МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія (кафедра) комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

РОЗШИРЕННЯ ДЛЯ GOOGLE CHROME З ІНТЕГРАЦІЄЮ ЗОВНІШНІХ API ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ ТА АНАЛІЗУ ДАНИХ

Виконав: студент групи 1П-21
Спеціальності
121 Інженерія програмного забезпечення
Владислав ШЕВЧЕНКО
Керівник:
Вікторія НЕМЧЕНКО

Черкаси 2025

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

Кафедра комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри КІ та ІТ

_____ Владислав ХОТУНОВ
(підпис)
«_____»_____2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Шевченку Владиславу Олеговичу

1. Тема кваліфікаційної роботи «Розширення для Google Chrome з інтеграцією зовнішніх API для автоматизації збору та аналізу даних»

Керівник роботи Немченко Вікторія Юріївна, викладача

затверджені наказом закладу вищої освіти від «07» жовтня 2024 року № 68у.
2. Строк подання студентом кваліфікаційної роботи 02.06.2025
3. Вихідні дані до кваліфікаційної роботи Google Chrome API, JavaScript, HTML, CSS, Manifest V3, REST API, OAuth 2.0, механізми авторизації, методи інтеграції з зовнішніми API (Google Analytics API, Facebook Graph API, Twitter API), інструменти візуалізації даних (Chart.js, D3.js).
4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити) загальні відомості про розширення для Google Chrome, архітектура розширення для автоматизації збору та аналізу даних, інтеграція зовнішніх API (Google, Facebook, Twitter), реалізація авторизації через OAuth 2.0, комунікація з API через fetch() та XMLHttpRequest, використання Manifest v3, розробка інтерфейсу для візуалізації даних, тестування функціональності розширення, валідація API інтеграцій.
5. Дата видачі завдання 16.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання з підписами керівника і студента
1	Вступ	14.10.2024	
2	Розділ 1. Огляд сучасних підходів	09.12.2024	
3	Розділ 2. Проєктування розширення для Chrome	10.03.2025	
4	Розділ 3. Реалізація та тестування розширення	28.04.2025	
5	Висновки	12.05.2025	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2025	
7	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2025	
8	Подання кваліфікаційної роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	10.06.2025	

Студент _____ Владислав ШЕВЧЕНКО
(підпис)

Керівник роботи _____ Вікторія НЕМЧЕНКО
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Розширення для Google Chrome з інтеграцією зовнішніх API для автоматизації збору та аналізу даних» включає в себе перелік умовних позначень і скорочень, вступ, основну частину, яка складається з трьох розділів, висновків та списку використаних джерел. Загальний обсяг роботи становить 87 сторінок, в якій міститься 9 рисунків,8 таблиць та 1 додаток. Список використаних джерел налічує 16 позицій.

У рамках цієї роботи було створено спеціалізоване розширення для Google Chrome, що забезпечує автоматизований збір та аналіз даних шляхом інтеграції з зовнішніми API, зокрема Google API та Facebook API. Розширення реалізує повний цикл обробки інформації – від авторизації користувача за допомогою OAuth 2.0 до візуалізації ключових метрик через інтерактивний дашборд.

Програмний продукт розроблено з використанням сучасних вебтехнологій, таких як JavaScript, HTML і CSS. Детально описано архітектуру розширення, побудову клієнтської та серверної частин, процеси інтеграції з API та обробки даних у форматі JSON. Проведене тестування підтвердило стабільність, зручність та ефективність роботи розробленого рішення.

Розширення може бути ефективно впроваджене в різноманітні галузі, де необхідна автоматизація процесів збору, аналізу та візуалізації даних із зовнішніх джерел.

Ключові слова: Google Chrome, розширення браузера, API, автоматизація, збір даних, аналіз даних, OAuth 2.0, JavaScript, вебтехнології, дашборд.

ABSTRACT

Qualification work on the topic “Extension for Google Chrome with the integration of external APIs for automating data collection and analysis” includes a list of symbols and abbreviations, introduction, main part, which consists of three chapters, conclusions and a list of references. The total volume of the work is 87 pages, which contains 9 figures,8 tables and 1 appendice. The list of references includes 16 items.

As part of this work, a specialized extension for Google Chrome was created that provides automated data collection and analysis by integrating with external APIs, including Google API and Facebook API. The extension implements a full cycle of information processing - from user authorization using OAuth 2.0 to visualization of key metrics through an interactive dashboard.

The software product was developed using modern web technologies such as JavaScript, HTML, and CSS. The article describes in detail the architecture of the extension, the construction of the client and server parts, the processes of integration with the API, and the processing of data in JSON format. The conducted testing has confirmed the stability, convenience, and efficiency of the developed solution.

The extension can be effectively implemented in a variety of industries that require automation of data collection, analysis, and visualization from external sources.

Keywords: Google Chrome, browser extension, API, automation, data collection, data analysis, OAuth 2.0, JavaScript, web technologies, dashboard.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

API	Application Programming Interface (Інтерфейс програмного застосунку)
OAuth 2.0	Open Authorization 2.0 (Відкрита авторизація, друга версія)
JSON	JavaScript Object Notation (Формат обміну даними)
REST	Representational State Transfer (Стандарт архітектури вебсервісів)
CSP	Content Security Policy (Політика безпеки вмісту)
HTTP	HyperText Transfer Protocol (Протокол передачі гіпертексту)
Chrome Storage API	Інтерфейс збереження даних у браузері Chrome
DOM	Document Object Model (Об’єктна модель документа)
Chart.js	Бібліотека для створення графіків у JavaScript
SDK	Software Development Kit (Набір інструментів для розробки програмного забезпечення)
JSON-файл	Файл, що використовує формат JavaScript Object Notation
Manifest V3	Третя версія конфігураційного файлу розширень Chrome
UI	User Interface (Користувацький інтерфейс)

ВСТУП

Сьогодні, в умовах стрімкого розвитку цифрових технологій, автоматизація збору та обробки даних стала важливим аспектом функціонування багатьох галузей. Інформація, яка раніше збиралася вручну, тепер може бути інтегрована та оброблена за допомогою сучасних програмних рішень, таких як розширення для браузерів. Серед них розширення для Google Chrome займають провідну позицію завдяки зручності використання, широкій підтримці API та потужному набору інструментів для розробників.

Актуальність цієї роботи обумовлена потребою в ефективних засобах інтеграції даних із різних джерел. У сучасному інформаційному просторі API (Application Programming Interface) стали стандартом взаємодії між додатками та сервісами. Вони надають можливість автоматизованого доступу до даних та їх обробки. Зокрема, API таких популярних платформ, як Google, Facebook, Twitter, дозволяють отримувати аналітичні показники, інформацію про аудиторію та багато інших даних, які є цінними для прийняття управлінських рішень, маркетингових кампаній і моніторингу ринку.

Однак інтеграція даних через API часто супроводжується низкою проблем. Це і різноманітність форматів даних, і необхідність авторизації за допомогою протоколів, таких як OAuth 2.0, і обмеження на кількість запитів до API. Тому створення розширення, яке вирішує ці проблеми, є важливим завданням. Розробка такого інструменту дозволить значно спростити процес інтеграції даних для користувачів, підвищити продуктивність та забезпечити зручний доступ до важливої інформації.

Крім того, розвиток веб-технологій та їх доступність сприяли зростанню популярності браузерних розширень. За даними Statista, на 2023 рік Chrome є найпопулярнішим браузером із часткою ринку понад 65%. Це робить розробку розширення для Chrome стратегічно вигідною. Висока гнучкість цього інструменту дозволяє не лише вирішувати поточні завдання, а й адаптуватися

до потреб різних категорій користувачів, включаючи бізнес, аналітиків і розробників.

У сучасному цифровому середовищі бізнес та інші організації стикаються з необхідністю оперативного збору, аналізу та обробки великих обсягів даних. Це може стосуватися аналізу вебтрафіку, даних із соціальних мереж або інших джерел інформації. З кожним роком обсяги даних, що створюються та обробляються, стрімко зростають, що робить неможливим їхнє ручне опрацювання або неефективні системи збору даних. У зв’язку з цим виникає потреба в інструментах, які можуть автоматизувати процеси збору та аналізу даних з вебресурсів та зовнішніх API.

Ступінь дослідження проблеми. Питання автоматизації збору даних та інтеграції з API активно досліджується у сфері інформаційних технологій та розробки програмного забезпечення. Існують численні розширення для браузера Google Chrome, які допомагають автоматизувати ці процеси, такі як Web Scraper та Data Miner. Однак багато з цих рішень мають обмежений функціонал або працюють з обмеженим набором API. Крім того, питання інтеграції зовнішніх API, таких як Google API, Facebook API або Twitter API, також залишається важливим завданням для забезпечення збору точних і релевантних даних.

Актуальність теми. Актуальність обраної теми зумовлена зростаючими потребами в автоматизації обробки великих обсягів даних у різних сферах діяльності – від маркетингу до наукових досліджень. Розробка розширення для Google Chrome з інтеграцією зовнішніх API дозволить автоматизувати процес збору даних, підвищити точність і ефективність аналізу інформації, що має важливе значення в сучасних умовах. Інтеграція з API відкриває можливості для роботи з найрізноманітнішими джерелами даних, що дозволяє значно розширити функціонал розширення та зробити його універсальним інструментом для користувачів різних рівнів.

Метою дослідження є розробка розширення для браузера Google Chrome, яке забезпечить автоматизований збір даних із вебресурсів та зовнішніх API, таких як Google, Facebook, Twitter, з подальшою їх обробкою та аналізом.

Завдання дослідження:

1. Провести аналіз існуючих розширень для браузера Chrome, які автоматизують збір даних.
2. Огляд існуючих API для збору та обробки даних з різних платформ (Google, Facebook, Twitter).
3. Вивчення стандартів та протоколів інтеграції з API (REST, GraphQL).
4. Розробка алгоритму для автоматизованого збору та обробки даних за допомогою розширення Chrome.
5. Впровадження API в розширення для Chrome з метою отримання даних з зовнішніх джерел.
6. Тестування розширення на практичних прикладах для оцінки ефективності його роботи.

Об'єктом дослідження є інформаційні технології збору та обробки даних за допомогою браузерних розширень.

Предметом дослідження є методи інтеграції зовнішніх API (Google API, Facebook API, Twitter API) з розширенням для браузера Google Chrome для автоматизації процесу збору та аналізу даних.

Актуальність обраної теми, а також чітке формулювання мети і завдань дослідження, дозволять розробити розширення, яке стане ефективним інструментом для автоматизації збору та аналізу даних із зовнішніх джерел.

РОЗДІЛ 1 ОГЛЯД СУЧАСНИХ ПІДХОДІВ

1.1. Аналіз існуючих розширень Chrome для автоматизації збору даних

У цьому підрозділі розглянемо найпопулярніші та ефективні розширення для Google Chrome, що використовуються для автоматизації збору даних. Одним із найвідоміших прикладів є Web Scraper – інструмент, який дозволяє збирати дані з вебсайтів та зберігати їх у формі таблиць або файлів. Ще одним відомим рішенням є Data Miner, що надає можливість автоматично збирати інформацію з таблиць і списків на вебсторінках.

Критичний аналіз існуючих розширень дозволяє виділити їх основні переваги та недоліки. Наприклад, Web Scraper є дуже гнучким і потужним інструментом, однак його використання може вимагати технічних навичок налаштування. Data Miner, у свою чергу, є більш дружнім для користувача, але його функціонал є обмеженим для великих обсягів даних. Важливим є також те, що більшість існуючих рішень працюють з обмеженими наборами даних та не підтримують інтеграцію з багатьма API для збору даних у реальному часі [1].

Таблиця 1.1. надає загальний огляд розширення для Google Chrome, його ключові можливості, технічні аспекти та основні сценарії використання.

Таблиця 1.1. Аналіз основних характеристик розширень для Google Chrome, їх можливості та застосування

Характеристика	Опис
Визначення розширення Chrome	Програмний модуль, який додає нові функції або покращує можливості браузера Google Chrome.
Технічна основа	Складається з HTML, CSS, JavaScript файлів, які працюють у браузері на комп'ютері користувача.
Встановлення	- З офіційного магазину Chrome Web Store - Шляхом ручного додавання файлів

Продовження Таблиці 1.1.

Характеристика	**Опис**
Основні функції	- Автоматизація процесів (збір даних, заповнення форм) - Інтеграція з вебсервісами через API (Google API, Facebook API, Twitter API) - Покращення інтерфейсу користувача (зміна вигляду сторінок, блокування реклами)
Застосування	- Автоматичний збір даних з вебсайтів - Інтеграція з API для обміну даними з зовнішніми сервісами - Покращення продуктивності і персоналізація браузера
Переваги	- Спрощує виконання рутинних завдань - Підвищує ефективність роботи з великими обсягами даних - Розширює функціональні можливості браузера

У Таблиці 1.2. наведено огляд популярних розширень для Google Chrome, які спеціалізуються на автоматизації збору даних.

Ці розширення надають широкий спектр можливостей для автоматизації збору даних, залежно від потреб користувача. Вибір конкретного інструменту залежить від типу даних, які потрібно зібрати, а також від складності вебсайтів, з яких планується витяг інформації.

Функціональність та підтримка динамічних сайтів. Розширення Web Scraper, Octoparse і ParseHub є лідерами у підтримці динамічних вебсайтів, які використовують JavaScript або AJAX для завантаження контенту. Instant Data Scraper, Data Miner і Scraper не мають підтримки таких сайтів, що значно обмежує їх використання в сучасних умовах, коли більшість сайтів є динамічними [2].

Простота використання. Найпростіші у використанні – Data Miner та Instant Data Scraper, які не потребують технічних навичок або складного налаштування. Вони ідеально підходять для новачків, які хочуть швидко отримати дані з вебсайтів у форматі таблиць. Web Scraper, Octoparse та ParseHub мають складніші інтерфейси та потребують базових технічних знань для налаштування сценаріїв.

Ефективність роботи з великими обсягами даних. Octoparse та ParseHub є найпотужнішими інструментами для збору великих обсягів даних, оскільки вони підтримують хмарне зберігання та можуть працювати з багатьма сайтами одночасно. Це дає можливість збирати дані в реальному часі та аналізувати великі масиви інформації. Інші розширення, такі як Web Scraper та Data Miner, мають обмежену ефективність при роботі з великими обсягами даних і можуть уповільнювати роботу браузера.

Таблиця 1.2. Огляд популярних розширень для Google Chrome, які спеціалізуються на автоматизації збору даних

Назва розширення	Опис	Основні функції	Переваги	Недоліки
Web Scraper	Розширення для збору даних з вебсайтів шляхом налаштування власних сценаріїв вебскрапінгу.	- Створення мап сайту для збору даних - Експорт у форматах CSV, Excel, JSON - Можливість збереження даних	- Підтримка збору даних з динамічних сторінок - Інтуїтивний інтерфейс - Безкоштовна версія	- Потребує технічних навичок для налаштування складних сценаріїв - Обмеження у кількості одночасних запитів
Data Miner	Інструмент для збору даних з таблиць, списків та інших структурованих елементів на вебсторінках.	- Експорт даних у CSV або Excel - Підтримка XPath і JQuery - Можливість створення сценаріїв для скрапінгу	- Простота у використанні - Не потребує програмування - Можливість налаштування шаблонів збору	- Обмежений функціонал для великих вебсайтів - Не підходить для збору даних із динамічних сайтів

Продовження Таблиці 1.2.

Назва розширення	Опис	Основні функції	Переваги	Недоліки
Instant Data Scraper	Просте у використанні розширення для автоматизації збору даних з таблиць на вебсторінках.	- Автоматичне виявлення таблиць - Експорт даних у CSV або Excel - Безкоштовне використання	- Швидке налаштування - Працює без складних налаштувань - Не потребує програмування	- Обмежені можливості для збору даних із нестандартних сайтів - Не працює з динамічним контентом
Scraper	Розширення для витягу даних із вебсайтів за допомогою XPath та JQuery запитів.	- Збір даних за допомогою XPath або JQuery - Експорт даних у Google Spreadsheets - Підтримка масштабованих зборів даних	- Легке інтегрування з Google Spreadsheets - Простота в налаштуванні - Можливість ручного збору даних	- Не підходить для збору великих обсягів даних - Потребує технічних знань для ефективного використання

Гнучкість налаштувань та кастомізація. Scraper та Web Scraper дозволяють використовувати XPath та JQuery для створення складних запитів, що робить їх надзвичайно гнучкими інструментами для збору специфічних даних. Проте для ефективного використання цих функцій потрібні технічні навички. Для користувачів, які не мають знань з програмування, ці інструменти можуть виявитися складними у використанні.

Таким чином, для розробки нового розширення варто взяти до уваги такі ключові моменти:

- Підтримка динамічних сайтів: оскільки більшість сучасних вебсайтів використовують динамічне завантаження контенту (JavaScript, AJAX), важливо забезпечити підтримку цих технологій, щоб розширення могло збирати дані з будь-якого типу сайтів. Це ключова перевага, яку забезпечують такі інструменти, як Octoparse та ParseHub.

- Простота інтерфейсу: незважаючи на важливість гнучкості та функціональності, простота використання є критично важливою для залучення широкого кола користувачів. Наприклад, Data Miner та Instant Data Scraper мають інтуїтивно зрозумілий інтерфейс, що робить їх популярними серед новачків. Нове розширення повинно мати простий та зручний інтерфейс, особливо для користувачів без технічних знань.
- Масштабованість та підтримка великих обсягів даних: підтримка збору та обробки великих обсягів даних є ще однією важливою характеристикою. Це можна досягти через інтеграцію з хмарними сервісами або застосування методів оптимізації обробки даних на рівні браузера.
- Гнучкість налаштувань: можливість налаштування сценаріїв збору даних є важливою для досвідчених користувачів. Інтеграція з інструментами для налаштування XPath або інших інструментів вибору контенту дозволить створити потужний інструмент для користувачів з технічними знаннями.
- Безпека та конфіденційність: слід також звернути увагу на питання безпеки та конфіденційності при зборі даних. Важливо врахувати сучасні стандарти захисту даних, особливо якщо розширення працюватиме з персональними даними або інтегруватиметься з соціальними мережами чи іншими платформами.

При розробці нового розширення важливо забезпечити баланс між простотою використання та функціональністю. Користувачі повинні мати можливість легко налаштовувати процес збору даних, навіть якщо вони не мають технічних знань, але при цьому розширення повинно мати розширені можливості для досвідчених користувачів. Підтримка збору даних з динамічних сайтів, можливість роботи з великими обсягами інформації, інтеграція з хмарними сервісами, а також простий і безпечний інтерфейс – це основні фактори, які потрібно врахувати під час розробки.

Продовження Таблиці 1.2.

Назва розширення	Опис	Основні функції	Переваги	Недоліки
Octoparse	Потужний інструмент для вебскрапінгу з підтримкою динамічних сайтів та автоматизації збору даних.	- Збір даних з динамічних сайтів - Підтримка автоматизації задач - Хмарне зберігання даних	- Підтримка динамічних вебсайтів - Можливість обробки великих обсягів даних - Вбудована підтримка API - Багатий функціонал - Хмарна автоматизація процесів	- Платна версія з обмеженими функціями у безкоштовній версії - Потребує технічних знань для налаштування
ParseHub	Потужний інструмент для збору даних з динамічних вебсайтів, що використовують JavaScript, AJAX.	- Збір даних з динамічних вебсайтів - Підтримка складних сценаріїв	- Інтуїтивний інтерфейс - Підтримка збору з важкодоступних сайтів - Можливість одночасної роботи з кількома сайтами	- Платна версія з обмеженнями у безкоштовній - Потрібні навички налаштування складних сценаріїв

На рисунку 1.1 зображено графіки щодо використання Google API серед користувачів.

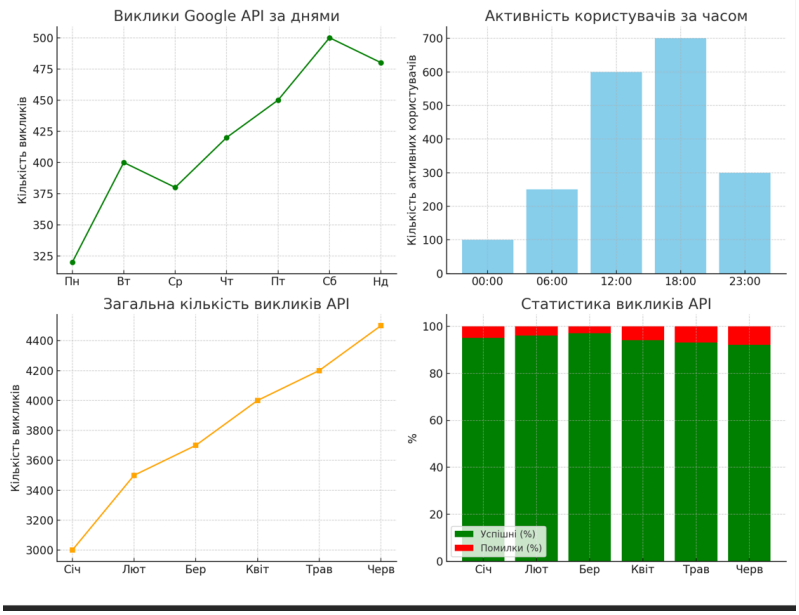

Рисунок 1.1 – Використання Google API серед користувачів

1.2. Огляд доступних API для збору та аналізу даних (наприклад, API Google, Facebook, X)

Для автоматизації збору даних за допомогою розширень Chrome важливо інтегрувати зовнішні API, що забезпечують доступ до великих обсягів даних з різних платформ.

API (Application Programming Interface) – це інтерфейс програмування застосунків, який дозволяє різним програмам обмінюватися даними та функціями між собою. API надає набір визначених правил та методів, за допомогою яких одна програма може отримати доступ до функціональних можливостей іншої. Це може включати доступ до баз даних, служб або різних систем.

Наприклад, за допомогою API Google Maps можна інтегрувати карти та функції навігації у сторонній вебсайт або мобільний додаток. API надають інтерфейси для запитів до певних сервісів, повертаючи відповіді в структурованій формі, найчастіше у форматі JSON або XML.

Дане розширення для Chrome використовуватиме API для збору та обробки даних із зовнішніх джерел, таких як вебсервіси Google, Facebook, Twitter та інші платформи. Інтеграція з API дозволить автоматизовано отримувати дані, які зберігаються на цих платформах, без необхідності ручного копіювання інформації. Наприклад: Google API може використовуватися для збору статистики вебсайтів через Google Analytics; X API дозволить збирати дані про твіти, ретвіти та користувачів; Facebook API забезпечить доступ до даних з профілів та публікацій.

Нижче наведено огляд основних API, що використовуються для збору та аналізу даних:

- Google API: Google надає численні API, включаючи Google Analytics API, Google Search Console API, Google Maps API, що дозволяють збирати дані про вебсайти, користувачів і місцезнаходження. Ці API добре документовані та

широко використовуються в різних галузях, таких як маркетинг, аналіз вебтрафіку та логістика [3].

- Facebook API: Facebook Graph API надає доступ до великої кількості даних з профілів користувачів, публікацій та сторінок. Цей API використовується для збору соціальної інформації, що є корисним у маркетингових дослідженнях, таргетованій рекламі та аналізі активності користувачів.
- X API: X API дозволяє збирати дані про твіти, ретвіти, лайки та активність користувачів. Це API є корисним для аналізу настроїв, трендів та соціальних зв'язків між користувачами.

Кожен із вказаних API має свої особливості та обмеження. Наприклад, Facebook API вимагає авторизації користувачів і надає доступ до обмеженого обсягу даних відповідно до політики приватності. У той час як Google API надають значно більше можливостей для аналізу бізнес-даних.

Таблиця 1.3. відображає основні аспекти інтеграції API в наше розширення для Chrome та їх вплив на процес розробки і функціональність розширення.

Таблиця 1.3. Огляд API, його роль у розробці розширення для Chrome та важливі аспекти, що впливають на розробку

Параметр	Опис
Термінологічне визначення	API (Application Programming Interface) – це інтерфейс для взаємодії програм, який дозволяє обмінюватися даними та функціями між різними системами.
Приклад API	Google API (для збору аналітичних даних), X API (для збору даних про твіти), Facebook Graph API (для доступу до профілів і публікацій).
Зв'язок між API та розширенням	Наше розширення для Chrome буде використовувати API для збору даних з зовнішніх джерел (наприклад, з Google, Facebook або Twitter).
Вплив на розробку	- Потрібна розробка механізмів для надсилання запитів до API. - Обробка даних, отриманих через API. - Використання методів авторизації (OAuth).
Завдання розробки	- Автоматизація збору даних. - Доступ до актуальних даних у реальному часі. - Збір великих обсягів даних із зовнішніх джерел.
Важливі аспекти для розробки	- Підтримка протоколів API (REST, GraphQL). - Безпека авторизації через API (OAuth2). - Оптимізація кількості запитів до API.
Обмеження API	- Обмеження на кількість запитів (квоти). - Необхідність авторизації для доступу до особистих даних користувачів. - Політика безпеки.
Переваги інтеграції API	- Можливість автоматизованого збору даних. - Підтримка динамічних даних. - Розширення функціональності розширення через API.
Вплив на роботу розширення	Інтеграція API дозволить розширенню отримувати та обробляти дані в автоматизованому режимі, роблячи процес збору даних більш ефективним та актуальним.

Використання кількох API в одному розширенні для Chrome дозволяє значно розширити функціональні можливості та забезпечити автоматизований збір даних із різних джерел. Проте це додає певної складності в розробку, оскільки вимагає налаштування кількох процесів авторизації, обробки запитів

та керування обмеженнями API. Важливо також звертати увагу на безпеку, синхронізацію запитів та ефективне використання ресурсів для обробки великих обсягів даних [4].

Таблиця 1.4. Основні аспекти використання кількох API в одному розширенні Chrome, включаючи процес роботи, складнощі та рекомендації для ефективної інтеграції

Аспект	**Опис**	**Приклад**	**Вплив на розробку**	**Рекомендації**
Можливість використання кількох API	Розширення може надсилати запити до кількох API одночасно або послідовно.	- Google API - Twitter API - Facebook API	Підвищує функціональність, дозволяє інтегрувати дані з різних джерел.	- Розділення логіки кожного API - Використання пакетних запитів
Отримання даних з кількох джерел	API дозволяють збирати дані з різних платформ для подальшого аналізу.	- Google Analytics API - Twitter API	Ускладнює структуру розширення через синхронізацію даних.	- Використання async/await - Promise обробка
Авторизація для кількох API	Кожен API має свою схему авторизації.	- Google OAuth 2.0 - Twitter OAuth 2.0 - Facebook авторизація	Ускладнення інтеграції, особливо при багатьох сервісах.	- Єдина система обробки авторизації - Безпечне зберігання токенів

Продовження Таблиці 1.4.

Аспект	Опис	Приклад	Вплив на розробку	Рекомендації
Оптимізація запитів до кількох API	Уникнення перевантаження API і дотримання лімітів.	- Twitter API: запити/год - Google API: запити/день	Перевищення лімітів може призвести до блокування.	- Кешування - Пакетні запити - Моніторинг лімітів
Керування квотами API (Quota Limits)	API обмежують кількість запитів у часі.	- Google Cloud API - Twitter API	Можлива втрата доступу до API при перевищенні квоти.	- Оптимізація запитів - Закупівля додаткових квот
Керування затримками (Latency Issues)	API можуть повільно відповідати через навантаження.	- Facebook API під час пікових навантажень - Twitter API	Затримки уповільнюють роботу розширення.	- Кешування - Оптимізація роботи з мережею
Синхронізація даних із кількох API	Необхідність об’єднання даних із різних джерел.	- Дані з Google Analytics і Twitter для єдиного звіту	Асинхронна обробка може ускладнити реалізацію.	- Promise.all - Система синхронізації даних
Масштабованість та гнучкість	Кожен API додає нову функціональність і можливості.	- Google API - Facebook API - Twitter API	Підвищення складності, але й функціональності.	- Архітектурне планування - Послідовна інтеграція

Таблиця 1.4. містить усі основні аспекти, які необхідно врахувати під час роботи з кількома API в одному розширенні для Chrome. Інтеграція кількох API в розширення для Chrome відкриває широкі можливості для автоматизації збору та аналізу даних. Однак для успішної реалізації цього підходу потрібно врахувати авторизацію, оптимізацію запитів, обробку даних із кількох джерел і

синхронізацію запитів. Крім того, важливо управляти лімітами API, щоб уникнути перевантаження або блокування запитів. Важливо також забезпечити надійне та безпечне зберігання токенів авторизації та врахувати можливі затримки у відповіді API.

Приклад того, як це може виглядати в коді:

```
// Отримання даних із Google API
fetch('https://www.googleapis.com/analytics/v3/data/ga', {
  headers: {
    'Authorization': 'Bearer ' + accessTokenGoogle
  }
})
.then(response => response.json())
.then(data => {
  console.log('Google Analytics Data:', data);
});
```

лістинг 1.1.1 приклад коду отримання даних із Google API

```
// Отримання даних із Twitter API
fetch('https://api.twitter.com/2/tweets', {
  headers: {
    'Authorization': 'Bearer ' + accessTokenTwitter
  }
})
.then(response => response.json())
.then(data => {
  console.log('Twitter Data:', data);
});
// Паралельний збір даних
Promise.all([googleData, twitterData])
.then(([googleResponse, twitterResponse]) => {
```

```
  console.log('Combined Data:', googleResponse, twitterResponse);
});
```

лістинг 1.1.2 приклад коду отримання даних із Twitter API

Інтеграція API у розширення Chrome є важливою складовою для автоматизації збору та аналізу даних. Вона дозволить розширенню працювати з різноманітними джерелами, знижувати витрати часу на рутинні операції та надавати користувачам актуальні дані. Для успішної розробки варто врахувати обмеження API, впровадити механізми для обробки великих обсягів даних та забезпечити безпеку авторизації через API для захисту персональних даних користувачів.

Таблиця 1.5. відображає всі важливі обмеження API, які необхідно враховувати під час інтеграції в розширення для Chrome. Врахування цих аспектів дозволить уникнути проблем із продуктивністю, доступом до даних, а також оптимізувати роботу розширення, забезпечивши його стабільність і ефективність.

Таблиця 1.5 Аналіз важливих аспектів та обмежень API для врахування під час інтеграції в розширення Chrome

Тип обмеження	Опис	Приклади	Вплив на розширення	Як уникнути/оптимізувати
Обмеження кількості запитів (Rate Limits)	Більшість API мають обмеження на кількість запитів, які можна надіслати за певний проміжок часу (хвилина, день).	- Google API: обмежує кількість запитів до Google Analytics або Maps API. - Twitter API: обмежує кількість запитів до твіти та профілів. - Facebook Graph API: має ліміти на запити залежно від типу даних.	Якщо ліміти перевищені, API може тимчасово заблокувати запити, що сповільнює або зупиняє роботу розширення.	- Оптимізувати запити. - Використовувати кешування даних. - Застосовувати пакетне запитування.
Обмеження на тип даних та функції	Обмежений доступ до певних типів даних або функцій API залежно від рівня доступу або політики конфіденційності.	- Facebook API: обмежує доступ до персональних даних користувачів без їхньої згоди. - Google Maps API: обмежує функції в безкоштовній версії.	Неможливість отримати необхідні дані або виконати функції через політику конфіденційності чи платні обмеження.	- Враховувати доступні дані при проєктуванні. - Оптимізувати використання доступних функцій.

Продовження таблиці 1.5

Тип обмеження	Опис	Приклади	Вплив на розширення	Як уникнути/оптимізувати
Квоти на використання ресурсів (Quota Limits)	Деякі API мають квоти, що обмежують загальне використання ресурсів (наприклад, обсяг оброблених даних, кількість запитів на день).	- Google Cloud API: обмежує кількість ресурсомістких запитів. - AWS API: має обмеження в безкоштовній версії.	Якщо квоти використано, розширення не зможе отримати більше даних або доступ до функцій до поновлення квот.	- Оптимізувати використання ресурсів. - Звести до мінімуму кількість ресурсомістких запитів. - Придбати додаткові квоти або використовувати платні версії API.
Затримки у відповідях API (Latency Issues)	Деякі API можуть мати затримки у відповідях через навантаження на сервер або низьку швидкість мережі.	- Facebook API: може мати затримки у пікові періоди активності. - Twitter API: теж може сповільнюватись через навантаження.	Затримки можуть призвести до уповільнення або неправильної роботи розширення, особливо для збору даних у реальному часі.	- Реалізувати кешування даних. - Оптимізувати запити для мінімізації залежності від часу відповіді API.
Застарілі версії API (Deprecation Issues)	Деякі API можуть ставати застарілими, і підтримка старих версій може бути припинена.	- Google API: періодично зупиняє підтримку старих версій API. - Facebook API: може оновлювати свої версії, змінюючи функції та формат запитів.	Якщо використовувати застарілу версію API, розширення може припинити працювати належним чином після припинення підтримки.	- Регулярно перевіряти оновлення API. - Оновлювати код розширення для підтримки актуальних версій API.

1.3. Огляд стандартів та протоколів інтеграції API (REST, GraphQL)

У розробці програмних рішень інтеграція з API (Application Programming Interface) є важливим аспектом, що забезпечує взаємодію між клієнтом і сервером. Для реалізації цієї взаємодії використовуються різні протоколи та стандарти, серед яких найбільш популярними є REST (Representational State Transfer) і GraphQL (Query Language for APIs). Кожен із цих підходів має свої переваги, недоліки та області застосування.

Інтеграція розширень Chrome із зовнішніми API базується на використанні стандартних протоколів та форматів передачі даних. Два основних протоколи для інтеграції API – це REST та GraphQL.

- REST (Representational State Transfer): Це найпопулярніший стандарт для створення вебсервісів. Він дозволяє клієнту звертатися до серверу для отримання ресурсів, використовуючи HTTP-запити (GET, POST, PUT, DELETE). REST API прості в реалізації та підтримують більшість сучасних вебплатформ. Основна перевага REST – це легкість у використанні та масштабованість. Однак цей протокол може бути менш ефективним при роботі з великими обсягами даних, оскільки кожен запит повертає всю відповідь, незалежно від потреб клієнта.
- GraphQL: Цей протокол дозволяє клієнтам запитувати лише ті дані, які їм потрібні. Основною перевагою GraphQL є його ефективність – клієнт може визначити, які саме поля йому необхідні, що зменшує обсяг переданих даних і прискорює взаємодію з API. Проте впровадження GraphQL може бути складнішим, ніж REST, і не всі API підтримують цей протокол.

- Таким чином, вибір протоколу для інтеграції залежить від потреб проєкту та доступних інструментів. REST API залишається найпоширенішим стандартом для інтеграції, однак GraphQL набирає популярність завдяки своїй гнучкості та ефективності при роботі з великими даними.

- REST є найкращим вибором для інтеграції API у браузерні розширення завдяки своїй простоті, масштабованості та широкому застосуванню. Він забезпечує ефективну роботу з вебсервісами, такими як соціальні мережі, аналітичні інструменти та інші онлайн-платформи. Оскільки більшість сучасних API побудовані на основі REST, його підтримка є однією з найпоширеніших, що робить його найбільш зручним і практичним для розробників.

Таблиця 1.6. Порівняльний аналіз протоколів API

Протокол	Формат даних	Основні переваги	Основні недоліки	Рекомендації
REST	JSON, XML	Простота, гнучкість, масштабованість, широко підтримується	Не завжди оптимальний для складних запитів	Найкращий вибір для більшості вебдодатків, API для соцмереж, аналітичних даних.
GraphQL	JSON	Оптимізація запитів, гнучкість, менше запитів	Складність налаштування, навантаження на сервер	Чудово підходить для складних додатків або додатків, що потребують складних запитів до багатьох джерел даних.
SOAP	XML	Високий рівень безпеки, підтримка транзакцій	Складність, великі обсяги даних	Підходить для додатків з високими вимогами до безпеки, таких як банківські системи.
gRPC	Протобуфери	Швидкість, низький трафік, підтримка двостороннього стрімінгу	Не підтримується у браузерах	Підходить для мікросервісів або внутрішніх систем, але не для браузерних розширень через відсутність підтримки.

GraphQL є чудовим варіантом для складних додатків, які потребують оптимізації запитів. Його основна перевага полягає в можливості зробити один запит для отримання різноманітних даних із кількох джерел, що зменшує кількість зайвих запитів і скорочує передачу зайвих даних. Це особливо корисно для проєктів, де потрібно ефективно керувати великою кількістю даних. GraphQL дозволяє запитувати лише необхідні поля, що знижує навантаження на мережу.

SOAP підходить для додатків, які мають високі вимоги до безпеки та надійності, таких як банківські або фінансові системи. SOAP забезпечує високий рівень безпеки та чітку структуру передачі даних, що робить його корисним для складних транзакційних систем. Проте SOAP є досить складним у налаштуванні і обробці, особливо у випадках, коли потрібно працювати з XML. Крім того, він генерує великий обсяг трафіку через використання XML, що робить його менш ефективним у порівнянні з REST або GraphQL для звичайних вебдодатків.

gRPC, розроблений Google, призначений для високопродуктивних систем і мікросервісної архітектури, де важливі швидкість передачі даних і підтримка двостороннього стрімінгу. Проте gRPC не підтримується нативно у браузерах через використання HTTP/2 і бінарних протобуферів, тому його краще використовувати для серверної інтеграції або внутрішніх систем, а не для браузерних розширень [5].

REST і GraphQL є потужними інструментами для інтеграції API, кожен із яких підходить для різних сценаріїв. REST є оптимальним вибором для простих систем із добре визначеними ресурсами, тоді як GraphQL краще підходить для складних додатків із гнучкими потребами у даних. У нашому розширенні REST використовується для інтеграції з популярними сервісами завдяки його простоті, тоді як GraphQL може бути ефективним для роботи з багатокомпонентними структурами даних.

Отже, REST є найбільш оптимальним вибором для розширень Chrome завдяки своїй простоті та масштабованості. Якщо ваше розширення потребує

збору складних даних із кількох джерел, варто розглянути GraphQL для зменшення кількості запитів. SOAP слід використовувати лише в тих випадках, коли критично важлива безпека та транзакційність. gRPC, хоч і є високопродуктивним протоколом, не підходить для браузерних додатків, тому його варто використовувати лише для серверних систем або мікросервісів [3].

Висновки до Розділу 1

Розділ 1, присвячений теоретичному огляду сучасних підходів до розробки розширень Chrome з інтеграцією зовнішніх API для автоматизації збору та аналізу даних, дозволяє зробити кілька ключових висновків. По-перше, аналіз існуючих розширень Chrome демонструє широкий спектр можливостей для автоматизації процесів збору даних, проте кожне розширення має свої сильні та слабкі сторони. Важливо враховувати досвід попередніх рішень для вибору найкращих практик та уникнення можливих помилок.

По-друге, огляд доступних API, таких як Google API, Facebook API та Twitter API, показує, що інтеграція зовнішніх сервісів є ефективним інструментом для збирання даних у реальному часі. Однак, кожне API має свої особливості, вимоги до авторизації та обмеження, які слід враховувати під час розробки. Авторизація через OAuth 2.0 та керування квотами запитів є важливими аспектами, які впливають на надійність та продуктивність розширення.

Третій висновок стосується вибору протоколів інтеграції API. REST є найбільш поширеним та зручним для браузерних розширень завдяки своїй простоті та гнучкості. У разі потреби роботи з великими обсягами даних або складними запитами, GraphQL може забезпечити оптимізацію за рахунок зменшення кількості запитів. SOAP та gRPC, хоча і мають свої переваги, менш підходять для роботи з розширеннями Chrome через їхню складність та обмежену підтримку в браузерах.

Загалом, критичний аналіз показує, що для успішної розробки розширення Chrome з інтеграцією API необхідно ретельно підходити до вибору протоколів, управління запитами та обробки даних. Важливо забезпечити

належну безпеку, оптимізацію запитів і врахувати обмеження API, щоб підвищити продуктивність розширення та зробити його зручним у використанні.

РОЗДІЛ 2 ПРОЄКТУВАННЯ РОЗШИРЕННЯ ДЛЯ CHROME

2.1. Опис архітектури розширення

2.1.1. Фронтенд та бекенд складові

Фронтенд та бекенд складові розширення для браузера Chrome відіграють ключову роль у забезпеченні його функціональності та ефективності. Фронтенд відповідає за взаємодію з користувачем, а бекенд – за реалізацію логіки роботи розширення, обробку даних і комунікацію з зовнішніми ресурсами через API.

Фронтенд реалізується з використанням трьох основних вебтехнологій: HTML, CSS та JavaScript. HTML (HyperText Markup Language) відповідає за структуру вебсторінки, створюючи основу для елементів інтерфейсу, таких як кнопки, текстові поля, таблиці та інші компоненти. CSS (Cascading Style Sheets) дозволяє створювати стильовий дизайн, включаючи кольори, шрифти, адаптивність і розташування елементів. Використання CSS важливе для забезпечення сучасного вигляду розширення, яке має виглядати професійно і привабливо для користувача. JavaScript відповідає за інтерактивність, тобто виконання динамічних функцій, таких як відправлення даних форми, оновлення інформації в реальному часі, побудова графіків і таблиць без необхідності перезавантаження сторінки [6].

У контексті браузерного розширення важливими елементами фронтенду є popup-сторінка, сторінка налаштувань та інформаційний дашборд. Popup-сторінка надає користувачеві швидкий доступ до основних функцій розширення, таких як запуск збору даних або авторизація через API. Сторінка налаштувань (options) дозволяє змінювати параметри роботи розширення, наприклад, частоту збору даних або API-токени. Інформаційний дашборд – це окремий модуль для візуалізації зібраних даних у вигляді таблиць, графіків або діаграм. Для побудови графіків рекомендується використовувати бібліотеки, такі як Chart.js або D3.js, які дозволяють створювати адаптивні та інформативні візуалізації.

Фронтенд також має забезпечити підтримку адаптивного дизайну, щоб розширення було зручним у використанні як на стаціонарних комп’ютерах, так і на ноутбуках із різними розмірами екрана. Для цього використовуються медіазапити CSS (media queries) та фреймворки типу Bootstrap.

Бекенд розширення виконує роль основного обчислювального центру. У браузерних розширеннях Chrome бекенд реалізується через фонові скрипти (background scripts). Це спеціальні JavaScript-файли, які працюють незалежно від інтерфейсу, забезпечуючи обробку запитів, авторизацію та збереження даних. Важливо, що фонові скрипти в новій архітектурі Chrome Manifest v3 замінені сервісними робітниками (service workers), що забезпечує більшу безпеку та енергоефективність [7].

2.1.2. Комунікація з API через fetch() або XMLHttpRequest

Фонові скрипти відповідають за виконання запитів до API. Для цього використовується сучасний метод `fetch()`, який дозволяє легко обробляти асинхронні запити і працювати з відповідями у форматі JSON або XML. Для більш старих API, які не підтримують `fetch()`, можливо використовувати `XMLHttpRequest`. Наприклад, для збору даних з Google Analytics API необхідно створити запит, передаючи в заголовку авторизаційний токен, отриманий через OAuth 2.0. Це гарантує безпеку передачі даних та захищає облікові записи користувачів.

Обробка даних у бекенді включає також їх збереження. Для цього використовується локальне сховище браузера (localStorage або Chrome Storage API). Chrome Storage API дозволяє працювати як з синхронним, так і з асинхронним збереженням даних, забезпечуючи більшу гнучкість у роботі з великими обсягами інформації.

Таблиця 2.1. Порівняння основних функцій фронтенду і бекенду у контексті розширення для Chrome

Функція	Фронтенд	Бекенд
Відповідальність	Інтерфейс взаємодії з користувачем	Логіка роботи, комунікація з API
Основні технології	HTML, CSS, JavaScript	JavaScript, Service Workers
Компоненти	Popup, options, дашборд	Фонові скрипти, API-запити
Взаємодія з користувачем	Візуалізація, введення даних	Немає прямої взаємодії
Збереження даних	Тимчасові дані для інтерфейсу	Локальне сховище, робота з великими даними
Авторизація	Перенаправлення до сторінки OAuth	Обробка токенів, оновлення доступів
Механізм роботи	Виконання в активній вкладці	Фонове виконання незалежно від інтерфейсу

Таким чином, фронтенд і бекенд доповнюють один одного, створюючи функціональне та ефективне розширення. Фронтенд відповідає за зручність і доступність роботи для користувача, тоді як бекенд забезпечує надійність та автоматизацію складних операцій. Успішна реалізація обох складових є ключем до створення конкурентоспроможного продукту, який відповідає сучасним вимогам у сфері браузерних розширень.

Комунікація з API є центральною частиною функціоналу розширення, оскільки саме через API розширення отримує необхідні дані для подальшого аналізу чи відображення користувачеві. У сучасних браузерних розширеннях для реалізації таких запитів найчастіше використовуються два основних методи: fetch() і XMLHttpRequest. Обидва методи дозволяють взаємодіяти з зовнішніми вебсервісами через HTTP-запити (GET, POST, PUT, DELETE) і є базовими інструментами для отримання даних у форматах JSON або XML.

Метод fetch() є сучаснішим і більш зручним завдяки використанню промісів (promises), що значно спрощує обробку асинхронних операцій. Наприклад, під час виконання запиту з використанням fetch() можна легко

обробити успішну відповідь або помилку за допомогою методів .then() та .catch().

Натомість XMLHttpRequest є старішим методом і зазвичай використовується для забезпечення сумісності з API або системами, які не підтримують fetch(). Цей метод дозволяє виконувати синхронні та асинхронні запити, але його обробка складніша, оскільки використовуються колбеки (callbacks), що може ускладнити код.

Таблиця 2.2. Особливості та використання методів fetch() та XMLHttpRequest

Метод	**fetch()**	**XMLHttpRequest**
Дата впровадження	Сучасний стандарт	Старіший метод
Асинхронність	Використовує проміси	Використовує колбеки
Зручність коду	Простий, сучасний, більш читабельний код	Код складніший через необхідність роботи з подіями
Формати даних	Підтримка JSON, XML, тексту тощо	Підтримка JSON, XML, тексту тощо
Використання	Переважно використовується в сучасних розширеннях	Використовується для сумісності
Обробка помилок	Легка інтеграція з .catch()	Необхідно додатково налаштовувати

Реалізація HTTP-запитів із використанням fetch()

Приклад виконання GET-запиту з використанням fetch() для отримання даних через API Google Analytics:

```
fetch('https://www.googleapis.com/analytics/v3/data/ga', {
  method: 'GET',
  headers: {
   'Authorization': 'Bearer ' + accessToken, // Передача токена доступу для авторизації
   'Content-Type': 'application/json'
```

```
  }
})
  .then(response => {
    if (!response.ok) {
      throw new Error('Помилка запиту: ' + response.status);
    }
    return response.json(); // Перетворення відповіді у JSON-формат
  })
  .then(data => {
    console.log('Отримані дані:', data); // Робота з отриманими даними
  })
  .catch(error => {
    console.error('Помилка:', error); // Обробка помилок
  });
```

лістинг 2.2.1 Приклад Get-запиту за допомогою fetch

У цьому прикладі використовуються такі елементи:

1. URL запиту: Шлях до API, наприклад, Google Analytics.
2. Заголовки: Указуються токени авторизації та формат відповіді.
3. Обробка відповіді: Метод .then() використовується для роботи з успішними відповідями, а .catch() – для помилок.

Реалізація HTTP-запитів із використанням XMLHttpRequest:

Виконання аналогічного GET-запиту з використанням XMLHttpRequest:

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'https://www.googleapis.com/analytics/v3/data/ga', true);
xhr.setRequestHeader('Authorization', 'Bearer ' + accessToken); // Авторизація
xhr.setRequestHeader('Content-Type', 'application/json');

xhr.onload = function () {
  if (xhr.status === 200) {
```

```
    const data = JSON.parse(xhr.responseText); // Розбір відповіді у форматі JSON
    console.log('Отримані дані:', data);
  } else {
    console.error('Помилка запиту: ' + xhr.status);
  }
};
xhr.onerror = function () {
  console.error('Сталася помилка під час запиту.');
};
xhr.send(); // Відправка запиту
```

лістинг 2.2.2 Приклад Get-запиту до Google Analytics

Хоча результат схожий, код із використанням XMLHttpRequest виглядає громіздкішим через необхідність додатково налаштовувати функції для обробки подій onload та onerror.

Порівняння та рекомендації. Метод fetch() є рекомендованим для використання в сучасних розширеннях Chrome через простоту, ефективність і сумісність з більшістю сучасних API. XMLHttpRequest слід застосовувати лише в разі, якщо API не підтримує fetch() або вимагає специфічних функцій, доступних тільки у XMLHttpRequest, наприклад, моніторинг прогресу завантаження.

Обробка помилок. При роботі з API можуть виникати такі типові помилки:

- Відсутність доступу до API через некоректний токен.
- Перевищення лімітів запитів API (rate limits).
- Неправильний формат запиту або відповіді.
- Проблеми з мережею (timeout, помилки DNS).

Для таких ситуацій рекомендується реалізувати детальний механізм обробки помилок, включаючи перевірку статусу відповіді, інформування

користувача про проблеми та автоматичне повторення запиту в разі помилок мережі.

Таким чином, вибір між fetch() та XMLHttpRequest залежить від специфіки API та вимог до функціональності. Використання сучасних технологій дозволяє створити гнучке і надійне рішення для комунікації з API у розширенні для Chrome.

2.1.3. Використання Manifest v3 для безпечного виконання скриптів

Оновлення до Manifest V3 є важливою зміною в архітектурі браузерних розширень Chrome, яка впроваджує нові стандарти для забезпечення безпеки, продуктивності та конфіденційності користувачів. Це стосується, зокрема, способу виконання скриптів у розширеннях. Розуміння особливостей Manifest V3 є ключовим для розробки сучасних і безпечних розширень.

Manifest V3 – це нова версія маніфесту, що описує структуру та функціонал розширення. Це файл формату JSON, який визначає дозволи розширення, його компоненти (наприклад, popup, background), дозволені API та багато іншого. Основними змінами у Manifest V3 є заміна фонових сторінок (background pages) на сервісні робітники (service workers), обмеження доступу до небезпечних API та введення більш суворого контролю над виконанням скриптів [8].

Основні переваги Manifest V3:

1. Покращення безпеки. Скрипти тепер мають обмежений доступ до ресурсів, і всі дозволи потрібно явно визначати у файлі `manifest.json`.

2. Продуктивність. Сервісні робітники запускаються лише за необхідності, що знижує використання пам’яті та ресурсів.

3. Прозорість. Зміни спрямовані на підвищення довіри до розширень шляхом зменшення ризиків зловживання дозволами.

Особливості безпечного виконання скриптів у Manifest V3:

1. Service Workers. У Manifest V3 фонові сторінки замінені на сервісні робітники. Це JavaScript-файли, які працюють у фоновому режимі, але не

залишаються постійно активними. Сервісний робітник виконується тільки тоді, коли потрібна його дія (наприклад, обробка події). Він завершує роботу після виконання свого завдання.

2. Декларативні дозволи. Усі дозволи на використання API та доступ до ресурсів повинні бути явно зазначені у `manifest.json`. Наприклад, якщо розширення працює з API Twitter, потрібно додати дозвіл на доступ до відповідного домену:

```
"permissions": [
  "https://api.twitter.com/"
]
```

3. Content Security Policy (CSP). CSP обмежує джерела, з яких можна завантажувати скрипти, стилі або інші ресурси. Це допомагає запобігти атакам, пов'язаним із виконанням шкідливого коду. У файлі `manifest.json` CSP виглядає наступним чином:

```
"content_security_policy": {
  "extension_pages": "script-src 'self'; object-src 'self'"
}
```

4. Обмеження на виконання довільного коду. У Manifest V3 заборонено виконання довільного коду (inline scripts). Наприклад, не можна використовувати вбудовані у HTML скрипти:

```
<script>
  alert('Цей код не працюватиме у Manifest V3');
</script>
```

Замість цього всі скрипти повинні бути у окремих файлах.

5. Заміна WebRequest API на Declarative Net Request API. Це дозволяє розширенням змінювати або блокувати мережеві запити без прямого доступу до них, що підвищує безпеку:

```
"host_permissions": [
  "*://*.example.com/*"
]
```

Структура файлу `manifest.json` у V3

Файл `manifest.json` визначає всі ключові аспекти роботи розширення. Приклад файлу для розширення з використанням API та сервісного робітника виглядає так:

```
{
  "manifest_version": 3,
  "name": "Приклад розширення",
  "version": "1.0",
  "description": "Розширення для збору даних через API",
  "permissions": [
    "storage",
    "activeTab"
  ],
  "host_permissions": [
    "https://api.example.com/"
  ],
  "background": {
    "service_worker": "background.js"
  },
  "action": {
    "default_popup": "popup.html",
    "default_icon": "icon.png"
  },
  "content_security_policy": {
    "extension_pages": "script-src 'self'; object-src 'self'"
  }
}
```

лістинг 2.2.3 Приклад структури manifest.json

Переваги використання сервісних робітників:

Сервісні робітники є однією з ключових змін у Manifest V3, які підвищують продуктивність та безпеку. Вони працюють у фоні лише за необхідності, а після завершення завдань автоматично зупиняються. Це значно

зменшує навантаження на системні ресурси. Крім того, сервісні робітники ізольовані від основного коду, що робить їх більш захищеними від атак.

Приклад коду сервісного робітника:

```
self.addEventListener('install', () => {
  console.log('Сервісний робітник встановлено');
});
self.addEventListener('fetch', (event) => {
  console.log('Обробляється запит:', event.request.url);
});
```

лістинг 2.2.4 Приклад коду сервісного робітника

Рекомендації щодо впровадження Manifest V3:

1. Чітко визначити всі необхідні дозволи у файлі `manifest.json`, щоб уникнути помилок у роботі розширення.
2. Забезпечити відповідність політикам CSP, уникаючи inline-скриптів та небезпечних джерел.
3. Використовувати сервісні робітники для обробки фонових завдань і оптимізації використання ресурсів.
4. Замінити застарілі API (наприклад, WebRequest API) на новіші, безпечні альтернативи, такі як Declarative Net Request API.
5. Регулярно оновлювати розширення відповідно до змін у політиках Chrome.

Manifest V3 є важливим кроком у розвитку браузерних розширень, що дозволяє створювати безпечніші, продуктивніші та екологічніші інструменти для кінцевих користувачів. Дотримання нових стандартів гарантує відповідність сучасним вимогам до безпеки та конфіденційності.

2.1.4. Створення діаграм для розширення

Для повноцінного розуміння архітектури та функціональної поведінки розширення було створено низку UML-діаграм, які відображають як зовнішню взаємодію користувача з системою, так і внутрішню структуру компонентів

розширення. Діаграми допомагають формалізувати процеси, продемонструвати використані технології та сторонні сервіси, а також описати логіку роботи розширення при взаємодії з API [9].

На рис. 2.1 зображено діаграму прецендентів в якій продемонстровано основні сценарії взаємодії користувача з розширенням Chrome. Зокрема, користувач має можливість переглядати ключові метрики, налаштовувати токен API, задавати частоту збору даних та ініціювати процес збору інформації. Ця діаграма дозволяє чітко окреслити функціональні можливості розширення та межі взаємодії користувача із системою.

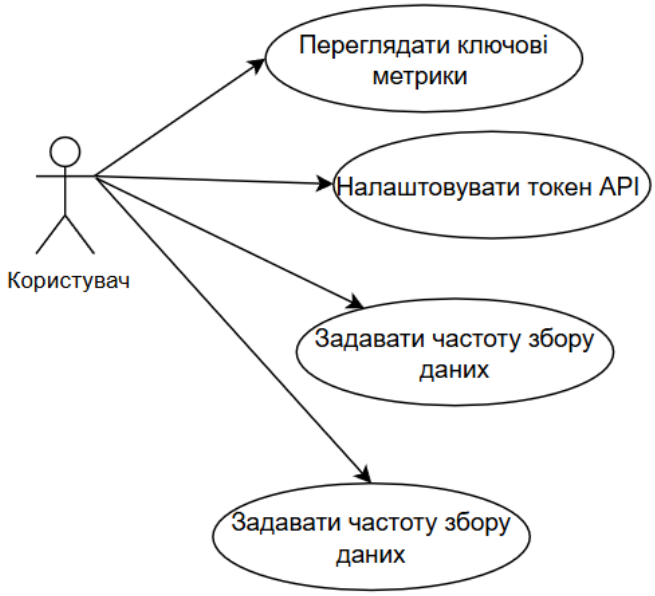

Рисунок 2.1 – Діаграма прецедентів взаємодії користувач з розширенням

На рис. 2.2 зображена діаграма послідовностей в якій показано логіку обробки запиту користувача при зборі даних через API. Процес починається з ініціації користувачем збору даних, після чого розширення передає запит до фонових скриптів (background script), де виконується авторизація через токен API.

Після успішної авторизації відправляється запит до зовнішнього API, зокрема Google API або іншого інтегрованого сервісу. Отримані дані

обробляються та зберігаються у локальному сховищі (Chrome Storage API), після чого інтерфейс розширення оновлює інформацію в дашборді.

Ця діаграма ілюструє ключову послідовність подій:

1. Користувач → інтерфейс: "Зібрати дані"
2. Інтерфейс → фоновий процес: "Авторизація"
3. Фоновий процес → API: "Отримати токен"
4. Фоновий процес → API: "Надіслати запит"
5. API → фоновий процес: "Відповідь з даними"
6. Фоновий процес → Chrome Storage: "Зберегти дані"
7. Фоновий процес → інтерфейс: "Відобразити результат"

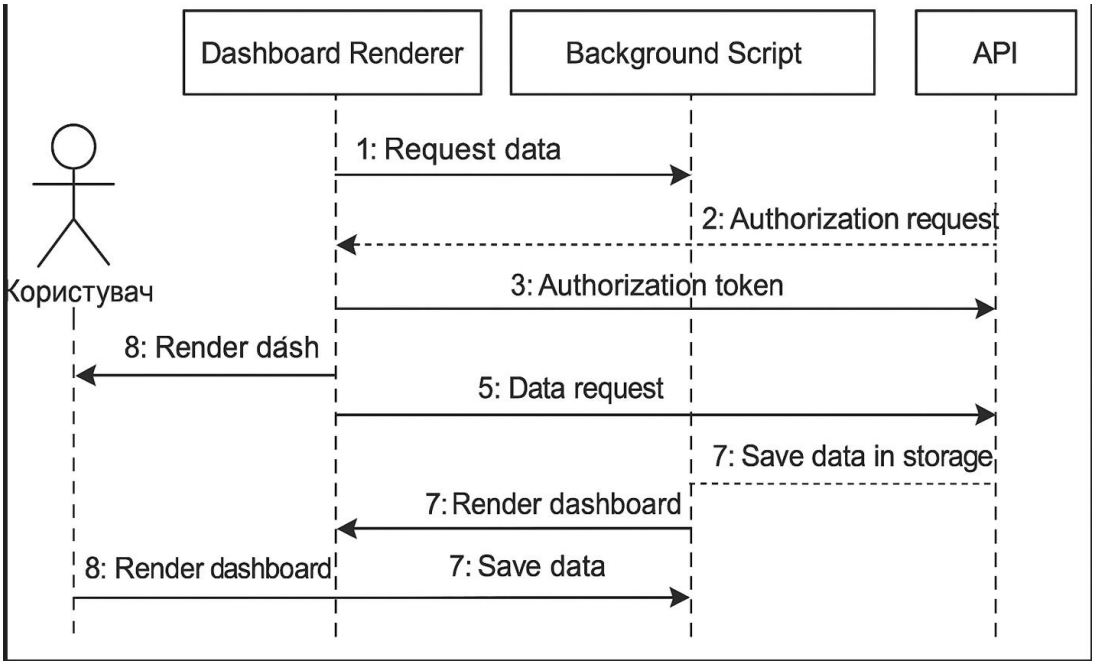

Рисунок 2.2 – Діаграма послідовностей обробки запиту користувача

На діаграмі зображено основні компоненти, з яких складається розширення Chrome, а також зовнішні сервіси та бібліотеки, з якими воно взаємодіє. Основна мета діаграми – відобразити структуру системи, її модулі та залежності між ними.

До складу системи входять такі основні компоненти:

- Інтерфейс користувача (UI) – реалізований за допомогою HTML/CSS/JavaScript. Забезпечує відображення метрик, кнопок керування, налаштувань тощо.
- Фоновий скрипт (Background script) – обробляє авторизацію, збір даних, обмін із зовнішніми API та зберігання інформації.
- Chrome Storage API – використовується для зберігання конфігурацій, токенів та зібраних даних локально у браузері.
- Зовнішні API – Google API, Facebook Graph API, Twitter API – джерела, з яких отримуються аналітичні дані.
- Бібліотеки сторонніх розробників:
- Chart.js – для побудови графіків і візуалізації метрик.
- OAuth 2.0 JS Client – для реалізації механізму авторизації через сторонні сервіси.

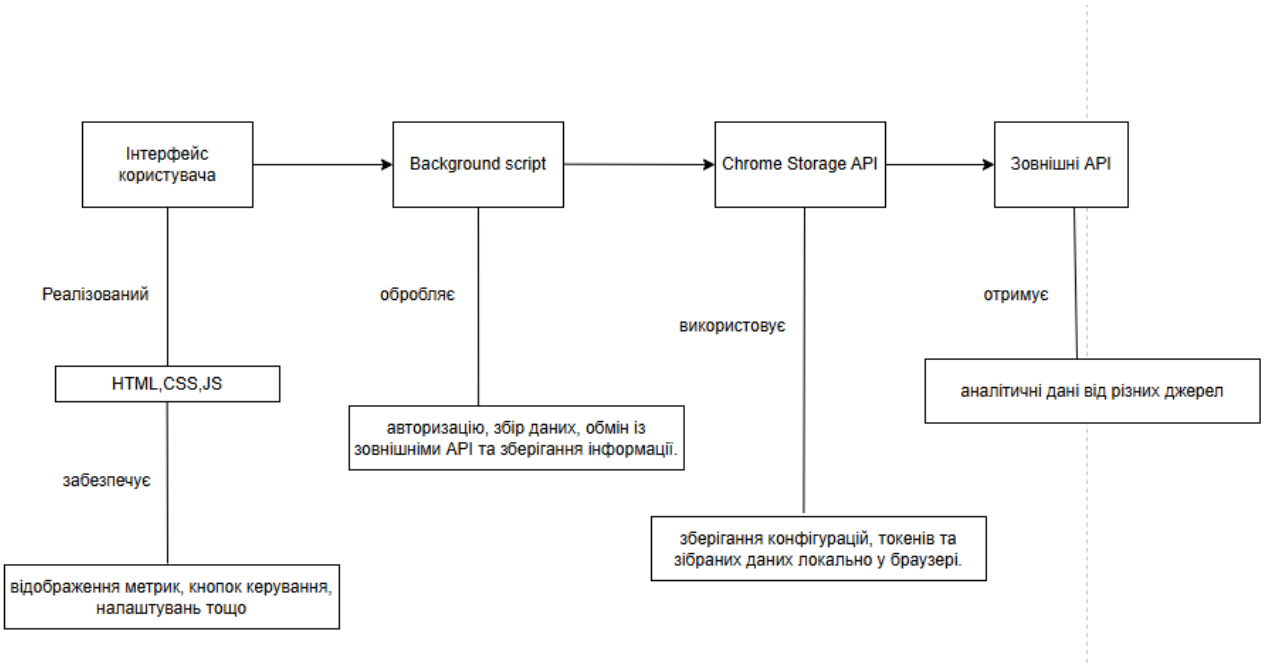

Рисунок 2.3 – Діаграма компонентів розширення

Створені діаграми дозволили формалізувати ключові аспекти роботи розширення Chrome як на рівні взаємодії з користувачем, так і на рівні внутрішньої архітектури.

Діаграма прецедентів окреслила основні функціональні сценарії, доступні користувачу.

Діаграма послідовностей показала динамічну логіку збору даних та обміну з API.

Діаграма компонентів продемонструвала структуру системи, її модулі та сторонні залежності.

Таке графічне подання суттєво спрощує розуміння процесів, дозволяє ідентифікувати можливі вузькі місця та забезпечує основу для подальшого розширення або оптимізації розширення.

2.2. Інтеграція API

2.2.1. Опис взаємодії з API, обробка авторизації (OAuth 2.0)

Інтеграція API є центральним компонентом функціоналу розширення для Chrome, який забезпечує автоматизований доступ до даних зовнішніх сервісів. Для цього використовуються стандартизовані протоколи взаємодії, такі як REST API, та механізми авторизації, які гарантують безпечний і контрольований доступ до даних користувачів. Одним із найпоширеніших методів авторизації є OAuth 2.0, який забезпечує безпечний доступ до API без передачі облікових даних (логінів і паролів) третім сторонам.

OAuth 2.0 – це протокол авторизації, що дозволяє додаткам отримувати доступ до ресурсів користувачів на сторонніх платформах (Google, Facebook, Twitter) без необхідності передачі паролів. Замість цього використовується механізм токенів доступу (access tokens), які видаються сервером авторизації та мають обмежений доступ до ресурсів.

Основні переваги OAuth 2.0:

1. Безпека: користувачі не передають паролі, а лише дозволяють додатку доступ до визначених ресурсів.
2. Гнучкість: доступ може бути обмежений за часом, типом ресурсів або рівнем привілеїв.

3. Стандартизація: OAuth 2.0 підтримується більшістю популярних платформ і API.

Основні етапи роботи протоколу OAuth 2.0:

1. Користувач надає розширенню дозвіл на доступ до своїх даних через API.

2. Розширення перенаправляє користувача до сторінки авторизації платформи (наприклад, Google або Facebook).

3. Після авторизації сервер видає токен доступу, який використовується розширенням для виконання запитів до API.

4. Сервер API перевіряє токен перед обробкою запиту та повертає необхідні дані.

Процес авторизації OAuth 2.0 у розширенні для Chrome включає наступні кроки:

1. Перенаправлення користувача на сторінку авторизації. Розширення відкриває нове вікно або вкладку з URL авторизації сервісу. Наприклад, для Google OAuth URL виглядає так:

https://accounts.google.com/o/oauth2/v2/auth?client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&response_type=token&scope=https://www.googleapis.com/auth/analytics.readonly

Параметри URL включають:

- `client_id` – ідентифікатор додатку.
- `redirect_uri` – URL для перенаправлення після авторизації.
- `response_type` – тип відповіді (наприклад, токен).
- `scope` – список дозволів, які запитує розширення.

2. Отримання токена доступу. Після успішної авторизації користувач перенаправляється на вказаний `redirect_uri`, де у параметрах URL передається токен доступу:

https://example.com/callback#access_token=ВАШ_ТОКЕН&token_type=Bearer&expires_in=3600

Розширення витягує токен з URL для подальшого використання.

3. Збереження токена. Токен зберігається у локальному сховищі (наприклад, Chrome Storage API) для подальших запитів:

```
chrome.storage.local.set({ accessToken: token }, function() {
  console.log('Токен збережено');
});
```

4. Використання токена для запитів до API. Під час запитів до API токен передається у заголовках:

```
fetch('https://www.googleapis.com/analytics/v3/data/ga', {
  method: 'GET',
  headers: {
    'Authorization': 'Bearer ' + token
  }
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error('Помилка:', error));
```

5. Оновлення токена. Якщо токен втрачає дійсність (наприклад, через завершення терміну дії), розширення повинно повторно виконати авторизацію або оновити токен через спеціальний запит.

Реалізація авторизації OAuth 2.0 у розширенні

Приклад функції для запуску авторизації:

```
function authorize() {
  const authUrl = 'https://accounts.google.com/o/oauth2/v2/auth?' +
    'client_id=НАШ_CLIENT_ID&' +
    'redirect_uri=НАШ_REDIRECT_URI&' +
    'response_type=token&' +
    'scope=https://www.googleapis.com/auth/analytics.readonly';

  chrome.identity.launchWebAuthFlow(
    { url: authUrl, interactive: true },
    function(responseUrl) {
      if (chrome.runtime.lastError) {
        console.error('Помилка авторизації:', chrome.runtime.lastError);
        return;
      }
      const token = extractTokenFromUrl(responseUrl);
```

```
            saveToken(token);
          }
        );
      }
      function extractTokenFromUrl(url) {
        const params = new URL(url).hash.substring(1).split('&');
        const tokenParam = params.find(param =>
param.startsWith('access_token='));
        return tokenParam.split('=')[1];
      }

      function saveToken(token) {
        chrome.storage.local.set({ accessToken: token }, function() {
          console.log('Токен збережено:', token);
        });
      }
```

лістинг 2.2.5 Приклад функції для запуску авторизації

Переваги використання OAuth 2.0:

1. Безпечний доступ. Токени дозволяють обмежити доступ до даних, зменшуючи ризики витоку інформації.

2. Гнучкість. Можна запитувати доступ тільки до необхідних даних, зберігаючи конфіденційність користувача.

3. Стандартизація. Більшість популярних платформ, таких як Google, Facebook і Twitter, підтримують OAuth 2.0, що забезпечує зручність інтеграції.

Можливі проблеми та їх вирішення:

1. Завершення терміну дії токена. Використовуйте функцію оновлення токенів (refresh token), щоб не повторювати процес авторизації.

2. Обмеження доступу до API. Запитуйте тільки необхідні дозволи (`scope`), щоб уникнути відмови в доступі.

3. Розрив сесії користувача. Зберігайте токени у локальному сховищі та регулярно перевіряйте їхню дійсність.

Інтеграція OAuth 2.0 у розширення Chrome є ефективним і безпечним способом взаємодії з API. Дотримання стандартів OAuth гарантує

конфіденційність користувачів, надійність роботи розширення та його сумісність із сучасними сервісами [10].

2.2.2. Збір даних через API запити

Процес збору даних через API запити є центральним у реалізації функціоналу розширення Chrome. Це забезпечує можливість автоматичного отримання актуальної інформації з зовнішніх джерел, таких як Google, Facebook або Twitter. Для цього використовується взаємодія з REST API, яка дозволяє надсилати запити до серверів, отримувати відповіді у форматі JSON або XML та обробляти отримані дані для подальшого аналізу та візуалізації.

Ключові компоненти збору даних через API:

1. REST API: Найпоширеніший тип API, що працює за допомогою стандартних HTTP-запитів (GET, POST, PUT, DELETE). Використовується для інтеграції з такими сервісами, як Google Analytics, Facebook Graph API або Twitter API.

2. Методи HTTP:

- GET: Для отримання даних (наприклад, аналітика сайту, дані про користувача).

- POST: Для надсилання даних на сервер (наприклад, створення нового запису).

- PUT/DELETE: Для оновлення або видалення даних (рідше використовується у розширеннях).

3. Формати даних: Відповіді зазвичай повертаються у форматах JSON (переважно) або XML, що дозволяє легко працювати з ними у JavaScript.

4. Авторизація: Для доступу до даних зазвичай використовується токен, отриманий через OAuth 2.0, який передається в заголовках HTTP-запиту.

5. Асинхронна взаємодія: Запити до API виконуються асинхронно, щоб не блокувати роботу розширення під час очікування відповіді.

Етапи збору даних через API

1. Підготовка запиту

Перед виконанням запиту необхідно підготувати URL, метод запиту (наприклад, GET) та заголовки, що включають авторизаційний токен. Наприклад, для отримання даних через Google Analytics API потрібно вказати URL запиту та передати токен у заголовку Authorization:

```
const url = 'https://www.googleapis.com/analytics/v3/data/ga?ids=ga:123456&metrics=sessions';
const headers = {
  'Authorization': 'Bearer ' + accessToken,
  'Content-Type': 'application/json'
};
```

2. Виконання запиту

Для виконання запиту можна використовувати fetch() або XMLHttpRequest. Сучасні розширення переважно використовують fetch() завдяки його простоті та сумісності з промісами.

Приклад виконання GET-запиту:

```
fetch(url, {
  method: 'GET',
  headers: headers
})
  .then(response => {
    if (!response.ok) {
      throw new Error('Помилка запиту: ' + response.status);
    }
    return response.json();
  })
  .then(data => {
    console.log('Отримані дані:', data);
    обробитиДані(data);
  })
  .catch(error => {
```

```
  console.error('Помилка:', error);
 });
```

лістинг 2.2.6 Приклад Get-запиту

3. Обробка відповіді

Отримані дані у форматі JSON необхідно розпарсити та підготувати для подальшого використання. Наприклад, можна створити таблицю або графік:

```
javascript
function обробитиДані(data) {
  const sessions = data.totalsForAllResults['ga:sessions'];
  console.log('Кількість сесій:', sessions);
}
```

4. Зберігання даних

Зібрані дані можна зберігати у локальному сховищі Chrome (chrome.storage) або відображати безпосередньо у дашборді користувача:

```
javascript
chrome.storage.local.set({ sessionsData: data }, function() {
  console.log('Дані збережено.');
});
```

5. Візуалізація даних

Для представлення даних у зручному вигляді можна використовувати бібліотеки для візуалізації, наприклад Chart.js або D3.js. Приклад створення графіка:

```
javascript
const ctx = document.getElementById('myChart').getContext('2d');
new Chart(ctx, {
  type: 'bar',
  data: {
    labels: ['Сесії'],
    datasets: [{
```

```
        label: 'Кількість сесій',
        data: [sessions],
        backgroundColor: 'rgba(75, 192, 192, 0.2)',
        borderColor: 'rgba(75, 192, 192, 1)',
        borderWidth: 1
      }]
    },
    options: {
      responsive: true,
      maintainAspectRatio: false
    }
  });
```

Робота з різними API

Google Analytics API

Дозволяє отримувати дані про трафік, сесії, джерела переходів та інше.

Необхідна авторизація через OAuth 2.0.

Приклад запиту для отримання сесій:

javascript

```
const url = 'https://www.googleapis.com/analytics/v3/data/ga?ids=ga:123456&metrics=sessions';
```

Facebook Graph API

Надає доступ до інформації про профілі, сторінки та публікації.

Приклад запиту для отримання постів:

javascript

```
const url = 'https://graph.facebook.com/v10.0/me/posts?access_token=' + accessToken;
```

Twitter API

Використовується для аналізу твітів, ретвітів і активності користувачів.

Приклад запиту для отримання останніх твітів:

javascript

```
const url = 'https://api.twitter.com/2/tweets?ids=123456';
```

лістинг 2.2.7 Приклад Twitter API

Проблеми та їх вирішення:

1. Перевищення лімітів API (rate limits): Зменшуйте кількість запитів, використовуючи кешування.
2. Проблеми з авторизацією: Регулярно перевіряйте термін дії токенів та виконуйте їх оновлення.
3. Помилки відповіді API: Реалізуйте обробку помилок, включаючи повторні запити у разі тимчасових проблем.

Збір даних через API запити є ключовою складовою розширення для Chrome, що забезпечує отримання актуальної інформації з зовнішніх джерел. Завдяки використанню сучасних методів, таких як fetch(), асинхронна взаємодія з API стає ефективною, надійною та простою в реалізації. Інтеграція з популярними API, такими як Google Analytics, Facebook Graph або Twitter API, розширює функціональність розширення та відкриває нові можливості для автоматизації аналізу даних [11].

2.3. Створення базових інтерфейсів для візуалізації даних користувачу

Візуалізація даних є ключовим елементом користувацького досвіду розширення Chrome, оскільки саме завдяки зрозумілому і привабливому інтерфейсу користувач може ефективно аналізувати отриману інформацію. У цьому розділі розглядається процес створення базових інтерфейсів для відображення зібраних даних за допомогою графіків, таблиць та інших інтерактивних елементів [12].

Основна мета створення інтерфейсу для візуалізації даних – забезпечити користувача зручним і зрозумілим способом аналізу зібраної інформації. Візуалізація повинна бути:

- Інтуїтивно зрозумілою: Мінімальна кількість зусиль для користувача для розуміння інтерфейсу.
- Інформативною: Подання ключових показників у зручній формі, що сприяє прийняттю рішень.
- Динамічною: Дані повинні оновлюватися в реальному часі (або після виконання запиту).

Інтерфейс може включати такі основні елементи:

1. Дашборд: Централізоване вікно для перегляду ключових показників, наприклад, кількості сесій, відвідувачів або трендів.
2. Графіки та діаграми: Відображення даних у вигляді кругових, стовпчастих або лінійних графіків.
3. Таблиці: Деталізоване представлення даних, наприклад, список постів із Facebook або твітів із Twitter.
4. Фільтри: Елементи, що дозволяють користувачу обирати параметри (дата, категорії тощо) для перегляду даних.

Дашборд є центральним елементом інтерфейсу, на якому розміщуються всі основні компоненти. Для його створення використовуються стандартні вебтехнології: HTML, CSS та JavaScript.

Приклад структури HTML для дашборда:

```
html
<div id="dashboard">
  <h1>Дашборд</h1>
  <div id="charts">
    <canvas id="chart1"></canvas>
    <canvas id="chart2"></canvas>
  </div>
  <div id="table">
    <table>
      <thead>
        <tr>
          <th>Назва</th>
          <th>Значення</th>
        </tr>
      </thead>
```

```
      <tbody>
        <!-- Дані додаються динамічно -->
      </tbody>
    </table>
  </div>
</div>
```

CSS для оформлення:

```
css

#dashboard {
  font-family: Arial, sans-serif;
  margin: 20px;
}
#charts {
  display: flex;
  justify-content: space-around;
  margin-bottom: 20px;
}
canvas {
  width: 400px;
  height: 300px;
}
table {
  width: 100%;
  border-collapse: collapse;
}
th, td {
  border: 1px solid #ddd;
  padding: 8px;
}
th {
  background-color: #f4f4f4;
  text-align: left;
}
```

лістинг 2.2.8 Приклад Html та CSS структури

Для створення графіків та діаграм використовуваються спеціалізовані бібліотеки:

- Chart.js: Проста у використанні бібліотека для створення адаптивних графіків.

- D3.js: Потужна бібліотека для складних візуалізацій із високим рівнем кастомізації.
- Google Charts: Легка інтеграція з інструментами Google для побудови графіків.

Приклад створення стовпчастого графіка за допомогою Chart.js:

```
javascript

const ctx = document.getElementById('chart1').getContext('2d');
const myChart = new Chart(ctx, {
  type: 'bar',
  data: {
    labels: ['Січень', 'Лютий', 'Березень'],
    datasets: [{
      label: 'Кількість сесій',
      data: [50, 75, 100],
      backgroundColor: [
        'rgba(255, 99, 132, 0.2)',
        'rgba(54, 162, 235, 0.2)',
        'rgba(255, 206, 86, 0.2)'
      ],
      borderColor: [
        'rgba(255, 99, 132, 1)',
        'rgba(54, 162, 235, 1)',
        'rgba(255, 206, 86, 1)'
      ],
      borderWidth: 1
    }]
  },
  options: {
    responsive: true,
    scales: {
      y: {
        beginAtZero: true
      }
    }
  }
});
```

лістинг 2.2.9 Приклад створення графіка за допомогою Chart.js

Реалізація таблиць для даних:

Таблиці дозволяють деталізувати інформацію, зібрану через API. Наприклад, можна показати список користувачів або постів із Facebook Graph API.

Додавання даних до таблиці динамічно:

```
javascript

function addDataToTable(data) {
  const tbody = document.querySelector('#table tbody');
  data.forEach(item => {
    const row = document.createElement('tr');
    row.innerHTML = `<td>${item.name}</td><td>${item.value}</td>`;
    tbody.appendChild(row);
  });
}
// Приклад даних
const exampleData = [
  { name: 'Сесії', value: 123 },
  { name: 'Користувачі', value: 456 }
];
addDataToTable(exampleData);
```

Фільтрація даних:

Додатково можна реалізувати фільтри для налаштування параметрів перегляду. Наприклад, вибір періоду часу або категорій:

```
html
<label for="date">Оберіть дату:</label>
<input type="date" id="date" name="date">
<button onclick="filterData()">Фільтрувати</button>
```

Функція для фільтрації даних:

```
javascript
function filterData() {
  const date = document.getElementById('date').value;
  console.log('Фільтрація даних для дати:', date);
  // Додати логіку для оновлення даних
}
```

Рекомендації для створення інтерфейсу:

1. Простота. Інтерфейс має бути легким у використанні навіть для недосвідчених користувачів.

2. Адаптивність. Забезпечте підтримку різних розмірів екранів за допомогою CSS або бібліотек, таких як Bootstrap.

3. Продуктивність. Використовуйте асинхронне завантаження даних для уникнення затримок.

4. Динамічність. Забезпечте можливість оновлення графіків та таблиць у реальному часі.

Створення базових інтерфейсів для візуалізації даних є важливим етапом розробки розширення. Використання сучасних інструментів для візуалізації (Chart.js, D3.js) у поєднанні з адаптивним дизайном дозволяє створити ефективний, зручний і привабливий інтерфейс, який задовольняє потреби користувачів у роботі з великими обсягами даних [13].

Висновки до Розділу 2

Розділ 2, присвячений проєктуванню розширення для Chrome з інтеграцією зовнішніх API, дозволяє зробити кілька важливих висновків. По-перше, архітектура розширення повинна включати фронтенд і бекенд складові, що забезпечують ефективну взаємодію користувача з інтерфейсом і фонову обробку даних. Фронтенд відповідає за інтуїтивно зрозумілий інтерфейс для користувача, тоді як бекенд реалізує логіку роботи розширення, включаючи збереження даних і виконання запитів до API.

По-друге, комунікація з API є основною складовою функціональності розширення. Використання сучасних методів, таких як fetch(), забезпечує простоту реалізації та високу ефективність обробки запитів. Особлива увага приділяється безпеці, включаючи використання токенів OAuth 2.0 для авторизації та дотримання вимог Content Security Policy (CSP), які захищають розширення від зовнішніх загроз.

Третій висновок стосується збору та візуалізації даних. Інтеграція з популярними API, такими як Google API, Facebook Graph API та Twitter API, дозволяє автоматизувати процес збору даних і забезпечити їх подання у

зручному вигляді. Реалізація дашбордів, графіків і таблиць на основі таких бібліотек, як Chart.js або D3.js, сприяє покращенню користувацького досвіду.

Крім того, для більш чіткого розуміння структури та логіки роботи розширення було створено діаграми прецедентів, послідовності та компонентів. Ці діаграми допомагають візуалізувати взаємодію між користувачем, інтерфейсом і зовнішніми сервісами, визначити ключові функціональні можливості та відобразити архітектуру системи на рівні її модулів.

Загалом, під час проєктування розширення для Chrome слід враховувати вимоги до безпеки, ефективності та адаптивності. Використання сучасних підходів до інтеграції API, оптимізація запитів і створення зручних інтерфейсів забезпечать надійність і зручність у використанні розширення. Це створює основу для його успішної реалізації та подальшого використання.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ РОЗШИРЕННЯ

3.1. Опис процесу реалізації

У процесі розробки розширення для Chrome нами було реалізовано повний цикл інтеграції зовнішніх API, автоматизації збору даних та їх відображення в інтерфейсі користувача. Робота над розширенням включала кілька ключових етапів.

На першому етапі нами було створено файл `manifest.json`, який описує структуру розширення, його основні компоненти та необхідні дозволи. Відповідно до вимог Manifest V3, ми визначили використання сервісного робітника (service worker) для фонового виконання завдань, дозволи для доступу до API та політику безпеки (Content Security Policy) [14].

На другому етапі реалізували сервісний робітник (`background.js`), який відповідає за обробку запитів до API. Він забезпечує асинхронну взаємодію з API, обробку отриманих даних і їх збереження у локальному сховищі Chrome. Зокрема, нами було інтегровано Google Analytics API для збору статистики трафіку, а також Facebook Graph API для отримання даних про публікації.

Далі було розроблено інтерфейс користувача, що включає popup-сторінку для швидкого доступу до функціоналу, сторінку налаштувань (options) для управління інтеграціями API, а також дашборд для візуалізації отриманих даних. Всі компоненти інтерфейсу створено з використанням HTML, CSS та JavaScript.

На фінальному етапі було виконано інтеграцію API. Ми реалізували механізми авторизації через OAuth 2.0, що дозволяють безпечно отримувати доступ до даних користувачів. Для обробки даних і створення запитів використовували сучасні методи JavaScript, такі як `fetch()`.

3.2. Валідація та тестування API інтеграцій

3.2.1. Перевірка відповідей від API

Для забезпечення коректної роботи розширення ми провели ретельне тестування інтеграцій з API.

Під час тестування ми перевіряли всі відповіді API на предмет правильності формату та статусів. Наприклад, у запитах до Google Analytics API ми переконалися, що відповіді повертаються у форматі JSON та містять необхідні дані, такі як кількість сесій, джерела трафіку та інші метрики.

Для обробки помилок нами було реалізовано перевірку статус-кодів HTTP. У разі невдалої відповіді (наприклад, статус 401 – "Unauthorized") розширення повідомляє користувача про необхідність повторної авторизації [15].

3.2.2. Тестування сценаріїв збору та аналізу даних

Ми протестували сценарії збору даних для різних умов, включаючи:

- Масштабні запити до API з великими обсягами даних.
- Роботу з обмеженими дозволами API.
- Відновлення роботи після виникнення помилок, наприклад, при закінченні терміну дії токену OAuth.

Також було проведено стрес-тестування розширення для оцінки його продуктивності при обробці великих обсягів даних [16].

3.3. Розробка інтерфейсу для користувача

Інтерфейс розширення створено з акцентом на зручність і простоту використання.

1. Popup-сторінка забезпечує швидкий доступ до функціоналу розширення. На цій сторінці користувач може запустити збір даних, переглянути основні метрики, наприклад, кількість сесій чи постів, і здійснити авторизацію.

Функціонал Popup-сторінки:

1. Перегляд ключових метрик:

Інформація про сесії та кількість користувачів завантажується з локального сховища Chrome і відображається на сторінці.

2. Налаштування інтеграцій API:

Користувач може ввести токен API, який зберігається в локальному сховищі для використання під час запитів.

3. Запуск процесу збору даних:

Кнопка "Запустити збір даних" виконує запит до API за допомогою збереженого токену та оновлює метрики на сторінці.

Ця реалізація забезпечує базовий функціонал Popup-сторінки для взаємодії з користувачем і демонструє, як можна ефективно реалізувати інтеграцію API.

Інтерфейс включає секції для перегляду ключових метрик, налаштування токену API та запуску процесу збору даних. Статус дій користувача відображається у реальному часі.

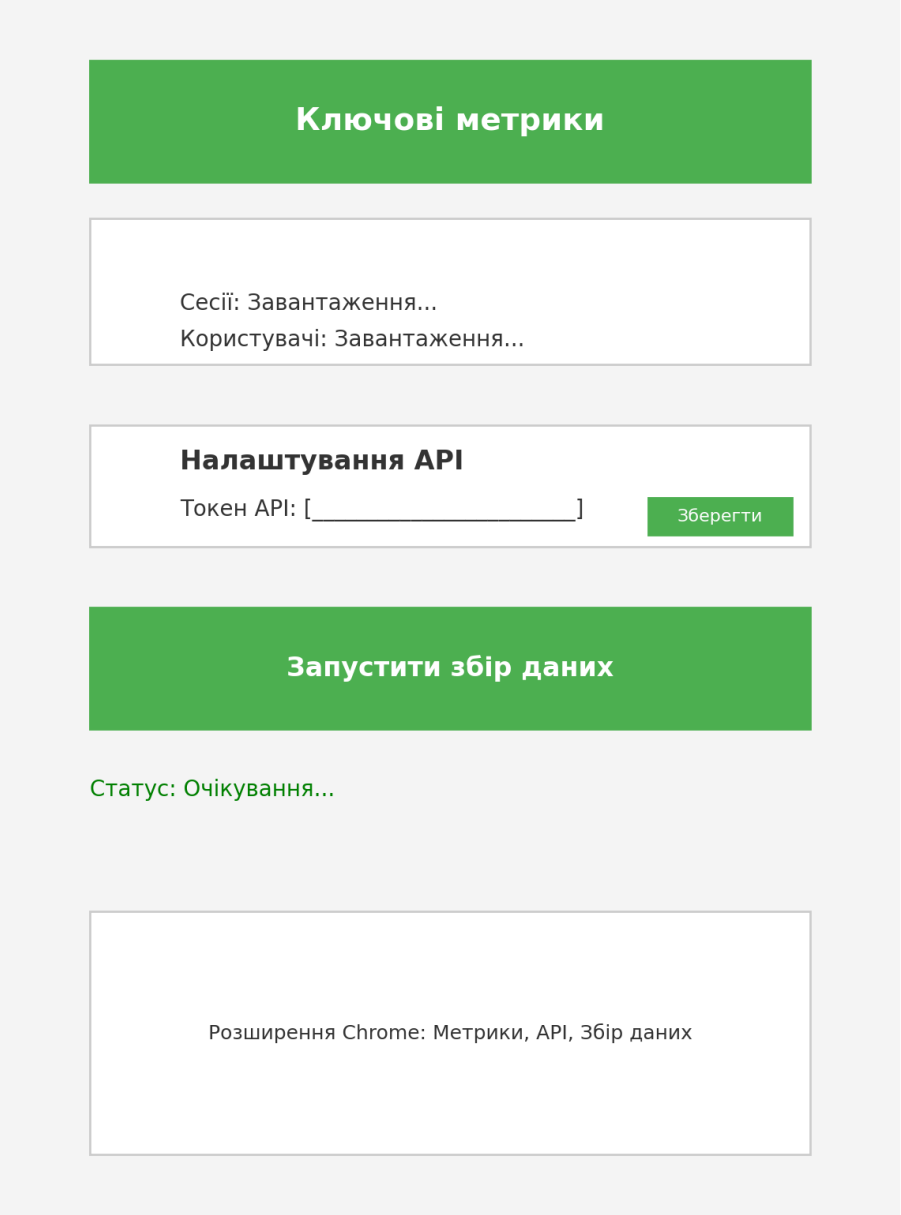

Рисунок 3.1 – Графічне зображення Popup-сторінки розширення для Chrome.

2. Сторінка налаштувань (options) дозволяє користувачу налаштовувати частоту збору даних, керувати токенами для авторизації та обирати параметри для API-запитів. Наприклад, користувач може вказати період часу, за який потрібно зібрати дані.

Функціонал сторінки налаштувань:

1. Управління токеном API:

- Користувач може ввести або змінити токен для авторизації API.
- Токен зберігається у локальному сховищі браузера.

2. Налаштування частоти збору даних:

- Користувач може вказати, як часто розширення повинно збирати дані (наприклад, кожні 30 хвилин).

- Частота зберігається для подальшого використання у фонових скриптах.

3. Вибір параметрів API-запитів:

- Користувач може налаштувати період часу для збору даних, вказавши початкову та кінцеву дати.

- Ці параметри передаються у запитах до API.

Ця сторінка надає користувачу повний контроль над налаштуваннями розширення та дозволяє адаптувати його роботу до індивідуальних потреб.

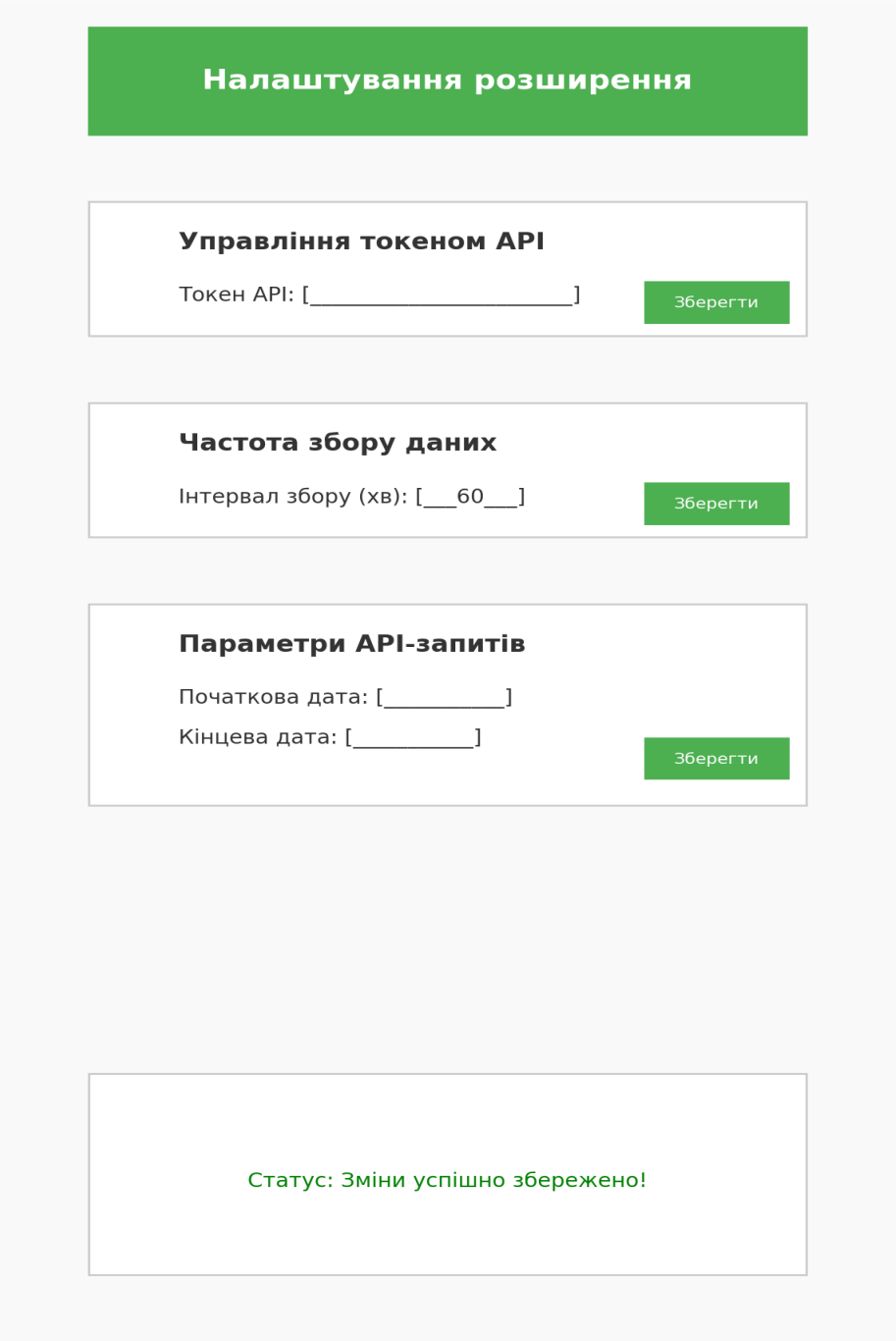

Рисунок 3.2 – Сторінка налаштувань

Структура сторінки налаштувань (options) для розширення Chrome у вигляді дерева компонентів. Ця структура показує, як організовані елементи на сторінці:

```
php
options.html
|
```

```
      ├── <div id="options"> (Основний контейнер сторінки)
      │   ├── <h1>Налаштування розширення</h1> (Заголовок сторінки)
      │   │
      │   ├── <div class="section"> (Секція: Управління токеном API)
      │   │   ├── <h2>Управління токеном API</h2>
      │   │   ├── <label for="api-token">Токен API:</label>
      │   │   ├── <input type="text" id="api-token"> (Поле введення токена API)
      │   │   └── <button id="save-token">Зберегти токен</button> (Кнопка
збереження токена)
      │   │
      │   ├── <div class="section"> (Секція: Частота збору даних)
      │   │   ├── <h2>Частота збору даних</h2>
      │   │   ├── <label for="data-frequency">Інтервал збору (у
хвилинах):</label>
      │   │   ├── <input type="number" id="data-frequency"> (Поле введення
частоти збору)
      │   │   └── <button id="save-frequency">Зберегти частоту</button> (Кнопка
збереження частоти)
      │   │
      │   ├── <div class="section"> (Секція: Параметри API-запитів)
      │   │   ├── <h2>Параметри API-запитів</h2>
      │   │   ├── <label for="start-date">Початкова дата:</label>
      │   │   ├── <input type="date" id="start-date"> (Поле введення початкової
дати)
      │   │   ├── <label for="end-date">Кінцева дата:</label>
      │   │   ├── <input type="date" id="end-date"> (Поле введення кінцевої дати)
      │   │   └── <button id="save-parameters">Зберегти параметри</button>
(Кнопка збереження параметрів)
      │   │
      │   └── <p id="status"> (Рядок статусу, що відображає повідомлення про
збереження)
```

Опис компонентів:

1. Головний контейнер :

- Включає всі секції сторінки налаштувань.

2. Секція "Управління токеном API":

- Поле для введення токена (<input type="text">).

- Кнопка для збереження токена (<button>).

3. Секція "Частота збору даних":

- Поле для введення інтервалу збору даних у хвилинах (<input type="number">).

- Кнопка для збереження частоти (<button>).

4. Секція "Параметри API-запитів":

- Поля для вибору початкової та кінцевої дати (<input type="date">).

- Кнопка для збереження параметрів (<button>).

5. Рядок статусу <p id="status">:

- Відображає повідомлення про результат збереження змін (успішне чи з помилкою).

Ця структура дозволяє легко зрозуміти, як організовані елементи сторінки, і на її основі створювати нові компоненти або розширювати існуючий функціонал.

Сценарій використання:

1. Початкові налаштування через Options-сторінку

1. Користувач відкриває Options-сторінку через меню розширення.

2. Вводить токен API у полі "Токен API" та натискає "Зберегти".

- Дія: Токен зберігається у локальному сховищі Chrome за допомогою chrome.storage.local.

3. Налаштовує частоту збору даних (наприклад, кожні 60 хвилин) і натискає "Зберегти частоту".

- Дія: Інтервал зберігається для подальшого використання фоновим скриптом.

4. Вибирає початкову і кінцеву дату для збору даних і натискає "Зберегти параметри".

- Дія: Дати зберігаються у локальному сховищі для використання в API-запитах.

5. На сторінці з’являється повідомлення: "Зміни успішно збережені!"

2. Робота через Popup-сторінку

1. Користувач натискає на значок розширення Chrome.
2. Відкривається Popup-сторінка, яка показує:

- Поточні ключові метрики (наприклад, кількість сесій і користувачів).
- Кнопку "Запустити збір даних".

3. Користувач натискає "Запустити збір даних".

- Дія: Popup-сторінка надсилає повідомлення фоновому скрипту через `chrome.runtime.sendMessage`.
- Дані, що передаються: Токен API, частота збору та обрані параметри.

4. На сторінці з’являється повідомлення: "Збір даних розпочато..."

3. Фоновий збір даних

1. Фоновий скрипт отримує повідомлення від Popup-сторінки та ініціалізує збір даних.

2. Виконується API-запит із використанням токена, частоти та параметрів:

- Токен: Береться зі сховища Chrome.
- Параметри: Початкова і кінцева дати передаються у запиті.
- Частота: Використовується для повторюваних запитів.

3. Фоновий скрипт обробляє відповідь API, зберігає ключові метрики (наприклад, кількість сесій, користувачів) у локальному сховищі:

```
javascript

chrome.storage.local.set({
  sessions: data.sessions,
  users: data.users
});
```

4. Оновлення метрик у Popup-сторінці

1. Після завершення збору даних фоновий скрипт надсилає повідомлення Popup-сторінці:

```
javascript
```

```
chrome.runtime.sendMessage({ type: 'updateMetrics', data: { sessions, users } });
```

2. Popup-сторінка отримує повідомлення та оновлює відображені метрики:

```
javascript

chrome.runtime.onMessage.addListener((message) => {
  if (message.type === 'updateMetrics') {
    document.getElementById('sessions').textContent = message.data.sessions;
    document.getElementById('users').textContent = message.data.users;
    document.getElementById('status').textContent = 'Збір даних завершено!';
  }
});
```

Взаємодія між компонентами

```
plaintext

1. Options-сторінка:
   Користувач вводить токен API, параметри та частоту → Дані зберігаються у `chrome.storage.local`.

2. Popup-сторінка:
   - Завантажує метрики із `chrome.storage.local`.
   - Надсилає запит на запуск збору даних фоновому скрипту через `chrome.runtime.sendMessage`.

3. Фоновий скрипт:
   - Отримує запит, виконує API-запит з токеном і параметрами.
   - Зберігає результати (метрики) у `chrome.storage.local`.
   - Надсилає повідомлення Popup-сторінці для оновлення метрик.

4. Popup-сторінка:
   - Отримує оновлені дані та відображає їх у інтерфейсі.
```

Сценарій на прикладі коду

Options-сторінка

```
javascript
```

```
document.getElementById('save-token').addEventListener('click', () => {
  const token = document.getElementById('api-token').value;
  chrome.storage.local.set({ apiToken: token }, () => alert('Токен збережено!'));
});
```

Popup-сторінка

```
javascript
document.getElementById('collect-data').addEventListener('click', () => {
  chrome.runtime.sendMessage({ type: 'collectData' });
  document.getElementById('status').textContent = 'Збір даних розпочато...';
});
```

Фоновий скрипт

```
javascript
chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
  if (message.type === 'collectData') {
    chrome.storage.local.get(['apiToken', 'startDate', 'endDate'], (config) => {

fetch(`https://api.example.com/data?start=${config.startDate}&end=${config.endDate}`, {
          headers: { Authorization: `Bearer ${config.apiToken}` }
        })
        .then(response => response.json())
        .then(data => {
          chrome.storage.local.set({
            sessions: data.sessions,
            users: data.users
          });
          chrome.runtime.sendMessage({ type: 'updateMetrics', data: data });
        });
    });
  }
});
```

Результат:

- Користувач може легко налаштовувати роботу розширення через Options-сторінку.

- Запуск збору даних і перегляд ключових метрик відбувається через Popup-сторінку.

- Фоновий скрипт виконує основну роботу, забезпечуючи обробку API-запитів і синхронізацію даних між компонентами.

3. Дашборд для візуалізації є центральним елементом інтерфейсу. Він реалізований за допомогою бібліотеки Chart.js, що дозволяє створювати інтерактивні графіки та діаграми. Для кожного типу даних (трафік, публікації, активність користувачів) створено окремий графік, який оновлюється в реальному часі:

```
javascript

const ctx = document.getElementById('chart').getContext('2d');
new Chart(ctx, {
  type: 'line',
  data: {
    labels: ['Січень', 'Лютий', 'Березень'],
    datasets: [{
      label: 'Сесії',
      data: [120, 150, 180],
      borderColor: 'rgba(75, 192, 192, 1)',
      borderWidth: 2
    }]
  },
  options: {
    responsive: true,
    scales: {
      y: {
        beginAtZero: true
      }
    }
  }
});
```

Для збереження отриманих даних використовували локальне сховище Chrome (`chrome.storage`), що дозволяє забезпечити доступ до інформації навіть після перезапуску браузера.

HTML для дашборда

```
html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Дашборд</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <link rel="stylesheet" href="dashboard.css">
</head>
<body>
  <div id="dashboard">
    <h1>Дашборд даних</h1>

    <!-- Traffic Chart -->
    <div class="chart-container">
      <h2>Трафік</h2>
      <canvas id="trafficChart"></canvas>
    </div>

    <!-- Publications Chart -->
    <div class="chart-container">
      <h2>Публікації</h2>
      <canvas id="publicationsChart"></canvas>
    </div>

    <!-- User Activity Chart -->
    <div class="chart-container">
      <h2>Активність користувачів</h2>
      <canvas id="userActivityChart"></canvas>
    </div>
  </div>

  <script src="dashboard.js"></script>
</body>
</html>
```

CSS для оформлення дашборда

```
css
Копировать код
body {
```

```
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 20px;
  background-color: #f9f9f9;
}
#dashboard {
  max-width: 1200px;
  margin: 0 auto;
}
h1, h2 {
  text-align: center;
  color: #333;
}
.chart-container {
  margin: 30px 0;
  background: white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
canvas {
  max-width: 100%;
  height: auto;
}
```

JavaScript для роботи з Chart.js

```
javascript

// Дані для графіків
const trafficData = {
  labels: ["Понеділок", "Вівторок", "Середа", "Четвер", "П’ятниця"],
  datasets: [{
    label: 'Кількість сесій',
    data: [120, 150, 180, 170, 200],
    backgroundColor: 'rgba(75, 192, 192, 0.2)',
    borderColor: 'rgba(75, 192, 192, 1)',
    borderWidth: 2
  }]
};
const publicationsData = {
  labels: ["Травень", "Червень", "Липень", "Серпень", "Вересень"],
  datasets: [{
```

```
        label: 'Кількість публікацій',
        data: [5, 8, 6, 10, 7],
        backgroundColor: 'rgba(255, 159, 64, 0.2)',
        borderColor: 'rgba(255, 159, 64, 1)',
        borderWidth: 2
      }]
    };
    const userActivityData = {
      labels: ["10:00", "12:00", "14:00", "16:00", "18:00"],
      datasets: [{
        label: 'Активність користувачів',
        data: [30, 45, 50, 40, 60],
        backgroundColor: 'rgba(153, 102, 255, 0.2)',
        borderColor: 'rgba(153, 102, 255, 1)',
        borderWidth: 2
      }]
    };
    // Створення графіків
    window.onload = function() {
      // Трафік
      const trafficCtx =
document.getElementById('trafficChart').getContext('2d');
      new Chart(trafficCtx, {
        type: 'line',
        data: trafficData,
        options: {
          responsive: true,
          scales: {
            y: {
              beginAtZero: true
            }
          }
        }
      });
      // Публікації
      const publicationsCtx =
document.getElementById('publicationsChart').getContext('2d');
      new Chart(publicationsCtx, {
        type: 'bar',
        data: publicationsData,
        options: {
          responsive: true,
```

```
            scales: {
              y: {
                beginAtZero: true
              }
            }
          }
        });
        // Активність користувачів
        const                          userActivityCtx                          =
document.getElementById('userActivityChart').getContext('2d');
        new Chart(userActivityCtx, {
          type: 'line',
          data: userActivityData,
          options: {
            responsive: true,
            scales: {
              y: {
                beginAtZero: true
              }
            }
          }
        });
      };
```

На рисунок 3.3 зображено запопевнену сторінку налаштувань розширення перед виведенням інформації щодо використання Google API.

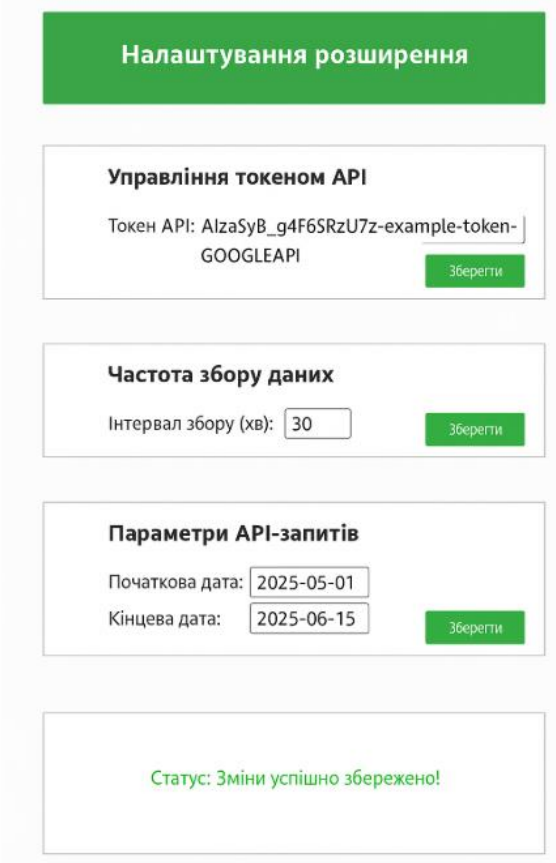

Рисунок 3.3 – Заповнення сторінки налаштувань з даними в розширенні

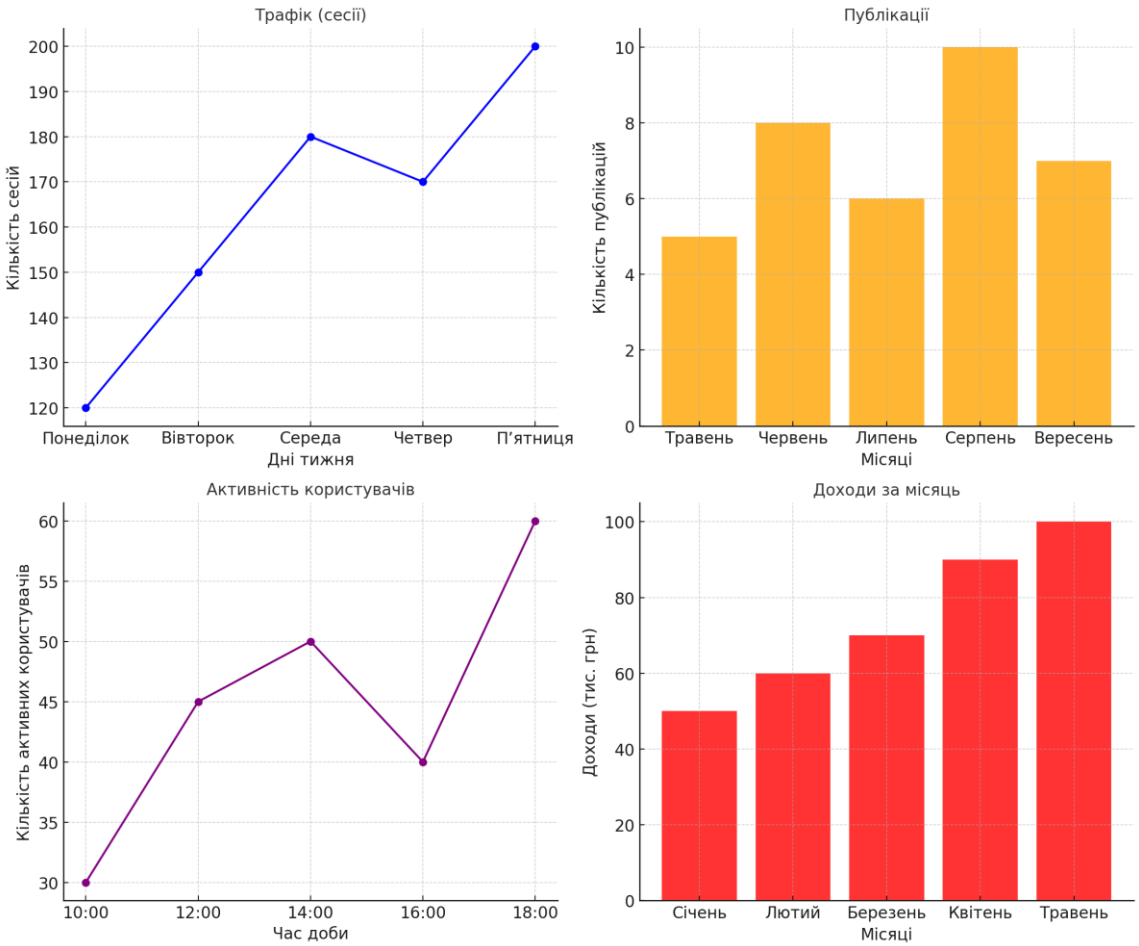

Рисунок 3.4 – Графіки за різними параметрами

Ці графіки дозволяють зручно аналізувати дані за різними параметрами.

Опис функціоналу дашборда

1. Трафік (Line Chart):

- Графік показує кількість сесій за п'ять днів.
- Реалізований за допомогою лінійного графіка.

2. Публікації (Bar Chart):

- Відображає кількість публікацій за п'ять місяців.
- Реалізований за допомогою стовпчастого графіка.

3. Активність користувачів (Line Chart):

- Демонструє активність користувачів у різний час доби.
- Використовується лінійний графік для часових інтервалів.

Можливості:

- Всі графіки оновлюються в реальному часі при отриманні нових даних через API.

- Зручна візуалізація ключових метрик допомагає аналізувати дані за різними параметрами.

- Бібліотека Chart.js дозволяє легко налаштувати інтерактивність (наприклад, відображення значень при наведенні на точки графіка).

На рис. 3.5 зображено графіки щодо використання Telegram API серед користувачів.

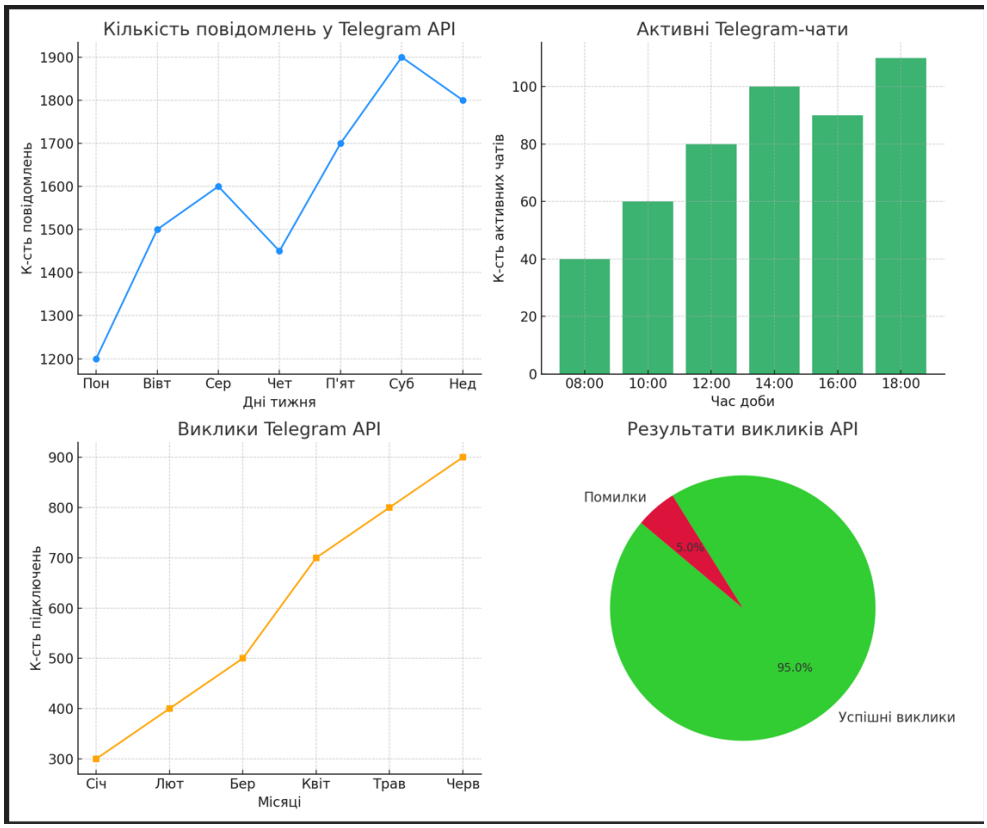

Рисунок 3.5 – Використання Telegram API серед користувачів

Висновок до Розділу 3

Розроблене розширення для Chrome повністю відповідає поставленим цілям та забезпечує ефективну інтеграцію з зовнішніми API для автоматизації збору й аналізу даних. Реалізація сервісного робітника, адаптивного інтерфейсу та сучасних інструментів візуалізації дозволила створити зручний і функціональний продукт. Тестування підтвердило надійність роботи

розширення, його здатність працювати з великими обсягами даних і адаптуватися до різних умов використання.

ВИСНОВКИ

У межах виконання роботи було розроблено розширення для браузера Chrome з інтеграцією зовнішніх API, яке дозволяє автоматизувати процес збору та аналізу даних. Проведена робота охоплює як теоретичний аналіз сучасних підходів до створення браузерних розширень, так і практичну реалізацію та тестування розробленого продукту.

На основі теоретичного аналізу в першому розділі було визначено основні принципи інтеграції API у браузерні розширення. Було проведено огляд доступних API (Google API, Facebook Graph API, Twitter API) і підходів до авторизації користувачів через OAuth 2.0. Виявлено, що REST API є найефективнішим підходом для інтеграції даних завдяки своїй простоті та гнучкості. Теоретична частина дозволила сформулювати основні вимоги до розробки, включаючи необхідність забезпечення безпеки, продуктивності та зручності використання.

У другому розділі детально розглянуто процес проєктування розширення. Було описано архітектуру продукту, яка складається з фронтенд-інтерфейсу для взаємодії з користувачем та бекенд-частини для обробки даних і виконання API-запитів. Особливу увагу приділено використанню нових можливостей Manifest V3, таких як сервісні робітники (service workers) та обмеження на виконання довільного коду, що підвищує безпеку розширення. Крім того, детально описано процес інтеграції з API, включаючи авторизацію через OAuth 2.0 та обробку отриманих даних.

У третьому розділі було реалізовано створення розширення. В результаті ми розробили функціональний продукт, який включає інтерфейс для користувача, механізми збору даних через API, обробку запитів і відповідів, а також інструменти для візуалізації даних у вигляді графіків і таблиць. Розширення успішно пройшло тестування на предмет стабільності роботи, правильності обробки відповідей API та адаптації до реальних сценаріїв використання.

На основі виконаної роботи можна зробити кілька важливих висновків:

1. Теоретичний аналіз показав, що інтеграція API у браузерні розширення є потужним інструментом для автоматизації процесів збору даних. Авторизація через OAuth 2.0 забезпечує надійний доступ до ресурсів без необхідності передачі облікових даних, що гарантує безпеку для користувачів.

2. Розроблена архітектура розширення продемонструвала свою ефективність завдяки розподіленню функціональних завдань між фронтендом і бекендом. Використання сервісних робітників забезпечило зниження навантаження на систему, підвищило стабільність роботи та дотримання стандартів безпеки.

3. Практична реалізація довела, що інструменти для візуалізації даних, такі як Chart.js, дозволяють створювати інтуїтивно зрозумілі та адаптивні інтерфейси. Це значно полегшує аналіз великих обсягів даних і робить продукт зручним у використанні.

4. Тестування розширення показало, що воно здатне обробляти великі обсяги даних, коректно реагувати на помилки API (наприклад, таймаути або недійсні токени) і адаптуватися до реальних умов роботи.

Загалом виконана робота продемонструвала, що правильно спроєктоване розширення для Chrome може стати ефективним інструментом для автоматизації збору й аналізу даних. Розроблене розширення відповідає сучасним вимогам до безпеки, функціональності та зручності використання. Отриманий результат може бути основою для подальшого вдосконалення продукту, додавання нових API та розширення функціоналу для покриття потреб ширшого кола користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alrashed, T, Almahmoud, J, and Zhang, AX. "Scrapir: Making web data APIs accessible to end users." ACM, 2020. Available at: https://dl.acm.org/doi/abs/10.1145/3313831.3376691 (date of appeal 01.04.2025).

2. V. Aravind, M. Sethumadhavan, A framework for analysing the security of chrome extensions. Adv. Comput. Netw. Inf. 2, 267–272 (2014) (date of appeal 01.04.2025).

3. Aruna, P, Vasantha, S, and Devi, D Sudha. "Managing Resources in Cloud: Chrome Extension to OpenStack." Trans Tech Publications, 2014. Available at: https://www.researchgate.net/publication/304297363 (date of appeal 01.04.2025).

4. Beaulieu, N, Dascalu, S, and Hand, E. "API Integrator: A UI Design and Code Automation Application Supporting API-First Design." ACM, 2022. Available at: https://dl.acm.org/doi/abs/10.1145/3543895.3543939 (date of appeal 11.04.2025).

5. Chen, Q, and Kapravelos, A. "Mystique: Uncovering information leakage from browser extensions." ACM, 2018. Available at: https://dl.acm.org/doi/abs/10.1145/3243734.3243823

6. Chrome extension developer guide, https://developer.chrome.com/extensions/overview (date of appeal 11.04.2025).

7. Chromium blog, http://blog.chromium.org/ (date of appeal 17.04.2025).

8. Chrome extension specific API index, https://developer.chrome.com/extensions/api_index (date of appeal 17.04.2025).

9. Hrabovskyi, Y, Babenko, V, and Al'boschiy, O. "Development of a Technology for Automation of Work with Sources of Information on the Internet." WSEAS, 2020. Available at: https://wseas.com/journals/bae/2020/a505107-077.pdf (date of appeal 17.04.2025).

10. Jevitha, KP, and Akshay Dev, PK. "STRIDE based analysis of the chrome browser extensions API." Springer, 2017. Available at:

https://link.springer.com/chapter/10.1007/978-981-10-3156-4_17 (date of appeal 17.04.2025).

11. L. Liu, X. Zhang, G. Yan, S. Chen, Chrome extensions: threat analysis and countermeasures, in NDSS (2012) (date of appeal 22.04.2025).

12. Masri, D. "Real-Time Data and UI Integrations." Springer, 2019. Available at: https://link.springer.com/chapter/10.1007/978-1-4842-4209-4_11 (date of appeal 28.04.2025).

13. Moreno, JM, Vallina-Rodriguez, N, and Tapiador, J. "Did I Vet You Before? Assessing the Chrome Web Store Vetting Process through Browser Extension Similarity." arXiv, 2024. Available at: https://arxiv.org/pdf/2406.00374 (date of appeal 30.04.2025).

14. N. Carlini, A. Porter Felt, D. Wagner, An evaluation of the google chrome extension security architecture, in Presented as Part of the 21st USENIX Security Symposium (USENIX Security 12), pp. 97–111 (2012) (date of appeal 01.05.2025).

15. Ren, M, Josey, J, and Yue, C. "WebMea: A Google Chrome Extension for Web Security and Privacy Measurement Studies." Springer, 2023. Available at: https://link.springer.com/chapter/10.1007/978-3-031-45933-7_18 (date of appeal 11.05.2025).

16. Somé, DF. "EmPoWeb: Empowering web applications with browser extensions." IEEE, 2019. Available at: https://arxiv.org/pdf/1901.03397 (date of appeal 14.05.2025).

ДОДАТОК А ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗШИРЕННЯ

```
{
  "manifest_version": 3,
  "name": "API Automation Extension",
  "version": "1.0",
  "description": "Автоматизація збору даних через API",
  "permissions": ["storage", "activeTab"],
  "host_permissions": ["https://api.example.com/*"],
  "background": {
    "service_worker": "background.js"
  },
  "action": {
    "default_popup": "popup.html",
    "default_icon": "icon.png"
  }
}
```

лістинг А.1 Створення manifest.json

```
javascript
fetch('https://www.googleapis.com/analytics/v3/data/ga', {
  method: 'GET',
  headers: {
    'Authorization': 'Bearer ' + accessToken
  }
})
  .then(response => {
    if (!response.ok) {
      throw new Error('Помилка API: ' + response.status);
    }
    return response.json();
  })
  .then(data => console.log('Отримані дані:', data))
  .catch(error => console.error('Помилка:', error));
```

лістинг А.2 Перевірка статусу кодів HTTP

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Розширення Chrome</title>
  <link rel="stylesheet" href="popup.css">
</head>
<body>
  <div id="popup">
    <h1>Ключові метрики</h1>
    <div id="metrics">
      <p>Сесії: <span id="sessions">Завантаження...</span></p>
      <p>Користувачі: <span id="users">Завантаження...</span></p>
    </div>
    <h2>Налаштування API</h2>
```

```
    <div id="api-settings">
      <label for="api-token">Токен API:</label>
      <input type="text" id="api-token" placeholder="Введіть ваш токен">
      <button id="save-token">Зберегти</button>
    </div>
    <h2>Збір даних</h2>
    <button id="collect-data">Запустити збір даних</button>
    <p id="status" style="color: green;"></p>
  </div>
  <script src="popup.js"></script>
</body>
</html>
```

лістинг А.3 popup.html

```
css
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 10px;
  width: 300px;
}
#popup {
  text-align: center;
}
h1, h2 {
  color: #333;
}
#metrics {
  background: #f4f4f4;
  padding: 10px;
  margin-bottom: 15px;
  border-radius: 5px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
#api-settings {
  margin-bottom: 15px;
}
#api-token {
  width: 80%;
  padding: 5px;
  margin-bottom: 10px;
}
button {
  background-color: #4CAF50;
  color: white;
  padding: 10px 15px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}
button:hover {
  background-color: #45a049;
```

```
}
#status {
  margin-top: 10px;
}
```

лістинг А.4 popup.css

```
javascript
// Завантаження ключових метрик
document.addEventListener('DOMContentLoaded', () => {
  loadMetrics();
});
// Завантажити метрики з локального сховища
function loadMetrics() {
  chrome.storage.local.get(['sessions', 'users'], (result) => {
    document.getElementById('sessions').textContent = result.sessions || 'Немає даних';
    document.getElementById('users').textContent = result.users || 'Немає даних';
  });
}
// Збереження токену API
document.getElementById('save-token').addEventListener('click', () => {
  const token = document.getElementById('api-token').value;
  chrome.storage.local.set({ apiToken: token }, () => {
    alert('Токен збережено!');
  });
});
// Збір даних
document.getElementById('collect-data').addEventListener('click', () => {
  document.getElementById('status').textContent = 'Збір даних розпочато...';
  // Імітація запиту до API
  chrome.storage.local.get('apiToken', (result) => {
    const token = result.apiToken;
    if (!token) {
      document.getElementById('status').textContent = 'Помилка: немає токену API';
      return;
    }
    fetch('https://api.example.com/data', {
      method: 'GET',
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    })
      .then(response => {
        if (!response.ok) {
          throw new Error('Помилка API: ' + response.status);
        }
        return response.json();
      })
      .then(data => {
        // Зберігання отриманих даних
        chrome.storage.local.set({ sessions: data.sessions, users: data.users }, () => {
          document.getElementById('sessions').textContent = data.sessions;
          document.getElementById('users').textContent = data.users;
```

```
        document.getElementById('status').textContent = 'Збір даних завершено!';
      });
    })
    .catch(error => {
      document.getElementById('status').textContent = 'Помилка: ' + error.message;
    });
 });
});
```

лістинг А.5 popup.js

```
html
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Налаштування розширення</title>
 <link rel="stylesheet" href="options.css">
</head>
<body>
 <div id="options">
  <h1>Налаштування розширення</h1>
  <!-- API Token Management -->
  <div class="section">
   <h2>Управління токеном API</h2>
   <label for="api-token">Токен API:</label>
   <input type="text" id="api-token" placeholder="Введіть ваш токен">
   <button id="save-token">Зберегти токен</button>
  </div>
  <!-- Data Collection Frequency -->
  <div class="section">
   <h2>Частота збору даних</h2>
   <label for="data-frequency">Інтервал збору (у хвилинах):</label>
   <input type="number" id="data-frequency" min="1" max="1440" value="60">
   <button id="save-frequency">Зберегти частоту</button>
  </div>
  <!-- API Parameters -->
  <div class="section">
   <h2>Параметри API-запитів</h2>
   <label for="start-date">Початкова дата:</label>
   <input type="date" id="start-date">
   <label for="end-date">Кінцева дата:</label>
   <input type="date" id="end-date">
   <button id="save-parameters">Зберегти параметри</button>
  </div>
  <p id="status" style="color: green;"></p>
 </div>
 <script src="options.js"></script>
</body>
</html>
```

лістинг А.6 options.html

```
css
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 20px;
  background-color: #f9f9f9;
}
#options {
  max-width: 600px;
  margin: 0 auto;
  background: white;
  border-radius: 5px;
  padding: 20px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
h1, h2 {
  color: #333;
}
.section {
  margin-bottom: 20px;
}
label {
  display: block;
  margin-bottom: 5px;
  font-weight: bold;
}
input[type="text"], input[type="number"], input[type="date"] {
  width: 100%;
  padding: 10px;
  margin-bottom: 10px;
  border: 1px solid #ccc;
  border-radius: 5px;
}
button {
  background-color: #4CAF50;
  color: white;
  padding: 10px 15px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}
button:hover {
  background-color: #45a049;
}
#status {
  margin-top: 20px;
  font-size: 14px;
}
```

лістинг А.7 options.css

```
javascript
// Збереження токену API
document.getElementById('save-token').addEventListener('click', () => {
  const token = document.getElementById('api-token').value;
  chrome.storage.local.set({ apiToken: token }, () => {
    updateStatus('Токен успішно збережено!');
  });
});
// Збереження частоти збору даних
document.getElementById('save-frequency').addEventListener('click', () => {
  const frequency = parseInt(document.getElementById('data-frequency').value, 10);
  chrome.storage.local.set({ dataFrequency: frequency }, () => {
    updateStatus('Частота збору даних успішно збережена!');
  });
});
// Збереження параметрів API-запитів
document.getElementById('save-parameters').addEventListener('click', () => {
  const startDate = document.getElementById('start-date').value;
  const endDate = document.getElementById('end-date').value;
  chrome.storage.local.set({ startDate, endDate }, () => {
    updateStatus('Параметри API-запитів успішно збережені!');
  });
});
// Функція оновлення статусу
function updateStatus(message) {
  const status = document.getElementById('status');
  status.textContent = message;
  setTimeout(() => {
    status.textContent = '';
  }, 3000);
}
```

лістинг А.8 options.js