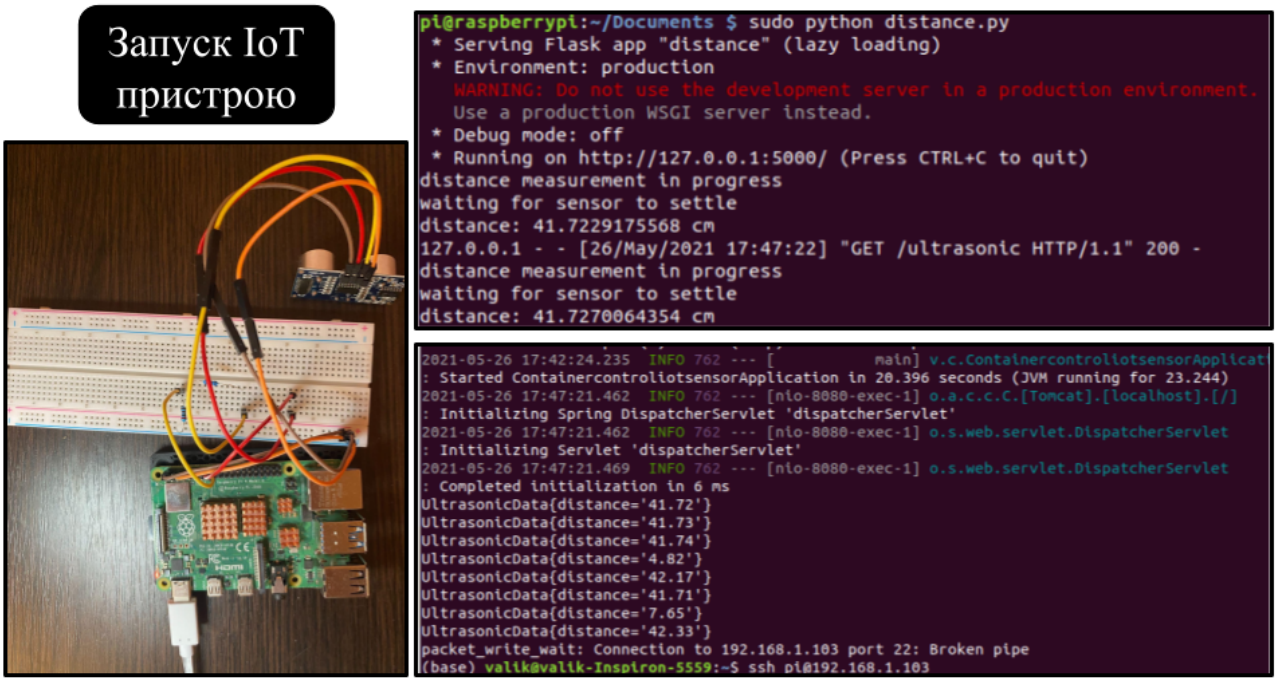


Слайд 5

Слайд 6



Слайд 7

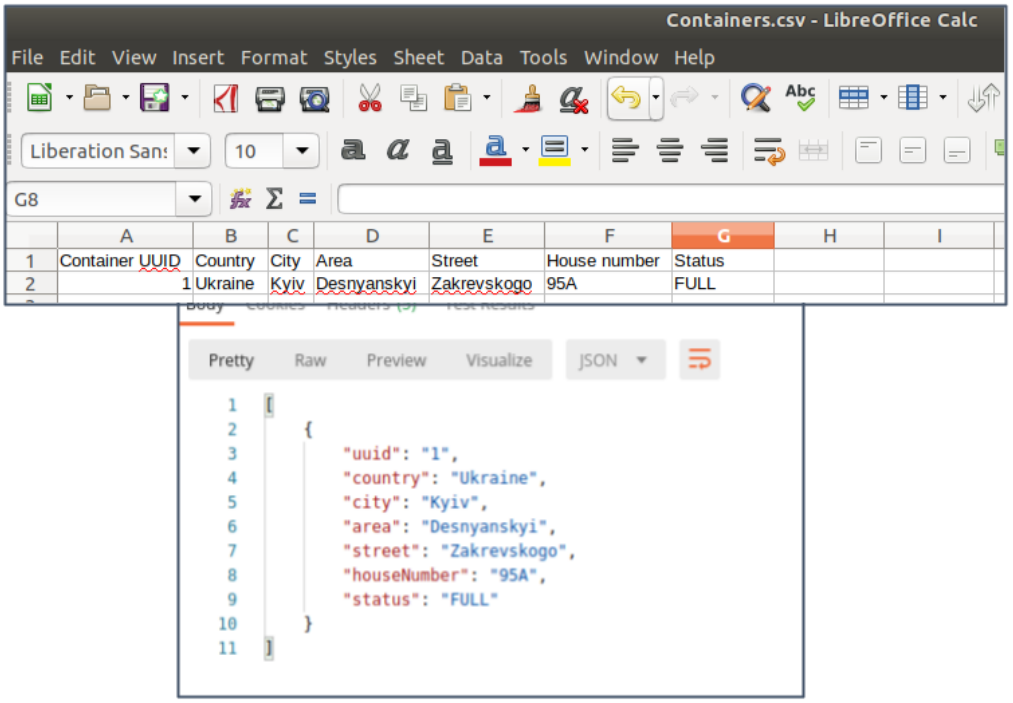


Слайд 8

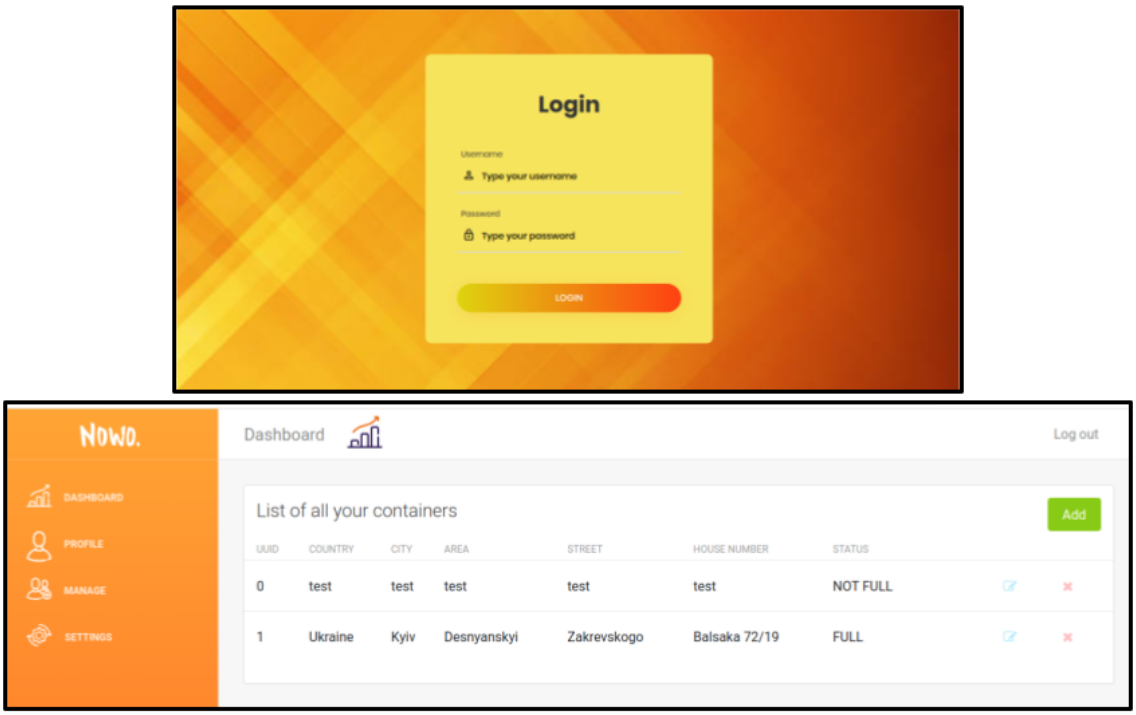
Запуск IoT пристрою

```
------------------------------
Temp : 24.8123529411764 7

Acc X : 7.953537915039062
Acc Y : 5.480327600097656
Acc Z : 2.04943662109375

Gyro X : -1.1908396946564885
Gyro Y : 3.66412213740458
Gyro Z : 2.213740458015267

------------------------------
Temp : 25.04764705882353

Acc X : 8.113949426269532
Acc Y : 5.329492895507812
Acc Z : 2.2864625854492187

Gyro X : -2.618320610687023
Gyro Y : 3.2748091603053435
Gyro Z : -0.648854961832061
```

Слайд 9



Слайд 10



Слайд 11

Висновки

Виконано:

- Аналіз датчиків та пристроїв, що можна використати для проектування пристрою.
- Аналіз існуючих систем IoT рішень, завдяки яким можна отримати оцінку рівня заповненості ємності для твердих відходів.
- Дослідження комунікаційних систем та технологій.
- Проектування логічних зв’язків вузлів системи.
- Розроблено структурну схему системи IoT пристрою.
- Створено апаратне та програмне забезпечення системи IoT пристрою.
- За матеріалами роботи опубліковано статтю та тези.

Результати здійснених у проекті досліджень можуть бути використані при проектуванні IoT пристроїв з подібним функціоналом, а спроектоване та розроблене рішення може бути завершене і доповнене для практичного використання і в комерційних цілях

Слайд 12