НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий Інститут телекомунікаційних систем

Кафедра електронних комунікацій та інтернету речей

«До захисту допущено»
ВО завідувача кафедри
_____ В’ячеслав НОСКОВ
«___»_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра

зі спеціальності 172 Телекомунікації та радіотехніка

на тему: «Розробка шлюзу IoT на основі мікроконтролера Raspberry PI та інтерфейсу LTE/NB-IoT SIM7000E»

Виконав:
студент IV курсу, групи ТС-12
Гаврилюк Олександр Володимирович _____

Керівник:
Доцент кафедри ЕКІР ІТС, к.т.н.
Осипчук С.О. _____

Рецензент:
доцент кафедри ТК НН ІТС, к.т.н., с.н.с.
Міночкін Д.А. _____

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

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий Інститут телекомунікаційних систем
Кафедра електронних комунікацій та інтернету речей

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 Телекомунікації та радіотехніка

Освітня програма – «Системи електронних комунікацій та інтернету речей»

ЗАТВЕРДЖУЮ
ВО завідувача кафедри
_____ В’ячеслав НОСКОВ
«___»_____2025 р.

ЗАВДАННЯ

на дипломну роботу студенту

Гаврилюку Олександру Володимировичу

1. Тема роботи «Розробка шлюзу IoT на основі мікроконтролера Raspberry PI та інтерфейсу LTE/NB-IoT SIM7000E»», керівник роботи Осипчук Сергій Олександрович, доцент кафедри ЕКІР, затверджені наказом по університету від «26» травня 2025 р. № 1755 –с.

2. Термін подання студентом роботи 13 червня 2025 року

3. Вихідні дані до роботи: технічна документація, специфікації модулів, офіційні API, приклади реалізацій, нормативні документи, матеріали статей та наукових видань

4. Зміст роботи

Теоретичні основи IOT-систем та бездротових технологій зв’язку

Проектування IOT-шлюзу на базі RASPBERRY PI та SIM7000E

Аналіз роботи системи та потенціал її використання

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо):1)Тема та мета дипломної роботи; 2)Загальна структура IoT-шлюзу; 3)Функціональні сценарії передачі даних (Wi-Fi/TCP); 4)Вибір те хнологій стільникового зв’язку: LTE Cat-M1 та NB-IoT; 5) Обробка GPS-координат та

логіка перемикання каналів; 6) Реалізація передачі даних через Telegram Bot API; 7) TCP-з’єднання з хмарним сервером у разі відсутності Wi-Fi; 8) Програмна реалізація IoT-системи; 9) Результати тестування та автономність системи; 10) Висновки та перспективи розвитку IoT-шлюзу

6. Дата видачі завдання 20.11.2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Теоретичні основи IoT-систем та бездротових технологій зв’язку	18.04.2025	Виконано
2	Проектування IOT-шлюзу на базі RASPBERRY PI та SIM7000E	28.05.2025	Виконано
3	Аналіз роботи системи та потенціал її використання	08.06.2025	Виконано
4	Оформлення роботи	12.06.2025	Виконано

Студент Олександр ГАВРИЛЮК

Керівник роботи Сергій ОСИПЧУК

РЕФЕРАТ

Текстова частина дипломної роботи: 80 с., 7 рис., 2 табл., 32 джерел.

Мета роботи — розробка автономного шлюзу Інтернету речей (IoT) на основі мікрокомп'ютера Raspberry Pi та стільникового модему SIM7000E, здатного передавати координати GPS до кінцевого користувача в умовах як наявного, так і відсутнього інтернет-з'єднання.

Об'єкт дослідження – стандарти та технології 3GPP (LTE/NB-IoT) для побудови рішень IoT.

Предмет досліджень – дослідження і налаштування рішень IoT на основі модуля зв'язку 3GPP SIM7000E та мікроконтролера Raspberry Pi.

У дипломній роботі розглянуто принципи побудови IoT-систем, структуру шлюзових рішень, можливості модуля SIM7000E та мікрокомп'ютера Raspberry Pi. Проаналізовано сучасні протоколи передавання даних (TCP, HTTPS), методи забезпечення захищеності з'єднання (TLS), а також технології мобільного зв'язку LTE Cat-M1 та NB-IoT.

Основну увагу приділено реалізації гібридної логіки передавання GPS-даних: за наявності Wi-Fi координати надсилаються через HTTPS-з'єднання напряму до Telegram-бота, у разі його відсутності – модем за ініціативою Raspberry Pi встановлює TCP-з'єднання із хмарним сервером, який пересилає координати до Telegram. Практична частина включає програмну реалізацію всіх логічних сценаріїв, взаємодію з SIM7000E через AT-команди, налаштування перевірки з'єднання, відправку даних через Telegram Bot API та створення сервісу TCP Listener на стороні хмарного сервера. Систему протестовано в умовах перемикання режимів із та без підключення до інтернету.

Результати підтверджують працездатність запропонованого рішення, яке забезпечує надійну передачу координат в реальному часі в умовах обмеженого доступу до мережі, мінімізує втрати даних та не потребує постійного втручання користувача.

Ключові слова: IoT, Raspberry Pi, SIM7000E, GPS, TCP, HTTPS, LTE Cat-M1, NB-IoT, Telegram Bot API, стільникова мережа, шлюз.

ABSTRACT

The text of the thesis: 80 pages, 7 figures, 2 table, 32 references.

The purpose of this work is to develop an autonomous Internet of Things (IoT) gateway based on the Raspberry Pi microcomputer and the SIM7000E cellular modem, capable of transmitting GPS coordinates to the end user under both connected and disconnected network conditions.

Object of research - 3GPP standards and technologies (LTE/NB-IoT) for building IoT solutions.

The subject of research is the study and configuration of IoT solutions based on the 3GPP SIM7000E communication module and the Raspberry Pi microcontroller.

The thesis covers the principles of building IoT systems, gateway architecture, features of the SIM7000E module and the Raspberry Pi, and an overview of data transmission protocols (TCP, HTTPS) and connection security methods (TLS). It also discusses mobile communication technologies such as LTE Cat-M1 and NB-IoT.

The main focus is on implementing hybrid logic for GPS data transmission: if Wi-Fi is available, the coordinates are sent via HTTPS directly to a Telegram bot; if Wi-Fi is unavailable, the Raspberry Pi initiates a TCP connection via the SIM7000E modem to a cloud server, which then forwards the coordinates to Telegram.The practical part includes full implementation of all logical scenarios, communication with the SIM7000E via AT commands, internet connection checking, data transmission through the Telegram Bot API, and the deployment of a TCP Listener service on the cloud server side. The system was tested under network switching conditions.

The results confirm the reliability of the proposed solution for real-time coordinate transmission in limited connectivity environments. The system minimizes data loss and does not require constant user interaction.

Keywords: IoT, Raspberry Pi, SIM7000E, GPS, TCP, HTTPS, LTE Cat-M1, NB-IoT, Telegram Bot API, cellular network, gateway.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface – інтерфейс прикладного програмування, що визначає спосіб взаємодії програмних компонентів.

AT-команда – набір команд, що використовується для керування модемами стандарту Hayes (зокрема SIM7000E), передається через послідовний порт.

CoAP – Constrained Application Protocol – протокол прикладного рівня, оптимізований для обмежених пристроїв та середовищ з низькою пропускною здатністю.

CPU – Central Processing Unit – центральний процесор.

GPIO – General Purpose Input/Output – універсальні цифрові порти введення/виведення на мікроконтролерах або SBC (як Raspberry Pi).

GPS – Global Positioning System – глобальна система позиціонування, яка забезпечує визначення координат об'єкта.

HTTPS – HyperText Transfer Protocol Secure – захищений протокол передачі гіпертексту з використанням шифрування TLS/SSL.

IoT – Internet of Things – Інтернет речей, концепція мережі фізичних пристроїв, які можуть обмінюватися даними через інтернет.

JSON – JavaScript Object Notation – текстовий формат зберігання структурованих даних.

LTE – Long-Term Evolution – стандарт бездротової передачі даних 4G покоління.

MCU – Microcontroller Unit – мікроконтролер, що виконує роль керуючого пристрою в IoT-системах.

MQTT – Message Queuing Telemetry Transport – легкий мережевий протокол публікації/підписки, розроблений для обмежених пристроїв і ненадійних мереж.

NB-IoT – Narrowband Internet of Things – енергоефективна технологія мобільного зв’язку для IoT-пристроїв, що передає дані в обмеженому діапазоні частот.

OS – Operating System – операційна система.

Pi – скорочене позначення мікрокомп’ютера Raspberry Pi.

RAM – Random Access Memory – оперативна пам'ять.

Raspberry Pi – одноплатний комп’ютер, що використовується як обчислювальний центр у багатьох IoT-рішеннях.

SBC – Single Board Computer – одноплатний комп’ютер (наприклад, Raspberry Pi).

SIM7000E – модем від компанії SIMCom, що підтримує стандарти LTE Cat-M1, NB-IoT та GPS.

SoC – System on Chip – система на кристалі, що інтегрує процесор, пам’ять та периферійні контролери в одному чипі.

TCP – Transmission Control Protocol – протокол транспортного рівня, що забезпечує надійну доставку даних через встановлення з’єднання між двома вузлами.

TLS – Transport Layer Security – криптографічний протокол, який забезпечує захист даних при передачі мережею.

UART – Universal Asynchronous Receiver-Transmitter – послідовний інтерфейс обміну даними між пристроями.

VoIP – Voice over IP – передача голосових даних через IP-мережі, зокрема через інтернет, з використанням спеціальних протоколів (наприклад, SIP, RTP).

Wi-Fi – Wireless Fidelity – технологія бездротової передачі даних на основі стандарту IEEE 802.11.

ВСТУП

Сучасні IoT-системи стрімко розвиваються, охоплюючи сфери логістики, аграрного сектору, транспорту, безпеки та інших галузей, де важливо збирати й оперативно передавати дані з фізичних об’єктів. Ключовою характеристикою таких систем є здатність функціонувати в реальному часі та незалежно від наявності стабільного інтернет-з’єднання.

Одним із найпростіших способів реалізації подібних систем є використання мікрокомп’ютерів на кшталт Raspberry Pi у поєднанні з зовнішніми модулями, які виконують окремі функції – зокрема, збирання геолокаційних даних або забезпечення мобільного зв’язку. Такі модулі, як SIM7000E, можуть працювати одночасно в кількох режимах — визначати координати через GPS, передавати дані через стільникову мережу LTE Cat-M1 або NB-IoT, а також підтримувати базову логіку управління через AT-команди.

Актуальність проблеми полягає в необхідності забезпечення надійної передачі даних у ситуаціях, коли традиційне інтернет-з’єднання недоступне, а зібрані дані потрібно все одно доставити до кінцевого користувача. Особливо це стосується мобільних, віддалених або тимчасових об'єктів — наприклад, транспортних засобів, польових станцій або автономних датчиків у сільському господарстві.

В умовах таких обмежень зростає значення адаптивних шлюзів — пристроїв, здатних самостійно визначати наявність з’єднання з мережею та змінювати логіку передачі даних відповідно до поточної ситуації. Одним з можливих рішень є розробка IoT-шлюзу, який у разі доступу до Wi-Fi надсилає координати через HTTPS-запити безпосередньо в Telegram-бот, а у разі його відсутності — ініціює TCP-з’єднання через стільникову мережу і пересилає дані на віддалений сервер для подальшої обробки.

Завдяки використанню відкритих протоколів, простих у реалізації мов програмування та публічного Telegram Bot API, така система може бути

впроваджена з мінімальними витратами, забезпечуючи при цьому високу надійність, масштабованість і автономність.

Таким чином, ця дипломна робота присвячена проєктуванню, реалізації та дослідженню IoT-шлюзу, що працює на базі Raspberry Pi з модулем SIM7000E, з можливістю автоматичного перемикання між каналами передачі даних залежно від стану підключення до мережі.

1 ТЕОРЕТИЧНІ ОСНОВИ IOT-СИСТЕМ ТА БЕЗДРОТОВИХ ТЕХНОЛОГІЙ ЗВ’ЯЗКУ

1.1 Основи Інтернету речей (IoT)

1.1.1 Поняття та архітектура IoT

Інтернет речей (IoT, Internet of Things) — це концепція, згідно з якою фізичні пристрої, що мають можливість обчислення, збирання, обміну та обробки даних, підключаються до мережі (переважно інтернету) з метою автоматизованого управління або передачі інформації в реальному часі.

На відміну від традиційних IT-систем, IoT включає величезну кількість децентралізованих пристроїв, які можуть функціонувати незалежно, збирати дані з навколишнього середовища та реагувати на події без безпосередньої участі людини.

Основні компоненти IoT-системи:

- Пристрої збору даних (датчики та актуатори)
    - Датчики вимірюють фізичні параметри: температуру, вологість, світло, тиск, положення тощо.
    - Актуатори(виконавчі механізми), навпаки, впливають на середовище (вмикають/вимикають пристрої, регулюють двигуни тощо).
- IoT-шлюз (gateway)
    - Збирає дані з сенсорів та передає їх далі - в хмару, локальний сервер чи інший вузол.
    - Проводить попередню обробку даних, фільтрацію, кешування.
    - Забезпечує перехід між протоколами (наприклад, UART → MQTT/HTTP).
- Мережеві інтерфейси (LTE, NB-IoT, Wi-Fi, LoRa, ZigBee тощо)
    - Служать для передачі даних на віддалені сервери.

    - Вибір залежить від обмежень у енергоспоживанні, дальності, пропускної здатності.
- Хмарна або серверна інфраструктура (cloud/backend)
    - Отримує дані з шлюзів, зберігає їх, обробляє, візуалізує.
    - Забезпечує аналітику, управління, прийняття рішень (наприклад, надсилання команд назад на пристрої).
- Кінцеві користувачі (user interface, mobile/web apps)
    - Отримують доступ до даних, історії, аналітики.
    - Мають змогу керувати пристроями віддалено.

У рамках дипломної роботи реалізована модель IoT-шлюзу, яка включає наступні компоненти:

Сенсорна частина: GPS-модуль, підключений до SIM7000E, здійснює визначення геолокації.

Комунікаційний модуль: SIM7000E підключений до Raspberry Pi через UART-інтерфейс. У випадку відсутності з'єднання з інтернетом, Raspberry Pi надсилає AT-команди модему для ініціації TCP-з’єднання безпосередньо з віддаленим сервером. Координати передаються на хмарний сервер Oracle Cloud за допомогою TCP-запитів, які SIM7000E надсилає напряму через стільникову мережу.

IoT-шлюз: Raspberry Pi 3 Model B обробляє отримані координати та, за наявності з’єднання з Wi-Fi, передає їх через HTTPS-запити до Telegram-бота. У разі відсутності доступу до Wi-Fi, система переходить в офлайн-режим: координати надсилаються напряму з SIM7000E до сервера через TCP, або тимчасово зберігаються локально у вигляді логів.

Канал передачі: залежно від стану підключення шлюз обирає відповідний канал: HTTPS-з’єднання через Wi-Fi (за наявності інтернету) або TCP-з’єднання через SIM7000E (у разі його відсутності).

Хмарне сховище: на сервері Oracle Cloud постійно працює служба TCP Listener, яка приймає координати з пристрою. Після обробки, дані надсилаються в Telegram через API-інтерфейс бота. Telegram-бот виступає в

ролі централізованого сховища, приймаючи повідомлення з локацією й координатами та зберігаючи журнал у чаті.

1.1.2 Класифікація пристроїв IoT за функціональністю

Пристрої, що входять до складу систем Інтернету речей (IoT), можна умовно розподілити за їх функціональним призначенням. Основними категоріями виступають сенсорні пристрої, виконавчі механізми (актуатори), контролери та шлюзи. Усі ці компоненти відіграють ключову роль у формуванні повного циклу збору, обробки та використання даних у системах IoT.

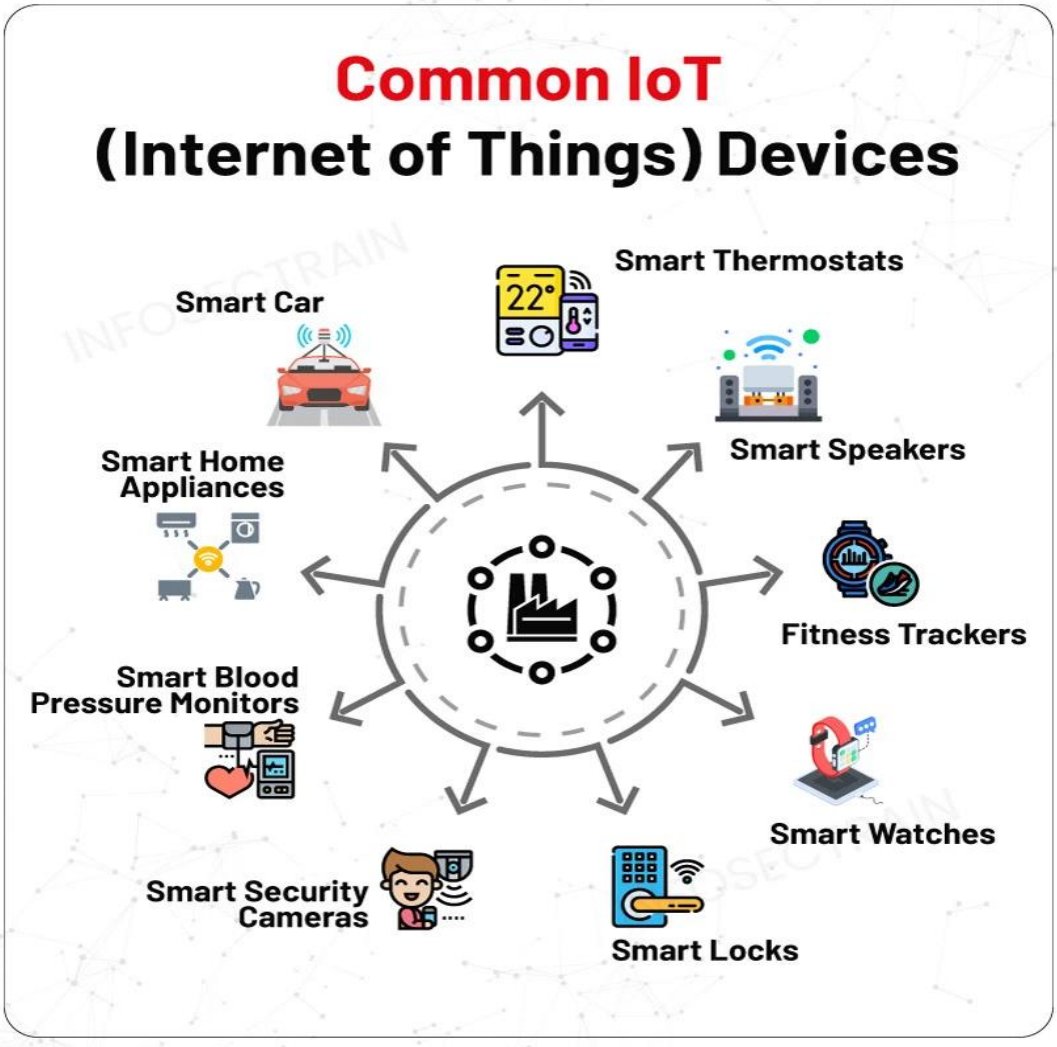

Рисунок 1.1 - Приклади популярних IoT пристроїв

Першу категорію становлять сенсорні пристрої. Їх головна функція полягає у фіксації змін у навколишньому середовищі або фізичних параметрів об’єкта. Наприклад, сенсори можуть вимірювати температуру, рівень вологості, тиск, рух, положення або навіть вібрації. У випадку реалізації даного проекту використовується GPS-модуль, який виступає у ролі сенсорного пристрою. Його основним завданням є фіксація координат місцезнаходження у форматі географічної широти та довготи.

Наступною функціональною ланкою є актуатори — пристрої, які виконують певну дію у відповідь на отриману команду або зміну параметрів середовища. На відміну від сенсорів, які лише збирають інформацію, актуатори здатні безпосередньо впливати на об’єкти: запускати механізми, вмикати або вимикати освітлення, змінювати положення сервоприводів тощо. Хоча в розглянутому проєкті актуатори не використовуються, у багатьох інших застосуваннях IoT вони є критично важливими.

Контролери виконують роль обчислювального центру в IoT-системі. Саме на них відбувається попередня обробка даних, реалізація логіки прийняття рішень та управління іншими модулями. У більшості проектів роль контролера виконує мікроконтролер або одноплатний комп’ютер. У нашому випадку цю функцію реалізує мікрокомп’ютер Raspberry Pi, який приймає дані з SIM7000E, обробляє їх та приймає рішення про подальшу передачу інформації в хмару.

Шлюз у контексті IoT-систем — це пристрій, який забезпечує комунікацію між внутрішньою мережею пристроїв та зовнішнім світом, зазвичай через інтернет. Він може виступати як простим передавачем даних, так і повноцінним вузлом з фільтрацією, шифруванням або кешуванням інформації. У даному проекти Raspberry Pi одночасно виконує функції контролера та шлюзу: він отримує GPS-координати з SIM7000E, формує повідомлення та надсилає його в чат Telegram-бота через Wi-Fi-з'єднання або, за відсутності з’єднання, надсилає команду з повідомленням до модуля

SIM7000E, щоб передати інформацію стільниковою мережею через протокол TCP.

Таким чином, IoT-система зазвичай включає ці чотири класи пристроїв, кожен з яких виконує унікальну функцію. Їх поєднання забезпечує повний життєвий цикл даних: від збору і обробки до прийняття рішень і виконання дій.

1.1.3 Сценарії застосування IoT у різних галузях

Інтернет речей (Internet of Things, IoT) дедалі активніше інтегрується у різноманітні сфери людської діяльності, створюючи передумови для переходу до повністю автоматизованих і взаємопов’язаних систем. Основною перевагою впровадження IoT є можливість збору, обробки та аналізу великої кількості даних у реальному часі з подальшим використанням цих даних для прийняття рішень або автоматичного управління процесами. Нижче розглянуто найпоширеніші галузі, в яких такі рішення вже набули широкого застосування.

Однією з найбільш популярних сфер є інтелектуальні системи керування в житлових приміщеннях — так звані розумні будинки. Завдяки вбудованим сенсорам, бездротовим мережам та програмованим контролерам, мешканці можуть дистанційно керувати освітленням, опаленням, вентиляцією, системами безпеки та побутовою технікою. Крім того, сучасні системи можуть автоматично реагувати на зміну умов середовища, наприклад, знижувати температуру в приміщенні при відсутності людей або вмикати сигналізацію у разі виявлення диму чи витоку газу.

У сфері логістики та транспорту IoT забезпечує підвищення прозорості та ефективності перевезень. GPS-модулі, інтегровані з мікроконтролерами, дозволяють у реальному часі відстежувати місцезнаходження транспортних засобів і вантажів. Водночас сенсори стану допомагають контролювати умови перевезення, зокрема температуру та вологість, що особливо актуально при

транспортуванні харчових продуктів, медикаментів або інших чутливих товарів. Аналітичні системи, підключені до IoT-пристроїв, можуть автоматично оптимізувати маршрути, прогнозувати зношування техніки та планувати технічне обслуговування.

Значні переваги IoT також демонструє у медичній галузі. Завдяки носимим пристроям, які відстежують життєво важливі показники пацієнтів (частоту серцевих скорочень, рівень глюкози, тиск тощо), лікарі отримують змогу дистанційно контролювати стан здоров’я пацієнтів у реальному часі. Такі системи особливо цінні для пацієнтів з хронічними захворюваннями або обмеженою рухливістю. Окрім того, IoT пристрої у клініках дозволяють оптимізувати облік медичного обладнання, покращувати внутрішньолікарняну логістику та забезпечувати безперервний моніторинг критичних станів.

Ще одним перспективним напрямком застосування IoT є сільське господарство. Тут Інтернет речей сприяє реалізації концепції "точного землеробства", яка базується на постійному зборі даних із полів та ферм. Датчики вологості ґрунту, температури, рівня освітленості та інших параметрів дозволяють оптимізувати процеси зрошення, добрива та захисту рослин. Завдяки цьому фермери можуть зменшити витрати ресурсів, підвищити врожайність та мінімізувати екологічний вплив. У тваринництві IoT використовується для моніторингу стану тварин, їхньої поведінки та місцезнаходження, що полегшує контроль великих поголів’їв.

У підсумку, впровадження IoT-рішень у різних секторах економіки дозволяє значно підвищити ефективність, безпеку та якість процесів, а також забезпечити гнучкість і масштабованість сучасних систем керування.

1.2 Принципи побудови IoT-шлюзів

1.2.1 Роль шлюзу в IoT-інфраструктурі

У сучасній архітектурі Інтернету речей шлюз (gateway) виступає ключовою складовою, що забезпечує взаємодію між сенсорними пристроями та хмарними або локальними обчислювальними системами. Основна мета IoT-шлюзу полягає у з’єднанні низькорівневих пристроїв, які часто працюють із нестандартними протоколами передачі даних, із високорівневими сервісами, що використовують стандартні протоколи мережевої взаємодії (наприклад, HTTPS, MQTT, TCP/IP).

Першочерговою функцією шлюзу є збір даних від сенсорів або модулів зв’язку. У контексті цієї дипломної роботи, мікроконтролер Raspberry Pi, взаємодіючи з модулем SIM7000E, регулярно ініціює зчитування геолокаційної інформації через AT-команди. Зібрані координати надходять у вигляді необроблених повідомлень через інтерфейс UART, і далі перетворюються у структурований формат для подальшого використання.

Другою важливою функцією шлюзу є обробка та попередній аналіз даних. Зазвичай ця обробка включає фільтрацію шумів, нормалізацію значень, перевірку на коректність або виявлення відхилень. Хоча в рамках запропонованої реалізації обробка є мінімальною (обмежується переведенням координат у зручний для відображення формат), система може бути легко розширена до повноцінного аналізу руху чи геозон.

Третій компонент функціональності — агрегація даних, тобто накопичення кількох повідомлень у проміжку часу з метою їх одноразової відправки. Це дозволяє зменшити навантаження на мережу передачі даних, а також забезпечити більшу надійність при нестабільному з’єднанні. У досліджуваній системі передбачено накопичення координат у вигляді логів, які при наявності з’єднання з мережею Wi-Fi передаються через зашифрований HTTPS-запит до Telegram-бота.

Нарешті, шлюз реалізує функцію передачі даних. Він транслює зібрану та оброблену інформацію до віддаленого сервера, бази даних або сервісу в інтернеті. Raspberry Pi у цьому випадку виконує роль хмарного клієнта, який за розкладом (кожну годину) ініціює з’єднання з Telegram-ботом, передаючи актуальну геолокацію через API. Такий підхід забезпечує автономність пристрою та мінімізацію витрат енергії і трафіку.

Таким чином, шлюз в IoT-інфраструктурі виступає універсальним адаптером, який не лише забезпечує зв'язок між різнорівневими компонентами, але й оптимізує передачу даних, підвищуючи загальну ефективність і надійність системи.

1.2.2 Функціональні компоненти шлюзу

IoT-шлюз складається з ряду апаратних і програмних компонентів, що спільно забезпечують ефективну комунікацію між пристроями збору даних і мережею Інтернет. Успішна реалізація шлюзу залежить від гармонійної інтеграції кожного з цих компонентів, що виконує специфічну роль у процесі обробки, маршрутизації та трансляції інформації.

1. Обчислювальний модуль (MCU / SoC)

Центральним компонентом IoT-шлюзу є мікроконтролер (MCU) або система-на-кристалі (SoC), яка виконує роль логічного ядра. Цей елемент відповідає за обробку вхідних даних, керування периферією та запуск вбудованого програмного забезпечення. В умовах систем з високою продуктивністю або складною логікою перевагу надають платформам на базі SoC, наприклад, Raspberry Pi. Такі платформи дозволяють використовувати повноцінну операційну систему, що значно розширює можливості для обробки, шифрування та передачі даних.

2. Модем або модуль зв’язку

Для забезпечення віддаленої передачі даних шлюзи оснащуються модулями бездротового зв’язку. Це можуть бути LTE/NB-IoT модеми, Wi-Fi,

LoRa або ZigBee. У рамках даного проекту використовується модуль SIM7000E, який підтримує мобільні технології зв’язку з низьким енергоспоживанням (NB-IoT, LTE Cat-M1), а також включає GPS-приймач для отримання координат. Цей модем взаємодіє з Raspberry Pi через USB-інтерфейс та керується набором AT-команд. Керування здійснюється через UART-інтерфейс, реалізований через USB-з'єднання.

3. Інтерфейси підключення

Фізична комунікація між компонентами шлюзу реалізується за допомогою стандартних інтерфейсів, зокрема UART, USB, I2C та SPI. У розробленій системі основну роль відіграє інтерфейс UART, що використовується для обміну AT-командами між Raspberry Pi та модемом SIM7000E. Передача даних здійснюється через USB-кабель, який одночасно забезпечує і живлення, і UART-комунікацію, завдяки вбудованому USB–UART перетворювачу на платі SIM7000E. Raspberry Pi, у свою чергу, живиться через окремий micro-USB-порт. Окрім цього, Raspberry Pi має розширену систему GPIO-виводів, що дозволяє підключати додаткові сенсори, реле, модулі зберігання або інші периферійні пристрої, розширюючи функціональні можливості шлюзу.

4. Програмне забезпечення шлюзу

Функціонування шлюзу залежить від коректної роботи програмного забезпечення, яке реалізує логіку взаємодії з апаратною частиною. Воно може бути представлено як низькорівневим скриптом, так і повноцінною прошивкою або сервісом. У запропонованій реалізації шлюзу використовується Python-скрипт, який перевіряє наявність інтернет-з’єднання через Wi-Fi та, у разі доступності мережі, ініціює зчитування координат через AT-команди та передає зібрані координати через HTTPS-запит до Telegram-бота. Якщо ж з’єднання з мережею відсутнє, то скрипт ініціює передачу даних через стільникову мережу, надсилаючи AT-команди до SIM7000E.

Шлюз IoT виступає як багаторівнева система, що поєднує в собі обчислювальні можливості (SoC), засоби мобільного зв’язку (модем LTE/NB-

IoT), а також функціональне програмне забезпечення, яке реалізує повний цикл роботи: від зчитування “сирих” даних до передачі їх у мережу Інтернет. У рамках дипломного проекту реалізовано шлюз на основі Raspberry Pi 3 Model B та модуля SIM7000E, з’єднаних через microUSB. За допомогою Python-скриптів реалізовано модулі збору, локального зберігання та передачі координат через захищені HTTPS або TCP канали до Telegram-бота, що слугує "хмарою" для візуалізації та збереження історії переміщень. Такий підхід демонструє можливість побудови функціонального та адаптивного шлюзу без використання складних або дорогих промислових систем, що є актуальним для багатьох IoT-сценаріїв, включаючи моніторинг транспорту, сільське господарство або персональне трекінг-рішення.

1.2.3 Топології та моделі комунікації

У системах IoT можуть застосовуватись різні топології комунікації в залежності від кількості пристроїв, завдань та вимог до надійності й масштабованості. Найпоширенішими є топології «зірка», «сітка» (mesh) та одноранговa (P2P).

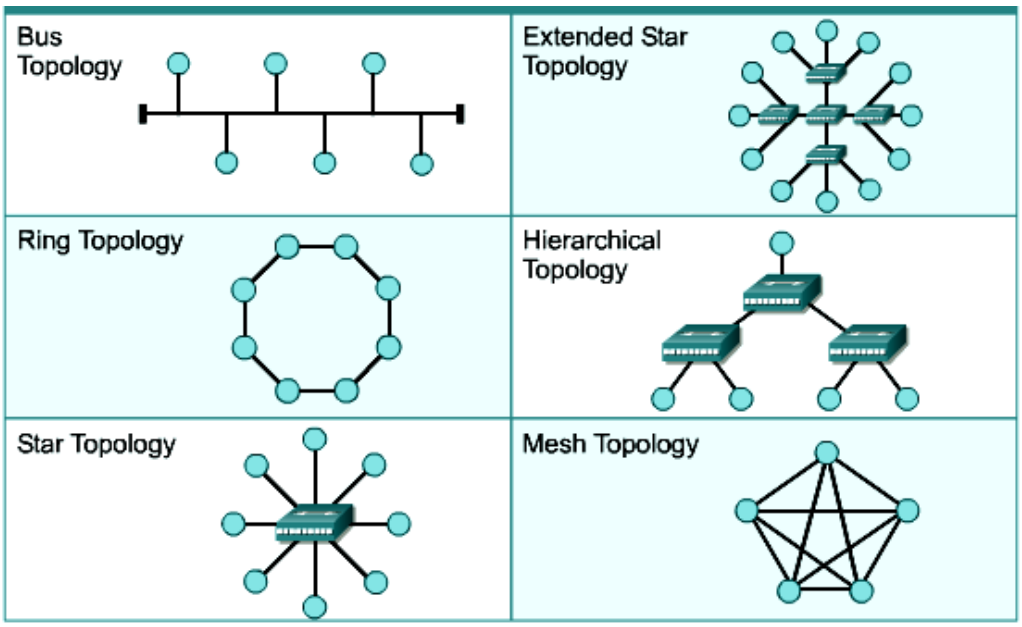

Рисунок 1.2 - Схематичне зображення найпоширеніших топологій

У топології «зірка» всі периферійні пристрої підключаються до центрального вузла (шлюзу або хаба), який виступає в ролі комунікаційного посередника. Така модель характерна для систем із низьким енергоспоживанням, де периферія надсилає дані лише за потреби. Сітка (mesh) забезпечує багатосторонній зв’язок між вузлами, що підвищує надійність та дозволяє маршрутизувати пакети у разі виходу окремих елементів з ладу. У однорангових (P2P) мережах кожен вузол може одночасно бути як клієнтом, так і сервером, що забезпечує гнучкість, але ускладнює керування.

На рівні програмної моделі часто використовуються клієнт-серверні архітектури. У цій моделі IoT-пристрій або шлюз виступає клієнтом, що надсилає дані до сервера чи хмарного застосунку. Одним з найпоширеніших протоколів у таких системах є HTTP (HyperText Transfer Protocol), який забезпечує обмін структурованими повідомленнями між клієнтом та сервером. У безпечніших реалізаціях використовується HTTPS, що базується на TLS-шифруванні, дозволяючи захищати дані від перехоплення. Це особливо важливо при передачі конфіденційної інформації, зокрема геолокації.

Іншим популярним протоколом у середовищі IoT є MQTT (Message Queuing Telemetry Transport) — легкий брокер-орієнтований протокол, призначений для пристроїв з обмеженими ресурсами та нестабільним зв’язком. MQTT реалізує модель «видавець–підписник», де дані передаються через центральний брокер, що значно знижує навантаження на клієнти.

Ще одним важливим протоколом у контексті IoT є TCP (Transmission Control Protocol) — протокол транспортного рівня, який забезпечує надійне встановлення з'єднання між пристроями та доставку даних у правильному порядку. На відміну від більш легковагових протоколів, TCP гарантує коректність передачі, що робить його доцільним для критичних систем, де втрата інформації неприпустима. У межах даного проєкту TCP використовується як альтернативний канал передачі координат з SIM7000E до

віддаленого сервера через стільникову мережу, коли недоступне основне Wi-Fi-з'єднання.

У розробленій системі застосовується топологія типу «зірка», де SIM7000E виступає як периферійний пристрій, що збирає GPS-дані, а Raspberry Pi — як центральний вузол (шлюз), який керує логікою передачі. У нормальному режимі система працює за клієнт-серверною моделлю, надсилаючи координати через HTTPS-запити з Raspberry Pi до Telegram-бота. У разі втрати підключення до Wi-Fi, система переходить у офлайн-режим: Raspberry Pi ініціює TCP-з'єднання через модем SIM7000E і передає координати безпосередньо на віддалений сервер, де працює постійний TCP Listener. Таким чином, дані завжди можуть бути доставлені, навіть без доступу до традиційного інтернет-з’єднання на шлюзі, що підвищує загальну надійність системи.

1.3 Технології передачі даних в IoT

1.3.1 Порівняння бездротових технологій

У контексті Інтернету речей вибір бездротової технології має критичне значення для ефективності, енергоспоживання, радіусу дії, вартості впровадження та сумісності пристроїв. Існує декілька популярних технологій зв’язку, що найчастіше застосовуються в IoT-системах: Wi-Fi, Zigbee, LoRa, NB-IoT та LTE Cat-M1. Нижче наведено порівняльний аналіз їхніх ключових характеристик.

Таблиця 1.1 - Характеристики технологій зв’язку

Характеристика	*Wi-Fi*	*Zigbee*	*LoRa*	*NB-IoT*	*LTE Cat-M1*
Смуга частот	2.4/5 ГГц	2.4 ГГц	<1 ГГц (залежить від регіону)	<1 ГГц (часто 800–900 МГц)	<1 ГГц (LTE)
Швидкість передачі	до 600 Мбіт/с	до 250 кбіт/с	до 50 кбіт/с	до 250 кбіт/с	до 1 Мбіт/с
Радіус дії	до 100 м у приміщенні	до 100 м	до 15 км	до 10 км	до 10 км
Споживання енергії	Високе	Низьке	Дуже низьке	Дуже низьке	Низьке
Тип мережі	WLAN (інфраструктура)	Mesh(Сітка)	Star (через шлюз)	Star (через базову станцію)	Star (через базову станцію)
Переваги	Висока пропускна здатність	Низьке споживання, mesh-підтримка	Великий радіус, автономність	Глибоке покриття, енергоефективність	Добре підходить для мобільних IoT
Недоліки	Високе споживання енергії	Обмежена швидкість і дальність	Мала швидкість, затримки	Висока затримка, залежність від операторів	Залежність від операторів
Підтримка мобільності	Слабка	Ні	Ні	Часткова	Так
Тип ліцензії	Неліцензований	Неліцензований	Неліцензований	Ліцензований (операторський)	Ліцензований (операторський)

Аналіз і застосування

Wi-Fi найчастіше використовується в системах з прямим доступом до локальної мережі та джерела живлення (наприклад, розумний дім,

відеоспостереження). Через високе енергоспоживання не підходить для автономних сенсорів.

Zigbee підходить для домашніх і офісних систем автоматизації, де важлива мережева топологія mesh і енергозбереження.

LoRa ідеально підходить для великих розподілених систем: сільське господарство, моніторинг навколишнього середовища, розумні міста. Працює без операторів зв’язку, але з низькою швидкістю.

NB-IoT та LTE Cat-M1 працюють у ліцензованому спектрі, потребують операторської інфраструктури, однак мають високу надійність, покриття в підвальних приміщеннях, підтримку QoS (quality of service) та можливість масштабування.

У нашому проєкті для передачі даних використовується гібридний підхід: за наявності підключення до Wi-Fi шлюз надсилає координати GPS через захищене HTTPS-з’єднання напряму до Telegram-бота. Якщо ж інтернет-з’єднання на Raspberry Pi недоступне, система переходить в офлайн-режим: модуль SIM7000E використовує стільникову мережу (LTE Cat-M1 або NB-IoT, залежно від наявності покриття) для передачі координат через TCP-запити до віддаленого хмарного сервера. На сервері постійно працює служба TCP Listener, яка приймає дані, обробляє їх та перенаправляє до Telegram, що виступає кінцевою точкою системи. Такий підхід дозволяє підтримувати зв’язок навіть у складних умовах покриття, гарантуючи надійність та автономність IoT-рішення.

1.3.2 Особливості NB-IoT та LTE для IoT

У світі IoT, де мільйони пристроїв повинні бути одночасно з’єднані, але при цьому залишатися максимально автономними, особливу увагу привертають технології NB-IoT (Narrowband IoT) та LTE Cat-M1. Обидві ці мережеві архітектури були спеціально розроблені як частина розвитку стандартів LTE, щоби забезпечити зв’язок для пристроїв із низьким

енергоспоживанням, довготривалим функціонуванням і здатністю працювати в умовах складного покриття.

NB-IoT вирізняється насамперед своєю здатністю функціонувати надзвичайно економно. Пристрої, що використовують цю технологію, можуть працювати від батареї до десяти років, що робить її ідеальною для сенсорів, розміщених у важкодоступних місцях, де часта заміна джерела живлення неможлива або економічно недоцільна. Завдяки вузькій смузі частот і оптимізованому протоколу, NB-IoT забезпечує стабільне з'єднання навіть у місцях із поганим покриттям, наприклад, у підвалах чи за межами міських зон. Щоправда, за це доводиться платити низькою швидкістю передачі даних і вищими затримками, що унеможливлює її використання у завданнях, де важлива оперативність або обробка великих обсягів інформації.

На противагу цьому, LTE Cat-M1 являє собою більш “швидкий” варіант енергоощадного зв’язку. Його пропускна здатність дозволяє передавати значно більші обсяги даних — до 1 Мбіт/с, що відкриває шлях до використання таких протоколів, як VoIP чи відеоспостереження з низькою роздільною здатністю. Завдяки зниженій затримці LTE Cat-M1 підходить для сценаріїв, де важлива оперативна передача інформації, як-от в індустріальних застосуваннях або у мобільних пристроях. Водночас енергоспоживання залишається досить низьким, щоб забезпечити багаторічну роботу на одному заряді.

Підсумуємо:

- NB-IoT:
    - Енергоспоживання: Дуже низьке, що дозволяє пристроям працювати на батареї до 10 років.
    - Радіус дії: Великий, забезпечує покриття в складних умовах, таких як підвали або віддалені райони.
    - Швидкість передачі даних: Низька, до 250 кбіт/с, підходить для передачі невеликих обсягів даних.

    - Затримка: Вища, ніж у LTE Cat-M1, що може бути критичним для деяких застосувань.
- LTE Cat-M1:
    - Енергоспоживання: Низьке, але вище, ніж у NB-IoT.
    - Радіус дії: Менший, ніж у NB-IoT, але достатній для більшості міських застосувань.
    - Швидкість передачі даних: Вища, до 1 Мбіт/с, дозволяє передавати більші обсяги даних.
    - Затримка: Низька, що робить її придатною для застосувань, чутливих до затримок.

У нашому проєкті модуль SIM7000E технічно підтримує обидві згадані технології — NB-IoT та LTE Cat-M1, однак їх використання в контексті передачі даних напряму в інтернет через HTTPS виявилось непрактичним. Через обмежену підтримку SSL/TLS та нестабільність роботи шифрованих запитів, було прийнято рішення передавати координати до Raspberry Pi через UART, а вже далі, залежно від доступності інтернету, шлюз самостійно вибирає канал: якщо Wi-Fi доступний — координати надсилаються до Telegram через HTTPS, якщо недоступний — Raspberry Pi ініціює TCP-передачу через стільникову мережу (LTE Cat-M1 або NB-IoT) із SIM7000E напряму до хмарного сервера. Такий підхід забезпечує стабільність і мінімізує ризики втрати даних.

Таким чином, запропонована архітектура дозволяє поєднати сильні сторони як мобільного модему, так і потужнішого обчислювального вузла. SIM7000E виступає як сенсорний пристрій збору геоданих із можливістю передачі координат через стільникову мережу TCP-запитами у випадках відсутності Wi-Fi, тоді як Raspberry Pi бере на себе логіку обробки, вибору каналу та передачі через HTTPS. Такий гібридний підхід підвищує гнучкість, надійність та адаптивність IoT-системи до умов роботи в середовищі з нестабільним інтернет-з’єднанням.

1.3.3 SIM7000E як інтерфейс зв’язку

Модуль SIM7000E — це компактне й енергоефективне рішення для реалізації бездротового зв’язку в IoT-проєктах, яке поєднує в собі підтримку декількох технологій передачі даних: NB-IoT, LTE Cat-M1 і GPRS/EDGE (для зворотної сумісності). Така універсальність робить його дуже привабливим для інженерів і розробників, що створюють пристрої з мінімальним енергоспоживанням і потребою в надійному мобільному зв’язку.

SIM7000E має компактний форм-фактор (24 мм × 24 мм) і оснащений необхідними інтерфейсами для інтеграції з більшістю мікроконтролерів та одноплатних комп’ютерів. Зокрема, він підтримує UART, I2C та USB, що дозволяє гнучко вибирати метод підключення залежно від апаратної конфігурації системи. UART — найбільш популярний варіант, адже через нього зручно передавати AT-команди, які є стандартним способом управління функціоналом модуля.

Говорячи про AT-команди, варто відзначити їхню роль як "мову спілкування" з модемом. За допомогою цих текстових команд можна виконувати повний спектр дій: підключатися до мережі, отримувати GPS-координати, надсилати HTTP-запити, налаштовувати параметри зв’язку або навіть зчитувати рівень сигналу. Наприклад, AT+CGATT? дозволяє перевірити, чи підключено модем до мережі, а AT+CGNSINF повертає повну інформацію про GPS-фіксацію, включно з широтою, довготою, точністю та статусом супутників. Команди прості, але роблять багато, і саме це дозволяє гнучко контролювати модуль навіть без складного програмного забезпечення.

Поза теорією — у практиці. У нашому проєкті SIM7000E використовується переважно як GPS-приймач, однак також виконує функцію передачі координат на хмарний сервер у випадку втрати інтернет-з'єднання на шлюзі. Через відсутність вбудованого Wi-Fi модуля, модем використовує стільникову мережу (LTE Cat-M1 або NB-IoT) для передачі даних за допомогою TCP-протоколу.

Передача координат ініціюється з боку Raspberry Pi, який надсилає відповідні AT-команди до SIM7000E. Сам модуль, у свою чергу, встановлює TCP-з'єднання та надсилає координати на віддалений сервер. Така модель дозволяє реалізувати автономну передачу даних без Wi-Fi, використовуючи лише стільниковий канал.

Таким чином, SIM7000E у цій архітектурі виступає не лише як GPS-сенсор, а й як модем для передачі координат через TCP, під повним контролем обчислювального блоку. Його надійна робота з AT-командами, підтримка стільникових мереж і універсальність робить його ефективним елементом для побудови гнучких IoT-рішень із резервними сценаріями обміну даними.

1.4 Висновки до розділу 1

1. Розроблена IoT-система базується на архітектурі клієнт-сервер з гнучким вибором каналу передачі даних. Основними компонентами виступають одноплатний комп’ютер Raspberry Pi, що виконує роль центрального вузла, модем SIM7000E з вбудованим GPS-приймачем, а також хмарний сервер, який приймає дані. Така структура дозволяє адаптивно обробляти і передавати координати залежно від доступності інтернет-з'єднання.

2. У реалізованій системі Raspberry Pi виступає як центральний обчислювальний вузол, який керує логікою роботи IoT-шлюзу, обробляє GPS-координати та обирає відповідний канал передачі даних. У випадку доступності Wi-Fi, дані передаються до Telegram-бота через HTTPS-запити. Якщо ж Wi-Fi відсутній, Raspberry Pi ініціює TCP-з’єднання з допомогою SIM7000E, і координати передаються через стільникову мережу до хмарного сервера для подальшого опрацювання.

3.Передача координат у режимі відсутності інтернету здійснюється за допомогою TCP-протоколу через мобільну мережу LTE Cat-M1 або NB-IoT, залежно від покриття. В цьому випадку SIM7000E виконує функцію модему,

який, за AT-командою від Raspberry Pi, встановлює TCP-з’єднання з хмарним сервером. Сервер обробляє координати та перенаправляє їх у Telegram, який є кінцевою точкою прийому даних. Такий підхід забезпечує гнучкість і надійність системи в умовах обмеженого підключення до мережі.

2 ПРОЕКТУВАННЯ ІОТ-ШЛЮЗУ НА БАЗІ RASPBERRY PI ТА SIM7000E

2.1 Аналіз функціональних вимог

2.1.1 Визначення задач та сценаріїв використання

IoT-шлюз, побудований на базі Raspberry Pi та модуля SIM7000E, розроблений для вирішення конкретного набору задач у контексті моніторингу місцеположення об’єктів у режимі близькому до реального часу. Основна мета — створити надійну систему, здатну збирати геодані за допомогою GPS-приймача SIM7000E, обробляти їх на стороні Raspberry Pi та передавати до кінцевого користувача через хмарний сервіс — у нашому випадку, через Telegram-бота.

Система покликана працювати автономно та з мінімальним втручанням з боку користувача. Це дає змогу застосовувати її у транспортних засобах, мобільних об’єктах чи пристроях, що мають змінне розташування. Поточне використання передбачає, що Raspberry Pi встановлюється в авто і збирає координати за допомогою SIM7000E. Потім, через внутрішню логіку, система визначає, чи є підключення до Інтернету (через Wi-Fi). У випадку доступу — координати передаються у Telegram через HTTPS. Якщо ж з’єднання немає — координати надсилаються через стільникову мережу з SIM7000E до хмарного сервера, звідки обробляються та переправляються до Telegram.

Telegram було обрано як кінцевий інтерфейс взаємодії не випадково. Цей месенджер підтримує API, що дозволяє легко створювати ботів для обміну інформацією, надсилання повідомлень, координат та іншої корисної інформації. Це дозволяє користувачу отримувати оновлення прямо у своєму смартфоні без встановлення додаткових додатків.

У перспективі система може бути масштабована — з підтримкою додаткових сенсорів (наприклад, температури, вібрації чи рівня пального), що дозволить застосовувати її в аграрному секторі, логістиці або безпекових системах.

Таким чином, функціональні задачі шлюзу включають:

- отримання координат GPS за допомогою модуля SIM7000E;
- обробку координат на стороні Raspberry Pi;
- визначення доступності інтернет-з’єднання через Wi-Fi;
- автоматичний вибір каналу передачі: через HTTPS (при наявності Wi-Fi) або TCP (через стільникову мережу у разі його відсутності);

надсилання координат кінцевому користувачеві в Telegram — безпосередньо або через хмарний сервер.

Така реалізація дозволяє IoT-шлюзу працювати як в онлайн-режимі, так і в автономному офлайн-сценарії, забезпечуючи гнучкість та надійність передачі даних у польових умовах.

Сценарії використання можуть включати:

- трекінг транспорту;
- моніторинг пересувних IoT-пристроїв;
- використання в автономних або слабо покритих мережами місцях.

2.1.2 Вимоги до апаратної/програмної частини

Оскільки розроблювана система призначена для моніторингу пересувного об’єкта з періодичною передачею GPS-координат у хмарний сервіс (Telegram-бот), як апаратна, так і програмна частина IoT-шлюзу повинні відповідати ряду ключових вимог.

З боку апаратного забезпечення важливо забезпечити мінімальне енергоспоживання, особливо якщо система живиться від бортової мережі або акумулятора. Модем SIM7000E має підтримку енергозберігаючих режимів, зокрема PSM (Power Saving Mode) та eDRX, що дозволяє скоротити споживання до кількох міліамперів у режимі очікування. Raspberry Pi, в свою чергу, має помірне енергоспоживання, особливо при відключенні непотрібних інтерфейсів, що дозволяє інтегрувати його у вбудовані системи.

Raspberry Pi 3 Model B споживає:

В режимі очікування (idle): ~1,5–2,5 Вт (300–500 мА при 5В)

Під навантаженням (Wi-Fi, USB, CPU): до ~4 Вт (800 мА при 5В)

Це робить його більш енергоефективним. Тим не менш, при роботі від акумулятора потрібно враховувати пікове споживання, особливо якщо підключено додаткові модулі, як-от SIM7000E чи GPS. Для зменшення споживання енергії можна відключити непотрібні інтерфейси та оптимізувати роботу системи.

Щодо типів оброблюваних даних, основним джерелом інформації в системі є координати, отримані від GPS-модуля SIM7000E. Дані надходять у вигляді NMEA-повідомлень та обробляються на стороні Raspberry Pi для подальшої передачі або зберігання.

Алгоритм обробки передбачає визначення поточного стану інтернет-з’єднання. У разі наявності Wi-Fi, координати одразу надсилаються через HTTPS-запит до Telegram-бота. Якщо ж підключення до інтернету відсутнє, система активує офлайн-режим — координати або тимчасово зберігаються у локальний лог-файл, або Raspberry Pi надсилає AT-команди до SIM7000E для встановлення TCP-з’єднання з віддаленим сервером через стільникову мережу.

Сервер, у свою чергу, приймає координати, обробляє їх і перенаправляє до Telegram, який виступає кінцевою точкою доставки. Такий підхід дозволяє гнучко адаптуватися до умов нестабільного або повністю відсутнього доступу до інтернету.

Таким чином, загальні вимоги до системи включають: стабільність у роботі в мобільних умовах, підтримку мобільного зв’язку або Wi-Fi, обробку простих телеметричних даних (GPS), мінімальне енергоспоживання та здатність захищеної передачі даних у хмарну систему.

2.1.3 Загрози та особливості середовища експлуатації

У процесі проектування IoT-шлюзу необхідно враховувати зовнішні фактори, які можуть негативно вплинути на стабільність його роботи. Однією з ключових загроз є перебої в живленні. Оскільки пристрій працює в режимі постійного моніторингу, навіть короткочасне знеструмлення може призвести до втрати даних або порушення логіки передачі координат. Щоб уникнути цього, доцільно передбачити використання джерел безперебійного живлення (наприклад, акумуляторів або суперконденсаторів), які зможуть підтримувати живлення пристрою при збої електромережі.

Іншою важливою загрозою є нестабільний сигнал мобільної мережі, особливо в сільській місцевості, у зоні щільної забудови або в автомобілі, що рухається. Модулі типу SIM7000E підтримують NB-IoT та LTE Cat-M1, які спеціально розроблені для роботи в умовах слабкого сигналу, однак у реальних умовах сигнал може часто обриватися або погіршуватися, що потребує реалізації механізмів повторної передачі даних, буферизації або локального збереження логів на Raspberry Pi.

Окрім цього, варто враховувати температурні та вологісні умови експлуатації. Якщо пристрій встановлено в автомобілі, він піддається значним перепадам температур, які можуть негативно вплинути на електронні компоненти або призвести до збоїв у передачі даних. Використання захищеного корпусу та якісних роз’ємів допоможе уникнути окислення контактів та коротких замикань.

Усі ці фактори безпосередньо впливають на надійність системи, тому при проектуванні слід приділити особливу увагу механізмам самовідновлення (Наприклад автозапуск скриптів при перезавантаженні), локальному збереженню даних і обробці помилок при втраті з’єднання.

Для забезпечення безперебійної роботи шлюзу у випадку перезавантаження або відновлення живлення, реалізовано автоматичний запуск скриптів. У поточній реалізації системи використовується скрипт

sim_setup.py, який викликається автоматично при старті операційної системи. Цей скрипт виконує попередню конфігурацію модуля SIM7000E, у тому числі із затримкою, необхідною для стабільного завантаження всіх служб Raspberry Pi. Після цього запускається файл *check_internet.py*, який перевіряє наявність підключення до інтернету. В залежності від результату, система або зберігає координати локально (*store_location.py*), або надсилає їх через HTTPS до Telegram-бота (*send_location.py*). Такий підхід гарантує адаптивність і незалежність роботи шлюзу навіть за відсутності мобільного зв'язку.

2.2 Архітектура IoT-шлюзу

2.2.1 Вибір та опис компонентів

У даному проєкті IoT-шлюз реалізується з використанням мікрокомп'ютера Raspberry Pi 3 Model B та модуля SIM7000E. Така конфігурація дозволяє реалізувати всі необхідні функції системи: обробку даних, отримання координат із GPS, та безпечну передачу зібраної інформації до Telegram через Wi-Fi. Компоненти обрано відповідно до технічних вимог, доступності та простоти інтеграції.

Raspberry Pi 3 Model B виступає як основна керуюча одиниця системи. Пристрій оснащено 1 ГБ оперативної пам’яті, чотириядерним процесором ARM Cortex-A53 з частотою 1.2 ГГц, а також вбудованими Wi-Fi та Bluetooth модулями. У системі використовується карта пам’яті microSD об'ємом 16 ГБ із встановленою Raspberry Pi OS (32-bit), що є найкращим вибором для цієї моделі завдяки стабільній роботі та сумісності з доступними бібліотеками. При наявності знайомої мережі Wi-Fi, Raspberry Pi автоматично підключається до неї, забезпечуючи інтернет-з'єднання, необхідне для відправки зібраної інформації.

Серед переваг цієї моделі можна виділити низьке енергоспоживання, наявність GPIO для потенційного підключення додаткових сенсорів, USB-порти для розширення, та вбудований Wi-Fi. Недоліками є обмежені апаратні

ресурси (1 ГБ RAM), що може стати критичним у складніших задачах, та необхідність зовнішнього живлення (microUSB).

SIM7000E – це багатофункціональний модем, який підтримує роботу в стільникових мережах стандартів LTE Cat-M1, NB-IoT та GSM. У нашому проєкті він виконує подвійну роль: з одного боку, працює як GPS-приймач, що забезпечує отримання координат у форматі NMEA через UART-з'єднання; з іншого — слугує засобом зв’язку для передачі даних. За ініціативи Raspberry Pi модем встановлює TCP-з’єднання з хмарним сервером через стільникову мережу, що дозволяє передавати координати навіть за відсутності Wi-Fi. SIM7000E підтримує мережі NB-IoT, LTE CAT-M1, а також має GPS/GNSS-модуль, що дозволяє використовувати його в якості універсального інтерфейсу зв’язку в IoT-системах.

GPS-антена, яка використовується в системі, є активною. Така антена має вбудований підсилювач сигналу і потребує додаткового живлення, яке подається по тому ж коаксіальному кабелю, що й сигнал. Її перевагами є краща ефективність в умовах слабкого сигналу, наприклад, в міській забудові або в салоні автомобіля. Завдяки цьому забезпечується надійніше отримання координат навіть у складних умовах. Активна антена зазвичай трохи дорожча, але суттєво покращує якість прийому.

У цій реалізації додаткові сенсори не використовуються, однак їх підтримка можлива завдяки виводам GPIO на Raspberry Pi. Це дозволяє в майбутньому розширити функціональність IoT-шлюзу без суттєвої перебудови системи.

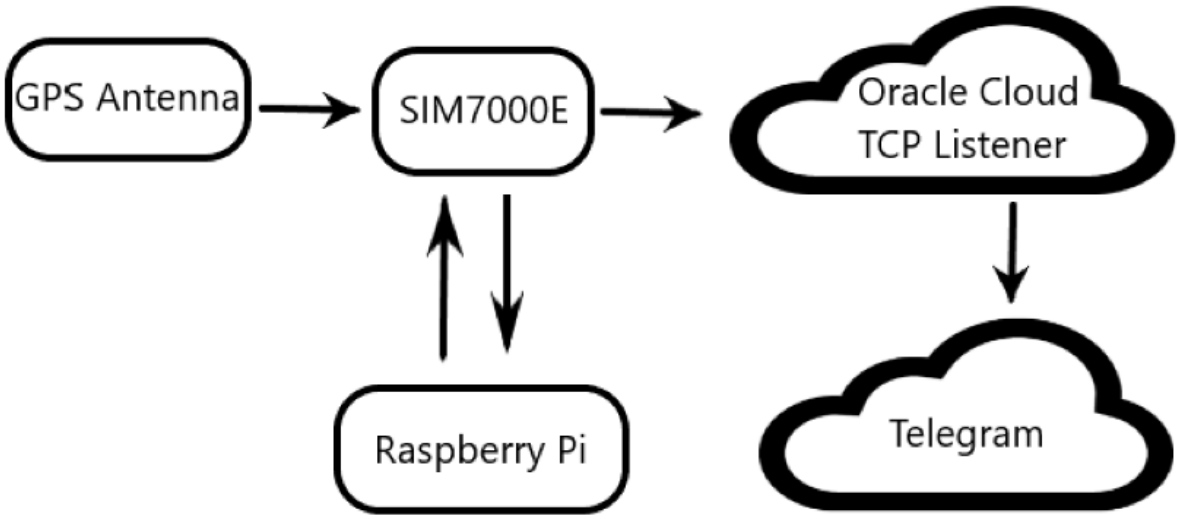

Рисунок 2.1 - Схема передачі даних при режимі «офлайн»

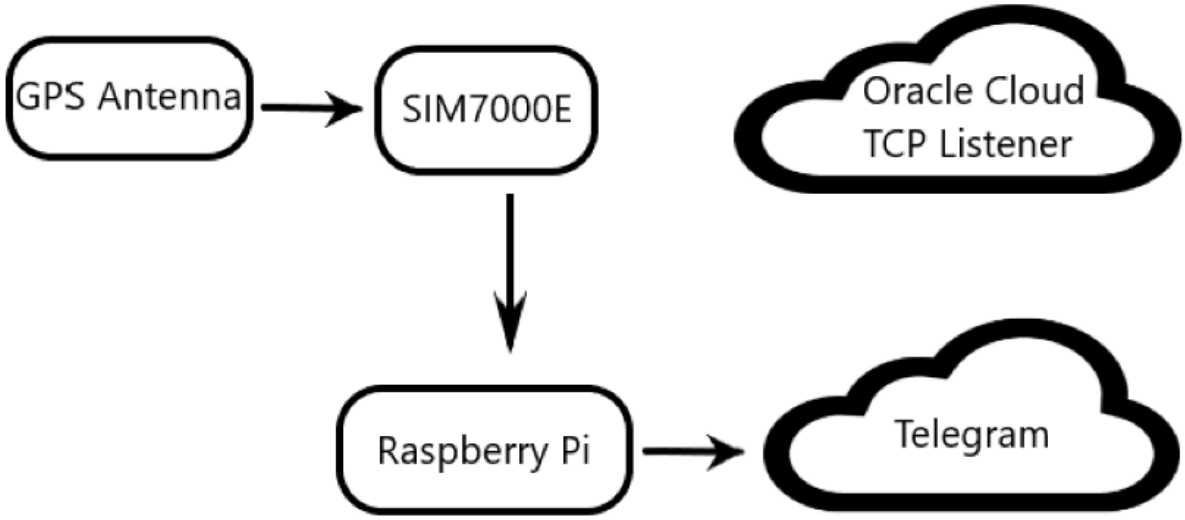

Рисунок 2.2 - Схема передачі даних при режимі «онлайн»

Завдяки такій архітектурі забезпечується гнучка та надійна робота шлюзу. Усі компоненти взаємодіють в межах єдиної системи, дозволяючи ефективно вирішувати поставлені завдання моніторингу та передачі даних.

Raspberry Pi виступає ядром логіки та обробки, тоді як SIM7000E забезпечує геолокацію, і в разі потреби може бути залучений до резервного каналу зв’язку через мобільні мережі

2.2.2 З’єднання та комунікація між модулями

Система IoT-шлюзу базується на взаємодії компонентів через інтерфейси UART, USB, GPIO та лінії живлення. Правильне з’єднання є критично важливим для стабільної роботи системи.

Модуль SIM7000E з’єднано з Raspberry Pi за допомогою microUSB-кабелю. Це дозволяє організувати передачу даних через емуляцію послідовного інтерфейсу (UART) у вигляді віртуального COM-порту. Завдяки цьому забезпечується стабільна комунікація між пристроями за допомогою AT-команд із боку Raspberry Pi.

Інтерфейс UART (послідовна передача даних) є стандартним способом взаємодії з SIM7000E. В альтернативних конфігураціях можливе використання GPIO-пінів TXD/RXD, але через microUSB-підключення ця необхідність відсутня.

GPIO-піни на Raspberry Pi не використовуються в поточній реалізації, однак потенційно можуть бути задіяні для розширення функціональності (підключення сенсорів, кнопок, LED-індикаторів тощо). Наявність GPIO надає системі гнучкість та дозволяє розширити її можливості без додаткових апаратних змін.

Живлення Raspberry Pi здійснюється через microUSB-порт із блока живлення на 5 В. SIM7000E отримує живлення від самого Raspberry Pi по USB-порту. GPS-антена, яка є активною, живиться безпосередньо від SIM7000E через коаксіальний кабель (антенний порт має подачу напруги 2.7–5 В).

Загальна схема живлення виглядає так:

1. Raspberry Pi отримує живлення з зовнішнього джерела 5 В;
2. SIM7000E живиться від Raspberry Pi через USB;
3. GPS-антена живиться через антенний вихід SIM7000E.

Така конфігурація мінімізує кількість зовнішніх з’єднань і дозволяє легко інтегрувати систему в обмежених умовах, наприклад, у транспортному засобі.

2.2.3 Сценарії взаємодії з сервером

В рамках реалізації IoT-шлюзу передача даних до хмарного середовища здійснюється за допомогою Telegram API.

Raspberry Pi збирає координати, отримані від SIM7000E (який використовується лише як GPS-приймач), обробляє їх у Python-скрипті, а далі здійснює HTTPS-запит до Telegram Bot API, передаючи повідомлення у встановлений чат (наприклад, приватні повідомлення або групу).

Чому саме HTTPS

Передача координат відбувається через захищений протокол HTTPS, що гарантує шифрування даних у процесі передачі. Це є критично важливим для запобігання перехопленню або спотворенню даних.

Можливість використання HTTPS напряму із SIM7000E є досить обмеженою: модуль вимагає складного конфігурування SSL-сертифікатів і часто не підтримує повноцінну роботу з TLS, що ускладнює реалізацію захищених запитів. Саме тому для надсилання з SIM7000E було обрано протокол TCP, що також є захищеним, проте, оскільки Telegram API не може отримувати TCP запити, з’явилась необхідність в додатковому хмарному сервері, що буде отримувати TCP запит, обробляти його, та пересилати по HTTPS до Telegram.

Telegram виступає у ролі хмарного отримувача даних. Завдяки відкритому API Telegram Bot, можливо легко реалізувати сценарії надсилання повідомлень. Це спрощує реалізацію системи моніторингу, дозволяючи миттєво отримувати оновлення з будь-якої точки світу.

2.3 Програмна реалізація

2.3.1 Протоколи обміну

У системах Інтернету речей (IoT) для обміну даними між пристроями та сервером або хмарною платформою використовуються різноманітні мережеві протоколи. Вибір протоколу безпосередньо впливає на надійність, швидкість, обсяг переданих даних і рівень безпеки.

Основні протоколи обміну, що застосовуються в IoT:

- HTTP (HyperText Transfer Protocol)

Тип: Клієнт-серверний, запит-відповідь

Переваги: Простий у реалізації, широко підтримується, зручний для REST API

Недоліки: Високе навантаження, відсутність шифрування (без HTTPS), не оптимізований для пристроїв з обмеженими ресурсами

Типова сфера застосування: IoT-системи, які періодично передають дані через API

Примітка: У цьому проекті HTTP не використовується напряму через SIM7000E, оскільки модуль не підтримує HTTPS належним чином, що робить передачу вразливою до перехоплення.

- HTTPS (HTTP Secure)

Тип: HTTP + SSL/TLS шифрування

Переваги: Захищена передача даних, підтримка сучасними веб сервісами

Недоліки: Більше ресурсів для обробки, особливо на малопотужних пристроях

Типова сфера застосування: Вебсервіси, API-запити, мобільні додатки

Використання в проекті: Raspberry Pi надсилає координати через HTTPS-запити до Telegram API, що забезпечує як безпечність, так і простоту реалізації завдяки бібліотекам на Python (наприклад, requests)

- MQTT (Message Queuing Telemetry Transport)

Тип: Публікація-підписка (Pub/Sub), поверх TCP

Переваги: Дуже легкий протокол, підходить для енергоефективних пристроїв, підтримує QoS (якість обслуговування)

Недоліки: Потребує наявності MQTT-брокера (наприклад, Mosquitto), складніший для реалізації в порівнянні з HTTPS

Типова сфера застосування: Сенсорні мережі, домашня автоматизація, Smart Home

- CoAP (Constrained Application Protocol)

Тип: REST-подібний протокол поверх UDP

Переваги: Дуже легкий, підтримує multicast, ідеальний для слабких пристроїв

Недоліки: Менша надійність у порівнянні з TCP-протоколами, обмеженість функціональності

Типова сфера застосування: Сенсорні мережі з жорсткими обмеженнями по енергії та обчисленням

- TCP (Transmission Control Protocol)

Тип: Орієнтований на з'єднання, надійна доставка

Переваги: Гарантована доставка, контроль цілісності, підтримка більшості мережевих протоколів

Недоліки: Повільніший, споживає більше трафіку через контрольні пакети

Типова сфера застосування: Дані, де критично важлива доставка (фінансові системи, API, бази даних)

- UDP (User Datagram Protocol)

Тип: Без з’єднання, передає дані пакетами без підтвердження

Переваги: Дуже швидкий, мінімальні затримки

Недоліки: Немає гарантії доставки, немає порядку

Типова сфера застосування: Стрімінг, онлайн-ігри, голосова передача

З урахуванням специфіки завдання — передача координат у вигляді повідомлення з мінімальним обсягом даних — та вимоги до захисту з'єднання,

оптимальним рішенням було обрати HTTPS через Telegram API та TCP, як резервний. Це дозволяє:

- уникнути складної реалізації серверної інфраструктури (замість MQTT-брокера або CoAP-серверу);
- забезпечити безпечну доставку (через TLS);
- зменшити час на розгортання і налагодження.

У той же час, MQTT або CoAP могли би бути кращими у сценаріях із великою кількістю пристроїв, що працюють у важких мережевих умовах або мають критичні обмеження по енергії. Якщо в подальшому система буде розширюватись, варто розглянути реалізацію MQTT-брокера для масштабованого збору даних або розподіленої обробки в локальній мережі.

2.3.2 Логіка обробки та надсилання даних

Принцип роботи розробленої IoT-системи базується на адаптивному виборі каналу передачі даних, залежно від доступності інтернет-з’єднання. Головним завданням є забезпечення доставки GPS-координат до кінцевого користувача в Telegram незалежно від наявності Wi-Fi-з'єднання на шлюзі.

Після ввімкнення пристрою на Raspberry Pi автоматично запускається головний скрипт sim_setup.py, який ініціалізує зв’язок із SIM7000E, активує GPS-функціонал та перевіряє доступність Wi-Fi. Залежно від результату, система переходить до одного з двох сценаріїв:

Онлайн-режим (Wi-Fi доступний):

Raspberry Pi отримує координати від SIM7000E, обробляє їх у Python-скрипті та передає до Telegram-бота через захищене HTTPS-з’єднання. Передача відбувається напряму з Raspberry Pi, що забезпечує шифрування даних та мінімальні затримки.

Офлайн-режим (Wi-Fi недоступний):

У разі втрати з’єднання, система активує резервний сценарій. Raspberry Pi формує координати, після чого надсилає AT-команди до SIM7000E для

ініціації TCP-з'єднання через стільникову мережу (LTE Cat-M1 або NB-IoT). SIM7000E встановлює TCP-зв’язок із віддаленим сервером, на якому працює служба TCP Listener. Сервер приймає координати, обробляє їх і пересилає у Telegram.

Система побудована з урахуванням періодичної перевірки статусу підключення, що дозволяє автоматично перемикатися між режимами в реальному часі. Це забезпечує стабільну та безперервну передачу геоданих незалежно від умов мережі.

Уся система поділяється на п’ять логічних блоків:

1.Налаштування SIM7000E при під’єднанні системи до живлення:

При старті системи Raspberry PI автоматично запускає скрипт sim_setup.py, що починає налаштування модулю SIM7000E, він використовує початкову затримку в 10 секунд, щоб дати час «провантажитись» усім системам, після чого надсилає AT-команди, що подають живлення на GPS-антену, розблоковують SIM-картку та знову очікують вже 20 секунд, щоб дати антені «прокинутись». Після виконання цих дій запускається наступний скрипт check_internet.py.

Таке наповнення має початковий скрипт(sim_setup.py):

```
import serial
import time
import subprocess

SERIAL_PORT = "/dev/ttyUSB3"
BAUDRATE = 115200
time.sleep(10)
def send_at_command(ser, command, delay=2):
    ser.write((command + '\r').encode())
    time.sleep(delay)
    response = ser.read_all().decode(errors='ignore')
    print(f"{command} -> {response}")
```

```
        return response
    try:
        with serial.Serial(SERIAL_PORT, BAUDRATE, timeout=2) as ser:
            send_at_command(ser, "AT+CGNSPWR=1")      # Power on GPS
            send_at_command(ser, "AT+CPIN=5776")
            time.sleep(20)  # Let GPS and network wake up
            send_at_command(ser, 'AT+CNACT=1,"internet"')
    except Exception as e:
        print("Error during setup:", e)
    #time.sleep(25)
    # START the next script
    subprocess.Popen(["python3", "/home/solnrned/check_internet.py"])
```

2.Перевірка з’єднання з Інтернетом

Після відпрацювання файлу *sim_setup.py*, спрацьовує наступний скрипт *check_internet.py*:

```
import os
import subprocess

def is_connected():
    try:
        subprocess.check_call(["ping", "-c", "1", "8.8.8.8"],
                              stdout=subprocess.DEVNULL,
                              stderr=subprocess.DEVNULL)
        return True
    except subprocess.CalledProcessError:
        return False

if is_connected():
    os.system("python3 send_logs.py")
    os.system("python3 send_location.py")
```

```
    exit()
else:
    os.system("python3 store_location.py")
    os.system("python3 send_tcp.py")
    exit()
```

Він виконує перевірку доступу до зовнішнього ресурсу (наприклад, Google DNS або Telegram API) шляхом пінгування або спроби встановлення HTTPS-з'єднання.

Якщо з’єднання встановлено — шлюз переходить до наступного кроку. Якщо ж ні — дані відправляються стільниковою мережею.

3.Отримання координат з SIM7000E та їх подальша відправка:

Raspberry Pi взаємодіє з SIM7000E через USB (емуляція послідовного порту UART). З Raspberry надсилаються AT-команди, наприклад AT+CGNSINF, які повертають координати у форматі NMEA або просто широту/довготу в одному рядку.

Відповіді парсяться Python-скриптом, де витягуються лише валідні координати з ознакою фікса GPS.

Для обробки використовується логіка *send_location.py*:

```
import serial
import time
import requests
import sys
import subprocess


subprocess.run(["pkill", "-f", "send_tcp.py"]) # Kill TCP transmission
# CONFIG
SERIAL_PORT = "/dev/ttyUSB3"
BAUDRATE = 115200
TG_BOT_TOKEN =
"8089860531:AAGk0B2oMnVge3vmN3U4gP6QQUdYM7mFkkY"
```

```
TG_CHAT_ID = "636002746"
# Check internet connection
def is_connected():
    try:
        requests.get("https://google.com", timeout=3)
        return True
    except requests.RequestException:
        return False
# Fallback to local storage
def fallback_to_store():
    print("No internet. Running store_location.py and send_tcp.py...")
    subprocess.run(["python3", "/home/solnrned/send_tcp.py"])
    subprocess.run(["python3", "store_location.py"])
    sys.exit()
# Get GPS Info from SIM7000E
def get_gps_info():
    try:
        with serial.Serial(SERIAL_PORT, BAUDRATE, timeout=3) as ser:
            ser.write(b'AT+CGNSINF\r')
            time.sleep(2)
            response = ser.read_all().decode(errors="ignore")
            if "+CGNSINF: " in response:
                parts = response.split(",")
                if parts[1] == "1":  # GPS fix is valid
                    lat = parts[3]
                    lon = parts[4]
                    return lat, lon
            print("GPS fix not available or invalid response.")
    except serial.SerialException as e:
        print("Serial error:", e)
```

```
            except BrokenPipeError:
                print("Broken pipe — device probably disconnected.")
            return None, None
        # Send location and message to Telegram
        def send_location_to_telegram(lat, lon):
            try:
                loc_url =
f"https://api.telegram.org/bot{TG_BOT_TOKEN}/sendLocation"
                msg_url =
f"https://api.telegram.org/bot{TG_BOT_TOKEN}/sendMessage"
                r1 = requests.get(loc_url, params={"chat_id": TG_CHAT_ID,
"latitude": lat, "longitude": lon})
                r2 = requests.get(msg_url, params={"chat_id": TG_CHAT_ID, "text":
f"GPS Update 🔑: {lat}, {lon}"})
                return r1.status_code, r2.status_code
            except Exception as e:
                print("Telegram send error:", e)
                return None, None
        # Main logic wrapped in a function
        def track_and_send():
            lat, lon = get_gps_info()
            if lat and lon:
                print(f"Location: {lat}, {lon}")
                code1, code2 = send_location_to_telegram(lat, lon)
                print("Sent to Telegram:", code1, code2)
            else:
                print("Failed to get GPS fix.")
        #Check connection, if present: Repeating every hour
        if __name__ == "__main__":
```

```
    if not is_connected():
        fallback_to_store()
    while True:
        if not is_connected():
            fallback_to_store()
        else:
            track_and_send()
            print("Waiting 1 hour before next send...\n")
            time.sleep(3597)
```

4.Надсилання координат (офлайн-режим):

Якщо з’єднання з Інтернетом зникає або ж якщо воно відсутнє від початку роботи Raspberry PI, координати так само збираються з модуля, обробляються в Raspberry PI та формується черга AT-команд, що відправляє повідомлення у зовнішній сервер(В нашому випадку Oracle Cloud) для подальшого опрацювання.

Для цього використовується *send_tcp.py*:

```
import serial
import time

SIM_PORT = '/dev/ttyUSB3'
BAUD_RATE = 115200
SERVER_IP = '89.168.105.230'
SERVER_PORT = 5000
SECRET_TOKEN = 'bTJqVPSPsIbL7'
def send_at(ser, command, delay=1):
    ser.write((command + '\r').encode())
    time.sleep(delay)
    return ser.read_all().decode(errors='ignore')

def get_gps_data(ser):
```

```
    resp = send_at(ser, 'AT+CGNSINF', 2)
    if '+CGNSINF:' in resp:
        try:
            line = [l for l in resp.split('\n') if '+CGNSINF:' in l][0]
            parts = line.split(',')
            if parts[1] == '1':  # GPS fix
                lat = parts[3]
                lon = parts[4]
                return lat, lon
        except:
            pass
    return None, None

def open_tcp(ser):
    print("[+] Opening TCP connection...")
    return send_at(ser, f'AT+CIPSTART="TCP","{SERVER_IP}",{SERVER_PORT}', 10)
def send_tcp_data(ser, message):
    send_at(ser, 'AT+CIPSEND')
    time.sleep(1)
    ser.write((message + '\x1A').encode())  # Ctrl+Z to end transmission
    time.sleep(3)
    return ser.read_all().decode(errors='ignore')

def close_tcp(ser):
    send_at(ser, 'AT+CIPCLOSE', 2)

def main_loop():
    while True:
        with serial.Serial(SIM_PORT, BAUD_RATE, timeout=5) as ser:
```

```
            lat, lon = get_gps_data(ser)
            if lat and lon:
                msg = f"{SECRET_TOKEN}|{lat},{lon}"
                print(f"[+]GPS fix:{msg}")
                open_resp = open_tcp(ser)
                if "CONNECT OK" in open_resp or "ALREADY CONNECT" in
open_resp:
                    print("[+] TCP connected.")
                    resp = send_tcp_data(ser, msg)
                    print(f"[+] Send response: {resp}")
                    close_tcp(ser)
                else:
                    print("[-] TCP connection failed.")
            else:
                print("[-] No GPS fix.")
        print("[*] Sleeping for 1 hour...")
        time.sleep(3582)
    if __name__ == '__main__':
        main_loop()
```

5.Відправка кінцевому користувачу з серверу(TCP Listener для офлайн-режиму):

На сервері реалізовано TCP Listener, який автоматично запускається при старті серверу. Основна функція цього компонента полягає у постійному очікуванні вхідних з’єднань на визначеному порту. Після встановлення TCP-з’єднання з боку клієнта (тобто SIM7000E), сервер приймає повідомлення, яке має починатися з попередньо визначеного секретного токена для автентифікації. У випадку, якщо токен не співпадає з очікуваним, повідомлення ігнорується, що запобігає несанкціонованому доступу або передачі некоректних даних. Якщо автентифікація успішна, з отриманого повідомлення парсяться координати GPS, на основі яких формується текстове

повідомлення. Це повідомлення надсилається кінцевому користувачеві через Telegram за допомогою HTTPS-запиту до Telegram Bot API.

Роль TCP Listener’a відіграє скрипт-файл tcp_to_tg.py з наступним наповненням:

```
import socket
import requests

# === CONFIGURATION ===
HOST = '0.0.0.0'
PORT = 5000
TOKEN = 'bTJqVPSPsIbL7'
BOT_TOKEN = '8089860531:AAGk0B2oMnVge3vmN3U4gP6QQUdYM7mFkkY'
CHAT_ID = '636002746'
# === TELEGRAM FUNCTION ===
def send_to_telegram(message: str):
    url = f"https://api.telegram.org/bot{BOT_TOKEN}/sendMessage"
    loc_url = f"https://api.telegram.org/bot{BOT_TOKEN}/sendLocation"
    coords = message.split(":")[1]
    lat = coords.split(",")[0]
    lon = coords.split(",")[1]
    print(lat,lon)
    payload = {'chat_id': CHAT_ID, 'text': message}
    loc_payload = {"chat_id": CHAT_ID, "latitude": lat, "longitude": lon}
    try:
        r = requests.post(url, data=payload)
        loc_r = requests.post(loc_url, data=loc_payload) #params={"chat_id": CHAT_ID, "latitude": lat, "longitude": lon})
        print(f"[+] Sent to Telegram: {message} | Status: {r.status_code}| Status_loc: {loc_r.status_code}")
```

```
    except Exception as e:
        print(f"[!] Failed to send to Telegram: {e}")
# === TCP SERVER ===
def start_server():
    print(f"[+] Starting TCP server on {HOST}:{PORT}")
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        while True:
            conn, addr = s.accept()
            with conn:
                print(f"[+] Connected by {addr}")
                try:
                    data = conn.recv(1024).decode().strip()
                    print(f"[>] Received: {data}")
                    if not data.startswith(TOKEN + "|"):
                        print("[!] Invalid token received, ignoring.")
                        continue
                    _, payload = data.split("|", 1)
                    send_to_telegram(f"GPS Update: {payload}")
                except Exception as e:
                    print(f"[!] Error: {e}")
if __name__ == "__main__":
    start_server()
```

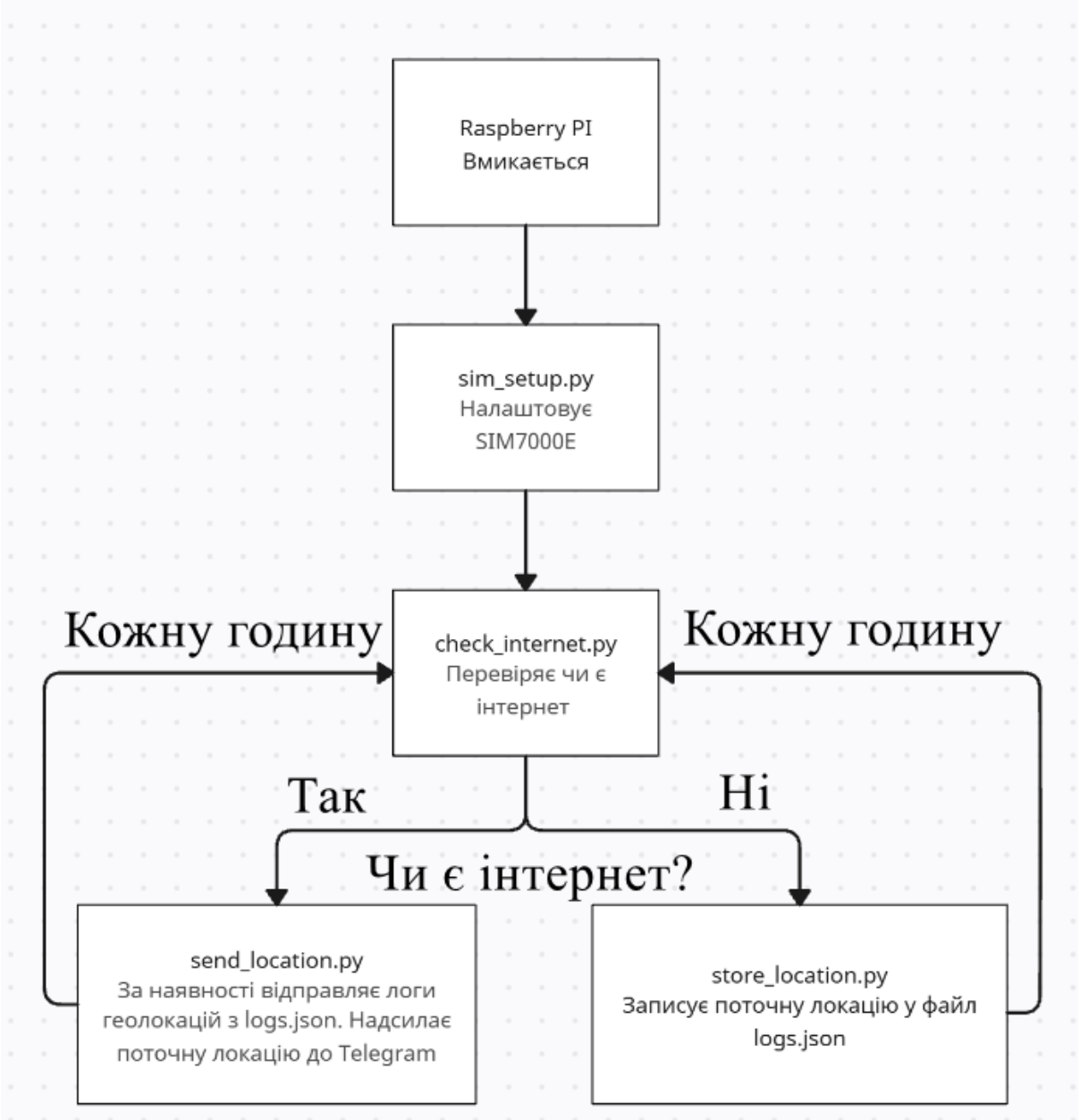

Рисунок 2.3 - Схема взаємодії скриптів Raspberry PI

Загальні переваги обраної логіки:

Надійність: система не втрачає дані у разі втрати мережі.

Безпека: передача координат здійснюється по HTTPS, уникнуто використання SIM7000E для цього, бо в ньому складне налаштування HTTPS.

Гнучкість: можливість масштабування під інші API, сенсори, сервіси.

Простота: код розділений на автономні логічні блоки, які легко підтримувати або змінювати.

2.3.3 Тестування локальної взаємодії

Локальна взаємодія між модулями системи IoT-шлюзу була протестована на предмет стабільності виконання задач, обробки помилок і правильності перемикання між режимами з’єднання.

Після запуску пристрою скрипт *check_internet.py* виконується один раз. Він визначає, чи є інтернет-з’єднання, та на основі цього запускає один із двох основних скриптів: *send_location.py* (у випадку активного з’єднання) або *send_tcp.py* (при його відсутності). Обидва ці скрипти запускаються далі раз на годину. Кожен із них під час запуску повторно перевіряє стан з’єднання. Якщо виявляється, що статус мережі змінився (з’єднання зникло або з’явилося), скрипт автоматично завершує свою роботу та запускає протилежний. Таким чином забезпечується гнучке й самовідновлюване перемикання між режимами "офлайн" та "онлайн" без необхідності перезапуску пристрою.

Усі помилки, що виникають під час роботи скриптів — наприклад, втрата GPS-зв’язку, проблеми з Telegram API або доступом до файлу — виводяться безпосередньо в консоль. Це дозволяє бачити причину збою при ручному запуску або відлагодженні.

За потреби можливо впровадити логування помилок у файл, однак у поточній реалізації це не є необхідним. Оскільки вся логіка перемикання вбудована у скрипти, а запуск відбувається періодично, система не вимагає постійного моніторингу з боку користувача.

Система була протестована у двох режимах: з активним інтернет-з’єднанням та у повністю офлайн середовищі, з подальшим перемиканням режимів на протилежний та назад у реальному часі. Логіка відпрацювала коректно в усіх випадках, з автоматичним переходом між офлайн та онлайн режимами.

2.4 Висновки до розділу 2

1. Реалізовано гнучку та енергоефективну архітектуру шлюзу, що адаптується до змін навколишнього середовища.

IoT-шлюз, розроблений у межах дипломного проєкту, базується на одноплатному комп’ютері Raspberry Pi 3 Model B та модулі SIM7000E. Вибір цих компонентів зумовлений поєднанням доступності, функціональності й енергоефективності. Завдяки активному GPS-модулю система здатна надійно фіксувати координати навіть у складних умовах (автомобіль, міська забудова). Архітектура дозволяє масштабування — через доступ до GPIO та можливість додавання нових датчиків.

2. Програмна логіка забезпечує автономну роботу пристрою незалежно від наявності інтернет-з’єднання.

В системі реалізовано програмну модель, що враховує як стабільні, так і аварійні сценарії роботи. У разі наявності інтернет-з’єднання координати автоматично надсилаються у Telegram через HTTPS-запити. Якщо ж підключення відсутнє — система переходить на «офлайн» режим та надсилає данні TCP запитом через стільникову мережу . Такий підхід дозволяє гарантувати цілісність даних та забезпечити повну автономність пристрою у реальних умовах експлуатації.

3. Передбачено ключові захисти від зовнішніх загроз: енергозбереження, автоматичне відновлення роботи та живлення.

Розробка враховує загрози, притаманні мобільним системам — перебої живлення, втрату зв’язку, зміну температури тощо. Автоматичний запуск скриптів після перезавантаження, використання буферизації координат та обробка збоїв у логіці забезпечують надійність. Передбачено можливість живлення від акумулятора або бортової мережі, а завдяки енергоощадним режимам модему та оптимізації роботи Raspberry Pi — досягається зниження загального енергоспоживання.

3 АНАЛІЗ РОБОТИ СИСТЕМИ ТА ПОТЕНЦІАЛ ЇЇ ВИКОРИСТАННЯ

3.1 Налагодження та перевірка шлюзу

3.1.1 Тестування стабільності з’єднання

У системі передача даних до Telegram відбувається за наявності Wi-Fi з’єднання, що забезпечується самим Raspberry Pi. Проте якщо з’єднання з мережею на мікроконтролері відсутнє, задіюється логіка надсилання даних з SIM7000E стільниковою мережею Для тестування стабільності з’єднання реалізовано кілька перевірок.

Перед кожним запуском скриптів *send_location.py* або *send_tcp.py* виконується перевірка наявності інтернету за допомогою запиту до публічного ресурсу. Якщо з’єднання відсутнє — скрипт автоматично перемикається на офлайн режим (send_tcp.py). У разі появи з’єднання — відбувається зворотне перемикання.

Псевдокод перевірки:

if ping("8.8.8.8") == success:

  інтернет наявний → передаємо дані

else:

  інтернету немає → вмикаємо офлайн режим

Для додаткового моніторингу можливе використання команд на зразок:

iwconfig wlan0 або *cnmcli dev wifi*, які показують силу сигналу Wi-Fi у dBm. Однак у поточній реалізації ця інформація не використовується в логіці системи, оскільки достатньо бінарного сигналу: «з’єднання є / немає».

У разі раптової втрати мережі:

скрипт send_location.py фіксує відсутність інтернету та автоматично запускає send_tcp.py;

навпаки, send_tcp.py при виявленні з’єднання автоматично запускає send_location.py;

логіка працює циклічно без зовнішнього моніторингу або втручання користувача.

Це дозволяє шлюзу автономно адаптуватися до змін у доступності мережі та не втрачати дані, навіть при довготривалих відключеннях зв’язку.

3.1.2 Перевірка передачі GPS-даних

Для перевірки коректної роботи модуля SIM7000E з GPS-приймачем та передачі координат до Telegram було проведено низку тестів із фіксацією отриманих значень.

Дані запитуються через AT-команду AT+CGNSINF, після чого парсяться скриптом і зберігаються та надсилаються з SIM7000E (при відсутності з’єднання) або одразу надсилаються (за наявності з’єднання).

У випадку відсутності інтернет-з’єднання координати записуються в файл logs.json. Структура цього лог-файлу:

```
[
  {
    "timestamp": "2025-06-05T16:10:00",
    "lat": 50.4501,
    "lon": 30.5234
  },
  {
    "timestamp": "2025-06-05T17:10:00",
    "lat": 50.4503,
    "lon": 30.5237
  }
]
```

Файл оновлюється з кожною новою ітерацією (раз на годину), додаючи останні координати.

При появі з’єднання запускається скрипт *send_location.py*, який: зчитує logs.json, формує повідомлення з координатами по кожному запису, надсилає

їх через Telegram Bot API у форматі HTTPS-запиту, після успішної передачі очищає файл.

Приклад HTTPS-запиту(одне повідомлення складається з двох запитів: локація та текст з координатами):

https://api.telegram.org/bot8089860531:AAGk0B2oMnVge3vmN3U4gP6QQUdYM7mFkkY/sendLocation?chat_id=636002746&latitude=50.407921&longitude=30.665612

https://api.telegram.org/bot8089860531:AAGk0B2oMnVge3vmN3U4gP6QQUdYM7mFkkY/sendMessage?chat_id=636002746&text=GPS Update 🔑: 50.407921, 30.665612

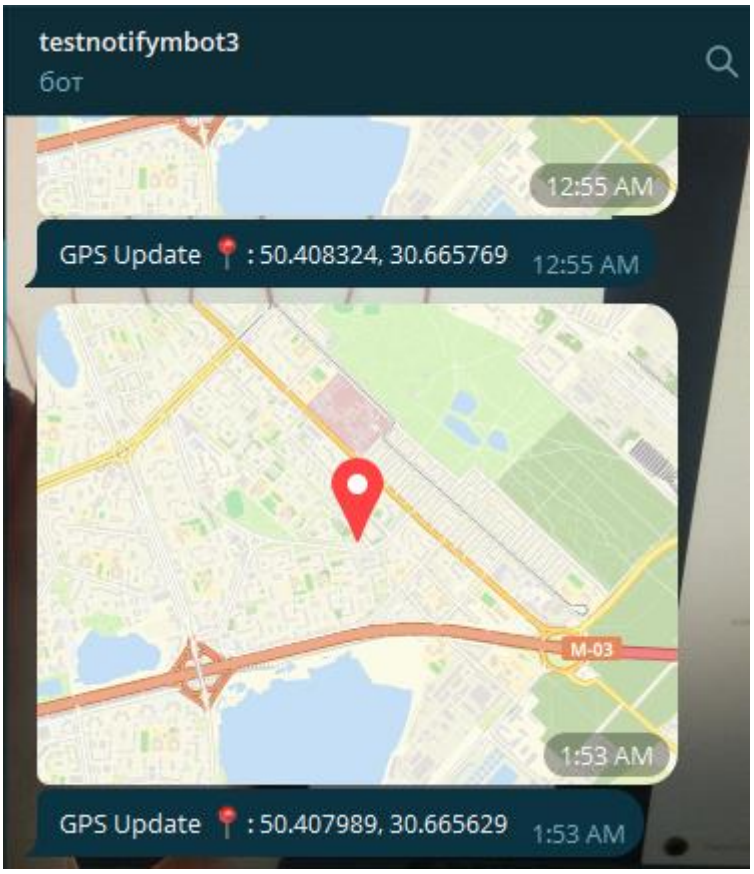

Рисунок 3.1 - Приклад повідомлення в Telegram:

Було зроблено декілька виїздів із встановленим модулем у транспортному засобі. У log-файлі з’явилися координати, які приблизно

відповідали реальному місцезнаходженню (похибка в межах 5–10 м у місті), також ці дані було надіслано стільниковою мережею з SIM7000E.

Після відновлення з’єднання дані були успішно надіслані до Telegram без втрат. Відсутність збоїв свідчить про стабільність логіки збереження та відправки.

3.1.3 Імітація збоїв системи

Кейс 1

Для перевірки стійкості системи до збоїв було проведено тестування, в якому вручну імітувалися непередбачувані зупинки окремих скриптів.

Зокрема, під час роботи скриптів *send_tcp.py* (режим локального збереження GPS-координат) та *send_location.py* (режим онлайн відправки GPS-координат) було примусово завершено їхні процеси за допомогою команди *kill*. Це дозволило змоделювати ситуацію, в якій скрипт несподівано перестає працювати — наприклад, через помилку в коді, нестачу пам’яті чи втрату доступу до серійного порту.

Після цього було виконано перезапуск Raspberry Pi. При запуску автоматично запускається скрипт *sim_setup.py,* що по відпрацюванню запускає файл *check_connection.py*, який визначає наявність інтернету. Якщо мережа відсутня, він викликає *send_tcp.py*, що і сталося в нашому випадку. Після відновлення мережі система автоматично переключилась на *send_location.py* та відправила логування, яке було збережено до та після примусового завершення процессу. Таким чином, роботу було успішно відновлено без втрати функціональності та геоданних навіть у випадку відсутності мережі стільникового сигналу.

Цей тест показав, що:

при збої одного зі скриптів систему можна автоматично відновити через перезапуск;

наявний механізм перемикання між онлайн- та офлайн-режимами не потребує ручного втручання.

Кейс 2

Під час тестування було змодельовано ситуацію втрати інтернет-з'єднання в процесі нормальної роботи системи. На початку скрипт *send_location.py* успішно надсилав GPS-координати до Telegram-бота в режимі онлайн.

Для перевірки реакції на зміну умов було вручну відключено інтернет-з'єднання за допомогою команди:

sudo rfkill block wifi

Після цього скрипт більше не мав доступу до мережі, і згідно з логікою роботи мав перейти в режим надсилання даних по LTE та їх збереження локально . Було зроблено паузу, щоб система встигла зібрати нові координати та зберегти їх у файл логів.

Однак після перезапуску Raspberry Pi виявилося, що дані, отримані в момент втрати інтернету, не були збережені. Це відбулося через те, що файл логів створювався в оперативній пам'яті, але не записувався на диск, а також через відсутність механізму відправлення накопичених логів після перезапуску.

Було внесено такі зміни до системи:

Додано примусовий запис файлу з логами одразу після кожного збереження координат, а також його бекап, щоб уникнути їх втрати при раптовому завершенні роботи або перезапуску.

Створено новий скрипт *send_logs.py*, що відповідає за відправку раніше накопичених логів до Telegram-бота.

Файл *check_internet.py* було модифіковано: тепер при наявності з'єднання при старті системи, він викликає *send_logs.py*, що забезпечує автоматичну відправку раніше збережених логів та їх видалення для економії пам'яті.

Після повторного тестування виявлено, що система коректно переходить у режим збереження логів при втраті мережі, а після перезапуску та відновлення з’єднання — автоматично надсилає всі збережені координати. Таким чином, реалізовано повноцінну підтримку автономної роботи з наступним синхронізуванням даних.

3.2 Показники ефективності

3.2.1 Буферизація, зменшення кількості запитів, розклад роботи

Оптимізація IoT-системи не обмежується лише вибором апаратного забезпечення або налаштуванням мережі. Велике значення має грамотна організація режиму роботи пристрою, що передбачає мінімізацію зайвих запитів, ефективне використання буфера пам’яті та контроль над розкладом виконання основних скриптів. Такі заходи дозволяють суттєво зменшити навантаження на систему, заощадити трафік та зменшити енергоспоживання, що особливо критично для пристроїв, які працюють автономно або в обмежених умовах.

У реалізованій системі розклад роботи реалізовано за допомогою базового інструменту — функції time.sleep() з модуля time стандартної бібліотеки Python. Цей підхід дозволяє задати фіксований інтервал між циклами виконання коду, не вимагаючи складних зовнішніх засобів планування (на кшталт *cron* або системних таймерів). Зокрема, після кожного сеансу передачі даних або запису в лог, скрипт переходить у режим "сну" на певний період (наприклад, одну годину), після чого автоматично відновлює роботу. Це дозволяє суттєво знизити споживання ресурсів і не створювати постійне навантаження на процесор чи мережу.

Така реалізація не залежить від зовнішніх тригерів і дозволяє створити достатньо стабільну систему з передбачуваним графіком.

Буферизація даних

Щоб не втрачати дані при відсутності мережевого з'єднання, у системі реалізовано буферизацію шляхом збереження координат та часу в окремий JSON-файл (logs.json). Якщо при виконанні основного скрипта надсилання повідомлення в Telegram не вдається через відсутність інтернету, інформація про поточні координати записується у цей файл.

Основна перевага такого методу — незалежність від зовнішнього з’єднання: навіть при тривалих відключеннях мережі всі зібрані дані зберігаються локально й можуть бути передані пізніше, коли з’єднання відновиться.

Удосконаленням цього підходу стало додавання backup-файлу та негайного збереження файлу на диск після кожного запису, що усуває ризик втрати даних при аварійному завершенні, перезавантаженні пристрою або пошкоджені файлу.

Сама архітектура передбачає мінімальну кількість мережевих запитів, що дозволяє суттєво економити трафік, особливо при використанні тарифів з обмеженим обсягом передачі даних. Такий підхід також знижує ризик перевантаження мобільної мережі або блокування через надмірну активність.

На відміну від підходів, де GPS-дані передаються в реальному часі, тут пріоритет надано "періодичній доставці", що добре підходить для трекінгу об’єктів із невисокою динамікою переміщення (наприклад, автомобіль, що стоїть на парковці або повільно рухається).

Переваги підходу

- Простота реалізації, що дозволяє швидко налагодити систему навіть без досвіду у роботі з планувальниками.
- Енергозбереження завдяки паузам між виконанням.
- Гнучкість: у майбутньому систему можна легко адаптувати до інших інтервалів або додати підтримку wake-up сигналів.
- Стабільність при відсутності мережі завдяки буферизації.

Обмеження

- time.sleep() блокує потік виконання, що унеможливлює паралельну роботу декількох процесів у межах одного скрипта.
- Відсутність точного контролю над часом запуску (похибка накопичується при кожному циклі).

Для складніших задач (наприклад, з розгалуженими умовами або обробкою подій) доцільно використовувати більш просунуті механізми: cron, systemd-timers, APScheduler тощо.

3.2.2 Витрати енергії в різних режимах

Енергоспоживання є критичним фактором для IoT-пристроїв, особливо тих, що працюють автономно. У цьому розділі розглянемо споживання енергії Raspberry Pi 3B та модуля SIM7000E в різних режимах роботи.

Згідно з вимірюваннями, проведеними на сайті Raspberry Pi Dramble, Raspberry Pi 3B має наступні показники споживання енергії:

Режим очікування (idle): приблизно 260 мА при 5 В (≈1.3 Вт)

Під навантаженням (наприклад, передача даних через Wi-Fi): до 400 мА при 5 В (≈2.0 Вт)

Ці дані підтверджуються також іншими джерелами, які вказують, що споживання енергії може варіюватися залежно від активності пристрою та підключених периферійних пристроїв.

Згідно з технічною документацією модуля SIM7000E, його споживання енергії в різних режимах становить:

- Режим очікування (idle): приблизно 11 мА
- Режим передачі даних через LTE: до 200 мА
- Режим передачі даних через NB-IoT: до 150 мА
- Режим сну (sleep): приблизно 1 мА
- Режим PSM (Power Saving Mode): приблизно 9 мкА

Ці значення можуть змінюватися залежно від умов роботи, таких як якість сигналу та частота передачі даних.

Таблиця 3.1 - Порівняльна таблиця енергоспоживання

Режим роботи	Raspberry Pi 3B	SIM7000E LTE	SIM7000E NB-IoT
Очікування (idle)	~260 мА	~11 мА	~11 мА
Передача даних	~400 мА	~200 мА	~150 мА
Режим сну (sleep)	-	~1 мА	~1 мА
Режим PSM (Power Saving Mode)	-	~9 мкА	~9 мкА

Raspberry Pi 3B споживає значну кількість енергії навіть у режимі очікування, що робить його менш придатним для енергоефективних IoT-застосувань без додаткових заходів оптимізації.

Модуль SIM7000E демонструє низьке споживання енергії, особливо в режимах сну та PSM, що робить його ідеальним для автономних IoT-пристроїв.

Використання режиму PSM для модуля SIM7000E дозволяє значно знизити споживання енергії під час простою.

Оптимізація роботи Raspberry Pi 3B, наприклад, шляхом вимкнення непотрібних служб або використання альтернативних мікроконтролерів з нижчим споживанням енергії, може покращити загальну енергоефективність системи.

3.2.3 Вдосконалення архітектури системи

Надійність є критичним аспектом у розробці IoT-систем, особливо у випадках, коли пристрій працює автономно, без постійного доступу до мережі

або фізичного втручання користувача. Під терміном fault-tolerance розуміється здатність системи продовжувати коректно функціонувати навіть у випадках часткових відмов або помилок.

Загальні принципи побудови надійної архітектури включають:

- Ізоляція компонентів — щоб помилка в одному модулі не порушила роботу всього пристрою.
- Автоматичне виявлення відмов — зазвичай через моніторинг статусів підсистем.
- Відновлення після помилок — повторні спроби, перезапуск компонентів, логування.
- Локальна буферизація даних — дозволяє уникнути втрати даних при розриві з’єднання.

У контексті low-power IoT-рішень fault-tolerance часто має бути реалізовано максимально просто, без зайвого навантаження на обчислювальні ресурси.

Під час початкового тестування системи було виявлено, що за деяких умов дані, зібрані під час відсутності з’єднання, не зберігалися належним чином або не передавались після перезапуску пристрою. Конкретно:

- при втраті інтернет-з'єднання (імітація через *sudo rfkill block wifi*) та наступному перезапуску Raspberry Pi система не надсилала раніше збережені координати;
- виявлено, що файл з логами не записувався на диск одразу, що призводило до втрати даних при раптовому вимкненні живлення або перезапуску.

У відповідь на виявлені проблеми були внесені наступні вдосконалення в архітектуру:

Було додано явне відкриття та закриття файлу при кожному записі нової координати. Це гарантує, що навіть у випадку раптового завершення роботи

дані не залишаться в буфері пам’яті, а фізично будуть записані у файл (*logs.json*).

Оновлений блок коду виглядає так:

```
# Write updated logs with flush + fsync
  with open(LOG_FILE, 'w') as f:
    json.dump(logs, f, indent=2)
    f.flush()              # Push from Python buffer to OS
    os.fsync(f.fileno())   # Force OS to write it to disk
```

Реалізовано окремий файл, який при наявності інтернет-з’єднання зчитує вміст *logs.json* і надсилає накопичені дані до Telegram-бота, після чого очищає файл логів. Це дозволяє передавати історичні координати, зібрані в офлайн-режимі.

У файл перевірки з’єднання (*check_internet.py*), який запускається при старті системи, було додано виклик *send_logs.py*, якщо інтернет доступний. Це дозволяє системі автоматично "доробити роботу", що накопичилась в офлайн-режимі, при наступному запуску.

Псевдокод:

```
if check_internet():
  run("python3 send_logs.py")
  run("python3 send_location.py") #start online mode
else:
  run("python3 store_location.py")
```

Після впровадження змін система пройшла успішне тестування за наступним сценарієм:

1. під час активної роботи було відключено Wi-Fi модуль Raspberry Pi;
2. система зберігала координати локально в logs.json;
3. після перезапуску пристрою (при відновленому з'єднанні) скрипт check_connection.py автоматично активував send_logs.py, і накопичені дані були успішно передані до Telegram-бота.

Це свідчить про значне підвищення надійності системи. Втрата даних у випадку розриву з'єднання або перезапуску була виключена.

Можливості подальшого вдосконалення:

- Запровадження log-аналітики — наприклад, аналіз частоти втрати з’єднання, середньої тривалості офлайн-режиму тощо.
- Додавання сигналізації про помилки до Telegram-бота (наприклад, повідомлення у разі накопичення великої кількості невідправлених записів).
- Використання crash recovery механізмів — таких як systemd Watchdog або перезапуск за помилки виконання скриптів.

3.3 Оптимізація

3.3.1 Мобільний GPS-трекінг автотранспорту

Одним із найбільш поширених та практично затребуваних сценаріїв використання IoT-шлюзів з вбудованими модулями GSM та GPS є мобільний моніторинг транспортних засобів. Така система дозволяє у режимі реального часу або з відкладеною передачею фіксувати координати об'єкта, що рухається, та передавати їх до центрального хмарного сервісу чи безпосередньо користувачу.

Актуальність трекінгу автотранспорту

GPS-моніторинг транспорту широко впроваджено в логістиці, каршерінгу, таксі, службах доставки, комунальному господарстві та аграрному секторі. Основні цілі впровадження таких систем включають:

- Контроль за переміщенням та маршрутом — дозволяє у будь-який момент визначити точне місцезнаходження транспорту.
- Оптимізація логістики — GPS-треки допомагають аналізувати маршрути, зменшувати холості пробіги, планувати зупинки.

- Зниження витрат — завдяки виявленню несанкціонованих поїздок, перевищення швидкості або нераціонального використання пального.
- Підвищення безпеки — моніторинг у реальному часі дає змогу швидко реагувати на викрадення або нестандартні ситуації.

Розроблений IoT-шлюз на базі Raspberry Pi 3 Model B у зв'язці з модулем SIM7000E повністю відповідає вимогам до базової GPS-трекінг системи:

- GPS-антена, підключена до SIM7000E, дозволяє отримувати координати з високою точністю;
- Передача даних через мережу LTE/NB-IoT або Wi-Fi дозволяє надсилати координати у Telegram-бот, який виконує функцію центрального сервера;
- Збереження логів при втраті мережі дозволяє працювати у режимі "офлайн-накопичення", що надзвичайно важливо для мобільних систем у зоні нестабільного покриття.

У ході випробування шлюз було встановлено в автомобіль, після чого проведено тест-драйв у міських умовах. Система не мала інтернет-з’єднання та були постійні перебої з живленням, проте вона: продовжувала зчитувати координати; надсилала їх стільниковою мережею; зберігала їх у локальний файл; при відновленні з'єднання автоматично надсилала всі накопичені координати у вигляді повідомлень Telegram-боту.

З’єднавши всі отримані точки на мапі в хронологічному порядку отримаємо приблизну хронологію пересувань автомобілю:

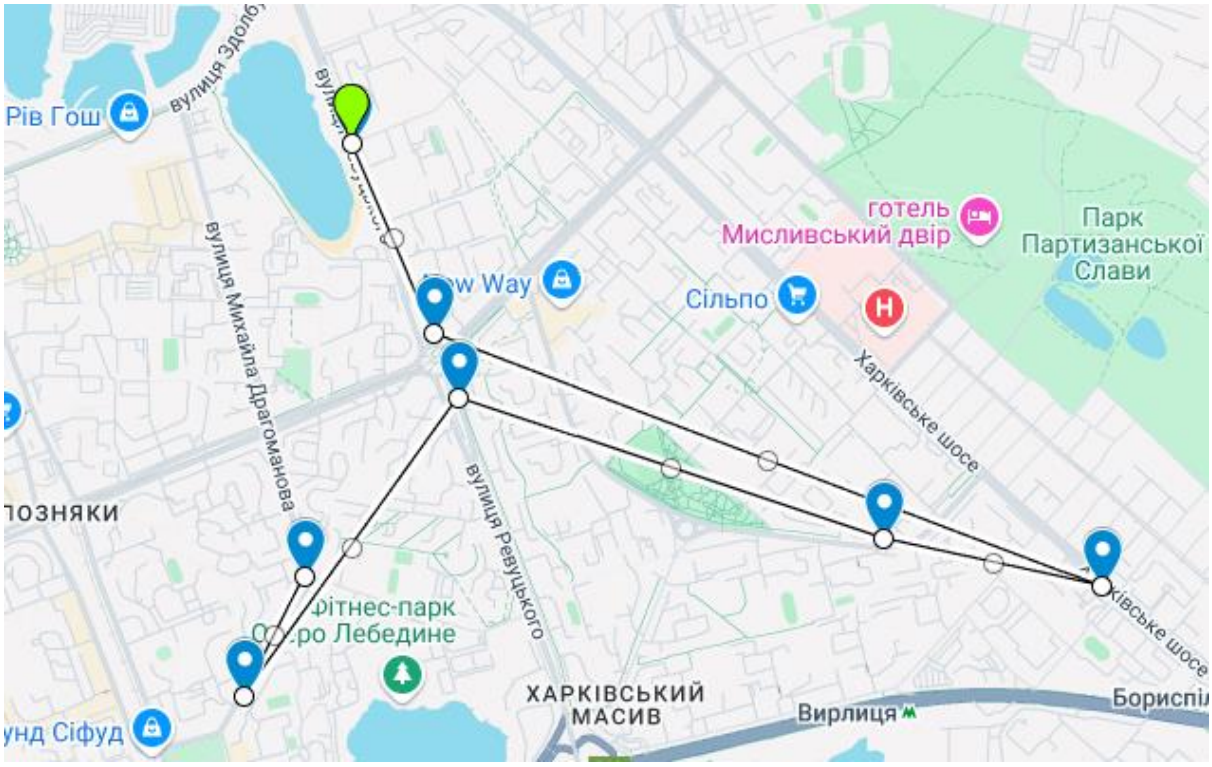

Рисунок 3.2 - Зображення хронології пересувань автомобілю згідно отриманих координат

Таким чином, забезпечено повноцінний безперервний GPS-трекінг, навіть у випадках втрати мобільного зв’язку.

Порівняно з комерційними рішеннями, система має низку переваг:

- Відкритість архітектури — можливість кастомізації під конкретні потреби.
- Низька вартість — використання дешевих компонентів та безкоштовного хмарного сервісу (Telegram).
- Автономність — можливість живлення від автомобіля або портативного акумулятора.

З огляду на гнучку архітектуру, систему можна масштабувати для таких задач:

- моніторинг автопарків з відображенням координат на мапі;
- розширення функціоналу (контроль температури, рівня пального тощо);

- побудова аналітичної панелі для звітності та сповіщень.

Крім цього, трекер можна інтегрувати з зовнішніми API, наприклад, OpenStreetMap або Google Maps API, для відображення маршрутів на карті.

3.3.2 Віддалений моніторинг техніки

Одним із ключових напрямів застосування IoT-шлюзів є віддалений контроль та моніторинг технічного обладнання, що працює в умовах обмеженого доступу, важкопрохідних місцевостей або автономного розміщення. Такі системи особливо актуальні в аграрному секторі, на виробничих об’єктах, в енергетиці та на будівельних майданчиках.

Віддалений моніторинг техніки дозволяє власнику або оператору: контролювати стан обладнання у реальному часі, отримувати регулярні оновлення про координати, температуру, рівень заряду тощо, оперувати на відстані — без необхідності фізичної присутності, зменшити час реагування на несправності або поломки, забезпечити історичний запис усіх подій та спостережень

У класичному сценарії мова йде не лише про трекінг, а й про збір телеметричних даних — показників із сенсорів, які фіксують температуру, вібрацію, вологість, рух або споживання енергії.

Розроблена в рамках цього проєкту система вже містить основні компоненти, необхідні для базового віддаленого моніторингу:

- SIM7000E, який забезпечує мобільний канал зв’язку для надсилання зібраних даних (через мобільну мережу LTE/NB-IoT).
- Raspberry Pi, який виступає як головний контролер та обробник даних із сенсорів.
- Логіка буферизації і логування, що забезпечує надійність при втраті інтернет-з’єднання.

- Telegram-бот, який слугує каналом зв’язку між системою та оператором.

Крім віддаленої взаємодії через Telegram, передбачено локальний доступ до пристрою за допомогою SSH (у тій мірі, в якій дозволяє оточення):

Якщо Raspberry Pi підключений до локальної Wi-Fi-мережі або через Ethernet — можливе стандартне SSH-з’єднання

У разі прямого фізичного підключення до ноутбука або ПК через USB-to-Ethernet або UART/Serial інтерфейс — також можна підключитися до Pi в якості віддаленого терміналу.При використанні USB OTG (якщо активовано відповідну функцію на Pi) — можливо емулювати Ethernet-інтерфейс через microUSB-кабель.В умовах повної ізоляції від мережі (наприклад, у полі) єдиним способом доступу лишається підключення монітора і клавіатури через HDMI та USB.Якщо Raspberry Pi працює виключно з SIM7000E як каналом зв’язку, SSH-підключення через мобільну мережу (тобто, по IP-адресі, отриманій через LTE/NB-IoT) буде практично недоступним:

- SIM-оператори, як правило, видають локальні або NAT’овані IP-адреси, які не доступні ззовні.
- Для реального SSH-доступу через мобільну мережу потрібно замовляти статичну "білу" IP-адресу, що дорого і не завжди можливо.

Як альтернатива можна було б налаштувати VPN-клієнт або reverse SSH tunnel до хмарного сервера, але це вже значно ускладнює архітектуру. Через Telegram-бот — отримуємо координати, повідомлення про статус, контрольні логи, тобто обмежений канал зв’язку з низьким трафіком. Через локальний SSH-доступ (якщо Wi-Fi або Ethernet підключення є) — повний контроль: можна переглядати логи, вручну запускати або зупиняти скрипти, аналізувати поточні процеси. Також можна автоматично логувати внутрішні помилки і записувати в окремий файл (*error.log*), який буде надсилатись при першій нагоді (через *send_logs.py*).

Типовий сценарій використання

Уявімо віддалену насосну станцію в полі, де немає постійного доступу до мережі та людей. На базі Raspberry Pi встановлюється шлюз з модулем SIM7000E та температурним сенсором. Кожну годину система:

Зчитує показники температури, вологості та GPS.

Перевіряє наявність з'єднання.

Якщо з'єднання є — надсилає інформацію в Telegram.

Якщо з'єднання немає — зберігає дані у лог-файл.

При наступному запуску або появі мережі — передає всі логи.

В Telegram приходить, наприклад:

Станція: Південна насосна

Температура: 37.2°C

Координати: 48.40125, 31.21342

Час: 13:05

Це дозволяє контролювати не лише геолокацію, а й фізичні умови роботи обладнання, що критично важливо для запобігання перегріву, замерзання чи аварій.

Переваги такого підходу:

- може працювати від сонячної панелі або батареї.
- передачі відбуваються раз на годину, дані стислі.
- можна змінювати сенсори чи алгоритми за потреби.
- Telegram-бот працює без складного серверного середовища.

Система може масштабуватись як вшир (додавання нових точок моніторингу), так і вглиб (розширення типів даних):

- Інтеграція з платними API для побудови веб-інтерфейсів
- Відправлення сповіщень про критичні значення (наприклад, висока температура)
- Підключення до SCADA-систем або хмарних платформ, таких як Blynk, ThingsBoard або Node-RED

3.3.3 Пропозиції щодо подальшого вдосконалення системи

Поточна версія IoT-системи успішно реалізує сценарії передачі геокоординат як у присутності інтернет-з'єднання (через Wi-Fi та HTTPS), так і в умовах його відсутності (через TCP-з’єднання по стільниковій мережі). Незважаючи на стабільність роботи та гнучкість архітектури, існує низка напрямів, які можуть покращити надійність, ефективність і масштабованість рішення. Одним із таких напрямів є вдосконалення механізму буферизації даних у разі повної втрати зв’язку — замість простого зберігання координат у текстовому файлі доцільно використати структуровану чергу (наприклад, на основі SQLite), що дозволить гарантувати доставку всіх повідомлень у хронологічному порядку. Крім того, серверна частина може бути розширена функцією підтвердження отримання, з повторною передачею координат у разі втрати відповіді. З погляду безпеки, варто розширити механізми автентифікації, наприклад, через обмеження за IP або шифрування TCP-пакетів. Також перспективним виглядає впровадження базового інтерфейсу моніторингу стану пристрою через Telegram або локальний веб-інтерфейс, що дозволило б оперативно перевіряти наявність з’єднання, стан GPS, кількість невідправлених записів тощо. Таким чином, навіть без радикальної зміни апаратної платформи, систему можливо суттєво вдосконалити за рахунок покращення обробки помилок, безпеки та контролю виконання.

3.4 Висновки до розділу 3

1. Система продемонструвала високу стабільність передачі даних і ефективність роботи в умовах реального навантаження.

У ході тестування перевірено стабільність GPS-з’єднання, надійність зчитування координат через AT-команди, а також правильність логіки передачі даних через API Telegram. Зафіксовано відсутність втрат даних навіть при перебоях інтернету, що свідчить про ефективну реалізацію буферизації та

повторної передачі. Система підтримує регулярний режим опитування та коректно функціонує з мінімальними затримками.

2. Завдяки застосуванню буферизації та щогодинного розкладу передачі, оптимізовано витрати трафіку та енергії.

Реалізований часовий інтервал між передачами даних дозволяє мінімізувати навантаження на інтернет-канал і зменшити енергоспоживання системи. Замість надсилання координат у реальному часі використано стратегію періодичного накопичення з подальшою відправкою. Такий підхід забезпечує баланс між актуальністю даних і ресурсозбереженням, що особливо важливо для мобільних або віддалених IoT-пристроїв.

3. Запропоновані напрями масштабування підтверджують практичну цінність і прикладний потенціал розробки.

У роботі розглянуто сценарії застосування системи в автотрекінгу, сільському господарстві, моніторингу техніки тощо. Передбачена можливість додавання нових сенсорів (температура, вологість, рівень пального) без зміни основної архітектури. Також запропоновано розвиток у напрямку додаткової безпеки серверу, використання структурованої черги логування та інтерфейс моніторингу

ВИСНОВКИ

У межах дипломної роботи розроблено та реалізовано функціональний IoT-шлюз на основі одноплатного комп’ютера Raspberry Pi 3 Model B та модему SIM7000E, орієнтований на автономне отримання та передавання GPS-координат до хмарного сервісу. Проведено глибокий теоретичний аналіз архітектури Інтернету речей, принципів побудови шлюзів та особливостей бездротових технологій зв’язку, що дозволило обґрунтовано підійти до вибору компонентів і методів реалізації системи.

Проектна частина роботи охоплює як апаратну, так і програмну реалізацію шлюзу, із врахуванням реальних обмежень середовища експлуатації — обриви зв’язку, перебої живлення, мобільність об’єкта. Реалізована система відзначається гнучкістю, надійністю та можливістю масштабування: забезпечується буферизація даних, автоматичне переключення режимів залежно від наявності інтернету, а передача здійснюється через захищені HTTPS або TCP-з’єднання у Telegram API. Тестування продемонструвало високу стабільність роботи в автономному режимі, ефективність енергоспоживання та потенціал практичного застосування в задачах моніторингу транспорту, сільському господарстві, безпеці та логістиці.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rayes A., Salam S. Internet of Things: From Hype to Reality. – Springer, 2016. – [https://doi.org/10.1007/978-3-319-44860-2]

2. Buyya R., Dastjerdi A. V. Internet of Things: Principles and Paradigms. – Elsevier, 2016.

3. Rescorla E. SSL and TLS: Designing and Building Secure Systems. – Addison-Wesley, 2000.

4. Sauter M. From GSM to LTE-Advanced Pro and 5G: An Introduction to Mobile Networks and Mobile Broadband. – Wiley, 2017.

5. Tanenbaum A. S., Wetherall D. J. Computer Networks, 5th Edition. – Pearson, 2011.

6. "A Survey of IoT Security: Techniques, Protocols and Challenges" – ACM Computing Surveys, 2020.

7. "A Review of GPS Data Loggers for Monitoring Vehicle Fleet Movement", IEEE, 2020 – [https://ieeexplore.ieee.org/document/9042222]

8. IBM. IoT Gateways and Edge Devices – [https://www.ibm.com/cloud/learn/iot-gateways]

9. u-blox. Cellular technology comparison: NB-IoT, LTE-M and more – [https://www.u-blox.com/en/blogs/cellular-technology-comparison]

10. GSMA. NB-IoT & LTE-M Deployment Guide – [https://www.gsma.com/iot/deployment-guide]

11. GSMA. Mobile IoT in the 5G Future – [https://www.gsma.com/iot/resources/mobile-iot-in-the-5g-future/]

12. SIMCom Wireless. SIM7000 Series AT Command Manual – [https://simcom.ee/documents/SIM7000/SIM7000%20Series_AT%20Command%20Manual_V1.09.pdf]

13. SIMCom. SIM7000E Specification – [https://simcom.ee/documents/SIM7000E/SIM7000E_SPEC_2017-9-21.pdf]

14. SIMCom. SIM7000E Hardware Design – [https://simcom.ee/documents/SIM7000E/]

15. Quectel. SIM7000 Series GNSS Application Note – [https://www.quectel.com/product/sim7000]

16. Raspberry Pi Foundation. Raspberry Pi Documentation – [https://www.raspberrypi.com/documentation/]

17. Eco Energy Geek. Raspberry Pi Power Consumption Guide – [https://www.ecoenergygeek.com/raspberry-pi-power-consumption/]

18. Raspberry Pi Dramble. Power Consumption Benchmarks – [https://www.pidramble.com/wiki/benchmarks/power-consumption]

19. Mozilla Developer Network. HTTPS Overview – [https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview]

20. MQTT Essentials – HiveMQ Blog – [https://www.hivemq.com/mqtt-essentials/]

21. EMQX. MQTT vs HTTP – [https://www.emqx.com/en/blog/mqtt-vs-http]

22. OASIS. MQTT Version 3.1.1 – [https://mqtt.org]

23. JSON.org – [https://www.json.org/json-en.html]

24. Python Telegram Bot Library – [https://github.com/python-telegram-bot/python-telegram-bot]

25. Telegram Bot API Documentation – [https://core.telegram.org/bots/api]

26. InfoSecTrain. A Comprehensive Guide to IoT Security – [https://www.infosectrain.com/blog/a-comprehensive-guide-to-iot-security/]

27. Flask Web Framework – [https://flask.palletsprojects.com/]

28. FastAPI Web Framework – [https://fastapi.tiangolo.com/]

29. RFC 7252: The Constrained Application Protocol (CoAP) – Shelby Z., Hartke K., Bormann C. – [https://datatracker.ietf.org/doc/html/rfc7252]

30. RFC 793: Transmission Control Protocol – Postel J. – [https://datatracker.ietf.org/doc/html/rfc793]

31. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 – [https://datatracker.ietf.org/doc/html/rfc8446]

32. RFC 9293: Transmission Control Protocol (TCP) Update (2022) – [https://datatracker.ietf.org/doc/html/rfc9293]