ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері»

на здобуття освітнього ступеня вищої освіти бакалавр
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Аліна Почтовик
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Аліна Почтовик
група КНД–42

Керівник: Юрій КАТКОВ
науковий ступінь, вчене звання д.т.н., доцент

Рецензент:
науковий ступінь, вчене звання _____

(Ім'я, ПРІЗВИЩЕ)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Комп'ютерних наук

Ступінь вищої освіти бакалавр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук
_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Почтовик Аліні Романівні

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері
керівник кваліфікаційної роботи Юрій КАТКОВ д.т.н., доцент,

(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» 02.2024р. №36.

2. Строк подання кваліфікаційної роботи «31» травня 2024р.

3. Вихідні дані до кваліфікаційної роботи:

- Огляд існуючих систем моніторингу та управління продуктивністю серверів у корпоративних та інтернет-просторах.
- Науково-технічна література з питань, пов'язаних з аналізом та обробкою сповіщень щодо продуктивності приладів на сервері.

- Аналіз оптимального розміщення та конфігурації пристроїв на сервері з метою максимізації продуктивності та ефективного використання ресурсів.
- Розробка та впровадження програмного забезпечення для ефективного управління ресурсами сервера, включаючи автоматичне реагування на навантаження та проблеми продуктивності.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Аналіз передумов для розробки системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері.

4.2 Дослідження застосування Webhook у системах обробки та управління сповіщеннями щодо продуктивності приладів на сервері.

4.3 Розробка практичних рекомендацій для впровадження Webhook у системах обробки та управління сповіщеннями щодо продуктивності приладів на сервері.

5. Перелік графічного матеріалу: *презентація*

1) Тема дипломної роботи

2) Мета роботи. Об’єкт дослідження. Предмет дослідження. Постановка завдання дослідження.

3) Актуальність роботи.

4) Аналіз використання технології Webhook з метою отримання даних про прилади.

5) Аналіз способів згладжування (smoothing) та кореляції приладів на сервері для оптимального управління сповіщеннями щодо їх продуктивності.

6) Оцінка потенційних загроз та уразливості, що можуть виникнути під час розробки та експлуатації системи

7) Апробація результатів дослідження

8) Висновки

Дата видачі завдання 23.02.2024 р

КАЛЕНДАРНИЙ ПЛАН

/п	Назва етапів бакалаврської Роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	10.02-25.02.24	виконане
2	Дослідження поставленої задачі	26.02-20.03.24	виконане
3	Опис системи обробки та управління сповіщеннями (СОУС) щодо продуктивності приладів на сервері	23.02- 25.02.24	виконане
4	Методи застосування СОУС	25.02- 20.03.24	виконане
5	Програмне забезпечення СОУС	21.03-27.03.24	виконане
6	Аналіз передумов щодо впровадження СОУС	28.03-30.03.24	виконане
7	Розробка методів застосування СОУС	01.04-07.04.24	виконане
8	Розробка практичних рекомендацій щодо впровадження СОУС	08.04-17.04.24	виконане
9	Вступ, висновки, реферат	18.04-24.04.24	виконане
10	Розробка обов’язкових демонстраційних матеріалів	25.04-28.04.24	виконане
11	Доповідь та презентація	29.04-20.05.24	виконане
12	Попередній захист роботи	До 31.05	виконане

Здобувач вищої освіти _____ Аліна Почтовик

(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник

кваліфікаційної роботи _____ Юрій КАТКОВ

(підпис) *(Ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина бакалаврської роботи 98 с., 21 рис., 52 джерел

Мета роботи – Підвищення ефективності моніторингу продуктивності приладів на сервері через розробку та впровадження системи обробки та управління сповіщеннями.

Об'єкт дослідження – Процес моніторингу продуктивності приладів на сервері системою обробки та управління сповіщеннями, спрямований на підвищення ефективності управління продуктивністю та пошуку несправних приладів на сервері.

Предмет дослідження – Система обробки та управління сповіщеннями щодо продуктивності приладів на сервері.

Короткий зміст роботи:

Наукове завдання – Оцінка ефективності та доцільності розробки системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері з використанням сучасних технологій.

Методи дослідження – Використання аналізу сучасних систем управління продуктивністю приладів на сервері, впровадження механізму Webhook для отримання миттєвих сповіщень та аналіз кореляційних зв'язків між параметрами системи.

Актуальність теми – В умовах постійного зростання обсягів даних та комплексності інформаційних систем, ефективне управління продуктивністю приладів на сервері стає критично важливим завданням для забезпечення безперебійної роботи бізнес-процесів та підтримки інфраструктури.

Галузь використання – Розроблена система може бути використана в широкому спектрі галузей, включаючи IT-підприємства, дата-центри, телекомунікаційні компанії та інші сфери, де важливо забезпечення стабільної та ефективної роботи серверної інфраструктури.

КЛЮЧОВІ СЛОВА: СЕРВЕР, ПРИЛАДИ, СПОВІЩЕННЯ, WEBHOOK, JSON, PYTHON, ПРОГРАМНИЙ, ПРОДУКТ.

ABSTRACT

Text part of the bachelor's thesis 98 p.,21 fig., 52 sources

Objective of the work – Enhancing the efficiency of monitoring device performance on the server through the development and implementation of a processing and notification management system.

Research object – The process of monitoring device performance on the server by the processing and notification management system aimed at improving performance management and identifying faulty devices on the server.

Research subject – Processing and notification management system for device performance on the server.

Summary of the work:

Research task – Evaluation of the effectiveness and feasibility of developing a processing and notification management system for device performance on the server using modern technologies.

Research methods – Analysis of modern systems for managing device performance on the server, implementation of Webhook mechanism for receiving instant notifications, and analysis of correlation relationships between system parameters.

Relevance of the topic – In the conditions of constant growth of data volumes and complexity of information systems, effective management of device performance on the server becomes a critically important task to ensure uninterrupted operation of business processes and infrastructure support.

Field of application – The developed system can be used in a wide range of industries, including IT enterprises, data centers, telecommunications companies, and other areas where ensuring stable and efficient operation of server infrastructure is crucial.

KEYWORDS: SERVER, DEVICES, NOTIFICATION, WEBHOOK, JSON, PYTHON, SOFTWARE, PRODUCT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ…..…….. 13
ВСТУП……. 14
1 АНАЛІЗ ПЕРЕДУМОВ ДЛЯ РОЗРОБКИ СИСТЕМИ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМИ ЩОДО ПРОДУКТИВНОСТІ ПРИЛАДІВ НА СЕРВЕРІ ……. 16
1.1 Аналіз сучасного стану систем обробки та управління сповіщеннями щодо продуктивності приладів на сервері ……..… 16
1.2 Аналіз способів сповіщення пристроїв та серверів (модель клієнт/сервер) ……..……. 20
1.3 Аналіз можливих методів пріоритезації та фільтрації сповіщень на сервері ……. 30
1.4 Аналіз інструментів, які дозволяють програмам надавати інформацію в реальному часі іншим програмам при виникненні певної події…….. 32

2 ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ WEBHOOK У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМИ ЩОДО ПРОДУКТИВНОСТІ ПРИЛАДІВ НА СЕРВЕРІ……..……... 42
2.1 Дослідження різноманітних галузей застосовується Webhook для отримання даних з серверу ……..…….. 42
2.2 Дослідження характеристик Webhook, щоб надавати програмам інформацію в реальному часі при виникненні певної події …… 43
2.3 Дослідження умов використання Webhook для отримання даних з серверу ……..…….. 48

3 РОЗРОБКА ПРАКТИЧНИХ РЕКОМЕНДАЦІЙ ДЛЯ ВПРОВАДЖЕННЯ WEBHOOK У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМИ ЩОДО ПРОДУКТИВНОСТІ ПРИЛАДІВ НА СЕРВЕРІ …….……. 54
3.1 Рекомендації щодо архітектури збору та обробки Webhook …… 54
3.2 Розробка поетапно програмного коду Webhook …… 63
3.2.1 Реалізація webhook_https_get_push.ua …… 63
3.2.2 Реалізація server_collector.ua ……. …… 70
3.2.3 Реалізація restart_webhook.sh …….…… 72
3.2.4 Реалізація restart_server_collector.sh . …… 73

ВИСНОВКИ…….….. 75
ПЕРЕЛІК ПОСИЛАНЬ……..... 78
ДОДАТОК А Приклад програмного коду webhook_install.sh……... 79
ДОДАТОК Б Приклад програмного коду webhook_https_get_push.ua……. 82
ДОДАТОК В Приклад програмного коду server_collector.ua……..... 86
ДОДАТОК Г Приклад програмного коду restart_webhook.sh……... 88
ДОДАТОК Д Приклад програмного коду restart_server_collector.sh ……... 89
ДОДАТОК Е Приклад даних, отриманих від Webhook..... 90
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) …….…… 91

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	-	Application Programming Interface
CI/CD	-	Continuous Integration/Continuous Deployment
CMS	-	Content Management System
CRM	-	Customer Relationship Management
gRPC	-	Google Remote Procedure Call
GTM	-	Google Tag Manager
HTTP	-	HyperText Transfer Protocol
IPC	-	Inter-Process Communication
JSON	-	JavaScript Object Notation
OSI	-	Open Systems Interconnection
REST	-	Representational State Transfer
SaaS	-	Software as a service
SIP	-	Session Initiation Protocol
SMS	-	Short Message Service
SOAP	-	Simple Object Access Protocol
SQL	-	Structured Query Language
SSL	-	Secure Sockets Layer
SSRF	-	Server-Side Request Forgery.
TCP	-	Transmission Control Protocol
UAC	-	User Account Control
UDP	-	User Datagram Protocol
URL	-	Uniform Resource Locator
VoIP	-	Voice over Internet Protocol
XML	-	Extensible Markup Language
XSS	-	Cross-Site Scripting
SDL	-	Service Definition Language

ВСТУП

В атестаційній роботі бакалавра розглядається актуальна проблема розробки системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері за рахунок застосування механізму Webhook (веб-перехоплювача).

Обґрунтування вибору теми та її актуальність. В умовах постійного зростання обсягів даних та ускладнення архітектури інформаційних систем, ефективне управління продуктивністю приладів на сервері стає критично важливим завданням для забезпечення безперебійної роботи бізнес-процесів, коли важливе отримувати нові дані про зміни в цих бізнес-процесах з серверів на веб-браузер користувача. Одним з способів вирішення цього завдання є застосування механізму Webhook (веб-перехоплювача).

Цієй механізм може ініціювати оновлення в системі інвентаризації та повідомляти службу доставки про необхідність надсилання продукту, чи платіжна платформа може повідомляти продавців про статус. Визначення цього моменту, коли відбувається подія, яка запускає інтеграцію веб-перехоплювача, що має вплив на підвищення продуктивності сервера, може бути визначена автоматичне за рахунок спеціальних команд. Webhook має великі перспективи використання у багатьох випадках використання, про що свідчить цінність використання Webhook на платформах електронної комерції:

- події веб-перехоплювача використовуються для оновлення статусу замовлення,
- відстеження постачання та управління запасами.

Наприклад, коли клієнт розміщує замовлення у веб-додатку електронної комерції.

Ступінь вивченої проблеми. Ступінь вивченості проблеми для диплома на тему «Розробка системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері» є складним. Питанню впровадження технології Webhook присвячено багато досліджень та наукових праць [1, 2, 3, 4, 5]. Це пов'язано з тим, що зростання швидкості інтеграції додатків SaaS з додатками користувача призвела до включення технології Webhook (веб-перехоплювачів) у більшість програмних інфраструктури бізнес-процесів, коли необхдне отримання миттєвих сповіщень. Webhook найбільш поширені на платформах SaaS, таких як GitHub, Shopify, Stripe, Twilio та Slack, оскільки вони підтримують різні типи подій залежно від дій, що відбуваються всередині них.

Постановка завдання дослідження:

Застосування засобів Webhook у системах управління сповіщення для підвищення продуктивності сервера - це інноваційна галузь, яка створює можливість комп'ютерам і програмам по іншому виконувати завдання. Webhook та API (інтерфейси прикладного програмування) мають багато спільного. Особливо те, що вони використовуються програмами для спілкування. Однак основна відмінність полягає в тому, що API вимагає від вас запити даних, а Webhook вас повідомляє, коли відбувається певна подія. Webhook – це метод відправки даних (push-дія), керована подія, тоді як типові API запитують дані через запит GET на запит користувача або за заданим інтервалом часу. Якщо враховувати ці відмінності під час роботи серверу, то виникає актуальна та своєчасна задача визначення моменту, коли відбувається подія, яка запускає інтеграцію веб-перехоплювача, що має вплив на підвищення продуктивності сервера. Застосування засобів Webhook у системах обробки та управління сповіщенням для підвищення продуктивності сервера вимагає ретельного розгляду. Для вирішення цього завдання пропонується розглянути такі завдання:

- коли потрібно використовувати Webhook;
- які переваги дозволяє отримати застосування Webhook;

- які недоліки використання Webhook;
- як виконувати процес налаштування та використання Webhook;
- які можна запропонувати рекомендації щодо використання Webhook.

Специфіка джерельної бази. Дослідження джерельної бази відзначає, що джерела інформації неоднорідні та її треба перебрати за певними критеріями щодо їх різновиду, а саме: до важливості, якості, унікальності та ін., щоб відмітити проблеми та недоліки.

Об'єкт дослідження: Процес моніторингу продуктивності приладів на сервері системою обробки та управління сповіщеннями, спрямований на підвищення ефективності управління продуктивністю та пошуку несправних приладів на сервері.

Предмет дослідження: Система обробки та управління сповіщеннями щодо продуктивності приладів на сервері.

Мета роботи: Підвищення ефективності моніторингу продуктивності приладів на сервері через розробку та впровадження системи обробки та управління сповіщеннями.

Метод дослідження. Метод дослідження заснований на відомих методах системного підходу, а саме: аналітично-розрахунковий, теоретичного аналізу з використанням моделювання, реконструювання і типологізації на основі аналізу емпіричних даних.

Завдання роботи:

1. Провести аналіз передумов щодо впровадження моніторингу продуктивності приладів на сервері за рахунок механізму Webhook для отримання миттєвих сповіщень.

2. Дослідити методи застосування системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері на основі механізму Webhook для отримання миттєвих сповіщень..

3 Розробити практичні рекомендації щодо впровадження системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері на основі механізму Webhook для отримання миттєвих сповіщень.

Результати дослідження. Підвищення ефективності моніторингу продуктивності приладів на сервері через розробку та впровадження системи обробки та управління сповіщеннями.

Наукова новизна. Наукова новизна цього дослідження полягає в оцінці ефективності та доцільності розробки системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері з використанням сучасних технологій.

Практична значущість результатів дослідження - полягає в розробці практичних рекомендацій щодо застосування системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері з використанням сучасних технологій.

Апробація результатів надається в 1 статті, 3 тезисах в рецензованих наукових журналах і виданнях.

Матеріали були опубліковані:

В статті:

1. Почтовик А.Р. ЗАСТОСУВАННЯ ЗАСОБІВ WEBHOOK У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ СЕРВЕРА// Ю.І. Катков// Наукові записки Державного університету телекомунікацій №3, 2023, Подано до друку. https://journals.dut.edu.ua/index.php/sciencenotes/issue/archive

В тезисах:

1. Почтовик А.Р. // V Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез 18 квітня 2024. – К.: ДУІКТ, https://duikt.edu.ua/ua/news-1-570-12409-v-mizhnarodna-naukovo-tehnichna-konferenciya-suchasniy-stan-ta-perspektivi-rozvitku-iot_kafedra-inzhenerii-programnogo-zabezpechennya-avtomatizovanih-sistem

2. Почтовик А. Р. ЗАСТОСУВАННЯ ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ СЕРВЕРА// IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті» Збірник тез 15 травня 2024. – К.: ДУІКТ, https://duikt.edu.ua/ua/news-1-576-12601-v-duikt-vidbudetsya-vseukrainska-konferenciya-prisvyachena-shtuchnomu-intelektu-yak-vzyati-uchast_kafedra-shtuchnogo-intelektu
3. Почтовик А.Р. ПРОБЛЕМИ ЗАСТОСУВАННЯ ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ СЕРВЕРА // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ» Збірник тез 24 квітня 2024. – К.: ДУІКТ, https://duikt.edu.ua/ua/news-1-1009-12379-vseukrainska-naukovo-tehnichna-konferenciya-zastosuvannya-programnogo-zabezpechennya-v-informaciyno-komunikaciynih-tehnologiyah_kafedra-inzhenerii-programnogo-zabezpechennya

1 АНАЛІЗ ПЕРЕДУМОВ ДЛЯ РОЗРОБКИ СИСТЕМИ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМИ ЩОДО ПРОДУКТИВНОСТІ ПРИЛАДІВ НА СЕРВЕРІ

1.1 Аналіз сучасного стану систем обробки та управління сповіщеннями щодо продуктивності приладів на сервері

Системи обробки та управління сповіщенням серверів – це сукупність програмних компонентів, що використовуються для збору, обробки та подання даних. Вони називаються системами моніторингу. Приклади: виявлення шахрайства, системи оцінки загроз, системи виявлення вторгнень, національної оборони тощо

Основні елементи систем керування серверами.

1. Управління обладнанням. Моніторинг стану устаткування (процесор, пам'ять, та ін.) задля забезпечення найкращої роботи сервера.

2. Управління програмним забезпеченням. Передбачає регулярне оновлення ПЗ, вбудованого ПЗ (firmware) та операційних систем.

3. Управління безпекою. Заходи безпеки, такі як: антивірусне ПЗ, фаєрволи, управління доступом та шифрування даних.

4. Управління резервуванням. Резервне копіювання даних та план відновлення даних у разі аварій

Системи обробки та управління сповіщенням серверів є невід'ємною частиною інформування користувачів, їх залучення та актуальності актуальної інформації через TPC/IP мережу

Системи обробки та управління сповіщенням серверів є одним з інструментів оптимізації бізнес-процесів, які зазвичай залежать від ручного введення, що в кінцевому підсумку заощадить час та енергію.

Оптимізувати веб-процеси можуть веб-сервери додатків, які є робочими засобами, що забезпечують взаємодію видавця послуг з клієнтами під час їх обслуговування. Система обробки та управління сповіщенням допомагає забезпечити точну доставку важливої інформації потрібній людині з додатку в потрібний час, не залежний від дій співробітників.

Системи обробки та управління сповіщенням є невід'ємною частиною інформування користувачів, їх залучення та забезпечення актуальної інформації. Користуючись цієї системою можна негайно повідомити про підтвердження замовлення, замовлення доставки або зміни специфікацій. Такі системи відрізняються за складністю та масштабом залежно від конкретних вимог додатків або послуг, які вони обслуговують, а також від місця розташування. В нашому випадку розглядаються системи обробки та управління сповіщенням в веб-сервер додатків.

Веб-сервери додатків (Web Server), що являє собою засіб (комп'ютер чи програму), призначений для обробки запитів і передачі даних на інші (клієнтам) комп'ютери по локальній мережі або через Інтернет. Сервер додатків виконує прикладні програми.(Рис.1.1).

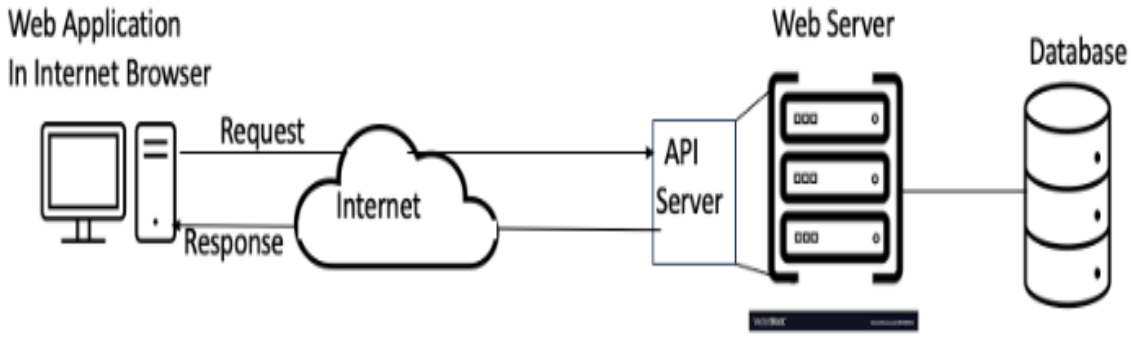


Рисунок 1.1 - Веб-сервери додатків (Web Server) [10]

Термін «Веб-сервери додатків» — це програмний компонент, який доставляє статичні дані, такі як зображення, файли та текст у відповідь на запити клієнтів. Сервер програм додає бізнес-логіку для обчислення відповіді веб-сервера. Тобто допомагає іншим комп'ютерам або пристроям отримувати та обмінюватися інформацією про функціонування додатку на кінцевих пристроях користувачів [6].

Веб-сервер – це система, яка здатна доставляти веб-контент кінцевим користувачам через Інтернет через веб-браузер.

Сервер додатків — це сервер, на якому розміщуються програмне забезпечення, яке надає бізнес-додаток через протокол зв'язку. У типовій веб-програмі сервер програм знаходиться за веб-серверами. Платформа сервера додатків є модель рівня обслуговування.

Програма, яка використовує HTTP для обслуговування файлів, що створюють веб-сторінки для користувачів у відповідь на їхні запити, надіслані HTTP-клієнтами комп'ютерів, називається веб-сервером. Якщо сервер доставляє XML-документ на інший пристрій, це може бути веб-сервер. Простіше кажучи, веб-сервер — це інтернет-сервер, який відповідає на запити HTTP для доставки контенту та послуг. Давайте розглянемо приклад: ви працюєте на своєму комп'ютері, переглядаєте веб-сторінки і з'являється повідомлення від вашого друга: «Я щойно прочитав відмінну статтю за наступною URL-адресою: https://www.milesweb.in/blog». .

Отже, вставте URL-адресу у свій браузер і натисніть клавішу Enter. От і все! Веб-сервер, на якому базується иеб-сайт, не має жодного значення, оскільки сторінка, яку ви переглядаєте, відразу ж з'являється на екрані вашого комп'ютера. Веб-сервер ніколи не відключається від Інтернету. Кожен із веб-серверів має унікальну адресу з чотирьох чисел від 0 до 255. Ці числа розділені точкою (.).

За допомогою веб-сервера хостинг-провайдери можуть керувати кількома доменами (користувачами) на одному сервері. А провайдер веб-

хостинга орендує місце на сервері або кластері серверів, щоб люди могли створити свою присутність в Інтернеті за допомогою веб-сайту.

Типи веб-серверів показані на рис.1.2 [51]

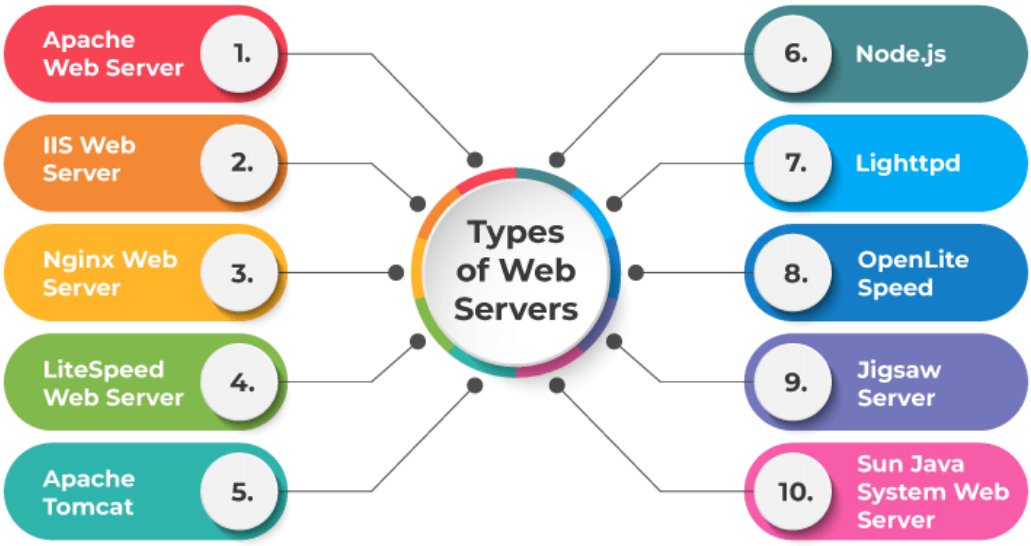


Рисунок 1.2 - Типи веб-серверів [51]

Принцип роботи веб-сервер додатку у тому, веб-сервер – це програмне та апаратне забезпечення, яке використовує HTTP. Основне завдання веб-сервера – відображати вміст веб-сайту за допомогою зберігання, обробки та доставки веб-сторінок користувачам.

Веб-сервери в основному використовуються для обробки та керування HTTP/HTTPS-запитами та відповідями клієнтської системи. Веб-сервер також може виконувати ряд інших функцій, таких як: Зберігати та захищати дані веб-сайту. Веб-сервер може зберігати та захищати важливі дані веб-сайту від неавторизованих користувачів.

Тобто вебсервери додатків зазвичай конфігуруються для обробки навантаження з обслуговування клієнтів відповідно до вимог отримання інформації від додатків.

HTTP (HyperText Transfer Protocol) — це протокол передачі гіпертекстових документів. HTTP забезпечує зв'язку між клієнтами та

серверами. HTTP – це протокол запиту-відповіді між клієнтом та сервером. Веб-перехоплювач забезпечує спрощений керований подіями зворотні зв'язок між двома інтерфейсами видавця-передплатника, що визначаються користувачем (Рис.1.3).

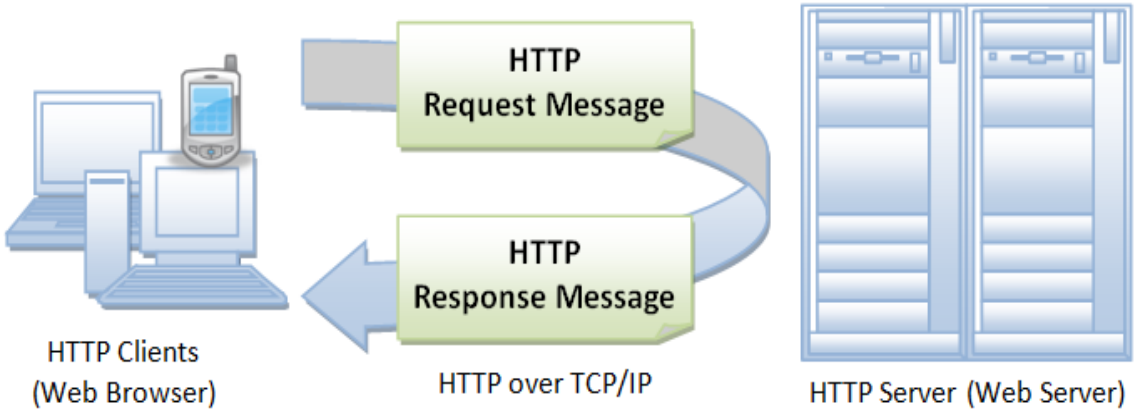


Рисунок 1.3 - HTTP (HyperText Transfer Protocol) [10]

HTTP, мабуть, найпопулярніший протокол додатків, що використовується в Інтернеті (або WEB).

HTTP - це асиметричний клієнт-серверний протокол запиту-відповіді, як показано на малюнку. HTTP-клієнт надсилає повідомлення запиту на сервер HTTP. Сервер, у свою чергу, повертає повідомлення у відповідь. Іншими словами, HTTP - це протокол запиту, клієнт отримує інформацію з сервера (замість того, щоб сервер відправляв інформацію клієнту).

HTTP – протокол без збереження стану. Іншими словами, поточний запит не знає, що було зроблено у попередніх запитах.

HTTP дозволяє узгоджувати тип і подання даних, що дозволяє створювати системи незалежно від даних, що передаються.

Веб-сервер додатків налаштовується таким чином, що він надає додаткову миттєву інформацію клієнтам, які використовують різноманітні додатки. Він обробляє навантаження відповідно до вимог отримання інформації від додатків. [52]

1.2 Аналіз способів сповіщення пристроїв та серверів (модель клієнт/сервер)

Способи сповіщення — це канали зв'язку, які використовуються для сповіщення людей про важливу інформацію, події або оновлення. В основному це стосується одностороннього зв’язку, коли сервер надсилає повідомлення, сповіщення або оновлення на пристрої чи інші сервери для передачі інформації. Основна увага зосереджена на доставці конкретних повідомлень одержувачам, часто без очікування негайної відповіді. До цих методів відносяться e-mail, SMS, push-сповіщення, прокручування тикерів та багато іншого.(Рис.1.4)

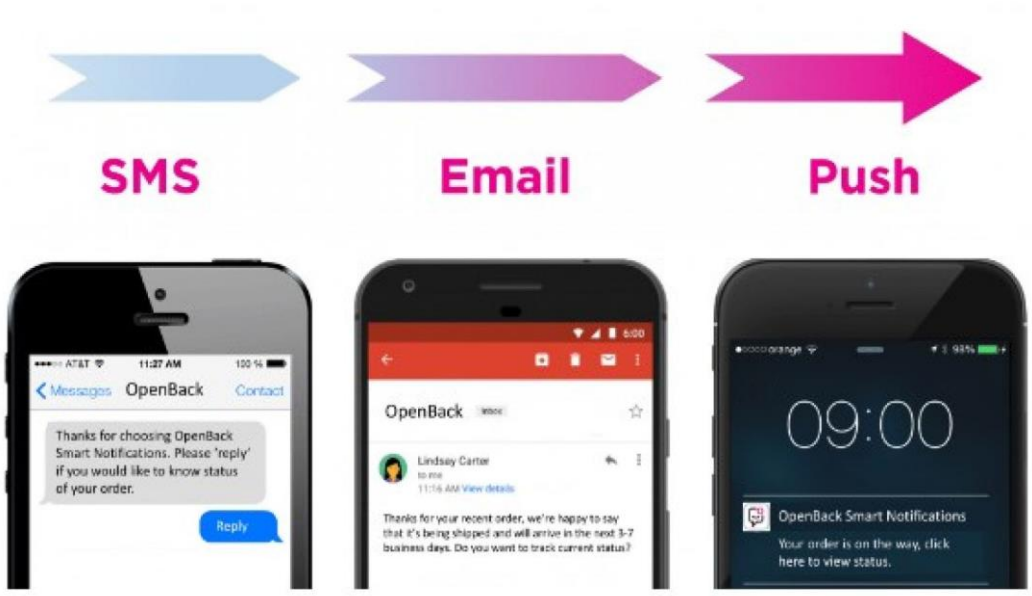


Рисунок 1.4 - Push - повідомлення [14]

Електронна пошта (e-mail) — це обмін повідомленнями, що зберігаються на комп'ютері, від одного користувача одному або декільком одержувачам через Інтернет. Електронна пошта – це швидкий, недорогий та доступний спосіб спілкування для бізнесу чи особистого використання. E-mail забезпечує можливість створення групових розсилок та автоматизації повідомлень за допомогою різноманітних програм і сервісів. Надійність і

безпека передачі інформації через електронну пошту також є однією з ключових переваг цього методу комунікації,

SMS означає "Служба коротких повідомлень" і широко відомий як текстові повідомлення. Це може надсилати між телефонами лише текстові повідомлення довжиною до 160 символів. SMS-повідомлення зазвичай доставляються миттєво, що робить їх ефективним способом для екстрених повідомлень. Незважаючи на обмеження в довжині, SMS залишається популярним.

Тикер (ticker) - це особливий вид повідомлення. Коли ви відправляєте повідомлення для перегляду, бігуча строчка прокручує повідомлення через рівні промежутки часу. Любий користувач, що має дозвіл на використання тикерів, може надсилати повідомлення в тикер (Рис.1.5).

Рисунок 1.5 - Ticker - повідомлення [14]

Цифровий рядок, що біжить, з прокруткою з'являється на екранах співробітників привабливим чином, і вони можуть клацнути заголовок або текст, який ви їм відправили, щоб отримати додаткову інформацію.

Push-повідомлення — це повідомлення, яке з'являється на мобільному пристрої, наприклад спортивний результат, запрошення на флеш-розпродаж або купон для скачування. Видавці програм можуть надсилати їх у будь-який час, оскільки для їх отримання користувачам не обов'язково перебувати в програмі або використовувати свої пристрої. (Рис.1.6)

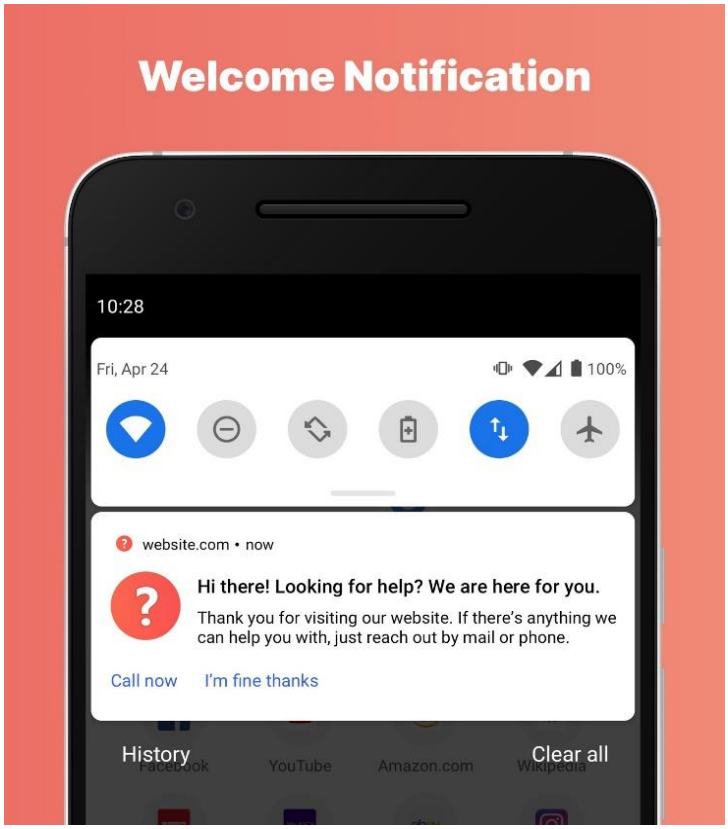


Рисунок 1.6 - Push - повідомлення [14]

Push-повідомлення використовуються наступним чином. Досягаючи екранів блокування користувачів, push-сповіщення забезпечують зручність для користувачів. Наприклад, він може отримати:

- Службові повідомлення, такі як звіти про пробки, погоду та сніг на лижах.
- Будь-яку інформацію про рейс, розклад та пересадки

push-сповіщення це спосіб безпосередньо звертатися до користувача. В результаті рейтинг кліків push-повідомлень може бути вдвічі вищим, ніж у електронних листів. Вони можуть нагадати користувачам використовувати програму, навіть якщо вона не відкрита. І вони можуть стимулювати такі дії, як:

- Просування продуктів чи пропозицій збільшення продажів
- Поліпшення якості обслуговування клієнтів
- Відправка квитанцій про транзакції відразу

• Залучення користувачів до інших маркетингових методів оповіщення в соціальних мережах.

Push - повідомлення – універсальний інструмент. Воні ефектівні у таких галузях:

- Електронна комерція;
- новинки сайтів та блогів;
- вебінари та інші онлайн-підрозділи;
- таксі та переїзди.

Працюють push-сповіщення наступним чином. Деякі з учасників відправлення push-повідомлень включають:

1. Служба push-повідомлень операційної системи (OSPNS). Кожна мобільна операційна система (ОС), включно з iOS, Android, Fire OS, Windows, має свій сервіс.

2. Видавець програми. Видавець програми включає в свою програму один або кілька OSPNS. Потім видавець завантажує програму в магазин додатків.

3. Клієнтський додаток. Ця програма для конкретної ОС, встановлена на пристрої користувача. Він отримує вхідні сповіщення.

Після обговорення традиційних методів, розглянемо більш просунуті методи, які забезпечують зв’язок у реальному часі між пристроями та серверами в рамках моделі клієнт/сервер. Методи Short Polling, Long Polling, Server Sent Events (SSE), Web Sockets, Сокети та API сокети пропонують динамічні способи обміну даними та полегшують інтерактивний досвід.

Сокети та API сокети використовуються для надсилання повідомлень по мережі. Вони забезпечують форму міжпроцесної взаємодії (IPC). Мережа може бути логічною, локальною для комп'ютера або фізично підключеною до зовнішньої мережі з власними підключеннями до інших мереж. Сокети дозволяють процесам обмінюватися даними незалежно від їхнього місця розташування, забезпечуючи прозорий механізм комунікації. Вони

підтримують як протоколи TCP, що забезпечують надійну передачу даних, так і UDP, який підходить для додатків, де важлива швидкість і низька затримка. Крім того, сокети можуть використовуватися для створення серверів і клієнтів, що дозволяє реалізовувати складні мережеві додатки, такі як веб-сервіси, онлайн-ігри та системи обміну повідомленнями в реальному часі.

Сокет – це одна з кінцевих точок взаємодії між двома програмами, що функціонують у мережі. Сокет асоціюється з номером порту, щоб рівень TCP міг визначити додаток, якому призначені дані надсилання.

Сокет прив'язаний до номера порту, щоб рівень TCP міг ідентифікувати додаток, якому призначені дані надсилання.(Рис.1.7)

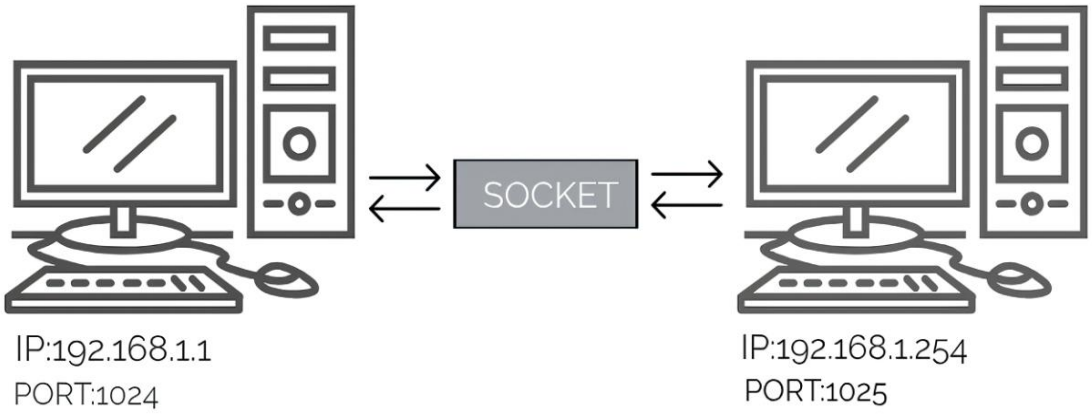


Рисунок 1.7 - Сокет [14]

Сокет дозволяє встановлювати з'єднання між клієнтами, які взаємодіють один з одним (наприклад, між клієнтом та серверною службою або між серверними службами). Черга повідомлень в основному діє як інтерфейс між різними серверними службами у системі, керованої повідомленнями. Сокети традиційно були рішенням, на основі якого побудовано більшість систем чату в реальному часі, забезпечуючи двонаправлений канал зв'язку між клієнтом та сервером. Це означає, що сервер може надсилати повідомлення клієнтам. Сокети зазвичай

використовуються для взаємодії клієнта та сервера. У типовій конфігурації системи сервер розміщується однією машині, а клієнти — на інших машинах. Клієнти підключаються до сервера, обмінюються інформацією, потім відключаються. Сокет має типовий потік подій.

WebSocket — це двонаправлений протокол, який може надсилати дані від клієнта до сервера або від сервера до клієнта, повторно використовуючи встановлений канал з'єднання. З'єднання підтримується, доки не буде розірвано клієнтом або сервером. Це дозволяє ефективно використовувати одне з'єднання для передачі даних в обидва напрямки без необхідності створення нового з'єднання для кожного обміну. Тому це особливо корисно для реалізації інтерактивних веб-додатків, де потрібна швидка взаємодія між клієнтом і сервером без зайвих затримок. WebSocket забезпечує значно меншу затримку у порівнянні з традиційними методами, такими як опитування (polling) або довгі запити (long polling). Крім того, протокол підтримує як текстові, так і двійкові дані, що робить його універсальним для різних типів додатків. Інтеграція WebSocket з існуючи ми веб-технологіями, такими як HTML5 і JavaScript, дозволяє легко реалізовувати його у сучасних веб-сайтах та додатках. Це робить WebSocket важливим інструментом для розробників, які прагнуть створювати більш інтерактивні та динамічні веб-рішення.

SSE(Server-Sent Events) — це технологія, яка забезпечує асинхронний зв’язок із потоком подій від сервера до клієнта через HTTP для веб-додатків. Сервер може надсилати ненаправлені повідомлення/події клієнту та може оновлювати клієнта асинхронно. Щойно клієнт встановлює з’єднання з сервером, сервер може надсилати оновлення або повідомлення клієнту без необхідності клієнту постійно надсилати запити. SSE підтримує майже кожен браузер.

Коротке опитування(Short Polling) — це метод, при якому клієнт періодично надсилає запити на сервер, щоб отримати оновлення. Сервер негайно відповідає з наявними даними. Якщо нових даних немає, клієнт надсилає запит знову після певного інтервалу. (Рис.1.8)

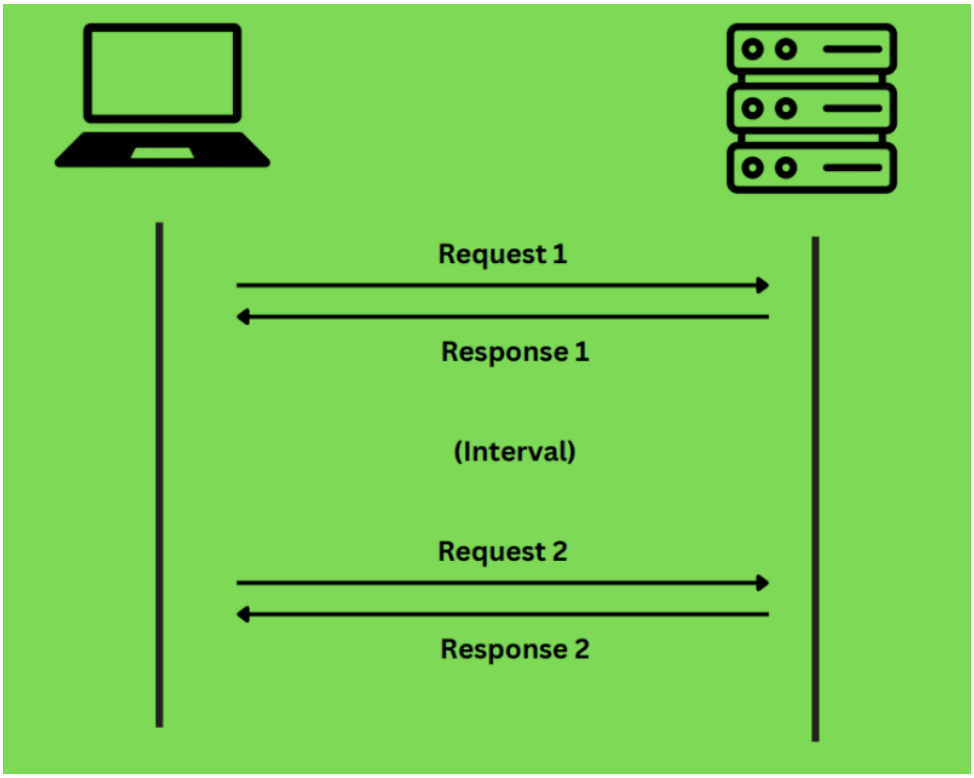


Рисунок 1.8 - Short Polling [14]

Приклад короткого опитування: Клієнт надсилає запит на сервер із запитом даних, але дані в цей час недоступні, і сервер відповідає порожньою відповіддю. Клієнт чекає 5 секунд, перш ніж відправити наступний запит. Це означає, що якщо за цей проміжок часу (5 секунд) будуть доступні будь-які дані, клієнт не буде повідомлений про це. Клієнт зможе отримати дані, коли надішле наступний запит.

Отже цей підхід простий у реалізації, але може призводити до збільшення навантаження на сервер та затримок в отриманні даних, оскільки клієнт змушений чекати між запитами.

Довге опитування(Long Polling) — це метод, коли клієнт відправляє запит, але сервер не відповідає негайно, якщо дані недоступні. Натомість сервер чекає певний час. Якщо протягом цього періоду відбувається якась подія або дані стають доступними, сервер надсилає їх клієнту. Якщо ж подій або нових даних немає, сервер відповідає порожньою відповіддю після закінчення часу очікування. Цей метод зменшує кількість запитів, надісланих клієнтом, що робить його більш ефективним, ніж коротке опитування. Незважаючи на те, що long polling є складнішим і споживає більше ресурсів сервера, він дозволяє сповіщати клієнта без затримки, що забезпечує кращий досвід роботи в реальному часі.

При виборі методу сповіщення для конкретної ситуації важливо враховувати як традиційні методи, так і сучасні технології зв'язку. Традиційні методи, такі як SMS, пошта та push-повідомлення, можуть бути найкращим варіантом для відправлення одноразових повідомлень або сповіщень, які не вимагають миттєвої реакції. Вони також можуть бути відмінним вибором для широкого кола аудиторії, оскільки є загальноприйнятими та доступними. З іншого боку, для ситуацій, де необхідна негайна реакція на події або потрібен актуальний стан даних, сучасні технології, такі як Short Polling, Long Polling, Server Sent Events (SSE) та Web Sockets, можуть бути кращим вибором. Вони забезпечують можливість миттєвого обміну даними між клієнтом та сервером у реальному часі, що дозволяє створювати інтерактивні та динамічні застосунки з високою швидкістю реакції.

Модель клієнт/сервер. Модель клієнт/сервер надає безліч мережевих переваг. Клієнт відправляє запит серверу і сервер відповідає, задовольняючи запит клієнта. У моделі клієнт/сервер нові клієнти та сервери можуть додаватися поступово в міру того, як все більше користувачів підключаються до мережі та збільшується попит на послуги. Інакше кажучи, модель клієнт/сервер легко розширюється і, отже, добре масштабується. Багато клієнтів можуть спільно використовувати ресурси, що надаються одним

сервером. Це позбавляє кожного клієнта необхідності мати власну «копію» ресурсів. Кожна інтернет-служба має свій власний набір клієнтів та серверів. Наприклад, у веб-домені браузери є клієнтами, а веб-сервери - серверами. (Рис.1.9)

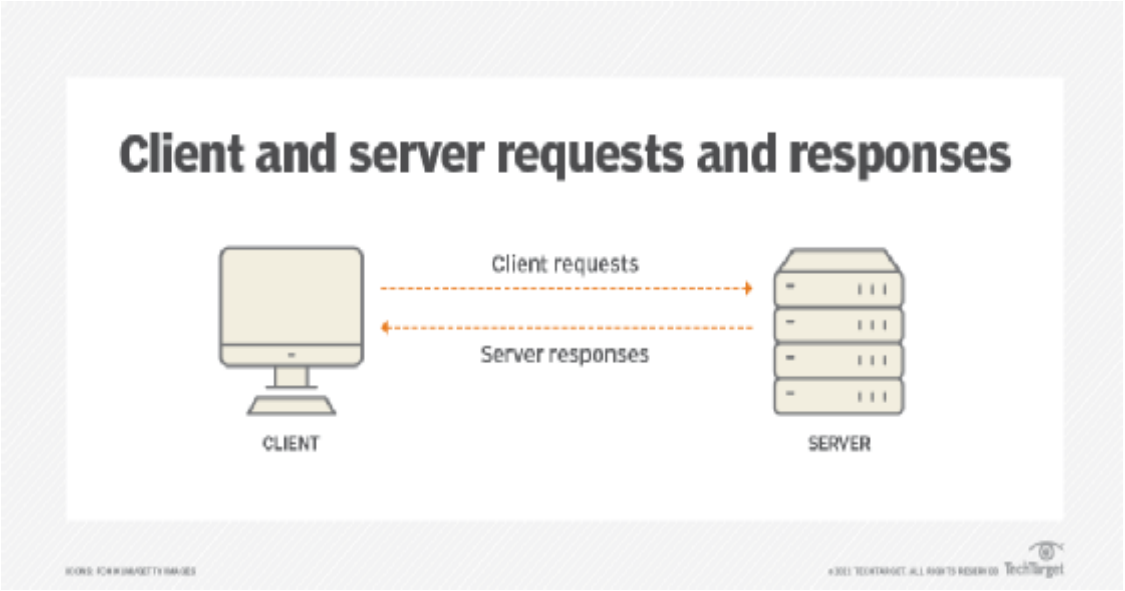


Рисунок 1.9 - Модель клієнт-сервер [14]

Робота моделі клієнт/сервер передбачає використання двох наборів програмного забезпечення:

- клієнтське програмне забезпечення, яке допомагає користувачу взаємодіяти з комп'ютерною системою. Наприклад, веб-браузер, який обговорюється пізніше в цьому документі.
- серверне програмне забезпечення для пошуку, наприклад, пошукові системи.

Сервер, як правило, є більш потужною комп'ютерною системою, ніж клієнт, і очікується, що він буде працювати з більш ніж одним клієнтом одночасно. Однією з основних переваг моделі клієнт/сервер є той факт, що якщо клієнти та сервери можуть розуміти дані, що передаються між ними, то не має значення, яка комп'ютерна система використовується для клієнта.

Типи повідомлень між серверами:

- запити, надіслані клієнтом для ініціювання дії на сервері,
- відповіді від сервера

Типи повідомлень між клієнтом та сервером. Клієнти зазвичай взаємодіють із серверами за допомогою TCP/IP.

TCP (Transmission Control Protocol) це протокол, орієнтований на з'єднання. Це означає, що протокол встановлює та підтримує з'єднання доти, доки прикладні програми на кожному кінці не завершать обмін повідомленнями.

Повідомлення клієнт-сервер передаються між собою наступним чином. Зв'язок є односпрямованим: клієнт надає запит серверу, і після обробки цього запиту сервер надає відповідь. Може існувати кілька клієнтських компонентів, які надсилають запити на сервер, який їх пасивно очікує.

1.3 Аналіз можливих методів пріоритезації та фільтрації сповіщень на сервері

До методів пріоритезації та фільтрації сповіщень на сервері відносяться:

Метод визначення критеріїв сповіщень. Як відомо, можна використовувати різні критерії, такі як серйозність(severity), джерело, тип, частота або час, щоб відфільтрувати інформацію та зменшити кількість попереджень. Наприклад, можна ігнорувати сповіщення, рівень серйозності яких нижчий за певний поріг або які походять від пристрою або програми, яка не є критичною для вашої мережі. Також можна використовувати регулярні вирази або шаблони для відповідності певному вмісту або формату сповіщення. Ще одним критерієм може бути час виникнення сповіщення. Можна зробити так, щоб сповіщення, які надходять у неробочий час або в періоди низької активності, мали пріоритет. Це дозволяє зосередитися на

вирішенні критичних проблем під час робочого дня, мінімізуючи відволікання. Крім того, частота сповіщень також є важливим фактором. Якщо певний тип сповіщень виникає дуже часто, це може свідчити про постійну проблему, яка потребує негайного вирішення, або ж про налаштування, що потребує коригування для зменшення шуму.

Метод призначення пріоритетів сповіщень. Призначення пріоритетів сповіщень, які становлять рівні важливості або терміновості, які ви призначаєте кожному сповіщенню на основі ваших критеріїв оповіщення. Можна використовувати різні методи визначення пріоритетності сповіщень:

- на основі оцінки або категоризації;
- на основі впливу, ймовірності та вартості;
- на основі ранжування їх від високого до низького;
- на основі різних груп. Наприклад: критичні, важливі, другорядні або інформаційні;
- на основі призначення різних дій або відповіді для кожної групи.

Метод налаштування повідомлень попередження. Налаштувати сповіщення про якісь дії передбачає визначення способів та каналів, які ви використовуєте для передачі сповіщень собі чи своїй команді. Можна використовувати різні відоми способи сповіщення, щоб доставляти сповіщення потрібним одержувачам у потрібний час. Надсилання занадто великої кількості або занадто частих повідомлень може призвести до стомлення сповіщень або перевантаження одержувачів. Для усунення цього недоліку треба застосовувати інструменти, які дозволяють програмам надавати інформацію в реальному часі іншим програмам при виникненні певної події. акі інструменти включають меседж-черги, системи публікації-підписки, або архітектури, орієнтовані на події.

Метод перегляду та уточнення налаштування оповіщень. Вам слід переглянути та уточнити налаштування сповіщень, тобто параметри та конфігурації, які ви використовуєте для керування системою сповіщень. Вам також слід регулярно відстежувати та аналізувати ефективність сповіщень та

зворотний зв'язок, такі як кількість, якість та точність сповіщень, а також час відгуку та ефективність ваших дій. Крім того, вам потрібно оновлювати та коригувати налаштування сповіщень у міру зміни умов або вимог вашої мережі. Наприклад, при додаванні або видаленні пристроїв або програм, або зміні критеріїв або пріоритетів оповіщень. Таким чином ви зможете оптимізувати свою систему сповіщень та переконатися, що вона відповідає вашим потребам та цілям моніторингу мережі. Також важливо враховувати зміни в бізнес-процесах або зміни в робочих графіках, які можуть вимагати коригування налаштувань сповіщень. Наприклад, збільшення робочого часу або впровадження нових технологій може вимагати налаштування додаткових сповіщень або зміни пріоритетів існуючих. Використання історичних даних про сповіщення може допомогти виявити тенденції та оптимізувати налаштування для покращення продуктивності.

1.4 Аналіз інструментів, які дозволяють програмам надавати інформацію в реальному часі іншим програмам при виникненні певної події.

Для усунення недоліків застосування сповіщень, що перераховані вище треба застосовувати інструменти, які дозволяють програмам надавати інформацію в реальному часі іншим програмам при виникненні певної події. Розглянемо їх.

Традиційне для роботи системи обробки та управління сповіщенням серверів застосовується інтерфейс API (Application Programming Interface) служби push-повідомлень.

API — це інтерфейси програмування застосунків, які містять набір функцій або методів для додавання, редагування та отримання даних. API діє як програмний посередник, який дозволяє двом програмам взаємодіяти

один з одним. Тобто це шлях, який встановлює зв'язок між клієнтом і сервером. API – це доступний спосіб отримання та обміну даними всередині та між організаціями. Це робить його потужним інструментом для інтеграції різних систем та платформ. (Рис.1.10).

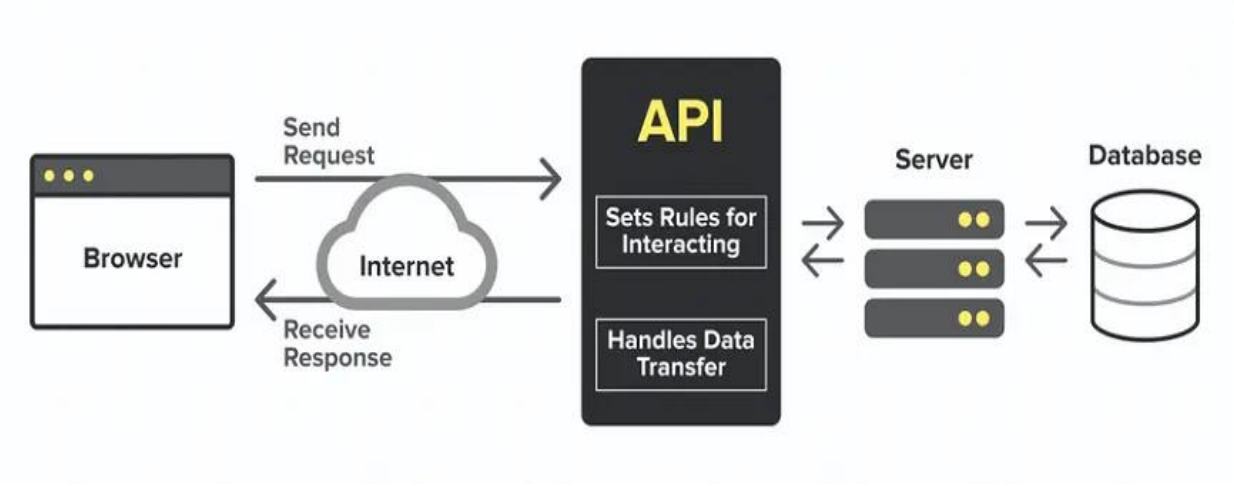


Рисунок 1.10 - API (Application Programming Interface) [10]

Використання API вимагає виконувати всю роботу щодо обробки та управління сповіщенням самостійно. Такі інтерфейси можна розглядати як взаємодію видавця послуг один з одним, використовуючи запити та відповіді, або з клієнтами, під час їх обслуговування. API-інтерфейси, часто використовуються для інтеграції різних систем, і помилка в одному API-інтерфейсі може викликати ефект доміно, що буде впливати на численні системи та служби. Ці недоліки можуть виникнути через відсутність заходів безпеки, неможливість перевірки запитів користувачів або неправильну обробку даних.

Іншим важливим аспектом є моніторинг та логування API-запитів, що дозволяє швидко виявляти та реагувати на проблеми. Це включає в себе відстеження помилок, перевантажень системи та аномальної активності, що може свідчити про можливі загрози або технічні збої. Крім того, для ефективного управління сповіщеннями варто використовувати різні стратегії резервування та відновлення. Це може включати налаштування резервних

API-інтерфейсів, які зможуть взяти на себе функції основного API у випадку його збою, а також плани аварійного відновлення, що забезпечать безперебійну роботу системи навіть у випадку критичних інцидентів.

Архітектурні стилі API- інтерфейсів – це REST, SOAP, GraphQL, WebSocket, gRPC, Falcor API, Webhook

REST API (Representational State Transfer) може задовольнити типові бізнес-потреби, оскільки для виконання завдання потрібно менше коду, а структура та логіка менш жорсткі, ніж SOAP API. REST API простий у використанні, працює швидше, ніж звичайні веб-сервіси, і може повертати результати у різних форматах даних. REST API масштабуються завдяки здатності кешувати дані, що знижує навантаження на сервер, а також можуть використовувати шифрування SSL для передачі даних, що усуває загрозу компрометації при передачі.(Рис.1.11)

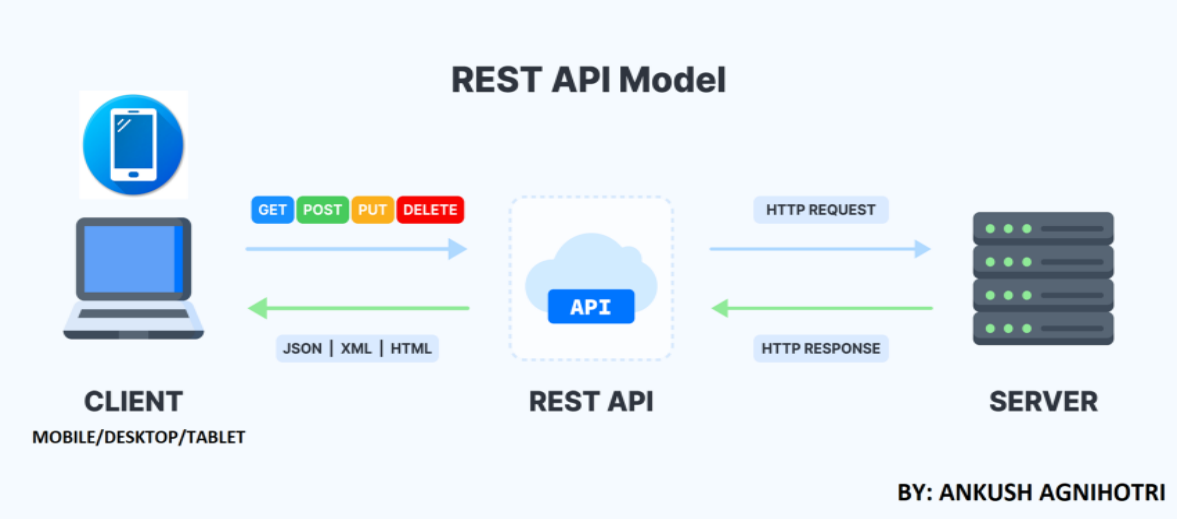


Рисунок 1.11 - Архітектура REST побудови розподілених гіпермедіа систем [13]

SOAP API(Simple Object Access Protocol)- це специфікація протоколу обміну повідомленнями для обміну структурованою інформацією під час реалізації веб-сервісів у комп'ютерних мережах. Він заснований на XML.

Специфіка SOAP полягає у форматі обміну даними. SOAP дозволяє програмам, які працюють на різних операційних системах, спілкуватися за допомогою різних технологій і мов програмування. Клієнт може використовувати SOAP API для створення, отримання, оновлення або видалення записів, таких як паролі, облікові записи, потенційні клієнти та спеціальні об’єкти, із сервера. (Рис.1.12)

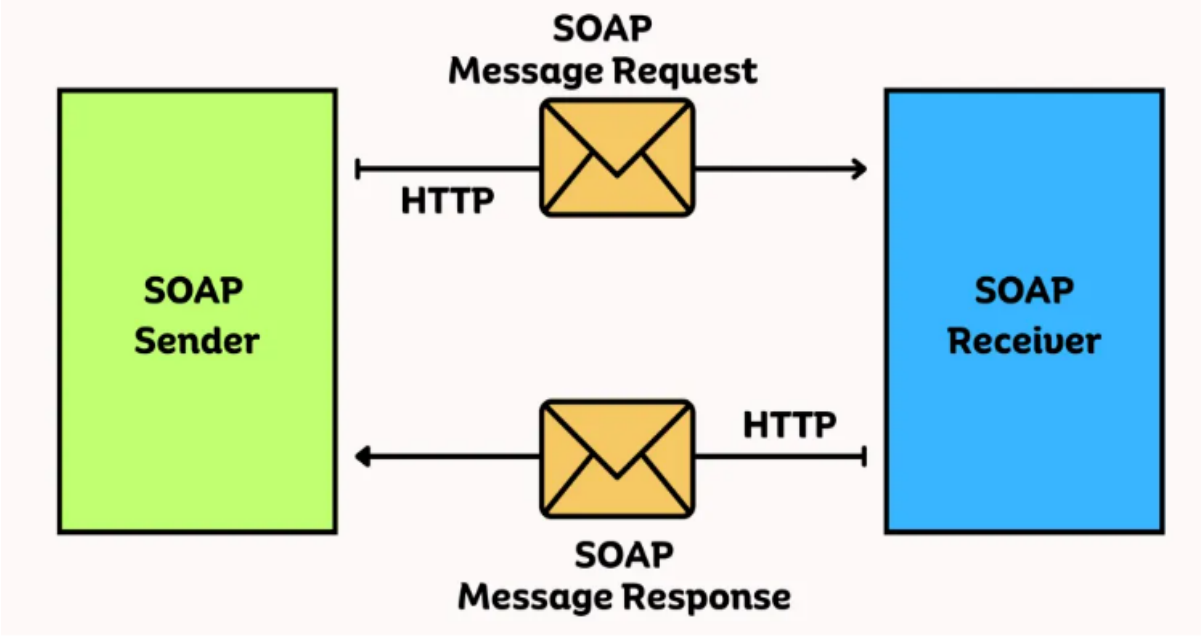


Рисунок 1.12 - Архітектура SOAP [13]

В даному випадку це завжди SOAP-XML, що представляє собою XML, який включає:

конверт (Envelope) - кореневий елемент, що визначає повідомлення та простір імен, що використовується в документі;

заголовок (Header) - включає в себе атрибути повідомлення, наприклад, інформацію про безпеку або інформацію про мережну маршрутизацію,

тіло (Body) – включає повідомлення, яким обмінюються додатки,

Fault – це елемент, який надає інформацію про помилки під час обробки повідомлень.

SOAP API також надає безпеку за допомогою різноманітних механізмів аутентифікації і шифрування даних, що забезпечує конфіденційність та

цілісність інформації. Однією з ключових переваг використання SOAP API є його можливість взаємодіяти з іншими веб-сервісами, які підтримують стандарт SOAP, без необхідності змінювати код програми. SOAP також підтримує асинхронний обмін повідомленнями, що дозволяє розробникам створювати складні та ефективні системи взаємодії. SOAP та REST – це два різні підходи до проектування API.

Підхід SOAP добре структурований, це формат базується на XML. Він здебільшого використовується в інтеграціях на рівні підприємства, де надійність, безпека та надійність є найважливішими. Також SOAP може бути більш корисним у сценаріях, коли серію запитів потрібно виконати в певній послідовності, і сервер повинен запам’ятовувати стан клієнта між запитами. З іншого боку REST більш гнучкий і дозволяє програмам обмінюватися даними в декількох форматах. REST є кращим вибором у сценаріях з легким зв’язком без збереження стану через HTTP. API RESTful часто віддають перевагу для загальнодоступних API, мобільних додатків і сценаріїв, де критично важливі продуктивність і масштабованість.

GraphQL— це мова запитів і середовище виконання на стороні сервера для інтерфейсів прикладного програмування, які пріоритетно надають клієнтам саме ті дані, які вони запитують. GraphQL розроблено, щоб зробити API швидкими, гнучкими та зручними для розробників. Як альтернатива REST, GraphQL дозволяє розробникам створювати запити, які отримують дані з кількох джерел даних за один виклик API. Також цей метод підтримує оновлення даних у реальному часі за допомогою підписок, дозволяючи клієнтам отримувати оновлення від сервера, як тільки вони стають доступними, замість того, щоб повторювати запити на оновлення.

Крім того, GraphQL надає розробникам API гнучкість для додавання або виключення полів, не впливаючи на існуючі запити. Розробники можуть створювати API будь-якими методами, які вони віддають перевагу. (Рис.1.13).

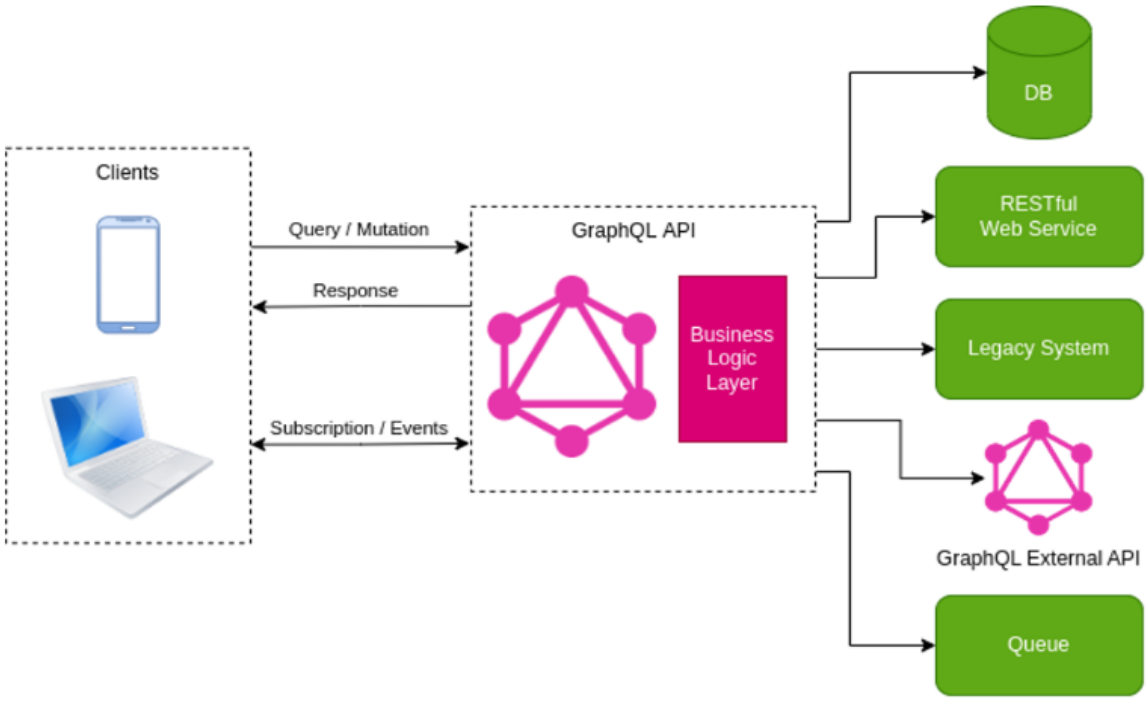


Рисунок 1.13 - Архітектура GraphQL [13]

GraphQL забезпечує декларативну вибірку даних, коли клієнт може вказати, які дані йому потрібні з API. Замість кількох кінцевих точок, які повертають окремі дані, сервер GraphQL надає одну кінцеву точку та відповідає саме тими даними, які запросив клієнт. Оскільки сервер GraphQL може отримувати дані з окремих джерел і представляти дані в єдиному графі, він не прив'язаний до будь-якої конкретної бази даних або механізму зберігання. Він може використовувати один виклик API, який може повертати дані з кількох джерел даних, а інший — API з відкритим вихідним кодом gRPC , який дозволяє програмі передавати дані функції іншій програмі в Інтернеті.

WebSocket API – це ще один тип API. WebSocket API – це передова технологія, що дозволяє відкрити сеанс двостороннього інтерактивного зв'язку між браузером користувача та сервером. WebSocket забезпечує постійний двосторонній канал зв'язку з малою затримкою. За допомогою цього API клієнт може надіслати повідомлення на сервер, і служба відповість

повідомленням клієнту. Служба на сервері може надсилати інформацію назад клієнту без запиту з боку клієнта (Рис.1.14)

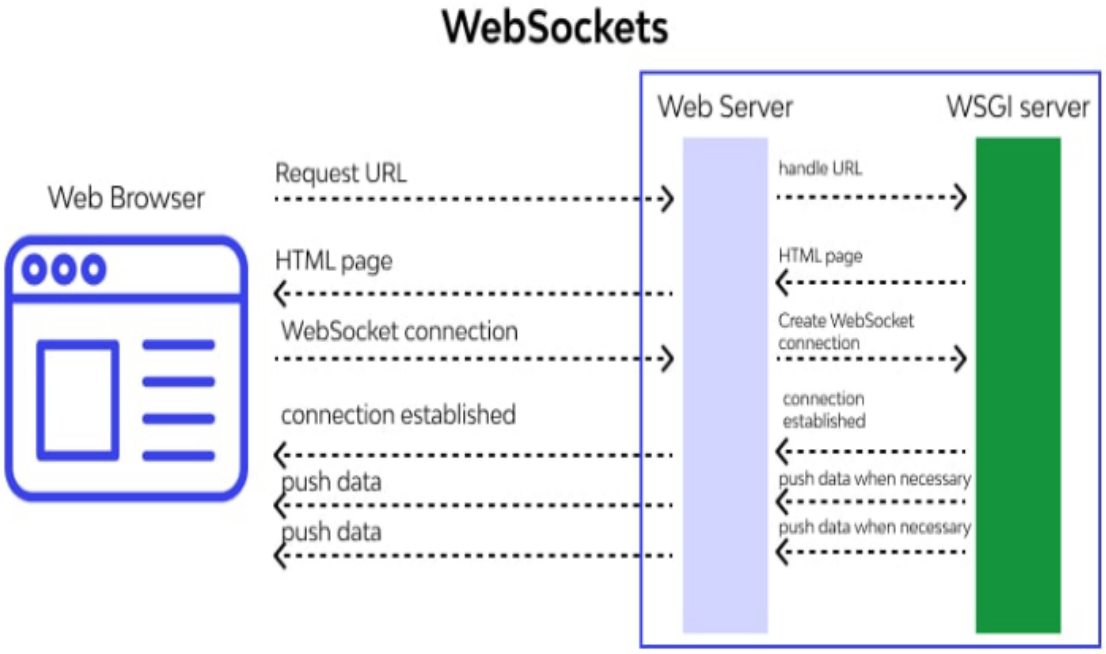


Рисунок 1.14 - Архітектура WebSocket API [13]

WebSockets використовуються для взаємодії в реальному часі між клієнтами та серверами на основі подій. Вони особливо корисні для створення програмних додатків, що вимагають миттєвих оновлень, таких як чат в реальному часі, обмін повідомленнями та багатокористувацькі ігри.

gRPC (Google Remote Procedure Call) - це відкритий фреймворк для реалізації взаємодії клієнт-сервер у розподілених системах. Основна ідея gRPC полягає в використанні віддалених процедурних викликів (RPC) для спілкування між різними частинами системи. gRPC використовує простий у використанні і декларативний інтерфейс опису служби (SDL), який дозволяє визначити методи, які доступні для виклику, та їх параметри та повертає значення. Це спрощує створення API та розуміння можливостей служби. (Рис.1.15).

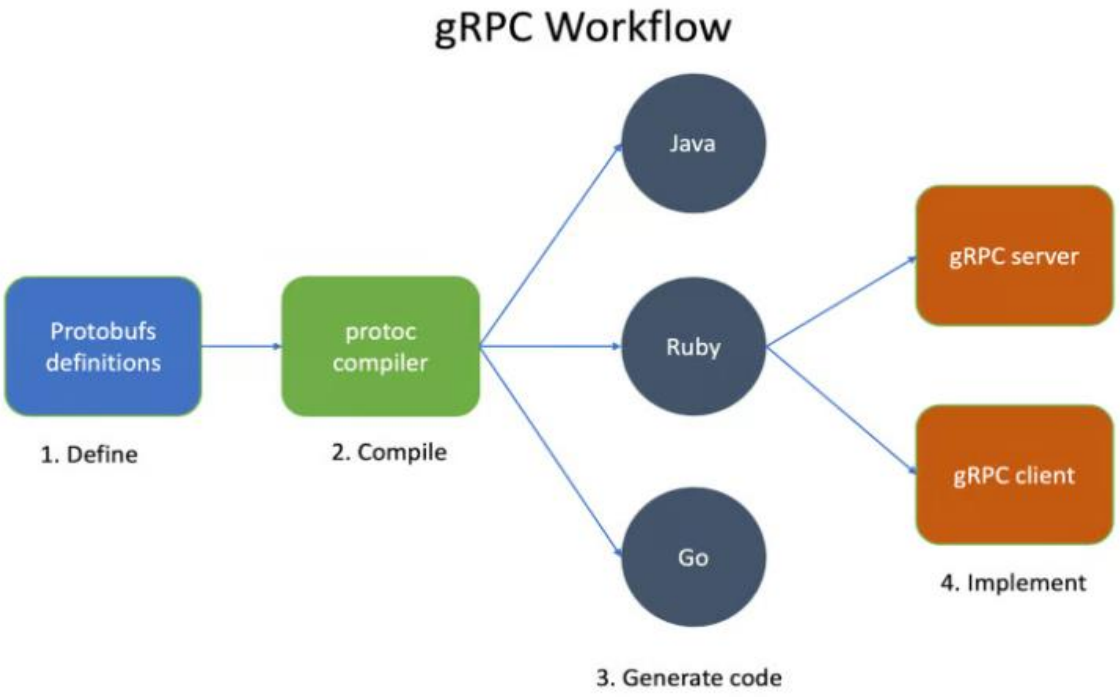


Рисунок 1.15 - Архітектура gRPC API [13]

gRPC, на відміну від REST, був розроблений спеціально для того, щоб дозволити розробникам створювати високопродуктивні API для мікросервісної архітектури у розподілених центрах обробки даних. Він найкраще підходить для внутрішніх систем, яким потрібна потокова передача в реальному часі та великі обсяги даних.

Незважаючи на свої переваги, gRPC не так широко використовується, як традиційний REST поверх HTTP/1.1 через кілька факторів: підтримка браузера: gRPC вимагає HTTP/2, який не повністю підтримується всіма веб-браузерами, особливо для потокової передачі з сервера.

Falcor API - це стиль API, схожий на GraphQL. (Рис.1.16).

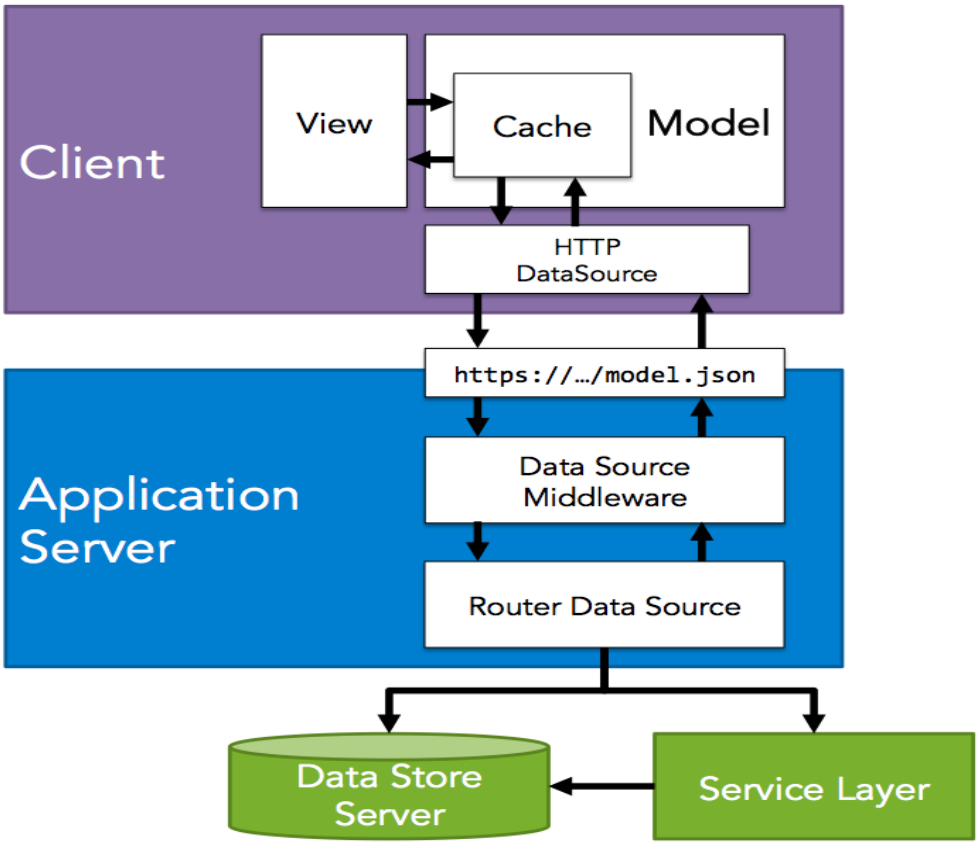


Рисунок 1.16 - Архітектура Falcor API [13]

Він представляє віртуальний рівень, який можна використовувати для відображення зовнішніх запитів на серверні служби. Основна ідея Falcor API полягає у створенні віртуального ресурсу JSON як центральної моделі даних, яка фактично використовується як контейнер. Цей віртуальний ресурс JSON опубліковано під URL-адресою. Потім Falcor зв'язує дані з різних серверних програм і джерел даних у цей віртуальний ресурс JSON. Віртуальний ресурс JSON є всеохоплюючим і розрахований на розширення з часом. Як наслідок, віртуальний ресурс JSON досить великий. Клієнтам не потрібно отримувати його повністю, тому, подібно до GraphQL, клієнт може вказати, яка частина ресурсу цікава, і лише цю конкретну частину слід отримати. Falcor особливо добре підходить для програм із комплексними вимогами до отримання даних та потребами оновлення в реальному часі.

Webhook — це метод, що представляє собою специфічний тип API, який працює на основі подій. На відміну від традиційних API, які реагують на запити інших програм, веб-перехоплювачі відправляють інформацію або виконують певні дії у відповідь на певні події або тригери, такі як певний час, натискання кнопки або отримання форми. Оскільки ініціатива щодо передачі даних належить програмі, яка відправляє їх, веб-перехоплювачі часто називаються "зворотніми API".(Рис.1.17)

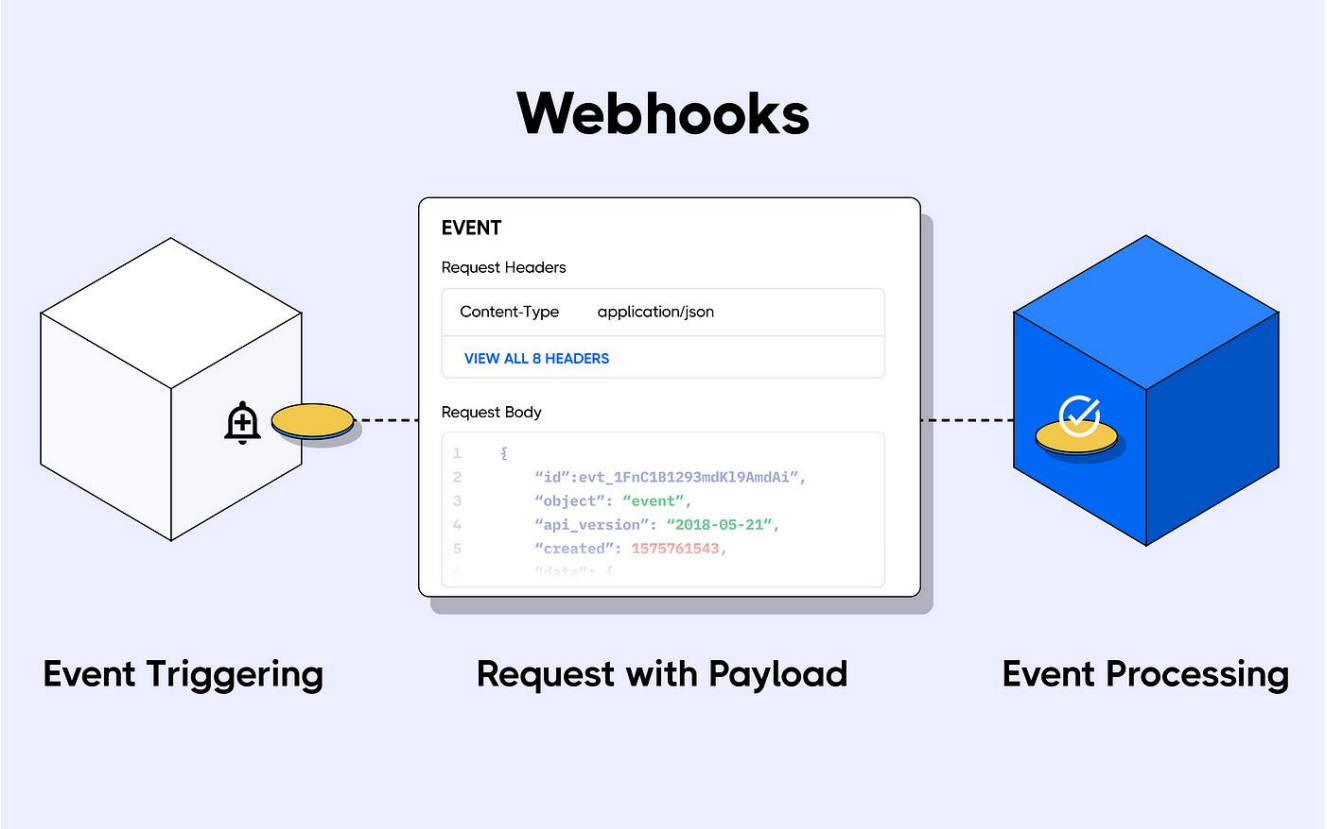


Рисунок 1.17 - Архітектура Webhook [13]

Веб-перехоплювач (Webhook) — це підхід, відмінний від звичайного веб-API. Webhook дуже схожі на API, але основна увага вебхуків буде на сповіщеннях POST. Це полегшена версія API. Оскільки вебхуки працюють на основі подій і не вимагають постійних запитів до сервера, вони можуть бути більш ефективними з точки зору використання ресурсів і забезпечувати швидший обмін даними між системами. В Вебхуках дані передаються тільки в одному напрямку. Вони являють собою прості API-ендпоїнти, визначені розробником, що робить їх

значно простішими у використанні порівняно з повноцінними API. Завдяки цьому, вебхуки можуть бути швидко налаштовані та легко інтегровані в існуючі системи. Вони часто використовуються для наступних випадків:

- *Сповіщення про нові підписки або реєстрації*: Можна використовувати Webhook для автоматичного відправлення вітальних листів або іншої інформації новим користувачам після реєстрації або підписки на сервіс.

- *Моніторинг та оповіщення у хмарних сервісах*: Коли відбувається певна подія у хмарному сервісі, наприклад, завершення резервного копіювання або досягнення певного ліміту використання, вебхук може автоматично відправляти сповіщення адміністраторам

- *Сповіщення про зміни у базі даних:* Webhook може використовуватися для інформування інших систем або служб про зміни в базі даних, наприклад, додавання нових записів, оновлення або видалення даних.

- *Сповіщення про нові замовлення в інтернет-магазині:* Коли клієнт робить покупку, вебхук може автоматично відправляти інформацію про нове замовлення до системи обліку або CRM для подальшої обробки, не вимагаючи ручного введення даних.

Використання Webhook у цих випадках дозволяє забезпечити оперативність та автоматизацію процесів, зменшуючи потребу у ручному втручанні та підвищуючи ефективність систем. Але важливо ретельно тестувати вебхук та регулярно перевіряти його роботу, щоб гарантувати коректне функціонування. Тому що, якщо запит вебхука має неправильний формат, детального пояснення причини збою ви не отримаєте — тільки код стану, такий як 400 або 404.

На відміну від звичайного розгортання API RESTful, коли на сервері розміщується кінцева точка API на основі HTTP, яку клієнти («споживачі API») отримують дані з одного запиту за раз, веб-перехоплювачі змінюють напрямок діалогу. Це клієнт, на якому розміщена кінцева точка API на основі HTTP, на яку сервер надсилає дані в міру їх появи. Ця кінцева точка називається веб-перехоплювачем. Веб-перехоплювачі не вимагають багато "розмов" - дані

передаються в одному напрямку, а не в двох. Це просто кінцеві точки API, вказані розробником, що робить їх досить простими проти повноцінними API.

Оскільки вони запрограмовані так, щоб не мати доступу до такої кількості інформації як цілі системи API, їх використання досить обмежене. Однак вони виявляються корисними, коли користувач хоче виконати функцію програми, навіть не відкриваючи програму. Ось кілька випадків, коли Webhook буде найбільш доречним:

- *Оновлення статусу передплати користувача* у вашій системі управління взаємовідносинами з клієнтами (CRM), коли користувач відмовляється від передплати

- *Надсилання автоматичних нагадувань про збори* за п'ять хвилин до їх початку

- *Надсилання повідомлень електронною поштою*, в яких користувачам, які намагаються зв'язатися зі співробітником на умовах PTO, повідомляється дата його повернення.

- *Повідомлення користувача, що володіє акціями компанії*, коли ціна акцій падає на 5% за день

Якщо запит веб-перехоплювача відформатовано неправильно, ви не отримаєте детальної відповіді, що пояснює, чому ваша функція не вдалася, — ви просто отримаєте код стану, наприклад 200 або 404. Тому важливо протестувати свій веб-перехоплювач (і регулярно перевіряти його), щоб переконатися, що він працює правильно.

2. ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ WEBHOOK У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМИ ЩОДО ПРОДУКТИВНОСТІ ПРИЛАДІВ НА СЕРВЕРІ

2.1 Дослідження різноманітних галузей застосовується Webhook для отримання даних з серверу

Треба відмітити, що Webhook потужний інструмент, який дозволяє розробникам інформувати кілька сервісів про оновлений контент або дані користувача. Його можна використовувати практично для чого завгодно, включаючи push-повідомлення про відправку товару або відправку платіжної інформації під час здійснення покупки. Він забезпечує простоту та зручність без необхідності ручного введення або проміжного програмного забезпечення, що робить веб-розробку більш простою, ніж будь-коли. Це дозволяє веб-сторінкам залишатися синхронізованими, оскільки веб-перехоплювачі дозволяють розробникам швидко підключати два веб-сервіси та автоматично передавати інформацію між ними в режимі реального часу. Це простий спосіб, за допомогою якого ваші онлайн-акаунти можуть "спілкуватися" один з одним і автоматично отримувати повідомлення, коли відбувається щось нове.

Webhook зазвичай використовуються для з'єднання двох різних програм. Коли в програмі-тригері відбувається подія, вона серіалізує дані про цю подію і відправляє їх на URL-адресу веб-перехоплювача з програми дії - тієї, яку ви хочете щось зробити на основі даних з першої програми. Прикладами різноманітних галузей застосування Webhook є: Телеграм, Shopify, GitHub, PayPal, Twilio, WooCommerce та інші.

Webhook в Телеграм (багатоплатформовий месенджер) - це URL, який ви надаєте телеграму, і за яким телеграм буде відправляти оновлення у форматі JSON щоразу, коли вашому боту надходить повідомлення або відбувається інша подія.

Webhook в Shopify(платформа, яка допомагає створювати та керувати інтернет-магазинами): Shopify надає webhooks для сповіщення зовнішніх систем про події, пов'язані з онлайн-магазинами, такі як нові замовлення, виконання замовлень або оновлення клієнтів. Це дозволяє розробникам створювати власні інтеграції з Shopify та автоматизувати завдання, такі як обробка замовлень або оновлення інформації про залишки товарів.

Webhook в GitHub: GitHub використовує webhooks для сповіщення зовнішніх систем про події, які відбуваються в репозиторіях. Це можуть бути нові коміти, запити на отримання(pull requests) або якісь проблеми. Розробники можуть використовувати ці сповіщення для запуску автоматизованих робочих процесів, таких як безперервна інтеграція, розгортання або відстеження проблем.

Webhook в Twilio. Twilio надає інструменти для здійснення та отримання телефонних дзвінків, надсилання та отримання текстових повідомлень перенаправляє телефонні дзвінки на ваш номер і як WooCommerce (плагін електронної комерції для WordPress) може повідомляти вас про нові замовлення в Slack (корпоративний месенджер).

Webhook в Stripe. Використовує webhooks для сповіщення зовнішніх систем про події, що відбуваються з платіжними транзакціями та обліковими записами. Це можуть бути успішні або невдалі платежі, створення нових підписок або зміни статусу підписок. Розробники можуть використовувати ці сповіщення для запуску автоматизованих робочих процесів, таких як оновлення статусу замовлень, відправка квитанцій клієнтам або інтеграція з бухгалтерськими системами для точного відстеження фінансових операцій.

2.2 Дослідження характеристик Webhook, щоб надавати програмам інформацію в реальному часі при виникненні певної події.

Для контролю за якістю функціонування веб-серверу додатків виконується моніторинг продуктивності цього сервера, що має вирішальне значення. Тому на показники продуктивності веб-серверу додатків слід завжди звертати увагу. Моніторинг веб-сервер додатків вирішує це завдання. Він дає уявлення про його продуктивність, що життєве важливий для запобігання збоям у бізнес-операціях. Одним з способів підвищення продуктивності веб-сервер додатків є застосування систем обробки та управління сповіщенням цих серверів.

Системи обробки та управління сповіщенням в серверах працюють наступним чином. Ваш веб-сервер додатків відправляє вміст сповіщення та токени пристроїв в API служби push-повідомлень.

Програмний інтерфейс — містить важливі підпрограми, протоколи взаємодій та особливості для створення програмного забезпечення. У контексті API слово «Додаток» стосується будь-якого програмного забезпечення з певною функцією. Інтерфейс можна розглядати як контракт обслуговування між двома програмами. Цей контракт визначає, як вони взаємодіють один з одним, використовуючи запити та відповіді. Тобто, як тільки сервер відправить запит до служби push-повідомлень, служба відправіть повідомлення на пристрої ваших користувачів. API — це програма з функціями або дзвінками для додавання, редагування та отримання даних. Використання API вимагає виконувати всю роботу щодо обробки та управління сповіщенням самостійно.

Проблеми безпеки API. Інфраструктури API можуть бути вразливими для порушень безпеки, таких як атаки SQL-ін'єкцій та атаки міжсайтового скриптингу (XSS), тому дуже важливо впровадити заходи безпеки для захисту від цих загроз. Помилка API означає, що сервер не може знайти запитаний ресурс у постачальника API. При виникненні збою API назад

надсилається числове повідомлення про помилку, яке намагається визначити, яка помилка була здійснена користувачем. Часті причини помилок API включають проблеми в кінцевій точці, неправильні параметри або проблеми з ключем API під час виклику запиту. Вирішення цих помилок має вирішальне значення для безперебійного зв'язку між програмами та забезпечення успішного отримання даних від постачальника API.

Складність розробки, що відповідає потребам бізнесу. Також однією з проблем API є складність його розробки, що відповідає потребам бізнесу та кінцевих користувачів. Проектування API вимагає глибокого розуміння бізнес-вимог, архітектури системи та потреб кінцевих користувачів.

Для подолання вказаних недоліків API-інтерфейсів під час автоматизації процесу обробки та управління сповіщенням в веб-сервері додатків розроблено новий підхід, якій має назву Webhook. Застосування засобів Webhook у системах управління сповіщення для підвищення продуктивності сервера - це інноваційна галузь, яка створює можливість комп'ютерам і програмам по іншому виконувати завдання.

Webhook — це метод, щоб збільшити або змінити поведінку веб-сторінки або програми додатку в веб-сервері за допомогою зворотних дзвінків.

Можна казати, що це механізм, який дозволяє веб-серверам надсилати повідомлення або дії в режимі реального часу іншим системам під час виникнення певних подій на веб-сторінці або у додатку.

Для розуміння як працює Webhook треба в першу чергу визначити що це таке. Назва webhook є простою комбінацією слова web, що відноситься до його зв'язку на основі HTTP, і функції hook програмування перехоплення (веб-перехоплювач), яка дозволяє програмам перехоплювати виклики або інші події, які можуть представляти інтерес.

Ось приклад повідомлення, яке надсилається до зовнішньої системи: CRM, інтернет-магазин, кол-центр. Тобто виглядає Webhook як посилання:

http://mysite.ru/webhook?id={id}&type={type}. Це програмний код складається з двох частин - змінної та самих даних.

Webhook є подібним до API, але простіший у використанні. Якщо вам потрібно здійснювати власні запити для додавання, редагування та отримання даних, то це може вимагати багато ручної роботи. Проте застосування Webhook автоматизує цей процес, дозволяючи веб-застосункам відправляти автоматичні повідомлення один одному безпосередньо. Звідси Webhook має функцію Callback, тобто зворотного виклику. Функція зворотного виклику створює код, що аналогічний коду, що реалізує зворотний виклик. Код передає повідомлення запрос в іншу функцію як аргумент, а потім викликається після завершення будь-якої дії результат.

Технічно Webhook є веб-перехоплювачем, що будується за допомогою протоколу HTTP.

Веб-перехоплювачі – це дані та виконувані команди, що надсилаються з однієї програми до іншої через HTTP, а не через командний рядок на вашому комп'ютері, у форматі XML, JSON або серіалізації з кодуванням форми. Їх називають веб-перехоплювачами, оскільки вони є програмними перехоплювачами — або функціями, які запускаються, коли щось відбувається, — і працюють через Інтернет.

Вони зазвичай захищені за рахунок скритності - кожен користувач програми отримує унікальну випадкову URL-адресу для відправки даних веб-перехоплювача - хоча за бажанням їх можна захистити за допомогою ключа або підпису.

Для розуміння як працює Webhook розглянемо приклад. Припустимо, ви хочете знати щоразу, коли погода опускається нижче 0 градусів за Цельсієм, Для цього треба, щоб метеорологічна служба передавала цю інформацію через інтеграцію веб-перехоплювача. Ви створюєте URL-адресу веб-перехоплювача у своїй системі, яка під час виклику повідомляє вам, що температура впала нижче 0 градусів за Цельсієм. Після реєстрації цієї URL-адреси у метеорологічної службі ця служба викликає автоматичне вашу URL-

адресу кожного разу, коли температура падає нижче 0 градусів за Цельсієм. Тобто запит веб-перехоплювача — це тип дзвінка HTTP, який визначається користувачем. По суті це повідомлення про подію через HTTP POST. Коли у вихідній програмі відбувається певна подія, вона запускає веб-перехоплювач, який потім відправляє повідомлення, зазвичай у форматі JSON або XML, на вказану URL-адресу — URL-адресу веб-перехоплювача. Це повідомлення містить інформацію про подію, яка щойно відбулася.

Звідси Webhook — це інструмент, який використовується у веб-розробці або розробці API, що дозволяє програмам надавати інформацію в реальному часі іншим програмам при виникненні певної події. У них є корисне навантаження, яке відправляється на унікальну URL-адресу, надаючи програмам простий спосіб надсилати дані один одному. Тобто Webhook — це URL-адреса, яку ви надаєте іншій системі, яку система викликає, коли відбувається якась подія, про яку вам потрібно знати.

Використовувати Webhook для відстеження на стороні сервера просто. За допомогою веб-перехоплювачів дані можуть автоматично відправлятися в серверний контейнер Диспетчера тегів Google щоразу, коли на вашому веб-сайті відбувається будь-яка подія або дія. Найбільш поширеним варіантом використання веб-перехоплювача при додаванні тегів на стороні сервера є надсилання веб-перехоплювачів із CRM або CMS. Наприклад, кожного разу, коли користувач створюється, оновлюється, розміщується замовлення та інші ви можете надсилати ці дані на сервер GTM.

Функція програмування перехоплення в Webhook, дозволяє створювати веб-перехоплювачі, які перехоплюють подію, що відбувається в серверній програмі, і пропонують серверу відправити корисне навантаження клієнту через Інтернет. Веб-перехоплювачі використовуються безліччю веб-додатків для отримання невеликих обсягів даних від інших програм, але веб-перехоплювачі також можна використовувати для запуску робочих процесів автоматизації в середовищах GitOps. GitOps — це операційне середовище, яке використовує найкращі практики DevOps, які використовуються для розробки

програм, такі як контроль версій, спільна робота, відповідність вимогам та CI/CD, та застосовує їх для автоматизації інфраструктури.[8]

Таким чином веб-перехоплювачі діють як віртуальні повідомлення між програмами, які запускаються якоюсь подією, наприклад, коли ви хочете отримувати повідомлення кожного разу, коли у вас з'являється новий передплатник. Працюючи в режимі реального часу, Webhook дозволяють передавати дані з одного додатку (застосунку) до іншого і можуть допомогти уточнити дії, які здійснюють ваші передплатники. Можна казати що, це новий підхід до архітектури програмного забезпечення, який дозволяє програмам і службам надсилати веб-повідомлення іншим програмам щоразу.

2.3 Дослідження умов використання Webhook для отримання даних з серверу

Дослідження умов використання Webhook для отримання даних з серверу треба виконати наступне:

По-перше, розглянемо коли потрібно використовувати Webhook

1. Односторонній зв'язок у реальному часі (від джерела до місця призначення)
2. Постійне з'єднання між зв'язком двох систем.
3. Ви хочете негайно відреагувати на подію з SaaS, яка підтримує веб-перехоплювачі.
4. Ви хочете використовувати модель push для негайного надсилання оновлень.

По-друге, розглянемо які переваги дозволяє отримати застосування Webhook:

1. *Усуває потребу опитування*. Ефективні та швидкі, оскільки усувають необхідність опитування чи оновлення для перевірки нових даних.

Опитування — це коли веб-програма неодноразово запитує дані в іншої веб-програми через певні проміжки часу, що може споживати смугу пропускання та ресурси. Оновлення – це коли користувач перезавантажує веб-сторінку для перегляду нових даних, що може бути незручно та повільно. З іншого боку, веб-перехоплювачі передають дані передплатнику, як тільки подія відбувається, що зменшує затримку та покращує взаємодію з користувачем. Це заощаджує ресурси клієнтської програми.

2. *Швидко налаштовуються*. Якщо програма підтримує веб-перехоплювачі, їх легко налаштувати через інтерфейс серверної програми. Тут клієнт вводить URL-адресу веб-перехоплювача своєї програми та встановлює деякі основні параметри, наприклад, яка подія його цікавить.

3. *Автоматизує передачу даних*. Корисне навантаження відправляється, коли вказана подія відбувається в серверній програмі. Цей обмін ініціює подія, тому він відбувається так швидко, як тільки дані можуть бути передані з сервера на пристрій клієнта — у режимі реального часу, наскільки це можливо при будь-якій передачі даних.

4. *Забезпечує обмін у формі коротких сповіщень про специфічні корисні навантаження*. Webhook покладаються на те, що сервер визначає обсяг даних, що надсилаються, залишаючи клієнту можливість інтерпретувати корисне навантаження і використовувати його продуктивним чином. Оскільки клієнт не контролює точний час або розмір передачі, веб-перехоплювачі обробляють невеликі обсяги інформації між двома кінцевими точками, часто у формі сповіщення.

5. *Оновлення в режимі реального часу*: веб-перехоплювачі забезпечують миттєві повідомлення, на відміну від механізмів опитування, які потребують регулярної перевірки оновлень.

6. *Ефективність*: вони знижують навантаження на сервер і підвищують продуктивність, оскільки усувають необхідність частого опитування. Прикладом є пакетна обробка: веб-перехоплювач може використовуватися для повідомлення про завершення завдання.

7. *Гнучкими для налаштування*: Ви можете визначити логіку для керування тим, які події запускають запити веб-перехоплювача та які дані надсилаються. Іноді вам потрібна лише конкретна інформація. Веб-перехоплювачі також можуть підтримувати кілька передплатників на ту саму подію, що забезпечує інтеграцію та автоматизацію різних веб-додатків. Наприклад, у Stripe є чудова пропозиція веб-перехоплювачів. Замість того, щоб отримувати інформацію про кожну подію, що відбувається в Stripe, а їх дуже багато, ви можете зареєструватися на конкретні події, які вас цікавлять.

8. *Можуть видаляти повідомлення*. Webhook може надсилати повідомлення до текстового каналу без необхідності входу в систему як бот. Вони також можуть отримувати, редагувати та видаляти власні повідомлення

По-третє, недоліки використання Webhook.

1. *Не гарантують доставку або підтвердження.* Дані або повідомлення не завжди можуть бути надіслані до цільової системи. Це означає, що видавець не знає, чи передплатник успішно отримав і обробив веб-перехоплювач. Це може призвести до втрати, неузгодженості або дублювання даних. Якщо вихідна система простоює, вона може не захопити відповідні дані або сповіщення; коли це станеться, повідомлення або дані не будуть надіслані до цільової системи.

2. *Доставка Webhook може завершитися невдачею з кількох причин.* Наприклад, якщо ваш сервер не працює або на відповідь потрібно більше 10 секунд, GitHub зареєструє доставку як збій. GitHub не виконує автоматичну повторну доставку невдалих поставок.

3. *Основною вразливістю будь-якого сервісу веб-перехоплювачів є підробка запитів на стороні сервера (SSRF).* SSRF – це коли зловмисник змушує вашу службу виконати внутрішній ненавмисний запит у власній мережі. Webhook - ідеальна мета для цього.

4. *Webhook не надійні та небезпечні*, а це означає, що вони схильні до збоїв, які можуть виникнути з кількох причин, наприклад: Збій мережі. Погані

запити. Помилка автентифікації. Якщо один з них перебуває в автономному режимі, перевантажений або скомпрометований, *Webhook* може вийти з ладу, затриматися або бути перехопленим.

5. *Webhook складні і ними важко керувати*, оскільки вони вимагають від видавця та передплатника налаштування та підтримки кінцевих точок веб-перехоплювачів, автентифікації та обробки помилок. Веб-перехоплювачі також збільшують зв'язок і залежність між веб-додатками, що може ускладнити їх масштабування, оновлення або налагодження.

По-четверте, розглянемо процес налаштування та використання Webhook.

Процес налаштування та використання Webhook включає кілька етапів: *конфігурація, подія-тригер, повідомлення та дія.*

Конфігурація. Застосування засобів Webhook у системах обробки та управління сповіщенням для підвищення продуктивності сервера має сценарії відносно наступної схеми: сервер веб-додатків/панель моніторингу веб-додатків. У цьому сценарії веб-додаток має інтерфейс панель та сервер. Наприклад,

крок 1 - можна налаштувати URL-адресу веб-перехоплювача на сервері для отримання повідомлення про успішну публікацію запланованого повідомлення на Facebook.

Крок 2 - Webhook сервера отримує це повідомлення.

Крок 3 - потім запускає процес оновлення деяких даних та панелі керування користувача.

Крок 4 - кінцевий користувач побачить на панелі керування нові дані, що повідомлення було успішно опубліковано.

Подія-тригер. Тригерні сповіщення — це push-сповіщення, які автоматично активуються, коли на вашому веб-сайті відбувається певна подія користувача. Події – це взаємодія користувача з контентом на вашій сторінці, яка може викликати автоматичне push-сповіщення. Щоб розпочати роботу з

веб-перехоплювачами на стороні сервера, треба спочатку вирішити, яку програму (додаток) ви хочете запускати і яка програма буде їх отримувати. Зазвичай надсилання додаток називається «джерелом», а приймає додаток - «призначенням». Після вибору обох програм наступним кроком буде налаштування параметрів вебхука для кожної з них.

Повідомлення та дія. Повідомлення та дія. Webhook активується, коли відбувається подія на вашому сайті, у CRM, або будь-яких інших системах. Запускається тригер Webhook - це автоматичний тип тригера, який прослуховує певний тип даних, подібно до тригерів подій. У той час як тригери подій використовуються для активації тригера на основі внутрішньої активності, веб-перехоплювачі замість цього використовуються при активації тригера на основі зовнішньої активності. Наприклад, людина написала коментар або додала новий товар до системи обліку товарів. Коли ця подія відбувається, сервер створює HTTP-запит і надсилає його на адресу, вказану клієнтом, для отримання веб-перехоплювачів. Клієнт задоволений, отримуючи своєчасно нові дані. Користувач може налаштувати вебхуки так, щоб події на одному сайті спричиняли виконання дій на інших. Наприклад, коли клієнт створює замовлення в інтернет-магазині, система автоматично відправляє вебхук у додаток власника, який повідомляє його та надсилає оцінку.

Налаштовування веб-перехоплювача. У випадку з веб-перехоплювачами процес налаштування зазвичай складається з трьох етапів:

1. Отримайте URL-адресу веб-перехоплювача з програми, до якої потрібно надіслати дані.

2. Використовуйте цю URL-адресу в розділі веб-перехоплювача програми, з якої ви хочете отримувати дані.

3. Виберіть тип подій, про які ви хочете, щоб програма повідомляла вас.

Після налаштування, коли додається новий контакт, програма автоматично передає дані на URL-адресу веб-перехоплювачів іншої програми. Це просте з'єднання один до одного, яке запускається автоматично.

У п'ятих, розглянемо рекомендації щодо використання Webhook.

Щоб подолати деякі недоліки веб-перехоплювачів і скористатися їхніми перевагами, наведені нижче рекомендації щодо використання веб-перехоплювачів для доставки даних у реальному часі.

1. Треба використовувати HTTPS та шифрування, щоб убезпечити зв'язок через веб-перехоплювач та захистити дані від несанкціонованого доступу чи зміни.

2. Використовуйте повтори та підтвердження, щоб забезпечити доставку та обробку веб-перехоплювачів, а також коректне обробляти помилки та збої.

3. Використовуйте фільтри та перехоплювачі, щоб вказати події та дані, які хочете надсилати та отримувати, та уникайте надсилання непотрібних або конфіденційних даних.

4. використовуйте журнали та моніторинг для відстеження та аналізу активності та продуктивності веб-перехоплювачів, а також для виявлення та усунення будь-яких проблем та аномалій.

3. РОЗРОБКА ПРАКТИЧНИХ РЕКОМЕНДАЦІЙ ДЛЯ ВПРОВАДЖЕННЯ WEBHOOK У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМИ ЩОДО ПРОДУКТИВНОСТІ ПРИЛАДІВ НА СЕРВЕРІ

3.1 Архітектура збору та обробки Webhook

У систему збору та обробки Webhook входять такі файли:

1. webhook_https_get_push.ua,
2. server_collector.ua,
3. restart_webhook.sh,
4. restart_server_collector.sh.

Webhook використовуються у стилі API, де сервер передає дані клієнту. Клієнту не потрібно робити повторні запити на сервер. Цей архітектурний стиль API push/streaming добре підходить для випадків, коли базові дані постійно оновлюються, наприклад, біржові котирування або потік соціальної активності.

Webhook відмінний від звичайного веб-API. На відміну від звичайного розгортання RESTful API, коли на сервері розміщується кінцева точка API на основі HTTP, з якої клієнти («споживачі API») отримують дані з одного запиту за раз, веб-перехоплювачі змінюють напрямок діалогу. Це клієнт, на якому розміщується кінцева точка API на основі HTTP, куди сервер передає дані в міру їхньої доступності. Ця кінцева точка відома як Webhook [32].

Веб-перехоплювачі — це стиль push-повідомлень, який, порівняно з можливостями маршрутизації інших API-інтерфейсів, у стилі push/streaming знаходиться на найнижчому кінці спектру з точки зору складності. У порівнянні з більш вузькоспрямованими механізмами push-повідомлень цей стиль надає обмежені можливості маршрутизації окремим користувачам додатків. Це можливо, але Webhook краще підходять для надсилання повідомлень на одну або невелику кількість кінцевих точок. Якщо повідомлення призначене для окремого користувача програми, власник кінцевої точки зазвичай бере на себе відповідальність за пересилання повідомлень, отриманих Webhook, правильному одержувачу(Рис.3.1).

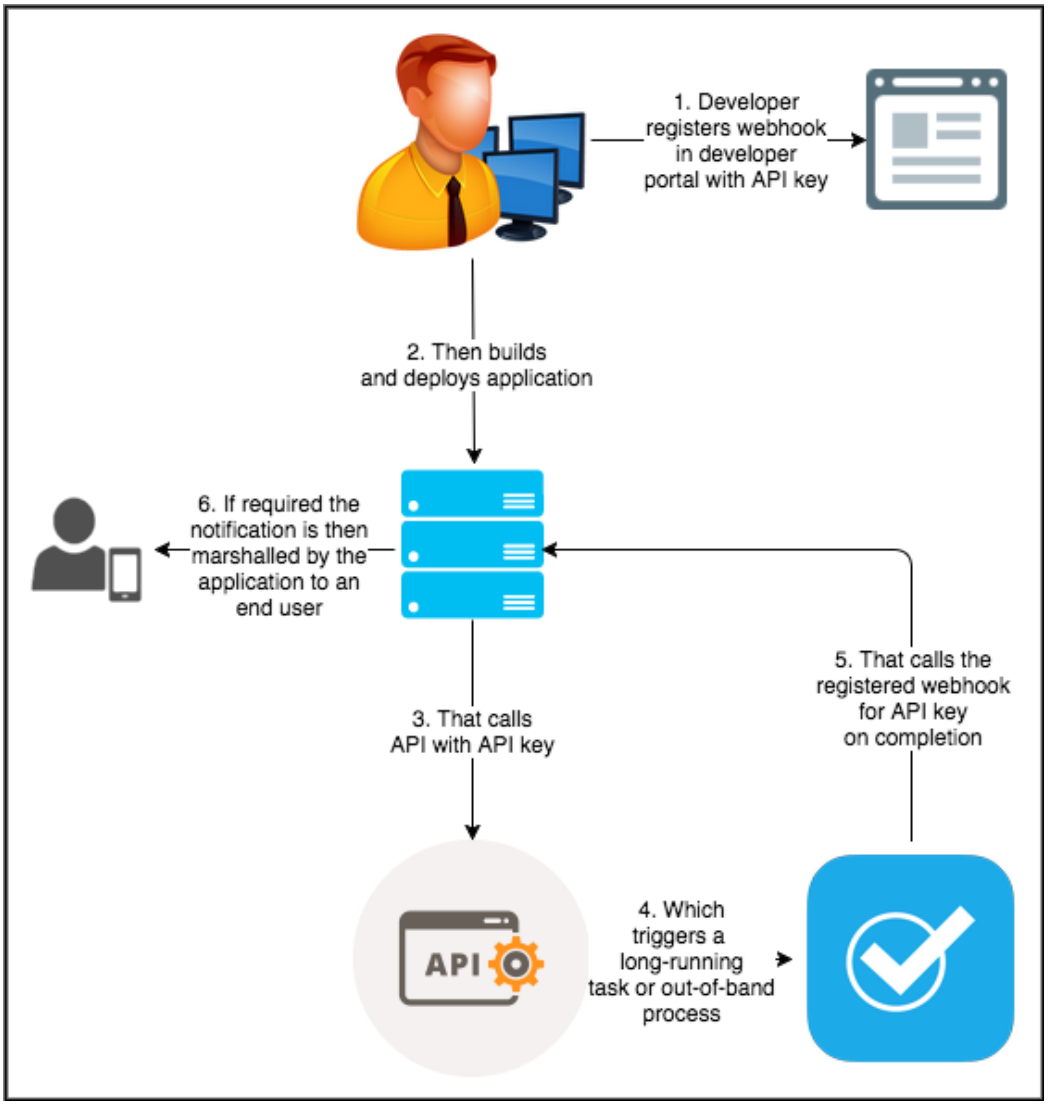


Рисунок 3.1 - Архітектура Webhook [13, 15]

Реалізація веб-перехоплювачів зазвичай включає три етапи, під час яких споживач API викликає API із запитом отримання сповіщень, а сервер виконує зворотний виклик зі своїм потоком. Ці кроки такі:

Постачальник API реалізує API, який викликає тривалі процеси, які неможливо дочекатись при синхронному з'єднанні або які генерують позасмугові події.

Далі потрібне сповіщення споживача API. Прикладом може бути API служби підтримки, яка створює заявки, для виконання яких потрібна взаємодія з людиною протягом кількох днів. Цей API також запускатиме оновлення статусу, які споживач API повинен знати протягом усього терміну дії заявки.

Споживач API реєструється для використання API та налаштовує його параметри (через портал розробника провайдера), використовуючи URL-адресу своєї загальнодоступної кінцевої точки (з деякими функціями безпеки).

Постачальник API може "пересилати" назад у цю кінцеву точку, коли тривалий процес завершується або коли цей процес запускає події, про які необхідно повідомити споживача.

Потім процес на стороні клієнта може продовжити деякий робочий процес на основі вмісту даних, які були передані його веб-перехоплювач.

Наприклад, у дусі програмованої торгівлі Webhook може належати біржовій брокерській фірмі, і потік даних, що передається на цей Webhook, може включати ціни на акції, які можуть ініціювати продаж або купівлю акцій, що публічно торгуються.

Описаний вище сценарій включає попередньо зареєстровані URL-адреси, але технічно можливо надати URL-адресу веб-перехоплювача "на льоту", коли споживач виконує виклик API. Обидва методи мають плюси та мінуси:

Попередньо зареєстровані веб-перехоплювачі менш гнучкі для споживачів API, які їх розміщують, оскільки зміни конфігурації потрібні щоразу, коли споживач бажає змінити адресу веб-перехоплювача.

Веб-перехоплювачі «на льоту» можуть наражатися на загрозу безпеці, якщо вхідний запит перехоплюється і змінюється в результаті атаки типу «людина посередині»(man-in-the-middle attack).

Атака «людина посередині» може статися, якщо зловмисник отримає доступ до каналу зв'язку між клієнтом і сервером. Зловмисник може перехопити запити на вебхук, надіслані із сервера клієнту, змінити дані, що містяться в цих запитах, а потім переслати їх клієнту.

Тому необхідно використовувати додаткову безпеку, така як підписання повідомлень або закріплення сертифіката, для унеможливлення відмови обох сторін.

*Приклад скрипта установки вебхука в system(***webhook_install.sh)***:* Створимо функцію handle_error, яка друкує повідомлення про помилку до стандартної помилки та виходить зі скрипту з кодом стану 1. Обробка помилок запобігає частковій або неправильній установці. Наприклад, якщо двійковий файл webhook не завантажується, продовження виконання скрипта може призвести до нефункціонального налаштування. Обробка помилок гарантує, що система залишається в узгодженому стані, припиняючи подальші дії, коли щось йде не так. Далі потрібно перевірити, чи доступна в системі команда wget. Це важливо, оскільки wget потрібен пізніше в сценарії для завантаження файлів, і сценарій не може продовжити без нього. Приклад програмного коду надається у Додатку А.

Після цього починається процес інсталяції webhook у системі Linux. На початку завантажуємо двійковий архів вебхука, розпаковуємо його, переміщуємо двійковий файл у /home/someone_username/bin, очищаємо тимчасові файли та надаємо відгук про процес. Приклад програмного коду надається у Додатку А.

Ще ми маємо встановити шляхи до каталогів, створити каталоги та налаштувати дозволи в нашому сценарії — це важливо для того, щоб процес налаштування вебхука був плавним, безпечним і надійним. Це забезпечить необхідне середовище та контроль доступу для правильного функціонування сценарію та ефективного виконання завдань, пов’язаних із вебхуками.

Програма **webhook_https_get_push.ua** налаштовує веб-сервер Flask для обробки вхідних POST-запитів вебхуків. Вона очікує надходження даних (подій) і повинна бути активною, щоб отримати дані, коли вони надходять. Як тільки дані надходять, webhook_https_get_push.ua ініціює відправку цих даних на сервери (**IP_to_push_to_1 і IP_to_push_to_2**). Повний пограмний код надається у Додатку Б.

server_collector.ua (колектор «Сервер-сервер») – це міжсерверний збирач. Міжсерверний збірник дозволяє збирати дані та повідомлення з вихідного сервера на цільовий сервер, екземпляр часових рядів, екземпляр Azure IoT HUb або кінцеву точку MQTT, Колектор «Сервер-сервер» включає багато функцій збирача обчислень. Основна відмінність полягає в тому, що збирач "сервер-сервер" зберігає результат у тегу призначення на цільовому сервері, тоді як збирач обчислень зчитує та записує на той же сервер.

server_collector.ua запускає HTTP сервер, який слухає запити на певному порту (8080). Коли надходить POST запит, дані вилучаються і записуються в лог-файл. Отже, server_collector.py використовується як компонент для збору даних, які відправляються з webhook_https_get_push.py.

Колектор "сервер-сервер" також може працювати як автономний компонент, коли вихідна та цільова бази даних знаходяться на віддалених комп'ютерах.

Коли відбувається спрацьовування цільового тега за часом або за подією:

- Формулу розрахунку для тега призначення буде виконано.
- Зазвичай це передбачає отримання даних із одного або кількох тегів на вихідному сервері.

- Визначається необроблена вибірка чи помилка розрахунку.

Можна використовувати умовну логіку у формулі розрахунку, щоб визначити, чи слід відправляти зразок до пункту призначення.

Необроблений зразок доставляється на сервер призначення, за потреби використовуючи функцію збереження та пересилання.

Реплікація повідомлень, якщо вона увімкнена, базується на подіях. Повідомлення та оповіщення надсилаються на сервер призначення у міру їх виникнення.

Тег призначення принципово відрізняється від джерела тега. Тому:

- Коли тег додається під час перегляду, лише певні властивості тега копіюються з вихідного тега до цільового тега. Тут треба подумайти, які властивості потрібні вашому додатку, і налаштуйте їх вручну. Інформацію про властивості, що копіюються, див. у розділі Копіювані властивості тегів.

- Якщо ви зміните властивість тега вихідного тега (обмеження EGU, описи тощо), ця властивість не зміниться автоматично у тезі призначення. Ви можете змінити властивості цільового тега вручну.

Потік даних у кількох міжсерверних збирачах. На наступному зображенні Рис.3.2 показано, що в програмі можна використовувати кілька збирачів «сервер-сервер» для передачі даних з декількох вузлів на один вузол, і що сервер може одночасно бути джерелом та місцем призначення. Кожен сервер надсилає свій набір тегів.

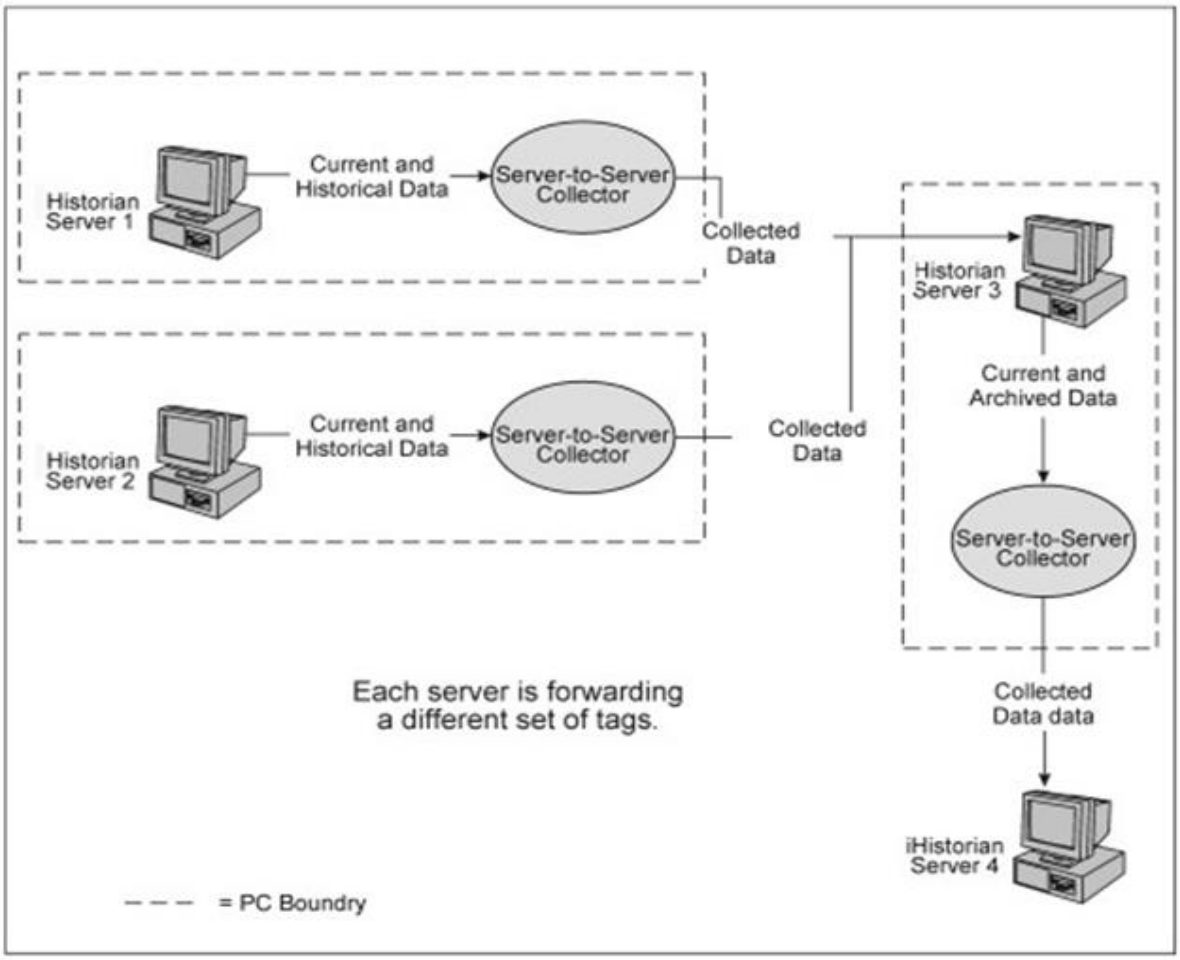


Рисунок 3.2 – Принцип збирання «сервер-сервер» для передачі даних з декількох вузлів на один вузол [15]

Потік даних у двонаправлених міжсерверних колекторах. Можна налаштувати двонаправлений збір даних, коли кожен збирач збирає свій набір тегів. На наступному рис.3.3, де показано двонаправлений збір даних між серверами.

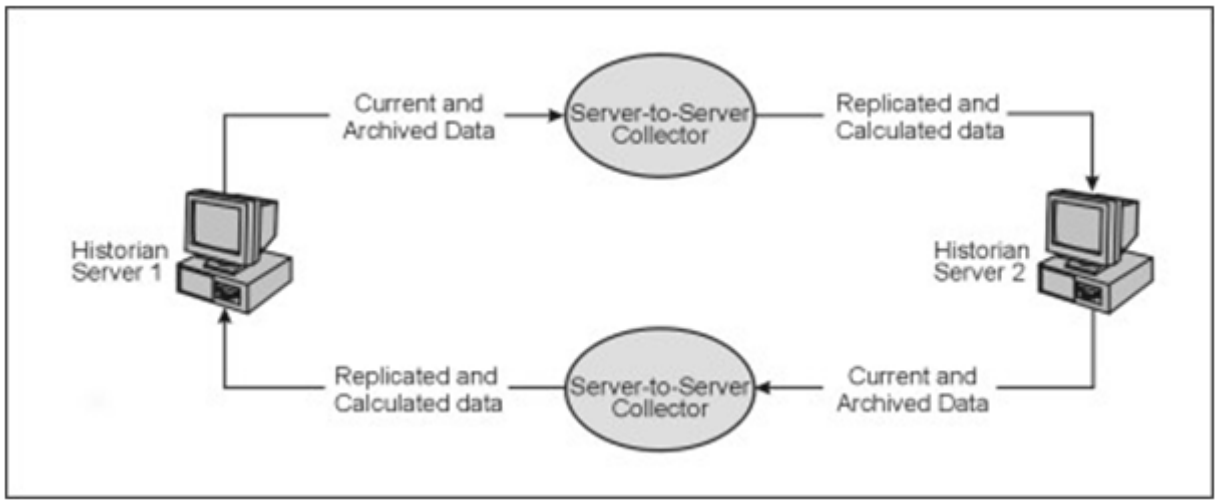


Рисунок 3.3 – Принцип збирання «сервер-сервер» для двонаправлених міжсерверних передач даних [15]

Файли **restart_webhook.sh і restart_server_collector.sh** служать для автоматичного перезапуску відповідних скриптів у випадку їх відсутності. Вони моніторять, чи запущений відповідний процес і, якщо ні, то запускають його знову.

3.2 Розробка поетапного програмного коду Webhook

3.2.1webhook_https_get_push.ua

У цьому файлі ми використовуємо фреймворк Flask для створення веб-сервера, який обробляє вхідні POST-запити вебхуків.

Flask — мікрофреймворк для вебдодатків, створений з використанням Python. Його основу складає інструментарій Werkzeug та рушій шаблонів Jinja2. Поширюється відповідно до умов ліцензії BSD[17].

Flask – це легкий веб-фреймворк для мови Python, який надає мінімальний набір інструментів для створення веб-застосунків. На ньому

можна зробити і лендинг, і багатосторінковий сайт з купою плагінів та сервісів. У Flask багато переваг, які виділяють його серед інших фреймворків:

- простий синтаксис – це все-таки Python;
- зручні шаблони – можна швидко створювати прототипи веб-додатків;
- купа інструментів для гнучкого налаштування сайтів під будь-які потреби.

За допомогою фреймворка Flask у нашому коді ми приймаємо та обробляємо вхідні запити, а також працюємо на захищеному HTTPS-з'єднанні для забезпечення безпеки передаваних даних.

Flask-додаток запускається з використанням SSL (HTTPS) за допомогою наданих сертифіката та ключа. Програма зчитує дані з лог-файлу та відправляє їх на віддалений сервер, який буде обробляти ці дані.

Flask кращий за інші фреймворки. Flask має ряд особливостей, за які його люблять веб-розробники. Давайте їх перерахуємо:

Мінімальний набір інструментів. Причому вони не нав'язують якоїсь архітектури чи жорсткої структури проектів. Розробники самі вирішують, як і що вони створюватимуть.

Гнучкість. Працюючи з Flask, програміст може вибирати лише необхідні вбудовані інструменти та підключати додаткові зовнішні, не перевантажуючи проект зайвими модулями.

Розширюваність. У Flask багато розширень та плагінів, які допомагають швидко додати нову функціональність. Наприклад, авторизацію, управління базами даних та роботу з формами.

Простота. Flask має простий синтаксис, що робить вивчення цього фреймворку більш простим, а також дозволяє швидше створювати прототипи веб-додатків.

Підтримка спільноти. Flask запустили у 2010 році, і майже з будь-якого пов'язаного з ним питання в інтернеті вже є відповіді.

Загалом, Flask начебто створений для новачків. Він нескладний, у ньому є всі необхідні базові функції та можливості для розширення. Але при цьому Flask може здатися слабким фреймворком, непридатним для великих

проектів. До речі, це теж можна виправити сторонніми плагінами та бібліотеками. Приклад надається Додатку Б.

Розглянемо детально файл **webhook_https_get_push.ua**. Цей скрипт представляє веб-додаток Flask, який приймає POST-запити на певний URL.

1) На початку файлу імпортуємо необхідні бібліотеки та модулі, такі як os, datetime, json, logging, time, threading, subprocess, requests, ssl і Flask.

2) Метод handle_webhook() є обробником вхідних POST-запитів на кореневому URL ('/') веб-додатку Flask. Якщо дані отримані у форматі JSON, вони логуються, а потім повертається відповідь з кодом успіху або помилки.

3) Функція tail_log() відповідає за моніторинг журналу, зчитуючи його построчно з кінця та передачею кожного нового рядка на обробку. file.seek(0, os.SEEK_END) переміщує вказівник поточної позиції в кінець файлу. Потім функція розпочинає безкінечний цикл while True, в якому зчитується кожен новий рядок з файлу за допомогою метода readline(). Потім відбувається кодування та декодування. Це необхідно для обробки рядків з різними кодуваннями та для запобігання помилок при читанні. Якщо в рядку є дані, він передається на обробку через оператор yield, що перетворює функцію у генератор. Це дозволяє функції повертати кожен рядок по мірі його генерації, зберігаючи стан між викликами та не завантажуючи весь файл у пам'ять одразу. Якщо в файлі немає нових даних, функція засинає на короткий час (0.1 секунди) за допомогою time.sleep(0.1), щоб не забирати процесорний час у порожньому циклі. Після цього цикл повторюється.

4) Функція post_log() призначена для відправки вмісту журналу на зазначений URL-адресу. Для кожного рядка з журналу виконується запит requests.post(url, data=line) і кожен рядок передається як дані (data) в POST-запит. Іншими словами, ця функція забезпечує безперервну відправку вмісту журналу на сервіс для подальшої обробки або аналізу.

5) Функція send_heartbeat(), періодично відправляє сигнал "heartbeat" на зазначені IP-адреси. Цей сигнал містить поточний час та інформацію про те, що сервіс, що генерує сигнал, перебуває у робочому стані. "Heartbeat"

сигнали допоможуть нам забезпечити безперервну роботу системи та виявляти будь-які проблеми або збої у обробці запитів.

6) Основний блок коду (if __name__ == '__main__'):

• Перевіряє наявність файлу журналу.

• Налаштовує Flask-додаток з використанням функції flask_logging() та вказанням маршруту для обробки вебхуків.

• Запускає окремі потоки для виконання функцій post_log та send_heartbeat.

• Запускає Flask-додаток для обробки вебхуків.

У даному коді використовуються потоки для виконання певних функцій паралельно, що дозволяє програмі ефективно обробляти кілька завдань одночасно. Ми їх створюємо за допомогою модуля threading. Кожна функція (post_log та send_heartbeat) виконується у власному потоці. Це дозволяє обробляти вхідні дані та відправляти серцеві сигнали паралельно, що збільшує продуктивність програми. Створені потоки є демонічними(daemon=True). Тобто потоки завершуються автоматично, коли основний потік програми завершується. Використання демонічних потоків у даному контексті забезпечує більш простий та зрозумілий механізм управління життєвим циклом потоків у додатку, оскільки немає необхідності явно зупиняти їх при завершенні програми. Приклад надається у Додатку Б.

3.2.2 server_collector.ua

Далі розглянемо файл server_collector.ua:

1) Імпортуємо необхідні модулі:

http.server для роботи з HTTP-сервером.

socketserver для створення власного TCP-сервера.

logging для логування інформації.

2) В строке `logging.basicConfig(filename=input_file,level=logging.INFO, format='%(message)s')` налаштовуємо систему логування за допомогою

модуля **logging.** Це дозволить нам більш детально відстежувати роботу сервера, а також аналізувати його та відлагоджувати при необхідності. Запис логів здійснюється у змінну input_file.

3) Визначаємо клас CustomHTTPRequestHandler, який успадковує(inherits) http.server.SimpleHTTPRequestHandler. Цей клас перевизначає метод do_POST(), який обробляє POST-запити. Усередині методу дані POST-запиту зчитуються з тіла запиту (self.rfile.read(content_length)), перетворюються у line і записуються у лог-файл. Потім сервер відправляє клієнту відповідне повідомлення з кодом 200 (Успішна відповідь) "POST request received and processed."

4) Створюємо екземпляр TCPServer з модуля **socketserver**, який прив'язується до порожнього хоста та вказаного порту (8080), а також визначає обробник запитів (CustomHTTPRequestHandler).

5) Викликається метод serve_forever() для початку безкінечного циклу обробки запитів. Це означає, що сервер буде активним і готовим до прийому запитів, доки його явно не зупинять.

Отже, код у файлі server_collector.ua створює TCP сервер, який слухає порт 8080 та обробляє HTTP POST запити, записуючи їх дані у лог-файл.

Приклад програмного коду надається у Додатку В

3.2.3 restart_webhook.sh

Файл **restart_webhook.sh** - це скрипт для перезапуску сервера, реалізованого у файлі webhook_https_get_push.py. Розглянемо його докладніше:

Скрипт виконує перевірку поточного стану сервера за допомогою команди **ps -ef**. Він шукає процес, пов'язаний із запущеним екземпляром **webhook_https_get_push.py.**

Якщо процес знайдений, то змінній **is_webhook_running** присвоюється значення 1, що означає, що сервер працює.

Якщо сервер не запущений (змінна is_webhook_running дорівнює 0), то виводиться повідомлення про те, що сервер не запущений, і починається процес його перезапуску.

Для перезапуску сервера використовується команда **nohup**, яка дозволяє запустити процес у фоновому режимі, і вказується шлях до файлу webhook_https_get_push.py.

Після цього виводиться повідомлення про те, що сервер перезапущений.

Цей скрипт корисний, коли необхідно забезпечити неперервну роботу сервера, перезапускаючи його у випадку зупинки або аварії. Приклад програмного коду наданий в Додатку Г.

3.2.4 restart_server_collector.sh

restart_server_collector.sh виконує аналогічні функції, пов'язані з перезапуском сервера, реалізованого у файлі server_collector.ua. Приклад програмного коду наданий в Додатку Д.

Дані, які ми отримуємо від Webhook, у кінци будуть доступні нам у файлі "/opt/data/some_project/webhook.log" у форматі JSON. Приклад того, як ці дані можуть виглядати надані в Додатку Е:

ВИСНОВКИ

У сучасному онлайн-світі з високим ступенем взаємопов'язаності нічого не може функціонувати оптимально ізольовано. Досягнення завдання (майже) завжди передбачає участь більш ніж однієї сутності. Програми електронної комерції повинні взаємодіяти з платіжними системами, платіжні системи повинні взаємодіяти з банківськими системами, банківські системи повинні взаємодіяти з рахунками клієнтів.

Здатність незалежних онлайн-систем взаємодіяти одна з одною та обмінюватися даними є основою того, що робить онлайн-сервіси цінними сьогодні.

В цих умовах постійного зростання обсягів даних та ускладнення архітектури інформаційних систем, ефективне управління продуктивністю приладів на сервері стає критично важливим завданням для забезпечення безперебійної роботи бізнес-процесів, коли важливе отримувати нові дані про зміни в цих бізнес-процесах з серверів на веб-браузер користувача. Одним з способів вирішення цього завдання є застосування механізму Webhook (веб-перехоплювача). Цієй механізм може ініціювати оновлення в системі інвентаризації та повідомляти службу доставки про необхідність надсилання продукту, чи платіжна платформа може повідомляти продавців про статус. Визначення цього моменту, коли відбувається подія, яка запускає інтеграцію веб-перехоплювача, що має вплив на підвищення продуктивності сервера, може бути визначена автоматично за рахунок спеціальних команд.

Інформація керує Інтернетом, а отримання інформації в режимі реального часу робить онлайн-сервіси високоефективними та оперативно реагують на потреби клієнтів. Технологія Webhook пропонує нескладний спосіб обміну інформацією між онлайн-платформами як реального часу. Один запит веб-перехоплювача також може бути розподілений за декількома

місцями призначення, яким потрібна інформація, в процесі відомий як розгалуження

Це дозволяє вихідним системам взаємодіяти з великою кількістю програм та краще поширювати інформацію в мережі.

Таким чином, Webhook пропонують простий та зручний спосіб отримувати оновлення повідомлень про дії в Інтернеті в режимі реального часу, дозволяючи швидко реагувати на ці повідомлення та діяти відповідно до них з мінімальними зусиллями. Крім того, веб-перехоплювачі є безпечним способом обміну інформацією з іншими веб-сервісами, такими як маркетингові або аналітичні платформи, оскільки вони не вимагають від розробників або програмістів надання користувачам облікових даних для доступу або конфіденційних даних. Визначення цього моменту, коли відбувається подія, яка запускає інтеграцію веб-перехоплювача, що має вплив на підвищення продуктивності сервера, може бути визначена автоматичне за рахунок спеціальних команд.

Webhook має значний потенціал в різноманітних випадках використання, що підтверджується його важливістю на платформах електронної комерції: події веб-перехоплювача використовуються для оновлення статусу замовлення, відстеження постачання та управління запасами.

В результаті проведених досліджень

1. Був проведений аналіз передумов щодо впровадження моніторингу продуктивності приладів на сервері за рахунок механізму Webhook для отримання миттєвих сповіщень.

2. Досліджено методи застосування системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері на основі механізму Webhook для отримання миттєвих сповіщень..

3 Розроблені практичні рекомендації щодо впровадження системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері на основі механізму Webhook для отримання миттєвих сповіщень.

Результати дослідження. Підвищення ефективності моніторингу продуктивності приладів на сервері через розробку та впровадження системи обробки та управління сповіщеннями.

Наукова новизна. Наукова новизна цього дослідження полягає оцінці ефективності та доцільності розробки системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері з використанням сучасних технологій.

Практична значущість результатів дослідження - полягає в розробці практичних рекомендацій щодо застосування системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері з використанням сучасних технологій.

Апробація результатів надається в 1 статті, 1 тезисах в рецензованих наукових журналах і виданнях.

Матеріали були опубліковані:

В статті:

1. Почтовик А.Р. ЗАСТОСУВАННЯ ЗАСОБІВ WEBHOOK У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ СЕРВЕРА// Ю.І. Катков// Наукові записки Державного університету телекомунікацій №3, 2023, Подано до друку. https://journals.dut.edu.ua/index.php/sciencenotes/issue/archive

В тезисах:

1. Почтовик А.Р. // V Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез 18 квітня 2024. – К.: ДУІКТ, https://duikt.edu.ua/ua/news-1-570-12409-v-mizhnarodna-naukovo-tehnichna-konferenciya-suchasniy-stan-ta-perspektivi-rozvitku-iot_kafedra-inzhenerii-programnogo-zabezpechennya-avtomatizovanih-sistem
2. Почтовик А. Р. ЗАСТОСУВАННЯ ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ

СПОВІЩЕННЯМ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ СЕРВЕРА// IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті» Збірник тез 15 травня 2024. – К.: ДУІКТ, https://duikt.edu.ua/ua/news-1-576-12601-v-duikt-vidbudetsya-vseukrainska-konferenciya-prisvyachena-shtuchnomu-intelektu-yak-vzyati-uchast_kafedra-shtuchnogo-intelektu

3. Почтовик А.Р. ПРОБЛЕМИ ЗАСТОСУВАННЯ ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ СЕРВЕРА // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ» Збірник тез 24 квітня 2024. – К.: ДУІКТ, https://duikt.edu.ua/ua/news-1-1009-12379-vseukrainska-naukovo-tehnichna-konferenciya-zastosuvannya-programnogo-zabezpechennya-v-informaciyno-komunikaciynih-tehnologiyah_kafedra-inzhenerii-programnogo-zabezpechennya

ПЕРЕЛІК ПОСИЛАНЬ

1. What is a Webhook? How They Work With Examples [Електронний ресурс] / режим доступу: https://www.ayrshare.com/what-is-a-webhook-how-they-work-with-examples/ (date of access: 02.04.2024).
2. What is a Webhook? Understanding the difference and how to use them [Електронний ресурс] / режим доступу: https://cyclr.com/blog/what-is-a-webhook (date of access: 02.04.2024).
3. What is a webhook? [Електронний ресурс] / режим доступу: https://www.redhat.com/en/topics/automation/what-is-a-webhook (date of access: 02.04.2024).
4. What are webhooks and their use in server-side tracking Examples [Електронний ресурс] / режим доступу: https://stape.io/blog/what-are-webhooks-and-their-use-in-server-side-tracking (date of access: 02.04.2024).
5. How To Effectively Monitor Server Performance Examples [Електронний ресурс] / режим доступу: https://theqalead.com/test-management/server-performance-monitoring/ (date of access: 02.04.2024).
6. Callback function Examples [Електронний ресурс] / режим доступу: https://developer.mozilla.org/ru/docs/Glossary/Callback_function (date of access: 02.04.2024)
7. What is a Push Notification Service? [Електронний ресурс] / режим доступу: https://aws.amazon.com/what-is/push-notification-service/(date of access: 02.04.2024).
8. How to Take Control of Your Webhook Reliability Examples [Електронний ресурс] / режим доступу: https://hookdeck.com/webhooks/guides/taking-control-of-your-webhook-reliability (date of access: 02.04.2024).
9. What are the benefits and drawbacks of webhooks for real-time data delivery? Examples [Електронний ресурс] / режим доступу:

https://www.linkedin.com/advice/0/what-benefits-drawbacks-webhooks-real-time-data (date of access: 02.04.2024).

10. How to Use an API: Just the Basics/ [Електронний ресурс] / режим доступу: https://technologyadvice.com/blog/information-technology/how-to-use-an-api/ (date of access: 02.04.2024).

11. HTTP (HyperText Transfer Protocol) / [Електронний ресурс] / режим доступу:https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP(date of access: 02.04.2024).

12. Server Management / [Електронний ресурс] / режим доступу: https://pinkelephant.co.uk/it-service-management/operating-system-licence-management-procurement-advice/ (date of access: 02.04.2024).

13. Архітектурні стилі API / [Електронний ресурс] / режим доступу: https://infdev.com.ua/docs/standards/api-architecture-styles/(date of access: 02.04.2024).

14. Що таке push-сповіщення та як правильно їх використовувати? / [Електронний ресурс] / режим доступу: https://wezom.com.ua/ua/blog/push-notification (date of access: 02.04.2024).

15. Overview of the Server-to-Server Collector / [Електронний ресурс] / режим доступу: https://www.ge.com/digital/documentation/historian/version2024/c_server_to_server_collector_overview.html (date of access: 02.04.2024).

16. What is a Webhooks Push-Styled API and How Does It Work? / [Електронний ресурс] / режим доступу: https://www.mulesoft.com/api-university/what-webhooks-push-styled-api-and-how-does-it-work (date of access: 02.04.2024).

17. Фреймворк Flask: как он работает и зачем нужен / [Електронний ресурс] / режим доступу: https://skillbox/media/code/freymvork-flask-kak-on-rabotaet-i-zachem-nuzhen/ (date of access: 02.04.2024).

18. .Штучний інтелект [Electronic resource]. – URL: https://uk.wikipedia.org/wiki/%D0%A8%D1%82%D1%83%D1%87%D0%BD%D

0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D1%82 (дата звернення: 16.04.2024).

19. Falcor: One model everywhere / [Електронний ресурс] / режим доступу: https://netflix.github.io/falcor (date of access: 02.04.2024).

20. Websocket explained / [Електронний ресурс] / режим доступу: https://http.dev/ws (date of access: 02.04.2024).

21. TCP/IP Protocol Architecture Model – How Does it Work? / [Електронний ресурс] / режим доступу: https://geekflare.com/tcp-ip-protocol-architecture-model/ (date of access: 02.04.2024).

22. Introduction to Socket API / [Електронний ресурс] / режим доступу: https://docs.socket.dev/reference/introduction-to-socket-api (date of access: 02.04.2024).

23. Mobile push notifications: everything you need to know / [Електронний ресурс] / режим доступу: https://www.getvero.com/resources/mobile-push-notifications/ (date of access: 02.04.2024).

24. What is Client-Server Architecture? Explained in Detail / [Електронний ресурс] / режим доступу: https://www.theknowledgeacademy.com/blog/client-server-architecture/ (date of access: 02.04.2024).

25. Understanding Short Polling, Long Polling, Server Sent Events and Web Sockets / [Електронний ресурс] / режим доступу: https://dev.to/shameel/understanding-short-polling-long-polling-server-sent-events-and-web-sockets-20kh (date of access: 02.04.2024).

26. What is Server-Sent Events (SSE) and how to implement it? / [Електронний ресурс] / режим доступу: https://medium.com/deliveryherotechhub/what-is-server-sent-events-sse-and-how-to-implement-it-904938bffd73 (date of access: 02.04.2024).

27. Real-time communication / [Електронний ресурс] / режим доступу: https://blog.bibekkakati.me/short-polling-vs-long-polling (date of access: 02.04.2024).

28. Artificial intelligence and data protection[Electronic resource]. – URL: https://edoc.coe.int/en/artificial-intelligence/8254-artificial-intelligence-and-data-protection.html (дата звернення: 16.04.2024).

29. Artificial Intelligence and Data Protection[Electronic resource]. – URL: https://towardsdatascience.com/artificial-intelligence-and-data-protection-62b333180a27 (дата звернення: 16.04.2024).

30. Архітектурні стилі API / [Електронний ресурс] / режим доступу: https://infdev.com.ua/ru/docs/standards/api-architecture-styles/ (date of access: 02.04.2024).

31. Push Notifications Explained / [Електронний ресурс] / режим доступу: https://www.airship.com/resources/explainer/push-notifications-explained/ (date of access: 02.04.2024).

32. Що таке вебхуки та їх використання в відстеженні на стороні сервера / [Електронний ресурс] / режим доступу: https://webcache.googleusercontent.com/search?q=cache:https://stape.io/ua/blog/scho-take-webhooks-ta-ih-vikoristannya-v-vidstezhenni-na-storoni-servera (date of access: 02.04.2024).

33. What is an API (Application Programming Interface)? / [Електронний ресурс] / режим доступу: https://aws.amazon.com/what-is/api/ (date of access: 02.04.2024).

34. What is API Integration? / [Електронний ресурс] / режим доступу: https://www.hcltech.com/knowledge-library/what-is-api-integration (date of access: 02.04.2024).

35. How do APIs work? / [Електронний ресурс] / режим доступу: https://tray.io/blog/how-do-apis-work (date of access: 02.04.2024).

36. What Are Webhooks And How Do They Work / [Електронний ресурс] / режим доступу: https://hookdeck.com/webhooks/guides/what-are-webhooks-how-they-work (date of access: 02.04.2024).

37. Webhooks vs API: Understanding the Key Differences / [Електронний ресурс] / режим доступу: https://mailchimp.com/resources/webhook-vs-api/ (date of access: 02.04.2024).

38. API vs Webhook vs WebSocket / [Електронний ресурс] / режим доступу: https://nonamesecurity.com/learn/api-vs-webhook-vs-websocket/ (date of access: 02.04.2024).

39. Webhook triggers / [Електронний ресурс] / режим доступу: https://api.slack.com/automation/triggers/webhook (date of access: 02.04.2024).

40. Webhooks Explained: What is a Webhook + Examples / [Електронний ресурс] / режим доступу: https://prismic.io/blog/what-is-a-webhook (date of access: 02.04.2024).

41. Webhooks explained simply: What they do and how they work / [Електронний ресурс] / режим доступу: https://www.techtarget.com/searchapparchitecture/tip/Webhooks-explained-simply-and-how-they-differ-from-an-API (date of access: 02.04.2024).

42. Slack webhooks are deprecated. Is there an official Netlify Slack app in the works? / [Електронний ресурс] / режим доступу: https://answers.netlify.com/t/slack-webhooks-are-deprecated-is-there-an-official-netlify-slack-app-in-the-works/25305 (date of access: 02.04.2024).

43. API vs Webhook vs WebSocket / [Електронний ресурс] / режим доступу: https://nonamesecurity.com/learn/api-vs-webhook-vs-websocket/ (date of access: 02.04.2024).

44. Webhooks vs REST APIs: when to use one over the other / [Електронний ресурс] / режим доступу: https://www.merge.dev/blog/rest-api-vs-webhooks (date of access: 02.04.2024).

45. Webhook security: a hands-on guide / [Електронний ресурс] / режим доступу: https://planetscale.com/blog/securing-webhooks (date of access: 02.04.2024).

46. What Are Webhooks and Why Would You Use Them? / [Електронний ресурс] / режим доступу: https://www.make.com/en/blog/what-are-webhooks (date of access: 02.04.2024).

47. Why developers love webhooks (and we do too!) / [Електронний ресурс] / режим доступу: https://www.linkedin.com/pulse/why-developers-love-webhooks-we-do-too-ashwin-bhatnagar-clrgc (date of access: 02.04.2024).

48. Webhook vs. API: differences (and when to use each) / [Електронний ресурс] / режим доступу: https://zapier.com/blog/webhook-vs-api/ (date of access: 02.04.2024).

49. What is a webhook: How they work and how to set them up / [Електронний ресурс] / режим доступу: https://www.getvero.com/resources/webhooks/ (date of access: 02.04.2024).

50. Webhooks Use Case: Configuring Zone Entry and Exit Events / [Електронний ресурс] / режим доступу: https://www.juniper.net/documentation/us/en/software/mist/automation-integration/topics/task/webhooks-usecase-configuring-zone-entry-exit-gui.html (date of access: 02.04.2024).

51. Understanding a Web Server and Types of Web Servers/ [Електронний ресурс] / режим доступу: https://www.milesweb.in/blog/hosting/web-server-types-web-servers/ (date of access: 02.04.2024).

52. What's the Difference Between a Web Server and an Application Server? / [Електронний ресурс] / режим доступу: https://aws.amazon.com/compare/the-difference-between-web-server-and-application-server/ (date of access: 02.04.2024).

ДОДАТОК А

Приклад програмного коду webhook_install.sh

```
#!/bin/bash

set -euo pipefail

# Function to display error messages and exit
display_error() {
    echo "Error: $1" >&2
    exit 1
}

# Check if script is running with root privileges
if [[ $EUID -ne 0 ]]; then
    display_error "This script must be run as root."
fi

# Check if wget is installed
if ! command -v wget &> /dev/null; then
    display_error "wget is required but not installed. Please
install wget and run the script again."
fi

echo "[-] Installing webhook.."
wget
https://github.com/adnanh/webhook/releases/download/2.6.11/webhook-
linux-amd64.tar.gz || display_error "Failed to download webhook
binary."
tar -xvf webhook-linux-amd64.tar.gz || display_error "Failed to
extract webhook binary."
mv webhook-linux-amd64/webhook /usr/local/bin || display_error
"Failed to move webhook binary to /usr/local/bin."
rm -rf webhook-linux-amd64* || display_error "Failed to clean up
downloaded files."
echo "[ok] Webhook installed."
```

```
# Validate and set directory paths
scripts_dir="/opt/scripts"
hooks_dir="/opt/hooks"
hooksjson_location="$hooks_dir/hooks.json"

# Create directories and set permissions
mkdir -p "$scripts_dir" "$hooks_dir" || display_error "Failed to
create directories."
chown -R "$SUDO_USER:$SUDO_USER" "$scripts_dir" "$hooks_dir" ||
display_error "Failed to set ownership of directories."

# Set up JSON configuration file
cat > "$hooksjson_location" <<EOF
[
   {
    "id": "server01",
    "execute-command": "$scripts_dir/deploy.sh",
    "working-directory": "$scripts_dir",
    "pass-arguments-to-command":
    [
      {
        "source": "payload",
        "name": "object_attributes.status"
      },
      {
        "source": "payload",
        "name": "project.name"
      },
      {
        "source": "payload",
        "name": "user.username"
      }
    ]
   }
]
EOF
```

```
echo "[-] Installing webhook in systemd.."
file_location="/etc/systemd/system/webhook.service"

# Set up systemd service
cat > "$file_location" <<EOF
[Unit]
Description=Webhook Server for CI
ConditionPathExists=/usr/local/bin/webhook
After=network.target

[Service]
Type=simple
WorkingDirectory="$hooks_dir"
ExecStart=/usr/local/bin/webhook -ip 0.0.0.0 -port 9225 -hooks
"$hooksjson_location" -verbose
Restart=on-failure
PrivateTmp=true

[Install]
WantedBy=default.target
EOF

# Reload systemd and start the webhook service
systemctl daemon-reload || display_error "Failed to reload
systemd."
systemctl start webhook.service || display_error "Failed to start
webhook service."
systemctl enable webhook.service || display_error "Failed to enable
webhook service."

echo "[ok] Webhook installed in systemd."
```

ДОДАТОК Б

Приклад програмного коду webhook_https_get_push.ua

```
import os
from datetime import datetime
import json
import logging
import time
import threading
import requests
import ssl
import sys
from flask import Flask, request, jsonify

## Config
logfile = "/opt/data/webhook_log/webhook.log"
IP_to_push_to_1 = ""
IP_to_push_to_2 = ""
cert_https = "/opt/app/webhook/https_crt_key/server.crt"
key_https = "/opt/app/webhook/https_crt_key/server.key"


################
################

def check_file_exists(file):
    if os.path.exists(file):
        return True


def flask_logging():
    logging.basicConfig(filename=logfile,                level=logging.INFO,
format='%(asctime)s - %(message)s')
    # Suppress default Flask logging
    log = logging.getLogger('werkzeug')
```

```
    log.setLevel(logging.ERROR)


app = Flask(__name__)


@app.route('/', methods=['POST'])
def handle_webhook():
    """
    Get the events from webhook
    """
    data = request.get_json()
    if data is None:
        return jsonify({'error': 'Invalid JSON'}), 400
    logging.info("Received data: %s", json.dumps(data))
    return jsonify({'success': 'Webhook handled'}), 200


def tail_log(log_file_path):
    """
    tail the log
    """
    with open(log_file_path, 'r') as file:
        file.seek(0, os.SEEK_END)
        while True:
            line = file.readline()
            if line:
                encoded_line = line.encode("ascii", "ignore")
                decoded_line = encoded_line.decode()
            else:
                time.sleep(0.1)
                continue
            yield decoded_line


def post_log(log_file, url):
    '''
```

```
    post log to poller
    '''
    lines = tail_log(log_file)
    for line in lines:
        response = requests.post(url, data=line)

def send_heartbeat(url):
    """
    Send a heartbeat signal every 5 minutes.
    """
    while True:
        # Get the current date and time
        now = datetime.now()
        # Format the date and time string
        timestamp = now.strftime("%Y-%m-%d %H:%M:%S,%f")[:-3]
        # Create a dictionary to send as the heartbeat
        heartbeat = {
            "timestamp": timestamp,
            "message": 'Received data: ' + json.dumps({"heartbeat": "alive"})
        }
        # Convert the heartbeat dictionary to a JSON string
        heartbeat_json = json.dumps(heartbeat)
        # Send the heartbeat

        response = requests.post(url, data=heartbeat_json)
        # Wait for 5 minutes
        time.sleep(300)


if __name__ == '__main__':
    while not check_file_exists(logfile):
        logging.error(f"File {logfile} not found!")
        time.sleep(3600)

    flask_logging()

    # Start the post_log function in a separate thread
```

```
    t = threading.Thread(target=post_log, args=(logfile, IP_to_push_to_1),
daemon=True)
    t.start()

    # Start the post_log function in a separate thread
    t = threading.Thread(target=post_log, args=(logfile, IP_to_push_to_2),
daemon=True)
    t.start()

    # Start the send_heartbeat function in a separate thread
    t = threading.Thread(target=send_heartbeat, args=(IP_to_push_to_1,),
daemon=True)
    t.start()

    # Start the send_heartbeat function in a separate thread
    t = threading.Thread(target=send_heartbeat, args=(IP_to_push_to_2,),
daemon=True)
    t.start()
    t.join()

    # Run the Flask application
    ctx = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
    ctx.load_cert_chain(cert_https, key_https)

    app.run(host='0.0.0.0', port=443, ssl_context=ctx)
```

ДОДАТОК В

Приклад програмного коду server_collector.ua

server_collector.ua

```
import http.server
import socketserver
import logging

## Config
input_file="/opt/data/some_project/webhook.log"

#####

# Configure logging
logging.basicConfig(filename=input_file,                    level=logging.INFO,
format='%(message)s')

class CustomHTTPRequestHandler(http.server.SimpleHTTPRequestHandler):
        def do_POST(self):
              # You can handle the POST request here
              content_length = int(self.headers['Content-Length'])
              post_data = self.rfile.read(content_length)
              post_data_str = post_data.decode()
              logging.info(post_data_str)
              # Send a response header
              self.send_response(200)
              self.send_header('Content-Type', 'text/html')
              self.end_headers()
              # Send a response body
              response = "POST request received and processed."
              self.wfile.write(response.encode())

# Start the server
```

```
port = 8080
handler = CustomHTTPRequestHandler
httpd = socketserver.TCPServer(("", port), handler)
print(f"Serving HTTP on port {port} (http://localhost:{port}/) ...")
httpd.serve_forever()
```

ДОДАТОК Г

Приклад програмного коду restart_webhook.sh

```
#!/bin/sh
## Restart the webhook_https_get_push.py

## Start the Webhook Collector if NOT running
is_webhook_running=`ps -ef | grep -i python3 | grep -i
webhook_https_get_push.py| grep -iv grep | wc -l`

if [[ $is_webhook_running == 1 ]]; then
   echo "webhook_https_get_push.py is running"
else
   echo "webhook_https_get_push.py is NOT running. Restarting"
   nohup /opt/app/python396/bin/python3.9
/opt/app/webhook/webhook_https_get_push.py > /dev/null 2>&1&
fi
```

ДОДАТОК Д

Приклад програмного коду restart_server_collector.sh

```
#!/bin/sh
## Restart the "server_collector"

is_server_collector_running=`ps -ef | grep -i python3 | grep -i
server_collector.py | grep -iv grep | wc -l`

if [[ $is_server_collector_running == 1 ]]; then
   echo "server_collector.py is running"
else
   echo "server_collector.py is NOT running. Restarting"
   nohup /opt/app/python396/bin/python3.9
/opt/app/pashe/http_server_collector.py > /dev/null 2>&1&
fi
```

ДОДАТОК Е

Приклад даних, отриманих від Webhook

2023-09-10 17:24:31 - {'objectName': 'some_objectName', 'objectId': 'some_objectId', 'policyName': ' Packet Loss alert test', 'policyId': '877643hhh', 'alertState': 'raised', 'processedTimestamp': '1693921799999', 'customer': 'bmw', 'from_city': 'bolinas', 'from_country': 'us', 'from_edge': 'c8500-clb5', 'from_link': 'some_link', 'to_city': 'bolinas', 'to_country': 'us', 'to_edge': 'aiu71x-clb', 'to_link': 'inet_int', 'triggeredValue': '7.143', 'value': '4.0'}

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (презентація)

1

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

ДИПЛОМНА РОБОТА
на ступінь вищої освіти бакалавр
із спеціальності 122 Комп'ютерні технології

«РОЗРОБКА СИСТЕМИ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМИ ЩОДО ПРОДУКТИВНОСТІ ПРИЛАДІВ НА СЕРВЕРІ»

Виконав: студент 4 курсу, групи КНД-42
Почтовик Аліна Романівна
Керівник: д.т.н., доцент, Катков Ю.І.

Київ - 2024

2

ЗАГАЛЬНА ХАРАКТЕРИСТИКА ДИПЛОМНОЇ РОБОТИ

Тема	Розробка системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері
Мета роботи	Підвищення ефективності моніторингу продуктивності приладів на сервері через розробку та впровадження системи обробки та управління сповіщеннями.
Наукове завдання	Оцінка ефективності та доцільності розробки системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері з використанням сучасних технологій.
Об'єкт дослідження	Процес моніторингу продуктивності приладів на сервері системою обробки та управління сповіщеннями, спрямований на підвищення ефективності управління продуктивністю та пошуку несправних приладів на сервері.
Предмет дослідження	Система обробки та управління сповіщеннями щодо продуктивності приладів на сервері.

3

Актуальність теми

В умовах постійного зростання обсягів даних та ускладнення архітектури інформаційних систем, ефективне управління продуктивністю приладів на сервері стає критично важливим завданням для забезпечення безперебійної роботи бізнес-процесів, коли важливе отримувати нові дані про зміни в цих бізнес-процесах з серверів на веб-браузер користувача. Одним з способів вирішення цього завдання є застосування механізму Webhook (веб-перехоплювача).

4

Наукова новизна

Наукова новизна цього дослідження полягає оцінці ефективності та доцільності розробки системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері з використанням сучасних технологій.

5

Аналіз способів сповіщення пристроїв та серверів (модель клієнт/сервер)

Способи сповіщення — це канали зв'язку, які використовуються для сповіщення людей про важливу інформацію, події або оновлення. До цих методів відносяться e-mail (електронна пошта), SMS, сокети, push-сповіщення, прокручування тикерів та багато іншого.

Push-повідомлення - це повідомлення, яке з'являється на головному екрані мобільного пристрою без необхідності входу користувача до програми або використання пристрою на даний момент.

Електронна пошта (e-mail) - це обмін повідомленнями, що зберігаються на комп'ютері, від одного користувача одному або декільком одержувачам через Інтернет. Це швидкий, недорогий та доступний спосіб спілкування для бізнесу чи особистого використання.

Сокети та API сокети - використовуються для надсилання повідомлень по мережі. Вони забезпечують форму міжпроцесної взаємодії (IPC).

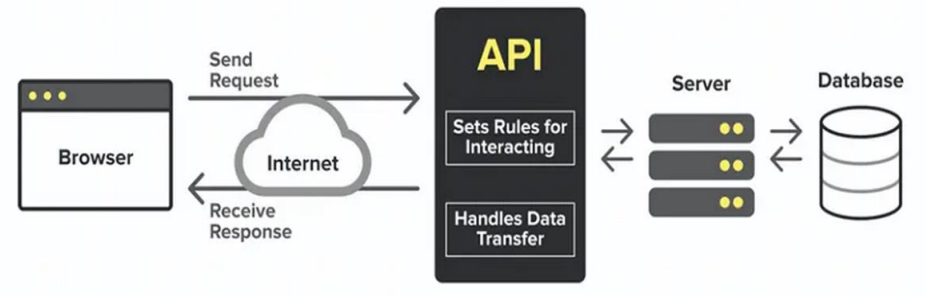
SMS - "Служба коротких повідомлень" широко відомий як текстові повідомлення. Це може надсилати між телефонами лише текстові повідомлення довжиною до 160 символів.

6

Аналіз інструментів, які дозволяють програмам надавати інформацію в реальному часі іншим програмам при виникненні певної події

Для усунення недоліків застосування сповіщень, треба застосовувати інструменти, які дозволяють програмам надавати інформацію в реальному часі іншим програмам при виникненні певної події. Традиційне для роботи системи обробки та управління сповіщенням серверів застосовується інтерфейс **API** (Application Programming Interface)

Архітектурні стилі API- інтерфейсів – це REST, SOAP, GraphQL, WebSocket, gRPC, Falcor API, Webhook



7

Архітектурні стилі API- інтерфейсів

REST API – це архітектурний стиль для створення веб-служб. Він простий у використанні, працює швидше, ніж звичайні веб-сервіси, і може повертати результати у різних форматах даних. REST API масштабуються завдяки здатності кешувати дані, що знижує навантаження на сервер, а також можуть використовувати шифрування SSL для передачі даних, що усуває загрозу компрометації при передачі.

SOAP API - це специфікація протоколу обміну повідомленнями для обміну структурованою інформацією під час реалізації веб-сервісів у комп'ютерних мережах. Він заснований на XML. Специфіка SOAP полягає у форматі обміну даними**.**

GraphQL - це відкрита мова запитів (це те, що означає QL) та обробки даних для API і середовище виконання для виконання цих запитів з існуючими даними. Він використовується для завантаження даних із сервера на клієнт — це спосіб отримати дані з API у вашу програму. Також він забезпечує декларативну вибірку даних, коли клієнт може вказати, які дані йому потрібні з API.

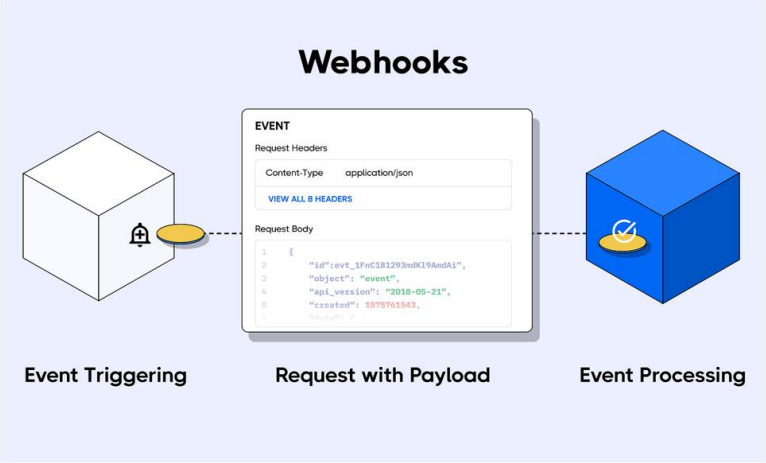
gRPC - це система віддаленого виклику процедур з відкритим кодом інфраструктури RPC Stubby. Клієнтський додаток gRPC може виконувати прямі запити до серверного додатку. І клієнт, і сервер використовують загальний інтерфейс. gRPC був розроблений спеціально для того, щоб дозволити розробникам створювати високопродуктивні API для мікросервісної архітектури у розподілених центрах обробки даних.

Falcor API - представляє віртуальний рівень, який можна використовувати для відображення зовнішніх запитів на серверні служби. Falcor особливо добре підходить для програм із комплексними вимогами до отримання даних та потребами оновлення в реальному часі

8

Характеристика Webhook

Webhook дозволяє веб-серверам надсилати повідомлення або дії в режимі реального часу іншим системам під час виникнення певних подій на веб-сторінці або у додатку. Замість того, щоб надсилати інформацію у відповідь на запит іншої програми, веб-перехоплювач відправляє інформацію або виконує певну функцію у відповідь на тригер — наприклад, час доби, натискання кнопки або отримання форми. Оскільки програма, яка надсилає дані, ініціює передачу, веб-перехоплювачі часто називають «зворотними API».



9

Переваги застосування Webhook:

- Усувають потребу опитування
- Швидко налаштовуються
- Автоматизують передачу даних
- Забезпечують обмін у формі коротких сповіщень про специфічні корисні навантаження
- Оновлення в режимі реального часу
- Можуть видаляти повідомлення.
- Ефективність

10

Процес налаштування та використання Webhook

Процес налаштування та використання Webhook включає кілька етапів: конфігурація, подія-тригер, повідомлення та дія.

Конфігурація. Наприклад,

крок 1 - можна налаштувати URL-адресу веб-перехоплювача на сервері для отримання повідомлення про успішну публікацію запланованого повідомлення на Facebook.

Крок 2 - Webhook сервера отримує це повідомлення.

Крок 3 - потім запускає процес оновлення деяких даних та панелі керування користувача.

Крок 4 - кінцевий користувач побачить на панелі керування нові дані, що повідомлення було успішно опубліковано.

Подія-тригер. Тригерні сповіщення — це push-сповіщення, які автоматично активуються, коли на вашому веб-сайті відбувається певна подія користувача.

Повідомлення та дія. Webhook спрацьовує, коли на вашому сайті, у CRM, чат-боті чи інших системах відбувається подія. Запускається тригер Webhook - це автоматичний тип тригера, який прослуховує певний тип даних, подібно до тригерів подій. Наприклад, людина створює замовлення в інтернет-магазині → система відправляє Webhook у додаток власника → додаток повідомляє власника та відправляє йому оцінку.

11

Рекомендації щодо використання Webhook

1. Треба використовувати HTTPS та шифрування, щоб убезпечити зв'язок через веб-перехоплювач та захистити дані від несанкціонованого доступу чи зміни.
2. Використовуйте повтори та підтвердження, щоб забезпечити доставку та обробку веб-перехоплювачів, а також коректне обробляти помилки та збої.
3. Використовуйте фільтри та перехоплювачі, щоб вказати події та дані, які хочете надсилати та отримувати, та уникайте надсилання непотрібних або конфіденційних даних.
4. Використовуйте журнали та моніторинг для відстеження та аналізу активності та продуктивності веб-перехоплювачів, а також для виявлення та усунення будь-яких проблем та аномалій.

12

Приклади застосування Webhook

Webhook в Shopify(платформа, яка допомагає створювати та керувати інтернет-магазинами): Shopify надає webhooks для сповіщення зовнішніх систем про події, пов’язані з онлайн-магазинами, такі як нові замовлення, виконання замовлень або оновлення клієнтів. Це дозволяє розробникам створювати власні інтеграції з Shopify та автоматизувати завдання, такі як обробка замовлень або оновлення інформації про залишки товарів.
Webhook в GitHub: GitHub використовує webhooks для сповіщення зовнішніх систем про події, які відбуваються в репозиторіях. Це можуть бути нові коміти, запити на отримання(pull requests) або якісь проблеми. Розробники можуть використовувати ці сповіщення для запуску автоматизованих робочих процесів, таких як безперервна інтеграція, розгортання або відстеження проблем.
Webhook в Телеграм (багатоплатформовий месенджер) - це URL, який ви надаєте телеграму, і за яким телеграм буде відправляти оновлення у форматі JSON щоразу, коли вашому боту надходить повідомлення або відбувається інша подія.

13

Результати проведених досліджень

1. Був проведений аналіз передумов щодо впровадження моніторингу продуктивності приладів на сервері за рахунок механізму Webhook для отримання миттєвих сповіщень.
2. Досліджено методи застосування системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері на основі механізму Webhook для отримання миттєвих сповіщень..
3. Розроблені практичні рекомендації щодо впровадження системи обробки та управління сповіщеннями щодо продуктивності приладів на сервері на основі механізму Webhook для отримання миттєвих сповіщень

Апробація результатів досліджень.

14

В статті:
Почтовик А.Р. ЗАСТОСУВАННЯ ЗАСОБІВ WEBHOOK У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ СЕРВЕРА// Ю.І. Катков// Наукові записки Державного університету телекомунікацій №3, 2023, Подано до друку.
https://journals.dut.edu.ua/index.php/sciencenotes/issue/archive
В тезисах:
Почтовик А.Р. // V Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез 18 квітня 2024. – К.: ДУІКТ, https://duikt.edu.ua/ua/news-1-570-12409-v-mizhnarodna-naukovo-tehnichna-konferenciya-suchasniy-stan-ta-perspektivi-rozvitku-iot_kafedra-inzhenerii-programnogo-zabezpechennya-avtomatizovanih-sistem

Почтовик А. Р. ЗАСТОСУВАННЯ ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ СЕРВЕРА// IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті» Збірник тез 15 травня 2024. – К.: ДУІКТ, https://duikt.edu.ua/ua/news-1-576-12601-v-duikt-vidbudetsya-vseukrainska-konferenciya-prisvyachena-shtuchnomu-intelektu-yak-vzyati-uchast_kafedra-shtuchnogo-intelektu
Почтовик А.Р. ПРОБЛЕМИ ЗАСТОСУВАННЯ ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ У СИСТЕМАХ ОБРОБКИ ТА УПРАВЛІННЯ СПОВІЩЕННЯМ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ СЕРВЕРА // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ» Збірник тез 24 квітня 2024. – К.: ДУІКТ, https://duikt.edu.ua/ua/news-1-1009-12379-vseukrainska-naukovo-tehnichna-konferenciya-zastosuvannya-programnogo-zabezpechennya-v-informaciyno-komunikaciynih-tehnologiyah_kafedra-inzhenerii-programnogo-zabezpechennya

15

Висновки

В умовах постійного зростання обсягів даних та ускладнення архітектури інформаційних систем, ефективне управління продуктивністю приладів на сервері стає критично важливим завданням для забезпечення безперебійної роботи бізнес-процесів, коли важливе отримувати нові дані про зміни в цих бізнес-процесах з серверів на веб-браузер користувача. Одним з способів вирішення цього завдання є застосування механізму Webhook (веб-перехоплювача). Цієй механізм може ініціювати оновлення в системі інвентаризації та повідомляти службу доставки про необхідність надсилання продукту, чи платіжна платформа може повідомляти продавців про статус. Визначення цього моменту, коли відбувається подія, яка запускає інтеграцію веб-перехоплювача, що має вплив на підвищення продуктивності сервера, може бути визначена автоматично за рахунок спеціальних команд.

Webhook має значний потенціал в різноманітних випадках використання, що підтверджується його важливістю на платформах електронної комерції