МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА
Навчально-науковий інститут енергетичної, інформаційної та транспортної інфраструктури
Кафедра автоматизації та комп'ютерно-інтегрованих технологій

РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНА ЗАПИСКА ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

на тему: Розробка багатоагентної архітектури IoT-системи для надання спеціалізованих послуг

Виконав: здобувач вищої освіти
4 курсу, групи Сінж-2022-1
курс, група
151 «Автоматизація та комп'ютерно-інтегровані технології»
напрям підготовки (спеціальність)
Безнос Владислав Сергійович
(прізвище та ініціали)
Керівник Піддубна Л.В., доц. каф. АКІТ
(прізвище та ініціали, наук. ступ., вч. звання)
Рецензент Ківіренко О.Б., начальник виробництва ТОВ «Альфа-Композіт»
(прізвище та ініціали, наук. ступ., вч. звання)

Харків – 2026

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА

Навчально-науковий інститут енергетичної, інформаційної та транспортної інфраструктури

Кафедра автоматизації та комп'ютерно-інтегрованих технологій
Освітньо-кваліфікаційний рівень – бакалавр
Галузь знань 15 «Автоматизація та приладобудування»
Спеціальність 151 «Автоматизація та комп'ютерно-інтегровані технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри АКІТ
_____ БАРАНОВ О.О.
« 19 » червня 2026 року

З А В Д А Н Н Я
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Безнос Владислав Сергійович

1.Тема роботи: Розробка багатоагентної архітектури IoT-системи для надання спеціалізованих послуг

Затверджена наказом університету від « 22 » травня 2026 року № 440-03.
Керівник роботи Піддубна Л.В., кандидат філософських наук, доцент, доцент кафедри АКІТ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
2. Строк подання роботи здобувачем вищої освіти «15» червня 2026 р.
3. Вихідні дані до роботи IoT-система для надання спеціалізованих послуг
4. Зміст розрахунково пояснювальної записки (перелік питань, які потрібно розробити): Вступ. Аналіз теоретичних основ та існуючих рішень. Проєктування багатоагентної IoT-системи. Програмна реалізація багатоагентної IoT-системи. Охорона праці. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація.

6. Консультанти розділів проєкту (роботи)

Розділ	Консультант	Підпис, дата	
		завдання видав	завдання прийняв
Аналіз проблеми	Піддубна Л.В.	11.05.2026	21.05.2026
Основна частина	Піддубна Л.В.	21.05.2026	27.05.2026
Спеціальний розділ	Піддубна Л.В.	27.05.2026	03.06.2026
Охорона праці	Малишева В.В.	01.06.2026	03.06.2026

7. Дата видачі завдання «_11_» травня_2026_р.

Керівник _____Піддубна Л.В
(підпис)

Завдання прийняв до виконання _____Безнос В. С.
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання розділів	Примітка
1	Розробка 1 го розділу бакалаврської роботи	11.05.2026 - 21.05.2026	
2	Розробка 2 го розділу бакалаврської роботи	21.05.2026 - 27.05.2026	
3	Розробка 3 го розділу бакалаврської роботи	27.05.2026 - 03.06.2026	
4	Розробка 4 го розділу з охорони праці	01.06.2026 - 03.06.2026	
5	Рецензування бакалаврської роботи	15.06.2026	Ківіренко О.Б
6	Захист на ДЕК	24.06.2026	

Здобувача вищої освіти _____Безнос В. С._
(підпис)

Керівник _____ Піддубна Л.В
(підпис)

РЕФЕРАТ

Розробка багатоагентної архітектури IoT-системи для надання спеціалізованих послуг – Безнос Владислав Сергійович, дипломна робота бакалавра, Харків, Харківський національний університет міського господарства імені О.М. Бекетова, кількість сторінок 96, кількість таблиць 39, кількість рисунків 10, кількість джерел літератури 34.

Актуальність дослідження. Інтернет речей об’єднує значну кількість пристроїв, які повинні взаємодіяти між собою та своєчасно надавати послуги у розподіленому середовищі. Постійне зростання кількості підключених пристроїв ускладнює процеси обробки даних, координації компонентів і забезпечення сервісів у режимі реального часу. Використання багатоагентної архітектури створює можливість організувати автономну та узгоджену роботу елементів системи, що підвищує ефективність і надійність надання спеціалізованих послуг.

Об’єктом дослідження є процеси побудови та функціонування багатоагентних IoT-систем.

Предметом дослідження є моделі, методи та архітектурні рішення організації взаємодії агентів у IoT-системі для надання спеціалізованих послуг.

Метою роботи є розробка багатоагентної архітектури IoT-системи, що забезпечує ефективну координацію агентів та адаптивне надання спеціалізованих послуг.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- Проаналізувати сучасні підходи до побудови IoT-систем та багатоагентних архітектур.
- Дослідити принципи взаємодії програмних агентів у розподіленому середовищі.
- Визначити вимоги до архітектури IoT-системи для надання спеціалізованих послуг.
- Розробити структурну модель багатоагентної IoT-системи.

У роботі використовуються методи системного аналізу для дослідження структури та взаємодії компонентів IoT-системи; методи моделювання для побудови багатоагентної архітектури; об’єктно-орієнтоване проєктування для розробки програмної моделі агентів; методи порівняльного аналізу для оцінки існуючих рішень.

КЛЮЧОВІ СЛОВА: Інтернет речей, IoT-система, програмні агенти, багатоагентна архітектура.

ABSTRACT

Development of a multi-agent architecture of an IoT system for providing specialized services – Beznos Vladyslav Serhiyovych, bachelor's thesis, Kharkiv, O. M. Beketov National University of Urban Economy in Kharkiv, number of pages 96, number of tables 39, number of figures 10, number of literature sources 34.

Relevance of the research. The Internet of Things unites a significant number of devices that must interact with each other and provide services in a timely manner in a distributed environment. The constant growth in the number of connected devices complicates the processes of data processing, coordination of components and provision of services in real time. The use of multi-agent architecture creates the opportunity to organize autonomous and coordinated operation of system elements, which increases the efficiency and reliability of the provision of specialized services.

The object of the research is the processes of building and functioning of multi-agent IoT systems.

The subject of the research is models, methods and architectural solutions for organizing the interaction of agents in an IoT system for the provision of specialized services.

The aim of the work is to develop a multi-agent architecture for an IoT system that ensures effective coordination of agents and adaptive provision of specialized services.

To achieve this goal, the following tasks must be solved:

• Analyze modern approaches to building IoT systems and multi-agent architectures.

• Investigate the principles of interaction of software agents in a distributed environment.

• Determine the requirements for the architecture of an IoT system for providing specialized services.

• Develop a structural model of a multi-agent IoT system.

The work uses systems analysis methods to study the structure and interaction of IoT system components; modeling methods to build a multi-agent architecture; object-oriented design to develop a software model of agents; and comparative analysis methods to evaluate existing solutions.

KEYWORDS: Internet of Things, IoT system, software agents, multi-agent architecture.

ЗМІСТ

Перелік умовних позначень, скорочень і термінів

IoT-системи – Internet of Things- системи

КІСУ – Комп’ютерно-інтегровані системи управління

BI – Business Intelligence

AI – штучний інтелект

MQTT – Message Queuing Telemetry Transport

ВСТУП

Кількість пристроїв, підключених до мережі, стрімко зростає. Датчики, контролери, мобільні пристрої та промислове обладнання постійно генерують великі обсяги даних. У таких умовах централізоване керування стає менш ефективним, оскільки виникають затримки обробки інформації, перевантаження серверів і складність масштабування системи. IoT-системи, які працюють у промисловості, транспорті або сервісних мережах, повинні швидко реагувати на зміни середовища та забезпечувати надання послуг без перебоїв. Для цього необхідні механізми автономного прийняття рішень та координації між компонентами системи.

Багатоагентний підхід дозволяє розподілити функції між окремими програмними агентами, які можуть взаємодіяти між собою, аналізувати дані та адаптувати свою поведінку. Така архітектура підвищує гнучкість системи, спрощує її розширення та покращує стійкість до відмов.

Отже, розробка багатоагентної архітектури IoT-системи для надання спеціалізованих послуг є актуальною задачею, оскільки вона спрямована на підвищення ефективності роботи розподілених технічних систем та оптимізацію процесів обробки даних.

Об’єктом дослідження є процес функціонування розподілених IoT-систем із використанням програмних агентів.

Предметом дослідження є моделі, методи та архітектурні рішення організації взаємодії агентів у IoT-системі під час надання спеціалізованих послуг.

Метою роботи є створення моделі багатоагентної IoT-системи, яка забезпечує узгоджену роботу агентів і підтримує адаптивне надання послуг у змінному середовищі.

Для досягнення поставленої мети необхідно:

- Проаналізувати існуючі архітектурні підходи до побудови IoT-систем.

- Дослідити принципи організації багатоагентних систем.
- Визначити функціональні вимоги до системи надання спеціалізованих послуг.
- Розробити структурну схему взаємодії агентів.

У роботі застосовано методи системного підходу для аналізу структури IoT-системи, методи моделювання для побудови архітектури, принципи об’єктно-орієнтованого проєктування для реалізації агентів, а також експериментальні дослідження для оцінювання ефективності функціонування розробленої моделі.

РОЗДІЛ 1 АНАЛІЗ ТЕОРЕТИЧНИХ ОСНОВ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Особливості побудови IoT-систем

Інтернет речей – це концепція побудови інформаційних систем, у яких фізичні об’єкти підключені до мережі та можуть передавати дані. Такими об’єктами є датчики, контролери, виконавчі механізми, мобільні пристрої та промислове обладнання. Вони збирають інформацію про стан середовища або процесу та передають її для обробки.

IoT-система складається з апаратних і програмних компонентів. Апаратна частина забезпечує збір і передачу даних. Програмна частина відповідає за збереження, аналіз і керування пристроями. Важливою особливістю IoT є розподілений характер системи. Пристрої можуть бути розташовані у різних місцях і працювати в різних мережах.

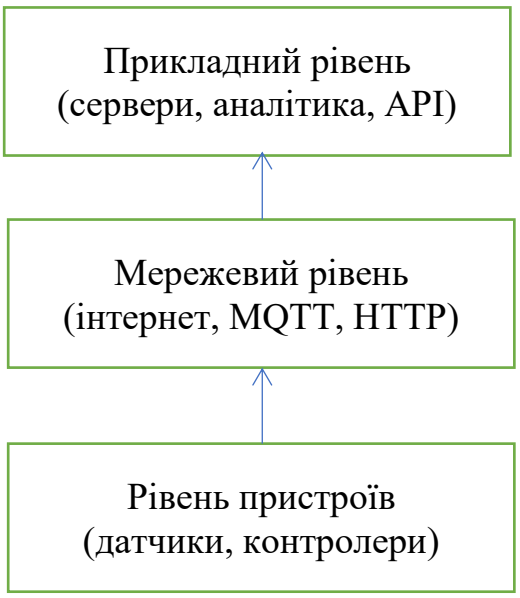


Рисунок 1.1 – Загальна структура IoT-системи

Основною метою побудови IoT-системи є автоматизація процесів та надання сервісів на основі отриманих даних. Наприклад, система може контролювати температуру в приміщенні, відстежувати місцезнаходження транспорту або здійснювати моніторинг виробничого обладнання.

IoT-системи мають багаторівневу структуру. Найчастіше під час побудови IoT-систем застосовується трирівнева модель, оскільки вона є простою для розуміння та реалізації. Кожен рівень виконує свої чіткі функції.

Перший рівень – рівень пристроїв. На цьому рівні працюють фізичні елементи системи. Це датчики, контролери та виконавчі механізми. Датчики вимірюють фізичні параметри, наприклад температуру, тиск, освітленість або рух. Контролери обробляють сигнали від датчиків та готують їх до передачі. Виконавчі механізми отримують команди і виконують дію, наприклад вмикають двигун або відкривають клапан. Цей рівень безпосередньо взаємодіє з реальним середовищем. Дані передаються через мережу на сервер або в хмарну платформу.

Другий рівень – мережевий рівень. Він забезпечує передачу даних від пристроїв до сервера і назад. Для цього використовуються протоколи зв’язку та дротові або бездротові мережі. Це може бути Wi-Fi, Ethernet, мобільний зв’язок або спеціалізовані протоколи, наприклад MQTT. Вибір протоколу залежить від вимог до швидкості, надійності та енергоспоживання. Мережевий рівень відповідає за маршрутизацію повідомлень, їх доставку та інколи базове шифрування.

Третій рівень – прикладний рівень, який відповідає за обробку даних, зберігання інформації та надання користувацьких сервісів**.** На цьому рівні знаходиться серверна частина системи. Сервер може бути розміщений локально або у хмарному середовищі та виконує такі функції, як прийом даних від пристроїв; збереження їх у базі даних; обробку інформації за заданими алгоритмами; формування керуючих команд; надання доступу користувачам через веб-інтерфейс або API. Тут працюють бази даних, аналітичні модулі та веб-інтерфейси. Наприклад, якщо датчик температури передає значення 85°C, сервер аналізує це значення. Якщо воно перевищує допустиму норму, система автоматично надсилає команду на виконавчий механізм або формує повідомлення для оператора.

Трирівнева модель дозволяє логічно розділити функції: пристрої збирають дані, мережа передає їх, а сервер обробляє і керує системою. Такий підхід спрощує проєктування, масштабування та підтримку IoT-системи.

У таблиці 1.1 наведено основні компоненти IoT-системи.

Таблиця 1.1 – Основні компоненти IoT-системи

Компонент	Призначення	Приклад
Датчик	Збір фізичних параметрів	Датчик температури
Контролер	Обробка сигналів	Arduino, ESP32
Мережевий модуль	Передача даних	Wi-Fi, GSM, LoRa
Сервер	Збереження та обробка даних	Хмарна платформа
Користувацький інтерфейс	Відображення інформації	Веб-додаток

Важливою особливістю IoT є масштабованість. Система повинна підтримувати підключення великої кількості пристроїв. При збільшенні кількості сенсорів продуктивність не повинна значно знижуватися.

Ще однією особливістю є обмежені ресурси пристроїв. Багато датчиків мають невелику пам’ять і низьку обчислювальну потужність. Тому обробка даних часто виконується на сервері або у хмарному середовищі.

Існують різні архітектурні підходи до побудови IoT-систем. Основними є централізована, розподілена та гібридна архітектури.

Таблиця 1.2 – Порівняння архітектур IoT-систем

Тип архітектури	Переваги	Недоліки
Централізована	Простота реалізації	Перевантаження сервера
Розподілена	Висока відмовостійкість	Складність керування
Гібридна	Баланс між гнучкістю і стабільністю	Потребує складного налаштування

Централізована модель передбачає, що всі дані надходять до одного центрального сервера. Такий підхід простий у реалізації, але має обмеження щодо продуктивності.

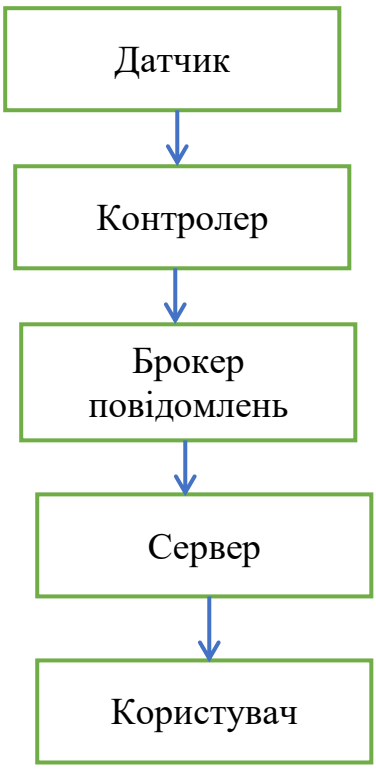


Рисунок 1.2 – Взаємодія компонентів IoT-системи

Таблиця 1.3 – Основні вимоги до IoT-систем

Вимога	Пояснення
Масштабованість	Підтримка великої кількості пристроїв
Надійність	Стійкість до відмов
Безпека	Захист даних і пристроїв
Гнучкість	Можливість розширення функцій
Енергоефективність	Мінімальне споживання енергії

Розподілена модель дозволяє виконувати обробку даних на різних вузлах мережі. Це зменшує навантаження на центральний сервер та підвищує стійкість до відмов.

Гібридна модель поєднує обидва підходи. Частина обробки виконується на периферійних пристроях, а частина – у хмарі. Такий підхід часто використовується у промислових IoT-системах.

У сучасних IoT-системах важливу роль відіграє безпека. Дані повинні передаватися у зашифрованому вигляді. Необхідно забезпечити автентифікацію пристроїв і контроль доступу до інформації.

Побудова IoT-системи потребує комплексного підходу. Необхідно враховувати структуру мережі, вибір протоколів, вимоги до безпеки та можливість масштабування. Архітектура повинна забезпечувати стабільну роботу пристроїв, швидку передачу даних і можливість інтеграції нових компонентів. Саме ці особливості визначають подальше проєктування багатоагентної IoT-системи для надання спеціалізованих послуг.

Принципи функціонування Інтернету речей базуються на взаємодії фізичних об’єктів через мережу з метою збору, передавання та обробки даних.

Перший принцип – це ідентифікація пристроїв. Кожен пристрій у системі повинен мати унікальний ідентифікатор. Це може бути IP-адреса, MAC-адреса або інший унікальний код. Завдяки цьому система розуміє, від якого саме пристрою надійшли дані.

Другий принцип – збір даних із фізичного середовища. Датчики вимірюють параметри навколишнього середовища або стан обладнання. Це можуть бути температура, вологість, тиск, швидкість, рівень освітлення або інші показники. Пристрої перетворюють фізичні величини у цифровий сигнал.

Третій принцип – передача даних через мережу. Отримана інформація надсилається до сервера або іншого вузла обробки. Передача може здійснюватися через локальну мережу або через Інтернет. Для цього використовуються спеціальні протоколи обміну даними.

Четвертий принцип – обробка та аналіз інформації. Дані, які надходять від пристроїв, зберігаються та аналізуються. Система може виявляти відхилення, порівнювати показники з нормою або виконувати розрахунки. На цьому етапі приймаються рішення.

П’ятий принцип – зворотний зв’язок і керування. Після аналізу система може сформувати команду для виконавчого механізму. Наприклад, увімкнути охолодження, змінити режим роботи або надіслати повідомлення оператору. Таким чином забезпечується автоматичне керування процесом.

Шостий принцип – автономність роботи. IoT-системи можуть функціонувати без постійного втручання людини. Пристрої виконують запрограмовані алгоритми та реагують на зміну параметрів.

Сьомий принцип – масштабованість. Система повинна підтримувати підключення нових пристроїв без значної зміни архітектури. Це дозволяє розширювати функціональні можливості без повної перебудови системи.

Важливим принципом є безпека. Дані повинні передаватися захищеними каналами, а доступ до системи має бути обмеженим, що запобігає несанкціонованому втручанню.

Інтернет речей функціонує за рахунок поєднання процесів ідентифікації пристроїв, збору даних, їх передачі, аналізу та формування керуючих дій. Усі ці етапи працюють як єдиний безперервний цикл обміну інформацією та управління.

1.2 Багатоагентні системи та їх архітектурні підходи

Багатоагентна система – це система, яка складається з кількох автономних програмних агентів. Кожен агент виконує певні функції та взаємодіє з іншими агентами. Такі системи застосовуються у розподілених середовищах, де необхідна координація великої кількості компонентів.

Програмний агент – це програмний модуль, який здатний самостійно приймати рішення в межах заданих правил. Агент має власний стан, може аналізувати інформацію та виконувати дії без постійного втручання людини. Він отримує дані з середовища, обробляє їх та реагує відповідно до алгоритму.

Основними властивостями програмного агента є автономність, здатність до взаємодії, реактивність та цілеспрямованість. Автономність означає, що агент працює самостійно. Реактивність означає здатність реагувати на зміни середовища. Цілеспрямованість означає, що агент має певну мету. Здатність до взаємодії дозволяє агентам обмінюватися повідомленнями.

Таблиця 1.4 – Основні властивості програмних агентів

Властивість	Пояснення
Автономність	Самостійне прийняття рішень
Реактивність	Реагування на події середовища
Проактивність	Ініціювання власних дій
Соціальність	Взаємодія з іншими агентами
Адаптивність	Здатність змінювати поведінку

У багатоагентній системі кожен агент виконує окрему роль. Наприклад, один агент може відповідати за збір даних, інший – за аналіз інформації, третій – за прийняття рішень. Такий розподіл функцій підвищує гнучкість системи. Існують різні типи агентів (див. табл. 1.5). Вони відрізняються рівнем складності та способом прийняття рішень.

Таблиця 1.5 – Типи програмних агентів

Тип агента	Характеристика
Реактивний	Діє за правилом «подія – реакція»
Делібертивний	Використовує модель середовища та планування
Гібридний	Поєднує реактивні та планувальні механізми
Мобільний	Може переміщуватися між вузлами мережі
Сервісний	Надає певну функцію іншим агентам

Реактивні агенти працюють за простими правилами. Вони швидко реагують на події, але не виконують складного аналізу. Делібертивні агенти

мають внутрішню модель середовища та можуть планувати свої дії. Гібридні агенти поєднують обидва підходи.

У розподіленому середовищі агенти повинні взаємодіяти між собою. Для цього використовуються повідомлення. Повідомлення містить інформацію про подію або запит на виконання дії. Взаємодія може бути синхронною або асинхронною.

Синхронна взаємодія означає, що агент очікує відповідь після надсилання запиту. Асинхронна взаємодія означає, що агент продовжує роботу і отримує відповідь пізніше.

Таблиця 1.6 – Моделі взаємодії агентів

Модель взаємодії	Опис
Запит–відповідь	Один агент надсилає запит, інший повертає результат
Публікація–підписка	Агент публікує подію, інші агенти отримують її
Переговори	Агенти узгоджують спільне рішення
Кооперація	Агенти спільно виконують задачу
Конкуренція	Агенти змагаються за ресурси

Для координації дій застосовуються спеціальні механізми. Один із підходів – централізована координація. У цьому випадку існує головний агент, який керує іншими. Інший підхід – децентралізована координація. У цьому випадку агенти самі домовляються між собою без центрального керівника.

Таблиця 1.7 – Підходи до координації агентів

Підхід	Переваги	Недоліки
Централізований	Просте керування	Залежність від одного вузла
Децентралізований	Висока стійкість	Складність узгодження
Ієрархічний	Чітка структура	Можлива затримка рішень

Багатоагентні архітектури можуть бути організовані за різними принципами. Найпоширенішими є централізована, ієрархічна та повністю розподілена архітектури.

У централізованій архітектурі всі агенти підпорядковуються одному координатору. У розподіленій архітектурі агенти рівноправні. Вони приймають рішення на основі локальної інформації. Ієрархічна архітектура поєднує обидва підходи.

Багатоагентні системи добре підходять для IoT-середовищ. У таких системах велика кількість пристроїв повинна працювати одночасно. Агенти можуть відповідати за обробку даних від окремих пристроїв, за управління ресурсами або за надання сервісів.

Перевагами багатоагентних систем є гнучкість, масштабованість і відмовостійкість. Якщо один агент припиняє роботу, інші можуть продовжувати виконання задач. Недоліком є складність реалізації та необхідність узгодження дій між агентами.

Багатоагентні системи є ефективним підходом до побудови розподілених інформаційних систем. Вони дозволяють розподілити функції між автономними компонентами, забезпечити координацію дій та підвищити стійкість до відмов. Саме ці особливості роблять багатоагентний підхід доцільним для використання в IoT-системах.

У розподіленому середовищі агенти працюють на різних вузлах мережі. Вони можуть бути фізично віддалені один від одного. Тому для їх узгодженої роботи необхідні чіткі правила взаємодії та координації.

Однією з основних моделей взаємодії є обмін повідомленнями. Агенти не мають прямого доступу до внутрішнього стану один одного. Вони передають інформацію через стандартизовані повідомлення. Повідомлення містить відправника, отримувача, тип запиту та зміст.

У розподілених системах важливо, щоб передача повідомлень була надійною. Для цього використовуються механізми підтвердження доставки.

Якщо відповідь не отримана, агент може повторити запит, що підвищує стійкість системи до збоїв.

Ще однією моделлю є взаємодія через спільне середовище. У цьому випадку агенти не надсилають повідомлення напряму. Вони взаємодіють через спільний ресурс, наприклад базу даних або спільну чергу подій. Один агент записує інформацію, а інший її зчитує. Такий підхід зменшує кількість прямих зв’язків між агентами.

Таблиця 1.8 – Способи взаємодії агентів у розподіленому середовищі

Спосіб взаємодії	Опис	Особливість
Прямий обмін повідомленнями	Агент надсилає повідомлення конкретному отримувачу	Чітке адресування
Через брокер повідомлень	Повідомлення передається через посередника	Менша залежність агентів
Через спільне сховище	Агенти читають і записують дані в спільний ресурс	Простота реалізації
Публікація–підписка	Агенти отримують повідомлення за темами	Масштабованість

Важливою частиною взаємодії є координація дій. Координація означає узгодження поведінки агентів для досягнення спільної мети. Без координації можливі конфлікти або дублювання задач. У розподіленому середовищі застосовуються різні механізми координації. Один із них – протокол переговорів. Агенти обмінюються повідомленнями, поки не досягнуть згоди щодо розподілу задач або ресурсів. Наприклад, один агент пропонує виконати задачу, а інший погоджується або відмовляється.

Іншим механізмом є аукціонна модель. Агент-координатор оголошує задачу. Інші агенти надсилають свої пропозиції щодо її виконання. Координатор обирає найкращу пропозицію. Такий підхід часто використовується для розподілу ресурсів.

Таблиця 1.9 – Механізми координації агентів

Механізм	Принцип роботи	Перевага
Переговори	Узгодження умов через обмін повідомленнями	Гнучкість
Аукціон	Вибір виконавця на основі пропозицій	Ефективний розподіл ресурсів
Контрактна модель	Призначення задач через систему контрактів	Чіткість ролей
Голосування	Прийняття рішення більшістю	Простота

У розподілених системах також застосовується модель самокоординації. У цьому випадку агенти приймають рішення на основі локальної інформації без центрального керівника. Вони орієнтуються на правила або обмеження, задані в системі. Такий підхід підвищує відмовостійкість. Ще одним важливим аспектом є синхронізація часу. У розподіленому середовищі різні вузли можуть мати різний системний час. Для узгодження подій використовуються механізми часових міток, що дозволяє правильно впорядковувати події. Також необхідно враховувати можливість відмов окремих агентів. Система повинна мати механізми відновлення. Наприклад, якщо агент перестає відповідати, його функції може тимчасово виконувати інший агент.

Таблиця 1.10 – Особливості координації в розподіленому середовищі

Особливість	Пояснення
Відсутність спільної пам’яті	Кожен агент має власний стан
Можливі затримки мережі	Повідомлення можуть приходити із запізненням
Часткові відмови	Окремі вузли можуть бути недоступні
Необхідність синхронізації	Події повинні впорядковуватися в часі

Моделі взаємодії та координації визначають ефективність роботи багатоагентної системи в розподіленому середовищі. Вони забезпечують обмін інформацією, узгодження рішень і стабільну роботу навіть у разі збоїв. Саме правильний вибір механізмів взаємодії дозволяє побудувати масштабовану та надійну IoT-систему.

Розглянемо декілька архітектурних моделей багатоагентних систем, серед яких реактивна, ієрархічна та гібридна архітектури (таблиця 1.11).

Таблиця 1.11 – Архітектурні моделі багатоагентних систем

Архітектурна модель	Основна характеристика	Принцип роботи	Переваги	Недоліки	Приклади застосування
Реактивна архітектура	Агенти швидко реагують на події середовища без складного планування	Робота базується на правилах типу «подія – реакція»	Висока швидкодія, простота реалізації, низьке споживання ресурсів	Обмежені можливості аналізу та планування, складність реалізації складної логіки	Системи моніторингу, автоматичне керування датчиками, IoT-пристрої реального часу
Ієрархічна архітектура	Система має структуру підпорядкування агентів	Центральний або керуючий агент координує роботу інших агентів	Зручне управління, чіткий розподіл ролей, простий контроль системи	Залежність від центрального агента, можливе перевантаження керівного рівня	Промислові системи управління, корпоративні мережі, системи диспетчеризації
Гібридна архітектура	Поєднує реактивний та ієрархічний підходи	Частина агентів працює автономно, а частина координується центральними агентами	Гнучкість, масштабованість, адаптивність, поєднання швидкодії та керованості	Складніша реалізація та налаштування	Інтелектуальні IoT-системи, розумні будівлі, адаптивні системи автоматизації

Для реалізації даної системи було обрано гібридний підхід, оскільки він дозволяє поєднати автономність агентів із централізованою координацією окремих процесів.

1.3 Аналіз сучасних платформ та технологій для реалізації IoT і багатоагентних систем

Для створення IoT-систем та багатоагентних рішень використовуються спеціалізовані програмні платформи. Вони забезпечують підключення

пристроїв, обробку даних, керування сервісами та організацію взаємодії агентів. Вибір платформи впливає на масштабованість, надійність і складність реалізації системи. Сучасні рішення можна умовно поділити на три групи, що зображено на рисунку 1.3.

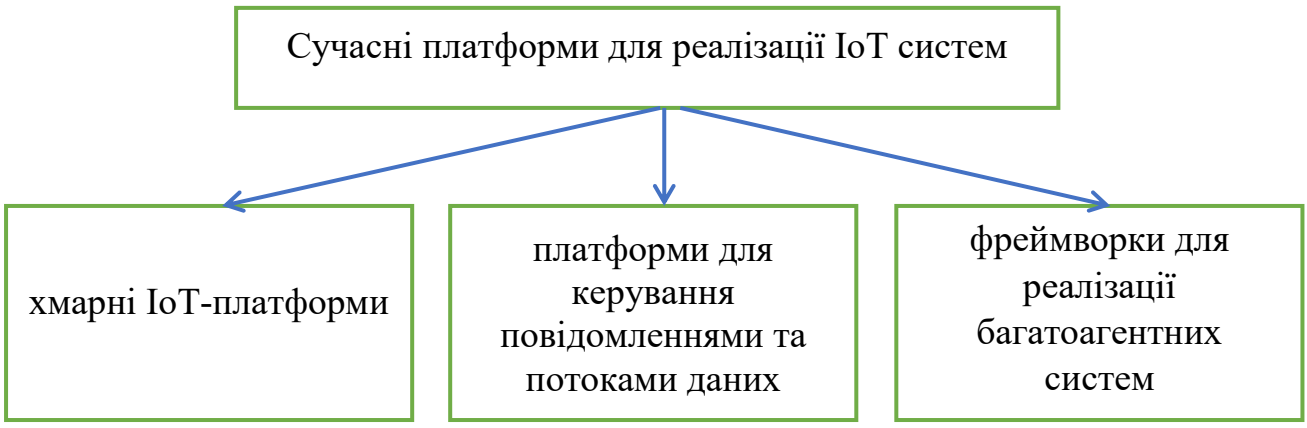


Рисунок 1.3 – Сучасні рішення для створення IoT-систем

Хмарні IoT-платформи дозволяють підключати пристрої через Інтернет, зберігати дані та керувати ними через веб-інтерфейс. Вони забезпечують готову інфраструктуру, що зменшує складність розробки.

Найпоширенішими є такі платформи, як Amazon Web Services (AWS IoT Core), Microsoft Azure (Azure IoT Hub), Google Cloud (Google Cloud IoT). Ці платформи підтримують масштабування, захищене підключення пристроїв та інтеграцію з іншими сервісами.

Таблиця 1.11 – Порівняння хмарних IoT-платформ

Платформа	Переваги	Недоліки
AWS IoT Core	Висока масштабованість, інтеграція з іншими сервісами AWS	Складність налаштування, платна модель
Azure IoT Hub	Гнучке керування пристроями, підтримка аналітики	Висока вартість при великій кількості пристроїв
Google Cloud IoT	Зручна інтеграція з	Обмеження в

Платформа	Переваги	Недоліки
	аналітичними сервісами	безкоштовному тарифі

Перевагою хмарних платформ є швидкий старт. Розробник не створює серверну інфраструктуру з нуля. Недоліком є залежність від постачальника послуг і постійні витрати на використання ресурсів.

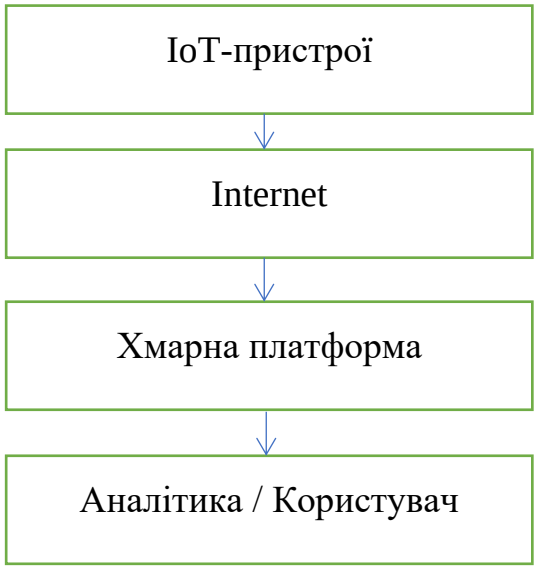


Рисунок 1.4 – Загальна схема використання хмарної IoT-платформи

Для взаємодії між компонентами IoT-систем часто застосовуються брокери повідомлень. Вони забезпечують передачу даних між пристроями та серверами. Поширеними рішеннями є Eclipse Foundation (Eclipse Mosquitto), Apache Software Foundation (Apache Kafka).

Eclipse Mosquitto реалізує протокол MQTT. Він підходить для пристроїв з обмеженими ресурсами. Apache Kafka орієнтований на обробку великих потоків даних у реальному часі.

Таблиця 1.12 – Порівняння брокерів повідомлень

Рішення	Призначення	Переваги	Недоліки
Eclipse Mosquitto	Передача повідомлень	Легкість, невелике навантаження	Менша продуктивність

Рішення	Призначення	Переваги	Недоліки
	MQTT		при великих обсягах
Apache Kafka	Потокова обробка даних	Висока швидкість, масштабованість	Складність конфігурації

Вибір залежить від вимог системи. Для невеликої IoT-мережі достатньо MQTT-брокера. Для великих промислових систем краще використовувати потокову платформу.

Для реалізації багатоагентної архітектури використовуються спеціальні програмні середовища – Фреймворки. Вони забезпечують створення агентів, організацію їх взаємодії та підтримку протоколів координації.

Таблиця 1.13 – Порівняння платформ для багатоагентних систем

Платформа	Мова	Переваги	Недоліки
JADE	Java	Підтримка стандартів, готові механізми взаємодії	Потребує знання Java
SPADE	Python	Простота реалізації, інтеграція з веб-сервісами	Менша кількість інструментів
Самостійна реалізація	Будь-яка	Гнучкість	Більші витрати часу на розробку

Одним із відомих рішень є JADE (Java Agent Development Framework). Він дозволяє створювати агентів на мові Java та підтримує стандарти FIPA. Також використовуються Python-фреймворки, які реалізують прості агентні моделі для наукових і прикладних задач.

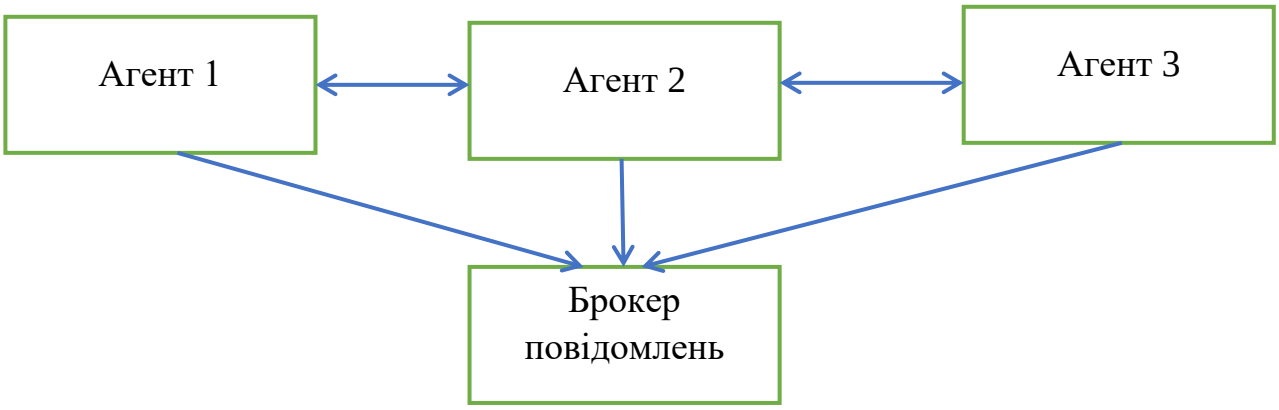


Рисунок 1.5 – Логіка роботи багатоагентної системи

У багатоагентних системах важливо забезпечити стандартизовану взаємодію. Часто використовується модель «публікація–підписка» або протоколи переговорів.

Для створення IoT-рішень використовуються такі технології, як мови програмування (Python, Java, C++); бази даних (PostgreSQL, MongoDB); контейнеризація (Docker); REST API для інтеграції сервісів. Ці інструменти дозволяють об’єднати пристрої, серверні компоненти та агентів в єдину систему. Потрібно забезпечити роботу пристроїв, серверної частини, зберігання даних та взаємодію між модулями. Кожна технологія виконує свою роль у цій структурі.

Мови програмування Python, Java та C++ застосовуються для реалізації різних частин системи. C++ часто використовується на рівні пристроїв, тому що він дозволяє ефективно працювати з апаратними ресурсами та має високу продуктивність. Python зручний для швидкої розробки серверної логіки, обробки даних та створення прототипів агентів. Java часто застосовується у корпоративних системах і багатоагентних платформах, оскільки забезпечує переносимість та стабільність роботи.

Таблиця 1.14 – Особливості мов програмування для IoT

Мова	Де використовується	Переваги	Обмеження
C++	Пристрої, контролери	Висока швидкодія, доступ до апаратури	Складність розробки
Python	Сервер, аналітика, агенти	Простота, швидка розробка	Нижча продуктивність
Java	Сервер, агентні системи	Платформонезалежність	Більше споживання ресурсів

Для збереження інформації використовуються бази даних. PostgreSQL є реляційною базою даних. Вона добре підходить для структурованих даних, наприклад інформації про пристрої, користувачів та події. MongoDB є нереляційною базою. Вона зручна для зберігання великих обсягів напівструктурованих даних, які часто генеруються IoT-пристроями.

Таблиця 1.15 – Порівняння баз даних

База даних	Тип	Переваги	Доцільність використання
PostgreSQL	Реляційна	Надійність, підтримка складних запитів	Облік пристроїв, логів, користувачів
MongoDB	NoSQL	Гнучка структура, масштабованість	Потокові дані, телеметрія

Контейнеризація за допомогою Docker дозволяє запускати всі компоненти системи у відокремлених середовищах. Кожен сервіс працює у власному контейнері. Це забезпечує однакову роботу системи на різних комп’ютерах і серверах. Контейнери спрощують розгортання та оновлення. Якщо потрібно змінити один модуль, це не впливає на інші частини системи.

REST API використовується для інтеграції сервісів. Це набір правил, за якими компоненти системи обмінюються даними через HTTP-запити. Завдяки REST API пристрої можуть передавати інформацію на сервер, а сервер —

надсилати команди або надавати доступ до даних іншим системам. Такий підхід забезпечує стандартизовану та зрозумілу взаємодію.

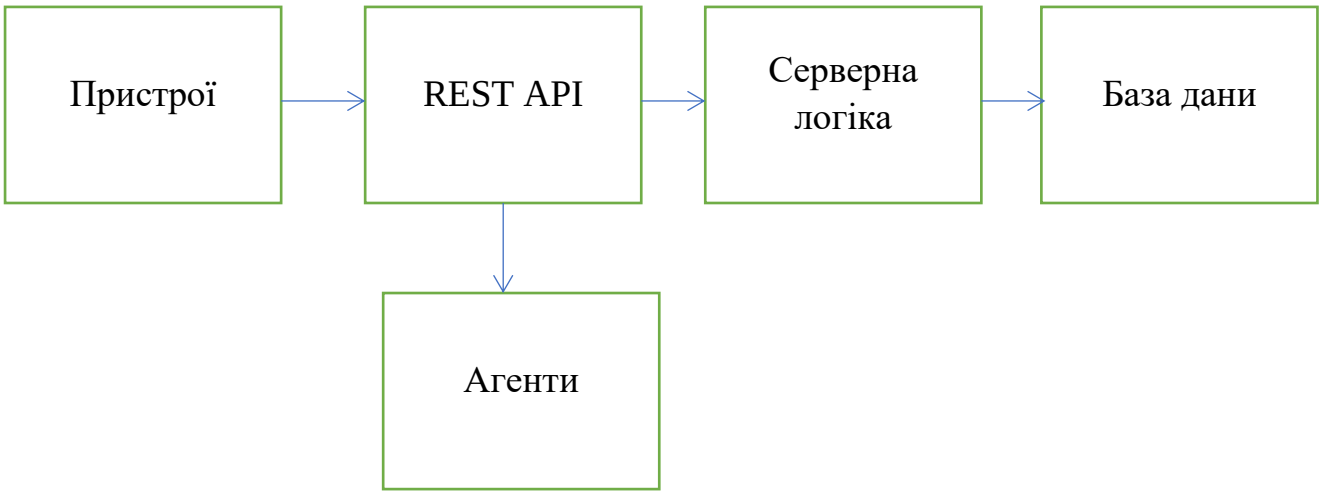


Рисунок 1.6 – Інтеграція компонентів IoT-системи

У схемі на рисунку 1.6 пристрої передають дані через API. Серверна частина обробляє інформацію та зберігає її у базі даних. Багатоагентні модулі аналізують події та приймають рішення. Усі компоненти можуть працювати в контейнерах.

Як ми бачимо, поєднання мов програмування, баз даних, контейнеризації та REST API створює цілісну технологічну основу IoT-системи. Мови програмування забезпечують реалізацію логіки, бази даних відповідають за збереження інформації, Docker гарантує стабільність розгортання, а REST API забезпечує взаємодію між компонентами. Разом ці інструменти дозволяють об’єднати пристрої, серверні модулі та агентів у єдину узгоджену систему.

Таблиця 1.16 – Переваги та обмеження сучасних технологій

Технологія	Перевага	Обмеження
Хмарні сервіси	Готова інфраструктура	Залежність від провайдера

Технологія	Перевага	Обмеження
MQTT	Легка передача даних	Обмежені можливості аналітики
Kafka	Потокова обробка	Потребує ресурсів
JADE	Стандартизована агентна модель	Складність інтеграції з IoT

Аналіз сучасних платформ показує, що універсального рішення не існує. Хмарні платформи забезпечують швидкий старт, але потребують фінансових витрат. Брокери повідомлень дозволяють організувати надійну передачу даних. Агентні фреймворки спрощують реалізацію координації в розподіленому середовищі.

Для реалізації багатоагентної IoT-системи доцільно поєднати кілька технологій. Наприклад, використовувати MQTT для передачі даних, серверну платформу для збереження інформації та агентний фреймворк для організації логіки прийняття рішень.

Аналіз сучасних програмних рішень дозволяє обґрунтовано обрати технологічну основу для подальшого проєктування багатоагентної архітектури IoT-системи.

Висновок до розділу 1.

У розділі 1 проведено детальний аналіз теоретичних основ та існуючих рішень у сфері Інтернету речей та багатоагентних систем. Було визначено особливості побудови IoT-систем, розглянуто їхню структуру та архітектурні рівні.

Аналіз багатоагентних систем дозволив виділити ключові типи агентів, моделі їхньої взаємодії та координації, що забезпечує ефективне функціонування розподілених систем. Це підкреслює важливість правильного вибору архітектурного підходу для досягнення гнучкості, надійності та масштабованості систем.

Огляд сучасних платформ та технологій для реалізації IoT та багатоагентних систем показав, що існують різні інструменти та середовища, які дозволяють створювати інтегровані рішення з високим рівнем автоматизації та адаптивності. Порівняння їхніх можливостей дає основу для обґрунтування вибору оптимальної платформи при розробці конкретного прикладного рішення.

Розділ 1 забезпечує теоретичну та практичну базу для подальшої розробки ефективної системи, що поєднує технології IoT та багатоагентних підходів, і формує основу для розділу 2, присвяченого проєктуванню та реалізації таких систем.

РОЗДІЛ 2 ПРОЄКТУВАННЯ БАГАТОАГЕНТНОЇ IoT-СИСТЕМИ

2.1 Формування вимог до системи надання спеціалізованих послуг

Проєктування багатоагентної IoT-системи починається з чіткого визначення вимог до її функціонування. Вимоги визначають, які завдання система повинна виконувати, які умови дотримувати та якими характеристиками володіти. Вони поділяються на функціональні та нефункціональні.

Функціональні вимоги описують конкретні дії та функції системи. Для багатоагентної IoT-системи можна виділити такі п’ять основних функцій.

1) *Збір даних з IoT-пристроїв*. Система повинна забезпечувати постійний збір інформації від усіх підключених сенсорів та пристроїв. Дані можуть включати температуру, вологість, рівень освітлення, стан обладнання тощо.

2) *Обробка та аналіз даних***.** Отримані дані повинні оброблятися агентами системи для прийняття рішень. Наприклад, у разі перевищення критичних значень сенсорів система повинна формувати сигнал тривоги.

3) *Надання спеціалізованих послуг користувачам***.** Користувачі системи отримують доступ до актуальної інформації через веб-інтерфейс або мобільний додаток. Це дозволяє контролювати стан системи та приймати необхідні рішення.

4) *Координація агентів***.** Агенти повинні обмінюватися інформацією та координувати свої дії для ефективного виконання завдань. Це забезпечує розподілену обробку даних та зменшує навантаження на центральний сервер.

5) *Сповіщення та повідомлення про події*. Система повинна інформувати користувачів про критичні ситуації, наприклад, про вихід параметрів за допустимі межі, несправність пристроїв або завершення певних процесів.

Утаблицях 2.1 – 2.2 сформовано перелік функціональних та нефункціональних вимог до багатоагентної IoT-системи

Таблиця 2.1 – Функціональні вимоги до багатоагентної IoT-системи

№	Вимога	Опис
F1	Збір даних	Постійний збір показників з усіх підключених сенсорів та пристроїв.
F2	Обробка даних	Аналіз даних агентами для прийняття рішень у реальному часі.
F3	Надання спеціалізованих послуг	Відображення результатів через веб- або мобільний інтерфейс.
F4	Координація агентів	Агенти обмінюються інформацією та узгоджують дії.
F5	Сповіщення	Інформування користувачів про критичні події та зміни стану системи.

Кожна функція системи повинна бути ретельно протестована на відповідність вимогам та надійність роботи у різних умовах.

Нефункціональні вимоги визначають якість системи та обмеження її роботи. Для багатоагентної IoT-системи важливими є нефункціональні вимоги щодо надійності, масштабованості, безпеки, продуктивності, зручності користування.

1) Надійність. Система повинна працювати без збоїв, навіть при тимчасових втручаннях або відмові окремих агентів.

2) Масштабованість. Система повинна підтримувати додавання нових пристроїв і агентів без зупинки основних процесів.

3) Безпека. Передача та зберігання даних повинні бути захищені від несанкціонованого доступу, з використанням сучасних методів шифрування та аутентифікації.

4) Продуктивність**.** Час обробки даних та реакції агентів повинен бути мінімальним, щоб забезпечувати своєчасне реагування на події.

5) Зручність користування. Інтерфейс для користувача має бути інтуїтивно зрозумілим, зручною навігацією та можливістю отримувати всю необхідну інформацію швидко.

Таблиця 2.2 – Нефункціональні вимоги до багатоагентної IoT-системи

№	Вимога	Опис
N1	Надійність	Робота 24/7 без критичних збоїв.
N2	Масштабованість	Додавання нових пристроїв та агентів без перерви в роботі.
N3	Безпека	Шифрування даних та контроль доступу користувачів.
N4	Продуктивність	Мінімальна затримка при обробці даних та виконанні дій агентів.
N5	Зручність	Інтерфейс має бути інтуїтивним та доступним для користувачів.

Розглянемо приклад системи моніторингу параметрів приміщення у режимі реального часу. Агенти збирають дані про температуру, вологість та рівень CO_2. Якщо температура перевищує 28°C, агент надсилає сигнал на кондиціонер та повідомлення користувачу. Дані зберігаються у базі для подальшого аналізу та побудови прогнозів. Користувач може переглядати стан приміщення через мобільний додаток або веб-інтерфейс.

Таким чином, чітке формування функціональних та нефункціональних вимог забезпечує основу для проєктування архітектури багатоагентної IoT-системи та подальшої реалізації її функцій.

2.2 Розробка структурної моделі багатоагентної архітектури

Розробка структурної моделі багатоагентної IoT-системи є важливим етапом проєктування. Вона дозволяє визначити загальну архітектуру системи, типи агентів, їхні функції та взаємодію між ними, що забезпечує ефективну роботу системи та правильний розподіл завдань між агентами.

Багатоагентна IoT-система складається з трьох рівнів (сенсорів і пристроїв, агентів, користувача). На рівні сенсорів і пристроїв розташовані всі IoT-пристрої та сенсори, які збирають дані з фізичного середовища.

На рівні агентів дані від сенсорів поступають, обробляються, дії між ними координуються та приймаються рішення. Агент – це програмний об’єкт або компонент системи, який самостійно виконує певні завдання у межах системи. Основна його риса – автономність, тобто агент може працювати самостійно, приймати рішення і взаємодіяти з іншими агентами без постійного втручання користувача. Простіше кажучи, агент – це «розумний модуль» у системі, який має конкретну роль і виконує конкретні дії.

У IoT-системах агентами зазвичай є програмні модулі, які збирають дані з сенсорів. Наприклад, агент може опитувати датчик температури і передавати його показники для обробки. Ними можуть бути і модулі обробки даних, які аналізують інформацію і приймають рішення. Наприклад, якщо температура перевищує допустимий рівень, агент обробки може вирішити увімкнути охолодження або надіслати повідомлення. Агенти координації стежать, щоб усі агенти працювали узгоджено. Вони «організовують роботу» інших агентів, щоб уникнути дублювання дій та оптимізувати процеси. Агенти взаємодії з користувачем передають результати обробки у веб- або мобільний інтерфейс та приймають команди користувача. Агенти сповіщень надсилають повідомлення у разі подій або критичних ситуацій.

В контексті IoT-систем агенти виконують такі основні функції, як отримання даних від сенсорів, обробка даних, координація дій між собою, прийняття рішень. Агенти «збирають інформацію» про стан фізичного

середовища (температура, вологість, освітлення, рух тощо). Після збору даних агенти аналізують їх, роблять висновки і вирішують, які дії потрібно виконати. Агенти обмінюються інформацією та узгоджують свої дії, щоб система працювала ефективно і без конфліктів. На основі аналізу даних і заданих правил агенти можуть самостійно приймати рішення та виконувати дії (наприклад, увімкнути пристрій, надіслати повідомлення, скорегувати параметри системи).

Рівень користувача передбачає, що користувачі взаємодіють з системою через інтерфейси (веб або мобільний додаток), отримують інформацію та управляють процесами.

У системі виділяють кілька типів агентів, залежно від їхніх функцій:

Агент збору даних (Data Collector Agent). Отримує інформацію від сенсорів і передає її агентам обробки.

Агент обробки даних (Data Processing Agent). Аналізує отримані дані, робить висновки та приймає рішення.

Агент координації (Coordinator Agent). Керує взаємодією між агентами, забезпечує ефективний розподіл завдань.

Агент інтерфейсу користувача (User Interface Agent). Забезпечує передачу інформації користувачу та прийом команд.

Агент сповіщень (Notification Agent). Відправляє повідомлення користувачам про важливі події або зміни стану системи.

Таблиця 2.3 – Типи агентів та їх функції

№	Тип агента	Функції
A1	Агент збору даних	Збір даних від сенсорів, передача інформації агентам обробки.
A2	Агент обробки даних	Аналіз даних, формування висновків, прийняття рішень.
A3	Агент координації	Координація дій між агентами, оптимізація виконання завдань.

№	Тип агента	Функції
A4	Агент інтерфейсу користувача	Надання даних користувачу, прийом команд та запитів.
A5	Агент сповіщень	Надсилання повідомлень про аварійні події або критичні зміни.

Кожен агент виконує конкретні функції, що забезпечують загальну роботу системи. Взаємодія між агентами відбувається за наступними принципами:

Агент збору даних передає інформацію агенту обробки.

Агент обробки аналізує дані і при потребі повідомляє агента сповіщень.

Агент координації контролює обмін даними між агентами та розподіляє завдання.

Агент інтерфейсу користувача отримує результати обробки і передає їх користувачу.

Таблиця 2.4 – Взаємодія агентів

Джерело	Приймач	Тип взаємодії	Мета
Агент збору даних	Агент обробки даних	Передача даних	Надання актуальної інформації для аналізу
Агент обробки	Агент сповіщень	Надсилання повідомлень	Інформування користувача про події
Агент обробки	Агент інтерфейсу користувача	Відображення результатів	Передача інформації користувачу
Агент координації	Всі агенти	Контроль та координація	Оптимізація взаємодії та виконання завдань

Розроблена структурна модель багатоагентної IoT-системи дозволяє чітко визначити архітектуру, ролі агентів та їх взаємодію. Така модель забезпечує ефективну обробку даних, координацію дій і надання спеціалізованих послуг користувачу. Визначення типів агентів і їх функцій створює основу для подальшої реалізації системи та її тестування.

2.3 Моделювання процесів взаємодії агентів

Моделювання процесів взаємодії агентів є важливим етапом проєктування багатоагентної IoT-системи. На цьому етапі визначаються алгоритми обміну повідомленнями, правила координації дій та механізми прийняття рішень. Модель взаємодії дозволяє зрозуміти, як агенти обмінюються інформацією, у якій послідовності виконуються дії та як система реагує на зміни зовнішнього середовища.

У багатоагентній системі кожен агент працює автономно, але при цьому взаємодіє з іншими агентами через повідомлення. Проаналізуємо Алгоритм обміну повідомленнями. Обмін повідомленнями є основою взаємодії агентів. Агенти передають один одному інформацію у вигляді структурованих повідомлень. Процес обміну повідомленнями відбувається у такій послідовності:

Агент формує повідомлення → Повідомлення містить тип, джерело, отримувача та дані → Повідомлення передається через комунікаційний модуль → Агент-отримувач приймає повідомлення → Агент аналізує зміст повідомлення → Агент виконує відповідну дію.

Таблиця 2.5 – Структура повідомлення між агентами

Поле повідомлення	Опис
ID повідомлення	Унікальний номер повідомлення
Відправник	Назва або ідентифікатор агента
Отримувач	Назва або ідентифікатор агента
Тип повідомлення	Запит, відповідь, сповіщення, команда
Дані	Інформація для обробки
Час відправлення	Час формування повідомлення

Таблиця 2.6 – Типи повідомлень у системі

Тип повідомлення	Призначення
Запит	Отримання інформації від іншого агента
Відповідь	Передача результатів обробки
Сповіщення	Інформування про подію або зміну стану
Команда	Ініціація виконання певної дії

Такий підхід забезпечує чіткість та структурованість взаємодії.

Розглянемо Алгоритм координації дій агентів. Координація потрібна для узгодженої роботи системи. Вона запобігає конфліктам та дублюванню завдань. Основними етапами координації є визначення завдання, передача завдання відповідальному агенту, підтвердження отримання завдання, виконання завдання, надсилання результату координатору. У системі може використовуватися централізована або децентралізована координація.

Таблиця 2.7 – Порівняння моделей координації

Характеристика	Централізована модель	Децентралізована модель
Керування	Один координуючий агент	Кожен агент приймає рішення

Характеристика	Централізована модель	Децентралізована модель
Надійність	Залежить від одного агента	Вища через розподіл функцій
Масштабованість	Обмежена	Вища
Складність реалізації	Простіша	Складніша

У даній системі доцільно використовувати змішану модель. Частина рішень приймається локально, а складні процеси координуються центральним агентом.

Опрацюємо Алгоритм прийняття рішень. Прийняття рішень відбувається на основі отриманих даних та заданих правил.

Агент використовує простий алгоритм:

Отримати нові дані → Перевірити дані на відповідність пороговим значенням → Якщо умова виконується – сформувати дію → Якщо умова не виконується – продовжити моніторинг →Передати результат іншим агентам або користувачу.

Таблиця 2.8 – Приклад правил прийняття рішень

Умова	Дія агента
Температура > 28°C	Увімкнути систему охолодження
Вологість < 30%	Увімкнути зволожувач
Відсутній сигнал від сенсора	Надіслати повідомлення про помилку
Рівень заряду < 20%	Сповістити про необхідність зарядки

Для складніших систем можуть застосовуватися алгоритми машинного навчання або адаптивні правила. Проте для бакалаврського рівня достатньо реалізації логіки на основі умов та правил.

Розглянемо узагальнену модель процесу взаємодії. Загальний процес роботи системи можна описати так:

Сенсор передає дані агенту збору → Агент збору надсилає повідомлення агенту обробки → Агент обробки аналізує інформацію → У разі необхідності агент формує команду або сповіщення → Результат передається користувачу.

Таблиця 2.9 – Послідовність взаємодії агентів

Крок	Агент	Дія
1	Агент збору	Отримує дані від сенсора
2	Агент збору	Формує повідомлення
3	Агент обробки	Аналізує дані
4	Агент координації	Узгоджує виконання дії
5	Агент сповіщень	Інформує користувача

Запропонована модель забезпечує структуровану та узгоджену роботу системи. Визначені алгоритми можуть бути реалізовані програмно під час розробки системи.

Проілюструємо на прикладах три типи алгоритмів, а саме, алгоритми обміну повідомленнями, координації дій, прийняття рішень.

Для опрацювання алгоритму обміну повідомленнями розглянемо Алгоритм «Запит–Відповідь» та Алгоритм «Публікація–Підписка».

Алгоритм «Запит–Відповідь» використовується, коли один агент потребує інформацію від іншого агента.

Послідовність роботи:

Агент ініціатор формує запит → У запиті вказується тип даних → Запит надсилається агенту-отримувачу → Агент-отримувач перевіряє коректність запиту → Агент-отримувач формує відповідь → Відповідь повертається ініціатору → Ініціатор аналізує отримані дані.

Таблиця 2.10 – Приклад роботи алгоритму

Крок	Агент	Дія
1	Агент обробки	Надсилає запит про температуру
2	Агент збору	Отримує запит
3	Агент збору	Формує відповідь
4	Агент обробки	Аналізує отримані дані

Алгоритм «Публікація–Підписка» використовується, коли кілька агентів повинні отримувати однакову інформацію.

Послідовність роботи:

Агент-джерело публікує повідомлення у спільному каналі → Інші агенти підписані на цей канал → Після появи нового повідомлення всі підписані агенти його отримують → Кожен агент самостійно приймає рішення щодо подальших дій.

Таблиця 2.11– Ролі у моделі «Публікація–Підписка»

Роль	Опис
Видавець	Формує та публікує дані
Підписник	Отримує повідомлення
Комунікаційний модуль	Забезпечує передачу даних

Для опанування алгоритму координації дій Простежимо Централізований та Децентралізований алгоритми координації.

У Централізованому алгоритмі координації є один координуючий агент.

Послідовність роботи:

Агенти повідомляють координатора про свій стан → Координатор аналізує інформацію → Координатор розподіляє завдання між агентами →

Агенти підтверджують отримання завдання → Після виконання завдання агенти надсилають результат.

Таблиця 2.12 – Приклад координації

Крок	Координатор	Виконавчий агент
1	Отримує дані	Надсилає свій стан
2	Формує завдання	Отримує завдання
3	Контролює виконання	Виконує дію

У цьому випадку Децентралізованого алгоритму координації агенти самі домовляються між собою.

Послідовність роботи:

Агент виявляє проблему → Агент надсилає пропозицію іншим агентам → Інші агенти оцінюють можливість виконання → Один агент бере на себе виконання → Інші агенти отримують повідомлення про вибір.

Таблиця 2.13 – Переваги та недоліки

Критерій	Перевага	Недолік
Надійність	Відсутній єдиний центр	Складність реалізації
Масштабованість	Легко додавати агентів	Можливі конфлікти рішень

Для аналізу алгоритму прийняття рішень простежимо за Правиловим алгоритмом (умова–дія) та Алгоритмом на основі пріоритетів.

Правиловий алгоритм (умова–дія) є найпростішим типом алгоритму.

Послідовність роботи:

Агент отримує нові дані → Агент перевіряє умови →Якщо умова істинна – виконується дія → Якщо умова хибна – система продовжує моніторинг.

Таблиця 2.14 – Приклад правил

Умова	Дія
Температура > 28°C	Увімкнути охолодження
Рівень газу > норма	Надіслати аварійне сповіщення
Сенсор не відповідає	Запустити перевірку системи

Алгоритм на основі пріоритетів використовується у системі, у якій можуть виникати кілька подій одночасно.

Послідовність роботи:

Агент формує список подій → Кожній події присвоюється пріоритет → Події сортуються за рівнем важливості → Виконується подія з найвищим пріоритетом → Після виконання система переходить до наступної події.

Таблиця 2.15– Приклад пріоритетів

Подія	Пріоритет
Пожежна тривога	Високий
Відсутність зв’язку з сенсором	Середній
Планове оновлення даних	Низький

Наведені приклади алгоритмів демонструють різні підходи до організації взаємодії у багатоагентній IoT-системі. Алгоритми обміну повідомленнями забезпечують передачу даних. Алгоритми координації дозволяють узгоджувати роботу агентів. Алгоритми прийняття рішень визначають поведінку системи у різних ситуаціях. Такі алгоритми можуть бути реалізовані у програмному коді під час створення системи.

Висновок до розділу 2.

У розділі 2 бакалаврської роботи виконано проєктування багатоагентної IoT-системи надання спеціалізованих послуг. Основна увага приділялася формуванню вимог до системи, розробці її структурної моделі та моделюванню процесів взаємодії агентів.

Сформовано функціональні та нефункціональні вимоги. Було визначено основні функції системи, зокрема збір даних з IoT-пристроїв, обробку інформації, координацію агентів, надсилання сповіщень та взаємодію з користувачем. Також встановлено вимоги до надійності, безпеки, масштабованості, продуктивності та зручності використання. Чітке формулювання вимог створило основу для подальшого проєктування системи.

Розроблено структурну модель багатоагентної архітектури. Було визначено рівні системи, зокрема рівень сенсорів, рівень агентів та рівень користувача. Описано типи агентів та їх функції. Кожен агент виконує конкретні завдання та взаємодіє з іншими агентами через механізм обміну повідомленнями. Такий підхід забезпечує розподіленість, гнучкість та можливість масштабування системи.

Змодельовано процеси взаємодії агентів. Було описано алгоритми обміну повідомленнями, координації дій та прийняття рішень. Запропоновані алгоритми дозволяють забезпечити узгоджену роботу всіх компонентів системи. Визначені правила прийняття рішень гарантують своєчасне реагування на зміни стану середовища та події.

Сформовано цілісну модель багатоагентної IoT-системи. Визначено її структуру, принципи взаємодії та логіку функціонування. Отримані результати створюють основу для практичної реалізації системи шляхом її програмної розробки, що буде розглянуто у наступному розділі дипломної роботи.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ БАГАТОАГЕНТНОЇ IoT-СИСТЕМИ

3.1 Вибір програмних засобів та середовища розробки

Розробка багатоагентної IoT-системи потребує комплексного підходу до вибору програмних засобів, що забезпечують масштабованість, надійність, безпеку та можливість інтеграції з різнорідними пристроями. З огляду на архітектуру системи, яка передбачає взаємодію агентів, сенсорних вузлів, серверної частини та клієнтських застосунків, було обрано стек технологій, орієнтований на модульність і гнучкість.

Для створення серверної частини та агентної логіки доцільним є використання середовища розробки Visual Studio Code або PyCharm, які забезпечують підтримку сучасних мов програмування, інструментів налагодження, систем контролю версій (Git) та розширень для роботи з IoT-платформами.

Для реалізації мікроконтролерної частини системи на базі ESP32 доцільно використовувати середовище розробки Arduino IDE, яке є зручним та широко застосовується для програмування вбудованих пристроїв. Arduino IDE забезпечує повний цикл розробки програмного забезпечення для мікроконтролера, а саме, написання програмного коду, його компіляцію (перетворення у машинний код, зрозумілий мікроконтролеру) та подальше завантаження прошивки безпосередньо в пам’ять пристрою через USB-інтерфейс, що спрощує процес налагодження та тестування роботи системи.

Важливою перевагою середовища є підтримка великої кількості готових бібліотек, що дозволяють реалізувати роботу з мережевими протоколами та IoT-сервісами без необхідності розробки низькорівневих механізмів взаємодії. Зокрема, для ESP32 доступні бібліотеки для підключення до Wi-Fi, роботи з протоколами MQTT, HTTP, WebSocket, а також для шифрування переданих даних. Використання Arduino IDE забезпечує швидку розробку, зменшення

складності програмної реалізації та підвищення надійності роботи мікроконтролерної частини багатоагентної IoT-системи.

Для реалізації програмної логіки багатоагентної IoT-системи було обрано декілька мов програмування, що відповідають різним рівням архітектури системи та забезпечують її ефективність, масштабованість і надійність. Мова програмування Python обрана для реалізації серверної логіки, координації агентів, обробки телеметричних даних та взаємодії з базами даних. Основними перевагами Python є висока швидкість розробки завдяки простому синтаксису; підтримка асинхронного програмування (через asyncio), що є важливим при роботі з великою кількістю IoT-пристроїв; наявність спеціалізованих бібліотек для IoT, зокрема paho-mqtt (для роботи з MQTT-брокером) та FastAPI (для створення REST API); зручна інтеграція з реляційними та нереляційними базами даних.

Завдяки цьому Python є оптимальним вибором для реалізації багатоагентної логіки, централізованої обробки подій та аналітики даних.

Для програмування мікроконтролерів ESP32 застосовується C/C++, оскільки ці мови забезпечують високий рівень продуктивності; дозволяють ефективно керувати пам’яттю та ресурсами пристрою; надають прямий доступ до апаратних регістрів і периферії; підтримуються більшістю середовищ розробки для вбудованих систем. З огляду на обмежені обчислювальні ресурси IoT-пристроїв, використання C/C++ є технічно обґрунтованим рішенням для реалізації низькорівневої взаємодії з датчиками та мережевими модулями.

JavaScript у середовищі Node.js використовується для створення серверних API, обробки подій у режимі реального часу або реалізації додаткових сервісів (наприклад, MQTT-брокера чи шлюзу повідомлень). Основними перевагами є неблокуюча (асинхронна) модель виконання; ефективна робота з потоками подій; зручність створення веб-сервісів; легка інтеграція з фронтендом. Node.js доцільний у випадках, коли необхідно забезпечити швидку передачу даних між сервером і клієнтським застосунком.

HTML та CSS застосовуються для створення користувацького веб-інтерфейсу, який забезпечує відображення даних із сенсорів; моніторинг стану системи в реальному часі; керування пристроями; формування звітів та візуалізацію інформації.

HTML відповідає за структуру веб-сторінки, а CSS – за її стилізацію та адаптивність. Такий підхід дозволяє створити кросплатформений інтерфейс, доступний через звичайний веб-браузер без встановлення додаткового програмного забезпечення.

Поєднання Python, C/C++, JavaScript та HTML/CSS забезпечує розподіл функцій між рівнями системи відповідно до їх технічних вимог. Такий підхід дозволяє оптимально використовувати апаратні ресурси; забезпечити високу швидкодію; реалізувати масштабовану багатоагентну архітектуру; спростити подальшу модернізацію системи. Обраний набір технологій є сучасним, широко підтримуваним спільнотою розробників та відповідає вимогам до побудови ефективних IoT-рішень.

Для організації обміну даними в багатоагентній IoT-системі обрано протокол MQTT (Message Queuing Telemetry Transport), оскільки він найбільш відповідає вимогам до роботи з великою кількістю пристроїв із обмеженими ресурсами. MQTT є легковаговим мережевим протоколом, що працює за моделлю publish/subscribe (публікація/підписка). У межах цієї моделі IoT-пристрої (датчики, контролери) публікують повідомлення в певні тематичні канали (topics), а серверні агенти або інші компоненти системи підписуються на них і отримують дані в режимі реального часу.

Перевагами MQTT для IoT-систем є мінімальне навантаження на мережу, адже невеликий розмір службових заголовків дозволяє передавати дані навіть через нестабільні або малошвидкісні канали зв’язку. Енергоефективність, яка є особливо важливою для пристроїв, що працюють від батарей. До переваг можна віднести асинхронну передачу даних, оскільки агенти отримують повідомлення одразу після їх публікації без постійних запитів до сервера. Підтримка рівнів якості обслуговування (QoS) гарантує

доставку повідомлень відповідно до обраного рівня надійності. Плюсом є масштабованість, бо через брокер повідомлень можна організувати взаємодію великої кількості пристроїв і програмних агентів. З огляду на те, що багатоагентна система передбачає постійний обмін телеметричними даними (температура, тиск, стан обладнання тощо), MQTT є технічно доцільним рішенням.

Альтернативно для взаємодії з веб-клієнтами та зовнішніми інформаційними системами може застосовуватися протокол HTTP із використанням REST-архітектури. HTTP/REST доцільний у випадках, коли необхідно реалізувати доступ до даних через веб-браузер; створити API для інтеграції із сторонніми сервісами; передавати дані за запитом користувача (on-demand); забезпечити просту взаємодію з фронтендом. На відміну від MQTT, HTTP працює за моделлю «запит–відповідь», що є менш ефективним для постійного потокового обміну телеметрією, але зручним для побудови веб-сервісів та адміністрування системи.

Поєднання MQTT і HTTP/REST дозволяє реалізувати гібридну архітектуру, адже MQTT слугує для швидкого та енергоефективного обміну даними між IoT-пристроями й агентами; HTTP/REST використовується для взаємодії з користувацькими інтерфейсами та зовнішніми сервісами. Такий підхід забезпечує оптимальний баланс між продуктивністю, масштабованістю та зручністю інтеграції, що відповідає вимогам до сучасних багатоагентних IoT-систем.

У якості бази даних для збереження даних сенсорів і журналів подій обрано MongoDB для зберігання неструктурованих або напівструктурованих даних; або PostgreSQL для структурованих даних і складних аналітичних запитів.

Для забезпечення стабільної роботи багатоагентної IoT-системи, її масштабованості та зручності адміністрування доцільно використовувати технології контейнеризації, зокрема Docker, що дозволяє розміщувати кожен компонент системи в окремому ізольованому контейнері. У межах

розроблюваної системи це можуть бути MQTT-брокер; сервер додатків (агентна логіка); база даних; веб-сервер або API.

Основними перевагами використання Docker є ізоляція сервісів, портативність, спрощення розгортання, масштабованість, підвищення надійності. Ізоляція сервісів передбачає, що кожен компонент працює у власному середовищі з визначеними залежностями, що усуває конфлікти між бібліотеками та версіями програмного забезпечення. Портативність означає, що система може бути розгорнута на будь-якому сервері (локальному або хмарному), де встановлено Docker, без додаткового налаштування середовища. Спрощення розгортання полягає у тому, що запуск системи зводиться до виконання однієї конфігурації (docker-compose), значно скорочуючи час інсталяції. Масштабованість сприяє тому, що окремі сервіси можна запускати у декількох екземплярах залежно від навантаження. Важливою перевагою є підвищення надійності, адже у разі збою одного контейнера інші продовжують працювати незалежно. Для багатоагентної IoT-системи, яка обробляє потоки даних у реальному часі, така ізоляція та гнучкість є особливо важливими.

У випадку розгортання системи в умовах високого навантаження або в хмарному середовищі доцільно застосовувати Kubernetes – систему оркестрації контейнерів. Kubernetes забезпечує автоматичне масштабування сервісів; балансування навантаження; автоматичне перезапускання контейнерів у разі збоїв; централізоване керування конфігурацією; оновлення сервісів без зупинки роботи системи (rolling updates). Використання Docker дозволяє стандартизувати процес розгортання та тестування багатоагентної IoT-системи, забезпечити її мобільність і гнучкість. Застосування Kubernetes, у разі потреби, підвищує рівень автоматизації управління та гарантує стабільність роботи системи в умовах масштабування.

Вибір програмних засобів та середовища розробки відповідає вимогам до побудови сучасної багатоагентної IoT-системи, орієнтованої на надійну

роботу в реальному часі та подальше розширення функціональних можливостей.

3.2 Реалізація програмних агентів та механізмів їх взаємодії

Програмна реалізація агентів багатоагентної IoT-системи виконана з урахуванням принципів багатоагентних систем та сучасних IoT-технологій [1; 2].

У розробленій системі кожен агент реалізований як окремий програмний модуль. Агент працює як незалежний процес або сервіс. Він має власну логіку обробки даних та прийняття рішень. Архітектура агента складається з таких компонентів, як модуль збору даних, модуль обробки даних, модуль прийняття рішень, модуль обміну повідомленнями, база локального стану агента.

Модуль збору даних отримує інформацію від IoT-пристроїв через мережеві протоколи. У реалізації використано протокол MQTT, який є легковаговим та підходить для IoT-середовищ [3].

Модуль обробки даних аналізує отримані показники. Він виконує фільтрацію, нормалізацію та перевірку даних. Обробка реалізована засобами мови програмування Python із використанням бібліотек для роботи з асинхронними подіями.

Модуль прийняття рішень працює за правиловим принципом. Він використовує набір умов типу «якщо–то». Наприклад, якщо значення датчика перевищує встановлений поріг, агент формує повідомлення тривоги. Такий підхід відповідає реактивній моделі агентів [1].

Модуль обміну повідомленнями забезпечує взаємодію з іншими агентами через брокер повідомлень. Для цього використано програмну платформу Mosquitto як MQTT-брокер.

У системі реалізовано такі типи агентів, як агент збору даних, агент обробки та аналізу, координаційний агент, користувацький агент (інтерфейсний). Усі агенти реалізовано мовою Python як окремі мікросервіси. Взаємодія між ними здійснюється через MQTT-брокер за моделлю publish/subscribe, що відповідає сучасним підходам до побудови багатоагентних систем [1; 2] та IoT-архітектур [3].

Рисунок 3.1 – Інтерфейс агента збору даних

Агент збору даних відповідає за отримання інформації від IoT-пристроїв (датчиків температури, вологості, руху тощо) та передачу її до інших агентів. Він публікує дані у відповідні MQTT-топіки.

Агент збору даних виконує такі дії:

1) Підключається до MQTT-брокера.
2) Зчитує дані з сенсора або емулятора.
3) Формує повідомлення у форматі JSON.

4) Публікує повідомлення у відповідний топік (наприклад, iot/sensors/temperature).
5) Повторює цикл через заданий інтервал часу.

Агент збору даних працює у безперервному циклі та генерує події для інших агентів.

Програмна реалізація на мові Python:

```
import json
import time
import random
import paho.mqtt.client as mqtt
from datetime import datetime

BROKER = "localhost"
TOPIC = "iot/sensors/temperature"

client = mqtt.Client()
client.connect(BROKER, 1883, 60)

def read_sensor():
    # Емуляція датчика температури
    return round(random.uniform(20, 35), 2)

while True:
    temperature = read_sensor()

    message = {
        "sender": "data_agent_1",
        "type": "sensor_data",
        "timestamp": datetime.utcnow().isoformat(),
        "data": {
```

```
            "temperature": temperature
        }
    }

    client.publish(TOPIC, json.dumps(message))
    time.sleep(5)
```

Агент обробки та аналізу підписується на ці топіки, аналізує дані та формує рішення.

Рисунок 3.2 – Інтерфейс агенту обробки та аналізу

Агент обробки та аналізу виконує такі дії:

1) Підписується на топік iot/sensors/temperature.
2) Отримує повідомлення.
3) Аналізує значення температури.
4) Якщо значення перевищує поріг (наприклад, 30°C), формує повідомлення типу alert.
5) Публікує повідомлення у топік iot/alerts.

Таким чином реалізовано реактивну модель агента [1].

Програмна реалізація на мові Python:

```
import json
import paho.mqtt.client as mqtt

BROKER = "localhost"
SENSOR_TOPIC = "iot/sensors/temperature"
ALERT_TOPIC = "iot/alerts"

THRESHOLD = 30

client = mqtt.Client()

def on_message(client, userdata, msg):
    payload = json.loads(msg.payload.decode())
    temperature = payload["data"]["temperature"]

    if temperature > THRESHOLD:
        alert_message = {
            "sender": "analysis_agent_1",
            "type": "alert",
            "data": {
                "temperature": temperature,
                "status": "critical"
            }
        }
        client.publish(ALERT_TOPIC, json.dumps(alert_message))

client.on_message = on_message
client.connect(BROKER, 1883, 60)
```

```
client.subscribe(SENSOR_TOPIC)
client.loop_forever()
```

Координаційний агент узгоджує дії інших агентів. Він отримує інформацію про події, стан інших агентів, визначає подальші дії інших агентів, контролює стан системи та розподіляє завдання. На основі цієї інформації він формує керуючі команди. Якщо один агент виявляє критичну ситуацію, координаційний агент може активувати додаткові сервіси; повідомити користувача; змінити режим роботи інших агентів. Для уникнення конфліктів використано механізм пріоритетів. Кожне завдання має рівень важливості. Агент обробляє спочатку найбільш критичні події. Такий підхід забезпечує масштабованість системи; можливість додавання нових агентів; стійкість до відмов окремих компонентів; гнучку координацію дій та відповідає ієрархічній моделі координації [2].

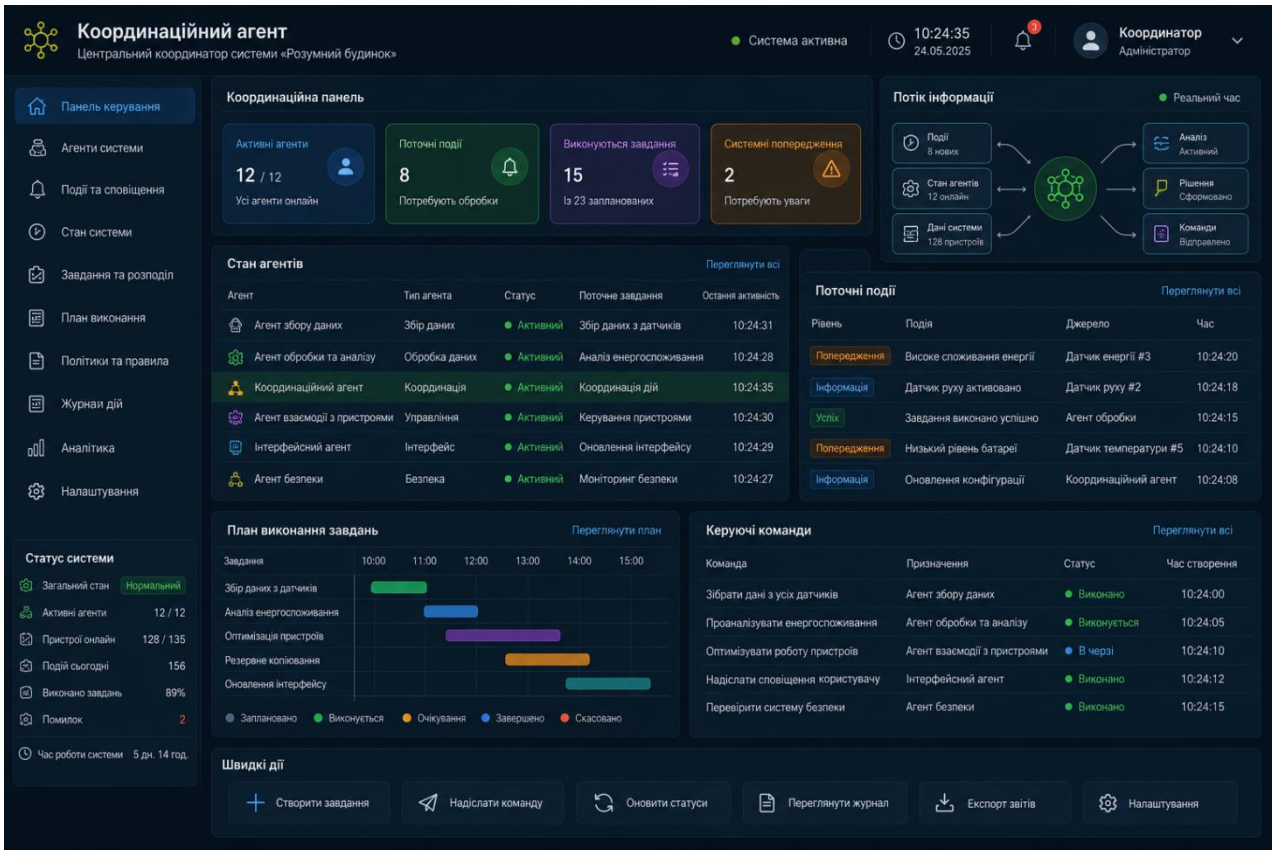


Рисунок 3.3 – Інтерфейс Координаційного агенту

Координаційний агент виконує такі дії:

1) Підписується на топік iot/alerts.
2) Аналізує тип події.
3) Приймає рішення щодо реакції системи.
4) Надсилає керуючі команди у топік iot/commands.

Наприклад, у разі критичної температури агент може надіслати команду на увімкнення вентиляції.

Програмна реалізація на мові Python:

```
import json
import paho.mqtt.client as mqtt

BROKER = "localhost"
ALERT_TOPIC = "iot/alerts"
COMMAND_TOPIC = "iot/commands"

client = mqtt.Client()

def on_message(client, userdata, msg):
    payload = json.loads(msg.payload.decode())

    if payload["type"] == "alert":
        command_message = {
            "sender": "coordination_agent",
            "type": "command",
            "data": {
                "device": "ventilation_system",
                "action": "ON"
            }
        }
        client.publish(COMMAND_TOPIC, json.dumps(command_message))
```

```
client.on_message = on_message
client.connect(BROKER, 1883, 60)
client.subscribe(ALERT_TOPIC)
client.loop_forever()
```

Користувацький (інтерфейсний) агент забезпечує взаємодію системи з веб-інтерфейсом. Він отримує запити користувача та передає їх до інших агентів.

Користувацький агент виконує такі дії:

1) Отримує HTTP-запит від користувача.
2) Перетворює його у MQTT-повідомлення.
3) Надсилає запит до відповідного топіка.
4) Отримує відповідь від інших агентів.
5) Повертає результат користувачу.

Для реалізації використано фреймворк Flask.

Рисунок 3.4 – Інтерфейс Користувацького агенту

Програмна реалізація на мові Python:

```
from flask import Flask, request, jsonify
import json
import paho.mqtt.publish as publish

app = Flask(__name__)
BROKER = "localhost"
COMMAND_TOPIC = "iot/commands"

@app.route("/device", methods=["POST"])
def control_device():
    data = request.json

    message = {
        "sender": "user_agent",
        "type": "command",
        "data": data
    }

    publish.single(COMMAND_TOPIC, json.dumps(message),
hostname=BROKER)

    return jsonify({"status": "Command sent"}), 200

if __name__ == "__main__":
    app.run(port=5000)
```

Система реалізована у вигляді мікросервісної архітектури. Кожен агент є окремим сервісом у контейнері Docker. Взаємодія між сервісами здійснюється через мережу. У розробленій системі агенти функціонують як автономні

програмні модулі. Вони взаємодіють через брокер повідомлень і не залежать безпосередньо один від одного. Кожен агент постійно очікує на подію. Подією може бути надходження даних від сенсора, отримання повідомлення від іншого агента, запит від користувача.

Після отримання події агент виконує такі чотири кроки:

1 крок Аналізує вхідні дані.

2 крок Оновлює свій внутрішній стан.

3 крок Приймає рішення.

4 крок Формує та надсилає повідомлення.

Для реалізації подієвої моделі використано асинхронну бібліотеку asyncio мови Python, що дозволяє агентам працювати паралельно без блокування виконання.

Обмін повідомленнями реалізовано за моделлю publish/subscribe. Кожен агент може бути видавцем або підписником.

Повідомлення мають формат JSON. Вони містять ідентифікатор агента-відправника, тип повідомлення, часову мітку; корисні дані.

Приклад структури повідомлення:

```
{
"sender": "sensor_agent_1",
"type": "alert",
"timestamp": "2026-03-03T10:15:00",
"data": {
"temperature": 85
}
}
```

Використання стандартного формату спрощує інтеграцію та масштабування системи [3].

Основними програмними засобами є мова програмування Python; MQTT-брокер Mosquitto; Docker для контейнеризації; REST API для взаємодії з користувацьким інтерфейсом. Запуск системи здійснюється за допомогою

docker-compose, що дозволяє швидко розгортати систему у тестовому або виробничому середовищі.

Програмна реалізація багатоагентної IoT-системи забезпечує модульність, масштабованість та надійність роботи. Використання стандартних протоколів та асинхронної обробки подій підвищує ефективність взаємодії агентів.

3.3 Розробка серверної частини та інтеграція компонентів IoT-системи

Серверна частина забезпечує прийом даних від агентів, їх обробку, зберігання, координацію роботи системи та взаємодію з користувачем. Реалізація виконана відповідно до принципів побудови розподілених систем та IoT-архітектур [1; 2].

Серверна частина розробленої багатоагентної IoT-системи реалізована за мікросервісною архітектурою. Кожен компонент (MQTT-брокер, сервер обробки даних, база даних, REST API) реалізований як окремий сервіс, що дозволяє змінювати або оновлювати один компонент без впливу на інші. Мікросервісний підхід в серверній частині IoT-системи обумовлений модульністю системи, масштабованістю, надійністю, гнучкістю розробки.

У разі збільшення кількості IoT-пристроїв можна масштабувати лише той сервіс, який обробляє навантаження (наприклад, сервіс аналізу даних), що відповідає вимогам масштабованості розподілених систем [4].

Надійність системи обумовлена тим, що якщо один сервіс виходить з ладу, інші продовжують працювати. Наприклад, збій REST API не зупиняє обмін повідомленнями між агентами через MQTT. Кожен сервіс можна розробляти та тестувати окремо, що спрощує налагодження системи. Основними характеристиками такого підходу є асинхронна передача даних; мінімальне навантаження на мережу; підтримка моделі publish/subscribe; слабка зв’язаність компонентів. Сервіси не звертаються один до одного

напряму. Вони обмінюються повідомленнями через брокер. Наприклад, Агент збору даних публікує повідомлення в топік iot/sensors. Сервер обробки підписаний на цей топік. Після отримання даних сервер зберігає їх у базі. Така схема зменшує залежність між сервісами. Відправник не знає, хто саме отримає повідомлення, що відповідає принципу слабкої зв'язаності компонентів [4].

Взаємодія між сервісами здійснюється через MQTT-брокер (асинхронна взаємодія) та REST API (синхронна взаємодія). Поєднання MQTT та REST API дозволяє розділити машинну та користувацьку взаємодію; забезпечити швидку передачу телеметрії; організувати зручний доступ до системи; мінімізувати затримки в обробці подій; підвищити загальну продуктивність. У результаті серверна частина стає центральним координуючим вузлом багатоагентної IoT-системи. Вона інтегрує всі компоненти в єдину програмну платформу.

До складу серверної частини входять MQTT-брокер; сервер прикладної логіки; база даних; REST API; модуль авторизації користувачів.

MQTT-брокер використовується для внутрішнього обміну повідомленнями між агентами та серверними сервісами. Він реалізує модель publish/subscribe, яка є стандартом для IoT-систем [3].

Сервер прикладної логіки обробляє отримані дані та координує роботу агентів. База даних зберігає історію вимірювань та подій. REST API використовується для взаємодії через синхронний механізм обміну даними з користувачами або зовнішніми системами, забезпечує доступ до системи через веб-інтерфейс або зовнішні сервіси.

Користувач надсилає HTTP-запит до серверу. Сервер обробляє запит і повертає відповідь. Наприклад, GET /data – отримати останні вимірювання; POST /device – надіслати команду пристрою.

Кожен сервіс запускається у власному Docker-контейнері. Для реалізації брокера використано Eclipse Mosquitto. Він підтримує стандарт MQTT версії 3.1.1 [3]. Сервіси взаємодіють через внутрішню мережу Docker. Такий підхід

спрощує розгортання, тестування та відповідає принципам побудови розподілених систем та сучасних IoT-платформ [1; 4].

Приклад конфігурації docker-compose:

```
version: '3'

services:
  mosquitto:
    image: eclipse-mosquitto
    ports:
      - "1883:1883"
    volumes:
      - ./mosquitto.conf:/mosquitto/config/mosquitto.conf
```

Сервер прикладної логіки реалізовано мовою Python із використанням фреймворку Flask. Він виконує такі функції, як отримання повідомлення від MQTT-брокера, обробка дані, збереження їх у базу даних, надання REST API для клієнтів.

Підключення до MQTT на мові Python

```
import json
import sqlite3
import paho.mqtt.client as mqtt

BROKER = "localhost"
TOPIC = "iot/sensors/#"

conn = sqlite3.connect("iot.db", check_same_thread=False)
cursor = conn.cursor()

cursor.execute("""
```

```
    CREATE TABLE IF NOT EXISTS sensor_data (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        temperature REAL,
        timestamp TEXT
    )
    """)
    conn.commit()

    def on_message(client, userdata, msg):
        payload = json.loads(msg.payload.decode())

        if payload["type"] == "sensor_data":
            temperature = payload["data"]["temperature"]
            timestamp = payload["timestamp"]

            cursor.execute(
                "INSERT INTO sensor_data (temperature, timestamp) VALUES (?, ?)",
                (temperature, timestamp)
            )
            conn.commit()

    client = mqtt.Client()
    client.on_message = on_message
    client.connect(BROKER, 1883, 60)
    client.subscribe(TOPIC)
    client.loop_start()
```

REST API забезпечує доступ до збережених даних та дозволяє керувати пристроями. Приклад реалізації REST API на мові Python

```
from flask import Flask, jsonify
import sqlite3

app = Flask(__name__)

@app.route("/data", methods=["GET"])
def get_data():
    conn = sqlite3.connect("iot.db")
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM sensor_data ORDER BY id DESC LIMIT 10")
    rows = cursor.fetchall()

    return jsonify(rows)

if __name__ == "__main__":
    app.run(port=5000)
```

Через цей API користувач може отримати останні показники сенсорів.

Інтеграція компонентів здійснюється через єдину мережеву інфраструктуру Docker. Усі сервіси запускаються через docker-compose.

Агент збору даних публікує повідомлення у MQTT. Сервер прикладної логіки підписується на відповідні топіки. Дані зберігаються у базу. Користувач отримує доступ до даних через REST API. У разі потреби сервер надсилає команди іншим агентам. Такий підхід відповідає багаторівневій архітектурі IoT-систем [2].

Для підвищення безпеки реалізовано автентифікацію клієнтів MQTT; перевірку вхідних даних; розмежування доступу до API; логування подій. Передача даних може здійснюватися через TLS, що відповідає рекомендаціям щодо безпеки IoT-систем [1].

Висновок до розділу 3.

У розділі 3 бакалаврської роботи було розглянуто програмну реалізацію багатоагентної IoT-системи для надання спеціалізованих послуг.

Обґрунтовано вибір програмних засобів та середовища розробки. Для створення системи було обрано мову програмування Python через її простоту, наявність бібліотек для роботи з мережевими протоколами та підтримку асинхронного програмування. Для організації обміну повідомленнями використано протокол MQTT, який є ефективним для IoT-середовищ. Контейнеризація за допомогою Docker забезпечила зручність розгортання та масштабування системи. Обрані технології відповідають сучасним вимогам до побудови розподілених та багатоагентних систем.

Реалізовано програмних агентів та механізми їх взаємодії. Було створено агент збору даних, агент обробки та аналізу, координаційний агент та користувацький агент. Кожен агент функціонує як окремий програмний модуль із власною логікою роботи. Взаємодія між агентами реалізована за моделлю publish/subscribe через MQTT-брокер. Такий підхід забезпечує слабку зв’язаність компонентів, гнучкість та можливість розширення системи. Реалізація підтвердила працездатність обраної багатоагентної архітектури.

Розроблено серверну частину системи та виконано інтеграцію всіх компонентів. Серверна частина реалізована за мікросервісним підходом. Вона включає сервіс обробки даних, базу даних та REST API для взаємодії з користувачем. Інтеграція здійснюється через MQTT та HTTP-протокол. Така архітектура забезпечує масштабованість, надійність та зручність подальшої модернізації системи.

Створена система демонструє можливість ефективної координації програмних агентів, обробки даних у реальному часі та надання спеціалізованих сервісів користувачам. Отримані результати підтверджують доцільність використання багатоагентного підходу для побудови інтелектуальних IoT-рішень.

РОЗДІЛ 4. ОХОРОНА ПРАЦІ

4.1. Організаційно-правові основи забезпечення безпеки праці

Під час розробки та експлуатації багатоагентної архітектури IoT-системи для надання спеціалізованих послуг важливе значення має забезпечення безпечних умов праці для персоналу, який виконує проєктування, налаштування, обслуговування та експлуатацію програмно-апаратних засобів системи. Організаційно-правові основи охорони праці визначають порядок створення безпечного виробничого середовища, регламентують права та обов'язки працівників і роботодавця, а також встановлюють вимоги щодо запобігання виробничим ризикам та аварійним ситуаціям.

Організація охорони праці в Україні здійснюється відповідно до нормативно-правових актів, державних стандартів та правил безпеки праці. Основним законодавчим документом є Закон України «Про охорону праці», який визначає основні принципи державної політики у сфері безпеки праці та регулює взаємовідносини між працівником і роботодавцем щодо забезпечення безпечних умов праці.

4.2 Характеристика об'єкта та виявлення потенційних небезпек

Під час роботи з IoT-системами працівники взаємодіють із комп'ютерною технікою, мережевим обладнанням, мікроконтролерами, сенсорами, серверами, системами електроживлення та телекомунікаційними пристроями. У зв'язку з цим виникає необхідність дотримання вимог електробезпеки, пожежної безпеки, ергономіки робочих місць та інформаційної безпеки.

Під час експлуатації IoT-систем важливе значення має організація безпечного робочого місця. Працівники тривалий час працюють за

комп’ютером, тому необхідно забезпечити належне освітлення, ергономічне розташування обладнання та дотримання режиму праці й відпочинку. Недотримання цих вимог може призвести до перевтоми, погіршення зору та розвитку захворювань опорно-рухового апарату.

Характеристика робочого місця. Робоче місце включає персональний комп’ютер або ноутбук; монітор; клавіатуру та комп’ютерну мишу; мережеве обладнання; електричні кабелі та подовжувачі; робочий стіл і стілець.

Приміщення має штучне освітлення та систему вентиляції. Живлення обладнання здійснюється від електромережі 220 В.

Під час виконання роботи можливий вплив електричних, пожежних, ергономічних, санітарно-гігієнічних та психофізіологічних факторів.

Таблиця 4.1 – Основні небезпечні та шкідливі фактори

№	Фактор	Джерело небезпеки	Можливі наслідки
1	Електричний струм	ПК, мережеве обладнання	Ураження струмом
2	Коротке замикання	Пошкоджена ізоляція	Пожежа
3	Перенапруження зору	Монітор	Погіршення зору
4	Неправильна поза	Робоче місце	Захворювання опорно-рухового апарату
5	Підвищена температура	Недостатня вентиляція	Перевтома
6	Психічне навантаження	Інтенсивна розробка	Стрес

Розглянемо основні небезпечні та шкідливі фактори більш детально.

Електричні небезпеки пов’язані з тим, що усі пристрої підключені до електромережі. У разі пошкодження ізоляції можливе ураження електричним струмом. Основними причинами цих небезпек є несправні блоки живлення;

використання неякісних подовжувачів; перевантаження мережі. Відповідно до Правил улаштування електроустановок (ПУЕ) необхідно забезпечити справність електрообладнання та наявність захисного заземлення [7].

Пожежна небезпека обумовлена тим, що джерелом займання може бути електронне обладнання. Причиною може бути коротке замикання або перегрів. Згідно з Кодексом цивільного захисту України [8], у приміщенні повинні бути справні вогнегасники; план евакуації; вільний доступ до виходів.

Ергономічні фактори викликані тим, що тривала робота за комп'ютером може спричинити втому очей; біль у спині; порушення постави. Відповідно до санітарних норм [6], тривалість безперервної роботи за комп'ютером не повинна перевищувати встановлених нормативів. Необхідно робити перерви кожні 60 хвилин.

Мікроклімат та освітлення мають велике значення при роботі за комп'ютером. Температура повітря у приміщенні повинна становити 18 – 24 °С. Вологість має бути в межах 40–60 %. Недостатнє освітлення призводить до швидкої втоми. Освітленість робочої поверхні повинна відповідати нормам ДБН [9].

Таблиця 4.2 – Нормативні параметри мікроклімату та освітленості

Показник	Нормативне значення
Температура	18–24 °С
Вологість	40–60 %
Освітленість	не менше 300 лк

Психофізіологічні фактори обов'язково потрібно враховувати для організації робочого міста користувача ПК. Розробка програмного забезпечення потребує високої концентрації уваги. Тривала розумова праця може викликати стрес та перевтому. Для зниження ризику необхідно

планувати робочий час; робити регулярні перерви; забезпечити комфортні умови праці.

4.3 Дослідження ризику реалізації потенційних небезпек на об'єкті проєктування та розробка заходів щодо їх попередження

Проведемо аналіз та оцінку ризиків, пов'язаних із розробкою багатоагентної архітектури IoT-системи для надання спеціалізованих послуг. Особливість цієї бакалаврської роботи полягає в тому, що IoT система включає програмних агентів, серверну частину, мережеву інфраструктуру та IoT-пристрої. Тому під час аналізу враховуються не лише ризики для здоров'я розробника, а й технічні та інформаційні ризики, які можуть вплинути на безпечність експлуатації системи. Оцінювання ризиків проводиться відповідно до принципів управління ризиками, визначених у стандарті ISO 31000 [10], а також вимог Закону України «Про охорону праці» [5].

Метою аналізу ризиків є визначення потенційних небезпек під час розробки та тестування IoT-системи; оцінка ймовірності їх виникнення; визначення тяжкості наслідків; встановлення рівня ризику; визначення напрямів його зниження.

Для оцінювання ризиків використано якісний метод. Ризик визначається як поєднання ймовірності події та тяжкості її наслідків.

Таблиця 4.3 – Шкала оцінки ймовірності

Рівень	Характеристика
Низький	Подія малоймовірна
Середній	Подія можлива
Високий	Подія виникає часто

Таблиця 4.4 – Шкала оцінки наслідків

Рівень	Характеристика
Незначні	Тимчасовий дискомфорт або незначні збої
Середні	Порушення працездатності або зупинка системи
Тяжкі	Травми, пожежа або значні матеріальні втрати

Зробимо аналіз ризиків під час розробки IoT-системи.

Під час тестування обладнання виникають електричні ризики. Вони повязані з тим, що під час підключення IoT-пристроїв до електромережі існує ризик ураження електричним струмом. Ймовірність – низька. Наслідки – тяжкі. Рівень ризику – середній.

Пожежний ризик існує при експлуатації серверного та мережевого обладнання. IoT-система працює безперервно. Обладнання може перегріватися. Ймовірність – низька або середня. Наслідки – тяжкі. Рівень ризику – середній.

Окрім того, при розробці IoT-системи є ризик втрати даних. Багатоагентна система обробляє телеметричні дані. У разі збою серверу або відключення електроенергії можлива втрата інформації. Ймовірність – середня. Наслідки – середні або тяжкі (втрата результатів дослідження). Рівень ризику – середній.

Кібербезпековий ризик пов'язаний з тим, що IoT-система підключена до мережі Інтернет. Існує ризик несанкціонованого доступу або атак на сервер. Ймовірність – середня. Наслідки – середні або тяжкі. Рівень ризику – середній або високий.

При розробці IoT-системи можуть бути ризики програмних помилок. У багатоагентній системі можливі логічні помилки у взаємодії агентів, що може

призвести до некоректного надання спеціалізованих послуг. Ймовірність – середня. Наслідки – середні. Рівень ризику – середній.

Не можна не враховувати і ергономічний ризик для розробника. Він пов'язаний з тим, що розробка архітектури та програмного коду потребує тривалої роботи за комп’ютером. Ймовірність – висока. Наслідки – незначні або середні. Рівень ризику – середній.

Узагальнена оцінка ризиків надана у таблиці 4.5.

Таблиця 4.5 – Загальна оцінка ризиків для IoT-системи

№	Небезпека	Ймовірність	Наслідки	Рівень ризику
1	Ураження струмом	Низька	Тяжкі	Середній
2	Пожежа	Низька/Середня	Тяжкі	Середній
3	Втрата даних	Середня	Середні	Середній
4	Кіберзагрози	Середня	Середні/Тяжкі	Середній
5	Програмні помилки	Середня	Середні	Середній
6	Перевтома	Висока	Незначні	Середній

У звязку з військовою агресією росії перелік ризиків в Україні значно збільшився. В умовах України можливі аварійні відключення електроенергії; пожежі, викликані обстрілами; воєнні загрози. У таких випадках можуть виникнути зупинка серверної частини; пошкодження обладнання; втрата результатів дослідження. Тому необхідно передбачити резервне копіювання даних та аварійне відключення обладнання.

Аналіз показав, що більшість ризиків під час розробки багатоагентної IoT-системи мають середній рівень. Найбільш критичними є електричні, пожежні та кібербезпекові ризики. Оцінка ризиків дозволяє визначити пріоритети у впровадженні заходів безпеки. У наступному підрозділі будуть розроблені конкретні заходи щодо зниження виявлених ризиків.

У сучасному сталому виробництві IoT-системи контролюють критичні параметри: від тиску в котлах до концентрації шкідливих викидів. Виникнення надзвичайної ситуації (техногенна аварія, відключення енергії, кібератака) потребує від архітектури системи здатності до самоорганізації та миттєвого реагування [13].

Традиційні централізовані системи мають «єдину точку відмови»: якщо виходить з ладу центральний сервер, вся система «сліпне». Багатоагентна архітектура складається з автономних програмних одиниць (агентів), які взаємодіють між собою [14]. До основних властивостей агентів у режимі надзвичайних ситуацій можна віднести автономність, реактивність, проактивність. Автономність означає, що агент може прийняти рішення про зупинку верстата локально, навіть якщо зв'язок із хмарою розірвано. Реактивність свідчить про здатність миттєво реагувати на зміну зовнішнього середовища (наприклад, різкий стрибок температури). Проактивність демонструє здатність передбачати розвиток надзвичайних ситуацій на основі аналізу даних від сусідніх агентів. Менеджмент безпеки в багатоагентних системах базується на каскадному переході режимів роботи (див. табл. 4.6).

Таблиця 4.6 – Алгоритм функціонування системи при виникненні НС

Стадія НС	Дія агентної мережі	Технологічне рішення
Виявлення	Агенти-сенсори фіксують відхилення від норми.	Edge Computing (обробка на межі мережі).
Ізоляція	Агенти локалізують зону аварії, щоб вона не поширилася.	Динамічна реконфігурація мережі.
Реагування	Агенти-актуатори переводять обладнання в безпечний стан.	Аварійні протоколи зупинки (Fail-safe).
Відновлення	Перерозподіл завдань між справними агентами.	Кооперативна реорганізація.

Надзвичайна ситуація може бути наслідком цілеспрямованої атаки на цифрову інфраструктуру. Для забезпечення сталого менеджменту в MAS-архітектурі використовуються такі інструменти, як Blockchain, візантійська відмовостійкість, локальні енергетичні острови. Децентралізований консенсус (Blockchain) використовується щоб зловмисник не міг надіслати фальшиву команду на аварійне відключення всьому заводу [13]. Візантійська відмовостійкість відображає здатність системи продовжувати роботу, навіть якщо частина агентів (вузлів) скомпрометована або видає некоректні дані. Локальні енергетичні острови використовуються у разі блекауту IoT-агенти переходять на живлення від відновлюваних джерел (ESS) для підтримки функцій безпеки [14]. У надзвичайних ситуаціях значну роль виконує менеджмент через організацію ефективної комунікації. Багатоагентна система повинна не лише діяти автономно, а й забезпечувати персонал оперативною інформацією шляхом пріоритезація трафіку або цифрових підказок. Пріоритезація трафіку реалізовується наступним чином: у режимі НС канали зв'язку виділяються виключно для тривожних сигналів та інструкцій з евакуації. Цифрові підказки здійснюються агентами через AR-інтерфейси або мобільні додатки, вказуючи працівникам найбезпечніші шляхи виходу або алгоритми гасіння пожежі [13].

Таблиця 4.7 – Заходи безпеки у надзвичайних ситуаціях для багатоагентної IoT-системи

№	Надзвичайна ситуація	Можливі причини	Потенційні наслідки	Організаційні заходи	Технічні заходи	Порядок дій
1	Пожежа	Коротке замикання, перегрів обладнання, перевантаження мережі	Загроза життю, знищення обладнання, втрата даних	Інструктаж з пожежної безпеки, план евакуації, контроль стану електромережі	Автоматичні вимикачі, заземлення, вентиляція, вогнегасники	Відключити живлення, викликати 101, евакуюватися
2	Відключення електроенергії	Аварія в мережі, воєнні пошкодження, планові роботи	Зупинка серверів, пошкодження БД, втрата телеметрії	Контроль стану UPS, перевірка резервного копіювання	Джерело безперебійного живлення (UPS), автозапуск сервісів, RAID, резервні копії	Коректно завершити роботу, перевірити відновлення системи
3	Кібератака	Несанкціонований доступ, DDoS, віруси	Викрадення або підміна даних, зупинка сервісів	Розмежування прав доступу, контроль доступу до серверів	TLS-шифрування, firewall, автентифікація, оновлення ПЗ, логування	Обмежити доступ, проаналізувати журнали, відновити дані з резерву
4	Воєнна або техногенна загроза	Повітряна тривога, вибух, пошкодження інфраструктури	Руйнування обладнання, втрата даних	Дотримання правил цивільного захисту, знання розташування укриття	Хмарне резервне копіювання, дублювання критичних даних	Зберегти дані, вимкнути обладнання, пройти до укриття
5	Перегрів серверного обладнання	Недостатня вентиляція, велике навантаження	Збій роботи агентів, пошкодження обладнання	Контроль температурного режиму	Системи охолодження, моніторинг температури	Перевірити вентиляцію, зменшити навантаження

Таблиця 4.8 – Комплекс заходів для забезпечення безперервності роботи IoT-системи

Напрям безпеки	Основні заходи	Очікуваний результат
Електробезпека	Заземлення, UPS, автоматичні вимикачі	Запобігання ураженню струмом та пошкодженню обладнання
Пожежна безпека	Вогнегасники, вентиляція, контроль навантаження	Зменшення ризику займання
Інформаційна безпека	TLS, firewall, автентифікація, резервне копіювання	Захист даних та стабільність сервісів
Організаційна безпека	Інструктажі, план евакуації, контроль доступу	Зменшення людського фактору
Безперервність роботи	Автозапуск сервісів, дублювання даних, хмарне зберігання	Стійкість до аварій та швидке відновлення роботи

Забезпечення безпеки в НС здійснюється відповідно до вимог Кодексу цивільного захисту України [8]. Комплексне застосування організаційних і технічних рішень, які систематизовано у таблицях 4.7 – 4.8, дозволяє мінімізувати наслідки надзвичайних ситуацій; забезпечити збереження даних; підтримувати безперервність функціонування програмних агентів; підвищити надійність і стійкість системи в умовах реальних загроз. Виявлення цих факторів дозволяє розробити заходи щодо забезпечення безпечних умов праці під час створення багатоагентної IoT-системи.

Висновок до розділу 4.

У розділі 4 бакалаврської роботи було розглянуто питання охорони праці та безпеки у надзвичайних ситуаціях під час розробки багатоагентної архітектури IoT-системи для надання спеціалізованих послуг.

Визначено умови виконання роботи та ідентифіковано потенційні небезпечні й шкідливі фактори. Встановлено, що основними ризиками є електричні та пожежні небезпеки, ергономічні навантаження, несприятливі

параметри мікроклімату, а також ризики, пов’язані з експлуатацією серверного та мережевого обладнання. Окремо враховано специфіку IoT-системи, яка передбачає постійне функціонування програмних агентів та обробку даних у реальному часі.

Проведено аналіз та якісну оцінку ризиків. Визначено ймовірність виникнення небезпечних подій та тяжкість їх наслідків. За результатами оцінювання встановлено, що більшість ризиків мають середній рівень. Найбільш критичними є електричні, пожежні та кібербезпекові ризики, а також ризик втрати даних у разі аварійного відключення електроенергії.

Розглянуто заходи безпеки у надзвичайних ситуаціях. Проаналізовано порядок дій у разі пожежі, аварійного відключення електроживлення, кібератаки та інших загроз. Запропоновано організаційні та технічні заходи, зокрема використання джерел безперебійного живлення, резервного копіювання даних, застосування засобів мережевого захисту та дотримання правил евакуації.

Проведений аналіз підтверджує, що під час розробки та експлуатації багатоагентної IoT-системи необхідно комплексно враховувати вимоги охорони праці та цивільного захисту. Реалізація запропонованих заходів дозволяє зменшити рівень ризику, забезпечити безпечні умови праці та підвищити надійність функціонування системи в штатних і надзвичайних ситуаціях.

ЗАГАЛЬНІ ВИСНОВКИ

У бакалаврській роботі виконано комплексне дослідження та практичну реалізацію багатоагентної архітектури IoT-системи для надання спеціалізованих послуг.

У розділі 1 проведено аналіз теоретичних основ побудови IoT-систем та багатоагентних систем. Розглянуто особливості функціонування IoT-середовища, зокрема розподіленість, масштабованість, обмеженість ресурсів пристроїв та необхідність роботи в реальному часі. Досліджено архітектурні підходи до побудови багатоагентних систем, їх переваги та принципи організації взаємодії агентів. Проаналізовано сучасні програмні платформи, протоколи та інструменти для реалізації IoT-рішень. За результатами аналізу обґрунтовано доцільність використання багатоагентного підходу для створення гнучкої та масштабованої системи надання спеціалізованих послуг.

У розділі 2 виконано проєктування системи. Сформовано функціональні та нефункціональні вимоги до розроблюваної IoT-системи. Розроблено структурну модель багатоагентної архітектури із визначенням типів агентів, їх ролей та функцій. Здійснено моделювання процесів взаємодії агентів на основі подієвого принципу та моделі publish/subscribe. Це дозволило забезпечити слабку зв’язаність компонентів та можливість подальшого масштабування системи.

У розділі 3 реалізовано програмну частину багатоагентної IoT-системи. Обґрунтовано вибір програмних засобів, мови програмування та середовища розробки. Реалізовано агент збору даних, агент обробки та аналізу, координаційний агент і користувацький агент. Розроблено механізми їх взаємодії через MQTT-брокер. Створено серверну частину системи з використанням мікросервісного підходу, реалізовано REST API та інтеграцію з базою даних.

У розділі 4 розглянуто питання охорони праці. Ідентифіковано потенційні небезпечні фактори під час розробки та експлуатації IoT-системи.

Проведено аналіз та оцінку ризиків, визначено їх рівень. Запропоновано організаційні та технічні заходи щодо зниження ризиків, забезпечення електробезпеки, пожежної безпеки, інформаційного захисту та безперервності роботи системи. Окрему увагу приділено заходам безпеки в умовах надзвичайних ситуацій, що є актуальним в сучасних умовах.

У результаті виконання бакалаврської роботи досягнуто поставленої мети – розроблено багатоагентну архітектуру IoT-системи для надання спеціалізованих послуг, виконано її програмну реалізацію та обґрунтовано заходи безпечної експлуатації. Отримані результати можуть бути використані як основа для подальшого розширення функціональності системи, впровадження у практичну діяльність та розвитку інтелектуальних розподілених сервісів на базі IoT-технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wooldridge M. An Introduction to MultiAgent Systems. – 2nd ed. – Chichester: John Wiley & Sons, 2009. – 484 p.
2. Jennings N. R., Wooldridge M. Agent Technology: Foundations, Applications, and Markets. – Berlin: Springer, 1998. – 282 p.
3. Banks A., Gupta R. MQTT Version 3.1.1. OASIS Standard. – 2014. – 81 p.
4. Tanenbaum A. S., Van Steen M. Distributed Systems: Principles and Paradigms. – 2nd ed. – Upper Saddle River: Prentice Hall, 2007. – 686 p.
5. Про охорону праці : Закон України від 14.10.1992 № 2694-XII (ред. станом на 2024 р.). URL: https://zakon.rada.gov.ua.
6. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. – Київ, 1998.
7. Правила улаштування електроустановок (ПУЕ). – Київ: Мінпаливенерго України, 2017. – 617 с.
8. Кодекс цивільного захисту України: Закон України від 02.10.2012 № 5403-VI // Відомості Верховної Ради України. – 2013. – № 34–35. – Ст. 458. URL: https://zakon.rada.gov.ua
9. ДБН В.2.5-28-2006. Природне і штучне освітлення. – Чинні від 01.10.2006. – Київ : Мінбуд України, 2006. – 76 с.
10. ISO 31000:2018 Risk management – Guidelines. – Geneva: International Organization for Standardization, 2018.
11. Правила пожежної безпеки в Україні : наказ МВС України № 1417 від 30.12.2014 (ред. станом на 2023 р.). URL: https://zakon.rada.gov.ua
12. ДСТУ ISO 45001:2019 (чинний у 2021–2026 рр.). Системи управління охороною здоров’я та безпекою праці. Вимоги та настанови щодо застосування. Київ : ДП «УкрНДНЦ», 2019.

13. Гальцова О. Л. Зелена економіка та цифровізація як стратегічні пріоритети післявоєнного відновлення України. Причорноморські економічні студії. 2023. Вип. 81. С. 12–18.

14. Світове господарство та міжнародні економічні відносини: трансформація в умовах «зеленого» переходу : монографія / за заг. ред. д.е.н. І. П. Макаренко. Суми : Університетська книга, 2023. 412 с.

15. Heizer J., Render B., Munson C. Operations Management: Sustainability and Supply Chain Management. 13th ed. Pearson, 2023. 892 p.

16. Laudon K., Laudon J. Management Information Systems: Managing the Digital Firm. 17th ed. Pearson, 2022. 656 p.

17. Методичні вказівки до виконання розділу «Охорона праці та безпека в надзвичайних ситуаціях». Харків : ХНУМГ ім. О. М. Бекетова, 2021.

18. ISO/IEC 30141:2024. Internet of Things (IoT) – Reference Architecture. – Geneva : International Organization for Standardization, 2024. – 74 p.

19. ISO/IEC 27400:2022. Cybersecurity – IoT security and privacy – Guidelines. – Geneva : International Organization for Standardization, 2022. – 52 p.

20. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection – Information security management systems – Requirements. – Geneva : International Organization for Standardization, 2022. – 34 p.

21. IEEE Std 2413-2021. IEEE Standard for an Architectural Framework for the Internet of Things (IoT). – New York : IEEE, 2021. – 45 p.

22. Newman S. Building Microservices. – 2nd ed. – Sebastopol : O’Reilly Media, 2021. – 617 p.

23. Richardson C. Microservices Patterns. – 2nd ed. – Shelter Island : Manning Publications, 2023. – 650 p.

24. Bordini R. H., Dastani M., Dix J., Seghrouchni A. E. Multi-Agent Programming: Languages, Platforms and Applications. – 2nd ed. – Cham : Springer, 2021. – 410 p.

25. Huhns M. N., Singh M. P. Multiagent Systems: Foundations and Applications. – 2nd ed. – Hoboken : Wiley-IEEE Press, 2022. – 560 p.

26. MQTT Version 5.0 [Electronic resource]. – OASIS Standard, 2023. – Available at: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (accessed: 03.03.2026).

27. NIST. IoT Device Cybersecurity Guidance for the Federal Government: NIST SP 800-213A [Electronic resource]. – Gaithersburg : National Institute of Standards and Technology, 2021. – Available at: https://csrc.nist.gov/publications/detail/sp/800-213a/final (accessed: 03.03.2026).

28. ENISA. Good Practices for IoT and Smart Infrastructure Security [Electronic resource]. – Brussels : European Union Agency for Cybersecurity, 2023. – Available at: https://www.enisa.europa.eu/publications (accessed: 03.03.2026).

29. Python Software Foundation. Python 3.12 Documentation [Electronic resource]. – 2024. – Available at: https://docs.python.org/3/ (accessed: 03.03.2026).

30. Eclipse Foundation. Eclipse Mosquitto Documentation [Electronic resource]. – 2025. – Available at: https://mosquitto.org/documentation/ (accessed: 03.03.2026).

31. Docker Inc. Docker Engine Documentation [Electronic resource]. – 2025. – Available at: https://docs.docker.com/ (accessed: 03.03.2026).

32. Pallets Projects. Flask Documentation [Electronic resource]. – 2025. – Available at: https://flask.palletsprojects.com/ (accessed: 03.03.2026).

33. Kleppmann M. Designing Data-Intensive Applications. – 2nd ed. – Sebastopol : O’Reilly Media, 2024. – 720 p.

34. Bures T., Gerostathopoulos I., Hnetynka P. Microservice Architecture in Practice. – Cham : Springer, 2022. – 320 p.

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ МІСЬКОГО ГОСПОДАРСТВА
імені О. М. БЕКЕТОВА
Навчально-науковий інститут енергетичної, інформаційної та транспортної інфраструктури
Кафедра автоматизації та комп'ютерно-інтегрованих технологій

РОЗРОБКА БАГАТОАГЕНТНОЇ АРХІТЕКТУРИ IoT-СИСТЕМИ ДЛЯ НАДАННЯ СПЕЦІАЛІЗОВАНИХ ПОСЛУГ

здобувач вищої освіти групи СІНЖ 2022-1
Спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології
Безнос Владислав Сергійович
Керівник Піддубна Л.В., к. філос. н., доц., доцент кафедри АКІТ

АКТУАЛЬНІСТЬ

Інтернет речей об'єднує значну кількість пристроїв, які повинні взаємодіяти між собою та своєчасно надавати послуги у розподіленому середовищі. Постійне зростання кількості підключених пристроїв ускладнює процеси обробки даних, координації компонентів і забезпечення сервісів у режимі реального часу. Використання багатоагентної архітектури створює можливість організувати автономну та узгоджену роботу елементів системи, що підвищує ефективність і надійність надання спеціалізованих послуг.

2

МЕТА

Метою роботи є розробка багатоагентної архітектури IoT-системи, що забезпечує ефективну координацію агентів та адаптивне надання спеціалізованих послуг.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- Проаналізувати сучасні підходи до побудови IoT-систем та багатоагентних архітектур.
- Дослідити принципи взаємодії програмних агентів у розподіленому середовищі.
- Визначити вимоги до архітектури IoT-системи для надання спеціалізованих послуг.
- Розробити структурну модель багатоагентної IoT-системи.

3

ОБ'ЄКТ І ПРЕДМЕТ ДОСЛІДЖЕННЯ

Об'єктом дослідження є процеси побудови та функціонування багатоагентних IoT-систем.

Предметом дослідження є моделі, методи та архітектурні рішення організації взаємодії агентів у IoT-системі для надання спеціалізованих послуг.

4

Особливості побудови IoT-систем

IoT – це концепція побудови мережі взаємопов'язаних фізичних пристроїв (сенсорів, контролерів, виконавчих механізмів, промислового обладнання, транспортних систем, побутової техніки та інших об'єктів), які здатні збирати, передавати, обробляти та обмінюватися даними через мережу Інтернет без безпосереднього втручання людини. Основною особливістю IoT-систем є інтеграція фізичного та цифрового середовища. Система отримує інформацію із реального світу за допомогою сенсорів, аналізує її та формує відповідні дії або повідомлення. Завдяки цьому IoT-технології дозволяють автоматизувати процеси моніторингу, управління та прийняття рішень.

5

Загальна структура IoT-системи

6

Основні компоненти IoT-системи

Компонент	Призначення	Приклад
Датчик	Збір фізичних параметрів	Датчик температури
Контролер	Обробка сигналів	Arduino, ESP32
Мережевий модуль	Передача даних	Wi-Fi, GSM, LoRa
Сервер	Збереження та обробка даних	Хмарна платформа
Користувацький інтерфейс	Відображення інформації	Веб-додаток

7

Основні вимоги до IoT-систем

Вимога	Пояснення
Масштабованість	Підтримка великої кількості пристроїв
Надійність	Стійкість до відмов
Безпека	Захист даних і пристроїв
Гнучкість	Можливість розширення функцій
Енергоефективність	Мінімальне споживання енергії

8

Архітектурні моделі багатоагентних систем

Архітектурна модель	Основна характеристика	Принцип роботи	Переваги	Недоліки	Приклади застосування
Реактивна архітектура	Агенти швидко реагують на події середовища без складного планування	Робота базується на правилах типу «подія – реакція»	Висока швидкодія, простота реалізації, низьке споживання ресурсів	Обмежені можливості аналізу та планування, складність реалізації складної логіки	Системи моніторингу, автоматичне керування датчиками, IoT-пристрої реального часу
Ієрархічна архітектура	Система має структуру підпорядкування агентів	Центральний або керуючий агент координує роботу інших агентів	Зручне управління, чіткий розподіл ролей, простий контроль системи	Залежність від центрального агента, можливе перевантаження керівного рівня	Промислові системи управління, корпоративні мережі, системи диспетчеризації
Гібридна архітектура	Поєднує реактивний та ієрархічний підходи	Частина агентів працює автономно, а частина координується центральними агентами	Гнучкість, масштабованість, адаптивність, поєднання швидкодії та керованості	Складніша реалізація та налаштування	Інтелектуальні IoT-системи, розумні будівлі, адаптивні системи автоматизації

9

Сучасні рішення для створення IoT-систем

10

Порівняння хмарних IoT-платформ

Платформа	Переваги	Недоліки
AWS IoT Core	Висока масштабованість, інтеграція з іншими сервісами AWS	Складність налаштування, платна модель
Azure IoT Hub	Гнучке керування пристроями, підтримка аналітики	Висока вартість при великій кількості пристроїв
Google Cloud IoT	Зручна інтеграція з аналітичними сервісами	Обмеження в безкоштовному тарифі

11

Загальна схема використання хмарної IoT-платформи

12

Порівняння брокерів повідомлень

Рішення	Призначення	Переваги	Недоліки
Eclipse Mosquitto	Передача повідомлень MQTT	Легкість, невелике навантаження	Менша продуктивність при великих обсягах
Apache Kafka	Потокова обробка даних	Висока швидкість, масштабованість	Складність конфігурації

13

Порівняння платформ для багатоагентних систем

Платформа	Мова	Переваги	Недоліки
JADE	Java	Підтримка стандартів, готові механізми взаємодії	Потребує знання Java
SPADE	Python	Простота реалізації, інтеграція з веб-сервісами	Менша кількість інструментів
Самостійна реалізація	Будь-яка	Гнучкість	Більші витрати часу на розробку

14

Особливості мов програмування для IoT

Мова	Де використовується	Переваги	Обмеження
C++	Пристрої, контролери	Висока швидкодія, доступ до апаратури	Складність розробки
Python	Сервер, аналітика, агенти	Простота, швидка розробка	Нижча продуктивність
Java	Сервер, агентні системи	Платформонезалежність	Більше споживання ресурсів

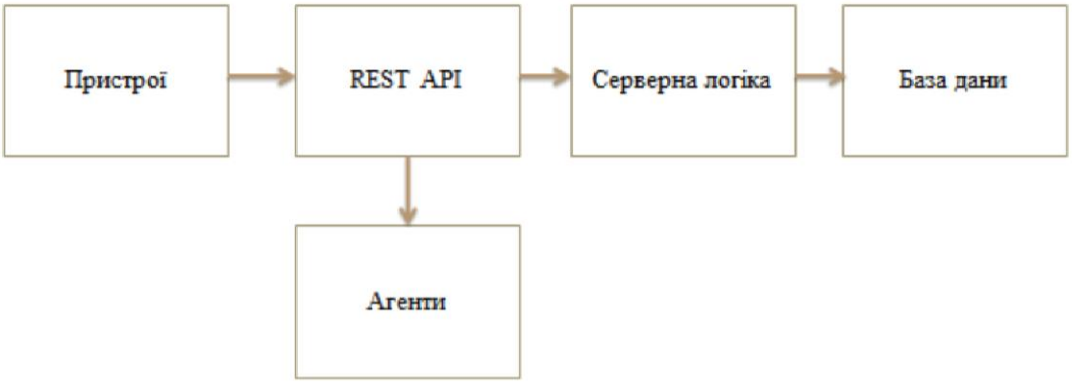
15

Порівняння баз даних

База даних	Тип	Переваги	Доцільність використання
PostgreSQL	Реляційна	Надійність, підтримка складних запитів	Облік пристроїв, логів, користувачів
MongoDB	NoSQL	Гнучка структура, масштабованість	Потокові дані, телеметрія

16

ІНТЕГРАЦІЯ КОМПОНЕНТІВ ІоТ-СИСТЕМИ



17

ФУНКЦІОНАЛЬНІ ВИМОГИ ДО БАГАТОАГЕНТНОЇ ІоТ-СИСТЕМИ

№	Вимога	Опис
F1	Збір даних	Постійний збір показників з усіх підключених сенсорів та пристроїв.
F2	Обробка даних	Аналіз даних агентами для прийняття рішень у реальному часі.
F3	Надання спеціалізованих послуг	Відображення результатів через веб- або мобільний інтерфейс.
F4	Координація агентів	Агенти обмінюються інформацією та узгоджують дії.
F5	Сповіщення	Інформування користувачів про критичні події та зміни стану системи.

18

НЕФУНКЦІОНАЛЬНІ ВИМОГИ ДО БАГАТОАГЕНТНОЇ IOT-СИСТЕМИ

№	Вимога	Опис
N1	Надійність	Робота 24/7 без критичних збоїв.
N2	Масштабованість	Додавання нових пристроїв та агентів без перерви в роботі.
N3	Безпека	Шифрування даних та контроль доступу користувачів.
N4	Продуктивність	Мінімальна затримка при обробці даних та виконанні дій агентів.
N5	Зручність	Інтерфейс має бути інтуїтивним та доступним для користувачів.

19

ТИПИ АГЕНТІВ ТА ЇХ ФУНКЦІЇ

№	Тип агента	Функції
A1	Агент збору даних	Збір даних від сенсорів, передача інформації агентам обробки.
A2	Агент обробки даних	Аналіз даних, формування висновків, прийняття рішень.
A3	Агент координації	Координація дій між агентами, оптимізація виконання завдань.
A4	Агент інтерфейсу користувача	Надання даних користувачу, прийом команд та запитів.
A5	Агент сповіщень	Надсилання повідомлень про аварійні події або критичні зміни.

20

Взаємодія агентів

Джерело	Приймач	Тип взаємодії	Мета
Агент збору даних	Агент обробки даних	Передача даних	Надання актуальної інформації для аналізу
Агент обробки	Агент сповіщень	Надсилання повідомлень	Інформування користувача про події
Агент обробки	Агент інтерфейсу користувача	Відображення результатів	Передача інформації користувачу
Агент координації	Всі агенти	Контроль та координація	Оптимізація взаємодії та виконання завдань

21

Прийняття рішень відбувається на основі отриманих даних та алгоритму

Отримати нові дані → Перевірити дані на відповідність пороговим значенням → Якщо умова виконується – Сформувати дію → Якщо умова не виконується – продовжити моніторинг →Передати результат іншим агентам або користувачу

22

Агент збору даних

Агент обробки та аналізу

КООРДИНАЦІЙНИЙ АГЕНТ

25

КОРИСТУВАЦЬКИЙ АГЕНТ

26

Заходи безпеки у надзвичайних ситуаціях для багатоагентної IoT-системи

№	Надзвичайна ситуація	Можливі причини	Потенційні наслідки	Організаційні заходи	Технічні заходи	Порядок дій
1	Пожежа	Коротке замикання, перегрів обладнання, перевантаження мережі	Загроза життю, знищення обладнання, втрата даних	Інструктаж з пожежної безпеки, план евакуації, контроль стану електромережі	Автоматичні вимикачі, заземлення, вентиляція, вогнегасники	Відключити живлення, викликати 101, евакуюватися
2	Відключення електроенергії	Аварія в мережі, воєнні пошкодження, планові роботи	Зупинка серверів, пошкодження БД, втрата телеметрії	Контроль стану UPS, перевірка резервного копіювання	Джерело безперебійного живлення (UPS), автозапуск сервісів, RAID, резервні копії	Коректно завершити роботу, перевірити відновлення системи
3	Кібератака	Несанкціонований доступ, DDoS, віруси	Викрадення або підміна даних, зупинка сервісів	Розмежування прав доступу, контроль доступу до серверів	TLS-шифрування, firewall, автентифікація, оновлення ПЗ, логування	Обмежити доступ, проаналізувати журнали, відновити дані з резерву
4	Воєнна або техногенна загроза	Повітряна тривога, вибух, пошкодження інфраструктури	Руйнування обладнання, втрата даних	Дотримання правил цивільного захисту, знання розташування укриття	Хмарне резервне копіювання, дублювання критичних даних	Зберегти дані, вимкнути обладнання, пройти до укриття
5	Перегрів серверного обладнання	Недостатня вентиляція, велике навантаження	Збій роботи агентів, пошкодження обладнання	Контроль температурного режиму	Системи охолодження, моніторинг температури	Перевірити вентиляцію, зменшити навантаження

27

ДЯКУЮ ЗА УВАГУ!

28