МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп’ютерної інженерії

ДУДАС Владислав Сергійович

Алгоритми керування пристроями «розумного» будинку з контролем доступу / Algoritms for controlling smart home devices with access control

спеціальність: 123 – Комп’ютерна інженерія
освітньо-професійна програма – Комп’ютерна інженерія

Випускна кваліфікаційна робота

Виконав: студент групи КІзм-21
Дудас Владислав Сергійович

Науковий керівник
к.т.н. Гураль І.В.

ТЕРНОПІЛЬ - 2024

АНОТАЦІЯ

Дудас В. С. Алгоритми керування пристроями «розумного» будинку з контролем доступу. – Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 123 «Комп'ютерна інженерія», освітньо-професійна програма. Західноукраїнський національний університет, Тернопіль, 2024.

Робота написана обсягом 112 сторінок і містить 30 ілюстрацій, 2 таблиці, 6 додатків та 53 джерела за переліком посилань. Метою кваліфікаційної роботи є розробка ядра для керування пристроями розумного дому.

Методи досліджень. У роботі проведено аналіз технологій управління розумним домом. За пропонована система дозволяє керувати та моніторити стан пристроїм розумного дому з подальшою можливістю автоматизувати взаємодію між ними масштабуючи систему. Не оминуло уваги питання безпеки, всі запити API, які обробляються на надсилаються проходять через алгоритм хеш шифрування HMAC SHA-256.

Результати дослідження: Розроблена система дозволяє користувачу за допомогою зручного інтерфейсу переглядати та взаємодіяти з пристроєм розумного дому. Це значно спрощує взаємодію з пристроями та дозволяє керувати ними будь звідки.

Орієнтовні напрямки розвитку досліджень: У результаті тестування системи було підтверджено її ефективність та функціональність. У процесі розробки було оптимізовано роботу API для кращої та безпечнішої інтеграції з пристроями розумного дому.

Ключові слова: КЕРУВАННЯ РОЗУМНИМ ДОМОМ, ПРИСТРОЇ РОЗУМНОГО ДОМУ, ІНТЕРНЕТ РЕЧЕЙ, API, TELEGRAM BOT

ANNOTANION

Dudas V. S. Algorithms for controlling devices of a “smart” house with access control. – Manuscript.

Qualification work for obtaining the degree of “Master” in specialty 123 “Computer Engineering”, educational and professional program. Western Ukrainian National University, Ternopil, 2024.

The work is written in 112 pages and contains 30 illustrations, 2 tables, 6 appendices and 53 sources according to the list of references. The purpose of the qualification work is to develop a kernel for controlling smart home devices.

Research methods. The work analyzes smart home control technologies. The proposed system allows you to control and monitor the state of smart home devices with the further possibility of automating the interaction between them by scaling the system. The issue of security was not overlooked, all API requests that are processed and sent go through the HMAC SHA-256 hash encryption algorithm.

Research results: The developed system allows the user to view and interact with a smart home device using a convenient interface. This greatly simplifies interaction with devices and allows you to control them from anywhere.

Approximate directions of research development: As a result of testing the system, its effectiveness and functionality were confirmed. During the development process, the API operation was optimized for better and safer integration with smart home devices.

Keywords: SMART HOME CONTROL, SMART HOME DEVICES, INTERNET OF THINGS, API, TELEGRAM BOT

ЗМІСТ

ВСТУП

Автоматизація та комфорт. Це ті дві речі, які, як на мене, людство так успішно прагне досягти. Нескінченні дослідження та приголомшливі розробки у різноманітних сферах з кожним днем спрощують життя як для великих компаній, так і для звичайних кінцевих користувачів. Фабрики та заводи з мільйонами комп’ютерних компонентів збирають на своїх лініях незамінні прилади для того, щоб ті опинились саме у вашому домі.

Автоматична кавоварка по заданому сценарію з самого ранку розбудить вас ароматом запашного напою, мультиварка зготує обід чи вечерю, робот-пилосмок зможе сам прибрати дім, а датчики різного типу та форми у кожному кутку вашого дому будуть пильно стежити за комфортом та увімкнуть обігрівач чи кондиціонер за потреби. Хтось називає це інтернетом речей, а хтось паноптикумом, де кожен аспект нашого життя під контролем технологій, що спостерігають за нами. І це не просто теоретичні побоювання, це реальні виклики, з якими стикаються користувачі. Так чи інакше, проблема безпеки стоїть гостро, і це саме те, що не потрібно відкладати у довгий ящик. До того ж цю проблему треба вирішити так, щоб не порушувати ідилію зручності та комфорту, без якого деякі люди будуть як без рук. Користувачі повинні мати змогу насолоджуватися тією самою зручністю без страху перед кібератаками або витоками даних. Це означає, що рішення, які будуть розроблені для забезпечення безпеки, повинні бути інтуїтивно зрозумілими, легко впроваджуваними та ненав'язливими, щоб не заважати ідилії зручності.

У сучасному світі технології розумного дому стають дедалі більш популярними, перетворюючи звичайні житлові простори на інтерактивні та автоматизовані середовища. Проте, зростаюча кількість підключених пристроїв створює нові вектори атак, що ставлять під загрозу особисті дані користувачів.

У зв'язку з цим, виникає необхідність у розробці універсального ядра для управління пристроями розумного дому з інтегрованим контролем доступу. Це

рішення має на меті не тільки полегшити управління розумними пристроями, але й забезпечити необхідний рівень безпеки, щоб користувачі могли насолоджуватися всіма перевагами технологій без побоювань.

Практична цінність полягає у розробці програмного засобу, який зміг би надати користувачам один інструмент для перегляду та налаштування своїх пристроїв розумного дому.

Актуальність даної роботи зумовлена зростаючою необхідністю інтеграції безпечних і зручних рішень для управління пристроями розумного дому. З огляду на збільшення кількості підключених пристроїв, важливо розробити універсальні системи, які забезпечують контроль доступу до їх функцій. Це дозволить користувачам безпечно використовувати розумні технології, не побоюючись за свої дані.

Об'єктом дослідження є технології автоматизації в системах розумного дому, предметом – алгоритми для управління такими системами, які забезпечують інтегрований контроль доступу.

Метою даної кваліфікаційної роботи є розробка універсального ядра для керування пристроями розумного дому з контролем доступу. Це рішення має на меті забезпечити зручне і безпечне управління різноманітними пристроями, інтегруючи функції автоматизації та захисту даних.

Отже, підсумувавши вищесказане, завдання які потрібно виконати для досягнення мети розробки, включають:

- провести аналіз потреб функціоналу системи;
- проаналізувати наявні технології підключення пристроїв розумного дому;
- проаналізувати можливі архітектури для програмної системи;
- спроєктувати архітектуру системи на основі аналізу;
- розробити програмне рішення, яке буде відповідати поставленій задачі;
- провести тестування та валідацію готового програмного продукту;

Отже, зважаючи на викладене, ця робота спрямована на розробку універсального ядра для управління пристроями розумного дому, яке не лише спростить взаємодію користувачів з технологіями, але й забезпечить їхню безпеку. В умовах зростаючих загроз кібербезпеки важливо створити систему, яка об'єднує зручність та надійність, дозволяючи користувачам без побоювань насолоджуватися перевагами сучасних технологій. Завдяки чітко визначеним завданням, таким як аналіз потреб, вибір технологій підключення, проектування та тестування, ця робота стане важливим кроком до інтеграції безпечних рішень у повсякденне життя.

Новизна одержаних результатів визначається наступним чином:

1) створено інноваційне рішення, яке поєднує в собі управлінські та безпекові функції, підвищуючи рівень захисту особистих даних;

2) оптимізовано роботу API для кращої та безпечнішої інтеграції з пристроями розумного дому.

Публікації та апробація КР. Основні результати досліджень опубліковано в тезах доповідей «Кібербезпека для пристроїв розумного дому» обсягом 3 сторінки на науково практичній конференції для молодих вчених та студентів «Інтелектуальні комп’ютерні системи та мережі» (ІКСМ-2024) [1] та «Програмна система оповіщення студентів кафедри комп’ютерної інженерії на основі технології Push API» обсягом 1 сторінка на науково практичній конференції для молодих вчених та студентів «Інтелектуальні комп’ютерні системи та мережі» (ІКСМ-2023) [2].

Зазначена кваліфікаційна робота містить три розділи [3, 4].

У першому розділі було проаналізовано алгоритми керування розумним домом від відомих компаній, технології розробки серверних додатків та їх аналіз.

У другому розділі було досліджено та вивчено основні функції та алгоритми роботи ботів, алгоритми роботи API та їх архітектури і досліджено механізми перевірки цілісності у вигляді хеш-функцій.

У третьому розділі розроблено серверний додаток, оптимізовано під потреби API та адаптовано під керування пристроєм розумного дому.

1 АНАЛІЗ СИСТЕМ ДЛЯ КЕРУАНЬ РОЗУМНИМ ДОМОМ

1.1 Основні алгоритми керування пристроями розумного дому

Аналіз здійснюється на основі вимог, що пред'являються до створюваної системи та з урахуванням особливостей предметної області. Результати повинні лягти в основу обґрунтування вибору платформи, технологій та середовища розробки програмного забезпечення. Для обґрунтування вибору застосовують методи прийняття рішень.

Існує безліч рішень та аналогів та додатків які дозволяють керувати пристроями розумного дому та дозволяють налаштовувати зв’язок між ними та їх налаштуваннями.

Amazon Alexa [5] є однією з найпопулярніших систем для керування пристроями розумного дому, яка популярна переважно на території сполучених штатів чи країн де працюють сервіси Amazon. Система підтримує широкий спектр обладнання, включаючи освітлення, системи безпеки та розважальні пристрої. Система забезпечує зручний голосовий інтерфейс, інтеграцію з продуктами Amazon і додаткові функції через «вміння» (skills). Проте основним її недоліком є залежність від хмарних сервісів, через що система не працює без Інтернету, а також існують питання конфіденційності, оскільки голосові команди зберігаються на серверах компанії [6].

Google Home [7] також вирізняється високим рівнем інтеграції з широким спектром пристроїв і сервісів, що дозволяє користувачам ефективно керувати розумним домом за допомогою голосових команд. Завдяки алгоритмам Google система пропонує персоналізований досвід і може аналізувати звички користувачів. Однак Google Home також має недоліки, схожі з Amazon Alexa, вона залежить від стабільного Інтернет-з’єднання, а питання збереження конфіденційних даних користувачів залишаються актуальними.

Apple HomeKit [8] надає високий рівень безпеки та захисту даних завдяки використанню локального шифрування і зосередженості на конфіденційності.

HomeKit інтегрується з продуктами Apple, такими як iPhone і iPad, та дозволяє налаштувати автоматизацію на базі голосового асистента Siri. Недоліком цієї системи є обмежена сумісність із пристроями інших виробників, що звужує її функціональні можливості для користувачів, які обирають різні бренди.

Samsung SmartThings [9] є платформою, що пропонує високу сумісність з різними пристроями, включаючи розумне освітлення, замки, сенсори та камери, надаючи користувачам розширені можливості автоматизації. SmartThings також дозволяє налаштувати систему за власними потребами, що робить її гнучкою та універсальною. Водночас користувачі можуть зіткнутися з певною складністю налаштувань, а для повноцінного використання часто потрібні додаткові компоненти, такі як центральний контролер SmartThings Hub.

Tuya Smart [10] є універсальною платформою для управління розумними пристроями, яка підтримує широкий спектр пристроїв і пропонує гнучкі можливості для створення кастомізованих рішень. Завдяки відкритому протоколу, Tuya дозволяє інтегрувати різноманітні розумні пристрої від різних виробників, що дає можливість користувачам створювати масштабовані системи автоматизації, зручні для повсякденного використання. Платформа також підтримує голосові асистенти, такі як Amazon Alexa і Google Assistant, що розширює можливості керування пристроями через голосові команди.

Сильним боком Tuya Smart є її доступність для розробників і виробників, що дозволяє створювати персоналізовані продукти під конкретні потреби. Крім того, система пропонує легкий у використанні мобільний застосунок для управління пристроями та автоматизації, а також розширену аналітику й можливості віддаленого доступу. Проте, основними недоліками є залежність від Інтернет-з’єднання, а також потенційні ризики щодо конфіденційності, оскільки обробка даних часто відбувається на віддалених серверах [11].

Таким чином, Tuya Smart є універсальною та масштабованою платформою, яка надає користувачам значні можливості для налаштування розумного дому, але водночас, як і інші платформи, вона має певні ризики, пов’язані з конфіденційністю даних і стабільністю роботи при відсутності Інтернету [12].

Впродовж дослідження та аналізу наявних технологій не було виявлено аналогів чи альтернатив серверних додатків(Telegram Ботів) які отримують дані з пристрою розумного дому чи можуть ним керувати.

Переваги такої новизни полягають у тому, що кінцевому користувачу не потрібно завантажувати окремий додаток, щоб взаємодіяти з пристроєм розумного дому. Такий підхід буде для кінцевого користувача вагомою перевагою у виборі платформи для керування та взаємодії з IoT [13,14].

1.2 Технології розробки серверних додатків

Серверний додаток – це програмне забезпечення, що працює на сервері і виконує різноманітні обчислення, обробляє запити користувачів, зберігає та обробляє дані. Серверний додаток є основною ланкою між користувачем і сервером: він отримує запити від клієнтських пристроїв (зазвичай через інтернет-протоколи, такі як HTTP або WebSocket), обробляє їх і повертає відповідь[15].

Розробка серверних додатків охоплює широкий спектр технологій, які забезпечують гнучкість, продуктивність і безпеку серверної частини додатка. У виборі інструментів враховуються потреби конкретного проєкту, вимоги до продуктивності, зручність масштабування, а також сумісність з іншими елементами IT-інфраструктури. Серед ключових технологій, які використовують для розробки серверних додатків виділяють:

- мови програмування (Python, PHP, JavaScript в парі з Node.js, Java, Go чи Ruby);
- серверні фреймворки (Django, Express.js для Node.js, Spring Boot, Laravel, FastAPI);
- бази даних, які в свою чергу поділяються на реляційні (PostageSQL, MySQL та Oracle Database) і нереляційні (MongoDB, Cassandra та Redis);

- хмарні сервіси та платформи (Amazon Web Services, Google Cloud Platform та Microsoft Azure);
- контейнеризація та оркестрація (Docker, Kubernetes);
- API та інтеграційні технології (REST API, GraphQL чи gRPC).

Кожна з перерахованих вище технологій має свої переваги та недоліки які створюють серйозний вплив на подальшу стабільність, відмовостійкість та масштабованість проєкту, тому до вибору так званого, стеку технологій, треба поставитись зі всією відповідальністю. Вибір стеку технологій залежить від потреб масштабованості, безпеки, простоти обслуговування і доступних ресурсів, що дає змогу адаптувати серверну частину додатка до конкретного контексту використання[16]. Наприклад, Java і Spring Boot гарантують стабільність і продуктивність для великих проєктів, але можуть бути складними для освоєння. Node.js підходить для реального часу, але обмежений в інтенсивних обчислювальних задачах. Контейнеризація через Docker дозволяє незалежність від інфраструктури, але може вимагати спеціальної компетенції для оркестрації через Kubernetes. Хмарні сервіси дають гнучкість в управлінні ресурсами, однак можуть мати обмеження у відповідності до регіональних і правових стандартів.

Серверні додатки мають низку переваг, що робить їх ефективним рішенням у багатьох сферах: електронній комерції, соціальних мережах, онлайн-банкінгу, освіті, охороні здоров'я, подорожах, розвагах тощо[17]. Вони забезпечують доступність на будь-якому пристрої з Інтернетом і працюють у браузері, що знімає потребу в локальному встановленні. Завдяки централізованій обробці даних і підтримці, серверні додатки легко масштабувати, швидко оновлювати, що підвищує надійність і зручність для кінцевих користувачів. Розглянемо їх схематичне порівняння детально наведено у таблиці 1.1.

Таблиця 1.1 - Порівняння мов програмування для розробки веб-додатків

Мова програмування	Швидкодія	Простота в освоєнні	Універсальність	Популярність
Java	Висока	Середня	Висока	Дуже популярна
Python	Середня	Висока	Висока	Дуже популярна
JavaScript	Середня	Висока	Висока	Дуже популярна
PHP	Середня	Висока	Висока	Дуже популярна
Ruby	Середня	Середня	Середня	Популярна
Go	Висока	Середня	Висока	Популярна

Ця таблиця надає загальний порівняльний огляд кількох мов програмування за вказаними критеріями. Зверніть увагу, що оцінки можуть бути умовними, оскільки кожна мова має свої особливості та відповідає різним вимогам проектів. Рішення щодо вибору мови програмування повинно ґрунтуватися на конкретних потребах, вміннях команди та інших факторах, що впливають на поточний проєкт.

Вибір мови програмування для розробки проєкту залежить від багатьох факторів, таких як особисті вподобання, потреби проекту, наявність ресурсів та специфікації проекту. Кожна згадана мова та технологія має свої переваги та недоліки, і кращим варіантом буде той, який найкраще відповідає потребам конкретного проекту - Java.

Java - це потужна та популярна об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (зараз є власністю компанії Oracle [18]). Вона була випущена в 1995 році і з тих пір стала однією з найпоширеніших мов програмування у світі.

Основні особливості мови Java наведені нижче.

1. Об'єктно-орієнтованість: Java побудована на концепції об'єктно-орієнтованого програмування, що дозволяє створювати модульні, універсальні та підтримувані програми.

2. Платформонезалежність: Java працює на основі віртуальної машини Java (Java Virtual Machine, JVM), що дозволяє виконувати Java-програми на різних операційних системах без необхідності перекомпіляції. Це робить Java платформонезалежною.

3. Безпека: Java має вбудовану систему безпеки, що дозволяє контролювати доступ до ресурсів, управляти пам'яттю та запобігати вразливостям.

4. Велика бібліотека: Java поставляється з великою стандартною бібліотекою, яка містить готові класи та методи для різних завдань, таких як робота з рядками, мережеве програмування, обробка введення-виведення, робота

з базами даних та багато іншого. Це спрощує розробку програм і зменшує кількість написаного коду.

5. Багатопоточність: Java надає потужні засоби для роботи з багатопоточними програмами, дозволяючи одночасно виконувати різні частини програми.

6. Розширюваність: Java підтримує розширення функціональності через використання плагінів та бібліотек сторонніх розробників.

Java знайшла застосування в багатьох сферах програмування, таких як розробка веб-додатків, мобільних додатків, вбудованих систем, наукових досліджень, фінансових програм, ігор та багато іншого. Велика спільнота розробників та багато доступних ресурсів роблять Java популярним вибором для багатьох проектів. Зараз Java має велику активну спільноту розробників, що дозволяє отримувати широку підтримку, документацію та розвиток мови та її інструментів, фреймворків та плагінів [19].

Java також має широкий спектр інструментів для розробки додатків, таких як Eclipse, NetBeans та IntelliJ IDEA.

IntelliJ IDEA , для прикладу, є інтегрованим середовищем розробки (Integrated Development Environment або IDE) для програмування, розробленою компанією JetBrains. Вона призначена для підтримки різних мов програмування, зокрема Java, Kotlin, Scala, Groovy та багатьох інших.

IntelliJ IDEA надає розширений набір інструментів для розробки програмного забезпечення, включаючи редактор коду з підсвічуванням синтаксису, автодоповненням, рефакторингом, аналізом коду, налалагодженням і багатьма іншими корисними функціями.

Це популярне середовище розробки серед професіоналів програмування, яке надає зручний та продуктивний спосіб розробки програмного забезпечення. IntelliJ IDEA також підтримує розширення плагінами, що дозволяє розширити його функціональність для задоволення конкретних потреб розробника[20].

Крім того, IntelliJ IDEA пропонує інтеграцію з різними засобами контролю версій, системами збирання проектів та іншими інструментами розробки, що сприяє покращенню ефективності роботи розробника.

1.3 Аналіз серверних додатків

Побудова серверного додатку - це комплекс заходів, який спрямований на те, щоб чітко визначити, як буде побудована система, а для кращого розуміння принципів побудови серверних додатків потрібно знати та розуміти як вони з’явились, та як розвивались.

Історія серверних додатків почалася з еволюції комп'ютерних мереж, коли стало можливим обробляти дані централізовано на потужних серверах і передавати їх на термінали, підключені до мережі. Перші серверні програми з'явилися ще в 1960-х роках і використовувалися на мейнфреймах — великих комп'ютерах, які обслуговували декілька терміналів. Ці програми дозволяли зберігати та обробляти інформацію централізовано, але мали значні обмеження щодо функціональності та інтерактивності.

У 1980-х роках з розвитком персональних комп’ютерів почала поширюватися архітектура «клієнт-сервер». Вона дала змогу створювати більш динамічні додатки, де обробка частково виконувалась на стороні клієнта, що покращувало користувацький досвід. Ця модель стала базою для багатьох бізнес-застосунків, зокрема в банківській та фінансовій галузях.

Справжній прорив стався в 1990-х роках із появою Інтернету. Веб-додатки почали розвиватися як серверні програми, до яких можна було отримати доступ через браузер. Це призвело до стрімкого зростання їх популярності, адже користувачі змогли взаємодіяти з інформацією, що зберігалась на сервері, в режимі реального часу з будь-якого комп'ютера з Інтернетом. У цей час стали з’являтися перші мови і фреймворки для веб-розробки, такі як CGI, а згодом і

PHP, ASP та Java Servlet, які дозволяли створювати більш інтерактивні й складні веб-додатки.

2000-ті роки стали ерою «веб 2.0», де основну роль почали відігравати інтерактивні та соціальні додатки. Поява технологій AJAX та JavaScript-фреймворків, як-от jQuery, зробила веб-додатки значно швидшими та приємнішими у використанні. З цього періоду серверні додатки почали включати інтерактивні елементи в реальному часі, що стало стандартом у соціальних мережах і платформах обміну контентом.

У 2010-х роках зростання мобільного інтернету призвело до необхідності створення масштабованих серверних додатків, здатних обробляти одночасно мільйони користувачів. З'явилися такі фреймворки, як Node.js, які дозволили створювати швидкодійні серверні додатки на основі JavaScript. Технології контейнеризації, як-от Docker, та хмарні платформи, такі як AWS, Azure і Google Cloud, забезпечили високу доступність і легке масштабування серверних додатків.

Сьогодні серверні додатки використовують різноманітні технології та архітектурні моделі, включаючи мікросервіси й хмарні обчислення, що дозволяє їм бути гнучкими, масштабованими та надійними.

Важливим складовим у виборі серверного додатку є його актуальність і охоплення аудиторії. За даними DataReportal [21] в останньому рейтингу мобільних додатків за кількістю активних користувачів за місяць Telegram посідає сьоме місце після WeChat, випереджаючи Snapchat та X (Twitter) маючи 950 мільйонів активних користувачів на місяць у світі. Telegram залишається однією з найпопулярніших соцмереж серед українців. Опитування Київського міжнародного інституту соціології у вересні 2023 року показало, що 44% українців використовують Telegram для отримання інформації та новин [22]. Месенджер став популярним завдяки своїй простоті, гнучкості кастомізації та технологічності. Останній пункт відноситься до так званих Telegram ботів.

Боти - це невеликі програми , які повністю працюють у програмі Telegram. Користувачі взаємодіють із ботами через гнучкі інтерфейси , які можуть

підтримувати будь-які завдання чи послуги. Платформа Telegram Bot містить понад 10 мільйонів ботів і є безкоштовною як для користувачів, так і для розробників [23].

Зараз, станом на 2024 рік кожен другий бізнес та приватна чи державна установа має свого власного телеграм бота для різноманітних цілей, від збору відгуків, отримання заявок чи простого зворотного зв’язку до повноцінних навчальних платформ, де в ігровій формі користувач зможе отримати інформацію як у вигляді тексту, так і будь які типи медіа файлів. Але функціонал на цьому не закінчується, оскільки бота можна запрограмувати будь як, ентузіасти створюють корисні інструменти завдяки цій потужній платформі, як от бот для зміни формату файлу, бот для переадресації листів з вашої електронної скриньки прямо у ваш улюблений месенджер, чи навіть повноцінні ігри. Що ж ще можна сказати, якщо навіть у ДІЯ є свій телеграм бот для служби підтримки користувачів [24].

Підсумувавши вищесказане та проаналізувавши статистичні дані, в якості серверного додатку було обрано платформу Telegram Bot, яка є інтерфейсом на основі HTTP.

HTTP (HyperText Transfer Protocol) - це протокол для отримання ресурсів, наприклад документів HTML. Він є основою будь-якого обміну даними в Інтернеті та є протоколом клієнт-сервер, що означає, що запити ініціюються одержувачем, зазвичай веб-браузером. Повний документ зазвичай складається з таких ресурсів, як текстовий вміст, інструкції щодо макета, зображення, відео, сценарії тощо (рисунок 1.1).

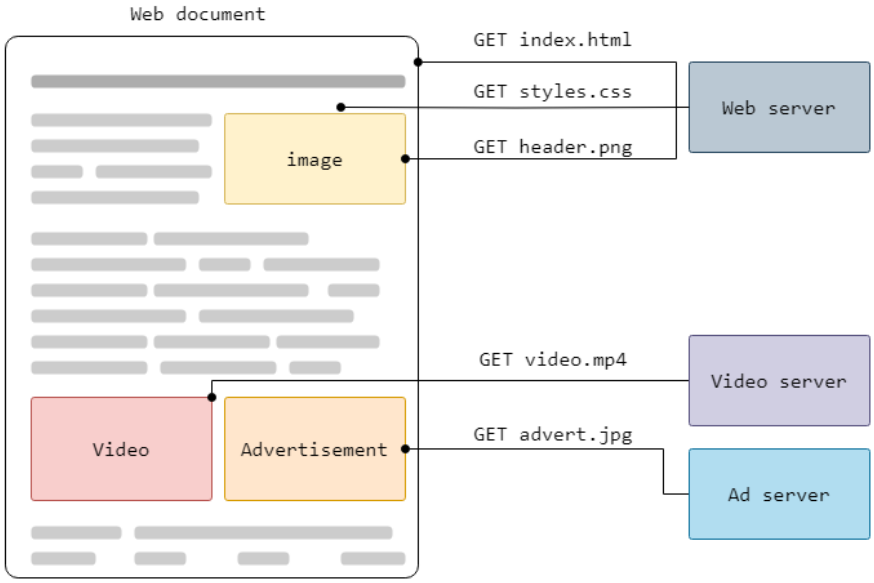

Рисунок 1.1 – Блок-схема принципу роботи HTTP протоколу

Клієнти та сервери спілкуються шляхом обміну окремими повідомленнями (на відміну від потоку даних). Повідомлення, надіслані клієнтом, називаються запитами, а повідомлення, надіслані сервером як відповідь, називаються відповідями.

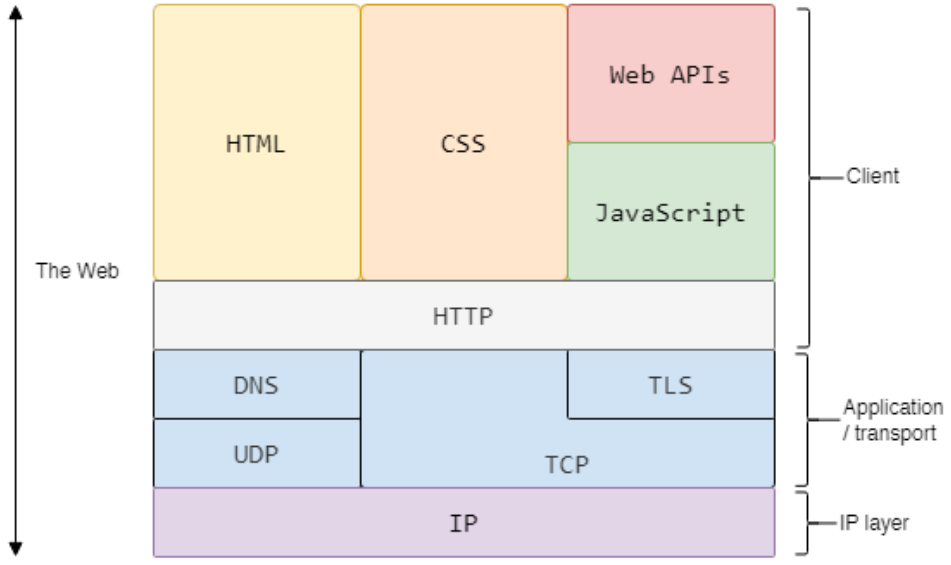

Рисунок 1.2 – Пошаровий огляд протоколів мережі

Розроблений на початку 1990-х років HTTP є розширюваним протоколом, який з часом розвивався. Це протокол прикладного рівня, який надсилається

через TCP або через TCP-з’єднання, зашифроване TLS , хоча теоретично можна використовувати будь-який надійний транспортний протокол. Завдяки своїй розширюваності він використовується не лише для отримання гіпертекстових документів, але й для зображень і відео або для публікації вмісту на серверах, як із результатами HTML-форм (рисунок 1.2). HTTP також можна використовувати для отримання частин документів для оновлення веб-сторінок на вимогу.

HTTP – це клієнт-серверний протокол: запити надсилаються однією сутністю, агентом користувача (або проксі-сервером від його імені). У більшості випадків агентом користувача є веб-браузер, але це може бути що завгодно, наприклад, робот, який сканує Інтернет, щоб заповнити та підтримувати індекс пошукової системи.

Кожен окремий запит надсилається на сервер, який обробляє його та надає відповідь, яка називається відповіддю. Між клієнтом і сервером є численні об’єкти, які разом називаються проксі-серверами , які виконують різні операції та діють , наприклад, як шлюзи або кеші (рисунок 1.3).

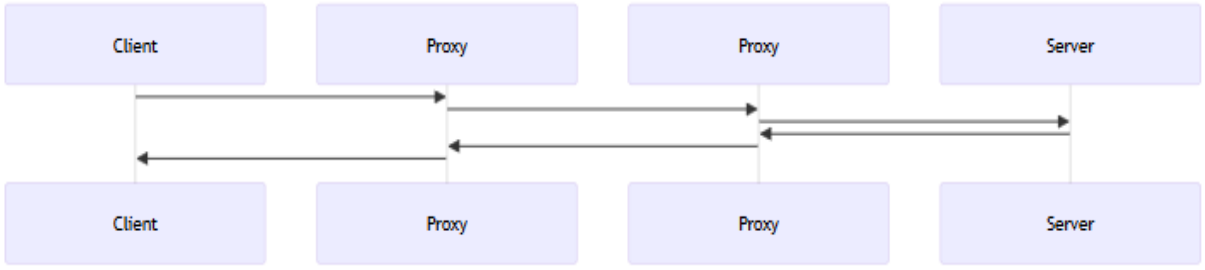

Рисунок 1.3 – Візуалізація HTTP запиту з використанням proxy

Агент користувача – це будь-який інструмент, який діє від імені користувача. Цю роль в основному виконує веб-браузер, але її також можуть виконувати програми, які використовуються інженерами та веб-розробниками для налагодження своїх програм.

Браузер завжди ініціює запит. Це ніколи не сервер (хоча протягом багатьох років було додано деякі механізми для імітації ініційованих сервером повідомлень).

Щоб відобразити веб-сторінку, браузер надсилає оригінальний запит для отримання HTML-документа, який представляє сторінку. Потім він аналізує цей файл, роблячи додаткові запити, що відповідають сценаріям виконання, інформації про макет (CSS) для відображення та підресурсів, що містяться на сторінці (зазвичай зображення та відео). Потім веб-браузер поєднує ці ресурси, щоб представити повний документ, веб-сторінку. Сценарії, які виконує браузер, можуть отримати більше ресурсів на наступних етапах, і браузер відповідно оновлює веб-сторінку.

Веб-сторінка – це гіпертекстовий документ. Це означає, що деякі частини відображуваного вмісту є посиланнями, які можна активувати (зазвичай клацанням миші) для отримання нової веб-сторінки, дозволяючи користувачеві керувати своїм агентом користувача та переходити в Інтернеті. Браузер перетворює ці вказівки в HTTP-запити, а потім інтерпретує відповіді HTTP, щоб надати користувачеві чітку відповідь.

На протилежній стороні каналу зв'язку знаходиться сервер, який обслуговує документ за запитом клієнта. Сервер віртуально виглядає лише як одна машина; але насправді це може бути сукупність серверів, які розподіляють навантаження (балансування навантаження), або інше програмне забезпечення (таке як кеш-пам’ять, сервер бази даних або сервери електронної комерції), яке повністю або частково генерує документ на вимогу.

Сервер – це не обов’язково одна машина, але на одній машині може бути розміщено кілька примірників серверного програмного забезпечення. За допомогою HTTP/1.1 і Host заголовка вони можуть навіть мати однакову IP-адресу.

Між веб-браузером і сервером багато комп’ютерів і машин передають HTTP-повідомлення. Завдяки багаторівневій структурі веб-стеку більшість із них працюють на транспортному, мережевому або фізичному рівнях, стаючи

прозорими на рівні HTTP та потенційно маючи значний вплив на продуктивність. Ті, що працюють на прикладних рівнях, зазвичай називаються проксі . Вони можуть бути прозорими, пересилаючи отримані запити, не змінюючи їх жодним чином, або непрозорими, у цьому випадку вони певним чином змінять запит перед тим, як передати його на сервер. Проксі можуть виконувати численні функції:

- кешування (кеш може бути публічним або приватним, як кеш браузера);
- фільтрація (наприклад, антивірусне сканування або батьківський контроль);
- балансування навантаження (щоб дозволити кільком серверам обслуговувати різні запити);
- аутентифікація (для контролю доступу до різних ресурсів);
- журналювання (що дозволяє зберігати історичну інформацію).

Серед основних аспектів HTTP можна виділити простоту, тобто HTTP, як правило, розроблений таким чином, щоб бути простим і зрозумілим для людини, навіть з додатковими складнощами, представленими в HTTP/2 завдяки інкапсуляції HTTP-повідомлень у фрейми. Люди можуть читати та розуміти HTTP-повідомлення, що полегшує тестування для розробників і зменшує складність для новачків.

HTTP є розширюваним, заголовки HTTP, представлені в HTTP/1.0, спрощують розширення та експерименти з цим протоколом. Нова функціональність може бути введена навіть шляхом простої угоди між клієнтом і сервером щодо семантики нового заголовка.

HTTP не має стану: немає зв’язку між двома запитами, які послідовно виконуються в одному з’єднанні. Це одразу створює проблеми для користувачів, які намагаються узгоджено взаємодіяти з певними сторінками, наприклад, використовуючи кошики для покупок електронної комерції. Але хоча сама ядро HTTP не має стану, файли cookie HTTP дозволяють використовувати сеанси з відстеженням стану. Використовуючи розширюваність заголовка, до робочого

процесу додаються файли cookie HTTP, що дозволяє створювати сеанс для кожного HTTP-запиту, щоб спільно використовувати той самий контекст або той самий стан.

З'єднання контролюється на транспортному рівні, а отже, принципово поза сферою дії HTTP. HTTP не вимагає, щоб базовий транспортний протокол ґрунтувався на з'єднанні; він лише вимагає, щоб він був надійним або не втрачав повідомлення (у таких випадках, як мінімум, представляючи помилку). Серед двох найпоширеніших транспортних протоколів в Інтернеті TCP є надійним, а UDP - ні. Тому HTTP покладається на стандарт TCP, який базується на підключенні.

Перш ніж клієнт і сервер зможуть обмінюватися парою HTTP-запит/відповідь, вони повинні встановити TCP-з'єднання, процес, який вимагає кількох зворотних передачі. Типовою поведінкою HTTP/1.0 є відкриття окремого TCP-з'єднання для кожної пари HTTP-запит/відповідь. Це менш ефективно, ніж спільне використання одного TCP-з'єднання, коли кілька запитів надсилаються підряд.

Щоб пом'якшити цей недолік, HTTP/1.1 запровадив конвеєрну передачу (яку виявилося важко реалізувати) і постійні з'єднання : основним з'єднанням TCP можна частково керувати за допомогою Connection заголовка. HTTP/2 пішов далі, мультиплексуючи повідомлення через одне з'єднання, допомагаючи підтримувати з'єднання теплим і ефективнішим.

Тривають експерименти з розробки кращого транспортного протоколу, який більше підходить для HTTP. Наприклад, Google експериментує з QUIC , який базується на UDP, щоб забезпечити більш надійний і ефективний транспортний протокол.

Що можна контролювати за допомогою HTTP? Ця розширювана природа HTTP з часом дозволила розширити контроль і функціональність Інтернету. Методи кешу та автентифікації були функціями, які оброблялися на початку історії HTTP. Здатність послабити обмеження походження , навпаки, була додана лише в 2010-х роках.

Ось список загальних функцій, якими можна керувати за допомогою HTTP:

- кешування. Спосіб кешування документів можна контролювати за допомогою HTTP. Сервер може давати вказівки проксі-серверам і клієнтам про те, що кешувати і як довго. Клієнт може наказати проміжним кеш-проксі ігнорувати збережений документ;
- пом’якшення обмеження щодо походження. Щоб запобігти відстеженню та іншим вторгненням у конфіденційність, веб-браузери забезпечують суворе розділення веб-сайтів. Лише сторінки з одного походження можуть отримати доступ до всієї інформації веб-сторінки. Хоча таке обмеження є тягарем для сервера, HTTP-заголовки можуть послабити це суворе розділення на стороні сервера, дозволяючи документу стати мозаїкою інформації, отриманої з різних доменів, для цього можуть бути навіть причини, пов’язані з безпекою;
- автентифікація. Деякі сторінки можуть бути захищені, щоб лише певні користувачі мали до них доступ. Базову автентифікацію може забезпечувати HTTP за допомогою або WWW-Authenticate подібних заголовків, або шляхом встановлення певного сеансу за допомогою файлів cookie HTTP;
- проксі та тунелювання. Сервери або клієнти часто знаходяться в інтрамережах і приховують свою справжню IP-адресу від інших комп’ютерів. Потім запити HTTP проходять через проксі-сервери, щоб подолати цей мережевий бар’єр. Не всі проксі є HTTP-проксі. Протокол SOCKS, наприклад, працює на нижчому рівні. Ці проксі-сервери можуть працювати з іншими протоколами, як-от ftp;
- сеанси. Використання файлів cookie HTTP дозволяє пов’язувати запити зі станом сервера. Це створює сеанси, незважаючи на те, що базовий HTTP є протоколом без стану. Це корисно не лише для кошиків для покупок електронної комерції, але й для будь-якого сайту, що дозволяє користувачеві налаштовувати вихідні дані.

Повідомлення HTTP, як визначено в HTTP/1.1 і раніше, читаються людиною. У HTTP/2 ці повідомлення вбудовано в двійкову структуру, фрейм,

що дозволяє оптимізувати, як-от стиснення заголовків і мультиплексування. Навіть якщо в цій версії HTTP надсилається лише частина оригінального HTTP-повідомлення, семантика кожного повідомлення не змінюється, і клієнт відновлює (практично) вихідний HTTP/1.1-запит. Тому корисно розуміти повідомлення HTTP/2 у форматі HTTP/1.1.

Існує два типи HTTP-повідомлень, запитів (рисунок 1.4) і відповідей (рисунок 1.5), кожен з яких має свій формат.

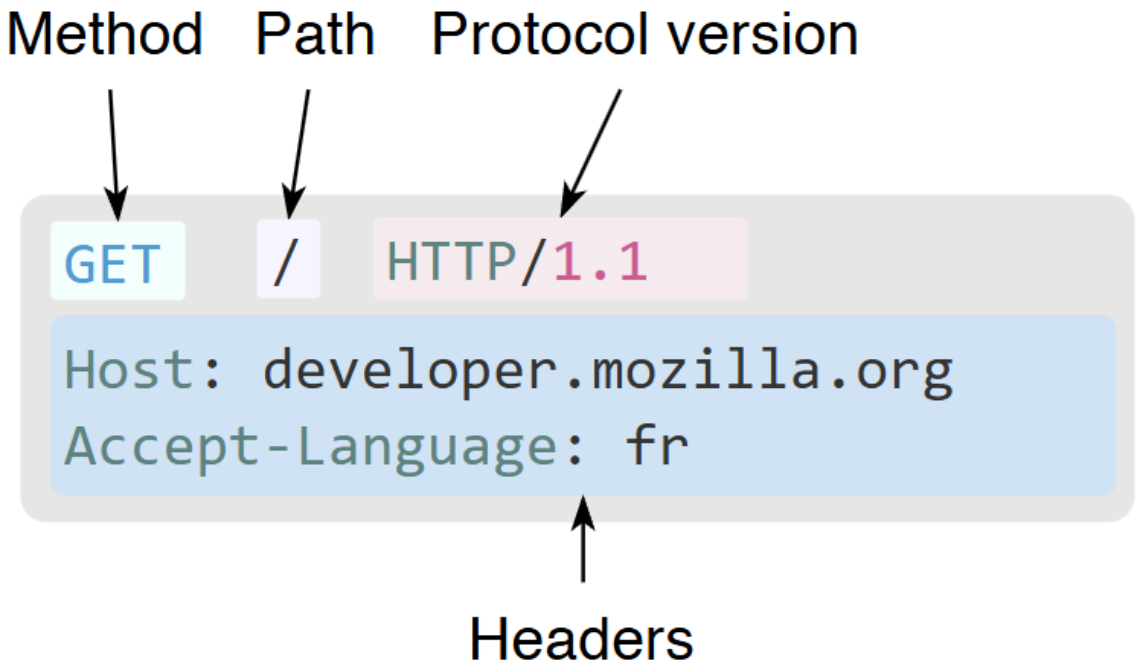

Рисунок 1.4 – Приклад HTTP запиту

Запити складаються з таких елементів:

- метод HTTP , як правило, дієслово на кшталт GET, POST або іменник на зразок OPTIONS чи HEAD, який визначає операцію, яку хоче виконати клієнт. Як правило, клієнт хоче отримати ресурс (за допомогою GET) або опублікувати значення форми HTML (за допомогою POST), хоча в інших випадках може знадобитися більше операцій;
- шлях ресурсу для отримання; URL-адреса ресурсу, позбавлена елементів, очевидних із контексту, наприклад, без протоколу (http://), домену (тут, developer.mozilla.org) або порту TCP (тут, 80);

- версія протоколу HTTP;
- додаткові заголовки , які передають додаткову інформацію для серверів;
- тіло для деяких методів, таких як POST, подібне до тих у відповідях, які містять надісланий ресурс.

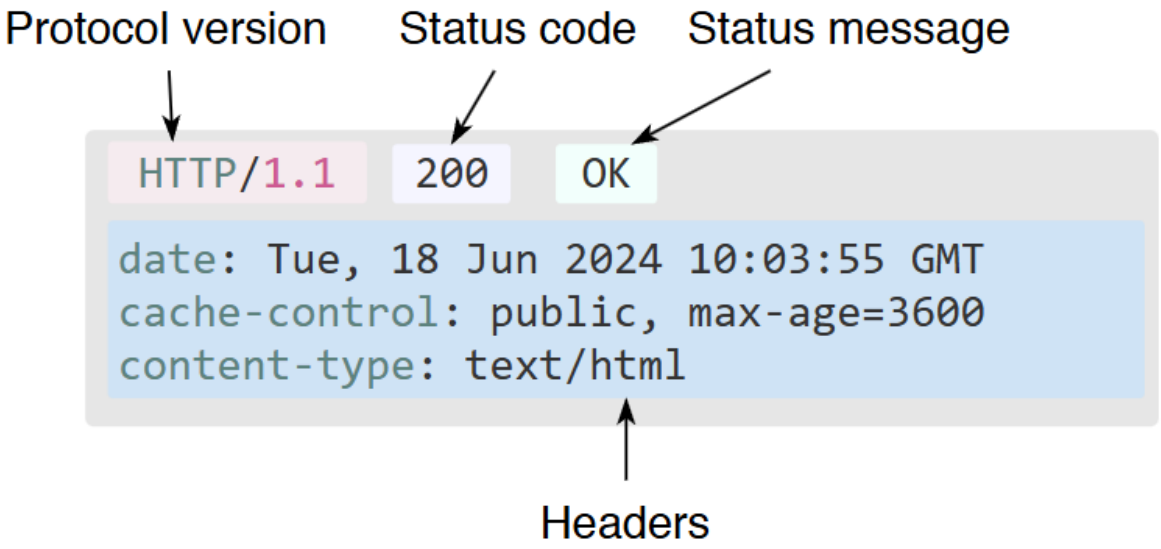

Рисунок 1.5 – Приклад HTTP відповіді

Відповіді складаються з таких елементів:

- версія протоколу HTTP, яку вони використовують;
- код статусу , що вказує, чи був запит успішним чи ні, і чому;
- повідомлення про статус, неавторитетний короткий опис коду статусу;
- HTTP- заголовки , такі як для запитів;
- необов'язково, тіло, що містить отриманий ресурс.

API на основі HTTP. Найпоширенішим API на основі HTTP є Fetch API , який можна використовувати для надсилання HTTP-запитів із JavaScript. API Fetch замінює XMLHttpRequestAPI.

Інший API, події, надіслані сервером , є односторонньою службою, яка дозволяє серверу надсилати події клієнту, використовуючи HTTP як транспортний механізм. Використовуючи EventSource інтерфейс, клієнт

відкриває з’єднання та встановлює обробники подій. Браузер клієнта автоматично перетворює повідомлення, які надходять через потік HTTP, у відповідні Event об’єкти. Потім він доставляє їх до обробників подій, які були зареєстровані для подій, type якщо вони відомі, або до onmessage обробника подій, якщо обробник подій для конкретного типу не встановлено.

Отже, HTTP – це розширюваний протокол, простий у використанні. Структура клієнт-сервер у поєднанні з можливістю додавання заголовків дозволяє HTTP розвиватися разом із розширеними можливостями Інтернету.

Хоча HTTP/2 додає певної складності, вбудовуючи HTTP-повідомлення у фрейми для підвищення продуктивності, основна структура повідомлень залишається незмінною з HTTP/1.0. Потік сеансу залишається базовим, що дозволяє його досліджувати та налагоджувати за допомогою монітора мережі HTTP .

1.4 Висновки по розділу

В ході дослідження аналогічних сервісів та серверних додатків, аналізу їх сильних і слабких сторін стало можливим оцінити, які саме проблеми необхідно подолати в даній системі.

Необхідно створити код телеграм бота та додати необхідний функціонал з можливістю масштабування системи в подальшому з використанням API для отримання даних з пристрою розумного дому.

Головною ціллю створення коду є оптимізація роботи зв’язки API та Telegram Бота у вигляді серверного додатку для кращої обробки інформації з датчиків.

Визначено та описано наукову новизну розробки, переваги якої це зручність для кінцевого користувача у вигляді відсутності потреби

завантажувати окремий додаток для того щоб взаємодіяти з пристроєм розумного дому.

Згідно з розробленими функціональними та нефункціональними вимогами встановимо комплекс завдань, що має вирішити розроблюваний програмний модуль:

- надати користувачу можливість вільно користуватись ботом;
- надати користувачу можливість використовувати повний функціонал бота
- надати користувачу можливість переглядати дані з датчика пристрою розумного дому;
- надати користувачу можливість додавати інших користувачів, які зможуть переглядати дані з того чи іншого датчика;
- надати користувачу можливість отримати статус роботи датчика;
- надати користувачу можливість та функціонал керування датчиком чи пристроєм розумного дому.

2 ДОСЛІДЖЕННЯ РОБОТИ БОТУ ТА API

2.1 Основні функції та алгоритми ботів

За своєю суттю можна розглядати Telegram Bot API як програмне забезпечення, яке надає JSON-кодовані відповіді на запити користувача.

З іншого боку, бот – це, по суті, програма, програмне забезпечення або сценарій, який запитує API за допомогою запиту HTTPS і чекає на відповідь. Є можливість робити кілька типів запитів, а також багато різних об'єктів, які можна використовувати та отримувати як відповіді.

Для роботи боту потрібне постійне підключення до інтернету, база даних, токен та код, який і буде диктувати поведінку бота (рисунок 2.1).

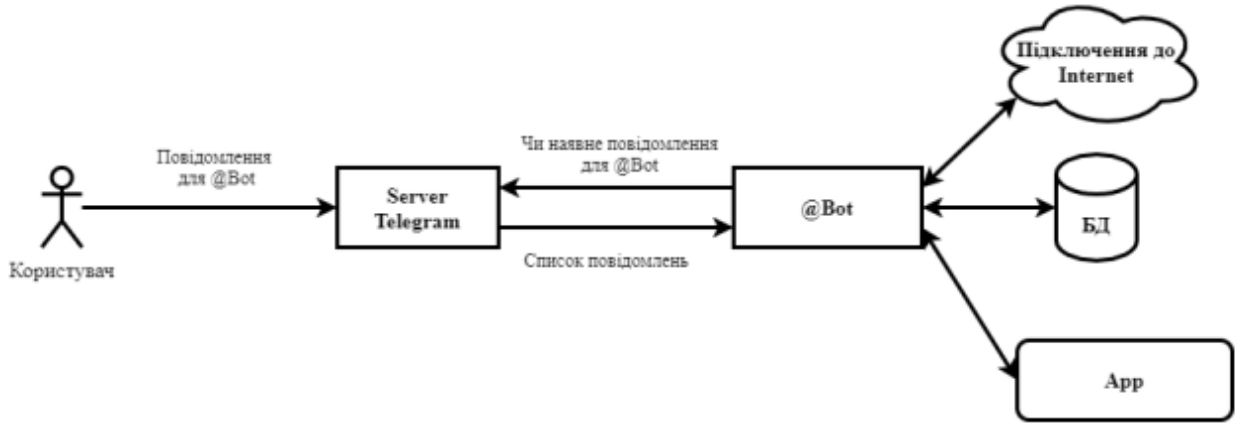

Рисунок 2.1 – Схема алгоритму роботи з Telegram ботом

Користувачі можуть надсилати ботам будь-які типи повідомлень, включаючи текст, файли, розташування, наклейки та голосові повідомлення. Однак боти Telegram пропонують багато інших інструментів для створення гнучких інтерфейсів, адаптованих до конкретних потреб будь якого користувача:

- команди, які підсвічуються в повідомленнях і можуть бути обрані зі списку після введення /;
- клавіатури, які замінюють клавіатуру користувача з попередньо визначеними варіантами відповідей;

- кнопки, які відображаються біля повідомлень від бота.

Для ще більшої гнучкості веб-програми підтримують 100% користувальницькі інтерфейси з JavaScript.

Боти Telegram можуть підтримувати кілька мов , які адаптуються до мовних налаштувань користувачів у програмі.

Розглянемо вищезгадані пункти детальніше. Проста команда /keyword вказує боту, що робити. Можливості програми Telegram для зручного керування ботом:

- виділіть команди в повідомленнях. Коли користувач торкається виділеної команди, ця команда негайно надсилається знову;
- запропонуйте список підтримуваних команд з описами, коли користувач вводить /(щоб це працювало, потрібно надати список команд @BotFather або за допомогою відповідного методу API). Вибір команди зі списку негайно надсилає її;
- показати кнопку меню , що містить усі або деякі команди бота (які можна налаштувати через @BotFather) (рисунок 2.2).

Команди завжди мають починатися з /символу та містити до 32 символів . Вони можуть використовувати латинські літери, цифри та підкреслення, хоча для чіткішого вигляду рекомендується простий текст у нижньому регістрі.

Ось кілька прикладів:

- /далі
- /скасувати
- /нове розташування
- /нове правило

Команди мають бути якомога конкретнішими – наприклад, /newlocation або /newrule краще, ніж /new команда, яка потім вимагає від користувача додатковий параметр, як-от «розташування» або «правило».

У всіх бот-чатах біля поля повідомлення з’являється кнопка меню. За замовчуванням натискання цієї кнопки відкриває меню , яке містить деякі або всі

команди бота, включаючи короткий опис для кожної. Потім користувачі можуть вибрати команду з меню, не вводячи її (рисунок 2.2).

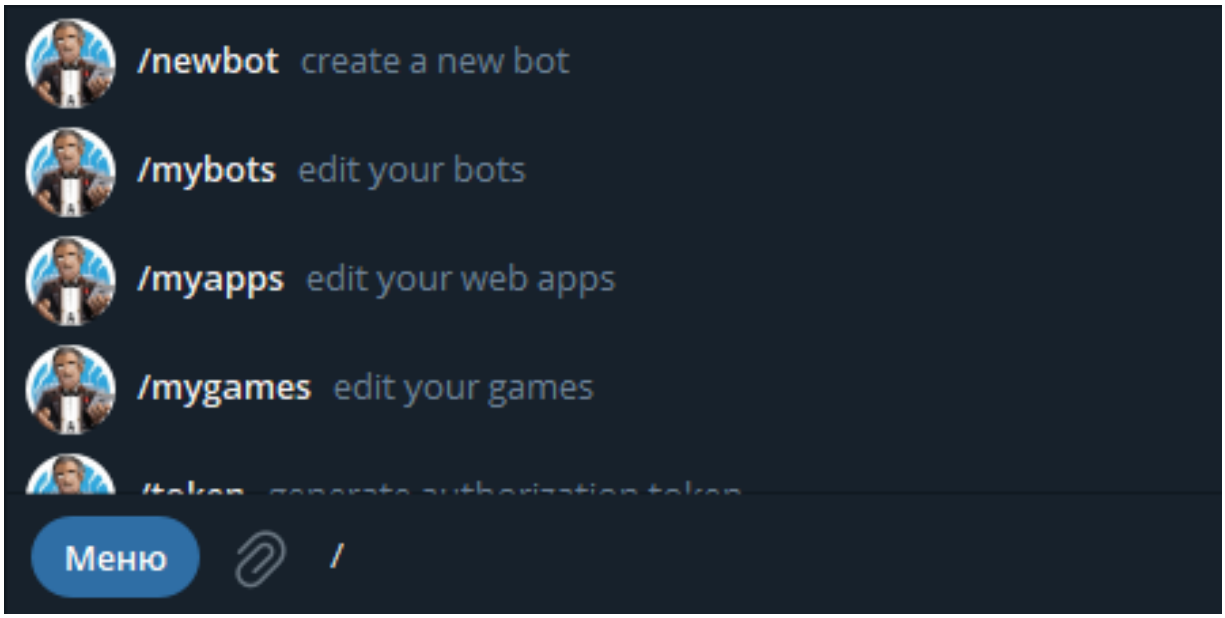

Рисунок 2.2 – Приклад використання команд та кнопки меню в боті

Боти можуть інтерпретувати вільний текст, введений користувачами, але надання конкретних пропозицій часто більш інтуїтивно зрозуміле – саме тут спеціальні клавіатури можуть бути надзвичайно корисними.

Щоразу, коли бот надсилає повідомлення, він може відображати спеціальну клавіатуру з попередньо визначеними параметрами відповіді. Програми Telegram, які отримують повідомлення, відображатимуть користувачеві клавіатуру. Використання будь-якої кнопки негайно надішле відповідний текст. Таким чином ви можете значно спростити та оптимізувати взаємодію користувача з вашим ботом. Непоганим прикладом для такого є офіційний чат-бот УкрПошти (рисунок 2.3).

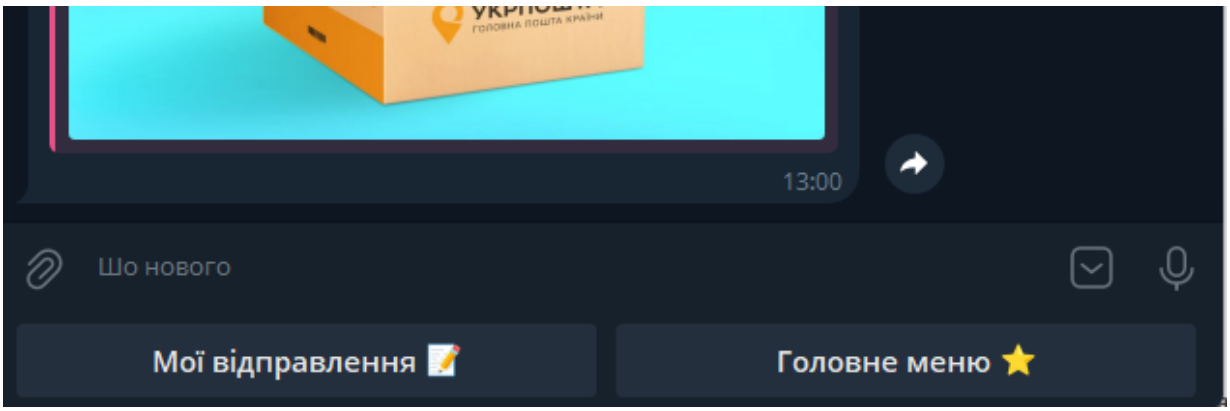

Рисунок 2.3 – Приклад використання клавіатури в боті

Корисним буде параметр one_time_keyboard , щоб автоматично приховати клавіатуру вашого бота, щойно вона використовується. Також є можливість налаштувати текстовий заповнювач у полі введення встановивши параметр input_field_placeholder.

Бувають випадки, коли є потреба робити щось, не надсилаючи жодних повідомлень у чат, наприклад, коли користувач змінює налаштування, перемикає параметри або переміщується в результатах пошуку. У таких випадках варто використовувати вбудовані клавіатури, які показано безпосередньо під відповідними повідомленнями (рисунок 2.4).

На відміну від спеціальних клавіатур з відповідями, натискання кнопок на вбудованих клавіатурах не надсилає повідомлення до чату . Натомість вбудовані клавіатури підтримують кнопки, які можуть працювати за лаштунками або відкривати різні інтерфейси, такі як кнопки зворотного виклику, кнопки URL-адреси, кнопки перемикання на вбудовану кнопку, кнопки гри та кнопки оплати .

Щоб забезпечити кращий досвід роботи з користувачем , можна врахувати можливість редагування клавіатури, коли користувач перемикає кнопку налаштувань або переходить на нову сторінку – це і швидше , і плавніше, ніж надсилання цілого нового повідомлення та видалення попереднього.

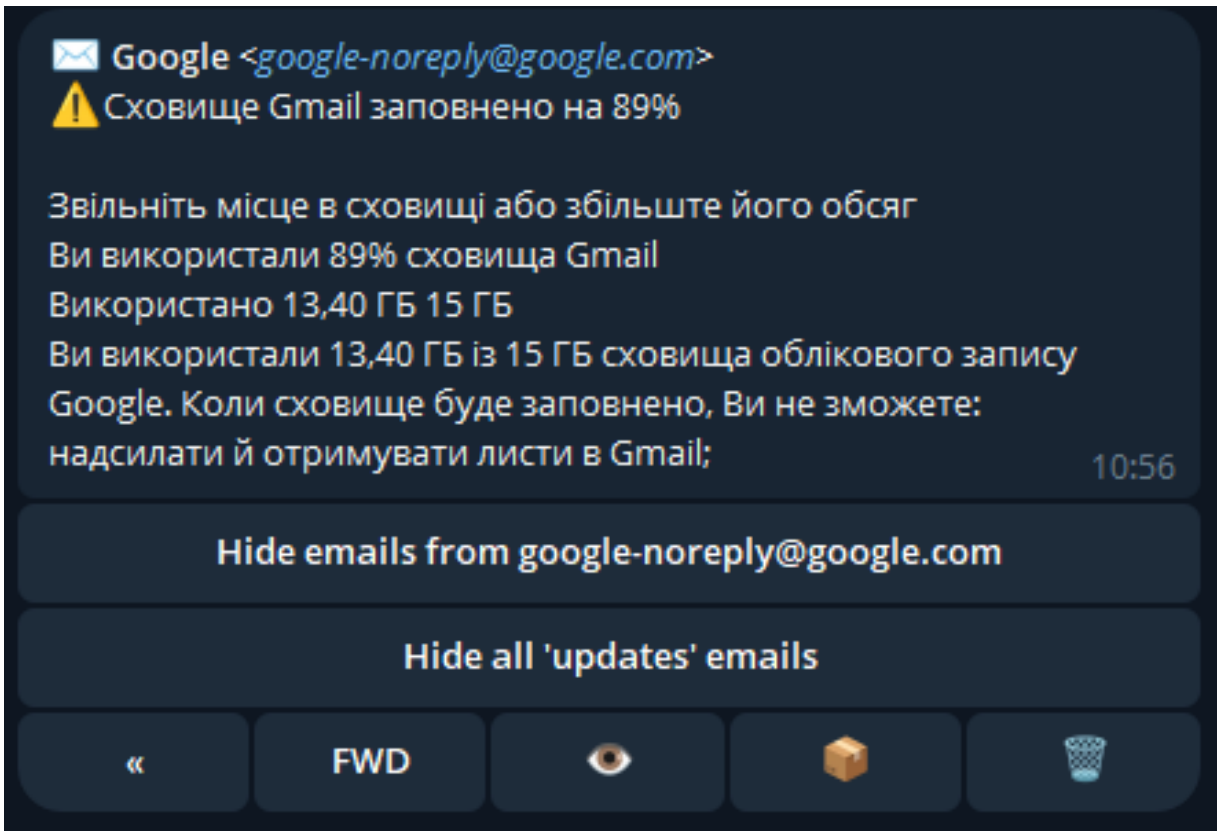

Рисунок 2.4 – Приклад використання вбудованих кнопок

Щоб зробити основні взаємодії більш уніфікованими, розробники мають підтримувати кілька основних команд. Додатки Telegram матимуть інтерфейсні ярлики для цих команд.

- /start – починає взаємодію з користувачем, як відправка вступного повідомлення. Цю команду також можна використовувати для передачі боту додаткових параметрів;
- /help – повертає довідкове повідомлення, наприклад короткий текст про те, що може робити ваш бот, і список команд;
- /settings – (якщо застосовано) показує налаштування бота для цього користувача та пропонує команди для їх редагування.

Користувачі побачать кнопку «Пуск» , коли вперше відкриють чат із вашим ботом. Посилання на довідку та налаштування будуть доступні в меню на сторінці профілю бота, якщо розробник додасть їх у @BotFather.

Боти можуть надати користувачеві зручний та інтуїтивно зрозумілий інтерфейс, який містить список будь-якої кількості груп, каналів або інших

користувачів відповідно до спеціального набору критеріїв. Торкання чату надсилає його ідентифікатор боту в службовому повідомленні та плавно закриває інтерфейс.

Бот для керування групами є ідеальним прикладом, де адміністратор може вибрати чат, яким має керувати бот, а потім вибрати користувача, якого він має просувати – це може відбуватись без жодного введення тексту за допомогою правильно запрограмованих кнопок, для цього:

- вибираємо набір критеріїв і збережіть їх в об’єкті KeyboardButtonRequestChat (або KeyboardButtonRequestUser для користувачів);
- створюємо KeyboardButton і збережіть критерії під request_chat або request_user відповідно;
- надсилаємо ReplyKeyboardMarkup , який містить кнопку, яку ви щойно створили.

Коли користувач вибере чат, ви отримаєте його ідентифікатор у службовому chat_shared або user_shared службовому повідомленні.

Окрім надсилання команд і повідомлень у чат із ботом, є кілька способів взаємодії з ними, не відкриваючи жодного конкретного чату чи групи.

Вбудований режим дозволяє надсилати запити ботам прямо з поля введення з будь-якого чату в Telegram.

Глибоке посилання дозволяє створювати спеціальні посилання, які надсилають певні параметри боту під час відкриття.

Інтеграція меню вкладень дає можливість використовувати ботів із меню вкладень у чатах.

Вбудовані запити. Користувачі можуть взаємодіяти з ботом за допомогою вбудованих запитів прямо з поля повідомлення в будь-якому чаті . Все, що їм потрібно зробити, це почати повідомлення з @username бота та ввести ключове слово (рисунок 2.5).

Отримавши запит, бот може видати деякі результати. Як тільки користувач вибирає один, він надсилається до відповідного чату. Таким чином, люди можуть

запитувати та надсилати вміст від бота в будь-якому зі своїх чатів, груп або каналів.

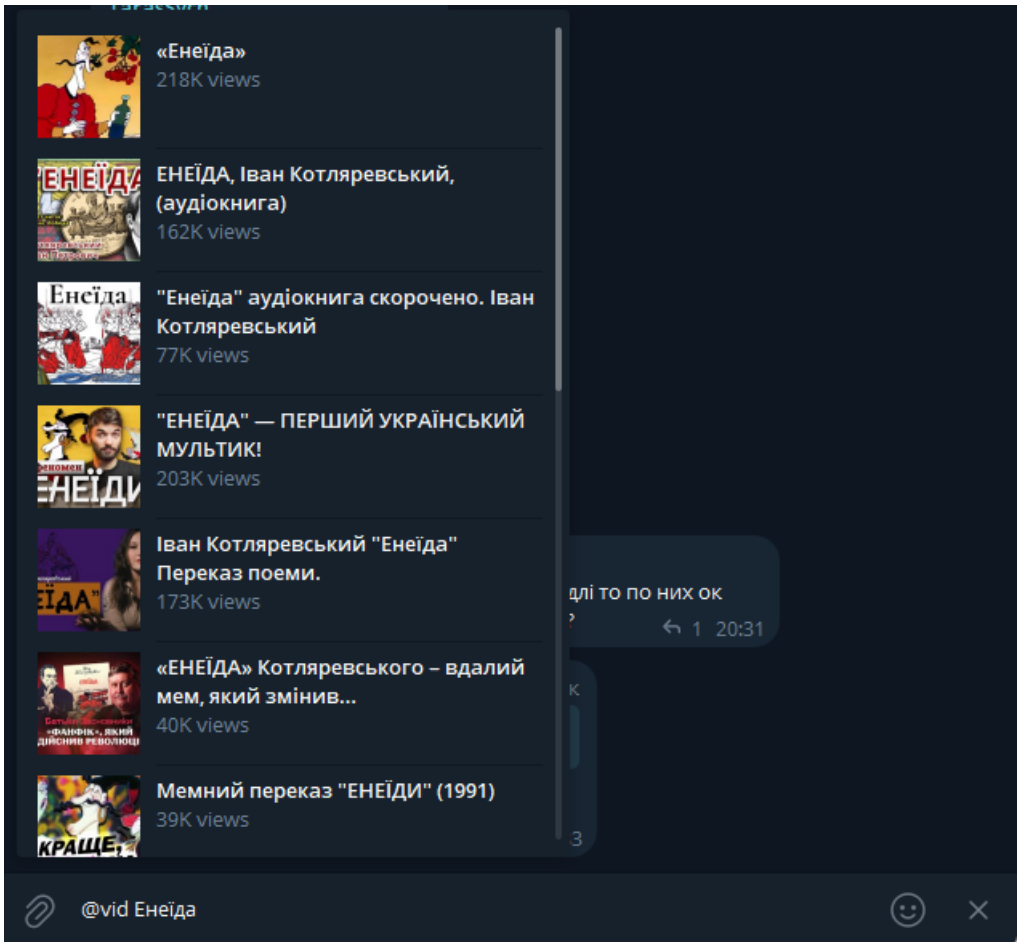

Рисунок 2.5 – Приклад використання вбудованого запиту

Глибоке посилання. Боти Telegram мають механізм глибокого посилання, який дозволяє передавати боту додаткові параметри під час запуску. Це може бути команда, яка запускає бота, або токен автентифікації для підключення облікового запису Telegram користувача до його облікового запису на іншій платформі.

Кожен бот має посилання, яке відкриває бесіду з ним у Telegram – https://t.me/<bot_username>. Параметри можна додавати безпосередньо до цього посилання, щоб дозволити вашому боту працювати з додатковою інформацією на льоту, без жодного введення користувача.

Допускаються AZ, az, 0-9, _ і рекомендується використовувати base64url для кодування параметрів у бінарному та інших типах вмісту. Параметр може мати до 64 символів.

Боти можуть увімкнути бізнес-режим , дозволяючи передплатникам Telegram Business підключати їх до свого облікового запису, щоб оптимізувати й автоматизувати керування приватним чатом і взаємодію зі своїми клієнтами.

Власник облікового запису може вказати, до яких чатів має доступ ваш бот – у цих чатах бот отримуватиме всі оновлення, які зазвичай підтримуються API бота, за винятком повідомлень, надісланих ним самим та іншими ботами. Залежно від налаштувань підключення до бізнесу ваш бот також може надсилати повідомлення та виконувати інші дії від імені власника облікового запису в чатах, які були активні протягом останніх 24 годин.

Користувачі, які підключають бота до свого облікового запису, побачать панель швидких дій у верхній частині кожного керованого чату – натискання «Керувати ботом» перенаправить їх до вашого бота, який отримає повідомлення глибокого посилання у форматі /start bizChat<user_chat_id>.

Також боти Telegram можуть приймати платежі за допомогою елегантного, оптимізованого інтерфейсу, який збирає всі необхідні дані від користувача. Telegram не збирає платіжні дані, як-от дані кредитної картки користувача, і надсилає їх безпосередньо одному з підтримуваних платіжних постачальників .

Боти можуть адаптувати свої інтерфейси для підтримки кількох мов , оновлюючи введені дані та інформацію на льоту. Language_code користувача включається в кожне відповідне оновлення як мовний тег IETF , що дозволяє роботам адаптуватися відповідно [25].

Розробники наполеглево рекомендують дотримуватися їхніх вказівок, щоб забезпечити найкращу взаємодію з користувачем, серед яких:

- інтерфейси, тексти та вбудовані результати мають адаптуватися до language_code без втручання користувача;
- підключені веб-програми отримають language_code користувача – HTML-сторінка має враховувати його;

- ігри HTML5 можуть отримати інформацію про мову, якщо ви вкажете її як параметр URL-адреси . Ви можете створити цей параметр із поля language_code в об’єкті User, який обслуговується початковою грою CallbackQuery;
- назва бота , опис і текст про нього можна локалізувати за допомогою відповідних методів.

Режим конфіденційності. Ботів часто додають до груп, щоб виконувати основні завдання або допомагати модераторам, наприклад автоматично публікувати оголошення компаній або навіть святкувати дні народження. За замовчуванням усі боти , додані до груп, працюють у режимі конфіденційності та бачать лише відповідні повідомлення та команди:

- команди, явно призначені для них (наприклад, /command@this_bot);
- загальні команди (наприклад, /start), якщо бот був останнім ботом, який надіслав повідомлення групі;
- вбудовані повідомлення, надіслані через бота;
- відповідає на будь-які повідомлення, приховано чи явно призначені для цього бота;
- всі боти також отримають, незалежно від режиму конфіденційності;
- всі службові повідомлення;
- всі повідомлення з приватних чатів;
- всі повідомлення з каналів, учасниками яких вони є.

Режим конфіденційності ввімкнено за замовчуванням для всіх ботів, крім ботів, які були додані до групи як адміністратори (адміністратори ботів завжди отримують усі повідомлення). Його можна вимкнути, щоб бот отримував усі повідомлення як звичайний користувач (щоб ця зміна вступила в силу, бота потрібно повторно додати до групи). Рекомендується робити це лише у випадках, коли це абсолютно необхідно для роботи бота. У більшості випадків використання параметра примусової відповіді для повідомлень бота має бути більш ніж достатньо.

Цей режим не тільки підвищує конфіденційність користувача, але й робить роботу бота більш ефективною, зменшуючи кількість вхідних даних, які йому необхідно обробити. Користувачі завжди можуть побачити поточні налаштування конфіденційності бота в списку учасників групи (рисунок 2.6).

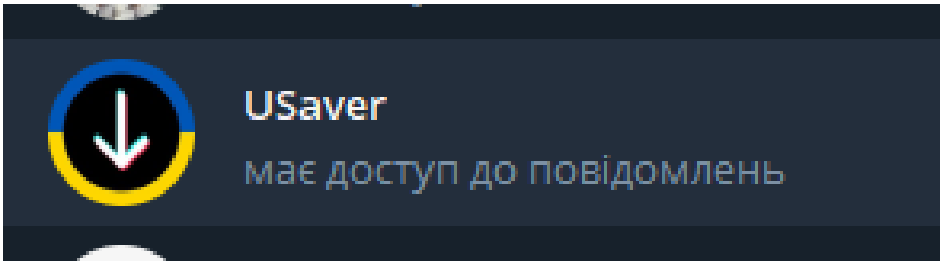

Рисунок 2.6 – Приклад показу налаштувань конфіденційності бота

Тестування бота. Розробник може швидко протестувати свого бота, не заважаючи його користувачам, просто запустивши інший екземпляр свого коду в іншому обліковому записі бота. Для цього потрібно створити нового бота через @BotFather , та отримавши токен і використовувати його в тестовому екземплярі коду.

Усе подальше тестування та налагодження можна проводити приватно на новому боті, не впливаючи на вихідний екземпляр. Якщо потрібно поділитися посиланнями на файли між ботами, варто зауважити, що file_id поле прив’язано до одного ідентифікатора бота, тому тестовий екземпляр не може використовувати спільну file_id базу даних для швидкого надсилання медіафайлів, файли потрібно повторно завантажувати окремо.

Про текст, опис і носії профілю. Коли нові користувачі відкриють бот, вони зустрінуться з корисним описом у вікні під назвою «Що може робити цей бот?».

Правильне налаштування цього поля в @BotFather дозволяє кожному відразу отримати уявлення про те, що може зробити той чи інший бот – його опис має бути коротким , по суті та по темі.

Також, можна додати фото чи відео в це поле Edit Description Picture за допомогою @BotFather .

Крім того, як і звичайні користувачі, боти також мають коротку біографію, доступну в їхніх профілях. Якщо розробник не вказав це поле під час першого створення свого бота, то він може будь-коли встановити його за допомогою /setabouttext команди в @BotFather . Користувачі можуть взаємодіяти з багатьма ботами, і вони не матимуть доступу до їхніх описів після їх запуску - швидке нагадування про мету бота може бути дуже корисним.

Варто зауважити, що текст «Опис» і «Про програму» можуть бути локально локалізовані - кожен користувач автоматично побачить правильний переклад для своєї мови.

Боти також можуть мати зображення профілю – варто вибрати щось унікальне та оригінальне, щоб користувачі могли швидко знайти це у своєму списку чатів.

Починаючи з 21 квітня 2023 року (версії Telegram 9.6), розробник можете редагувати свого бота безпосередньо зі сторінки його профілю (рисунок 2.7), зокрема встановлювати спеціальне відео профілю.

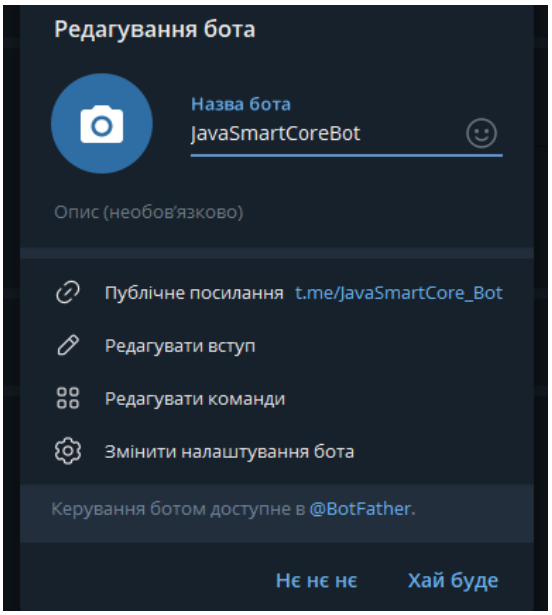

Рисунок 2.7 – Функція налаштування без посередньо в боті

Передача права власності. Розробник можете передати право власності на свого бота іншому розробникові.

Для цього потрібно надіслати /mybots, виберіть свого бота, а потім передайте право власності. Розробник можете передати бота лише тим користувачам, які взаємодіяли з ним хоча б один раз.

Передача права власності дасть повний контроль над ботом іншому користувачеві, то він зможе отримати доступ до повідомлень бота та навіть видалити його. Перенесення є постійним, тому, рекомендується подумати двічі перш ніж це робити.

API локального бота. Користувач може розміщувати та працювати з власним екземпляром нашого відкритого коду Bot API.

Головне, після встановлення сервера потрібно не забути використати метод виходу з системи , перш ніж перенаправляти запити на нову URL-адресу локального API.

Локальний екземпляр працює на порту 8081 за замовчуванням і прийматиме лише запити HTTP, тому для обробки віддалених запитів HTTPS потрібно використовувати проксі-сервер завершення TLS [26].

Розміщуючи API локально, користувач отримаєте доступ до деяких оновлень, які наведені у таблиці 2.2 .

Таблиця 2.2 – Перелік оновлень API

API	Максимальне завантаження файлу	Максимальне завантаження файлів	URL-адреса Whook	Порт WHook	Максимальна кількість підключень WHook
Офіційний	20 МБ	50 МБ	HTTPS	443, 80, 88, 8443	1-100
Локальний	Необмежене	2000 МБ	HTTP	Будь який порт	1-100000

2.2 Алгоритми роботи API та їх архітектури

Інтерфейс прикладного програмування, зазвичай скорочений до API, — це набір правил, які визначають, як одна програмна програма може отримати доступ до даних або функцій, наданих іншою програмною програмою.

API є важливою частиною сучасної розробки програмного забезпечення. Вони дозволяють різним системам і програмам взаємодіяти одна з одною та ділитися функціями гнучким і ефективним способом. Вони використовуються в різноманітних контекстах, включаючи веб-розробку, мобільні програми та програми Інтернету речей (IoT) [27].

API часто мають форму бібліотеки, яку розробник програмного забезпечення може включити у свій код. Ця бібліотека надає набір функцій, які можна використовувати для виконання різноманітних завдань. API визначає виклики функцій, вхідні дані, які вони приймають, і результати, які вони повертають.

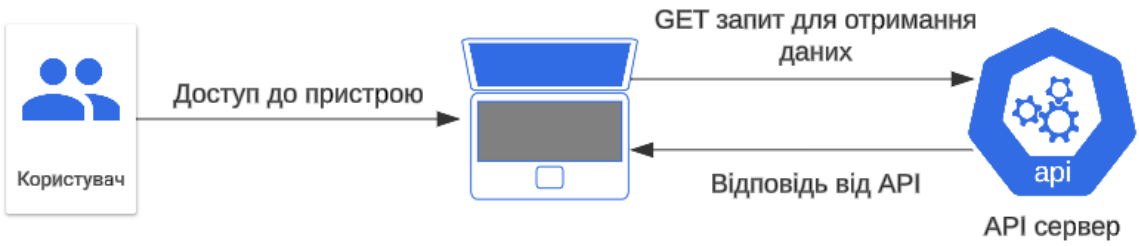

Рисунок 2.8 – Схематична демонстрація роботи API

Спрощений сценарій роботи API може виглядати так (рисунок 2.8):

- клієнт надсилає запит на сервер API, зазвичай через Інтернет або локальну мережу. Запит виконується за допомогою спеціального протоколу (наприклад, HTTP) і містить інформацію про операцію, яку хоче виконати клієнт, наприклад, отримання даних або оновлення ресурсу;

- сервер API отримує запит і обробляє його. Він може підтверджувати запит, автентифікувати клієнта, авторизувати запит або виконувати інші необхідні операції;
- сервер API надсилає відповідь клієнту, яка може містити дані, повідомлення про помилку або код стану, що вказує на результат операції;
- клієнт отримує відповідь і обробляє її.

Виклик API – це запит до API для доступу до даних або функцій. Клієнт здійснює виклик API і надсилає запит на сервер API, а сервер надсилає відповідь. Запит і відповідь використовують певний формат і структуру та передаються за допомогою певного протоколу (наприклад, HTTP).

Щоб здійснити виклик API, ви повинні мати доступ до документації API. Ця документація надає інформацію про можливості API, структуру запитів і відповідей, а також будь-які вимоги до автентифікації чи авторизації. Вам також може знадобитися отримати ключ API або інші облікові дані, перш ніж ви зможете здійснювати виклики API.

Ключ API – це унікальний ідентифікатор, який використовується для автентифікації запитів API. Ключі API зазвичай відстежують і контролюють, як ви можете використовувати API. Вони також можуть надавати доступ до даних і функцій API [28].

Ключі API керують правильною роботою API відповідно до умов використання. Наприклад, вони можуть обмежити кількість запитів API, які може зробити клієнт. Ключ API також може відстежувати використання API або переконатися, що дані, надані API, використовуються належним чином.

Вони зазвичай надаються постачальником API і потрібні для виконання запитів API. Їх можна створити та керувати ними через портал розробника постачальника API, і вони часто стосуються конкретного API або програми.

Такі ключі зазвичай присутні в заголовку запиту API та використовуються для автентифікації запиту. Інші облікові дані автентифікації та авторизації, як-от маркер OAuth або веб-токен JSON (JWT), також можна використовувати разом.

Кінцева точка API (endpoint) – це певне розташування на сервері, яке може отримувати запити API і відповідати на них. Кінцева точка API ідентифікується URI (уніфікованим ідентифікатором ресурсу) і зазвичай пов’язана з певним API або службою.

І API, і веб-хуки – це два способи, за допомогою яких різні програмні компоненти можуть спілкуватися один з одним через Інтернет. Однак вони відрізняються тим, як працюють і коли використовуються.

Як пояснювалося вище, API – це набір правил і протоколів, які визначають, як різні компоненти програмного забезпечення можуть взаємодіяти один з одним. Вони надають можливість різним системам і програмам спілкуватися одна з одною, обмінюватися даними та функціями, а також працювати разом для виконання завдань. Щоб використовувати API, клієнт надсилає запит на сервер API за допомогою певного протоколу (наприклад, HTTP) і отримує відповідь. Клієнт і сервер спілкуються за допомогою серії запитів і відповідей, а API визначає конкретний формат і структуру цих повідомлень [29].

Чим API відрізняються від веб-хуків? Веб-хуки – це спосіб, за допомогою якого сервер надсилає дані клієнту в режимі реального часу, без необхідності клієнту робити запит. Коли на сервері відбувається певна подія, сервер може надіслати повідомлення на вказану URL-адресу. Зазвичай повідомлення має форму запиту HTTP POST. Потім клієнт може обробити повідомлення та діяти на основі отриманих даних. Веб-хуки часто використовуються для забезпечення зв’язку в реальному часі та архітектури, керованої подіями, і можуть використовуватися для створення більш ефективних і чутливих систем.

І API, і веб-хуки є неймовірно корисними інструментами для створення взаємопов’язаних систем, але вони застосовуються в різних ситуаціях і для різних цілей. Щоб дізнатися більше про відмінності, перегляньте нашу публікацію про порівняння Webhooks і API [30] .

Сфери застосування API безмежні завдяки тій самій універсальності та гнучкості, до прикладу:

- веб-розробка. API є основним компонентом веб-додатків, які доступні з будь-якого пристрою з підключенням до Інтернету. Наприклад, API дозволить програмі у будь якому веб-браузері спілкуватися з програмою на стороні сервера. Крім того, це дозволить веб-програмі отримувати дані з бази даних;
- мобільні додатки. Часто API використовується для створення мобільних додатків, які можуть отримувати доступ до даних і функціональних можливостей із серверної програми або хмарної служби. Наприклад, програма соціальних медіа на смартфоні може використовувати API, щоб отримати канал користувача або опублікувати нову публікаціюж
- інтернет речей(IoT). API також забезпечують зв'язок між пристроями IoT та іншими системами та програмами. Наприклад, API може полегшити зв'язок між розумним термостатом і системою домашньої автоматизації. В іншому випадку це може дозволити пристрою IoT надсилати дані в хмарний сервіс для аналізу [31, 32];
- мікросервіси. API надзвичайно популярні для створення архітектур мікросервісів . Мікросервіс є результатом розбиття великої програми на менші незалежні служби, які спілкуються один з одним через API. За допомогою окремих частин різні служби можна розробляти, тестувати та розгортати незалежно, що полегшує масштабування та підтримку загальної програми [33];
- сторонні служби та послуги. API дозволяють різним програмам і службам працювати разом і обмінюватися даними. Наприклад, система управління взаємовідносинами з клієнтами (CRM) може використовувати API для інтеграції з платформою автоматизації маркетингу. Або інструмент керування проектом може використовувати API для інтеграції з програмою відстеження часу [34].

Не варто забувати, шо існує багато різних типів API, типи яких наведені у таблиці 2.3.

Таблиця 2.3 - Типи API

Тип	Опис
Відкриті API	Вони надаються розробникам за межами організації, яка їх створила. Відкриті API надають доступ до певного продукту чи послуги та доступні кожному, хто погоджується з умовами використання. Вони також відомі як зовнішні або публічні API.
Внутрішні API	Вони використовуються в організації для спільного використання ресурсів і функцій між різними командами або системами. Зазвичай вони недоступні розробникам за межами організації.
Партнерські API	Ці API доступні певній групі розробників, які мають ділові стосунки з організацією, яка створила API. Зазвичай ця група складається з партнерів або сторонніх розробників.
Композитні API	Це API, які поєднують кілька основних API в один інтерфейс. Вони надають розробникам спрощений спосіб доступу до кількох ресурсів або функцій за один виклик.

Варто зазначити, що ці категорії не є взаємовиключними, і API може належати до кількох категорій залежно від того, як він використовується.

Сучасні серверні додатки та інтелектуальні системи часто використовують API як основний спосіб обміну даними між компонентами. Архітектура API забезпечує структуровану комунікацію, дозволяючи додаткам взаємодіяти один з одним через стандартизовані запити і відповіді. Вона є критично важливою для забезпечення швидкості, надійності та безпеки обміну даними, особливо в умовах складних, розподілених систем. Далі розглянемо ключові аспекти архітектури API, які формують основу для створення продуктивних і масштабованих рішень [35].

REST API – це тип веб-API, який використовує HTTP-запити для маніпулювання даними. Вони розроблені як легкі та гнучкі та чудово підходять

для створення веб-сервісів, які масштабуються та прості в обслуговуванні. API REST використовують фіксований набір дієслів HTTP для виконання операцій над ресурсами, ідентифікованими за допомогою URI (уніфікованого ідентифікатора ресурсу). Прикладами таких дієслів є GET, POST, PUT і DELETE. REST API також корисні для створення програм CRUD (Create, Read, Update, Delete).

SOAP API – це тип веб-API, який використовує XML (розширювану мову розмітки) для кодування повідомлень. Вони розроблені таким чином, щоб бути розширюваними та нейтральними для створення веб-сервісів, сумісних між різними мовами програмування та платформами. API SOAP використовують стандартизований формат обміну повідомленнями та транспортний протокол (зазвичай HTTP) для надсилання та отримання повідомлень.

RPC API – це тип API, призначений для полегшення виклику методів на віддалених об’єктах, що корисно для створення розподілених систем. API RPC базуються на моделі клієнт-сервер. Клієнт надсилає серверу запит на виконання певної процедури чи функції, а сервер повертає результат клієнту.

GraphQL – це мова запитів і середовище виконання, розроблені Facebook як альтернатива REST і SOAP API. GraphQL створює представлення ваших даних, яке спроектоване так, щоб воно було знайомим і природним, як візуальний графік. Графова частина GraphQL описує структуру даних колекції об’єктів, або вузлів, які з’єднані один з одним через набір посилань або ребер. Зв’язки між різними об’єктами можуть бути представлені в інтерфейсі користувача як результат цієї графової структури. Ключова концепція полягає в тому, що структура даних є нелінійною, тобто один об’єкт може бути пов’язаний з кількома іншими об’єктами, а зв’язки також можуть бути циклічними.

То які тут переваги? Використовуючи GraphQL, клієнти можуть запитувати лише ті дані, які їм потрібні, а не отримувати фіксований набір даних із сервера. Він гнучкий і ефективний, а також простий у використанні та вивченні, має простий синтаксис і потужні інструменти для створення та тестування запитів [36].

Ви можете здивуватися, дізнавшись, що інтерфейси програмування прикладних програм виникли ще до існування Всесвітньої павутини. Ще в 1940-х роках, власне, з появою перших комп'ютерів.

Британські комп’ютерні вчені Маріс Вілкс і Девід Уілер працювали над бібліотекою програмного забезпечення для електронного автоматичного калькулятора зберігання затримок (EDSAC). Пара запрограмувала EDSAC приймати різноманітні інструкції, такі як додавання, віднімання, друк, завантаження та збереження. Ця поведінка подібна до того, як сучасні веб-API взаємодіють із даними сьогодні, наприклад, «додати публікацію блогу», «видалити публікацію блогу» або «отримати інформацію про публікацію блогу» [37].

Вілкс і Уілер задокументували використання EDSAC через каталог приміток щодо його функціональності та способів його інтеграції з іншими програмами. Це був перший приклад того типу документації API, який ми знаємо сьогодні.

Після прочитаного вище, виникає питання хто створює та підтримує API зараз. Насправді, API може створювати будь-яка група чи окрема особа, включаючи компанії, що займаються програмним забезпеченням, урядові установи, некомерційні організації та проекти з відкритим кодом [38]. До відомих компаній які створюють API входять такі гіганти як:

- Google – багатонаціональна технологічна компанія, яка створює широкий спектр API, у тому числі API для хмарних обчислень, карт, пошуку та машинного навчання;
- Amazon – це глобальна компанія електронної комерції, яка надає різноманітні API, зокрема API для електронної комерції, виконання й обробки платежів;
- Microsoft є технологічною компанією, яка створює широкий спектр API, у тому числі API для хмарних обчислень, продуктивності та машинного навчання.

Не варто забувати питання про питання безпеки, і на жаль, як і все, що підключено до мережі, API вразливі до експлуатації та зловживань [39]. Поширені атаки API включають наступне:

- атаки на основі автентифікації. Автентифікація є важливою частиною забезпечення надсилання та отримання викликів API із законних джерел. Однак зловмисники можуть обійти ці заходи для здійснення атак, перехопивши маркери автентифікації, викравши ключі API або використовуючи інші тактики для отримання конфіденційних облікових даних;
- експлойти вразливостей. Вразливості API – наприклад, порушена авторизація на рівні об’єкта, порушена автентифікація користувача, надмірний доступ до даних та інші з топ-10 безпеки API OWASP – стосуються недоліків в API, які можуть дозволити зловмисникам отримати доступ до них без дозволу. Використовуючи ці недоліки, зловмисники можуть здійснити витік даних або використовувати API для здійснення більш складних атак;
- DDoS-атаки. Зловмисники можуть переповнювати API об’ємним трафіком, намагаючись перервати (або повністю зупинити) службу, яку він надає.

Але всім добре відомі сервіси на подобі Cloudflare API Gateway допомагають пом’якшити ці атаки, забезпечуючи надійну автентифікацію, скануючи корисні дані на наявність конфіденційних даних, перевіряючи схеми API, а також виявляючи та запобігаючи зловживанням API [40, 41].

2.3 Механізм перевірки цілісності

Для запитів по API до Tuya є можливість використовувати хешування HMAC-SHA256.

Забезпечення способу перевірки цільності переданої інформації або захисту в ненадійному носії є першочерговою захищеністю у світі відкритих обчислень та комунікацій. Механізми, які забезпечують такі перевірки цілісності на основі секретного ключа, зазвичай називаються кодами автентифікації повідомлень (MAC). Як правило, коди автентифікації використовуються між двома сторонами, які мають спільний секретний ключ для перевірки повідомлення, що передається між ними. Розглянемо механізм на основі криптографічних хеш-функцій [42].

Код автентифікації повідомлення на основі хешування (HMAC) – це метод шифрування повідомлень, який використовує криптографічний ключ у поєднанні з хеш- функцією. Він надає серверу та клієнту особистий ключ, який відомий лише цьому конкретному серверу та клієнту, забезпечуючи більш безпечний засіб шифрування даних, ніж простий код автентифікації повідомлення (MAC).

HMAC – це техніка криптографічної автентифікації . Він використовує як криптографічну хеш-функцію, так і спільний секретний ключ для шифрування інформації та захисту її від несанкціонованого доступу.

Хеш-функція – це алгоритм або математична функція, яка перетворює повідомлення, що складається зі змінної кількості символів, у рядок із фіксованою кількістю символів. Вихідне значення відоме як дайджест повідомлення , хеш-значення або просто хеш . Секретний криптографічний ключ дозволяє користувачеві зробити зашифроване повідомлення доступним для читання після того, як воно було зашифровано за допомогою алгоритму [43].

У транзакції HMAC клієнт і сервер повинні узгодити секретний ключ. Це забезпечує спосіб декодування повідомлень, які повинні залишатися таємними, щоб підтримувати цілісність транзакції. Сторони також повинні вибрати та погодити хеш-функцію для своїх повідомлень.

HMAC можна використовувати для перевірки цілісності даних і автентифікації сторін, залучених до транзакції. Багато протоколів зв'язку та передачі використовують HMAC, включаючи HTTPS , SFTP і FTPS . Криптографічна хеш-функція в HMAC зазвичай SHA-1, SHA-256, MD5 або RIPEMD-128/160.

Розглянемо означення [44]:

$$\mathrm{HMAC}(\mathrm{K}, \mathrm{m}) = \mathrm{H}((\mathrm{K}` \oplus \mathrm{opad}) \parallel \mathrm{H}(\mathrm{K}` \oplus \mathrm{ipad}) \parallel \mathrm{m})) \quad (2.1)$$

$$\mathrm{K}` = \begin{cases} \mathrm{H}(\mathrm{K}) \\ \mathrm{K} \end{cases} \quad (2.2)$$

де H – криптографічна хеш-функція,

m – повідомлення до автентифікації,

K – секретний ключ,

K' – є ключем довжина якого дорівнює блоку отриманого з секретного ключа K,

$\parallel$ – позначення конкатенації

$\oplus$ – позначення побітової виключної диз'юнкції (або XOR)

opad – зовнішнє доповнення завдовжки з блок, що складається із повторень байта 0x5c

ipad – це внутрішнє доповнення завдовжки з блок, що складається із повторень байта 0x36

SHA-256 у свою ж чергу, це алгоритм сімейства криптографічних хеш-функцій. «256» у його назві означає довжину хешу, який він виробляє, який становить 256 біт. Цей алгоритм відіграє вирішальну роль у забезпеченні цілісності даних і безпеки в цифрових комунікаціях [45].

Роботу SHA-256 можна розділити на окремі етапи. Спочатку вхідні дані проходять процес, відомий як попередня обробка. Потім дані розбиваються на блоки, кожен з яких обробляється за допомогою ряду математичних функцій. Кінцевим результатом є унікальне хеш-значення, що представляє вихідні дані.

Безпека, яку пропонує SHA-256, є високою, перш за все завдяки його складним математичним операціям і унікальному хеш-значенню, яке він генерує. Однак, як і будь-який криптографічний алгоритм, він не повністю захищений від загроз безпеці. Потенціал хеш-колізій існує, хоча й надзвичайно низький [46].

Сфери використання цього алгоритму включають:

- перевірку даних;
- зберігання паролів;
- цифрові підписи;
- технологія «Блокчейн».

Серед переваг цього алгоритму можна виділити такі критерії як:

- цілісність даних. SHA-256 гарантує, що дані залишаються недоторканими та незмінними під час передачі. Будь-яка зміна вихідних даних, якою б незначною вона не була, призводить до зовсім іншого хеш-значення;
- безпека. Як криптографічна хеш-функція SHA-256 забезпечує високий рівень безпеки. Практично неможливо отримати вихідні дані з його хеш-значення;
- ефективність. Незважаючи на свою складність, SHA-256 є обчислювально ефективним. Він швидко генерує хеш-значення, сприяючи швидшій обробці даних.

А до недоліків можна віднести:

- негнучкість. Після того, як дані перетворені в хеш-значення, їх неможливо повернути або розшифрувати назад у вихідні дані;
- потенційна вразливість. Хоча це рідко, існує теоретична можливість, коли два різні вхідні дані створюють одне й те саме хеш-значення, ситуація, відома як хеш-колізія [47, 48].

2.4 Висновки по розділу

В цьому розділі було детально алгоритми роботи та можливості Telegram ботів, їх можливі варіанти, та безліч дрібниць які допоможуть якомога краще реалізувати функціонал кінцевого програмного продукту.

Також, було детально описано та розглянуто алгоритми роботи API, їх типи та архітектури, з яких, як найпопулярнішу, можна окремо виділити REST API. Підсумувавши, можна виділити важливість та незамінність такої технології, адже API є важливою частиною сучасної розробки програмного забезпечення. Вони дозволяють різним системам і програмам взаємодіяти одна з одною та ділитися функціями гнучким і ефективним способом. Вони використовуються в різноманітних контекстах, включаючи веб-розробку, мобільні програми та програми Інтернету речей (IoT) [49, 50].

Окремо можна виділити механізм цілісності у вигляді хеш алгоритмів, яких як SHA-256 – це потужна криптографічна хеш-функція, яка відіграє вирішальну роль у цифрових комунікаціях. Незважаючи на деякі недоліки, його переваги у підтримці цілісності даних, забезпеченні безпеки та ефективності обчислень роблять його безцінним інструментом у світі інформаційних технологій. Як і будь-яка технологія, вона вимагає обережного використання та регулярних оновлень для пом’якшення потенційної вразливості.

3 РЕАЛІЗАЦІЯ СЕРВЕРНОГО ДОДАТКУ

3.1 Створення та налаштування проєкту

Як раніше було визначено, одним з найкращих варіантів в якості середовища розробки буде використовуватись IntelliJ IDEA.

Створюю новий проєкт натискаючи File, далі New, потім Project. В новому вікні(рисунок 3.1) вводимо назву для проєкту, обираємо версію JDK, у випадку для проєкту це openjdk-21, 21.0.5. Також не буде зайвим обрати тип проєктного менеджера, в якості якого було обрано Maven. Після налаштувань проєкту натискаю кнопку Create.

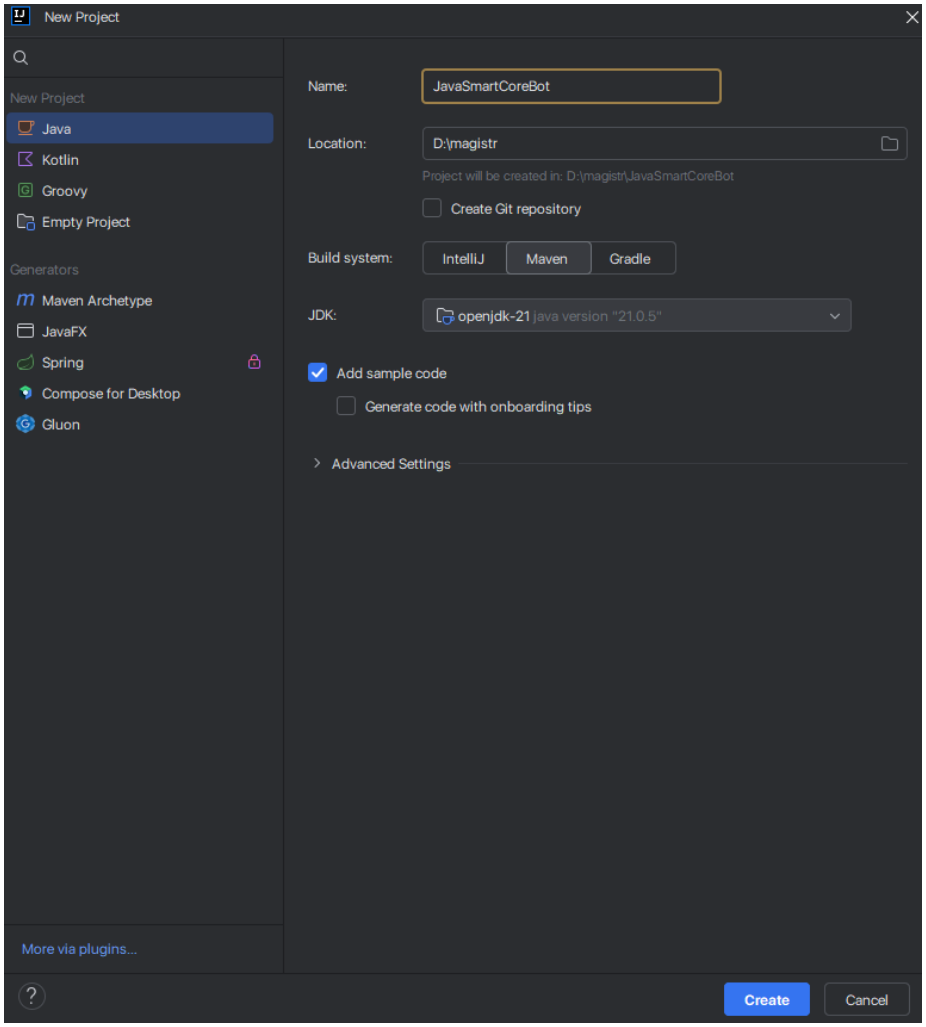

Рисунок 3.1 – Вікно створення нового проєкту

Після створення проєкту програма автоматично побудує його шаблонну структуру (рисунок 3.2). Ця структура виглядає однаково та на неї не впливає вибір іншого проєктного менеджера чи версії JDK. Також, при створенні була можливість додати базовий код, який буде виводити «Hello, World», але це зайве тому, що, так чи інакше доведеться це видалити.

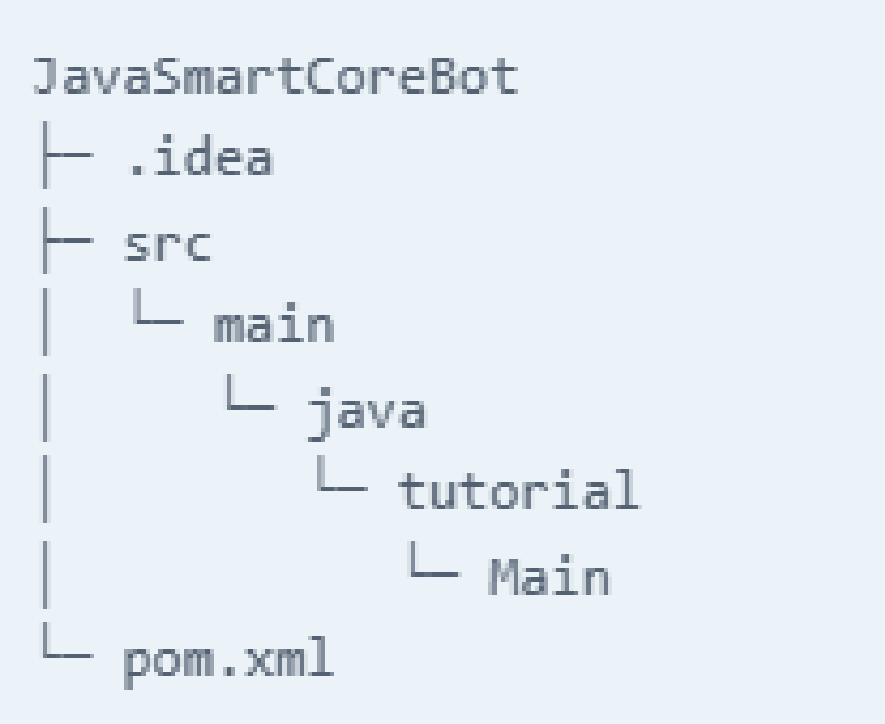

Рисунок 3.2 – Базова структура шаблонного проєкту

Якщо структура папок і файлів може бути цілком зрозумілою, тоді що таке pom.xml?

pom.xml (Project Object Model) – це XML-файл, який використовується в Maven, Файл pom.xml є основою кожного Maven-проєкту та містить конфігурацію, необхідну для його збірки, управління залежностями, плагінами та іншими аспектами проєкту(рисунок 3.3).

Структура pom.xml файлу зазвичай містить такі основні секції:

<modelVersion> – версія моделі pom;

<groupId> – унікальний ідентифікатор групи, до якої належить проєкт;

<artifactId> – ідентифікатор самого проєкту;

<version> – версія проєкту;

<dependencies> – блок, де зазначаються всі залежності проєкту;

<build> – налаштування збірки, зокрема плагіни;

<profiles> – профілі для різних середовищ.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>my-application</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <version>2.5.4</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Рисунок 3.3 – Приклад pom.xml файлу

Зручність Maven та інших проєктних менеджерів полягає у тому, як тільки користувачу потрібно підключити якусь сторонню бібліотеку чи залежність в свій проєкт, то користувач додає залежність pom.xml файлу проєкту, до прикладу залежність для роботи з json стрічками, яка точно буде не зайвою можна отримати з репозиторію проєктів [51], та помістити у файл залежностей відповідні рядки на з сайту(рисунок 3.4) між тегами <dependencies> та </dependencies>.

```
<!-- https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple -->
<dependency>
    <groupId>com.googlecode.json-simple</groupId>
    <artifactId>json-simple</artifactId>
    <version>1.1.1</version>
</dependency>
```

Рисунок 3.4 – Залежність бібліотеки для роботи json

3.2 Розробка API алгоритму та логіки підпису

API, це наступний, та один з найголовніших етапів розробки, так як воно і буде елементом, який зв’язує датчик з ботом. Одним з найкращих на мою думку способів перевіряти API запити є застосунок Postman. Наявний в мене датчик температури та вологості використовує Tuya API. В цієї компанії є дуже зручний сайт з детальною документацією, та навіть можливістю тестувати запити прямо в браузері, хоча, як на мене, Postman все ж зручніше.

Для початку роботи майже з будь яким API потрібно отримати до нього доступ. Це ключовий етап, оскільки більшість API сервісів потребують підтвердження особи та надають спеціальний ключ або токен для кожного користувача. Це допомагає платформам захищати дані та контролювати доступ до своїх ресурсів. Забігаючи наперед, у випадку з Tuya API, все буде складніше, але і безпечніше водночас.

Для того, щоб виконати будь який запит потрібно отримати access token, або токен доступу. Для його отримання потрібно раз в певний проміжок часу, який різниться для кожного API, виконати спеціальний запит, який містить Client ID, Client Secret та часовий штамп розміром в 13 символів. Також є можливість надсилати параметр Signature-Headers, це скрипт, який виконується перед запитом та створює підпис, який унікальний для кожного користувача та

використовує метод хеш шифрування HMAC-SHA256, що значно збільшує показник безпеки за буде запобігати несанкціонованому доступу в майбутньому.

Для отримання Client ID та Client Secret реєструюсь на платформі Tuya API Developer Platform та подаю заявку на безкоштовне використання API. Важливо при реєстрації вказати вірну часову зону, це буде враховуватись при створенні запитів, але цей параметр можна змінити в особистому кабінеті в будь який час.

Після того, як заявку погодять, на сторінці проєкту можна отримати Client ID, Client Secret та Project Code з Device ID за потреби (рисунок 3.5).

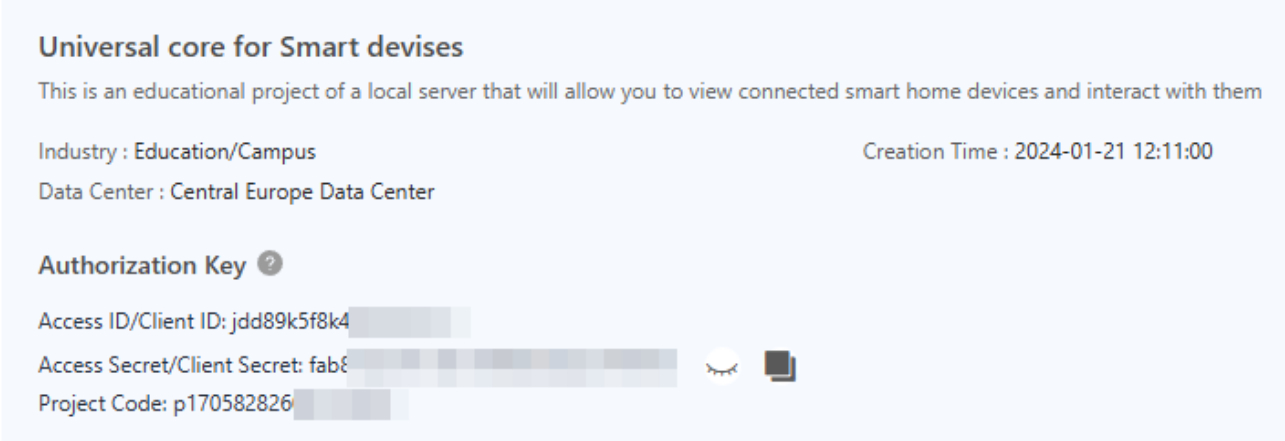

Рисунок 3.5 – Сторінка перегляду проєкту з даними авторизації

Створення коду для запиту не є складним завданням. Бібліотека okhttp3 є найзручнішою для виконання цієї задачі тому, що отримує дані запиту типом об’єкт, з яким зручно далі взаємодіяти. Додаю в pom.xml файл та імпортую використовуючи import okhttp3.*;.

Оголошую отримані раніше отримати Client ID і Client Secret в коді у вигляді статичних приватних змінних типу String (рисунок 3.6). Також, треба вказати так звану кінцеву точку. Кінцеві точки API можуть відрізнятися залежно від країни чи регіону. Варто вибрати кінцеву точку для країни або регіону, де розгорнуто ваші пристрої. Це робить ваші служби більш чуйними на запити API та запобігає несанкціонованому доступу з іншого регіону.

Рисунок 3.6 Оголошення статичних змінних в коді

Створюю новий метод execute (рисунок 3.7) для обробки об’єктів, де я починаю з перевірки того, чи є інформація про розробника в контейнері. Якщо даних немає, одразу викидаю виняток, повідомляючи, що розробник не ініціалізований. Потім будую URL-адресу, використовуючи базовий URL з налаштувань і додаючи шлях, який прийшов у параметрах. Далі, в залежності від методу HTTP, який я отримую (GET, POST, PUT чи DELETE), обираю відповідний спосіб запиту. Якщо жоден із цих методів не підходить, створюю виключення, зазначаючи, що дозволені тільки ці чотири методи.

Переходжу до заголовків: якщо додаткові заголовки не передано, створюю порожню карту. Потім збираю всі необхідні заголовки через спеціальний метод, який враховує accessToken, тіло запиту та налаштування заголовків. На цьому етапі також оновлюю URL, додаючи до нього параметри, відсортовані в потрібному порядку. Коли запит готовий, я надсилаю його та отримую відповідь.

Отримане тіло відповіді зчитую і, використовуючи gson, перетворюю його на об’єкт. Якщо під час виконання запиту виникає помилка, я перехоплюю її і викидаю у вигляді TuyaCloudSDKException, щоб сигналізувати про проблему.

Цей метод буде одним з найголовніших, бо через нього буде проходити все, що буде виводитись в боті.

```java
public static Object execute(String accessToken, String path, String method, String body, Map<String, String> customHeaders) {
    try {
        // Verify developer information
        if (MapUtils.isEmpty(Constant.CONTAINER)) {
            throw new TuyaCloudSDKException("Developer information is not initialized!");
        }

        String url = Constant.CONTAINER.get(Constant.ENDPOINT) + path;

        Request.Builder request;
        if ("GET".equals(method)) {
            request = getRequest(url);
        } else if ("POST".equals(method)) {
            request = postRequest(url, body);
        } else if ("PUT".equals(method)) {
            request = putRequest(url, body);
        } else if ("DELETE".equals(method)) {
            request = deleteRequest(url, body);
        } else {
            throw new TuyaCloudSDKException("Method only support GET, POST, PUT, DELETE");
        }
        if (customHeaders.isEmpty()) {
            customHeaders = new HashMap<>();
        }
        Headers headers = getHeader(accessToken, request.build(), body, customHeaders);
        request.headers(headers);
        request.url(Constant.CONTAINER.get(Constant.ENDPOINT) + getPathAndSortParam(url));

        Response response = doRequest(request.build());
        return gson.fromJson(response.body().string(), Object.class);
    } catch (Exception e) {
        throw new TuyaCloudSDKException(e.getMessage());
    }
}
```

Рисунок 3.7 – Код методу execute

Цей метод є ключовим для інтеграції з API, оскільки дозволяє безпечно виконувати HTTP-запити до сервера, враховуючи різні типи методів (GET, POST, PUT, DELETE). Він автоматизує побудову запиту, додає необхідні заголовки, включаючи токен доступу, та обробляє відповіді. Таким чином, метод забезпечує зручний і надійний спосіб взаємодії з API, спрощуючи обробку запитів і управління помилками.

Створюю та оптимізовую для кращої швидкодії шифрування та хешування клас Sha256Util. Цей клас надає утиліти для роботи з хешуванням і шифруванням за алгоритмом SHA-256 і HMAC-SHA256. Клас містить методи для створення

SHA-256 хешів із рядків та байтових масивів, а також для генерування HMAC-SHA256. Розглянемо детальніше кожен із методів.

```
public static String encryption(String str) throws Exception {
    return encryption(str.getBytes(StandardCharsets.UTF_8));
}
```

Рисунок 3.8 – Код методу encryption(String str)

Метод encryption(String str)(рисунок 3.8) приймає рядок str, перетворює його в масив байтів у форматі UTF-8, а потім викликає інший метод encryption(byte[] buf), щоб створити хеш. Він повертає SHA-256 хеш рядка у вигляді шістнадцяткового (hex) рядка.

Вхідний параметр: String str – рядок, який потрібно зашифрувати.

Вихідне значення: Повертає SHA-256 хеш рядка у шістнадцятковому форматі.

```
public static String encryption(byte[] buf) throws Exception {
    MessageDigest messageDigest;
    messageDigest = MessageDigest.getInstance( algorithm: "SHA-256");
    messageDigest.update(buf);
    return byte2Hex(messageDigest.digest());
}
```

Рисунок 3.9 – Код методу encryption(byte[] buf)

Метод encryption(byte[] buf)(рисунок 3.9) приймає байтовий масив buf, ініціалізує алгоритм хешування SHA-256 за допомогою класу MessageDigest, обчислює хеш, а потім повертає його у шістнадцятковому форматі, викликаючи метод byte2Hex.

Вхідний параметр: byte[] buf – масив байтів, який потрібно зашифрувати.

Вихідне значення: Повертає SHA-256 хеш у шістнадцятковому форматі.

Метод byte2Hex(byte[] bytes)(рисунок 3.10) перетворює байтовий масив bytes у шістнадцятковий рядок. Кожен байт масиву обробляється й додається до результату у форматі двозначного шістнадцяткового числа.

Вхідний параметр: byte[] bytes – масив байтів, який потрібно перетворити.

Вихідне значення: Шістнадцятковий рядок, що представляє байтовий масив.

```
private static String byte2Hex(byte[] bytes) {
    StringBuilder stringBuffer = new StringBuilder();
    String temp;
    for (byte aByte : bytes) {
        temp = Integer.toHexString( i: aByte & 0xFF);
        if (temp.length() == 1) {
            stringBuffer.append("0");
        }
        stringBuffer.append(temp);
    }
    return stringBuffer.toString();
}
```

Рисунок 3.10 – Код методу byte2Hex(byte[] bytes)

Метод sha256HMAC(String content, String secret)(рисунок 3.11) обчислює HMAC (Хеш-код повідомлення) для рядка content, використовуючи секретний ключ secret за допомогою алгоритму HmacSHA256. Він працює наступним чином:

1. Створює екземпляр Mac для HmacSHA256.
2. Ініціалізує ключ (SecretKey) на основі секретного рядка secret.
3. Обчислює HMAC для рядка content.
4. Перетворює результат в шістнадцятковий рядок.

Метод sha256HMAC приймає два вхідні параметри: текстовий рядок content, який містить дані, що потрібно захистити, та secret – секретний ключ,

також у вигляді рядка, який забезпечує унікальність HMAC. Використовуючи алгоритм HMAC-SHA256, метод обчислює HMAC (код автентифікації повідомлення), щоб забезпечити перевірку цілісності та автентичності даних. Вихідним результатом є шістнадцяткове представлення обчисленого HMAC, повернене як рядок у верхньому регістрі.

```
public static String sha256HMAC(String content, String secret) {
    Mac sha256HMAC = null;
    try {
        sha256HMAC = Mac.getInstance( algorithm: "HmacSHA256");
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    SecretKey secretKey = new SecretKeySpec(secret.getBytes(StandardCharsets.UTF_8),  algorithm: "Hmac
    try {
        sha256HMAC.init(secretKey);
    } catch (InvalidKeyException e) {
        e.printStackTrace();
    }
    byte[] digest = sha256HMAC.doFinal(content.getBytes(StandardCharsets.UTF_8));
    return new HexBinaryAdapter().marshal(digest).toUpperCase();
}
```

Рисунок 3.11 – Код методу sha256HMAC(String content, String secret)

Отже, клас Sha256Util призначений для шифрування даних з використанням алгоритмів SHA-256 та HMAC-SHA256. Він забезпечує функціонал для створення криптографічних хешів рядків та байтових масивів, а також обчислення HMAC для рядків, захищених секретним ключем. Метод encryption дозволяє отримувати SHA-256 хеші в шістнадцятковому форматі, а метод sha256HMAC генерує HMAC для переданого тексту, використовуючи секретний ключ. Клас також містить допоміжну функцію для перетворення байтових масивів у зручний для читання шістнадцятковий рядок.

3.3 Створення бота та його логіки

За своєю суттю можна розглядати Telegram Bot API як програмне забезпечення, яке надає JSON-кодовані відповіді на запити користувача.

З іншого боку, бот - це програма, програмне забезпечення або сценарій, який запитує API за допомогою запиту HTTPS і чекає на відповідь. Є можливість робити кілька типів запитів, а також багато різних об’єктів, які можна використовувати та отримувати як відповіді.

Оскільки браузер здатний надсилати запити HTTPS, можна використовувати його, щоб швидко випробувати API.

Для використання запитів необхідно зареєструвати бота та отримати токен, і це, як не дивно виконується у боті - @BotFather [52].

Токен – це рядок, який автентифікує бота (а не обліковий запис користувача, який його реєструє) в API бота. Кожен бот має унікальний токен, який також можна відкликати в будь-який час через @BotFather.

Створити бота можна за кілька хвилин, єдине що потрібно зробити користувачу, це придумати боту ім’я. Створення бота починається з команди /newbot, після введення якої, BotFather запропонує дати боту ім’я, та ідентифікатор який має закінчуватись на «bot» (рисунок 3.12).

Якщо все зроблено правильно, ви отримаєте токен який виглядає так: 4839574812:AAFD39kkdpWt3ywyRZergyOLMaJhac60qc.

Важливо підмітити, що токен потрібно зберігати в безпечному місці та ставитись до нього як до пароля і не повідомляти його нікому.

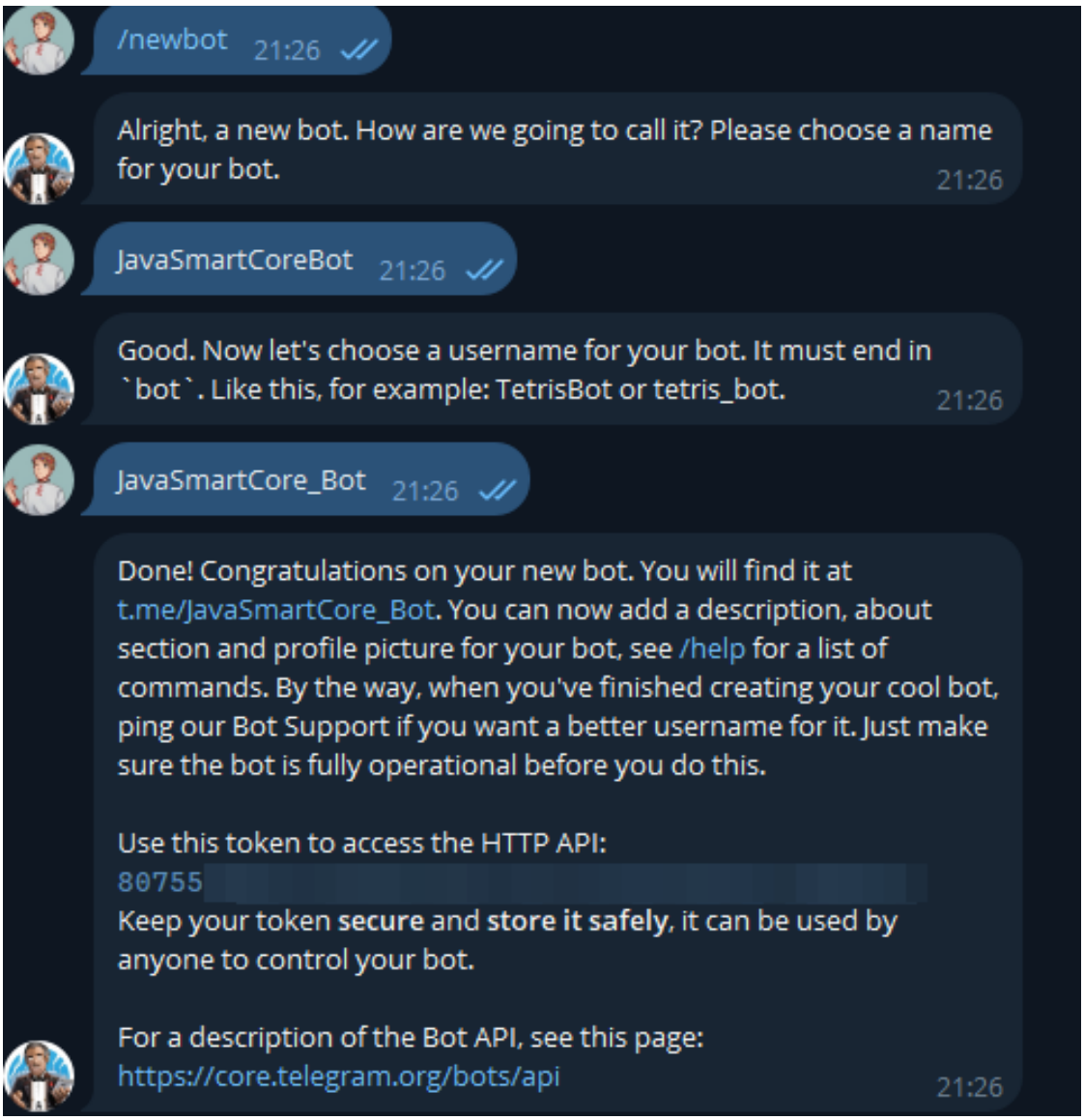

Рисунок 3.12 – Процес створення бота

Ось приклад запиту, який можна виконати через пошуковий рядок будь якого браузера:

https://api.telegram.org/<ТОКЕН _ВАШОГО_БОТА>/getMe

Якщо все виконано правильно, то браузер покаже сторінку з відповіддю, в якій може бути корисна інформація для початку розробки (рисунок 3.13).

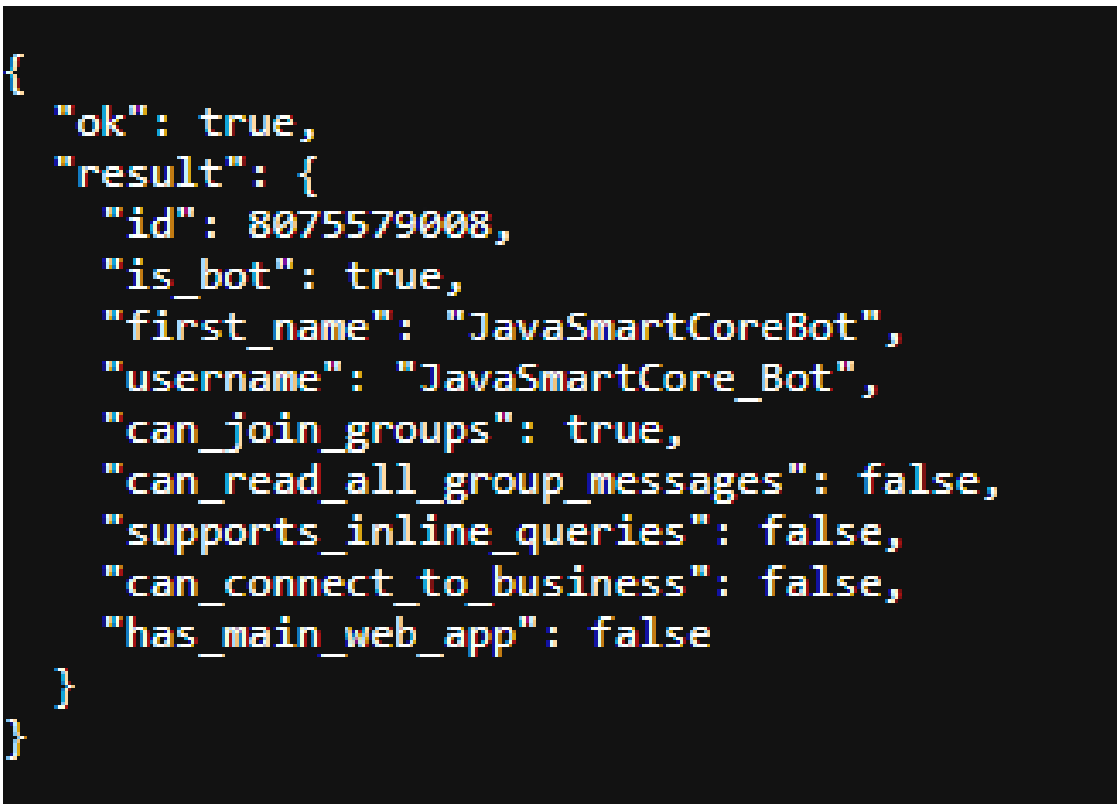

Рисунок 3.13 – Відповідь на тестовий запит в браузері

Теоретично можна взаємодіяти з API за допомогою базових запитів через браузер, або через інші спеціально створені інструменти, такі як cURL або Postman [53]. Хоча це може працювати для простих запитів, як у прикладі вище, це непрактично для великих програм і погано масштабується.

В корінній директорії проєкту створюю файл «TGBot.java» для коду бота, в якому буде базовий код, який є у кожного телеграм бота (рисунок 3.14).

Розберемо його методи детальніше. Метод getBotUsername завжди повинен повертати ім’я бота та його потрібно замінити з null на JavaSmartCoreBot, бо саме це ім’я було задано при створенні бота (рисунок 3.12). З назви метода getBotToken можна зрозуміти, що він має містити токен бота, для його подальшої авторизації. onUpdateReceived це найважливіший метод. Він буде викликатися автоматично щоразу, коли буде доступне нове оновлення від користувача та буде містити весь основний код бота. Після введення необхідних корективів, бот налаштований і готовий до роботи – час зареєструвати його в API та почати обробляти оновлення.

```
import org.telegram.telegrambots.bots.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.api.objects.Update;

public class TGBot extends TelegramLongPollingBot {

  @Override
  public String getBotUsername() {
      return null;
  }

  @Override
  public String getBotToken() {
      return null;
  }

  @Override
  public void onUpdateReceived(Update update) {}

}
```

Рисунок 3.14 – Базовий код телеграм бота

Для запуску бота необхідно провести його реєстрацію в класі Main. Для цього потрібно створити екземпляр класу TelegramBotsApi як telegramBotsApi, який, власне, і відповідає за запуск бота, та екземпляр створеного нами раніше класу TGBot як bot. Для подальшої зручності налагодження та використання бота додаю повідомлення про його успішний запуск та поміщаю весь написаний код класу main в блок try-catch, щоб провести базове тестування та перевірку на помилки, які будуть виводитись в консоль. Також, у випадку виникнення помилки в консоль буде виведено повідомлення про невдалий запуск бота (рисунок 3.15).

```java
public class Main {
    public static void main(String[] args) {
        try {
            TelegramBotsApi telegramBotsApi = new TelegramBotsApi(DefaultBotSession.class);

            TGBot bot = new TGBot();
            telegramBotsApi.registerBot(bot);

            System.out.println("Бот успішно запущено!");
        } catch (TelegramApiException e) {
            e.printStackTrace();
            System.out.println("Виникла помилка під час запуску бота!");
        }
    }
}
```

Рисунок 3.15 – Реєстрація бота в класі Main

Переходжу до файлу класу для написання логіки роботи бота. Метод onUpdateReceived (рисунок 3.16) є центральним у роботі Telegram-бота, оскільки він обробляє всі вхідні повідомлення та події, які бот отримує від користувачів. Кожного разу, коли користувач взаємодіє з ботом – будь то надсилання текстового повідомлення, натискання кнопки, використання команди або надсилання файлу – саме onUpdateReceived відповідає за прийняття й обробку цих оновлень. У ньому реалізується логіка реагування бота, зокрема аналіз вхідних даних, вибір відповіді, взаємодія з API, а також виклик додаткових методів чи функцій для складніших операцій.

Завдяки onUpdateReceived бот може надавати індивідуальні відповіді та реакції на дії користувачів, забезпечуючи ефективну комунікацію. Цей метод є основою для побудови функціональності бота, і його коректна робота безпосередньо впливає на стабільність і користувацький досвід.

Рисунок 3.16 – Метод onUpdateReceived

Також, в класі з ботом створюю методи для отримання даних з датчика та їх подальшому виведенні для користувача в зручному меню. І для такої цілі ідеально підійдуть кнопки типу keyboardMarkup, які, як було досліджено, знаходяться під полем введення тексту та завжди будуть у швидкій доступності без необхідності тягнутись за кнопкою дальні області екрану, у випадку, якщо користувач буде користуватись ботом з мобільного пристрою.

Для створення меню користуюсь switch-case конструкцією, це зручно тим, що можна налаштувати дію, яка буде опиратись на тому, яку кнопку натисне користувач. В цій конструкції розміщуватиму створені методи телеграм бота, такі як виведення температури, інформації про користувача, вологості, інформація про бота та вітальне повідомлення, яке надсилатиметься при введенні команди /start.

Розглянемо детальніше вищезгадані методи. Хорошим прикладом буде метод для отримання температури з датчика розумного дому, так як він

використовуватиме для своєї роботи методи класу APICore, в якому було розміщено всю взаємодію з Tuya API.

Створюю метод та називаю його sendTemperature (рисунок 3.17). Як і всі методи в цьому класі, він отримує значення chatId, та викликає метод класу APICore getTemperature. Температуру відповідно форматую та виводжу в повідомленні: «Температура: 24.2 °C».

```
private void sendTemperature(long chatId) {
    String stringTemp = (APICore.getTemperature());
    double temp = Double.parseDouble(stringTemp);
    temp /= 10;
    SendMessage message = new SendMessage();
    message.setChatId(String.valueOf(chatId));
    message.setText("Температура: " + temp + " °C");

    try {
        execute(message);
    } catch (TelegramApiException e) {
        e.printStackTrace();
    }
}
```

Рисунок 3.17 – Код методу sendTemperature

Також, не зайвим буде показувати відповідне повідомлення, якщо команда чи слово введене користувачем не було розпізнано. За це відповідає клас sendUnknownCommandMessage який знаходиться в default значенні конструкції switch-case та відправляє повідомлення про те, що команда не розпізнана.

3.4 Висновок по розділу

У цьому розділі було створено та описано процес розробки проєкту в середовищі Intellij IDEA, де покроково реалізовано телеграм-бот із набором функцій, спрямованих на забезпечення доступу користувача до різних даних і налаштувань. У рамках розробки телеграм-бота було додано низку методів, зокрема для виведення даних про температуру та вологість, які отримуються від датчика, а також для надання інформації про профіль користувача й загальні відомості про самого бота. Такі функції дозволяють користувачам легко отримувати доступ до релевантних даних через простий і зручний інтерфейс.

Розробка API, яка була зосереджена в класі APICore, включає реалізацію механізмів для інтеграції з Tuya API, що забезпечує зв'язок між ботом і пристроями розумного дому. Ключовим у цьому класі є оптимізований метод execute, який несе відповідальність за побудову, відправлення та обробку відповіді API. Метод execute приймає запити від телеграм-бота, формує необхідні параметри й заголовки, відправляє їх до Tuya API, а потім отримує дані у відповідь, що стосуються поточного стану сенсорів. Отримана інформація обробляється в боті й виводиться користувачам у зручному форматі за запитом. Така структура дозволяє гнучко й безпечно отримувати та обробляти дані від пристроїв розумного дому в рамках телеграм-бота, надаючи користувачам інтерактивний досвід взаємодії з IoT-пристроями.

ВИСНОВКИ

Розробка телеграм-бота для інтеграції з пристроями розумного дому передбачала детальне дослідження існуючих аналогічних рішень та виявлення їх сильних і слабких сторін. Це дозволило визначити основні проблеми, які потрібно вирішити в рамках проєкту, і розробити план дій для ефективної реалізації системи. Основною метою було створення функціонального коду телеграм-бота, що не лише забезпечує простий і зручний доступ до даних з різних датчиків, а й дозволяє масштабувати систему в майбутньому. Важливою вимогою було інтегрувати бот з API пристроїв розумного дому для того, щоб забезпечити зручне управління ними через Telegram без необхідності завантажувати додаткові додатки.

У ході виконання роботи було здійснено:

1. Аналіз та низка функціональних вимог до системи. Бот повинен надавати користувачам можливість безперешкодно взаємодіяти з пристроями розумного дому. Це включає перегляд актуальних даних з датчиків, таких як температура, вологість, а також стан роботи пристроїв. Крім того, була реалізована можливість додавати інших користувачів, які можуть отримувати доступ до даних, управляти пристроями та отримувати статуси їх роботи.

2. Одним з основних аспектів цієї розробки є використання API для взаємодії між ботом і пристроями. У проєкті було детально розглянуто механізми роботи з API, зокрема популярний тип REST API, який забезпечує зручний і гнучкий спосіб обміну даними між різними системами. Завдяки такій інтеграції, бот отримує необхідні дані від пристроїв розумного дому і надає їх користувачам у зручному вигляді через Telegram. Це дозволяє значно спростити користування пристроями розумного дому, знижуючи складність і потребу в додаткових мобільних додатках.

3. Крім того, у розробці були використані алгоритми хешування, зокрема SHA-256, для забезпечення цілісності та безпеки даних. Ці алгоритми

виконують важливу роль у захисті даних, що передаються між сервером і користувачем, забезпечуючи їх незмінність і захист від несанкціонованого доступу. SHA-256 є однією з найпотужніших криптографічних хеш-функцій, яка гарантує високу надійність і захищає від атак, зокрема підробки даних.

4. Важливим етапом розробки було створення та оптимізація основного методу execute, який відповідає за взаємодію з Tuya API. Цей метод виконує кілька важливих функцій: він формує запит до API, відправляє його на сервер, отримує відповідь і обробляє дані, що надійшли. Дані, отримані від Tuya API, містять важливу інформацію про стан пристроїв і сенсорів. Після цього отримана інформація передається в телеграм-бот, де вона форматуються для зручного відображення в інтерфейсі.

5. Цей процес взаємодії між ботом і API є основною частиною архітектури системи і дозволяє забезпечити гнучкість, масштабованість і безпеку в роботі з пристроями розумного дому. В результаті користувачі можуть отримувати дані про температуру, вологість, стан датчиків та іншу важливу інформацію без необхідності запускати додаткові програми. Всі ці можливості інтегровані безпосередньо в Telegram, що робить користування максимально зручним і доступним.

Отже, у кінцевому підсумку була розроблена потужна система, що дозволяє користувачам ефективно взаємодіяти з пристроями розумного дому через телеграм-бота. Інтеграція з Tuya API забезпечує отримання актуальних даних, а використання SHA-256 гарантує безпеку та цілісність переданих даних. Це дозволяє створити ефективну, зручну та безпечну платформу для управління пристроями розумного дому, що не вимагає додаткових програм і спрощує процес взаємодії з технологіями IoT.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дудас В. С., Кібербезпека для пристроїв розумного дому, X Всеукраїнська науково-практична конференція Інтелектуальні комп’ютерні системи та мережі, 2024р
2. Дудас В. С., Програмна система оповіщення студентів кафедри комп’ютерної інженерії на основі технології Push API, 2024р, с 14
3. ДСТУ 3008:2015 Національний стандарт України. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Введ. 01.07.2017. К.: ДП "УкрНДНЦ, 2016. 25 с
4. ДСТУ 8302:2015 Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Введ. 01.07.2016. К.: ДП «УкрНДНЦ», 2017. 16 с.
5. Amazon Alexa. URL: https://developer.amazon.com/en-GB/alexa
6. Melkov, D., & Paulikas, S. (2021, April). Security benefits and drawbacks of software-defined networking. // In 2021 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream) (pp. 1-4).
7. Google Home. URL: https://home.google.com/welcome/
8. Apple. вебсайт URL : https://developer.apple.com/documentation/homekit/
9. Samsung SmartThings. URL : https://www.samsung.com/ua/smartthings/
10. Tuya Smart. URL : https://www.tuya.com/
11. MOUHA, Radouan Ait Radouan Ait, et al. Internet of things (IoT). // Journal of Data Analysis and Information Processing, 2021, 9.02: 77. 2-7 p.
12. Laaychi, A., Tanana, M., & Lazaar, S. (2022). Security issues of the Web of Things: challenges and solutions. // In E3S Web of Conferences (Vol. 351, p. 113).
13. Schiller, E., Aidoo, A., Fuhrer, J., Stahl, J., Ziörjen, M., & Stiller, B. (2022). // Landscape of IoT security. Computer Science Review, 44, 100467. 23-26 p.

14. Qazi, F. (2022). Application Programming Interface (API) Security in Cloud Applications. // EAI Endorsed Transactions on Cloud Systems, 7(23), e1-e1. 5-6 p.

15. Mousavi, Z., Islam, C., Babar, M. A., Abuadbba, A., & Moore, K. (2023). Detecting Misuses of Security APIs: A Systematic Review. p. 13-22

16. Emeka, B. O., Hidaka, S., & Liu, S. (2023). A Practical Model Driven Approach for Designing Security Aware RESTful Web APIs Using SOFL. // IEICE TRANSACTIONS on Information and Systems, 106(5), p. 986-1000.

17. Dahiya, S. (2024). Cloud Security Essentials for Java Developers Protecting Data and Applications in a Connected World. // Advances in Computer Sciences, 7(1) p 1-2.

18. About Oracle Company. URL : https://www.oracle.com/uk/corporate/

19. What is Java technology and why do I need it? URL : https://www.java.com/en/download/help/whatis_java.html

20. IntelliJ IDEA overview. URL : https://www.jetbrains.com/help/idea/discover-intellij-idea.html

21. DataReportal. URL : https://datareportal.com/social-media-users

22. The Kyiv Independent URL : https://kyivindependent.com/telegram-in-ukraine/

23. Bots: An introduction for developers URL : https://core.telegram.org/bots

24. Дія – Державні послуги онлайн | Telegram contact. URL : https://t.me/Diia_help_bot

25. Jamshed, M. A., Ali, K., Abbasi, Q. H., Imran, M. A., & Ur-Rehman, M. (2022). Challenges, applications, and future of wireless sensors in Internet of Things: A review. // IEEE Sensors Journal, 22(6), 5482-5494.

26. Aivaliotis, V., Tsantikidou, K., & Sklavos, N. (2022). IoT-based multi-sensor healthcare architectures and a lightweight-based privacy scheme. Sensors, 22(11), 4269. p. 34-37

27. Kumar, S., & Deora, S. S. (2021, October). Security Challenges and Issues in IoT. // In 2021 6th International Conference on Signal Processing, Computing and Control (ISPCC) (pp. 171-175).

28. Hasan, N. B. B. (2024). Security Issues and Preventions in IoT Devices. Authorea Preprints. 43-50 p.

29. Garg, H., & Dave, M. (2019, April). Securing iot devices and securelyconnecting the dots using rest api and middleware. // In 2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU) (pp. 1-6).

30. Ozerchuk, I. (2021). Формування стійкого каналу передачі даних у мережі Інтернет. // Computer-integrated technologies: education, science, production, (43), p. 212-217.

31. Mousavi, Z., Islam, C., Babar, M. A., Abuadbba, A., & Moore, K. (2023). Detecting Misuses of Security APIs: A Systematic Review. p. 13-22

32. Peng, K., Li, M., Huang, H., Wang, C., Wan, S., & Choo, K. K. R. (2021). Security challenges and opportunities for smart contracts in Internet of Things: A survey. // IEEE Internet of Things Journal, 8(15), 12004-12020.

33. Bestari, D. N., & Wibowo, A. (2023). An IoT-Based Real-Time Weather Monitoring System Using Telegram Bot and Thingsboard Platform. // International Journal of Interactive Mobile Technologies, 17(6). p. 82

34. Петрова, Р. В., & Кузяєв, Д. Р. (2020). КІБЕРБЕЗПЕКА СИСТЕМИ «РОЗУМНИЙ ДІМ». \\ In The 4 th International scientific and practical conference—Priority directions of science and technology development‖(December 20-22, 2020) SPC—Sci-conf. com. ua‖, Kyiv, Ukraine. 2020. 1472 p. (p. 491).

35. Hafiz, N., Briliyant, O. C., Priambodo, D. F., Hasbi, M., & Siswanti, S. (2023). Remote Penetration Testing with Telegram Bot. //Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi), 7(3), 705-714.

36. Achar, S. (2022). Cloud computing security for multi-cloud service providers: Controls and techniques in our modern threat landscape. // International Journal of Computer and Systems Engineering, 16(9), 379-384.

37. Benmocha, G., Biham, E., & Perle, S. (2021). Unintended features of APIs: cryptanalysis of incremental HMAC. // In Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27 (pp. 301-325)

38. Таченко, І. А., Коробейнікова, Т. І., & Захарченко, С. М. (2021). Огляд сучасного стану питання в галузі оцінювання ризиків мережевої безпеки. // In Scientific Collection «InterConf»: with the Proceedings of the 5th International Scientific and Practical Conference «Theory and Practice of Science: Key Aspects», November 7-8, 2021: p 417-432.

39. Sindhwani, N., Anand, R., Niranjanamurthy, M., Verma, D. C., & Valentina, E. B. (2022). IoT based smart applications. // New York City: Springer International Publishing AG. p 14

40. Lesi, V., Jakovljevic, Z., & Pajic, M. (2021). Security analysis for distributed IoT-based industrial automation. // IEEE Transactions on Automation Science and Engineering, 19(4), 3093-3108. 19 p.

41. Mamdiwar, S. D., Shakruwala, Z., Chadha, U., Srinivasan, K., & Chang, C. Y. (2021). Recent advances on IoT-assisted wearable sensor systems for healthcare monitoring. Biosensors, 11(10), p. 372.

42. Ramdani, F. C., Rahmatulloh, A., & Shofa, R. N. (2023). Implementation of JSON Web Token on Authentication with HMAC SHA-256 Algorithm. // Sistemasi: Jurnal Sistem Informasi, 12(1), 194-205.

43. Rawal, B. S., Aleti, S. N., & Reddy, S. (2022). Optimization of SHA 256 with Finetune Pipeline and Parallel Processing with Split Techniques. // Mathematical Statistician and Engineering Applications, 71(3s), 460-472.

44. HMAC: Keyed-Hashing for Message Authentication. URL : https://datatracker.ietf.org/doc/html/rfc2104

45. Plotnikov, A., & Levina, A. (2023, July). Algorithm for simplifying the SHA-256 operations tree. // In 2023 IEEE International Conference on Cyber Security and Resilience (CSR) (pp. 592-597).

46. Melkov, D., & Paulikas, S. (2021, April). Security benefits and drawbacks of software-defined networking. // In 2021 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream) (pp. 1-4).

47. Lulla, G., Kumar, A., Pole, G., & Deshmukh, G. (2021, March). IoT based smart security and surveillance system. // In 2021 international conference on emerging smart computing and informatics (ESCI) (pp. 385-390).

48. Озерчук, І. М. (2023). БЕЗПЕКА ТА ПРИВАТНІСТЬ В СИСТЕМАХ ІНТЕРНЕТУ РЕЧЕЙ ДЛЯ СМАРТ-ЕНЕРГЕТИКИ. // Світ наукових досліджень. Випуск 23: матеріали Міжнародної мультидисциплінарної наукової інтернет-конференції (м. Тернопіль, Україна, м. Ополе, Польща, 24-25 жовтня 2023 р.)/за ред.: О. Патряк та ін. ГО “Наукова спільнота”, WSZIA w Opolu. Тернопіль: ФО-П Шпак ВБ 2023. 294 с. Збірник наукових публікацій укладено за матеріалами доповідей наукової, 287.

49. Baniya, P., Agrawal, A., Abid, K., Nath, J., Chaudhary, B. K., & Kunwar, B. (2024, February). The Internet of Things: Security Challenges and Opportunities. // In 2024 3rd International conference on Power Electronics and IoT Applications in Renewable Energy and its Control (PARC) (pp. 153-158).

50. Chantzis, F., Stais, I., Calderon, P., Deirmentzoglou, E., & Woods, B. (2021). Practical IoT hacking: the definitive guide to attacking the internet of things.// No Starch Press. Vol 44, 13 p.

51. MVN Repository. URL : https://mvnrepository.com/

52. BotFather | Telegram contact. URL : https://t.me/botfather

53. Postman. URL : https://www.postman.com/