Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«IoT засіб з чат-ботом для керування електричними пристроями»

08-54.БКР.027.00.000 ПЗ

Виконав: студент 4 курсу, групи 2КІ-21б
спеціальності 123 — «Комп'ютерна інженерія»
освітня програма — «Комп'ютерна інженерія»
Звездін Р. С.
Керівник к.т.н., доц. каф. ОТ
Савицька Л. А.
Рецензент к.т.н., доц. каф. ПЗ
Катєльніков Д. І.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«12» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень бакалавр

Галузь знань — 12 Інформаційні технології

Спеціальність — 123 Комп'ютерна інженерія

Освітня програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н.проф._____Азаров О. Д.

«21» березня 2025 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Звездіну Родіону Сергійовичу

1 Тема роботи: IoT засіб з чат-ботом для керування електричними пристроями, керівник роботи Савицька Людмила Анатоліївна, к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від «20» березня 2025 року №97.

2 Строк подання студентом роботи 05.06.2025

3 Вихідні дані до роботи: технічний опис розумної розетки на базі мікроконтролера, специфікації реле для комутації електричних кіл, технології розробки: мови програмування C++ та Python, бібліотеки для роботи з Wi-Fi, інтеграції з Telegram Bot API, бібліотеки для роботи з JSON та HTTP-запитами, платформа системи — ESP32, середовища розробки — Arduino IDE та PyCharm.

4 Зміст розрахунково-пояснювальної записки: вступ; аналіз і теоретичні засади розробки IoT-систем керування електричними пристроями; розробка IoT-системи розумної розетки; макетування та тестування пристрою, висновки.

5 Перелік графічного матеріалу: схема підключення компонентів IoT-пристрою, схема електрична принципова.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	к.т.н., доц. каф. ОТ Богомолов С. В.	21.03.25	13.05.25
Нормоконтроль	ас.. каф. ОТ Швець С. І.	14.05.25	30.05.25

7. Дата видачі завдання 21.03.2025 р.

8. Календарний план приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання	Підпис
1	Постановка задачі роботи	21.03.25	вик.
2	Пошук матеріалів та інформаційних джерел по розумних розетках та Telegram Bot API	21.03.25— 30.03.25	вик.
3	Аналіз існуючих технологій IoT та визначення вимог до створюваної системи	21.03.25— 30.03.25	вик.
4	Обґрунтування структури та принципів роботи системи розумної розетки	31.03.25— 13.04.25	вик.
5	Розробка апаратної частини розумної розетки та програмування мікроконтролера	14.04.25— 20.04.25	вик.
6	Створення та налаштування Telegram-бота, тестування системи	21.04.25— 27.04.25	вик.
7	Оформлення пояснювальної записки та ілюстративного матеріалу	28.04.25— 11.05.25	вик.
8	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.25	вик.

Студент Родіон ЗВЕЗДІН

Керівник роботи к.т.н., доц. Людмила САВИЦЬКА

АНОТАЦІЯ

УДК 004.9

Звездін Р. С. «IoT засіб з чат-ботом для керування електричними пристроями». Бакалаврська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна Інженерія, Вінниця: ВНТУ, 2025, с.

На укр.мові. Бібліогр.: 16 назв, рис.: 20, табл.: 6.

В даній бакалаврській роботі було проведено аналіз існуючих IoT рішень для керування електричними пристроями, виявлено їхні обмеження щодо інтерфейсу взаємодії та відсутність гнучкого керування через звичайні текстові команди. Розроблено IoT-пристрій з інтегрованим чат-ботом, який дозволяє керувати електричними пристроями, адаптуватися до потреб користувача та контролювати енергоспоживання в режимі реального часу, що підвищує зручність його використання.

Ключові слова: інтернет речей, чат-бот, електропристрої, розумний дім, текстове управління.

ABSTRACT

Zvezdin R. S. «IoT device with chat-bot for electrical appliances control». Bachelor's qualification work in specialty 123 - Computer Engineering, Vinnytsia: VNTU, 2025, p.

In Ukrainian. Bibliogr.: 16 titles, figures: 20, tables: 6.

This bachelor's thesis conducted an analysis of existing IoT solutions for electrical appliances control, identified their limitations regarding interaction interface and the lack of flexible control through ordinary text commands. An IoT device with integrated chat-bot was developed, which allows controlling electrical appliances, adapting to user needs and monitoring energy consumption in real time, thereby improving the convenience of its use.

Keywords: Internet of Things, chat-bot, electrical appliances, smart home, text control.

ЗМІСТ

ВСТУП

Актуальність автоматизації управління електричними пристроями у побуті стає все більш важливою у сучасному світі. Розвиток технологій Інтернету речей (IoT) відкриває нові можливості для створення розумних систем, які здатні покращити комфорт та енергоефективність побутових приладів. Сучасні користувачі стикаються з проблемою відсутності зручного та доступного способу дистанційного керування електричними пристроями. Існуючі на сьогоднішній день рішення здебільшого мають складний інтерфейс, потребують спеціалізованих додатків або коштують великих коштів. Це призводить до неефективного використання електроенергії, відсутності контролю за роботою приладів та неможливості автоматизації рутинних процесів.

Багато людей досить часто забувають вимикати електричні прилади, що призводить до більших витрат електроенергії та потенційних ризиків безпеки. Відсутність можливості контролювати стан пристроїв на відстані створює незручності, особливо коли потрібно керувати приладами, перебуваючи поза домом. Крім того, традиційні системи розумного дому часто вимагають складного налаштування та спеціальних знань.

Метою дослідження є розробка доступного та зручного IoT-пристрою з інтегрованим чат-ботом для керування електричними приладами через популярний месенджер Telegram. Це включає створення простого у використанні інтерфейсу, забезпечення надійності роботи системи та надання можливостей користувачам для ефективного контролю енергоспоживання.

Об'єктом дослідження є процес автоматизації керування електричними пристроями в побутових умовах.

Предметом дослідження є IoT-системи керування електричними приладами з використанням технологій віддаленого керування.

Задачі дослідження:

— аналіз існуючих IoT-рішень для керування електричними пристроями та виявлення їх обмежень;

— аналіз та вибір технологій для розробки IoT-пристрою на базі мікроконтролера;

— розробка апаратної частини розумної розетки з можливістю підключення до мережі Wi-Fi;

— створення програмного забезпечення для інтеграції з Telegram Bot API;

— тестування розробленого IoT-пристрою та Telegram-бота.

Новизна дослідження полягає у створенні доступного IoT-рішення, яке використовує популярний месенджер Telegram як інтерфейс керування, що робить систему зрозумілою для широкого кола користувачів без необхідності встановлення додаткових програм або складного налаштування.

Апробація результатів бакалаврської роботи: зроблено доповідь на LIV науково-технічну конференцію підрозділів ВНТУ [1].

Публікації за темою роботи: IoT засіб з чат-ботом для керування електричними пристроями / Звездін Р. С., Савицька Л. А., Богомолов С. В. //Матеріали Міжнародної науково-практичної Інтернет-конференції студентів, аспірантів та молодих науковців, ВНТУ 2025 р. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/24533/20297.

1 АНАЛІЗ І ТЕОРЕТИЧНІ ЗАСАДИ РОЗРОБКИ IoT-СИСТЕМ КЕРУВАННЯ ЕЛЕКТРИЧНИМИ ПРИСТРОЯМИ

1.1 Сучасні тенденції розвитку IoT у системах розумного дому

Термін «Інтернет речей» (Internet of Things, IoT) був запропонований Кевіном Ештоном у 1999 році, який був одним із трьох засновників Центру автоматичної ідентифікації Массачусетського університету (Auto-ID Center) [2]. Ештон ввів цей термін для того, щоб продемонструвати можливості радіочастотної ідентифікації (RFID), яка використовується в корпоративних системах поставок, щоб порахувати і відстежити товари без потреби людського втручання. Сьогодні, інтернет речей став популярним терміном для опису сценаріїв, у яких інтернет з'єднання і обчислювальна здатність поширюються на безліч об'єктів, пристроїв та повсякденних об'єктів [3]. Офіційне визначення Інтернету речей наведено в Рекомендації сектора стандартизації телекомунікацій Міжнародного союзу електрозв'язку (МСЕ-Т) Y.2060, згідно з яким IoT — глобальна інфраструктура інформаційного суспільства, що забезпечує передові послуги за рахунок організації зв'язку між фізичними або віртуальними об’єктами на основі існуючих та розвитку сумісних інформаційних та комунікаційних технологій [4].

Відомо, що першу у світі інтернет-річ створив один із авторів протоколу TCP/IP Джон Ромкі у 1990 році, коли він підключив до мережі Інтернет свій тостер. Але лише в 21 столітті, у зв'язку з бурхливим розвитком інфо-комунікаційних технологій, сформувалася концепція IoT і набула свого практичного втілення. Окрім поняття «річ», МСЕ-Т також використовує термін «пристрій», який означає обладнання та має обов'язкові функції комунікації, а також додаткові можливості, такі як сенсоринг і зондування, приведення в дію речей, збір, обробка та збереження даних. Основну роль у цій системі відіграють пристрої, які можуть збирати різноманітну інформацію та поширювати її через комунікаційні мережі різними способами, наприклад, через шлюзи, через мережу або напряму між собою, зображено на рисунку 1.1. Рекомендація Y.2060 визначає різні варіанти таких з'єднань, що свідчить про використання в IoT широкого спектра мережевих

технологій, зокрема глобальних і локальних мереж, а також бездротових ad-hoc та mesh-мереж. Дані, що зібрані пристроями, передаються у відповідні програми, а також команди від програмних додатків передають до пристроїв [4].

Інтернет речей можна записати у вигляді наступної формули:

IoT = Сенсори (датчики) + Дані + Мережі + Послуги

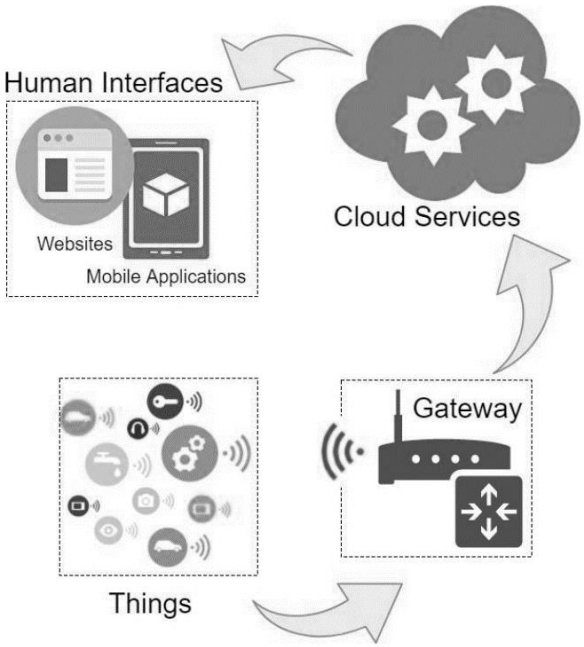

Рисунок 1.1 — Чотири основні компоненти IoT

Розумні технології та Інтернет речей все більше проникають у різні сфери життя та економіки, а їхній потенціал тільки збільшується. Під терміном «Інтернет речей» розуміється сукупність різноманітних приладів, датчиків, пристроїв, об'єднаних у мережу за допомогою каналів зв'язку, що використовують різні протоколи взаємодії між собою та єдиний протокол доступу до глобальної мережі Інтернет, також розглянемо приклади використання IoT:

— розумні будинки: IoT допомагає людям керувати побутовою технікою через мобільні додатки, економити енергію та покращувати безпеку;

— розумні транспортні системи: у мегаполісах, таких як Лондон та Сінгапур, діють системи розумних світлофорів та дорожніх датчиків, що допомагають оптимізувати потоки трафіку, зменшити кількість заторів та знизити об’єм викидів вуглекислого газу;

— розумні ферми: сільське господарство також стає високотехнологічним, оскільки IoT-системи широко використовуються для контролю вологості ґрунту, рівню освітленості та внесення добрив, завдяки цьому підвищується врожайність [5].

На початку 2000-х, із стрімким розвитком Інтернету та мереж із пакетною комутацією, телекомунікаційне співтовариство створило та впровадило нову концепцію комунікацій мережі наступного покоління (NGN, Next Generation Networks). Технології NGN пройшли шлях еволюції від гнучких комутаторів до мультимедійних підсистем зв'язку IMS (IP Multimedia Subsystem) та бездротових мереж LTE (Long Term Evolution). Спочатку NGN орієнтувалися виключно на людей як основних користувачів, що обмежувало кількість абонентів чисельністю населення Землі. Однак останнім часом значного поширення набули технології радіочастотної ідентифікації (RFID), бездротових сенсорних мереж (WSN), комунікацій малого радіусу дії (NFC) та міжмашинної взаємодії (M2M). Їх інтеграція з Інтернетом забезпечує простий зв'язок між технічними пристроями, кількість яких може бути величезною. Згідно з прогнозами компанії Cisco, ми все швидше наближаємося до концепції «Всеохоплюючого Інтернету» (Internet of Everything, IoE), де «неживі» предмети зможуть аналізувати контекст, використовувати розширені обчислювальні ресурси та сенсорні можливості. Cisco визначає IoE як інтеграцію людей, процесів, даних і речей, що підвищує ефективність мережевих з'єднань до безпрецедентного рівня [4].

Розглянемо основні проблеми впровадження IoT-технологій.

Відсутність єдиних стандартів для інтернету речей, що ускладнює можливість інтеграції запропонованих рішень на ринку та, відповідно, появу нових.

IoT використовує єдиний протокол взаємодії, що забезпечує рівноправний доступ кожного вузла мережі до надання своїх сервісів. Проте реалізація концепції Інтернету речей зіткнулася з обмеженнями протоколу IPv4, ресурси вільних мережевих адрес якого практично вичерпано. Впровадження нової версії протоколу IPv6 вирішує цю проблему, шляхом наближення IoT до

повномасштабної реалізації. Перехід від IPv4 до IPv6 супроводжується значними фінансовими витратами для телекомунікаційних операторів та провайдерів, оскільки цей процес потребує модернізації мережевого обладнання. Незважаючи на ці труднощі, впровадження IPv6 відкриває нові можливості для розвитку Інтернету речей, розширюючи адресний простір та підвищуючи ефективність комунікації між пристроями.

Проблема захисту даних у глобальних мережах та доступу до інформації, яка збирається «розумними речами» (відстеження місцезнаходження людей та їх власності).

Датчики повинні навчитися отримувати енергію із навколишнього середовища, а не працювати від батарейок, як це відбувається зараз, тобто стати автономними.

Необхідність зміни загальноприйнятих та перевірених бізнес-процесів та стратегій, що може призвести до значних фінансових витрат та ризиків.

У структурі системи також виділяються два вертикальні рівні - рівень управління та рівень безпеки охоплюють усі чотири горизонтальні рівні архітектури. Вертикальний рівень експлуатаційного управління включає функції управління наслідками відмов, мережевими можливостями, конфігурацією, безпекою та обробкою даних для білінгу. Основними об'єктами управління є пристрої, локальні мережі, їхня топологія, а також контроль трафіку та перевантаження мереж. Функціональність вертикального рівня безпеки залежить від конкретного горизонтального рівня. Для рівня підтримки програм та послуг передбачено механізми AAA, антивірусний захист та перевірку цілісності даних. На мережевому рівні реалізуються авторизація, аутентифікація та захист інформації у протоколах сигналізації. Для рівня пристроїв забезпечуються функції авторизації, аутентифікації, контролю доступу та захисту конфіденційності даних.

Розглянемо систему об’єднаних пристроїв, яка має назву — розумний дім.

Розумний дім — система об'єднаних пристроїв, здатних виконувати дії та вирішувати певні повсякденні завдання без участі людини. Зазвичай усі пристрої

підключені до локальної мережі або Інтернету й централізовано керуються через спеціальний хаб, або сервер [6].

Розумний будинок може мати декілька способів керування:

— за допомогою пульта;

— додатку на мобільному пристрої;

— голосових команд.

Завдяки розумному дому всі системи в будівлі можуть працювати разом, виконуючи завдання відповідно до заданих сценаріїв або змін у навколишньому середовищі.

Функції, що забезпечують комфорт, безпеку та ефективність житлового простору у системі «розумного дому»:

Керування освітленням, що дозволяє налаштувати автоматичне вмикання світла за графіком. Система підтримує оптимальний рівень освітлення, дозволяє регулювати яскравість та змінювати колір підсвітки. Зручна функція — вимкнення усього світла однією кнопкою, а також існує сценарій під назвою "підготовка до сну", що поступово знижує яскравість освітлення ввечері, готуючи власників житла до відпочинку.

Клімат-контроль, що керує температурою у всьому помешканні або окремо в кожній кімнаті та автоматично підтримує задану власником температуру. Під час відсутності мешканців — переходить в економний режим, а також регулює температуру за графіком, наприклад, під час сну.

Безпека, що дозволяє отримати доступ до камер відеоспостереження через смартфон та у разі небезпеки сповістити охоронну службу або поліцію. Також існує функція, яка має назву «імітація присутності», під час роботи якої система вмикає та вимикає світло в різних кімнатах у разі відсутності власників помешкання.

Домофон та автоматика входу, що дозволяє контролювати двері, ворота та вікна дистанційно через додаток, а саме відчиняти та зачиняти їх. Доступ через домофон можливий за допомогою карти-ключа, телефону, відбитка пальця чи розпізнавання обличчя.

Мультимедіа, що дозволяє керувати всім мультимедійним обладнанням, наприклад, закривання штор, вимикання світла, опускання екран та вмикання проєктора всього лише одним натиском кнопки у смартфоні.

Керування шторами та жалюзі, що дозволяє автоматично регулювати їх положення залжено від сонячного світла, погоди чи часу доби.

Управління прибудинковою територією, яке допомагає оптимізувати роботу басейну або системи поливу, шляхом регулювання температури води, її подачу та фільтрацію, забезпечувати полив за розкладом з урахуванням погодних умов. Також зі смартфону можна керувати освітленням і на території будинку.

Додаткові функції, які дозволяють дистанційно перекривати воду за допомогою смартфону або відключати напругу з певної розетки, якщо власник помешкання забув вимкнути праску або інший пристрій з розетки [7].

Розвиток «розумних» технологій у житловому секторі стає все більш значущим завдяки штучному інтелекту (ШІ) та інтернету речей (IoT). Прогнозується, що впродовж наступних декількох років системи «розумного дому» стануть ще більш адаптивними, здатними передбачати поведінку мешканців та автоматично налаштовувати роботу побутових приладів. Завдяки використанню штучного інтелекту системи у «розумних будинках» зможуть отримати можливість аналізувати споживання електроенергії, прогнозувати витрати та оптимізувати режими роботи будь-якого з пристроїв [8].

Найближчими роками очікується активне зростання попиту на модульні системи «розумного дому», що дозволять забезпечити мешканцям будинку самостійно обирати функції від — базового контролю безпеки до автоматизації всього житла. Такий підхід допоможе зменшити стартові інвестиції та забезпечити поступове розширення функціоналу, що зробить технологію більш доступною для широкого колу користувачів.

Одна з ключових тенденцій майбутнього — інтеграція «розумного дому» з технологіями «розумних міст» та відновлюваними джерелами енергії. Це дозволить взаємодіяти з електромережами та транспортною інфраструктурою,

сприяючи більш ефективному використанню ресурсів та спрощенню управління житловими об'єктами [8].

При виборі системи «розумний дім» необхідно враховувати бюджет, технічні особливості житла та особисті потреби. Важливими критеріями є функціональність, сумісність з іншими пристроями, простота використання та наявність технічної підтримки. Крім того, варто звернути увагу на можливість оновлення та розширення системи в майбутньому [8]. Базові функції, такі як енергоефективність, автоматизація освітлення та системи безпеки, повинні бути пріоритетними. Для додаткового комфорту варто звернути увагу на автоматизацію побутових приладів та інтелектуальне управління кліматом.

Ціна системи «розумного дому» напряму залежить від її функціоналу: базовий функціонал доступний для більшості користувачів, тоді як комплексні системи з розширеними можливостями коштують в рази дорожче. Проте автоматизація здатна зменшити витрати на електроенергію до 30%, що робить такі інвестиції економічно доцільними у довгостроковій перспективі. Незважаючи на стартові витрати, зменшення комунальних платежів забезпечує швидке повернення вкладень [8].

Отже, сучасні тенденції розвитку IoT у системах «розумного дому» спрямовані на підвищення комфорту, енергоефективності та безпеки. Однією з ключових технологій є адаптивні системи, які аналізують поведінку мешканців і автоматично налаштовують параметри житла, такі як освітлення, температура та вентиляція задля їх комфортного перебування у помешканні. Зростає популярність голосових асистентів і систем керування жестами, що дозволяє користувачам легко взаємодіяти з пристроями без потреби у фізичних кнопках чи мобільних застосунках. Інтеграція loT із відновлюваними джерелами енергії, такими як сонячні панелі та системи накопичення енергії, сприяє зменшенню залежності від традиційних електромереж. Автоматизація управління водопостачанням та опаленням дозволяє скоротити витрати ресурсів та забезпечити раціональне використання енергії. Розвиток систем безпеки, зокрема розпізнавання обличчя та біометричної аутентифікації, допомагає захистити будинок від несанкціонованого

доступу. Усе більше уваги приділяється взаємодії «розумного дому» із системами «розумного міста», що дає можливість покращити транспортну інфраструктуру та енергоспоживання на рівні громади. Використання штучного інтелекту для аналізу даних про споживання електроенергії дозволяє прогнозувати витрати та оптимізувати роботу побутових приладів. Також розвивається концепція модульних систем, які дозволяють поступово додавати нові функції відповідно до потреб.

1.2 Архітектура IoT-систем: рівні, компоненти, топології

Інтернет речей (IoT) концептуально належить до мереж наступного покоління, тому його архітектура має багато спільного з чотиришаровою моделлю NGN (Next Generation Network, NGN). IoT об’єднує різні інфо-комунікаційні технології, які забезпечують його функціонування, а його архітектура демонструє взаємозв'язок між цими технологіями.

Структура IoT включає чотири основні функціональні рівні, кожен з яких відіграє важливу роль у функціонуванні системи, зображено на рисунку 1.2.

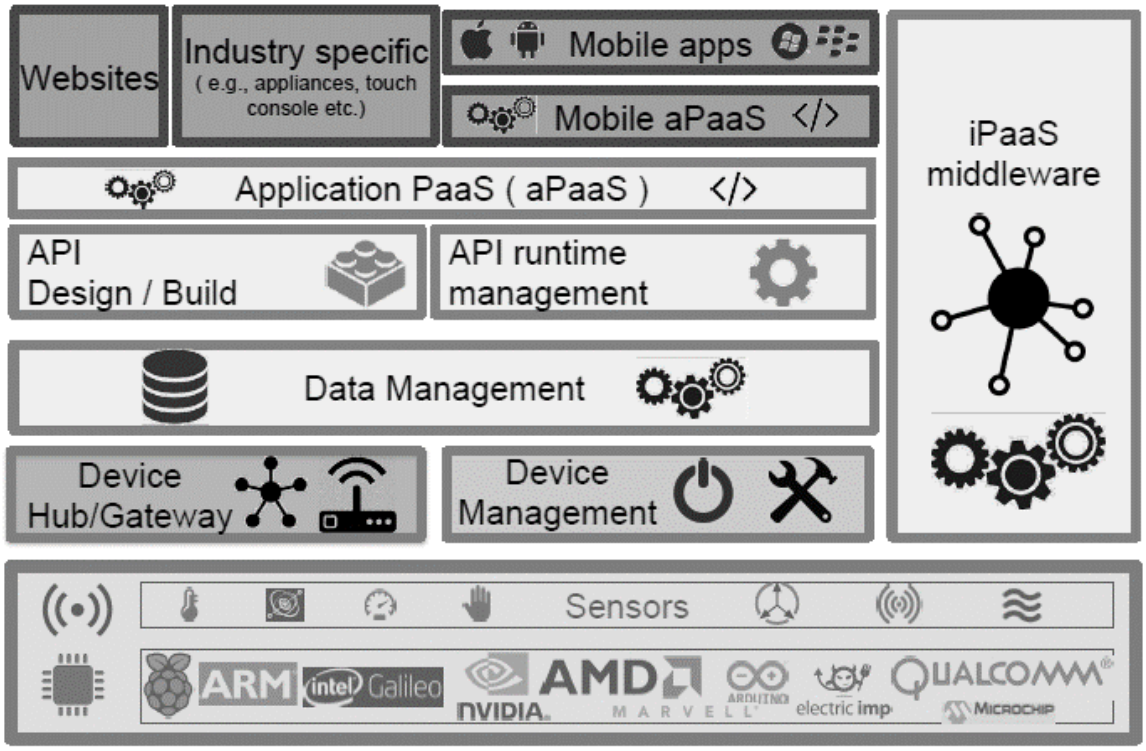

Рисунок 1.2 — Чотири основні функціональні рівні структури IoT

Роль архітектури IoT полягає у тому, щоб:

— запропонувати мережевим адміністраторам контрольний список для оцінки функціональності та повноти різноманітних пропозицій, висунутих різними постачальниками;

— надати розробникам вказівки щодо необхідних функцій, необхідних для Інтернету речей, і того, як вони повинні взаємодіяти;

— діяти як основа для стандартизації, яка сприяє сумісності та скорочує витрати.

Рівень сенсорів та сенсорних мереж — це найнижчий рівень архітектури IoT, який складається з «розумних» об'єктів, інтегрованих з сенсорами (датчиками). Сенсори реалізують поєднання фізичного та віртуального (цифрового) світів, забезпечуючи збирання та обробку інформації в реальному часі. Мініатюризація апаратних сенсорів сприяла зменшенню їхніх фізичних розмірів, що дозволило інтегрувати їх безпосередньо в об'єкти фізичного світу. Існує багато типів сенсорів, призначених для різних завдань, зокрема вимірювання температури, тиску, швидкості руху та місця розташування. Деякі сенсори оснащені невеликою пам'яттю, що дає змогу записувати кілька результатів вимірювань. Сенсори здатні вимірювати фізичні параметри контрольованого об'єкта або явища та перетворювати їх у сигнал, який може бути прийнятий відповідним пристроєм. Вони класифікуються за своїм призначенням, наприклад, сенсори навколишнього середовища, сенсори для моніторингу стану тіла, сенсори для побутової техніки та транспортних засобів [8].

Більшість сенсорів потребує з'єднання з агрегатором сенсорів (шлюзом), який може бути реалізований через локальні обчислювальні мережі (LAN, Local Area Network), такі як Ethernet і Wi-Fi, або персональні мережі (PAN, Personal Area Network), зокрема ZigBee, Bluetooth Та ультраширокосмуговий бездротовий зв'язок на малих відстанях (UWB, Ultra-Wide Band). Для сенсорів, які не потребують підключення до агрегатора, зв'язок із серверами та додатками здійснюється через глобальні бездротові мережі WAN, такі як GSM, GPRS і LTE. Сенсори з низьким енергоспоживанням і невеликою швидкістю передачі даних

формують бездротові сенсорні мережі (WSN, Wireless Sensor Network), які стають дедалі популярнішими. WSN дозволяють використовувати значно більшу кількість сенсорів із підтримкою роботи від батарей, що забезпечує їх ефективне функціонування на великих територіях.

Рівень шлюзів і мереж — це великий обсяг даних, що створюється завдяки численним мініатюрним сенсорам на першому рівні IoT, і потребує надійної та високопродуктивної провідної або бездротової мережевої інфраструктури для ефективного транспортування інформації. Існуючі мережі зв'язку, що використовують різні протоколи, можуть підтримувати міжмашинні комунікації (M2M) та їхні додатки. Для забезпечення широкого спектра послуг і додатків в Io необхідно інтегрувати різні мережеві технології та протоколи доступу в єдину гетерогенну конфігурацію. Ці мережі повинні відповідати вимогам щодо якості передачі даних, зокрема мінімізувати затримки, забезпечувати достатню пропускну спроможність і гарантувати безпеку. Конвергентна мережева інфраструктура формується шляхом об'єднання різнорідних мереж у єдину платформу, що дозволяє ефективно керувати ресурсами. Конвергентний абстрактний мережевий рівень IoT через відповідні шлюзи забезпечує незалежний і спільний доступ користувачів до ресурсів однієї мережі, зберігаючи конфіденційність, безпеку та продуктивність.

Сервісний рівень включає набір інформаційних послуг, спрямованих на автоматизацію технологічних і бізнес-операцій у IoT. Він забезпечує підтримку операційної та бізнес-діяльності (OSS/BSS - Operation Support System / Business Support System), аналітичну обробку даних, включаючи статистичний аналіз, інтелектуальну обробку інформації, аналіз текстів та прогностичну аналітику. Крім того, сервісний рівень відповідає за зберігання даних, інформаційну безпеку, управління бізнес-правилами (BRM — Business Rule Management) та бізнес-процесами (BPM — Business Process Management). Завдяки цим функціям IoT системи можуть ефективно обробляти інформацію, оптимізувати робочі процеси та підвищувати рівень автоматизації бізнесу.

Рівень додатків — це четвертий рівень архітектури IoT, який включає різноманітні програмні рішення, призначені для різних галузей промисловості та сфер діяльності. До таких сфер належать енергетика, транспорт, торгівля, медицина, освіта та багато інших. Додатки на цьому рівні можуть бути «вертикальними», тобто спеціалізованими для конкретної галузі. Наприклад, у медицині це можуть бути системи моніторингу стану пацієнтів, а в енергетиці — рішення для управління розподілом електроенергії. Окрім вертикальних, існують «горизонтальні» додатки, які застосовуються в різних секторах економіки. До них належать системи управління автопарком, відстеження активів, моніторинг логістичних процесів та інші універсальні рішення. Завдяки рівню додатків IT забезпечує ефективну інтеграцію технологій у бізнес-процеси, автоматизує управління ресурсами та сприяє підвищенню продуктивності в різних сферах діяльності.

1.3 Огляд платформ для побудови IoT-пристроїв: ESP8266, ESP32, Arduino

У сучасному світі стрімкого розвитку технологій Інтернету речей (IoT) ключовим кроком при створенні "розумних" пристроїв залишається вибір належної апаратної основи, яка виконуватиме функції головного обчислювального елемента. Для систем IoT така платформа має виконувати різноманітні завдання: опрацьовувати дані від датчиків, виконувати розрахунки згідно з запрограмованими алгоритмами, керувати такими компонентами як релейні модулі та індикатори, а також підтримувати комунікацію з іншими пристроями чи безпосередньо з користувачем через різні канали зв'язку. Тобто, мікроконтролерна платформа стає "серцем" IoT-рішення, і саме від неї залежать ключові характеристики системи: швидкість обробки даних, споживання електроенергії, наявний функціонал, можливості подальшого розширення і загальна стабільність роботи пристрою. При розробці компактних, економічних щодо споживання енергії та доступних за вартістю пристроїв (особливо тих, що працюють від батарейок або мають обмежені обчислювальні ресурси) розробники частіше обирають саме мікроконтролери, а не повнофункціональні мікропроцесори.

Мікроконтролер — це мікросхема, яка використовується для керування електронними пристроями. Використання однієї мікросхеми значно знижує розміри, енергоспоживання і вартість пристроїв, побудованих на базі мікроконтролерів. Мікроконтролери можна зустріти в багатьох сучасних приладах, таких як телефони, пральні машини, вони відповідають за роботу двигунів і систем гальмування сучасних автомобілів, з їх допомогою створюються системи контролю і системи збору інформації. На основі мікроконтролерів проєктують та створюють вимірювальні пристрої, системи керування об'єктами та процесами, вони є основою охоронних, протипожежних систем, домофонів та сигналізацій [10].

До основних переваг мікроконтролерів можна віднести наступні:

— наявність вбудованих периферійних модулів (GPIO, АЦП, UART, SPI, I2C, таймери, PWM);

— низьке енергоспоживання, що є критичним для автономних пристроїв;

— компактність, що дозволяє використовуватися у форм-факторі DIP або SMD;

— відсутність ОС, що надає можливість миттєвого запуску пристрою.

— Серед найбільш популярних у розробників рішень для побудови IoT-систем виділяються наступні платформи:

— ESP8266 — один із перших масових мікроконтролерів із вбудованим Wi-Fi-модулем;

— ESP32 — вдосконалена версія ESP8266 з додатковою продуктивністю, Bluetooth та покращеною енергоефективністю;

— Arduino — універсальна платформа на базі AVR-мікроконтролерів (переважно ATmega328P), яка підходить для навчання, прототипування та простих електронних проектів.

Порівняння характеристик цих трьох платформ наведено у таблиці 1.

Таблиця 1 — порівняння характеристик платформ ESP8266, ESP32, Arduino.

Характеристика	**ESP8266**	**ESP32**	**Arduino Uno**
Процесор	32-бітний, 80 МГц	32-бітний, 2 ядра, до 240 МГц	8-бітний, 16 МГц
Пам'ять (RAM)	96 КБ	520 КБ	2 КБ
Флеш-пам'ять	512 КБ – 4 МБ	4 МБ – 16 МБ	32 КБ
GPIO	17	До 34	14
Аналогові входи	1 (10 біт)	До 18 (12 біт)	6 (10 біт)
Wi-Fi	Так	Так	Ні
Bluetooth	Ні	Так	Ні
Напруга живлення	3,3 В	3,3 В	5 В
Підтримка IDE	Arduino IDE, Lua, MicroPython	Arduino IDE, ESP-IDF, MicroPython	Arduino IDE

Отже, вибір між ESP8266, ESP32 та Arduino залежить від конкретних вимог проєкту. Наприклад, ESP8266 підходить для простих IoT-пристроїв, де потрібен Wi-Fi, ESP32 підійде для більш складних задач, що вимагають високої продуктивності, Bluetooth та розширених інтерфейсів, в той час як Arduino є чудовим вибором саме для навчання та прототипування.

1.4 Протоколи зв’язку в IoT: MQTT, HTTP, HTTPS, WebSockets

Однією з головних особливостей IoT є здатність пристроїв взаємодіяти між собою та з користувачем. Для цього вони повинні передавати й отримувати дані через цифрові канали зв’язку. Проведемо порівняння: якщо мікроконтролер — це

так званий «мозок» пристрою, то мережевий протокол це мова, якою він обмінюється інформацією з іншими пристроями або сервісами, забезпечуючи її ефективний обмін.

Вибір правильного протоколу зв'язку в IoT-системі є критично важливим, оскільки він впливає на такі параметри, як:

— швидкість обміну даними;
— енергоефективність пристрою;
— стійкість до втрат зв’язку;
— безпека передавання даних;
— простота реалізації та можливість масштабування системи.

В залежності від призначення пристрою, способу його керування, наприклад, вручну через бота або автоматично, та типу взаємодії (команди чи потік даних у реальному часі) найпоширенішими протоколами зв’язку є MQTT, HTTP, HTTPS та WebSockets. Розглянемо кожен з протоколів детальніше.

1.4.1 Протокол MQTT

Протокол MQTT був винайдений у 1999 році для використання в нафтогазовій галузі. Інженерам потрібен був протокол з мінімальною пропускною спроможністю та мінімальними втратами заряду батареї для моніторингу нафтопроводів через супутник. Спочатку протокол був відомий як Message Queuing Telemetry Transport через продукт IBM серії MQ, який вперше підтримав його початкову фазу. У 2010 році IBM випустила MQTT 3.1 вже як вільний та відкритий протокол, який може впроваджувати будь-хто, а в 2013 році він був переданий на підтримку специфікаційного органу Організації з просування стандартів структурованої інформації (OASIS). У 2019 році OASIS випустила оновлену версію 5 MQTT [11].

Протокол MQTT працює за принципами моделі «публікація-підписка» (publish/subscribe). У традиційному мережевому спілкуванні клієнти та сервери спілкуються один з одним напряму. Клієнти запитують ресурси або дані у сервера, сервер обробляє їх і надсилає відповідь. Однак, MQTT використовує модель

«публікація/підписка», щоб відокремити відправника повідомлення (видавця) від одержувача повідомлення (підписника). Замість цього третій компонент, який називається брокером повідомлень, обробляє зв'язок між видавцями та підписниками. Завдання брокера — це фільтрація всіх вхідних повідомлень від видавців і правильне розподілення їх серед підписників. Брокер розділяє видавців і підписників наступним чином:

— просторове розділення;

— часове розділення;

— розділення синхронізації.

Просторове розділення означає, що видавець і підписник не знають мережеве розташування один одного і не обмінюються такою інформацією, як IP-адреси або номери портів. Це підвищує рівень безпеки та конфіденційності даних у системі.

Часове розділення передбачає, що видавець і підписник не працюють одночасно та можуть бути відключені від мережі в різний час. Це дозволяє обробляти інформацію незалежно від безперервної доступності всіх учасників взаємодії.

Розділення синхронізації забезпечує можливість надсилання і отримання повідомлень без взаємних затримок. Наприклад, підписник не зобов’язаний чекати, поки видавець надішле повідомлення – дані надходять у зручний для кожного час.

Переваги протоколу MQTT:

— низьке споживання ресурсів, що ідеально підходить для пристроїв з обмеженими можливостями;

— мінімальний обсяг трафіку завдяки компактному формату повідомлень;

— підтримка різних рівнів якості обслуговування (QoS), що забезпечує доставку повідомлень;

— можливість обміну повідомленнями без необхідності постійного з’єднання.

Недоліки протоколу MQTT:

— потрібен центральний сервер (брокер) для маршрутизації повідомлень;

— не всі платформи та мови програмування мають вбудовану підтримку MQTT.

Сфери застосування протоколу MQTT:

— сенсори та датчики: передача даних про температуру, вологість, тиск тощо;

— розумні будинки: керування освітленням, опаленням, безпекою;

— промислові системи: моніторинг обладнання та процесів у реальному часі.

1.4.2 Протокол HTTP

HTTP — це протокол для отримання ресурсів, таких як HTML-документи. Він лежить в основі будь-якого обміну даними в Інтернеті та є клієнт-серверним протоколом. Це означає, що запити ініціюються одержувачем, як правило, веб-браузером.

Повноцінний документ зазвичай складається з таких ресурсів, як текстовий вміст, інструкції з верстки, зображення, відео, скрипти тощо. Окремий веб-документ складається з декількох ресурсів з різних серверів. Клієнти і сервери спілкуються, обмінюючись окремими повідомленнями (на відміну від потоку даних). Повідомлення, надіслані клієнтом, називаються запитами, а повідомлення, надіслані сервером у відповідь, називаються відповідями [12].

API багатьох програмних продуктів передбачає використання протоколу HTTP для передавання даних, які можуть мати будь-який формат, зокрема XML або JSON. Як правило, передавання даних за допомогою HTTP здійснюється через TCP/IP-з'єднання. Серверне програмне забезпечення зазвичай використовує TCP-порт 80. У випадку, якщо порт не вказаний явно, клієнтське програмне забезпечення, як правило, за замовчуванням відкриває HTTP-з'єднання саме через 80-й порт, хоча можливе використання і будь-якого іншого порту.

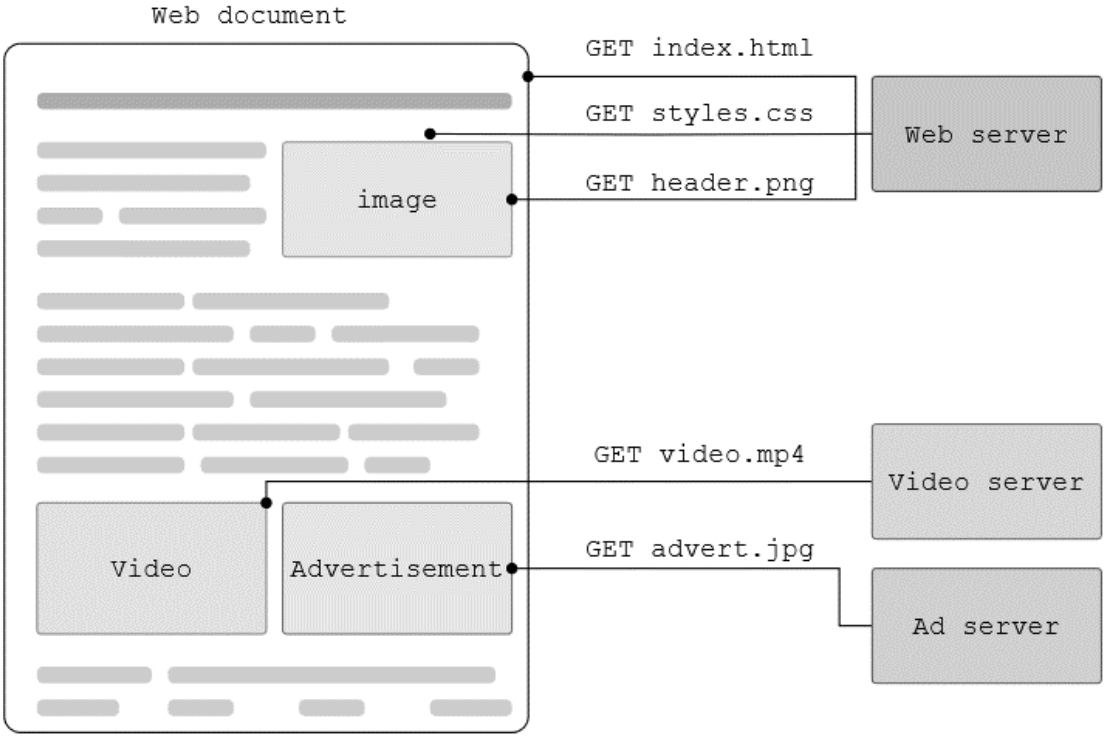

Рисунок 1.3 — Схема запитів HTTP

У ролі клієнта зазвичай виступає браузер, а у ролі сервера виступає веб-сервер — спеціальна програма на фізичному сервері, де зберігається сайт. Наприклад, для того щоб відобразити будь-яку сторінку, браузер надсилає серверу запит на отримання HTML-документа з її вмістом. Далі браузер вивчає документ і запитує додаткові файли, необхідні для відображення сторінки: стилі елементів, зображення, скрипти, зображено на рисунку 1.3. Наступним кроком браузер збирає всю інформацію разом, як зазначено в HTML-документі, та виводить на екран комп’ютера результат.

Інший приклад — коли браузер надсилає якісь дані серверу. Наприклад, логін і пароль для входу до облікового запису. Щоб сформувати HTML-документ з головною сторінкою облікового запису, сервер перевіряє деталі входу в спеціальній таблиці всередині бази даних, де зберігаються дані всіх користувачів. Якщо у таблиці є користувач із зазначеними даними, сервер формує HTML-документ і відправляє браузеру. У випадку відсутності такого користувача або параметри входу неправильні, в середині сайту прописаний спеціальний сценарій

на цей випадок — надіслати повідомлення з помилкою, яку браузеру необхідно відобразити на екрані.

Переваги протоколу HTTP:

— широко підтримується всіма веб-браузерами та серверами;

— легко реалізується та інтегрується з існуючими веб-сервісами;

— використовує стандартні методи (GET, POST, PUT, DELETE) для взаємодії з ресурсами.

— Недоліки протоколу HTTP:

— кожен запит є незалежним, що може призводити до надмірного трафіку;

— затримка, через необхідність постійного запиту на сервер;

— відсутність безпеки, оскільки дані передаються у відкритому вигляді.

— Сфери застосування протоколу HTTP:

— інтеграція IoT-пристроїв з веб-сервісами;

— моніторинг та передача даних про стан пристроїв на сервер для подальшого аналізу.

1.4.3 Протокол HTTPs

HTTPs — це розширення для протоколу HTTP, що робить його безпечним. HTTPs розшифровується як HyperText Transfer Protocol Secure — безпечний протокол передачі гіпертексту. У протоколі HTTP дані передаються у відкритому вигляді, що створює ризик перехоплення зловмисниками конфіденційної інформації. Протокол HTTPs вирішує цю проблему, шляхом шифрування даних. Безпека досягається за рахунок об’єднання протоколу HTTP з криптографическим протоколом TLS [13]. Перед початком передачі даних відбувається обмін криптографічними ключами між клієнтом і сервером. Це дозволяє створити захищений канал, через який інформація передається у зашифрованому вигляді. Встановлення безпечного з’єднання протоколу HTTPs схематично зображено на рисунку 1.4.

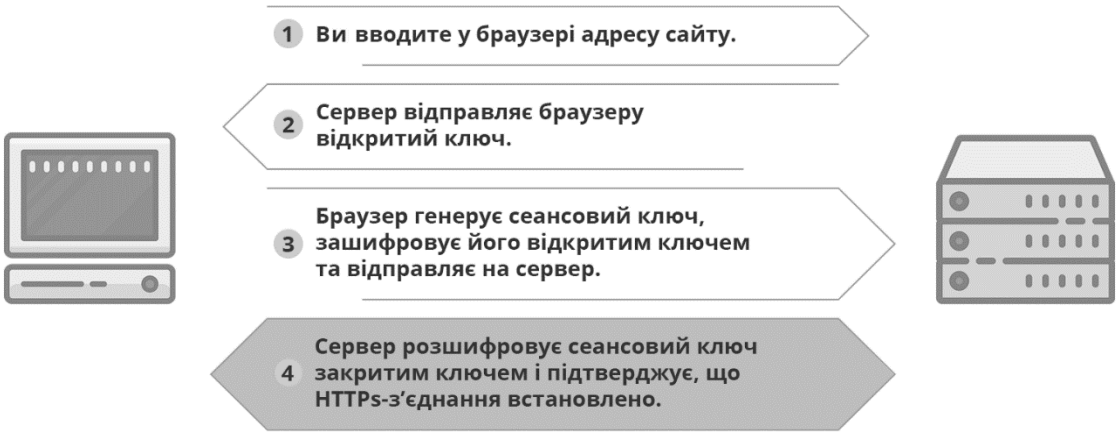

Рисунок 1.4 — встановлення безпечного з'єднання протоколу HTTPs

Протокол HTTPs передбачає, що під час встановлення з'єднання «клієнт-сервер» обидва домовляються про використання тимчасового ключа, за допомогою якого будуть зашифровувати та розшифровувати повідомлення. Цей ключ має назву «сеансовий», оскільки діє протягом поточного сеансу на сайті. Під час кожного нового сеансу на сайті генерується новий ключ. Метод, у якому для зашифрування та розшифровування повідомлень використовується один ключ, називають «симетричне шифрування». Його задача — це безпечно передати ключ від однієї сторони до іншої.

У роботі протоколу HTTPs для цього використовується ще один метод, який має назву «асиметричне шифрування». Його суть полягає в тому, щоб використовуапти два ключі: один для зашифровування повідомлень (відкритий ключ), інший — для розшифровування (закритий ключ). Обома ключами володіє одна сторона — сервер. Закритий ключ є секретним, а відкритий можна вільно передавати клієнтам, щоб вони з його допомогою зашифровували повідомлення та надсилали їх на сервер [13].

Переваги протоколу HTTPs:

— захищає дані від перехоплення та підробки;

— забезпечує впевненість у справжності сайту або сервісу;

— підтримується всіма сучасними браузерами та пристроями.

Недоліки протоколу HTTPs:

— шифрування та дешифрування даних вимагає додаткових обчислювальних ресурсів;

— потребує отримання та встановлення SSL-сертифікатів.

Сфери застосування протоколу HTTPs:

— інтернет-магазини, захист платіжних даних користувачів;

— онлайн-сервіси, забезпечення безпеки особистих даних;

— IoT-пристрої, безпечна передача конфіденційної інформації, наприклад, в системах розумного дому.

1.4.4 Протокол WebSocket

WebSocket — це протокол, описаний у специфікації RFC 6455, забезпечує спосіб обміну даними між браузером і сервером через постійне з’єднання. Дані можна передавати в обох напрямках у вигляді «пакетів», без розриву з’єднання та додаткових HTTP-запитів. WebSocket підходить для сервісів, що потребують безперервного обміну даними, наприклад онлайн-ігри, системи торгівлі в реальному часі тощо [14]. Процес взаємодії через WebSocket зображено на рисунку 1.5.

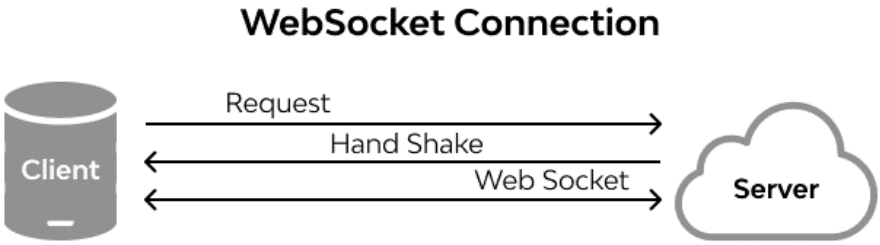

Рисунок 1.5 — Процес взаємодії через WebSocket

WebSocket-з'єднання складається з двох кінцевих точок: одна з них належить браузеру, інша — серверу. Браузер ініціює запит на з'єднання, а клієнт надсилає заголовок оновлення зі значенням «WebSocket». Потім сервер відповідає 101 Switching Protocols та заголовками оновлення, повідомляючи, що він приймає запит

на оновлення з'єднання від клієнта. Після цього обидві сторони можуть надсилати дані в будь-який час, допоки одна зі сторін не вирішить припинити з'єднання.

Після завершення запиту встановлюється з'єднання WebSocket, і через цей канал можна надсилати повідомлення в обох напрямках. WebSocket-повідомлення є важливим аспектом протоколу WebSocket, який забезпечує повнодуплексні канали зв'язку через одне TCP-з'єднання.

WebSocket-повідомлення надсилаються через встановлене WebSocket-з'єднання у двійковому форматі. Кожне повідомлення включає в себе заголовок і дані корисного навантаження. Заголовок містить метадані про повідомлення, включаючи його тип (текстове або двійкове). Текстове повідомлення містить символи UTF-8, тоді як двійкове повідомлення кодується за допомогою масиву байт, тому воно може передаватися без додаткової обробки браузером або сервером.

API WebSocket — це API браузера, який надає механізм для встановлення з'єднання між клієнтським сервером і віддаленим хостом через мережу Інтернет. Це з'єднання є відкритим, що дозволяє даним надсилатися і прийматися безперервно. З'єднання WebSocket встановлюється за допомогою протоколу, схожого на HTTP-запит на оновлення. Специфікація протоколу WebSocket визначає конкретний формат кадрів даних, які використовуються для зв'язку між клієнтом та сервером. Клієнт може надсилати необмежену кількість кадрів даних під час одного з'єднання, кожен з яких ідентифікується типом кадру, зареєстрованим в IANA.

API WebSocket дозволяє розробникам використовувати одне з'єднання сокетів як для початкових запитів, так і для відповідей, а також для надсилання клієнту push-повідомлень від сервера в будь-який час впродовж існування такого з'єднання. На відміну від HTTP-з'єднань, сервер може надсилати дані клієнту не чекаючи на його запит до нього. Відмінності протоколу WebSocket від протоколу HTTP наведено у таблиці 2.

Таблиця 2 — відмінності протоколу WebSocket від протоколу HTTP.

HTTP	WebSocket
Однонаправлений протокол; лише клієнт може відправляти запити, а сервер може тільки відповідати.	Двонаправлений протокол, який дозволяє будь-якій стороні відправляти повідомлення після встановлення з'єднання.
З'єднання нетривале, за замовчуванням закривається після відповіді сервера.	З'єднання підтримується до завершення комунікації.
Під час комунікації витрачається час на запит клієнта, навіть якщо важлива лише відповідь сервера.	Сервер може відправити повідомлення, не чекаючи на запит клієнта.
Обов'язкове використання хедерів у повідомленнях.	Після встановлення з'єднання у хедерах немає потреби, що дозволяє значно зменшити розмір кожного повідомлення.
Підтримує кешування.	За замовчуванням не підтримує кешування.

Socket.IO — це один з найпопулярніших інструментів для роботи з WebSocket протоколом. Він складається з WebSocket-сервера та клієнтської бібліотеки. Спочатку вони були реалізовані на JavaScript, але згодом з'явилися реалізації на багатьох інших мовах програмування.

Socket.IO реалізує додатковий функціонал, якого немає у чистому WebSocket. Наприклад, автоматичні перепідключення, якщо з'єднання втрачене, fallback до HTTP long polling, якщо WebSocket протокол не підтримується, а також впровадження неймспейсів і кімнат [15].

Namespaces необхідні для розділення відповідальності (separation of concerns) у межах одного застосунку. Socket.IO дозволяє створювати декілька неймспейсів, які будуть поводитись як окремі канали комунікації. Але водночас в середині вони будуть використовувати одне й те саме з'єднання [15].

Кімнати — це другий рівень ієрархії, у кожному з неймспейсів, у тому числі по замовчуванню, можна створювати окремі канали, які мають назву кімнати, до яких клієнти можуть долучатися та виходити з них. Таким чином, користувачі можуть транслювати повідомлення у «кімнату», і його отримають всі клієнти, котрі приєдналися до неї. Це може бути зручно для одночасної відправки повідомлення групі користувачів або збору повідомлень з декількох девайсів для одного користувача [15].

Переваги протоколу WebSocket:

— миттєвий обмін повідомленнями між клієнтом і сервером;

— зменшує накладні витрати на встановлення з’єднання;

— підходить для широкого спектру додатків.

Недоліки протоколу WebSocket:

— більш складна інфраструктура порівняно з HTTP;

— необхідні додаткові заходи для забезпечення безпеки з’єднання.

Сфери застосування протоколу WebSocket:

— чат-додатки, забезпечення миттєвого обміну повідомленнями;

— онлайн-ігри, синхронізація дій гравців у реальному часі;

— фінансові сервіси, передача біржових котирувань та інших даних у реальному часі.

Отже, вибір протоколу в IoT-проєкті залежить від конкретних потреб проєкту. MQTT є ідеальним для пристроїв з обмеженими ресурсами та потребою в реальному часі. MQTT залишається основним протоколом для IoT-проєктів завдяки своїй легкості, гнучкості та стійкості. Він широко застосовується у промислових системах, таких як SCADA, інфраструктурах «розумного дому» та аграрних технологіях.

Протоколи HTTP та HTTPS підходять для інтеграції з веб-сервісами, мобільними додатками, а також коли потрібна проста передача даних у відповідь на запит користувача. Але у випадках, коли потрібно забезпечити конфіденційність та цілісність передачі даних, протокол HTTPs є необхідним.

Протокол WebSockets підходить для задач, де необхідна постійна двостороння передача даних у реальному часі. Він є необхідним у тих випадках, коли необхідно швидко реагувати на події або підтримувати постійний потік даних, наприклад, у відеоспостереженні, графічних панелях моніторингу або чатах.

Порівняльна таблиця протоколів наведена у таблиці 3.

Таблиця 3 — порівняльна таблиця протоколів.

Характеристика	**MQTT**	**HTTP**	**HTTPS**	**WebSockets**
Модель	Публікація/ підписка	Запит/ відповідь	Запит/ відповідь	Двосторонній зв’язок
З’єднання	Постійне	Кожен запит окремо	Кожен запит окремо	Постійне
Реальний час	Так	Ні	Ні	Так
Безпека	Залежить від реалізації	Низька	Висока	Залежить від реалізації
Складність	Середня	Низька	Середня	Висока
Підтримка	Обмежена	Широка	Широка	Широка
Використання	IoT-пристрої, сенсори	Веб-сервіси, IoT-пристрої, API	Безпечні веб-сервіси	Чати, ігри, фінансові сервіси

1.5 Що таке Telegram API?

Поговоримо про такий інструмент, як Telegram API, та яку роль він виконуватиме у даному проєкті. Перш за все, Telegram API — це набір інструментів та інтерфейсів програмування, які дозволяють розробникам взаємодіяти з месенджером Telegram та створювати різноманітні додатки, боти та сервіси, що використовують його функціонал. Він працює поверх HTTP/HTTPs-запитів до сервера api.telegram.org забезпечуючи надсилання повідомлень, обробку команд, отримання зображень, кнопок, файлів тощо. Роль Telegram API у моїй бакалаврській роботі є ключовою, оскільки саме через нього буде відбуватися

взаємодія між користувачем та IoT-пристроєм. Важливо зазначити, що Telegram API не є комунікаційним протоколом у класичному розумінні IoT, але є зручним способом для створення інтерфейсу керування IoT-пристроями без необхідності створення веб-сайту або мобільного додатку.

Для використання API Telegram спочатку потрібно отримати API-ключ, який є ідентифікатором нашого додатку або бота. Цей ключ виконує важливу роль, а саме — забезпечує взаємодію між вашим додатком та Телеграм API. Після отримання API-ключа, ми зможемо почати використовувати функції API Телеграм, надсилаючи HTTP/HTTPs-запити до відповідних ендпоінтів. Наприклад, HTTP/HTTPs-запити можна використовувати для надсилання повідомлень, створення груп чи каналів, отримання інформації про користувачів та інше. Для складніших операцій, таких як оновлення, робота із зображеннями чи відео, створення кнопок, інтерактивних елементів, Telegram API дозволяє користувачам використовувати спеціалізовані бібліотеки та SDK [16].

Переваги Telegram API в IoT-системах:

— повністю безкоштовний, не потребує ліцензій чи оплат;
— працює на всіх пристроях: Android, iOS, Windows, Linux, Web;
— працює через HTTP -запити;
— самостійно зберігає історію, обробляє запити, надсилає повідомлення;
— підтримує HTTPs, обмеження за chat_id, секретний токен;
— автоматичні сповіщення.

Недоліки Telegram API:

— флуд-контроль, ліміти на кількість повідомлень в секунду;
— затримка під час запитів з пристрою;
— заборона обробки великого обсягу даних мультимедіа.

Сфери застосування Telegram API:

— розумний дім;
— аграрні системи;
— офісні автоматизовані системи;
— промислові моніторингові системи;

— системи охорони.

Порівняння Telegram API з іншими інтерфейсами керування наведену у таблиці 4.

Таблиця 4 — порівняльна таблиця Telegram API з іншими інтерфейсами керування.

Характеристика	**Telegram API**	**Веб-інтерфейс**	**Мобільний застосунок**
Простота реалізації	Легко	Складно	Складно
Потреба у сервері	Ні	Так	Так
Платформа	Кросплатформенність	Обмежено браузером	Необхідно створити
Зручність використання	Зручно	Зручно	Зручно

Отже, Telegram API є досить ефективним, простим та безпечним інструментом для реалізації інтерфейсу для керування IoT-пристроями. У контексті розумної розетки, що є темою моєї бакалаврської роботи, Telegram API дозволяє швидко та просто реалізувати керування пристроєм та зворотний зв’язок із користувачем без потреби створення додатків або веб-сервісів. Інтеграція Telegram API суттєво зменшує вартість та складність проєкту, але і забезпечує високу надійність та гнучкість у використанні.

2 РОЗРОБКА IoT-СИСТЕМИ РОЗУМНОЇ РОЗЕТКИ

2.1 Вибір апаратної платформи та компонентів IoT-пристрою

Під час планування апаратної частини пристрою було проведено аналіз сучасних мікроконтролерів, сенсорів та модулів, які можуть бути використані для бакалаврської роботи. Важливими критеріями під час пошуку були компактність, невисока вартість, енергоспоживання, підтримка Wi-Fi та Bluetooth, а також широкий функціонал. Основною платформою для розробки IoT-системи розумної розетки було обрано ESP32, мікроконтроллер, який повністю відповідає критеріям пошуку.

Мікроконтролер ESP32 дозволяє реалізувати зчитування аналогових сигналів, наприклад, для вимірювання струму, а також для керування реле через цифрові виходи. Модуль ESP32 підтримується в Arduino IDE, що дозволяє полегшити процес розробки програмного забезпечення, також є наявність великої кількості бібліотек. Високий рівень енергетичної ефективності та підтримка режимів сну, дозволить адаптувати систему до автономної роботи у майбутньому. Мікроконтроллер ESP32 зображено на рисунку 2.1.

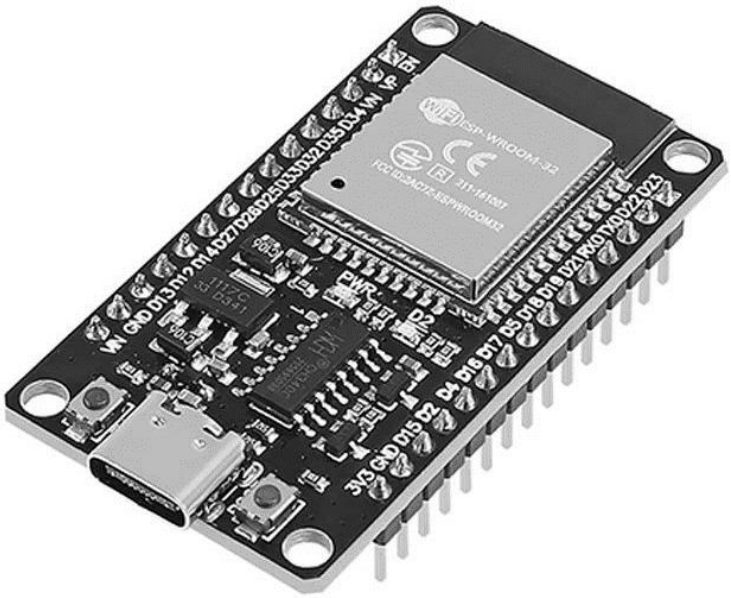

Рисунок 2.1 — Модуль розробника ESP32 WROOM-32

Основні переваги ESP32:

— вбудовані Wi-Fi та Bluetooth;

— 34 GPIO-піни;

— підтримка SPI, I2C, UART, PWM, ADC, DAC;

— флеш-пам’ять — від 4МБ до 16 МБ;

— підтримка Arduino IDE, PlatformIO.

Щоб реалізувати таку функцію, як моніторинг енергоспоживання потрібно використати датчик струму ACS712, який працює на основі ефекту Холла та дозволяє отримувати сигнал у вигляді аналогової напруги, пропорційної струму, що проходить через датчик. Сигнал подається на вхід мікропроцесора ESP32, після чого він обробляється вже програмно. Датчик струму ACS712 дозволяє вимірювати струм до 20 А з достатньою точністю для побутових задач. Простота інтеграції, наявність детальної документації та низька вартість роблять його ідеальним вибором для даного проєкту. Датчик струму ACS712 зображений на рисунку 2.2.

Рисунок 2.2 — Датчик струму ACS712

Основні переваги датчика струму ACS712:

— напруга живлення — 5В;

— простота підключення;

— аналоговий вихід (вимірюється через ADC ESP32);

— висока точність для побутових задач.

Керування навантаженням реалізується за допомогою релейного модуля на 5В, який містить вбудовану оптопару для гальванічної розв’язки та захисту мікроконтролера. Реле використовується для замикання та розмикання кола живлення, потребує 5-20мА для спрацьовування, тобто може керуватися безпосередньо з виводів мікроконтролера Arduino або подібних. Реле вмикається логічною одиницею, вимикається логічним нулем. На модулі є два світлодіоди: червоний, який сигналізує про наявність напруги живлення та зелений, який сигналізує про спрацьовування реле. Модуль реле 5В зображено на рисунку 2.3.

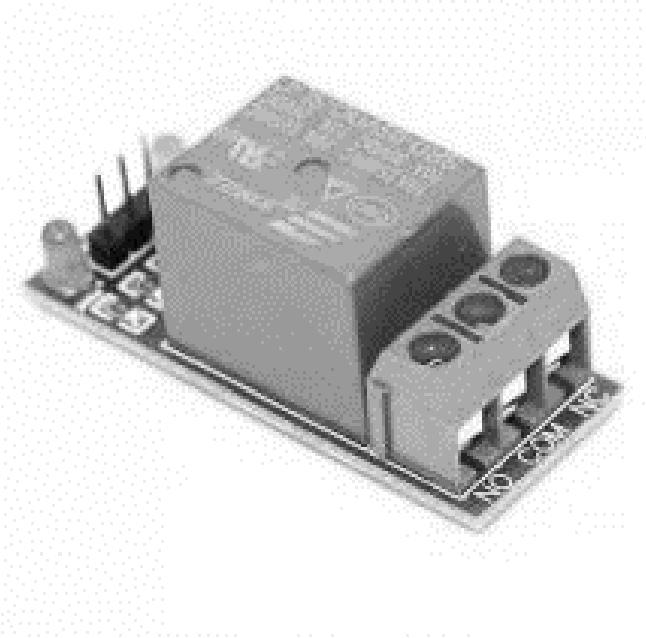

Рисунок 2.3 — Модуль реле 5В

Основні переваги модуля реле 5В:

— максимальний струм навантаженняда до 10А при 220В;

— легка інтеграція з ESP32;

— захист ESP32 від перенапруг.

Для забезпечення живлення 5В від мережі 220В використовується компактний імпульсний перетворювач HLK-PM01, зображено на рисунку 2.4. Це зручне рішення для живлення IoT-проєкта від побутової мережі. Слід виділити

функцію захисту від короткого замикання. Цей модуль дозволяє перетворювати змінний струм 220В на постійний 5В при струмі до 600мА, що є достатнім для ESP32, реле та сенсора.

Рисунок 2.4 — Імпульсний перетворювач HLK-PM01

Основні характеристики імпульсного перетворювача HLK-PM01:

— діапазон вхідної напруги від 90В до 264В;

— максимальний вхідний струм 0.2А;

— максимальний вихідний струм: 600мА;

— вихідна напруга: 5В.

Діапазон робочих температур: -20 – 60 °C; Діапазон температур зберігання: -40 – 80 °C; Відносна вологість повітря: 5-95 %; Атмосферний тиск: 80-106кПа;

Для живлення мікроконтролера та релейного модуля використовується стандартний імпульсний адаптер, який може бути замінений на будь-який сумісний блок живлення, включаючи перероблений зарядний пристрій від мобільного телефону. Важливою умовою є наявність стабілізації напруги, оскільки коливання можуть призвести до некоректної роботи ESP32.

Вибір кожного з елементів було зроблено, опираючись на такі параметри як доступність, функціональність та оптимальна ціна. Усі компоненти можуть бути безпроблемно придбані на сучасному ринку електроніки, що дозволить виготовити дану систему іншим користувачам. Таким чином, сформовано повний набір апаратних засобів, необхідних для реалізації IoT-пристрою.

2.2 Розробка схеми з’єднань пристрою

Наступним кроком є побудова схеми з'єднання пристрою та опис логіки їхньої взаємодії в межах загальної архітектури.

Як фізична схема IoT-пристрою розумної розетки, так і його логічна архітектура, є визначальним етапом під час побудови надійного, безпечного та повністю автономного електронного пристрою. Запропоноване рішення передбачає роботу з напругою 220В змінного струму тому має інтегруватися в побутову електромережу. IoT-пристрій живиться безпосередньо від мережі 220 В, через імпульсний перетворювач HLK-PM01, що уникнути зовнішніх блоків живлення, що є надзвичайно важливим аспектом для компактності та автономності пристрою.

2.2.1 Загальна архітектура схеми

Загальна архітектура IoT-пристрою розумної розетки базується на принципі модульності, де кожен компонент пристрою виконує визначену функцію та взаємодіє з іншими компонентами. Побудова логічної архітектури необхідна для забезпечення взаємодії компонентів, ізоляції силових та сигнальних ланцюгів, забезпечення безпеки та надійності функціонування пристрою в автономному режимі.

Архітектура IoT-пристрою складається з таких функціональних блоків.

Блок живлення — забезпечує перетворення змінного струму 220В у стабілізовану напругу 5В.

Блок керування — мікроконтролер ESP32, який обробляє дані та опрацьовує запити з Telegram API.

Блок виконавчих елементів — модуль реле, який здійснює замикання або розмикання кола змінного струму (фази) до розетки.

Блок моніторингу вхідного струму — сенсор ACS712, який вимірює рівень споживання струму і передає аналоговий сигнал на вхід ESP32.

Розетка 220В — кінцевий виконавчий елемент, до якого підключається побутовий прилад.

Розглянемо взаємозв’язок блоків у системі IoT-пристрою.

HLK-PM01 безпосередньо підключається до входу змінного струму 220В і генерує постійний струм 5 В, використовується для живлення мікропроцессора ESP32, модуля реле та ACS712.

ESP32 є центральним процесором IoT-пристрою, він ініціалізує Wi-Fi-з'єднання, опрацьовує запити з Telegram API та приймає команди від користувача.

GPIO-пін ESP32 керує реле: при надходженні команди /on — подається високий рівень сигналу, і реле замикає фазу на розетку.

Датчик струму ACS712 зчитує значення струму і подає сигнал на аналоговий вхід ESP32, де відбувається перетворення в цифрове значення за допомогою АЦП;

ESP32 самостійно надсилає повідомлення до Telegram-бота та відключає навантаження у випадку перевищення критичного порогу струму.

Такий підхід до побудови архітектури системи дозволяє досягти найкращого результату та роботи пристрою. Насамперед, пристрій є повністю автономним: він не потребує зовнішніх адаптерів або джерел постійного струму, оскільки живиться безпосередньо від мережі змінного струму 220В за допомогою вбудованого імпульсного перетворювача HLK-PM01. Це рішення дозволяє зменшити кількість зовнішніх елементів і забезпечує компактність, що є надвичайно важливим при розміщенні електроніки всередині стандартного корпусу розетки. Крім того, схема відповідає вимогам безпеки, оскільки HLK-PM01 має вбудовану ізоляцію та захист від перевантажень і коротких замикань. Це означає, що у разі стрибка напруги в мережі, мікроконтролер та інші логічні елементи пристрою залишаться ізольованими від потенційно небезпечної напруги.

Мікроконтролер ESP32 дозволяє реалізувати весь необхідний функціонал в одному пристрої, оскільки він має достатню обчислювальну потужність та обсяг пам’яті для паралельної роботи з модулем реле, зчитування даних з аналогового сенсора струму ACS712 та підтримки захищеного з’єднання з Telegram API. Тобто, пристрій може виконувати як локальне керування, так і обробку команд користувача з хмарної платформи в реальному часі.

Також важливою перевагою обраної архітектури є її модульність. Це означає, що усі компоненти системи об’єднані таким чином, щоб їх можна було легко замінити в разі несправності, не порушуючи загальну логіку роботи системи.

2.2.2 Ізоляція, безпека, електротехнічні вимоги

Дотримання правил електричної та пожежної безпеки є невід’ємною складовою для розробки будь-якого пристрою. В нашому випадку це розробка пристрою, що взаємодіє з мережею змінного струму 220В. На відміну від пристроїв, які живляться від USB або батарей, IoT-система розумної розетки взаємодіє із потенційно небезпечною напругою, здатною спричинити ураження струмом, коротке замикання або перегрів будь-якого з елементів. Тому при проектуванні архітектури, виборі компонентів та під час побудови схеми з’єднань досить важливо враховувати вимоги електротехнічних стандартів і рекомендацій та дотримуватися правил безпеки. У підпункті 2.2.2 розглянуто всі рекомендації, щодо забезпечення електробезпеки, захисту мікроконтролера та користувача, ізоляції сигнальних та силових ліній, а також вимоги до монтажу пристрою.

Першим і найважливішим аспектом захисту системи є забезпечення гальванічної розв’язки, фізичного та електричного відділення високовольтної частини схеми (220В) від низьковольтної логіки (5В). Гальванічна розв’язка досягається завдяки двом ключовим елементам: використанню оптопари в релейному модулі та інтегрованій ізоляції в імпульсному джерелі живлення HLK-PM01.

Оптопара дозволяє ESP32 керувати реле, без прямого фізичного з’єднання з контактами, які комутують фазу 220В. Це означає, що навіть при збої або

короткому замиканні на виході реле, електричний імпульс не досягне мікроконтролера та інших елементів пристрою.

Імпульсний перетворювач HLK-PM01 повністю відокремлює мережеве живлення від стабілізованої напруги 5В, яка живить ESP32, сенсор струму ACS712 та модуль реле. Ізоляційна напруга HLK-PM01 становить понад 3000В, що гарантує безпечну експлуатацію навіть у разі короткочасних стрибків напруги.

Підхід, де всі низьковольтні елементи мають спільну шину "землі", яка не має жодного електричного зв’язку з нульовим проводом мережі змінного струму, повністю відповідає нормам електробезпеки. Це досить важливо для зменшення електромагнітного впливу та усунення ризику ураження струмом від дотику до логічних елементів системи.

Для підвищення надійності безпеки пристрою рекомендовано використовувати додаткові захисні компоненти на вході імпульсного перетворювача HLK-PM01. До захисних компонентів належить варистор, який спрацьовує при стрибках напруги і захищає схему від перенапруг, а також запобіжник, який миттєво розірве коло у випадку перевищення допустимого струму. Щоб зменшити пусковий струм при ввімкненні доцільно застосувати термістор (NTC), а для боротьби з високочастотними шумами — конденсатори Y-класу. Всі описані елементи дозволяють сформувати базовий ланцюг захисту.

У контексті теплової безпеки важливо враховувати, що хоча загальне енергоспоживання пристрою є невеликим, модуль реле та HLK-PM01 при тривалій роботі можуть нагріватися, особливо в умовах недостатньої вентиляції. Саме тому корпус пристрою повинен бути виготовлений із матеріалів, які не підтримують горіння, наприклад, самозагасаючий полікарбонат або поліамід. Під час монтажу всі елементи повинні бути надійно закріпленими, а високовольтні з’єднання — ізольовані термозбіжною трубкою або діелектричним лаком.

Таким чином, IoT-пристрій розумної розетки повністю враховує ключові вимоги до безпеки під час роботи з електроенергією. Завдяки правильно спроєктованій системі ізоляції, реалізованій схемі живлення, гальванічному

розділенню, а також технічним заходам захисту, розробка пристрою має високий рівень надійності.

2.3 Програмне забезпечення IoT-пристрою

Функціональність сучасного електронного пристрою неможлива без ефективного програмного забезпечення. Програмна частина надає мікроконтролеру розуміння того, які функції він має виконувати для забезпечення потрібного функціоналу пристрою. Це дозволяє запрограмувати мікроконтролер для обробки сигналів з датчиків, прийому команд користувача через інтернет, реакції на перевантаження, виконання алгоритмів автоматизації, підтримку стабільного з’єднання з мережею та інше. У IoT-пристрої розумної розетки, яка розробляється в рамках цієї бакалаврської роботи, програмне забезпечення реалізоване у вигляді прошивки для мікроконтролера ESP32, написаної на мові C++ з використанням середовища Arduino IDE.

Прошивка мікроконтролера ESP32 складається з декількох функціональних блоків, кожен з яких виконує свою задачу. Саме модульність дозволяє легко масштабувати функціонал у разі потреби.

2.3.1 Архітектура програмної логіки

Програмне забезпечення реалізовано у вигляді структури, де кожен функціональний модуль відповідає за окрему частину логіки пристрою.

Модуль ініціалізації: конфігурує GPIO-піни, запускає Wi-Fi-з'єднання, налаштовує інтерфейси UART, ADC.

Модуль з'єднання з Telegram: перевіряє наявність нових повідомлень, обробляє текстові команди користувача.

Модуль обробки навантаження: вмикає або вимикає реле залежно від отриманих команд або виміряного струму.

Модуль зчитування струму: взаємодіє з аналоговим входом ESP32, обробляє дані з ACS712.

Модуль безпеки: контролює перевищення заданого порогу струму і надсилає попередження в Telegram.

Модуль конфігурації Wi-Fi: за відсутності підключення активує точку доступу і веб-інтерфейс налаштувань (через WiFiManager).

Всі зазначені модулі взаємодіють між собою в межах головного циклу програми, реалізованого у функції loop(). Цей цикл виконується безперервно і забезпечує постійний моніторинг вхідних даних, стан сенсорів та команд від користувача. Така структура дозволяє пристрою працювати в стані реального часу, реагуючи на події доволі швидко, що є необхідною вимогою для сучасних систем IoT-проєктів.

2.3.2 Гнучке налаштування мережі Wi-Fi

Наявність стабільного підключення до інтернету є досить важливим фактором у роботі сучасного IoT-пристрою, особливо якщо має функціонал керування через хмарні сервіси або месенджери. У розробці розумної розетки Wi-Fi-з'єднання дозволяє реалізувати взаємодію з Telegram API та забезпечити гнучке налаштування пристрою в домашніх умовах без необхідності фізичного підключення до комп’ютера. Однією з важливих функціональних особливостей цього проєкту є можливість динамічного налаштування параметрів підключення до мережі Wi-Fi без необхідності перепрошування пристрою. Це означає, що навіть після встановлення пристрою в корпус розетки, користувач може легко змінити назву точки доступу (SSID) або пароль, у випадку зміни домашньої мережі. Для реалізації описаного функціоналу використовується спеціалізована бібліотека WiFiManager, яка дозволяє організувати простий і зручний процес налаштування.

Принцип роботи бібліотеки WiFiManager полягає в тому, що під час запуску мікроконтролер ESP32 перевіряє, чи є збережені параметри мережі Wi-Fi. Якщо параметри наявні та з'єднання з мережею може бути встановлене — пристрій автоматично підключається до інтернету. У випадку, коли пристрій не може підключитися, наприклад, пароль було змінено, автоматично створюється точка доступу (Access Point) з власною назвою, наприклад «Rozetka-Setup». Користувач

підключається до цієї точки доступу зі смартфона або комп’ютера, після чого відкриває в браузері автоматично згенеровану сторінку конфігурації. На цій сторінці він бачить перелік доступних Wi-Fi-мереж, вибирає потрібну та вводить пароль. Після цього ESP32 зберігає нові дані до внутрішньої пам’яті та перезавантажується вже з новими параметрами.

Такий підхід забезпечує високу гнучкість і дозволяє уникнути найпоширенішої проблеми IoT-пристроїв — запрограмованої в код назви мережі без можливості заміни без перепрошивання пристрою. З технічної точки зору, бібліотека WiFiManager працює з EEPROM або внутрішнім flash-сховищем ESP32, де зберігає параметри конфігурації, при цьому жодні додаткові дії з боку користувача, окрім підключення до тимчасової точки доступу, не потрібні.

Завдяки інтеграції модуля гнучкого налаштування Wi-Fi, IoT-пристрій розумна розетка має можливість швидкого й безпечного підключення до будь-якої мережі без додаткових інструментів. Саме такий підхід підвищує зручність використання пристрою.

2.3.3 Взаємодія з Telegram API

Однією з основних переваг Інтернету речей є можливість керування пристроями дистанційно, без фізичного доступу. У контексті розумної розетки, зручною та ефективною формою такого керування є використання месенджера Telegram, що дозволяє створювати інтерактивних ботів завдяки відкритому API. У цьому проєкті Telegram API виконує роль основного інтерфейсу між користувачем і мікроконтролером ESP32, забезпечуючи передачу команд, зворотний зв’язок, а також моніторинг у режимі реального часу.

Застосунок Telegram був обраний засібом комунікації з кількох причин.

Багатоплатформеність — працює на Android, iOS, Windows, macOS, Web.

Безпечне з’єднання — усі API-запити працюють за захищеним протоколом HTTPS (TLS/SSL).

Надійність — мільйони користувачів щодня користуються Telegram без проблем із доступністю.

Інтуїтивна взаємодія — керування реалізується простими командами, які будуть зрозумілі користувачам без попереднього досвіду.

Усі команди користувача надходять до Telegram-бота. В свою чергу бот отримує повідомлення, які надсилаються до серверів Telegram. Мікропроцесор ESP32, періодично опитує Telegram API на наявність нових повідомлень через захищене з'єднання HTTPS, використовуючи бібліотеку UniversalTelegramBot. У випадку отримання команди, пристрій аналізує її та виконує відповідну дію — вмикає/вимикає реле, надсилає повідомлення, вимірює струм тощо.

Між ESP32 та Telegram API використовуються стандартні запити REST-типу, наприклад: GET для отримання нових повідомлень та POST для надсилання відповіді. Це дозволяє працювати без проміжного сервера чи обробника, що значно зменшує складність архітектури та забезпечує повну автономність пристрою.

Telegram-бот має набір команд для керування IoT-пристроєм, команди наведені у таблиці 5.

Таблиця 5 — команди телеграм-бота

Команда	**Призначення**
/start	Привітальне повідомлення з інструкцією
/on	Увімкнення розетки
/off	Вимкнення розетки
/status	Вивід поточного стану реле
/power	Поточне споживання струму в амперах
/help	Інструкція з усіма доступними командами

Команди обробляються у функції, яка викликається з головного циклу loop(). Функція використовує метод getUpdates() бібліотеки UniversalTelegramBot, що повертає список повідомлень, які пристрій ще не обробив. Після цього кожне повідомлення аналізується на відповідність відомим командам, і у разі збігу

виконується відповідна дія. Результати команд повертаються користувачу у вигляді текстових повідомлень, емодзі або індикаторів стану.

2.3.4 Виявлення та обробка струму

Моніторинг електричних параметрів є важливою функцією для сучасних IoT-пристроїв, що працюють з мережевим живленням. Під час розробки розумної розетки було прийняте рішення використати цю функцію для активного захисту при перевищенні допустимого струму, що дозволяє автоматично вимикати пристрій під час навантаження та сповіщати користувача. Для реалізації цієї функції у проєкті використано датчик струму ACS712, який працює на основі ефекту Холла, що дозволяє точно вимірювати струм, який протікає через фазовий провід, без потреби в прямому контакті із провідником. Це дає змогу забезпечити електричну ізоляцію між високовольтною частиною і низьковольтною логікою ESP32.

Мікроконтролер ESP32 оснащений високоточним аналогово-цифровим перетворювачем (ADC), що дозволяє зчитувати напругу з виходу сенсора з роздільністю до 12 біт (4096 рівнів). Вимірювання напруги з виходу сенсора обраховується за формулою:

$$I = \frac{U_{\text{вихід}} - U_0}{S},$$

де I — сила струму;

$U_{\text{вихід}}$ — напруга на виході сенсора;

$U_0 \approx 2.5\text{В}$ — нульовий рівень;

S = 0.1В/А — чутливість ACS712-20A.

Зчитування значення з ADC наведено у лістингу 2.1.

Лістинг 2.1 — Код зчитування аналогового значення.

```
int rawValue = analogRead(currentSensorPin);
float voltage = rawValue * (3.3 / 4096.0);
float current = (voltage - 2.5) / 0.1;
```

Захист від перевантаження є важливою функцією для забезпечення безпеки. Якщо обчислене значення перевищує допумтиме значення, пристрій автоматично вимикає реле, змінює логічний стан системи, надсилає повідомлення у Telegram, яке має вигляд «Перевищено допустимий струм: 12.4A. Розетка відключена автоматично!».

Розглянемо переваги вбудованої логіки захисту.

— простота реалізації: не потребує зовнішніх мікросхем.

— надійність: реагує за мілісекунди, встигає вимкнути реле до перегріву.

— автономність: не потребує доступу до мережі Інтернет для вимкнення.

— інформативність: користувач отримує сповіщення про причиу небезпеки у Telegram-боті.

— безпека: розмикання фази, що гарантує повне припинення живлення.

Цей підхід є не лише способом інформування користувача, але й превентивним захистом, що дозволяє уникнути перегріву, пожежі або виходу з ладу побутових приладів, підключених до розетки.

2.3.5 Telegram-бот: архітектура, вибір мови, приклад інтерфейсу

Telegram-бот у даному проєкті є ключовим елементом взаємодії між користувачем і мікроконтролерною системою. Саме він забезпечує інтуїтивно зрозуміле та швидке управління IoT-пристроєм. Відмова від окремого мобільного додатку на користь Telegram-бота дозволяє спростити розробку, зменшити обсяг клієнтського коду, а також максимально розширити платформну доступність — бот доступний на Android, iOS, Windows, Linux, macOS, Web. Такий підхід не потребує використання окремого хостингу, розробки backend-сервера або бази даних.

Telegram-бот обробляє текстові команди, надсилає відповіді, реагує на події (наприклад, перевищення струму), а також має підтримку двох мов, а саме українську та англійську. Архітектура Telegram-бот в проєкті IoT-пристрою розумної розетки складається з наступних компонентів.

Парсер повідомлень — перевіряє, чи нові повідомлення містять відомі команди.

Обробник команд — реагує на команди /on, /off, /status, /help.

Модуль повідомлень — формує відповіді.

Автоматичні сповіщення — бот надсилає попередження у випадку небезпеки (наприклад, струм > 10А).

Вибір мови спілкування — зміна мови інтерфейсу здійснюється за допомогою спеціальної команди /lang, після чого всі відповіді Telegram-бота відображаються вибраною користувачем мовою.

У боті реалізовано двомовний інтерфейс (українська / англійська), що дозволяє зручно користуватись пристроєм як україномовним, так і англомовним користувачам. Мову можна встановити за допомогою команди /lang. Приклад української версії боту зображено на рисунку 2.5, англійської на рисунку 2.6 відповідно.

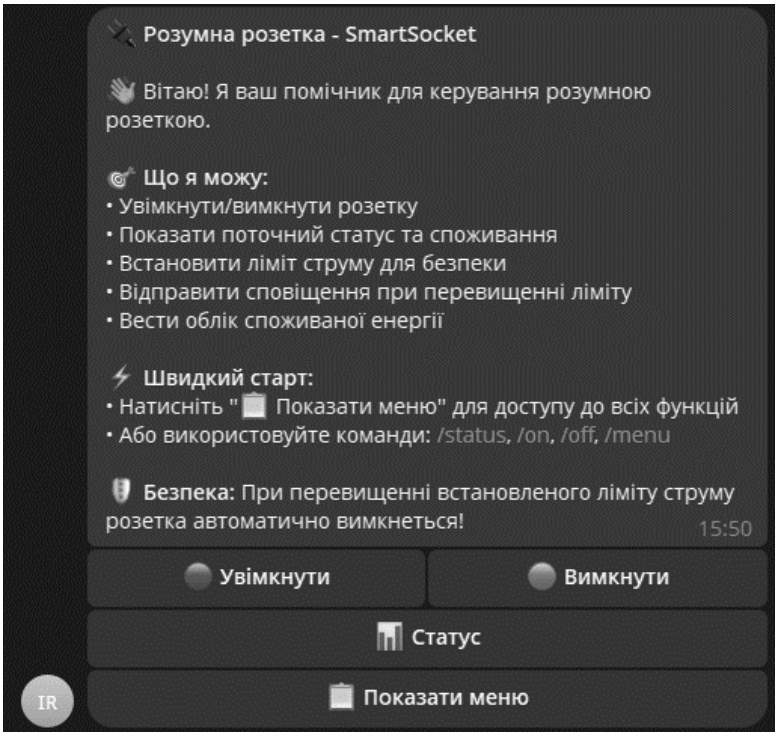

Рисунок 2.5 — Приклад інтерфейсу бота українською мовою

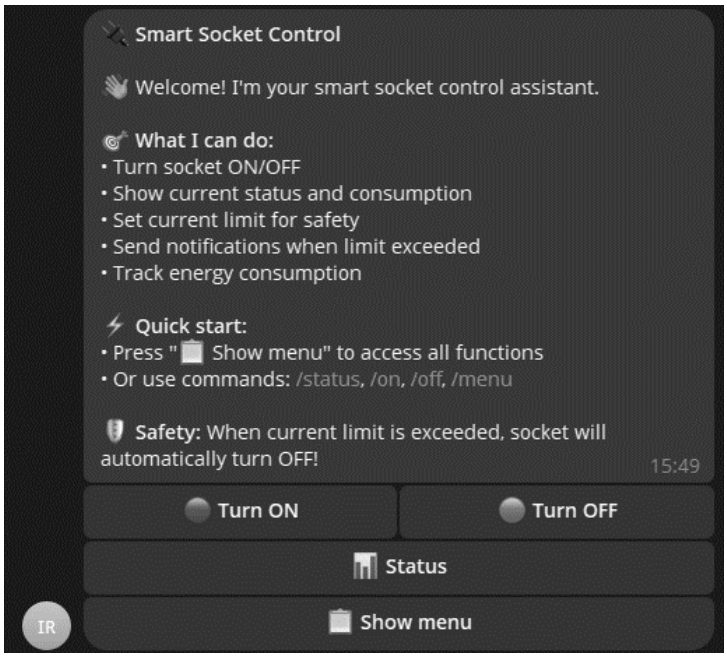

Рисунок 2.6 — Приклад інтерфейсу бота англійською мовою

Інтерфейс користувача — один із основних компонентів взаємодії між користувачем та IoT-пристроєм. Проєкт розумної розетки не передбачає використання власного фізичного дисплея або кнопок, а основним каналом взаємодії виступає програмний текстовий інтерфейс, реалізований через Telegram-бота. Саме завдяки цьому інтерфейсу користувач може здійснювати повноцінне керування пристроєм: вмикати чи вимикати розетку, отримувати дані про споживання струму, перевіряти стан реле, а також отримувати сповіщення про аварійні ситуації.

Telegram-бот реалізує універсальний, легкий та інтуїтивно зрозумілий текстовий інтерфейс. Управління IoT-пристрою здійснюється за допомогою відповідних команд. Розроблений інтерфейс не перевантажений зайвими повідомленнями чи візуальними елементами — відповіді сформульовані чітко, коротко та зрозуміло. Наприклад, команда /on відобразить користувачу відповідь від IoT-пристрою, яка має вигляд «Розетка увімкнена». Це повідомлення чітко сигналізує користувачу про виконану дію.

Ще однією особливістю розробленого інтерфейсу є доступ до довідкової інформації за допомогою команди /help. Ця команда відображає список усіх

доступних команд і пояснює їх призначення, що дозволяє новим користувачам швидко та без зайвих проблем зрозуміти як керувати пристроєм через Telegram-боту.

2.3.6 Зберігання конфігурації та налаштувань пристрою в пам’яті ESP32

У більшості систем IoT-пристроїв збереження параметрів налаштувань є вкрай важливим. Змоделюємо ситуацію коли розумна розетка раптово знеструмлюється або перезавантажується. Без реалізованої логіки збереження налаштувань, після ввімкнення у пам’яті пристрою не буде збережено останній стан реле, мову інтерфейсу, параметри безпеки (наприклад, граничне значення струму) та інші налаштування. Це призведе до порушення роботи та небезпечних ситуацій. Саме тому у даній роботі було реалізовано механізм постійного зберігання ключових налаштувань у вбудованій пам’яті ESP32.

Мікроконтролер ESP32 має два основні способи зберігання даних:

EEPROM — програмний модуль, який емуляторно реалізує EEPROM у flash-пам’яті. Він дозволяє читати й записувати байти за певною адресою.

NVS (Non-Volatile Storage) — просунута система збереження, яка дозволяє зберігати пари “ключ-значення” (string, int, bool тощо), більш стійка до зношення пам’яті.

У цьому проєкті використовується бібліотека EEPROM, оскільки вона має простий інтерфейс та не потребує складної ініціалізації. Для збереження невеликого набору параметрів (до 512 або 4096 байт) використання даної бібліотеки є достатнім.

У розумній розетці необхідно зберігати такі параметри, як стан реле, мова інтерфейсу, граничне значення струму.

Щоб реалізувати логіку збереження параметрів в пам’яті мікропроцессора ESP32 необхідно спочатку виділити місце в пам’яті, а потім записувати або перезаписувати змінні параметрів.

Приклад коду виділення пам’яті наведено у лістингу 2.2.

Лістинг 2.2 — Код для виділення пам’яті у мікропроцессорі ESP32.

```
#include <EEPROM.h>
void setup() {
      EEPROM.begin(64);  // виділення 64 байти пам’яті
      relayState = EEPROM.read(0);     // стан реле
      currentLanguage = EEPROM.read(1); // мова
}
EEPROM.write(0, relayState);
EEPROM.write(1, currentLanguage);
EEPROM.commit(); // збереження в пам’ять
```

Отже, у другому розділі було детально описано кожен з етапів розробки IoT-пристрою розумної розетки, що керується за допомогою Telegram-бота. Розробка охоплює повний цикл — від вибору апаратної платформи та електричних компонентів до реалізації програмного забезпечення з використанням Wi-Fi, датчика струму, захисту та хмарного інтерфейсу.

В основі пристрою лежить мікроконтролер ESP32, який забезпечує достатню обчислювальну потужність для IoT-пристрою. Для вимірювання значення струму було використано датчик ACS712. Реле, кероване через ESP32, виконує функцію комутації фази живлення 220 В. Бібліотека WiFiManager спрощує процес налаштування параметрів без необхідності перерошивання мікропроцессора. Інтерфейс Telegram-бота дозволяє керувати пристроєм за допомогою звичних команд та отримувати сповіщення у разі аварійних ситуацій. Важливим етапом розробки було створення функціоналу збереження конфігураційних параметрів у пам’яті ESP32.

Таким чином, реалізована IoT-система розумної розетки є прикладом компактного, безпечного, енергоефективного і керованого в реальному часі пристрою, який об’єднує сучасні технології мікроконтролерів, телеметрії та хмарних сервісів.

3 ПРОЄКТУВАННЯ IoT-ПРИСТРОЮ РОЗУМНОЇ РОЗЕТКИ

3.1 Проєктування структури системи керування розумною розеткою

Під час аналізу вимог до системи керування розумною розеткою було визначено ключові функціональні параметри, що забезпечують ефективну та безпечну експлуатацію пристрою в реальних умовах. В рамках даної бакалаврської роботи прийнято рішення спроектувати IoT-пристрій, що дозволяє віддалено керувати ним через інтегрований Telegram-бот, а також здійснювати моніторинг електричних параметрів, таких як струм, напруга, потужность та споживання електроенергії. Розроблена система орієнтована на застосування у побутових та виробничих умовах, де висока швидкодія обробки даних і оперативне реагування на зміни навантаження відіграють надвичайно важливу роль.

Основна концепція системи базується на інтеграції апаратних компонентів мікроконтролера ESP32, датчика струму ACS712 та електромеханічного реле, що забезпечують базову функціональність пристрою. Для забезпечення стабільного живлення компонентів пристрою використовується імпульсний перетворювач HLK-PM01, який перетворює змінну напругу на стабільну постійну, що є важливим для коректної роботи мікроконтролера ESP32 та електронних компонентів системи. Використання імпульсного перетворювача сприяє підвищенню енергетичної ефективності, дозволяє зменшити теплові втрати та забезпечує автономну роботу пристрою навіть при змінних параметрах мережі.

Обмін даними здійснюється через HTTP API, що дозволяє передавати команди у форматі JSON та отримувати актуальні дані роботи пристрою. За допомогою мережевого зв’язку ESP32 пристрій може підтримувати безперервну комунікацію з віддаленим інтерфейсом, яким виступає Telegram-бот, що надає користувачеві зручний та легкий спосіб управління. Таким чином, впровадження стандартних мережевих протоколів забезпечує мінімальні затримки у передачі інформації і дає можливість у реальному часі реагувати на зміну стану пристрою або умов експлуатації.

Система здатна автоматично виявляти перевищення критичного ліміту струму, розмикати реле з метою запобігання аварійним ситуаціям як у самому пристрої, так і в електричній мережі та надсилати повідомлення користувачеві про перевищення струму. Впровадження механізмів збереження ключових параметрів роботи через внутрішню пам’ять мікропроцесору EEPROM дозволяє пристрою відновлювати свій стан після перезавантаження.

Спроєктована система керування розумною розеткою демонструє ефективну інтеграцію апаратних та програмних компонентів із врахуванням вимог до безпеки, енергетичної ефективності та стабільності роботи. Розроблена архітектура спрямована на мінімізацію затримок у передачі даних і забезпечує високий рівень адаптивності до умов змінного навантаження.

3.2 Програмна реалізація мікроконтролера ESP32

Програмна реалізація мікроконтролера ESP32 є центральною частиною системи керування розумною розеткою. Код написано на мові C++ з використанням середовища розробки Arduino IDE та бібліотек ESP32. Для впровадження системи керування розумною розеткою розроблено модульну програмну архітектуру, яка забезпечує ініціалізацію апаратних засобів, організацію мережевого з’єднання та обробку запитів користувача.

На початковому етапі розробки було створено окремий модуль, який відповідає за базову ініціалізацію мікроконтролера ESP32, налаштування апаратних ресурсів та запуск мережевих сервісів. Процес починається із встановлення послідовного зв’язку для цілей відлагодження, налаштування пінів для управління електромеханічним реле (що безпосередньо впливає на перемикання розетки), індикації за допомогою LED та зчитування аналогового сигналу з датчика струму ACS712. Важливо, що у цьому модулі також реалізовано ініціалізацію механізму збереження конфігураційних параметрів через Preferences, що дозволяє відновлювати стан пристрою після перезавантаження.

Лістинг 3.1 — Код функції setup

```
void setup() {
  Serial.begin(115200);
  Serial.println("\n=== ESP32 Smart Socket Starting ===");
  pinMode(RELAY_PIN, OUTPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(ACS712_PIN, INPUT);
  digitalWrite(RELAY_PIN, LOW);
  digitalWrite(LED_PIN, LOW)
  preferences.begin("smart_socket", false);
  loadSettings();
  device_state.device_start_time = millis();
  device_state.last_energy_update = millis();
  Serial.println("Hardware initialized");
  setupWiFi();
  setupWebServer();
  Serial.println("=== Setup Complete ===");
  printStatus();
}
```

Функція setup() є точкою входу в прошивку. За допомогою функції Serial.begin(115200) встановлюється послідовний зв'язок для налагодження. Функція pinMode() дозволяє налаштувати піни управління реле та LED-індикатора. Ініціалізація підсистеми збереження налаштувань реалізована за допомогою бібліотеки Preferences, що дозволяє завантажувати параметри у пам’ять та відновлювати стан пристрою після перезавантаження. Потім виконується ініціалізація підсистеми збереження налаштувань через EEPROM та завантаження збережених параметрів. Завершальним етапом є налаштування мережевого з’єднання за допомогою функцій setupWiFi() та setupWebServer() із подальшим виведенням статусу за допомогою функції printStatus().

Наступним кроком була реалізація функції, що відповідає за встановлення стабільного з’єднання з мережею Wi-Fi. Вона має назву setupWiFi().

Лістинг 3.2 — Код функції setupWiFi

```
void setupWiFi() {
  Serial.println("Setting up WiFi...");
   WiFi.mode(WIFI_STA);
  for (int i = 0; i < num_networks; i++) {
    Serial.printf("Trying to connect to: %s\n", wifi_networks[i][0]);
    WiFi.begin(wifi_networks[i][0], wifi_networks[i][1]);
    int attempts = 0;
```

```
    while (WiFi.status() != WL_CONNECTED && attempts < 20) {
      delay(500);
      Serial.print(".");
      attempts++;
    }
    if (WiFi.status() == WL_CONNECTED) {
      Serial.printf("\nConnected to %s\n", wifi_networks[i][0]);
      Serial.printf("IP Address: %s\n", WiFi.localIP().toString().c_str());
      Serial.printf("Signal Strength: %d dBm\n", WiFi.RSSI());
      device_state.wifi_connected = true;
      device_state.wifi_rssi = WiFi.RSSI();
      return;
    }
    Serial.println(" Failed");
  Serial.println("Starting WiFi configuration portal...");
  wifiManager.setAPCallback(configModeCallback);
  wifiManager.setSaveConfigCallback(saveConfigCallback);
  if (!wifiManager.autoConnect("SmartSocket_Config", "12345678")) {
    Serial.println("Failed to connect and hit timeout");
    ESP.restart();
  }
  Serial.printf("WiFi connected via manager\n");
  Serial.printf("IP Address: %s\n", WiFi.localIP().toString().c_str());
  device_state.wifi_connected = true;
  device_state.wifi_rssi = WiFi.RSSI();
}
```

За допомогою циклу у функції setupWiFi() здійснюється спроба підключення до кожної з мереж, заданих у масиві wifi_networks. Для кожної мережі виконуються повторні спроби підключення протягом обмеженого часу (20 циклів з паузою у 500 мс). У випадку успішного з’єднання друкуються інформаційні повідомлення (IP-адреса, сила сигналу), а стан з’єднання фіксується в змінних пристрою. Якщо автоматичне підключення не здійснюється, запускається конфігураційний портал через WiFiManager, що дозволяє вручну задати параметри з’єднання.

Після встановлення зв’язку через мережу WiFi необхідно налаштувати функцію обробки HTTP-запитів, що надходять від Telegram-бота.

Лістинг 3.3 — Код функції handleRelay

```
void handleRelay() {
  setCORSHeaders();
  if (!server.hasArg("plain")) {
    server.send(400, "application/json", "{\"success\":false,\"error\":\"No
data\"}");
    return;
  }
  DynamicJsonDocument doc(128);
```

```
  deserializeJson(doc, server.arg("plain"));
  bool new_state = doc["state"];
  if (new_state && device_state.limit_exceeded) {
    server.send(400, "application/json", "{\"success\":false,\"error\":\"Current
limit exceeded\"}");
    return;
  }
  bool previous_state = device_state.relay_state;
  device_state.relay_state = new_state;
  digitalWrite(RELAY_PIN, new_state ? HIGH : LOW);
  if (new_state && !previous_state) {
    device_state.relay_start_time = millis();
    device_state.relay_uptime_seconds = 0;
  } else if (!new_state) {
    device_state.relay_uptime_seconds = 0;
  DynamicJsonDocument response(128);
  response["success"] = true;
  response["relay_state"] = device_state.relay_state;
  String responseStr;
  serializeJson(response, responseStr);
  server.send(200, "application/json", responseStr);
}
```

Функція handleRelay() встановлює необхідні заголовки для підтримки крос-доменного доступу (CORS). Далі перевіряється наявність параметра «plain» у вхідному запиті. З JSON-даних вилучається параметр state, що містить в собі стан розумної розетки. Наприклад, якщо користувач надсилає боту команду “увімкнути” функція спочатку перевіряє наявність необхідних параметрів у JSON-запиті та вилучає значення state. Якщо запит передбачає увімкнення, але пристрій знаходиться в режимі перевищення струму, команда відхиляється з повідомленням про помилку. У випадку, коли перевірка успішна функція змінює стан реле через digitalWrite(), фіксує час переходу та повертає JSON-відповідь з результатом виконання.

Для забезпечення точного моніторингу електричних параметрів розумної розетки було розроблено функцію updateMeasurements(), яка відповідає за періодичне оновлення показників струму, напруги та потужності. Ця функція є ключовою для системи безпеки та контролю споживання електроенергії.

Лістинг 3.4 — Код функції updateMeasurements

```
void updateMeasurements() {
  device_state.wifi_connected = WiFi.status() == WL_CONNECTED;
```

```
  if (device_state.wifi_connected) {
    device_state.wifi_rssi = WiFi.RSSI();
  }
  if (simulate_overcurrent) {
    device_state.current = simulated_current;
  } else {
    int sensorValue = analogRead(ACS712_PIN);
    float voltage = (sensorValue / 4095.0) * VOLTAGE_REFERENCE;
    float actual_voltage = voltage * (5.0 / 3.3);
    device_state.current = abs((actual_voltage - 2.5) / ACS712_SENSITIVITY);
    if (device_state.current < 0.1) {
      device_state.current = 0.0;
    }
    if (!device_state.relay_state) {
      device_state.current = random(0, 3) / 100.0;
    }
  }
  device_state.power = device_state.voltage * device_state.current;
}
```

Функція updateMeasurements() спочатку перевіряє стан WiFi-з'єднання та оновлює відповідні параметри пристрою. Основна частина функції зосереджена на зчитуванні показників з датчика струму ACS712. Аналоговий сигнал з піна ACS712_PIN перетворюється у цифрове значення через АЦП з 12-бітною роздільною здатністю (4095 рівнів). Отримане значення нормалізується відповідно до опорної напруги та калібрується згідно з чутливістю датчика ACS712_SENSITIVITY. Для усунення шумів та незначних коливань встановлено поріг у 0.1А, нижче якого струм вважається нульовим. Завершальним етапом є розрахунок потужності як добутку напруги та струму.

Наступною важливою є функція updateEnergyCounter(), що забезпечує точний облік спожитої електроенергії та її періодичне збереження у постійній пам'яті мікроконтролера.

Лістинг 3.5 — Код функції updateEnergyCounter

```
void updateEnergyCounter() {
  unsigned long current_time = millis();
  unsigned long time_diff = current_time - device_state.last_energy_update;
  if (time_diff >= 1000 && device_state.relay_state) {
    float hours = time_diff / 3600000.0;
    device_state.energy += (device_state.power / 1000.0) * hours;
    device_state.last_energy_update = current_time;
    static unsigned long last_save = 0;
    if (current_time - last_save > 10000) {
```

```
      saveSettings();
      last_save = current_time;
    }
  } else if (time_diff >= 1000) {
    device_state.last_energy_update = current_time;
  }
}
```

Функція updateEnergyCounter() реалізує алгоритм інтегрування потужності для обчислення спожитої енергії. Вона використовує системний лічильник millis() для визначення часових інтервалів між вимірюваннями. Обчислення виконуються лише за умови, що реле знаходиться у ввімкненому стані та пройшов мінімум одну секунду з попереднього оновлення. Часовий інтервал перетворюється у години шляхом ділення на 3600000 мілісекунд, після чого розраховується приріст енергії як добуток потужності у кіловатах на годину. Для забезпечення збереження даних у випадку раптового відключення живлення функція автоматично викликає saveSettings() кожні 10 секунд. Навіть коли реле вимкнене, функція оновлює час останнього виміру для підтримки синхронізації.

Для безпеки експлуатації пристрою реалізовано функцію checkCurrentLimit(), яка здійснює безперервний моніторинг струму та забезпечує автоматичне відключення у випадку перевищення встановленого ліміту.

Лістинг 3.6 — Код функції checkCurrentLimit

```
void checkCurrentLimit() {
  bool was_exceeded = device_state.limit_exceeded;
  device_state.limit_exceeded = device_state.current > device_state.current_limit;

  if (device_state.limit_exceeded && !was_exceeded) {
    Serial.printf("CURRENT LIMIT EXCEEDED! %.2f A > %.1f A\n",
                  device_state.current, device_state.current_limit);
    if (device_state.relay_state) {
      device_state.relay_state = false;
      digitalWrite(RELAY_PIN, LOW);
      Serial.println("Relay turned OFF due to overcurrent");
    }
  } else if (!device_state.limit_exceeded && was_exceeded) {
    Serial.println("Current back to normal");
  }
}
```

Функція checkCurrentLimit() виконує роль механізму захисту від перевантаження струмом. Перш за все зберігається попереднє значення встановленого ліміту перевищення струму, після чого виконується порівняння поточного значення струму з встановленим порогом device_state.current_limit. При виявленні перевищення ліміту функція надсилає повідомлення з точними значеннями струму та автоматично вимикає реле за допомогою digitalWrite(RELAY_PIN, LOW). Такий підхід дозволяє захистити обладнання від перевантаження.

3.3 Програмна реалізація Telegram-бота

Програмна реалізація Telegram-бота є ключовою складовою системи дистанційного керування розумною розеткою. Бот написано на мові Python з використанням бібліотеки python-telegram-bot та модуля requests для HTTP-комунікації з мікроконтролером ESP32.

Архітектура бота побудована за принципом обробки команд та callback-запитів, що забезпечує інтуїтивний інтерфейс користувача та надійну взаємодію з апаратною частиною системи. Центральним елементом програмної реалізації є клас SmartSocketBot, який інкапсулює всю логіку взаємодії з користувачем та ESP32. Даний клас містить багатомовну підтримку інтерфейсу, систему управління користувацькими налаштуваннями та методи для форматування повідомлень.

Важливою особливістю реалізації є асинхронна обробка запитів. Для забезпечення стабільного зв'язку між ботом та мікроконтролером було реалізовано універсальний метод HTTP-комунікації. Цей метод обробляє всі типи запитів до ESP32 та забезпечує належну обробку помилок з'єднання.

Лістинг 3.7 — Код функції make_request

```
async def make_request(self, endpoint, method='GET', data=None):
    try:
        url = f"{ESP32_IP}/api/{endpoint}"
        if method == 'GET':
            response = requests.get(url, timeout=REQUEST_TIMEOUT)
        elif method == 'POST':
            headers = {'Content-Type': 'application/json'}
```

```
            response = requests.post(url, json=data, headers=headers,
timeout=REQUEST_TIMEOUT)
        if response.status_code == 200:
            return response.json()
        else:
            logger.error(f"HTTP {response.status_code}: {response.text}")
            return None
    except requests.exceptions.RequestException as e:
        logger.error(f"Request error: {e}")
        return None
```

Функція make_request() дозволяє встановлювати комунікацію між ботом та ESP32. Функція приймає параметр endpoint для визначення API-точки, method для типу HTTP-запиту та data для передачі JSON-даних. Функція автоматично формує повну URL-адресу, встановлює необхідні заголовки для POST-запитів та обробляє винятки мережевого рівня. У випадку успішного виконання запиту функція повертає JSON-відповідь, а при помилках записує інформацію в лог та повертає None.

Наступним важливим кроком у реалізації бота є обробка команд від користувача. Основною точкою входу є обробник команди /start, який ініціалізує взаємодію з новим користувачем.

Лістинг 3.8 — Код функції start_command

```
async def start_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    user_id = update.effective_user.id
    if user_id not in user_settings:
        user_settings[user_id] = {
            'lang': 'ua',
            'waiting_for_limit': False,
            'menu_shown': False
        }
    welcome_text = bot.get_text(user_id, 'welcome')
    keyboard = bot.get_main_menu_keyboard(user_id
    await update.message.reply_ (
        welcome_text,
        reply_markup=keyboard
    )
```

Функція start_command виконує ініціалізацію нового користувача в системі. Спочатку вона отримує унікальний ідентифікатор користувача та перевіряє його наявність у глобальному словнику user_settings. Якщо користувач новий,

створюється запис з базовими налаштуваннями: українська мова за замовчуванням, стан очікування введення ліміту та стан відображення меню. Після ініціалізації формується привітальне повідомлення відповідною мовою та створюється inline-клавіатура з основними функціями системи.

Для забезпечення швидкого доступу до основних функцій керування було реалізовано систему inline-клавіатури. Ключовою є функція створення головного меню.

Лістинг 3.9 — Код функції start_command

```
def get_main_menu_keyboard(self, user_id):
    lang = self.get_user_lang(user_id)
    buttons = self.texts[lang]['buttons']
    return InlineKeyboardMarkup([
        [
            InlineKeyboardButton(buttons['turn_on'], callback_data='turn_on'),
            InlineKeyboardButton(buttons['turn_off'], callback_data='turn_off')
        ],
            InlineKeyboardButton(buttons['status'], callback_data='status')
        ],
        [
            InlineKeyboardButton(buttons['show_menu'], callback_data='show_menu')
        ]
    ])
```

Функція get_main_menu_keyboard() динамічно створює inline-клавіатуру відповідно до поточної мови користувача. Клавіатура організована у логічні ряди: перший ряд містить кнопки увімкнення та вимкнення розетки, другий — кнопку отримання статусу, третій — кнопку відображення розширеного меню. Усі кнопки мають відповідні callback_data для ідентифікації натиснутої дії в обробнику callback-запитів.

Центральною частиною функціональності Telegram-боту є обробка callback-запитів від inline-кнопок. Цей обробник реалізує основну логіку взаємодії користувача з системою.

Лістинг 3.10 — Код функції button_callback

```
async def button_callback(update: Update, context: ContextTypes.DEFAULT_TYPE):
    query = update.callback_query
    await query.answer()
```

```
    user_id = query.from_user.id
    callback_data = query.data
    if callback_data == 'turn_on':
        result = await bot.control_relay(True)
        if result and result.get('success'):
            message = bot.get_text(user_id, 'turned_on')
        else:
            message = bot.get_text(user_id, 'connection_error')
        await context.bot.send_message(chat_id=user_id, text=message,
parse_mode='HTML')
    elif callback_data == 'turn_off':
        result = await bot.control_relay(False)
        if result and result.get('success'):
            message = bot.get_text(user_id, 'turned_off')
        else:
            message = bot.get_text(user_id, 'connection_error')
        await context.bot.send_message(chat_id=user_id, text=message,
parse_mode='HTML')
```

Функція button_callback() відіграє роль обробника команд після натискань inline-кнопок. Спочатку вона відповідає на callback-запит для уникнення тайм-ауту, потім аналізує тип дії за параметром callback_data. Для команд увімкнення та вимкнення розетки функція викликає метод control_relay() з відповідним параметром стану, перевіряє результат виконання та формує повідомлення користувачу про успіх або помилку операції.

Важливою складовою системи є функція отримання та форматування статусу пристрою. Вона забезпечує детальне відображення всіх параметрів розумної розетки.

Лістинг 3.11 — Код функції format_status_message

```
def format_status_message(self, user_id, data):
    if not data or not data.get('success'):
        return self.get_text(user_id, 'connection_error')
    text = self.get_text(user_id, 'status_title') + "\n\n"
    relay_state = data.get('relay_state', False)
    if relay_state:
        text += self.get_text(user_id, 'relay_on') + "\n"
    else:
        text += self.get_text(user_id, 'relay_off') + "\n"
    text += "\n"
    text += self.get_text(user_id, 'current').format(data.get('current', 0)) +
"\n"
    text += self.get_text(user_id, 'voltage').format(data.get('voltage', 0)) +
"\n"
    text += self.get_text(user_id, 'power').format(data.get('power', 0)) + "\n"
```

```
    text += self.get_text(user_id, 'energy').format(data.get('energy', 0)) +
"\n\n"
    return text
```

Функція format_status_message() перетворює JSON-дані, отримані від ESP32, в структуроване текстове повідомлення. Функція спочатку перевіряє валідність отриманих даних, потім послідовно формує рядки з інформацією про стан реле, електричні параметри (струм, напруга, потужність, споживана енергія), час роботи та стан WiFi-з'єднання. Всі значення форматуються з урахуванням одиниць вимірювання та локалізації повідомлень.

Для забезпечення безпеки експлуатації розумної розетки реалізовано функцію встановлення ліміту струму. Ця функція дозволяє користувачу задати максимально допустимий струм споживання.

Лістинг 3.12 — Код функції limit_command

```
async def limit_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    user_id = update.effective_user.id
    if context.args:
        try:
            limit_value = float(context.args[0])
            if 0.5 <= limit_value <= 25:
                processing_msg = await
update.message.reply_html(bot.get_text(user_id, 'processing'))
                result = await bot.set_current_limit(limit_value)
                if result and result.get('success'):
                    message = bot.get_text(user_id,
'limit_set').format(limit_value)
                else:
                    message = bot.get_text(user_id, 'connection_error')
                await context.bot.edit_message_text(
                    chat_id=user_id,
                    message_id=processing_msg.message_id,
                    text=message,
                    parse_mode='HTML'
                )
            else:
                await update.message.reply_html(bot.get_text(user_id,
'invalid_limit'))
        except ValueError:
            await update.message.reply_html(bot.get_text(user_id,
'invalid_limit'))
    else:
        user_settings[user_id]['waiting_for_limit'] = True
        await update.message.reply_html(bot.get_text(user_id, 'enter_limit'))
```

Функція limit_command() реалізує два сценарії встановлення ліміту струму. Перший сценарій передбачає передачу значення ліміту як аргументу команди, наприклад, /limit 10.5). У цьому випадку функція перевіряє валідність числового значення та його діапазон (від 0.5 до 25 ампер), після чого надсилає запит до ESP32. Другий сценарій активується при виклику команди без аргументів, а саме функція встановлює прапорець очікування введення ліміту та просить користувача ввести значення у наступному повідомленні.

Комунікація між ботом та ESP32 організована через набір спеціалізованих методів для різних типів операцій. Ключовими є методи керування реле та отримання статусу.

Лістинг 3.13 — Код функцій для комунікації з ESP32

```
async def get_status(self):
    return await self.make_request('status')
async def control_relay(self, state):
    return await self.make_request('relay', 'POST', {'state': state})
async def reset_energy(self):
    return await self.make_request('reset', 'POST')
async def set_current_limit(self, limit):
    return await self.make_request('limit', 'POST', {'limit': limit})
```

Ці функції відповідають за взаємодію з ESP32 через HTTP API. Вони асинхронно відправляють запити для отримання статусу пристрою, дистанційного керування реле, скидання лічильника енергії та установки нового ліміту струму. При передачі даних використовуються відповідні HTTP-методи (GET або POST) із JSON-форматом.

Система багатомовної підтримки реалізована через централізований словник текстових повідомлень та методи їх отримання. Це забезпечує легку локалізацію інтерфейсу.

Лістинг 3.14 — Код функцій для отримання мови користувача та текстів

```
def get_user_lang(self, user_id):
    return user_settings.get(user_id, {}).get('lang', 'ua')
def get_text(self, user_id, key):
    lang = self.get_user_lang(user_id)
    return self.texts[lang][key]
```

Ці функції забезпечують динамічне отримання мовних налаштувань та відповідних текстів, що використовуються в повідомленнях Telegram-бота. Функція get_user_lang() повертає поточну мову користувача (за замовчуванням українська), а функція get_text() отримує локалізований текст за ключем з відповідного мовного розділу словника texts.

Завершальним етапом ініціалізації бота є налаштування обробників команд та запуск Telegram-бота.

Лістинг 3.15 — Код функції main

```
def main():
    logger.info("Запуск SmartSocket Bot...")
    application = ApplicationBuilder().token(BOT_TOKEN).build()
    application.add_handler(CommandHandler("start", start_command))
    application.add_handler(CommandHandler("menu", menu_command))
    application.add_handler(CommandHandler("status", status_command))
    application.add_handler(CallbackQueryHandler(button_callback))
    application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND,
handle_text))
    asyncio.get_event_loop().run_until_complete(set_bot_commands(application))
    logger.info("SmartSocket Bot готовий до роботи!")
    application.run_polling(allowed_updates=Update.ALL_TYPES)
```

Функція main() є точкою входу в програму та виконує повну ініціалізацію Telegram-бота. Спочатку створюється екземпляр Application з токеном бота, потім реєструються всі обробники команд, callback-запитів та текстових повідомлень. Важливим етапом є встановлення команд меню бота через Telegram API, що дозволяє користувачам бачити доступні команди у інтерфейсі додатку. Завершується ініціалізація запуском механізму long polling для отримання оновлень від Telegram API, що забезпечує постійну роботу бота у режимі очікування повідомлень користувачів.

3.4 Інтерфейс Telegram-бота для керування розумною розеткою

Користувацький інтерфейс системи керування реалізовано у вигляді Telegram-бота, що відповідає за дистанційне управління розеткою, моніторинг її роботи та налаштування параметрів пристрою. Базуючись на інтуїтивно зрозумілому наборі команд та inline-клавіатурі, бот дозволяє користувачеві легко

змінювати стан розетки (увімкнути/вимкнути), перевіряти електричні параметри (струм, напруга, потужність, енергія) та встановлювати ліміт струму.

Коли користувач надсилає команду /start Telegram-боту, бот перевіряє, чи є профіль користувача у глобальній базі налаштувань. Якщо профіль відсутній, то його буде створено. Після цього бот надсилає користувачу привітальне повідомлення та виводить інтерактивну inline-клавіатуру з кнопками “Увімкнути”, “Вимкнути”, “Статус” та “Показати меню”. Результат команди /start зображено на рисунку 3.1.

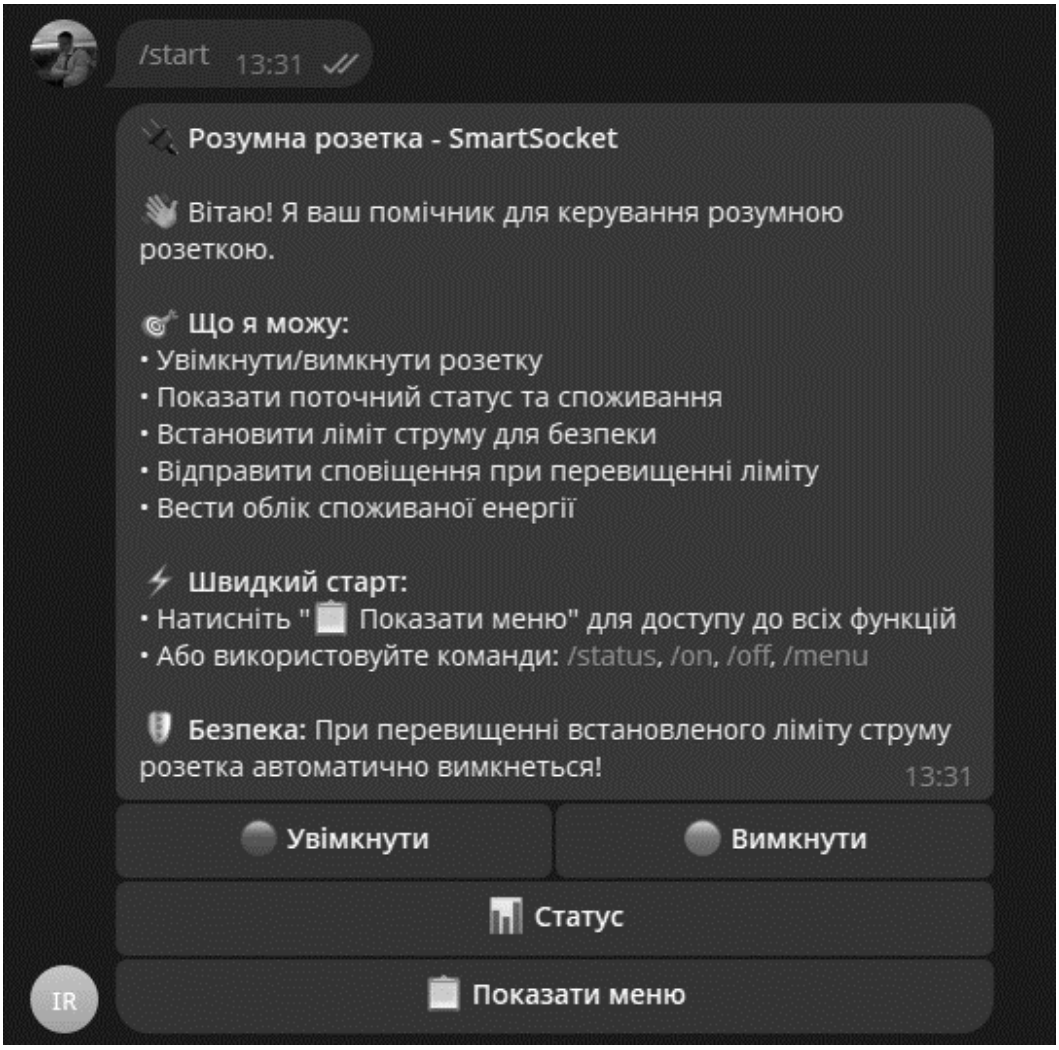

Рисунок 3.1 — Результат команди /start

Якщо користувач хоче увімкнути розетку, йому потрібно натиснути кнопку «Увімкнути» або надіслати команду /on. Коли клієнт надсилає відповідну команду Telegram-боту, він надсилає callback-запит, який обробляється функцією

button_callback(). Бот перевіряє, чи пристрій може увімкнути розетку (наприклад, чи не перевищено ліміт струму), і якщо все гаразд, викликає функцію control_relay(True), що відправляє POST-запит до ESP32 для увімкнення розетки. Після цього користувач отримує повідомлення «Розетка увімкнена». Аналогічно, у випадку коли користувач хоче вимкнути розетку і надсилає відповідну команду або натискає кнопку. Результат натискання на кнопки «Увімкнути» та «Вимкнути» зображено на рисунку 3.2.

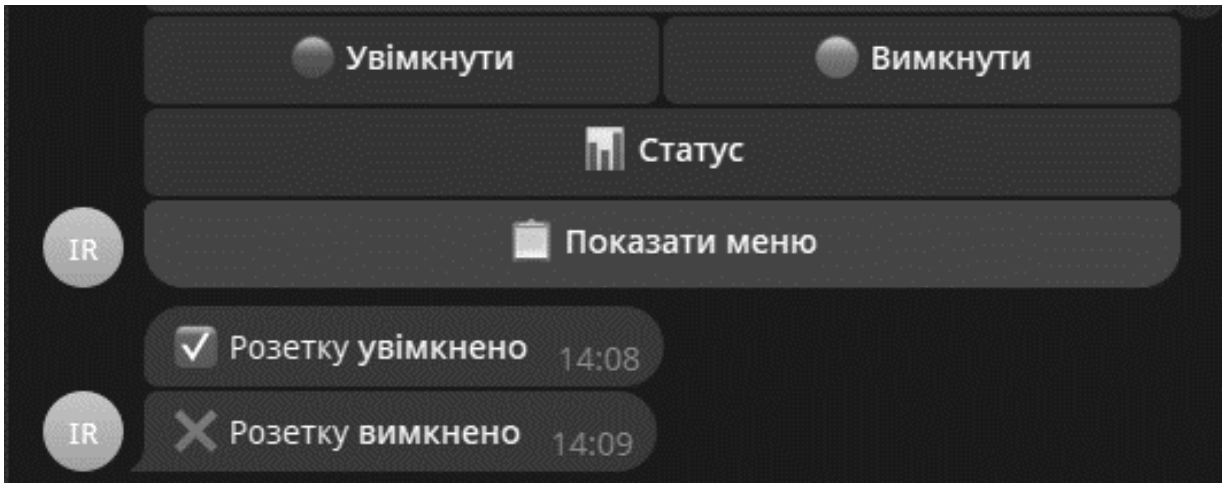

Рисунок 3.2 — Результат натискання на кнопки управління розеткою

У випадку, коли користувач бажає переглянути поточний стан розумної розетки йому потрібно надіслати боту команду /status або натиснути на відповідну кнопку «Статус». Після отримання команди бот запускає виконання асинхронної функцію, яка надсилає GET-запит до ESP32 для отримання поточного стану пристрою. У відповіді містяться ключові параметри, такі як стан реле (увімкнене чи вимкнене), значення струму, напруги, потужності та накопичена енергія. Потім ці дані обробляються функцією format_status_message() для перетворення у зручне для читання користувачем повідомлення, він отримує у вигляді структурованого тексту з усіма необхідними даними. Результат команди /status зображено на рисунку 3.3.

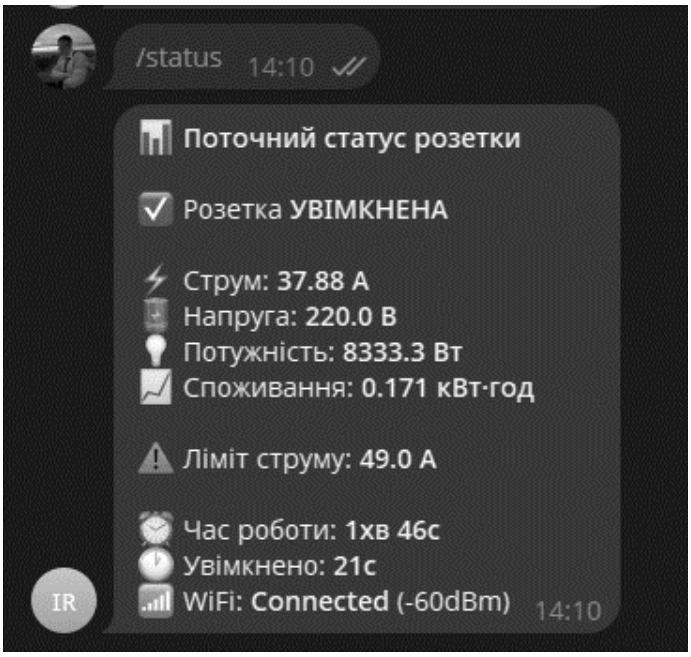

Рисунок 3.3 — Результат команди /status

Команда /limit призначена для зміни конфігураційних параметрів пристрою, а саме для встановлення нового ліміту струму. Коли користувач надсилає команду /limit або натискає відповідну кнопку з меню, бот надсилає користувачу повідомлення, де просить ввести нове значення у вказаному діапазоні. Якщо значення правильне, бот відправляє POST-запит через функцію set_current_limit(limit) для оновлення налаштувань пристрою. Після чого користувач отримує повідомлення про успішну зміну ліміту або інформацію про помилку у випадку невірного формату, великого числа, яке виходить за вказаний діапазон, чи проблеми з підключенням. Результат команди /limit зображено на рисунку 3.4.

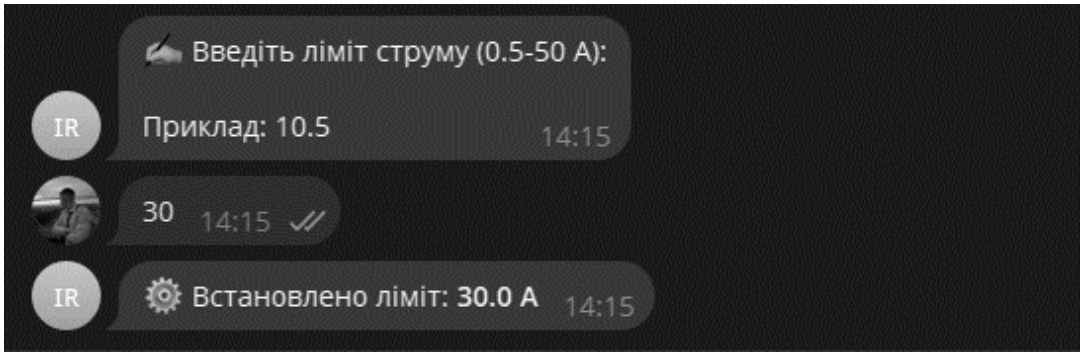

Рисунок 3.4 — Результат команди /limit

Якщо користувач не впевнений, яку команду використовувати або як правильно її застосовувати, він завжди зможе відправити боту команду /help або натиснути на відповідну кнопку «Довідка». Команда /help дозволяє отримати детальний опис всіх доступних команд і функцій бота. Опис команд та функцій бота, який користувач побачить, зображено на рисунку 3.5.

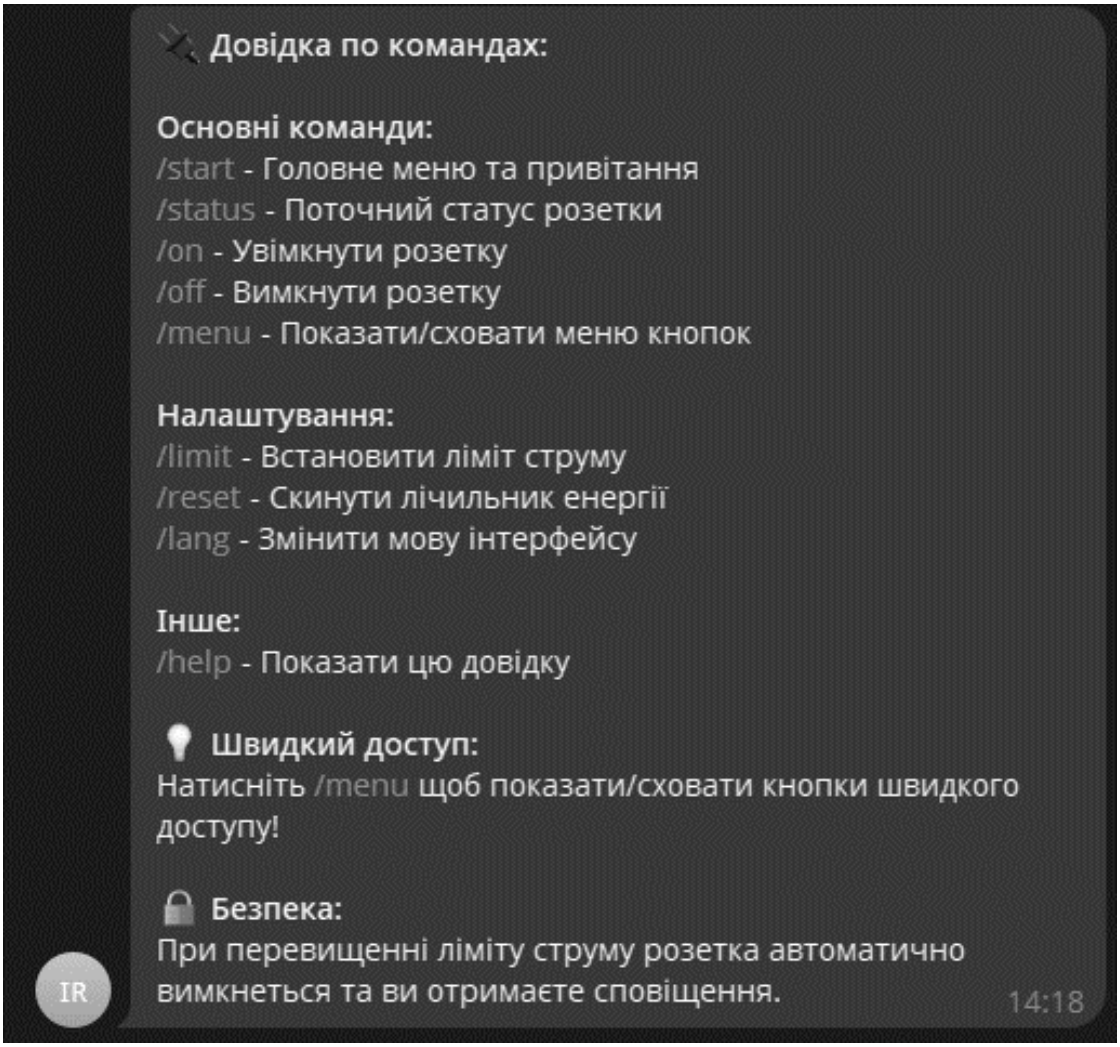

Рисунок 3.5 — Результат команди /help

Підтримка англійської мови робить систему доступнішою для користувачів з різних країн. Це забезпечує зручність використання для тих, хто не розуміє українську. Інтерфейс англійською дозволяє швидко орієнтуватися і отримувати зрозумілі повідомлення без складних інструкцій. Завдяки цьому, IoT-пристрій є універсальним і простим у використанні. Англомовний інтерфейс зображено на рисунку 3.6.

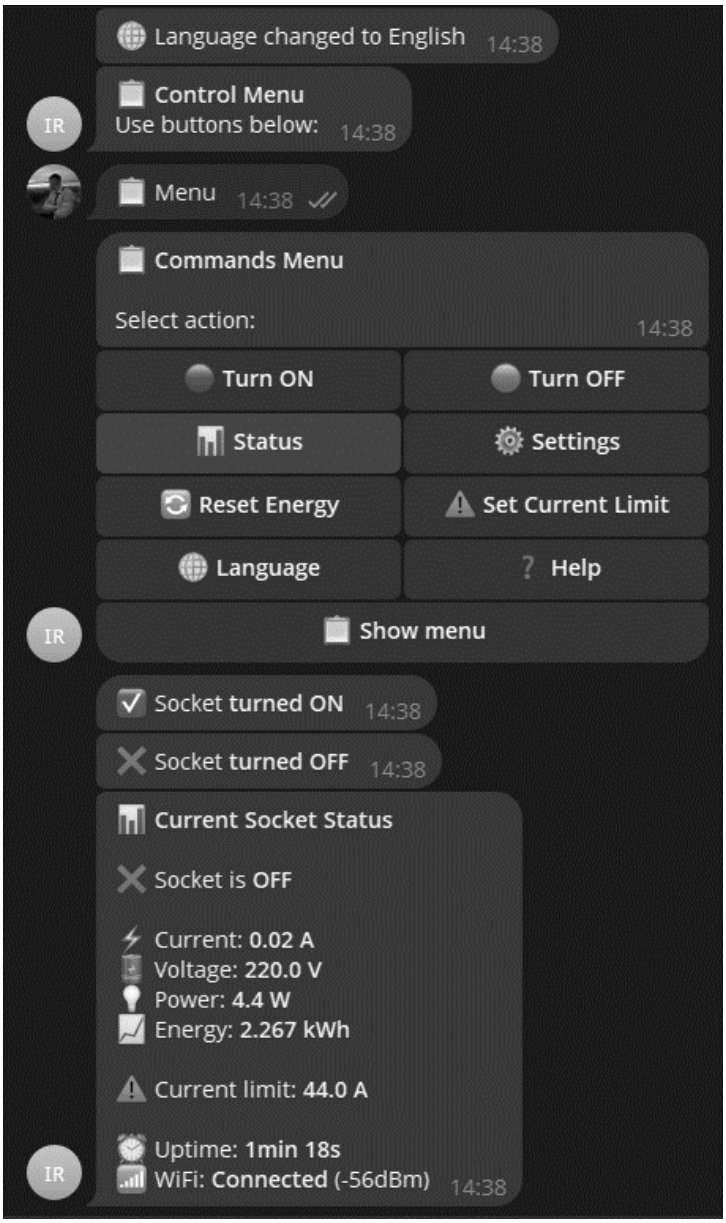

Рисунок 3.6 — Англомовний інтерфейс Telegram-бота

Таким чином, використання Telegram-бота для керування розумною розеткою надає можливість зручного та простого віддалено управління. Кожна команда виконується асинхронно, що дозволяє отримувати миттєвий зворотній зв’язок. Інтерфейс пристрою спроєктовано з метою забезпечення простоти та легкості у використанні за допомогою простих команд та довідки про них. Це дозволяє користувачу швидко орієнтуватися серед доступних команд Telegram-бота. Такий підхід сприяє ефективному керуванню пристроєм віддалено, що є надзвичайно важливим компонентом сучасних IoT-систем.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Після реалізації всіх компонентів апаратно-програмного комплексу IoT-пристрою з керуванням через Telegram-бота було проведено комплексне тестування функціональності системи з метою перевірки її працездатності, надійності та відповідності вимогам, визначеним на етапі проєктування. Тестування спрямоване на підтвердження коректності реалізованого функціоналу, оцінку ефективності взаємодії компонентів та виявлення потенційних проблем експлуатації. Методологія тестування базувалась на реальних сценаріях використання пристрою в побутових умовах. Особливу увагу приділено перевірці надійності комунікації між ESP32 та Telegram API, точності моніторингу електричних параметрів, а також стійкості системи до зовнішніх факторів впливу.

Для тестування було обрано павербанк із LED-індикаторами рівня заряду, який дозволяє візуально підтверджувати подачу або відключення живлення. Такий підхід забезпечив швидкий зворотній зв'язок при оцінці реакції системи на команди користувача.

4.1 Розробка алгоритму тестування

Тестування проводилось згідно з попередньо складеним алгоритмом, який охоплює усі ключові етапи взаємодії між користувачем, Telegram-ботом, мікроконтролером та релейним модулем, що відповідає за комутацію живлення.

Основні цілі тестування:

— перевірка стабільності зв’язку між Telegram-ботом і пристроєм;

— реакція пристрою на стандартні команди;

— визначення швидкості реакції системи на запити користувача;

— виявлення потенційних помилок або зависань у разі втрати інтернет-з'єднання;

— перевірка збереження логіки керування після перезавантаження мікроконтролера.

Методика включала тестування в реальному середовищі з використанням стандартного Wi-Fi з'єднання 2.4 ГГц, а також Telegram-бота, розгорнутого через API Telegram. Розетка підключалась до стандартного джерела 220 В через силове реле, а павербанк виступав у ролі навантаження. Сценарії тестування реалізовувались як вручну, так і шляхом автоматичної послідовної подачі команд на інтерфейс бота. Блок-схема алгоритму тестування зображена на рисунку 18.

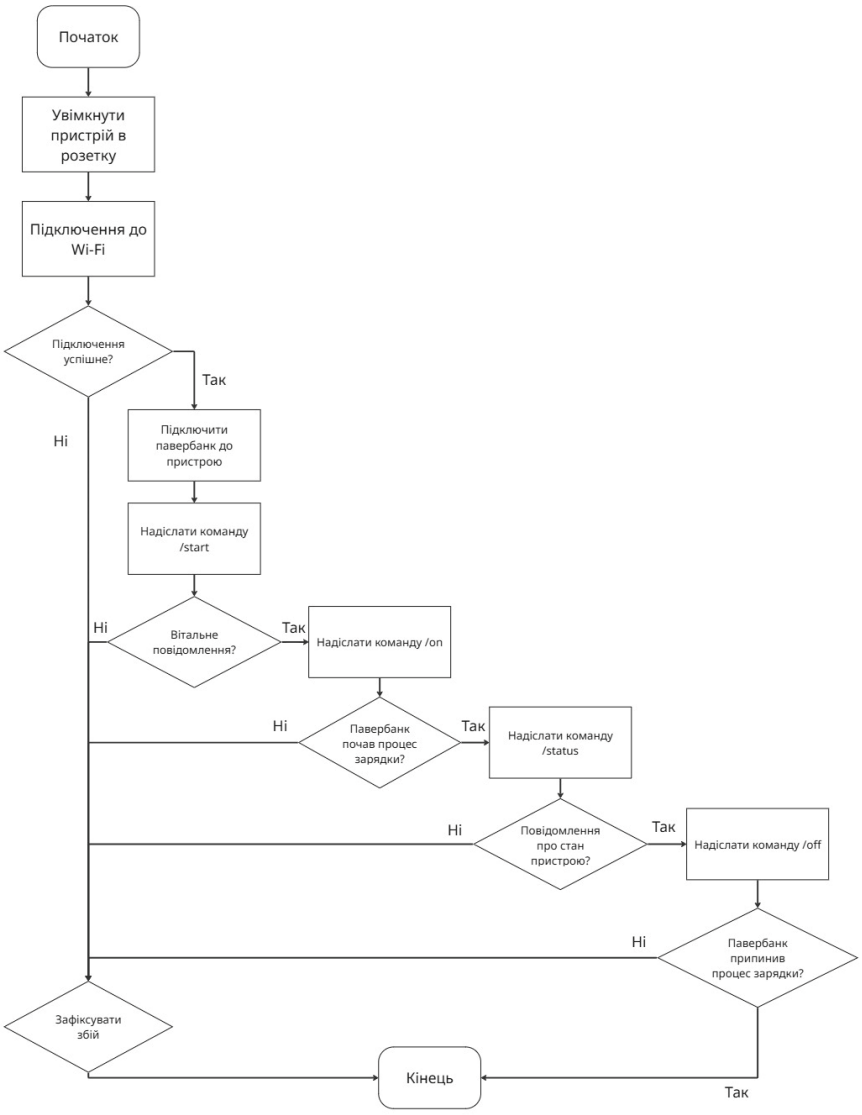

Рисунок 18 — Блок-схема алгоритму тестування IoT-пристрою

4.2 Тестування

Процес тестування IoT-пристрою із Telegram-ботом відбувався у кілька етапів, кожен з яких був спрямований на перевірку конкретних функціональних

блоків системи. Було застосовано як модульне тестування, так і інтеграційне, що дозволило оцінити не лише окрему роботу частин пристрою, але й взаємодію між ними. Перед початком тестування було проведено базову перевірку працездатності елементів.

На початковому етапі тестування особливу увагу було приділено безпечності процесу. Щоб уникнути підключення побутових приладів, які могли б бути пошкоджені у разі помилкової роботи пристрою, було вирішено використовувати портативний павербанк як тестове навантаження. Павербанки мають вбудовані світлодіодні індикатори, які чітко сигналізують про те, чи надходить на них струм: індикатор починає підсвічуватись у разі зарядки, і гаснути у разі припинення подачі живлення. Таким чином, павербанк слугував простим і ефективним засобом візуального контролю стану розетки.

Процедура тестування почалася з підключення павербанка до пристрою. Далі через Telegram надсилалася команда /on, після чого фіксувалася реакція павербанка — його індикатор миттєво вмикався, сигналізуючи про початок зарядки, що свідчило про подачу напруги на вихід розетки. У відповідь на команду /off індикатор згасав, що підтверджувало відключення реле і припинення подачі струму. Таке наочне візуальне підтвердження дало змогу швидко і безпомилково перевіряти реакцію пристрою. Тестування повторювалося багаторазово з різними інтервалами та варіантами сценаріїв — надсилалися команди /on, /off у швидкій послідовності, з затримками та без, а також команди однакового типу кілька разів поспіль (наприклад, /on, /on). У результаті пристрій стабільно виконував команди без помилок, подвійних спрацьовувань чи зависань. Усі реакції індикатора павербанка підтверджували правильну роботу реле. Окрім цього, тестування дало змогу перевірити затримку між надсиланням команди та фізичним увімкненням/вимкненням живлення — у середньому вона становила 1.2 секунди, що є прийнятним для побутового пристрою.

Таким чином, використання павербанка дозволило безпечно й ефективно перевірити функціональність розетки, підтвердити її готовність до подальших

етапів випробувань із реальними приладами та впевнитись у стабільній роботі керування за допомогою Telegram-бота.

4.3 Аналіз результатів проведення тестування

Проведене тестування допомогло здійснити комплексну оцінку функціонування IoT-пристрою з керуванням через Telegram у різних умовах експлуатації. Система продемонструвала високу стабільність, швидку реакцію на команди користувача та коректну роботу при обробці як стандартних, так і критичних сценаріїв. Завдяки використанню мікроконтролера ESP32 та бібліотек для роботи з Telegram API, вдалось досягти надійної взаємодії між апаратною та програмною частиною системи. Навіть при спробах навмисного перевантаження системи, наприклад, надсилання кількох команд підряд, введення помилкових запитів, бот реагував, надаючи коректний зворотний зв’язок та ігноруючи навмисні запити.

Telegram-бот зберігає логіку взаємодії, наприклад, якщо користувач надсилає команду «/on», реле спрацьовує і користувач одразу отримує повідомлення про зміну стану. Це дозволяє користувачу впевнено контролювати пристрій, незалежно від того, чи перебуває він в дома, чи він поза містом чи на роботі. Така гнучкість і віддалене керування — одна з ключових переваг розробленого IoT-пристрою.

Однак, попри стабільну роботу, було виявлено і певні обмеження. Найважливішим з них є повна залежність від доступу до Інтернету. У разі втрати Wi-Fi зв’язку або відсутності мобільного Інтернету користувач втрачає можливість взаємодіяти з розеткою. Крім того, наразі немає програмно реалізованого механізму автоматичного оновлення статусу. Тобто, якщо стан реле змінюється поза межами комунікації з Telegram (наприклад, у результаті програмної помилки), бот не повідомляє про це автоматично. Таким чином, результати аналізу показують, що проєкт є працездатним, ефективним і з високим потенціалом до майбутнього маштабування у «розумний дім».

ВИСНОВКИ

У процесі виконання бакалаврської дипломної роботи було успішно вирішено поставлені задачі щодо розробки IoT пристрою з чат-ботом для керування електричними пристроями. Під час проведення аналіз існуючих технологій дистанційного керування електричними приладами було виявлено основні недоліки сучасних IoT-рішень: високу вартість, складність налаштування та необхідність використання спеціалізованого програмного забезпечення.

Мною було прийнято розробити нове рішення, яке б включало всі необхідні функції для ефективного керування побутовими електричними пристроями за допомогою бота у популярному месенджері Telegram. Проведений аналіз підтвердив, що існує значна потреба у розробці доступного IoT-пристрою, який дозволить контролювати енергоспоживання та забезпечить зручне дистанційне керування.

Розроблена система демонструє ефективність використання мікроконтролера ESP32 та інтеграції з Telegram Bot API, що дозволило створити простий інтуїтивний інтерфейс керування без необхідності розробки окремого мобільного додатку.

Експериментальна перевірка показала стабільну роботу розробленого пристрою в умовах домашньої Wi-Fi мережі, при цьому швидкість обробки команд користувача не перевищувала 2-3 секунди. Система продемонструвала надійність роботи протягом тестового періоду без критичних збоїв або втрати з'єднання. Пристрій успішно виконує функції дистанційного увімкнення та вимкнення електропристроїв і може бути легко масштабований або інтегрований у системи «розумного дому».

Подальший розвиток системи може включати додавання сенсорних модулів для моніторингу параметрів навколишнього середовища, а також впровадження алгоритмів машинного навчання для адаптивного управління енергоспоживанням на основі поведінки користувача.

Реалізований IoT-засіб забезпечує комплексне вирішення завдань дистанційного керування побутовими електричними пристроями з дотриманням вимог безпеки та функціональності. Етапи розробки включали проектування апаратної архітектури, програмну реалізацію мікроконтролерної системи, створення програмного інтерфейсу Telegram-бота, налаштування мережевих протоколів та комплексне тестування функціональних можливостей.

У результаті, мету бакалаврської дипломної роботи було досягнуто. Реалізовано зручний, інтуїтивно зрозумілий та повнофункціональний IoT-пристрій, що дозволяє користувачам контролювати побутові електричні прилади та надає можливість управління енергоспоживанням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. IoT засіб з чат-ботом для керування електричними пристроями // Звездін Р. С., Савицька Л. А., Богомолов С. В. Матеріали Міжнародної науково-практичної Інтернет-конференції студентів, аспірантів та молодих науковців (Вінниця, 2025 р.) [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/24533/20297

2. Internet of Things, IoT : вебсайт. URL: https://www.it.ua/knowledge-base/technology-innovation/internet-veschej-internet-of-things-iot (дата звернення 21.03.2025)

3. Інтернет речей : вебсайт. URL: https://itsinternetofthings.blogspot.com/p/blog-page_50.html (дата звернення 21.03.2025)

4. Архітектура і технології IoT [Електронний ресурс]. Режим доступу: https://learn.ztu.edu.ua/pluginfile.php/205968/mod_resource/content/2/%D0%9B%D0%B5%D0%BA_1.pdf

5. Прогноз тенденцій розвитку інтернету речей у 2025 році : вебсайт. URL: https://jooby.eu/uk/blog/prognoz-tendencij-rozvitku-internetu-rechej-u-2025-roci/ (дата звернення 24.03.2025)

6. Розумний дім : вебсайт. URL: https://uk.wikipedia.org/wiki/Розумний_дім (дата звернення 25.03.2025)

7. Розумний будинок : вебсайт. URL: https://livolo.in.ua/smart-home/ (дата звернення 25.03.2025)

8. Технології «розумного дому» в українських новобудовах: сучасні можливості та перспективи : вебсайт. URL: https://riel.ua/blogs/tekhnologiyi-rozumnogo-domu-v-ukrayinskikh-novobudovakh-suchasni-mozhlivosti-ta-perspektivi (дата звернення 30.03.2025)

9. Інтернет речей в електроніці [Електронний ресурс]. Режим доступу: https://elib.lntu.edu.ua/sites/default/files/elib_upload/page10.html

10. Мікроконтролер : вебсайт. URL: https://uk.wikipedia.org/wiki/Мікроконтролер (дата звернення 02.04.2025)

11. What is MQTT? : вебсайт. URL: https://aws.amazon.com/what-is/mqtt/?nc1=h_ls# (дата звернення 05.04.2025).

12. Офіційна документація MDN Web Docs (Mozilla Developer Network). An overview of HTTP : вебсайт. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview (дата звернення 05.04.2025)

13. Різниця між HTTP та HTTPs : вебсайт. URL: https://hostiq.ua/wiki/ukr/http-https/ (дата звернення 05.04.2025)

14. WebSocket : вебсайт. URL: https://uk.javascript.info/websocket (дата звернення 05.04.2025)

15. WebSockets: навіщо потрібні та як з ними працювати : вебсайт. URL: https://proit.ua/websockets-navishcho-potribni-ta-iak-z-nimi-pratsiuvati-2/ (дата звернення 05.04.2025)

16. Чи можна використовувати API Telegram? : вебсайт. URL: https://telegramm.com.ua/pytannya-ta-vidpovidi/chi-mozhna-vykorystovuvaty-api-telegram/ (дата звернення 07.04.2025)

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н., проф. Азаров О.Д.

«21» березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

«IoT засіб з чат-ботом для керування електричними пристроями»

08-54.БКР.027.00.000 ТЗ

Науковий керівник: к.т.н., доц. каф. ОТ

_____Савицька Л. А.

Студент групи 2КІ-21б

_____Звездін Р. С.

м. Вінниця — 2025

1 Підстава для використання бакалаврської кваліфікаційної роботи (БКР)

1.1 У сучасному світі технології «розумного дому» стрімко розвиваються та стають невід'ємною частиною побуту. Вони не лише забезпечують комфорт і зручність, але й допомагають економити ресурси, підвищують рівень безпеки та сприяють покращенню якості життя користувачів. Актуальність автоматизації управління електричними пристроями у побуті стає все більш важливою у сучасному світі.

1.2 Наказ про затвердження теми кваліфікаційної роботи.

2 Мета БКР і призначення розробки

2.1 Мета роботи — розробка доступного та зручного IoT-пристрою з інтегрованим чат-ботом для керування електричними приладами через популярний месенджер Telegram.

2.2 Призначення розробки — створення функціональної системи віддаленого керування електроприладами з можливостями моніторингу енергоспоживання, програмування режимів роботи та забезпечення електробезпеки.

3 Вихідні дані для виконання КБКР

3.1 Проведення аналізу сучасних тенденцій розвитку IoT у системах розумного дому, архітектури IoT-систем та огляд платформ для побудови IoT-пристроїв;

3.2 Дослідження протоколів зв'язку в IoT (MQTT, HTTP, HTTPS, WebSockets) та можливостей інтеграції з Telegram API;

3.3 Вибір оптимальної апаратної платформи та компонентів для IoT-пристрою розумної розетки;

3.4 Розробка схеми з'єднань пристрою з урахуванням вимог ізоляції, безпеки та електротехнічних стандартів;

3.5 Програмна реалізація мікроконтролера ESP32 з функціями гнучкого налаштування Wi-Fi, взаємодії з Telegram API та виявлення струму;

3.6 Розробка архітектури та інтерфейсу Telegram-бота для керування розумною розеткою;

3.7 Проведення комплексного тестування системи та аналіз отриманих результатів.

4 Вимоги до виконання КБКР

Головна вимога — створити безпечну, надійну та функціональну систему IoT пристрою з чат-ботом для керування електричними пристроями розумної розетки, яка відповідає стандартам електробезпеки та забезпечує стабільну роботу з різними типами електроприладів потужністю до 16А (3,5 кВт).

5 Етапи КБКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

№ етапу	Назва етапу	Термін виконання	Очікувані результати
1	Аналіз і теоретичні засади розробки IoT-систем керування електричними пристроями	21.03.25 – 30.03.25	Аналітичний огляд
2	Розробка IoT-системи розумної розетки	31.03.25 – 13.04.25	Розділ 2
3	Проєктування IoT-пристрою розумної розетки	14.04.25 – 27.04.25	Розділ 3
4	Тестування та аналіз результатів	14.04.25 – 27.04.25	Розділ 4
5	Оформлення пояснювальної записки, графічного матеріалу та презентації	28.04.25 – 24.05.25	Пояснювальна записка, графічного матеріалу та презентація

6 Матеріали, що подаються до захисту КБКР

До захисту подаються: пояснювальна записка БКР, графічні матеріали, протокол попереднього захисту на кафедрі, відгук наукового керівника, відгук

рецензента, протоколи складання державних екзаменів, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення вимогам.

7 Порядок контролю виконання та захисту БКР

Контроль виконання етапів здійснюється науковим керівником згідно з встановленими термінами. Захист БКР проходить на засіданні екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлення та порядок виконання КБКР

8.1 При оформленні БКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія») — кафедра обчислювальної техніки, ВНТУ 2023.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:

IoT засіб з чат-ботом для керування електричними пристроями

Тип роботи: Бакалаврська кваліфікаційна робота

(БДР, МКР)

Підрозділ кафедра обчислювальної техніки

(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 87,5% Схожість 12,5%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С. М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism щодо роботи.

Автор роботи _____ Звездін Р. С.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Савицька Л. А.
(підпис) (прізвище, ініціали)

ДОДАТОК В

ІЛЮСТРАТИВНА ЧАСТИНА

ІОТ ЗАСІБ З ЧАТ-БОТОМ ДЛЯ КЕРУВАННЯ ЕЛЕКТРИЧНИМИ ПРИСТРОЯМИ

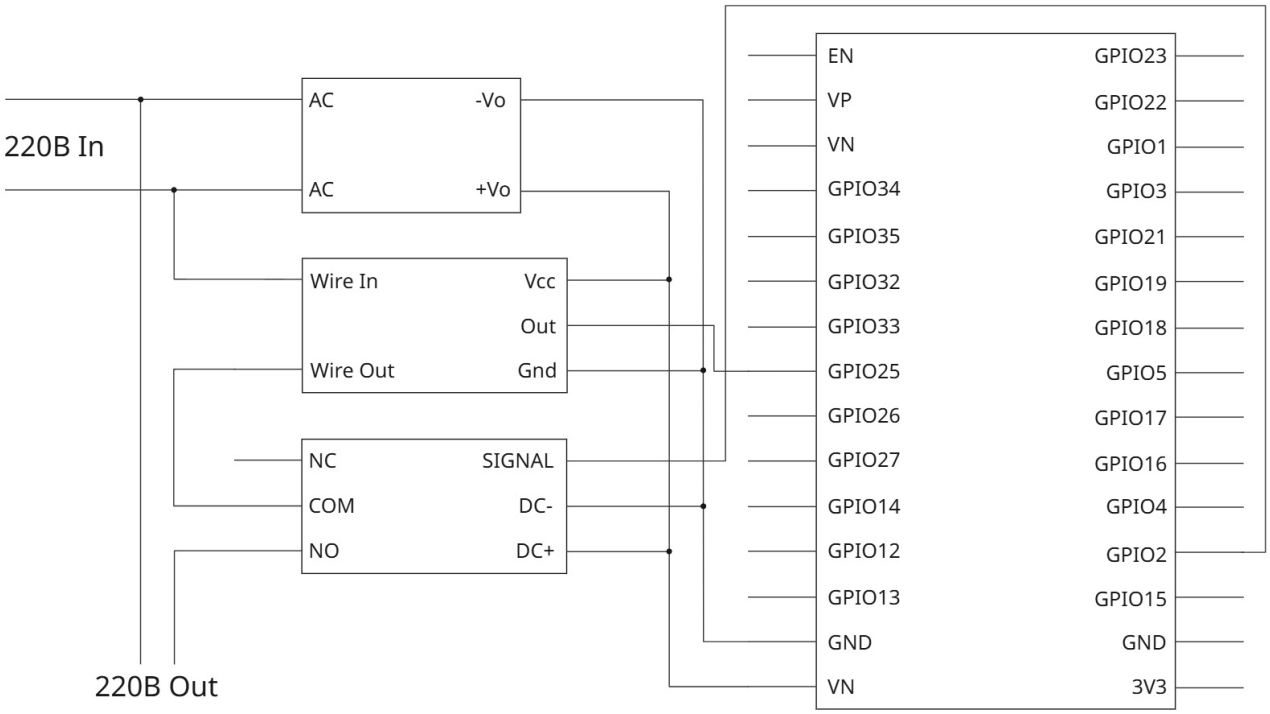

Рисунок В.1 — Принципова схема IoT засобу з чат-ботом для керування електричними пристроями

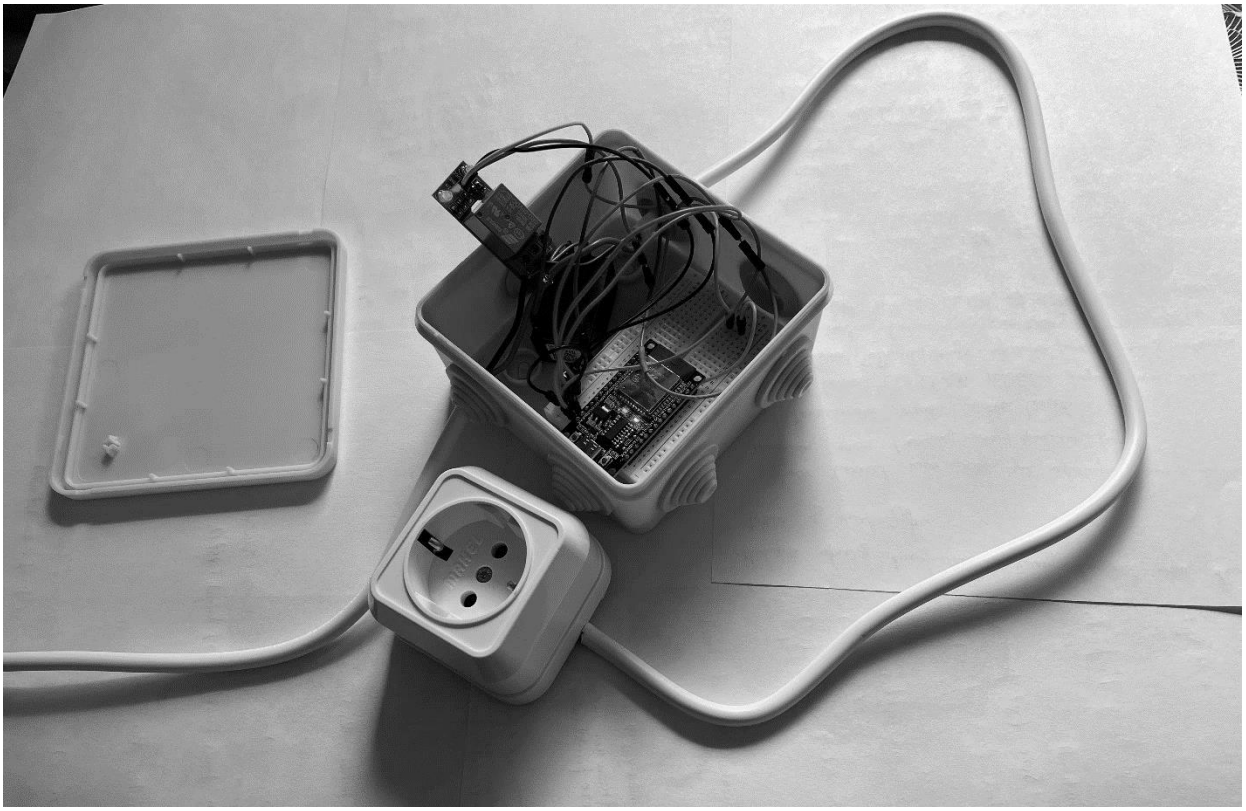

Рисунок В.2 — Макет IoT-пристрою з чат-ботом для керування електричними пристроями

ДОДАТОК Г

Лістинг програмного коду для мікропроцесора ESP32

```
#include <WiFi.h>
#include <WebServer.h>
#include <ArduinoJson.h>
#include <EEPROM.h>
#include <WiFiManager.h>
#include <DNSServer.h>
#include <Preferences.h>
#define RELAY_PIN 2
#define ACS712_PIN A0
#define LED_PIN 5
#define ACS712_SENSITIVITY 0.066
#define ZERO_POINT 2.5
#define VOLTAGE_REFERENCE 3.3
const char* wifi_networks[][2] = {
  {"TP-Link_BC9C", "09598614"},
  {"Invisible", "200420044"}
};
const int num_networks = sizeof(wifi_networks) / sizeof(wifi_networks[0]);
WebServer server(9080);
WiFiManager wifiManager;
Preferences preferences;
struct DeviceState {
  bool relay_state = false;
  float current = 0.0;
  float voltage = 220.0;
  float power = 0.0;
  float energy = 0.0
  float current_limit = 10.0;
  bool limit_exceeded = false;
  unsigned long uptime_seconds = 0;
  unsigned long relay_uptime_seconds = 0;
  bool wifi_connected = false;
  int wifi_rssi = 0;
  unsigned long last_energy_update = 0;
  unsigned long relay_start_time = 0;
  unsigned long device_start_time = 0;
} device_state;
bool simulate_overcurrent = false;
float simulated_current = 0.0;
void setup() {
  Serial.begin(115200);
  Serial.println("\n=== ESP32 Smart Socket Starting ===");
  pinMode(RELAY_PIN, OUTPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(ACS712_PIN, INPUT);
  digitalWrite(RELAY_PIN, HIGH);
  digitalWrite(LED_PIN, LOW);
  preferences.begin("smart_socket", false);
  loadSettings();
  device_state.device_start_time = millis();
  device_state.last_energy_update = millis();
```

```
  Serial.println("Hardware initialized");
  setupWiFi();
  setupWebServer
  Serial.println("=== Setup Complete ===");
  printStatus();
}
void loop() {
  server.handleClient();
  updateMeasurements();
  updateUptime();
  updateEnergyCounter();
  checkCurrentLimit();
  updateLED();
  delay(100);
}
void setupWiFi() {
  Serial.println("Setting up WiFi...");
  WiFi.mode(WIFI_STA);
  for (int i = 0; i < num_networks; i++) {
    Serial.printf("Trying to connect to: %s\n", wifi_networks[i][0]);
    WiFi.begin(wifi_networks[i][0], wifi_networks[i][1]);
      int attempts = 0;
      while (WiFi.status() != WL_CONNECTED && attempts < 20) {
      delay(500);
      Serial.print(".");
      attempts++;
      }
      if (WiFi.status() == WL_CONNECTED) {
      Serial.printf("\nConnected to %s\n", wifi_networks[i][0]);
      Serial.printf("IP Address: %s\n", WiFi.localIP().toString().c_str());
      Serial.printf("Signal Strength: %d dBm\n", WiFi.RSSI());
      device_state.wifi_connected = true;
      device_state.wifi_rssi = WiFi.RSSI();
      return;
      }
    Serial.println(" Failed");
  }
  Serial.println("Starting WiFi configuration portal...");
  wifiManager.setAPCallback(configModeCallback);
  wifiManager.setSaveConfigCallback(saveConfigCallback);
  if (!wifiManager.autoConnect("SmartSocket_Config", "12345678")) {
    Serial.println("Failed to connect and hit timeout");
    ESP.restart();
  }
  Serial.printf("WiFi connected via manager\n");
  Serial.printf("IP Address: %s\n", WiFi.localIP().toString().c_str());
  device_state.wifi_connected = true;
  device_state.wifi_rssi = WiFi.RSSI();
}
void configModeCallback(WiFiManager *myWiFiManager) {
  Serial.println("Entered config mode");
  Serial.println("AP: SmartSocket_Config");
  Serial.println("Password: 12345678");
  Serial.println("IP: 192.168.4.1");
}
void saveConfigCallback() {
```

```
  Serial.println("Should save config");
}
void setupWebServer() {
  Serial.println("Setting up web server...");
  server.onNotFound([]() {
      if (server.method() == HTTP_OPTIONS) {
      server.sendHeader("Access-Control-Allow-Origin", "*");
      server.sendHeader("Access-Control-Allow-Methods", "GET,POST,OPTIONS");
      server.sendHeader("Access-Control-Allow-Headers", "Content-Type");
      server.send(200);
      } else {
      server.send(404, "text/plain", "Not Found");
      }
  });
  server.on("/api/status", HTTP_GET, handleStatus);
  server.on("/api/relay", HTTP_POST, handleRelay);
  server.on("/api/reset", HTTP_POST, handleReset);
  server.on("/api/limit", HTTP_POST, handleLimit);
  server.on("/api/simulate", HTTP_POST, handleSimulate);
  server.on("/", HTTP_GET, []() {
      String html = "<h1>ESP32 Smart Socket</h1>";
      html += "<p>Status: " + String(device_state.relay_state ? "ON" : "OFF") +
"</p>";
      html += "<p>Current: " + String(device_state.current, 2) + " A</p>";
      html += "<p>Power: " + String(device_state.power, 1) + " W</p>";
      html += "<p>IP: " + WiFi.localIP().toString() + "</p>";
    server.send(200, "text/html", html);
  });
  server.begin();
  Serial.printf("Web server started on http://%s:9080\n",
WiFi.localIP().toString().c_str());
}
void handleStatus() {
  setCORSHeaders();
  DynamicJsonDocument doc(512);
  doc["success"] = true;
  doc["relay_state"] = device_state.relay_state;
  doc["current"] = round(device_state.current * 100) / 100.0;
  doc["voltage"] = device_state.voltage;
  doc["power"] = round(device_state.power * 10) / 10.0;
  doc["energy"] = round(device_state.energy * 1000) / 1000.0;
  doc["current_limit"] = device_state.current_limit;
  doc["limit_exceeded"] = device_state.limit_exceeded;
  doc["uptime_seconds"] = device_state.uptime_seconds;
  doc["relay_uptime_seconds"] = device_state.relay_uptime_seconds;
  doc["wifi_connected"] = device_state.wifi_connected;
  doc["wifi_rssi"] = device_state.wifi_rssi;
  String response;
  serializeJson(doc, response);
  server.send(200, "application/json", response);
  Serial.println("Status requested");
}
void handleRelay() {
  setCORSHeaders();
  if (!server.hasArg("plain")) {
```

```
    server.send(400, "application/json", "{\"success\":false,\"error\":\"No
data\"}");
      return;
  }
  DynamicJsonDocument doc(128);
  deserializeJson(doc, server.arg("plain"));

  bool new_state = doc["state"];
  if (new_state && device_state.limit_exceeded) {
    server.send(400, "application/json", "{\"success\":false,\"error\":\"Current
limit exceeded\"}");
      return;
  }
  bool previous_state = device_state.relay_state;
  device_state.relay_state = new_state;
  digitalWrite(RELAY_PIN, new_state ? LOW : HIGH);
  if (new_state && !previous_state) {
    device_state.relay_start_time = millis();
    device_state.relay_uptime_seconds = 0;
  } else if (!new_state) {
    device_state.relay_uptime_seconds = 0;
  }
  Serial.printf("Relay %s (previous: %s)\n",
                new_state ? "ON" : "OFF",
                previous_state ? "ON" : "OFF");
  DynamicJsonDocument response(128);
  response["success"] = true;
  response["relay_state"] = device_state.relay_state;
  String responseStr;
  serializeJson(response, responseStr);
  server.send(200, "application/json", responseStr);
}
void handleReset() {
  setCORSHeaders();
  device_state.energy = 0.0;
  device_state.last_energy_update = millis();
  saveSettings();
  Serial.println("Energy counter reset");
  server.send(200, "application/json", "{\"success\":true}");
}
void handleLimit() {
  setCORSHeaders();
  if (!server.hasArg("plain")) {
    server.send(400, "application/json", "{\"success\":false,\"error\":\"No
data\"}");
      return;
  }
  DynamicJsonDocument doc(128);
  deserializeJson(doc, server.arg("plain"));
  float new_limit = doc["limit"];
  if (new_limit < 0.5 || new_limit > 50.0) {
    server.send(400, "application/json", "{\"success\":false,\"error\":\"Invalid
limit\"}");
      return;
  }
  device_state.current_limit = new_limit;
```

```
  device_state.limit_exceeded = false;
  saveSettings();
  Serial.printf("Current limit set to %.1f A\n", new_limit);
  DynamicJsonDocument response(128);
  response["success"] = true;
  response["current_limit"] = device_state.current_limit;
  String responseStr;
  serializeJson(response, responseStr);
  server.send(200, "application/json", responseStr);
}
void handleSimulate() {
  setCORSHeaders();
  if (!server.hasArg("plain")) {
    server.send(400, "application/json", "{\"success\":false,\"error\":\"No
data\"}");
      return;
  }
  DynamicJsonDocument doc(128);
  deserializeJson(doc, server.arg("plain"));
  simulate_overcurrent = doc["enable"];
  simulated_current = doc.containsKey("current") ? doc["current"].as<float>() :
device_state.current_limit + 1.0;

  Serial.printf("Overcurrent simulation %s (%.1f A)\n",
                simulate_overcurrent ? "enabled" : "disabled",
                simulated_current);

  server.send(200, "application/json", "{\"success\":true}");
}

void setCORSHeaders() {
  server.sendHeader("Access-Control-Allow-Origin", "*");
  server.sendHeader("Access-Control-Allow-Methods", "GET,POST,OPTIONS");
  server.sendHeader("Access-Control-Allow-Headers", "Content-Type");
}
void updateMeasurements() {
  device_state.wifi_connected = WiFi.status() == WL_CONNECTED;
  if (device_state.wifi_connected) {
    device_state.wifi_rssi = WiFi.RSSI();
  }
  if (simulate_overcurrent) {
    device_state.current = simulated_current;
  } else {
      int sensorValue = analogRead(ACS712_PIN);
      float voltage = (sensorValue / 4095.0) * VOLTAGE_REFERENCE;
      float actual_voltage = voltage * (5.0 / 3.3);
    device_state.current = abs((actual_voltage - 2.5) / ACS712_SENSITIVITY);
      if (device_state.current < 0.1) {
      device_state.current = 0.0;
      }
      if (!device_state.relay_state) {
      device_state.current = random(0, 3) / 100.0;
      }
  }
  device_state.power = device_state.voltage * device_state.current;
}
```

```
void updateUptime() {
  device_state.uptime_seconds = (millis() - device_state.device_start_time) /
1000;
  if (device_state.relay_state && device_state.relay_start_time > 0) {
    device_state.relay_uptime_seconds = (millis() - device_state.relay_start_time)
/ 1000;
  }
}
void updateEnergyCounter() {
  unsigned long current_time = millis();
  unsigned long time_diff = current_time - device_state.last_energy_update;
  if (time_diff >= 1000 && device_state.relay_state) {
      float hours = time_diff / 3600000.0;
    device_state.energy += (device_state.power / 1000.0) * hours;
    device_state.last_energy_update = current_time;
      static unsigned long last_save = 0;
      if (current_time - last_save > 10000) {
      saveSettings();
      last_save = current_time;
      }
  } else if (time_diff >= 1000) {
    device_state.last_energy_update = current_time;
  }
}
void checkCurrentLimit() {
  bool was_exceeded = device_state.limit_exceeded;
  device_state.limit_exceeded = device_state.current > device_state.current_limit;
  if (device_state.limit_exceeded && !was_exceeded) {
    Serial.printf("CURRENT LIMIT EXCEEDED! %.2f A > %.1f A\n",
                  device_state.current, device_state.current_limit);
      if (device_state.relay_state) {
      device_state.relay_state = false;
      digitalWrite(RELAY_PIN, LOW);
      Serial.println("Relay turned OFF due to overcurrent");
      }
  } else if (!device_state.limit_exceeded && was_exceeded) {
    Serial.println("Current back to normal");
  }
}
void updateLED() {
  static unsigned long last_blink = 0;
  static bool led_state = false;
  if (device_state.limit_exceeded) {
      if (millis() - last_blink > 200) {
      led_state = !led_state;
      digitalWrite(LED_PIN, led_state);
      last_blink = millis();
      }
  } else if (device_state.relay_state) {
    digitalWrite(LED_PIN, HIGH);
  } else {
      if (millis() - last_blink > 1000) {
      led_state = !led_state;
      digitalWrite(LED_PIN, led_state);
      last_blink = millis();
      }
```

```
  }
}
void loadSettings() {
  device_state.energy = preferences.getFloat("energy", 0.0);
  device_state.current_limit = preferences.getFloat("limit", 10.0);
  Serial.printf("Settings loaded: energy=%.3f kWh, limit=%.1f A\n",
                device_state.energy, device_state.current_limit);
}
void saveSettings() {
  preferences.putFloat("energy", device_state.energy);
  preferences.putFloat("limit", device_state.current_limit);
  Serial.println("Settings saved");
}
void printStatus() {
  Serial.println("\n=== DEVICE STATUS ===");
  Serial.printf("WiFi: %s (%d dBm)\n",
                device_state.wifi_connected ? "Connected" : "Disconnected",
                device_state.wifi_rssi);
  Serial.printf("IP Address: %s\n", WiFi.localIP().toString().c_str());
  Serial.printf("Relay: %s\n", device_state.relay_state ? "ON" : "OFF");
  Serial.printf("Current: %.2f A\n", device_state.current);
  Serial.printf("Power: %.1f W\n", device_state.power);
  Serial.printf("Energy: %.3f kWh\n", device_state.energy);
  Serial.printf("Limit: %.1f A\n", device_state.current_limit);
  Serial.printf("Uptime: %lu seconds\n", device_state.uptime_seconds);
  Serial.println("====================\n");
}
```

ДОДАТОК Д

Лістинг програмного коду для Telegram-боту

```
import logging
import requests
import asyncio
import json
from datetime import datetime, timedelta
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup,
BotCommand, KeyboardButton, \
    ReplyKeyboardMarkup, ReplyKeyboardRemove
from telegram.ext import ApplicationBuilder, CommandHandler, CallbackQueryHandler,
ContextTypes, MessageHandler, filters
BOT_TOKEN = "8029639585:AAFZkmTgnHi2hO9v0TPNcJmNTHIeezhdWxU"
ESP32_IP = "http://192.168.0.102:9080"
REQUEST_TIMEOUT = 10
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    level=logging.INFO
)
logger = logging.getLogger(__name__)
user_settings = {}
class SmartSocketBot:
def __init__(self):
        self.texts = {
            'ua': {
                'welcome': """<b>Розумна розетка - SmartSocket</b>
Вітаю! Я ваш помічник для керування розумною розеткою.
<b>Що я можу:</b>
• Увімкнути/вимкнути розетку
• Показати поточний статус та споживання
• Встановити ліміт струму для безпеки
• Відправити сповіщення при перевищенні ліміту
• Вести облік споживаної енергії
 <b>Швидкий старт:</b>
• Натисніть "Показати меню" для доступу до всіх функцій
• Або використовуйте команди: /status, /on, /off, /menu
<b>Безпека:</b> При перевищенні встановленого ліміту струму розетка автоматично
вимкнеться!""",
                  'status_title': "<b>Поточний статус розетки</b>",
                  'relay_on': "Розетка <b>УВІМКНЕНА</b>",
                  'relay_off': "Розетка <b>ВИМКНЕНА</b>",
                'current': "Струм: <b>{:.2f} A</b>",
                'voltage': "Напруга: <b>{:.1f} В</b>",
                'power': "Потужність: <b>{:.1f} Вт</b>",
                'energy': "Споживання: <b>{:.3f} кВт·год</b>",
                'uptime': "Час роботи: <b>{}</b>",
                'relay_uptime': "Увімкнено: <b>{}</b>",
                'limit': "Ліміт струму: <b>{:.1f} A</b>",
                'limit_exceeded': "<b>ПЕРЕВИЩЕННЯ ЛІМІТУ!</b>",
                'wifi_status': "WiFi: <b>{}</b> ({}dBm)",
                'turned_on': "Розетку <b>увімкнено</b>",
                'turned_off': " Розетку <b>вимкнено</b>",
                'energy_reset': "Лічильник енергії <b>скинуто</b>",
```

```
                'limit_set': " Встановлено ліміт: <b>{:.1f} A</b>",
                'connection_error': "Помилка з'єднання з розеткою",
                'invalid_limit': " Некоректний ліміт. Введіть число від 0.5 до
50",
              'enter_limit': "Введіть ліміт струму (0.5-50 A):\n\nПриклад:
10.5",
              'language_changed': "Мову змінено на українську",
              'menu_shown': "<b>Меню керування</b>\nВикористовуйте кнопки
нижче:",
                'menu_hidden': "Меню приховано\nВикористовуйте команди або
натисніть /menu",
                'help_text': """<b>Довідка по командах:</b>
<b>Основні команди:</b>
/start - Головне меню та привітання
/status - Поточний статус розетки
/on - Увімкнути розетку
/off - Вимкнути розетку
/menu - Показати/сховати меню кнопок
<b>Налаштування:</b>
/limit - Встановити ліміт струму
/reset - Скинути лічильник енергії
/lang - Змінити мову інтерфейсу
<b>Інше:</b>
/help - Показати цю довідку
<b>Швидкий доступ:</b>
Натисніть /menu щоб показати/сховати кнопки швидкого доступу!
 <b>Безпека:</b>
При перевищенні ліміту струму розетка автоматично вимкнеться та ви отримаєте
сповіщення.""",
                'menu_text': " <b>Меню команд</b>\n\nОберіть потрібну дію:",
                'limit_waiting': " Очікую введення ліміту струму...",
                'processing': " Обробляю запит...",
                'buttons': {
                  'turn_on': ' Увімкнути',
                  'turn_off': ' Вимкнути',
                    'status': ' Статус',
                  'settings': ' Налаштування',
                  'reset_energy': 'Скинути енергію',
                  'set_limit': 'Встановити ліміт',
                  'language': 'Мова',
               'help': 'Довідка',
                  'menu': 'Меню',
                  'show_menu': 'Показати меню',
                  'hide_menu': 'Сховати меню',
                  'back': 'Назад'
              }
          },
           'en': {
              'welcome': """ <b>Smart Socket Control</b>
Welcome! I'm your smart socket control assistant.
<b>What I can do:</b>
• Turn socket ON/OFF
• Show current status and consumption
• Set current limit for safety
• Send notifications when limit exceeded
• Track energy consumption
```

```
<b>Quick start:</b>
• Press "Show menu" to access all functions
• Or use commands: /status, /on, /off, /menu
<b>Safety:</b> When current limit is exceeded, socket will automatically turn
OFF!""",
                'status_title': " <b>Current Socket Status</b>",
                'relay_on': " Socket is <b>ON</b>",
                'relay_off': " Socket is <b>OFF</b>",
                'current': " Current: <b>{:.2f} A</b>",
                'voltage': " Voltage: <b>{:.1f} V</b>",
                'power': "Power: <b>{:.1f} W</b>",
                'energy': " Energy: <b>{:.3f} kWh</b>",
                'uptime': " Uptime: <b>{}</b>",
                'relay_uptime': " ON time: <b>{}</b>",
                'limit': " Current limit: <b>{:.1f} A</b>",
                'limit_exceeded': " <b>LIMIT EXCEEDED!</b>",
                'wifi_status': " WiFi: <b>{}</b> ({}dBm)",
                'turned_on': " Socket <b>turned ON</b>",
                'turned_off': " Socket <b>turned OFF</b>",
                'energy_reset': " Energy counter <b>reset</b>",
                'limit_set': " Limit set: <b>{:.1f} A</b>",
                'connection_error': " Connection error with socket",
                'invalid_limit': " Invalid limit. Enter number 0.5-50",
                'enter_limit': " Enter current limit (0.5-50 A):\n\nExample:
10.5",
                 'language_changed': " Language changed to English",
                'menu_shown': " <b>Control Menu</b>\nUse buttons below:",
                'menu_hidden': " Menu hidden\nUse commands or press /menu",
                'help_text': """<b>Commands Help:</b>
<b>Main Commands:</b>
/start - Main menu and welcome
/status - Current socket status
/on - Turn socket ON
/off - Turn socket OFF
/menu - Show/hide menu buttons
<b>Settings:</b>
/limit - Set current limit
/reset - Reset energy counter
/lang - Change interface language
<b>Other:</b>
/help - Show this help
<b>Quick Access:</b>
Press /menu to show/hide quick access buttons!
<b>Safety:</b>
When current limit is exceeded, socket will automatically turn OFF and you'll get
notification.""",
                'menu_text': " <b>Commands Menu</b>\n\nSelect action:",
                'limit_waiting': " Waiting for current limit input...",
              'processing': " Processing request...",
                'buttons': {
                    'turn_on': ' Turn ON',
                    'turn_off': ' Turn OFF',
                    'status': ' Status',
                    'settings': ' Settings',
                    'reset_energy': ' Reset Energy',
                    'set_limit': ' Set Current Limit',
```

```
                    'language': ' Language',
                    'help': ' Help',
                    'menu': ' Menu',
                     'show_menu': ' Show menu',
                    'hide_menu': ' Hide menu',
                    'back': ' Back'
                }
            }
        }
def get_user_lang(self, user_id):
        return user_settings.get(user_id, {}).get('lang', 'ua')
def get_text(self, user_id, key):
        lang = self.get_user_lang(user_id)
        return self.texts[lang][key]
def format_time(self, user_id, seconds):
        lang = self.get_user_lang(user_id)
        if lang == 'en':
            if seconds < 60:
                return f"{seconds}s"
            elif seconds < 3600:
                return f"{seconds // 60}min {seconds % 60}s"
            elif seconds < 86400:
                hours = seconds // 3600
                minutes = (seconds % 3600) // 60
                return f"{hours}h {minutes}min"
            else:
                days = seconds // 86400
                hours = (seconds % 86400) // 3600
                return f"{days}d {hours}h"
        else:
            if seconds < 60:
                return f"{seconds}с"
            elif seconds < 3600:
                return f"{seconds // 60}хв {seconds % 60}с"
            elif seconds < 86400:
                hours = seconds // 3600
                minutes = (seconds % 3600) // 60
                return f"{hours}г {minutes}хв"
            else:
                days = seconds // 86400
                hours = (seconds % 86400) // 3600
                return f"{days}д {hours}г"
def get_control_keyboard(self, user_id):
        lang = self.get_user_lang(user_id)
        buttons = self.texts[lang]['buttons']
        keyboard = [
            [
                KeyboardButton(buttons['turn_on']),
                KeyboardButton(buttons['turn_off'])
            ],
            [
                KeyboardButton(buttons['status']),
                KeyboardButton(buttons['menu'])
            ],
            [
                KeyboardButton(buttons['hide_menu'])
```

```
            ]
      ]
      return ReplyKeyboardMarkup(keyboard, resize_keyboard=True,
one_time_keyboard=False)
def get_main_menu_keyboard(self, user_id):
      """Inline клавіатура для головного меню"""
      lang = self.get_user_lang(user_id)
      buttons = self.texts[lang]['buttons']
      return InlineKeyboardMarkup([
            [
                InlineKeyboardButton(buttons['turn_on'], callback_data='turn_on'),
                InlineKeyboardButton(buttons['turn_off'],
callback_data='turn_off')
            ],
            [
                InlineKeyboardButton(buttons['status'], callback_data='status')
            ],
            [
              InlineKeyboardButton(buttons['show_menu'],
callback_data='show_menu')
            ]
      ])
def get_commands_menu_keyboard(self, user_id):
      lang = self.get_user_lang(user_id)
      buttons = self.texts[lang]['buttons']
      return InlineKeyboardMarkup([
            [
                InlineKeyboardButton(buttons['turn_on'], callback_data='turn_on'),
                InlineKeyboardButton(buttons['turn_off'],
callback_data='turn_off')
            ],
            [
                InlineKeyboardButton(buttons['status'], callback_data='status'),
                InlineKeyboardButton(buttons['settings'],
callback_data='settings')
            ],
            [
                InlineKeyboardButton(buttons['reset_energy'],
callback_data='reset_energy'),
                InlineKeyboardButton(buttons['set_limit'],
callback_data='set_limit')
            ],
            [
                InlineKeyboardButton(buttons['language'],
callback_data='language'),
                InlineKeyboardButton(buttons['help'], callback_data='help')
            ],
            [
                InlineKeyboardButton(buttons['show_menu'],
callback_data='show_menu')
            ]
      ])
def get_settings_keyboard(self, user_id):
      lang = self.get_user_lang(user_id)
      buttons = self.texts[lang]['buttons']
      return InlineKeyboardMarkup([
```

```
            [
                InlineKeyboardButton(buttons['reset_energy'],
callback_data='reset_energy'),
                InlineKeyboardButton(buttons['set_limit'],
callback_data='set_limit')
            ],
            [
                InlineKeyboardButton(buttons['language'],
callback_data='language')
            ],
            [
                InlineKeyboardButton(buttons['back'],
callback_data='back_to_menu')
            ]
      ])
async def make_request(self, endpoint, method='GET', data=None):
      try:
            url = f"{ESP32_IP}/api/{endpoint}"
            if method == 'GET':
                response = requests.get(url, timeout=REQUEST_TIMEOUT)
            elif method == 'POST':
                headers = {'Content-Type': 'application/json'}
                response = requests.post(url, json=data, headers=headers,
timeout=REQUEST_TIMEOUT)
            if response.status_code == 200:
                return response.json()
            else:
                logger.error(f"HTTP {response.status_code}: {response.text}")
               return None
      except requests.exceptions.RequestException as e:
            logger.error(f"Request error: {e}")
            return None
async def get_status(self)
      return await self.make_request('status')
async def control_relay(self, state):
      return await self.make_request('relay', 'POST', {'state': state})
async def reset_energy(self):
      return await self.make_request('reset', 'POST')
async def set_current_limit(self, limit):
      return await self.make_request('limit', 'POST', {'limit': limit})
def format_status_message(self, user_id, data):
      if not data or not data.get('success'):
            return self.get_text(user_id, 'connection_error')
      text = self.get_text(user_id, 'status_title') + "\n\n"
        relay_state = data.get('relay_state', False)
      if relay_state:
            text += self.get_text(user_id, 'relay_on') + "\n"
      else:
            text += self.get_text(user_id, 'relay_off') + "\n"
      text += "\n"
      text += self.get_text(user_id, 'current').format(data.get('current', 0)) +
"\n"
      text += self.get_text(user_id, 'voltage').format(data.get('voltage', 0)) +
"\n"
      text += self.get_text(user_id, 'power').format(data.get('power', 0)) + "\n"
```

```
      text += self.get_text(user_id, 'energy').format(data.get('energy', 0)) +
"\n\n"
      text += self.get_text(user_id, 'limit').format(data.get('current_limit', 0))
+ "\n"
      if data.get('limit_exceeded'):
          text += self.get_text(user_id, 'limit_exceeded') + "\n"
      text += "\n"
      uptime = self.format_time(user_id, data.get('uptime_seconds', 0))
      text += self.get_text(user_id, 'uptime').format(uptime) + "\n"
        if relay_state and data.get('relay_uptime_seconds', 0) > 0:
            relay_uptime = self.format_time(data.get('relay_uptime_seconds', 0))
            text += self.get_text(user_id, 'relay_uptime').format(relay_uptime) +
"\n"
        wifi_status = "Connected" if data.get('wifi_connected') else
"Disconnected"
      rssi = data.get('wifi_rssi', 0)
      text += self.get_text(user_id, 'wifi_status').format(wifi_status, rssi)
      return text
bot = SmartSocketBot()
async def start_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
user_id = update.effective_user.id
if user_id not in user_settings:
        user_settings[user_id] = {
            'lang': 'ua',
            'waiting_for_limit': False,
            'menu_shown': False
      }
welcome_text = bot.get_text(user_id, 'welcome')
keyboard = bot.get_main_menu_keyboard(user_id)
await update.message.reply_html(
        welcome_text,
        reply_markup=keyboard
)
async def menu_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
user_id = update.effective_user.id
    current_menu_state = user_settings.get(user_id, {}).get('menu_shown', False)
if current_menu_state:
        user_settings[user_id]['menu_shown'] = False
      message = bot.get_text(user_id, 'menu_hidden')
      await update.message.reply_html(message, reply_markup=ReplyKeyboardRemove())
else:
        user_settings[user_id]['menu_shown'] = True
      message = bot.get_text(user_id, 'menu_shown')
      keyboard = bot.get_control_keyboard(user_id)
      await update.message.reply_html(message, reply_markup=keyboard)
async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
user_id = update.effective_user.id
help_text = bot.get_text(user_id, 'help_text')
await update.message.reply_html(help_text)
async def on_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
user_id = update.effective_user.id
    processing_msg = await update.message.reply_html(bot.get_text(user_id,
'processing'))
result = await bot.control_relay(True)
if result and result.get('success'):
      message = bot.get_text(user_id, 'turned_on')
```

```
else:
      message = bot.get_text(user_id, 'connection_error')
await context.bot.edit_message_text(
        chat_id=user_id,
        message_id=processing_msg.message_id,
        text=message,
        parse_mode='HTML'
)
async def off_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
user_id = update.effective_user.id
    processing_msg = await update.message.reply_html(bot.get_text(user_id,
'processing'))
result = await bot.control_relay(False)
if result and result.get('success'):
      message = bot.get_text(user_id, 'turned_off')
else:
      message = bot.get_text(user_id, 'connection_error')
await context.bot.edit_message_text(
        chat_id=user_id,
        message_id=processing_msg.message_id,
        text=message,
        parse_mode='HTML'
)
async def status_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
user_id = update.effective_user.id
    processing_msg = await update.message.reply_html(bot.get_text(user_id,
'processing'))
status_data = await bot.get_status()
message = bot.format_status_message(user_id, status_data)
await context.bot.edit_message_text(
        chat_id=user_id,
        message_id=processing_msg.message_id,
        text=message,
        parse_mode='HTML'
)
async def limit_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
user_id = update.effective_user.id
if context.args:
      try:
            limit_value = float(context.args[0])
            if 0.5 <= limit_value <= 50:
                processing_msg = await
update.message.reply_html(bot.get_text(user_id, 'processing'))
                result = await bot.set_current_limit(limit_value)
                if result and result.get('success'):
                    message = bot.get_text(user_id,
'limit_set').format(limit_value)
                else:
                    message = bot.get_text(user_id, 'connection_error')
                await context.bot.edit_message_text(
                    chat_id=user_id,
                    message_id=processing_msg.message_id,
                    text=message,
                    parse_mode='HTML'
                )
            else:
```

```
                await update.message.reply_html(bot.get_text(user_id,
'invalid_limit'))
      except ValueError:
            await update.message.reply_html(bot.get_text(user_id,
'invalid_limit'))
else:
        user_settings[user_id]['waiting_for_limit'] = True
      await update.message.reply_html(bot.get_text(user_id, 'enter_limit'))
async def reset_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
user_id = update.effective_user.id
    processing_msg = await update.message.reply_html(bot.get_text(user_id,
'processing'))
result = await bot.reset_energy()
if result and result.get('success'):
      message = bot.get_text(user_id, 'energy_reset')
else:
      message = bot.get_text(user_id, 'connection_error')
await context.bot.edit_message_text(
        chat_id=user_id,
        message_id=processing_msg.message_id,
        text=message,
        parse_mode='HTML'
)
async def lang_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
user_id = update.effective_user.id
current_lang = bot.get_user_lang(user_id)
new_lang = 'en' if current_lang == 'ua' else 'ua'
    user_settings[user_id]['lang'] = new_lang
message = bot.get_text(user_id, 'language_changed')
await update.message.reply_html(message)
if user_settings[user_id].get('menu_shown', False):
      keyboard = bot.get_control_keyboard(user_id)
        menu_message = bot.get_text(user_id, 'menu_shown')
      await context.bot.send_message(
            chat_id=user_id,
            text=menu_message,
            reply_markup=keyboard,
            parse_mode='HTML'
      )
async def button_callback(update: Update, context: ContextTypes.DEFAULT_TYPE):
query = update.callback_query
await query.answer()
user_id = query.from_user.id
    callback_data = query.data
if callback_data == 'turn_on':
      result = await bot.control_relay(True)
      if result and result.get('success'):
            message = bot.get_text(user_id, 'turned_on')
      else:
            message = bot.get_text(user_id, 'connection_error')
      await context.bot.send_message(chat_id=user_id, text=message,
parse_mode='HTML')
elif callback_data == 'turn_off':
      result = await bot.control_relay(False)
      if result and result.get('success'):
            message = bot.get_text(user_id, 'turned_off')
```

```
      else:
            message = bot.get_text(user_id, 'connection_error')
      await context.bot.send_message(chat_id=user_id, text=message,
parse_mode='HTML')
elif callback_data == 'status':
        status_data = await bot.get_status()
      message = bot.format_status_message(user_id, status_data)
      await context.bot.send_message(chat_id=user_id, text=message,
parse_mode='HTML')
elif callback_data == 'show_menu':
       user_settings[user_id]['menu_shown'] = True
      message = bot.get_text(user_id, 'menu_shown')
      keyboard = bot.get_control_keyboard(user_id)
      await context.bot.send_message(
            chat_id=user_id,
            text=message,
            reply_markup=keyboard,
            parse_mode='HTML'
      )
elif callback_data == 'settings':
      message = "<b>Налаштування</b>" if bot.get_user_lang(user_id) == 'ua' else
"<b>Settings</b>"
      keyboard = bot.get_settings_keyboard(user_id)
      await query.edit_message_text(message, reply_markup=keyboard,
parse_mode='HTML')
elif callback_data == 'reset_energy':
      result = await bot.reset_energy()
      if result and result.get('success'):
            message = bot.get_text(user_id, 'energy_reset')
      else:
            message = bot.get_text(user_id, 'connection_error')
      await context.bot.send_message(chat_id=user_id, text=message,
parse_mode='HTML')
elif callback_data == 'set_limit':
        user_settings[user_id]['waiting_for_limit'] = True
      message = bot.get_text(user_id, 'enter_limit')
      await context.bot.send_message(chat_id=user_id, text=message,
parse_mode='HTML')
elif callback_data == 'language':
        current_lang = bot.get_user_lang(user_id)
      new_lang = 'en' if current_lang == 'ua' else 'ua'
        user_settings[user_id]['lang'] = new_lang
      message = bot.get_text(user_id, 'language_changed')
      await context.bot.send_message(chat_id=user_id, text=message,
parse_mode='HTML')
      if user_settings[user_id].get('menu_shown', False):
            keyboard = bot.get_control_keyboard(user_id)
            menu_message = bot.get_text(user_id, 'menu_shown')
            await context.bot.send_message(
                chat_id=user_id,
                text=menu_message,
                reply_markup=keyboard,
                parse_mode='HTML'
            )
elif callback_data == 'help':
      message = bot.get_text(user_id, 'help_text')
```

```
      await context.bot.send_message(chat_id=user_id, text=message,
parse_mode='HTML')
elif callback_data == 'back_to_menu':
      message = bot.get_text(user_id, 'menu_text')
      keyboard = bot.get_commands_menu_keyboard(user_id)
      await query.edit_message_text(message, reply_markup=keyboard,
parse_mode='HTML')
async def handle_text(update: Update, context: ContextTypes.DEFAULT_TYPE):
user_id = update.effective_user.id
text = update.message.text
lang = bot.get_user_lang(user_id)
buttons = bot.texts[lang]['buttons']
if user_settings.get(user_id, {}).get('waiting_for_limit', False):
      try:
            limit_value = float(text)
            if 0.5 <= limit_value <= 50:
                 user_settings[user_id]['waiting_for_limit'] = False
                processing_msg = await
update.message.reply_html(bot.get_text(user_id, 'processing'))
                result = await bot.set_current_limit(limit_value)
                if result and result.get('success'):
                    message = bot.get_text(user_id,
'limit_set').format(limit_value)
                else:
                    message = bot.get_text(user_id, 'connection_error')
                await context.bot.edit_message_text(
                    chat_id=user_id,
                    message_id=processing_msg.message_id,
                    text=message,
                    parse_mode='HTML'
                )
            else:
                await update.message.reply_html(bot.get_text(user_id,
'invalid_limit'))
      except ValueError:
            await update.message.reply_html(bot.get_text(user_id,
'invalid_limit'))
      return
if text == buttons['turn_on']:
        processing_msg = await update.message.reply_html(bot.get_text(user_id,
'processing'))
      result = await bot.control_relay(True)
      if result and result.get('success'):
            message = bot.get_text(user_id, 'turned_on')
      else:
            message = bot.get_text(user_id, 'connection_error')
      await context.bot.edit_message_text(
            chat_id=user_id,
            message_id=processing_msg.message_id,
            text=message,
             parse_mode='HTML'
      )
elif text == buttons['turn_off']:
        processing_msg = await update.message.reply_html(bot.get_text(user_id,
'processing'))
      result = await bot.control_relay(False)
```

```
      if result and result.get('success'):
            message = bot.get_text(user_id, 'turned_off')
      else:
            message = bot.get_text(user_id, 'connection_error')
      await context.bot.edit_message_text(
            chat_id=user_id,
            message_id=processing_msg.message_id,
            text=message,
            parse_mode='HTML'
      )
elif text == buttons['status']:
        processing_msg = await update.message.reply_html(bot.get_text(user_id,
'processing'))
        status_data = await bot.get_status()
      message = bot.format_status_message(user_id, status_data)
      await context.bot.edit_message_text(
             chat_id=user_id,
            message_id=processing_msg.message_id,
            text=message,
            parse_mode='HTML'
      )
elif text == buttons['menu']:
      message = bot.get_text(user_id, 'menu_text')
      keyboard = bot.get_commands_menu_keyboard(user_id)
      await update.message.reply_html(message, reply_markup=keyboard)
elif text == buttons['hide_menu']:
        user_settings[user_id]['menu_shown'] = False
      message = bot.get_text(user_id, 'menu_hidden')
         await update.message.reply_html(message,
reply_markup=ReplyKeyboardRemove())
async def error_handler(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обробка помилок"""
    logger.error(f"Update {update} caused error {context.error}")
async def set_bot_commands(application):
commands = [
        BotCommand("start", " Головне меню"),
        BotCommand("status", " Поточний статус"),
      BotCommand("on", "Увімкнути розетку"),
        BotCommand("off", " Вимкнути розетку"),
        BotCommand("menu", "Показати/сховати меню"),
        BotCommand("limit", "Встановити ліміт струму"),
      BotCommand("reset", "Скинути лічильник"),
        BotCommand("lang", "Змінити мову"),
      BotCommand("help", "Допомога")
]
await application.bot.set_my_commands(commands)
def main():
    logger.info("Запуск SmartSocket Bot...")
application = ApplicationBuilder().token(BOT_TOKEN).build()
    application.add_handler(CommandHandler("start", start_command))
    application.add_handler(CommandHandler("menu", menu_command))
    application.add_handler(CommandHandler("help", help_command))
    application.add_handler(CommandHandler("on", on_command))
    application.add_handler(CommandHandler("off", off_command))
    application.add_handler(CommandHandler("status", status_command))
    application.add_handler(CommandHandler("limit", limit_command))
```

```
    application.add_handler(CommandHandler("reset", reset_command))
    application.add_handler(CommandHandler("lang", lang_command))
    application.add_handler(CallbackQueryHandler(button_callback))
    application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND,
handle_text))

    application.add_error_handler(error_handler)
    asyncio.get_event_loop().run_until_complete(set_bot_commands(application))
    logger.info("SmartSocket Bot готовий до роботи!")
    logger.info(f"ESP32 адреса: {ESP32_IP}")
application.run_polling(allowed_updates=Update.ALL_TYPES)
if __name__ == '__main__':
main()
```