МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Аналіз методів та алгоритмів скорочення URL посилань

Виконав:
Студент 2 курсу групи ІПЗм-21-1
Шамрай Є. О.
(прізвище, ініціали)

Спеціальність 121 — Інженерія програмного забезпечення

Тип програми Освітньо-наукова

Керівник доц., к.т.н. Кириченко І. В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. Кафедри _____ З. В. Дудар

2023

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інжереній

Рівень вищої освіти другий (магістерський)

Спеціальність 121 — Інженерія програмного забезпечення

Тип програми освітньо-наукова програма

Спеціалізація Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. Кафедри _____

«_____» _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Шамраю Єгору Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Аналіз методів та алгоритмів скорочення URL посилань затверджена наказом університету від « 29 » березня 2023 р. № 302 Ст

2. Термін подання студентом роботи до екзаменаційної комісії « 15 » травня 2023 р.

3. Вихідні дані до роботи *Алгоритми скорочення URL-адрес, оцінка продуктивності, швидкодія скрочення, порівняльний аналіз, алгоритми на основі хешування, оцінка продуктивності, алгоритми кодування, атаки грубої сили.*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз предметної галузі і постановка задачі, дослідження різних алгоритмів та методів скорочення посилань, вплив на швидкодію обробки даних, отримання та аналіз результатів, їх подальше використання.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	26.01.2023	виконано
2	Огляд існуючих методів, алгоритмів та інструментів для скорочення URL посилань	30.02.2023	виконано
3	Постановка задачі	14.02.2023	виконано
4	Вибір методів та критеріїв оцінки швидкої скорочення URL посилань	16.03.2023	виконано
5	Дослідження швидкодії алгоритмів для скорочення URL-адреси в залежності від об’єму даних	26.03.2023	виконано
6	Підготовка пояснювальної записки	18.04.2023	виконано
7	Перевірка роботи на антиплагіат	09.05.2023	виконано
8	Нормоконтроль	10.05.2023	виконано
9	Рецензування	11.05.2023	виконано
10	Підготовка презентації та доповіді	12.05.2023	виконано
11	Попередній захист	13.05.2023	виконано
12	Занесення роботи в електронний архів	14.05.2023	виконано
13	Допуск до захисту у зав. кафедри	15.05.2023	виконано

Дата видачі завдання 24 січня 2023 р.

Студента _____
(підпис)

Керівник роботи _____ доцент, к.т.н. Кириченко І. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Звіт до кваліфікаційної роботи містить: 88 с., 11 рис., 8 табл., 24 джерел, 7 додатків.

АЛГОРИТМИ, АДРЕСА РЕСУРСУ, ХЕШУВАННЯ, СКОРОЧЕННЯ, ПОСИЛАННЯ

Об’єктом дослідження є аналіз методів та алгоритмів для скорочення уніфікованого локатора ресурсів.

Метою роботи є дослідження чинних методів та алгоритмів, бібліотек, сервісів що дозволяють скоротити URL-адресу і спрямувати на потрібну сторінку, задля економії довжини повідомлення та ненавмисному спотворенню посилань.

В результаті роботи було здійснено аналіз швидкодії і колізій методів та алгоритмів скорочення посилань, огляд популярних алгоритмів та методів скорочення посилань та експериментальне порівняння швидкодії обробки даних за допомогою цих методів і алгоритмів.

ALGORITHMS, RESOURCE ADDRESS, HASHING, SHORTENING, LINKS

The object of research is to analyze and evaluate algorithms for reducing the unified resource locator.

The purpose of the work is to study existing algorithms, libraries, services that allow you to shorten the URL and direct to the desired page to save message length and unintentional distortion of links.

As a result of the work, an analysis of the performance and collisions of methods and algorithms for link shortening was carried out, a review of popular algorithms and methods for link shortening was conducted, and experimental comparison of data processing performance using these methods and algorithms was performed.

Я, Шамрай Єгор Олегович, студент гр. ІПЗм-21-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Аналіз методів та алгоритмів скорочення URL посилань», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлена з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

У сучасній цифровій епохи URL-адреси стали невід'ємною складовою нашого повсякденного життя. Вони використовуються для доступу до різноманітних ресурсів в Інтернеті, від веб-сайтів до додатків. Однак, довжина URL-адрес може стати перешкодою, особливо коли йдеться про їх поширення на платформах, які мають обмеження на кількість символів, таких як Twitter. Саме тут використання скорочення URL-адрес стає доцільним. Скорочення URL-адрес є процесом створення короткої, зручної версії довгого URL-адреси, яка перенаправляє на оригінальну адресу. Цей процес став все більш популярним протягом років, і наразі на Інтернеті доступні численні сервіси скорочення URL-адрес. Однак, з поширенням кіберзагроз, безпека скорочення URL-адрес стала серйозною проблемою. Для вирішення цієї проблеми дослідники розробили різні алгоритми [2] скорочення URL, які пріоритетні та ставлять якість безпеки і продуктивності. Ці алгоритми мають на меті створення коротких URL-адрес, які стійкі до кібератак, забезпечуючи швидку та ефективну переадресацію на оригінальний URL.

URL скорочення – це процес перетворення довгого URL [3] у більш короткий, який перенаправляє на оригінальний URL. Скорочений URL використовується для поширення посилань, збереження символів у дописах в соціальних мережах або зручності запам'ятовування URL користувачами. Сервіси скорочення URL стали популярними завдяки своїй зручності, але вони також мають деякі недоліки, такі як проблеми з безпекою та продуктивністю. У цій кваліфікаційній роботі ми розглянемо та порівняємо різні алгоритми скорочення URL з урахуванням їх продуктивності та безпеки.

Основною метою кваліфікаційної роботи є дослідження та порівняння найбільш оптимізованого алгоритму та методу для скорочення URL посилань в залежності від вхідних даних з точки зору швидкодії.

Дана тема є досить актуальною, оскільки інтернет-ресурси постійно зростають в обсязі та кількості, а короткі URL-адреси можуть допомогти зекономити простір та полегшити доступ до потрібних даних. Дослідження в галузі

методів та алгоритмів скорочення URL може допомогти покращити ефективність цих процесів та забезпечити безпеку відкриття скорочених посилань. Крім того, скорочені URL-адреси є корисним інструментом для маркетингових кампаній та реклами, оскільки вони дозволяють створювати зручні та запам'ятовуванні посилання на різні продукти та послуги. Проте, важливо мати на увазі, що зловмисники можуть використовувати скорочені URL-адреси для злочинних дій, таких як шахрайство та фішинг.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– зробити аналіз та порівняння існуючих методів та алгоритмів скорочення URL посилань;

– визначити метрики, за допомогою яких буде проведено оцінювання;

– розробити серверний застосунок для проведення експерименту;

– виміряти та підрахувати значення обраних метрик для кожного алгоритму для скорочення посилання;

– надати рекомендації щодо використання кожного з методів.

Об’єктом дослідження є швидкодія методів та колізія алгоритмів, які розроблені за допомогою бібліотек та надають таку функціональність як скорочення посилань.

Предметом дослідження є методи та алгоритми для скорочення URL посилань.

Для проведення дослідження використовуються методи вимірювання розміру вихідного коду та обчислення метрик, що базуються на цих даних. Також у якості методу дослідження використано бібліотеку BenchmarkDotNet для вимірювання швидкодії.

Отримані в ході дослідження результати можуть використовуватися для прийняття рішень відносно використання алгоритмів кодування, хешування або стиснення даних в комплексних застосунках, які містять функціонал скорочення URL-адрес.

Результати даної кваліфікаційної роботи було опубліковано в журналі «Біоніка інтелекту». який включено до Переліку наукових фахових видань

України, категорія «Б», технічні науки (затверджено наказом МОНУ від 02.07.2020 № 886):

– Comparative analysis of url shortening algorithms: performance considerations in the application domain of web search and information retrieval technology / I. Kyrychenko, Y. Shamrai// Bionics of Intelligence. – Kharkiv: Kharkiv National University of Radio Eltctronics. – 2022. – № 2 (99). – С. 15–22.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналітичний огляд

Скорочення URL-адреси – це підхід у всесвітній мережі Інтернет, за допомогою якого уніфікований локатор ресурсів (URL-адреса) може бути значно меншим, який повинна вказувати на веб-сторінку. Це досягається за допомогою перенаправлення, яке посилається на веб-сторінку, що має довгу URL-адресу [1]. Часто доменне ім'я перенаправлення коротше, ніж оригінальне. Дружня URL-адреса може бути бажаною для технологій обміну повідомленнями, які обмежують кількість символів у повідомленні (наприклад, SMS), для зменшення кількості необхідного набору тексту, якщо читач копіює URL-адресу з друкованого джерела, для створення постійного посилання чи зручного читання посилання людиною.

Інші способи використання скорочення URL-адрес – це зробити більш привабливий вид для посилання, відстежувати статистику переходу на посилання або приховати основну адресу. Це пов'язано з тим, що скорочення URL-адреси може перенаправляти практично на будь-який веб-домен, навіть на зловмисний. Таким чином, хоча приховання основної адреси може бути бажаним з законних ділових та особистих причин, воно відкрите для зловживань.

Сервіси для скорочень URL-адрес дозволяють шахраям використовувати скорочені URL-адреси для розповсюдження небезпечних посилань, оскільки кінцевий користувач не зможе одразу відтворити повну адресу. В результаті, перехід за такими посиланнями може призвести до різних ризиків, таких як шкідливе програмне забезпечення та фішинг.

Однак безпека браузерів пройшла довгий шлях, тому сучасні безпечні браузери повинні виявляти будь-які підозрілі посилання.

Інструмент скорочення посилання скоротить будь-яку URL-адресу, може зробити її більш керованою, і її можна адаптувати до того, що найкраще підходить для веб-сайту або до того, з чим пов'язується. Через перенаселеність веб-сайтів в Інтернеті майже неможливо отримати будь-яку коротку URL-адресу для абсолютно нового веб-сайту, а також практично неможливо запам'ятати

оригінальну URL-адресу, оскільки вона, як правило, являє собою випадкову комбінацію букв і цифр.

Скорочення веб-посилань є дуже корисним інструментом оскільки має у використанні наступні переваги:

– зручність;
– відстеження посилання;
– ефективність рейтингу кліків;
– надійність;
– впізнаваність бренду.

Розглянемо окремо кожну характеристику більш детально.

Короткі посилання є більш зрозумілими, простими та легкими для запам'ятовування. Замість того, щоб стикатися з довгою URL-адресою, краще перейти за коротким посиланням, яким легко ділитися і запам'ятовувати.

Посилання, яке відстежується, допомагає вам зрозуміти, скільки кліків ви отримали і скільки ще можете отримати. Це допомагає вам зрозуміти, чи є ваше посилання, яким ви ділитеся, зручним для переходів чи ні.

Якщо ви отримуєте менший коефіцієнт кліків (CTR), то ви можете перейти до етапу, де ви отримуєте найвищий CTR. Ось чому відстеження посилання набагато важливіше, ніж обмін випадковим посиланням.

Більшість людей переходять за надійним посиланням, яке має чітку концепцію. Це більш перспективно, ніж довге незрозуміле посилання.

Сервіси скорочення посилань дозволяють створювати впізнавані брендовані посилання за допомогою налаштування унікального домену, який може бути використаний в маркетингових кампаніях та рекламних матеріалах. Це допоможе підвищити та підтримувати репутацію бренду. Можна поширювати посилання на свій бренд, щоб його помітила цільова аудиторія продукту.

Сервіси скорочення посилань є надійними, оскільки використовують перевірені часом математичні хеш-функції, які дозволяють перетворити вхідний масив даних довільної довжини у вихідний бітовий рядок фіксованої довжини.

Принцип роботи сервісів дуже простий та зрозумілий не технічному користувачу. Нижче наведена схема роботи сервісу скорочення URL-посилань (див. рис. 1).

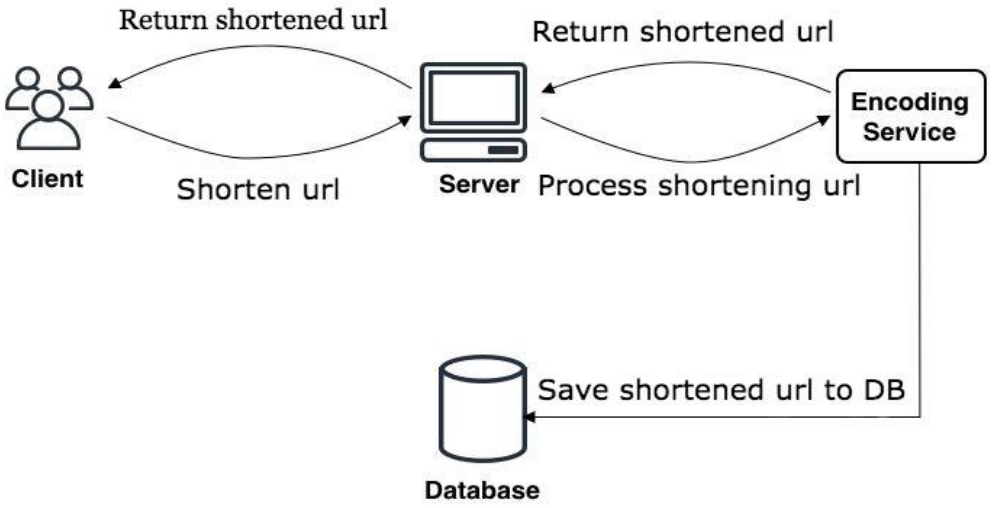

Рисунок 1 – Базова архітектура додатку для роботи скорочення адресних посилань

Базова архітектура додатку для роботи скорочення адресних посилань полягає у використанні серверної та клієнтської частин програми. Серверна частина забезпечує обробку запитів від клієнтів, генерацію скорочених посилань та зберігання їх у базі даних. Клієнтська частина програми дозволяє користувачам генерувати скорочені посилання та використовувати їх для доступу до відповідного вмісту. Додаток для скорочення адресних посилань може використовувати БД для збереження коротких URL-адрес та відповідних їм довгих URL-адрес. Кожен короткий URL-адрес може бути збережений в БД разом зі своїм відповідним довгим URL-адресом та іншими додатковими даними, такими як дата створення, кількість переходів та ін. БД може бути реляційною або нереляційною, залежно від конкретної реалізації додатку та вимог до збереження даних.

1.2 Проблематика алгоритмів скорочення посилань

URL-скорочення – це техніка, яка за останні роки стала дедалі популярнішою, оскільки вона дозволяє користувачам ділитися довгими, складними URL-адресами в компактному і легко читабельному форматі. Однак, практика

скорочення URL-адресів викликає також низку питань та викликів, які потрібно вирішити, щоб забезпечити користувачам можливість використовувати скорочені URL-адреси безпечно та ефективно. Алгоритми скорочення URL використовуються для перетворення довгих URL-адрес в коротші. Це зроблено для того, щоб спростити можливість ділитися посиланнями для соціальних медіа та інших місцях, де існують обмеження на кількість символів. Хоча алгоритми скорочення URL-адресів стали досить популярними, вони також мають деякі проблеми, які потрібно вирішити. Виділяють основні 5 проблем, а саме:

- проблеми безпеки;
- недійсне посилання;
- залежність від сторонніх сервісів;
- колізія посилань;
- залежність від вхідних даних;

Зазвичай існуючі сервіси дозволяють позбутися цих проблем, проте виникають певні ускладнення відносно способів зберігання вже скорочених посилань, а також підтримки постійності даних в разі горизонтального масштабування архітектури додатку, що містить функціональність скорочення даних. Звичайно, можна спиратися на вибір оптимального алгоритму для швидкого перетворення великої частини інформації у більш компактну, проте треба брати до уваги як зазначено вище фізичні способи зберігання вже перетворених посилань, перевірку на шкідливе ПО при взаємодії з посиланням і інші проблеми. Певні обмеження на роботу алгоритмів скорочення також накладають самі вхідні дані.

Існують бібліотеки, що дозволяють реалізувати алгоритми для скорочення посилань. Проте не всі бібліотеки можуть використовуватися у всіх програмах, оскільки є певні відмінності у семантиці мові на якій реалізовані самі алгоритми, а також способі їх інтерпретації апаратною частиною. Платформа .NET містить вже реалізацію певних алгоритмів на базі яких можна будувати вже комплексні сервісі для скорочення посилань. Крім того замість вбудованих алгоритмів існують авторські реалізації алгоритмів, що також можуть використовуватися у розробці.

Проблема скорочення посилань полягає в тому, що деякі посилання мають дуже довгі URL-адреси, що ускладнює їх використання, особливо в соціальних мережах та на мобільних пристроях. Щоб зробити посилання більш доступними для користувачів, використовуються алгоритми скорочення, які замінюють довгий URL-адрес на короткий.

Однак, існує кілька проблем з цим підходом. Перш за все, короткий URL-адрес може бути складнішим для розуміння, оскільки не містить описової інформації. Крім того, короткі URL-адреси можуть бути складні для запам'ятовування, особливо якщо їх багато. Не менш важливо, що короткі посилання можуть бути використані для шахрайства, фішингу та інших зловмисних дій.

Одним з можливих рішень є використання URL-адрес з кількома словами, що відображають зміст посилання. Це зробить їх більш зрозумілими. Також важливо підтримувати стандарти безпеки при скороченні посилань та використовувати надійні сервіси скорочення, які запобігають зловмисним діям. Для забезпечення надійності та доступності сервісу скорочення посилань, можна використовувати розподілену архітектуру з горизонтальним масштабуванням. Такий підхід дозволяє забезпечити високу продуктивність та швидкість роботи сервісу, а також забезпечує його стійкість до відмов.

1.3 Постановка задачі

В Інтернеті деякі сайти використовують параметри запиту в URL-адресах для передачі інформації між сервером та клієнтом. Це може призвести до появи довгих URL-адрес, особливо якщо параметри запиту містять багато інформації або веб-сайти що генерують сторінки динамічно на основі інформації, переданої користувачем або іншими системами, що також можуть призвести до появи довгих посилань.

Правильно підібраний алгоритм може пришвидшити та уникнути проблем відносно колізій. Це робить дослідження методів управління станом з точки зору швидкодії справді важливим та актуальним.

Метою цього дослідження є порівняння певних алгоритмів хешування [4], алгоритми рандомізації та алгоритми текстового кодування для скорочення посилань та надання певних рекомендацій щодо їх використання.

Враховуючи усе вищеперераховане, в рамках дослідження потрібно вирішити наступні завдання:

– розглянути існуючі сервіси для скорочення [5] url посилань та алгоритми, які вони використовують;

– проаналізувати та обрати алгоритми для скорочення посилань;

– проаналізувати існуючі підходи до вимірювання швидкодії скорочення посилань;

– визначити метрики, які будуть використані для проведення експерименту та подальшого оцінювання;

– визначити функціональні вимоги для веб серверного застосунку, який буде використано для проведення експерименту, та розробити його;

– виміряти та підрахувати значення обраних метрик для кожного з алгоритмів;

– надати рекомендації щодо використання кожного з методів.

2 ВИБІР МЕТОДІВ ТА АЛГОРИТМІВ ДЛЯ СКОРОЧЕННЯ URL ПОСИЛАНЬ

2.1 Аналіз існуючих сервісів для скорочення посилань

У сучасному цифровому світі вже є певна кількість сервісів та бібліотек, написаних на різних мовах програмування, які вирішують задачу скорочення уніфікованого локатора ресурсів. Більшість з API [6] використовують один і той самий принцип – хешування вхідного масиву даних. В основному, в якості вхідних даних у скороченні URL-посилання бере участь звичайна послідовність байтів, тобто послідовність символів алфавіту. Крім скорочення посилань, сучасні сервіси надають можливість тимчасового існування скороченої прихованої адреси. Це дає можливість користувачам створювати скорочені адреси на певний проміжок часу для своїх потреб. Розглянемо TinyURL сервіс, що дозволяє швидко створювати скорочені URL-посилання (див. рис. 2).

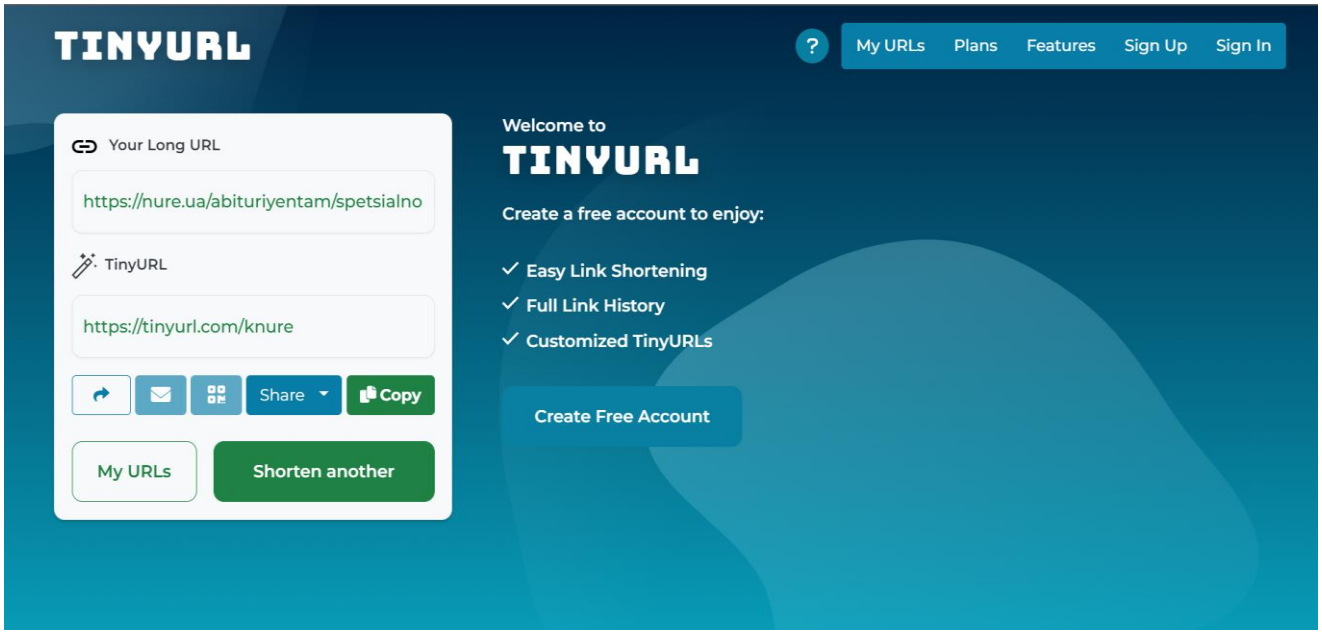

Рисунок 2 – TinyURL сервіс для скорочення URL-адрес

Сервіс TinyURL надає користувачу наступні переваги:

– скорочення URL-адрес;
– налаштування URL-адрес;
– відстеження кліків;

– налаштування брендингу посилання;
– звітність та статистику;
– панель інформаційної та моніторингу активності.

TinyURL – це інструмент для керування посиланнями, розроблений, щоб допомогти користувачам скорочувати та налаштовувати URL-адреси. Він дозволяє користувачам створювати брендовані скорочені посилання, термін дії яких ніколи не закінчується. Компанії можуть використовувати TinyURL [7] для покращення цифрових кампаній, впроваджуючи посилання, які клієнти легко впізнають і яким можуть довіряти. Інструмент використовує технологію моніторингу посилань, щоб надавати необмежену кількість даних про кліки за скороченими URL-адресами. Використовуючи аналітичну панель TinyURL, користувачі можуть отримати доступ до детальних показників кліків для URL-адрес, створених за допомогою інструменту. Дані про кліки в режимі реального часу розбиваються за регіонами та типами пристроїв, щоб допомогти користувачам краще зрозуміти свою аудиторію. Інформаційна панель надає дані про ефективність як для окремих посилань, так і для груп посилань. Коли скорочувач URL-адрес отримує нову URL-адресу, він повинен створити нову коротку URL-адресу, яка ще не зайнята, і повернути її. Потім він зберігає коротку і довгу URL-адреси у сховищі ключів-значень і використовує їх під час пошуку.

Проблема полягає в тому, як придумати цю коротку URL-адресу дуже швидко. Один з підходів – просто збільшити лічильник, але це створить вузьке місце, оскільки можна отримати більше паралельних запитів, тому альтернативою є просто генерувати випадковий рядок і сподіватися, що він ще не зайнятий. Щоб уникнути колізій (генерування рядка, який вже було згенеровано), ви можете просто збільшити довжину рядка (обійти проблему скорочення URL-адреси) або перевірити, що URL-адреса є дійсно унікальною. Оскільки URL-адреси генеруються частіше, ніж їх шукають (більшість з них ніколи не натискають), ви можете використовувати підхід на кшталт фільтрів розпускання для перевірки

існування заданого ключа, без необхідності отримувати значення ключа. Наступним розглянемо сервіс Short.io (див. рис 3).

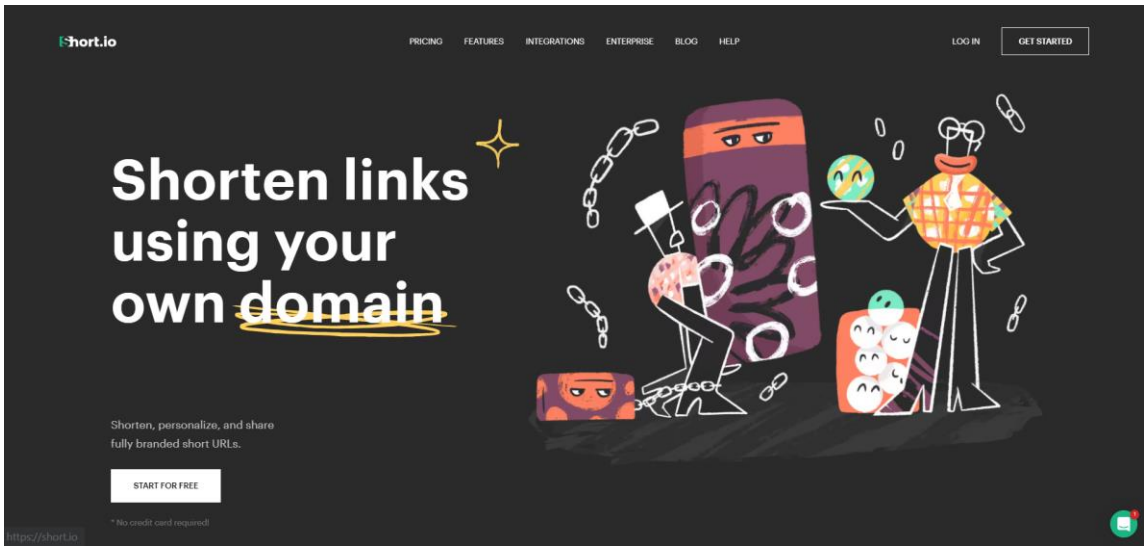

Рисунок 3 – Short.io сервіс для скорочення URL-адрес

Так як і попередній сервіс, Short.io надає користувачам змогу скорочувати URL-посилання. Сервіс пов'язує внутрішні операції з інтеграціями, щоб автоматизувати ручні процеси. Ключові функції включають відстеження кліків у режимі реального часу, кілька власних доменів, команди з ролями, відстеження кампаній, API для розробників, детальну статистику, глибокі посилання. Short.io інтегрується з Zapier, Google Analytics, GTM, Facebook Pixel, Segment та AdRoll. Інтеграції дозволяють правильно оцінювати успішність посилань та автоматизувати роботу.

Сервіс Short.io надає багато особливостей щодо зберігання та доступу до скороченого посилання, серед них:

– мобільні посилання;
– закінчення терміну дії посилання;
– геотаргетинг;
– генерація QR-коду;

Розглянемо коротко кожну з особливостей окремо.

Мобільні посилання підтримують різновид глибоких посилань, який перенаправляє користувача до вже встановлених додатків. Цей тип глибокого посилання не вимагає модифікації мобільного додатку, як звичайне глибоке посилання, яке ми знаємо.

Функція «Тимчасова URL-адреса» [8] перенаправляє користувачів на нову URL-адресу призначення після закінчення терміну дії старої. Це відбувається без зміни короткого посилання. Користувач встановлює довге посилання, час або кількість кліків, коли URL-адреса повинна оновлюватися. Коли посилання втрачає свою актуальність, користувачі автоматично перенаправляються на нову URL-адресу призначення.

Більшість веб-сайтів використовують інформацію, яка є важливою лише для конкретних користувачів.

Іноді клієнти можуть стикатись з незручностями, коли вони переходять за скороченим посиланням і бачать дані, які не стосуються їхнього регіону або країни. Це може бути викликано застосуванням геотаргетингу, який перенаправляє користувачів на відповідні веб-сторінки залежно від їхнього місцезнаходження. Однак, щоб уникнути плутанини, краще масово вибирати країни та регіони, а не додавати їх поодинці.

Short.io дозволяє налаштовувати QR-коди для скорочених адрес. Користувач може змінювати колір та додавати логотипи в сам код. Незважаючи на те, що короткі посилання легко набирати в адресному рядку браузера смартфона, спосіб передачі URL-адреси за допомогою QR-коду зручніший.

2.1.1 Огляд алгоритму кодування Base64 для скорочення посилань

Кодування Base використовуються для зберігання або передачі даних у середовищах, які обмежені US-ASCII. Існують різні стандарти відносно кодування даних, такі як Base32, Base62 та Bas64 [10], проте у досліджені буде розглянуто саме Base64, так як цей вид кодування є одним з найпоширеніших. Base кодування також може використовуватися у нових додатках, які не мають обмежень, просто тому, що це дозволяє маніпулювати об'єктами з текстовими редакторами.

Base-кодування використовує конкретний, обмежений алфавіт для кодування бінарних даних. В закодованих даних можуть бути символи, які не належать алфавіту, що може бути наслідком пошкодження даних або задуманою функцією. Такі символи можуть бути використані як прихований канал, де можна передавати не-протокольні дані зі злочинними метою. Також можуть бути відправлені символи, які не належать алфавіту, щоб використовувати помилки в реалізації, що можуть призвести до атак на переповнення буфера, тощо.

Base64 – це алгоритм, який використовує концепцію сучасних алгоритмів шифрування. Це блоковий шифр-алгоритм, який оперує бітами, але режим Base64 простіший у реалізації, ніж інші.

Base64 – це загальний термін для деяких схожих схем кодування, які кодують двійкові дані і переводять їх у представлення системи числення з основою 64. Термін походить від Base64 MIME, що кодує специфічний вміст.

Схема кодування Base64 зазвичай використовується, коли є потреба кодувати двійкові дані, які потрібно зберігати і передавати через носії, призначені для роботи з текстовими даними. Це робиться для того, щоб гарантувати, що дані залишаються недоторканими і не будуть змінені під час транспортування. Base64 широко використовується в різних додатках, включаючи електронну пошту через MIME і зберігання складних даних в XML. Base64 потрібно вивчити, оскільки перетворення Base64 широко використовується в Інтернеті як засіб передачі даних, форматування даних. За рахунок того, що результатом перетворення base64 буде звичайний текст, то це значення буде набагато зручніше пересилати, порівняно з формою двійкового формату даних. Base64 використовується в багатьох сферах. Наприклад, деякі перетворення використовують у таких сферах, як PEM base64, MIME, UTF-7 [11] та OpenPGP.

Раніше в різних додатках використовувалися різні типи кодування Base, але тепер протоколи часто використовують Base кодування загалом, а зокрема Base64 без точного опису або посилання на стандарт. MIME часто використовується як посилання для base64 без урахування наслідків для розбивки на рядки або неприпустимих символів. Багатоцільове розширення інтернет-пошти, що

використовує «base64» як одну з двох схем кодування двійковий-текст (інша – «кодування Base64 з можливістю цитування та друку»). MIME базується на RFC 1421 версії PEM. Використовує 64-символьний алфавіт, той самий механізм кодування, що і PEM, і використовує символ «=» для виведення пробілів так само, як описано в RFC 1521. MIME не визначає фіксовану довжину фіксованої довжини для каналів, закодованих Base64, але не визначив максимальну довжину у 76 символів. Він також передбачає, що будь-які алфавітні символи повинні ігноруватися сумісним декодером, хоча більшість реалізацій використовують пару CR/LF для обмеження кодованих рядків. Таким чином, фактична довжина MIME-сумісних двійкових даних, закодованих у Base64, зазвичай становить близько 137% від довжини вихідних даних, хоча для дуже коротких повідомлень накладні витрати можуть бути набагато вищими через накладні витрати на заголовок. Остаточний розмір двійкових даних, закодованих у Base64, дорівнює 1,37 розміру вихідних даних + 814 байт. Іншими словами, можна оцінити розмір кодованих даних за цією формулою:

$$b = \frac{L(E)) - 814}{1.37}, \quad (1)$$

де b – розмірність байт;

E – закодована строка;

L – довжина закодованої строки.

Перетворення Base64 – це один з алгоритмів кодування і декодування даних у формат ASCI в основі якого лежить число 64. Символи, згенеровані з Base64, складаються з «A..Z», «a..z» і «0..9», а останні два символи – це «/» і «+».

Техніка кодування Base64 проста. Розглянемо по пунктам основні кроки алгоритму:

– пошук ASCII код кожного тексту;

– пошук двійкового числа, 8 біт ASCII коду якого існують;

– об'єднати останні 8 біт до 24 біт;

– розбити попередні 24 біти до 6 біт;

– кожен фрагмент перетворюється в десяткове значення;

– додати значення – десяткове значення до індексу, щоб вибрати символ, що входить до складу base64, і максимально 63 або 64 до індексу.

Рисунок 4 відображає частину таблиці ASCII [12], яка в свою чергу відображається у нову 64-символьну таблицю.

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Рисунок 4 – Таблиця Bas64 кодування

Алгоритм Base64 може комбінуватися з різними іншими видами алгоритмів такими як MD5 [13] або SHA-1, SHA-2 в залежності від поставлених цілей. Крім того можна використати інші варіанти Base, які будуть використовувати інший алфавіт кодування. Для створення унікальної короткої URL-адреси з існуючої URL-адреси ми можемо використовувати методи хешування. Хеш, який потрібно закодувати, може бути в Base36, Base62 або Base64. Ми розглядаємо кодування Base64, оскільки воно містить всі символи, які можуть бути включені в URL-адресу, тоді якої довжини має бути відповідна довжина скороченої URL-адреси.

Тобто, для посилання довжиною 8 буде можливість генерувати 280 трильйонів посилань, оскільки 64^8 , що є дуже великим показником відносно унікальності створених посилань, проте це містить накладні витрати відносно зберігання цих скорочених посилань. Це не безпечний метод скорочення посилань. Посилання, закодовані Base64, можна легко розшифрувати, тому їх не слід використовувати для захисту конфіденційної інформації, тому бажано комбінувати цей алгоритм з хешуванням.

2.1.2 Огляд алгоритму стиснення Deflate для скорочення посилань

Алгоритм Deflate – це алгоритм для стиснення даних без втрат, який використовує комбінацію алгоритмів Хаффмана і LZ77. Він працює у два етапи: стискання та розпакування. У процесі стиснення дані розбиваються на блоки, які потім стискаються за допомогою алгоритму Deflate. Розглянемо основні компоненти алгоритми.

Алгоритм, описаний Девідом Хаффманом, ставить кожен символ у відповідність листку дерева двійкового коду. Ці вузли зважуються за кількістю входжень відповідного символу, що називається частотою або вартістю.

Структура дерева утворюється в результаті покрокового об'єднання вузлів, поки всі вони не будуть вбудовані в кореневе дерево. Алгоритм завжди об'єднує два вузли з найнижчою частотою у висхідному порядку. Новий внутрішній вузол отримує суму частот обох дочірніх вершин.

Розглянемо роботу префіксного дерева Хаффмана на прикладі телефонних номерів. Починаючи з гудка, ви натискаєте послідовність з п'яти, семи, восьми, одинадцяти, дванадцяти або іншої кількості клавіш – і кожна послідовність клавіш потрапляє на іншу конкретну телефонну лінію. Тепер уявімо, що ви перебуваєте в офісі з внутрішнім комутатором, як у багатьох великих компаніях. На всіх інших телефонах у банку потрібно набирати лише п'ять номерів, а не сім – це тому, що очікується, що ви будете телефонувати на ці номери частіше. Однак вам все одно може знадобитися зателефонувати на інші номери – тому перед усіма цими номерами додається цифра «9». Це префіксний код. Кожен елемент, який ви хочете

вказати, має код, що складається з цифр, і оскільки жоден код для одного елемента не починається з коду для будь-якого іншого елемента, ви можете ввести цей код і не буде ніякої двозначності щодо того, що ви маєте на увазі саме цей елемент.

Код Хаффмана – це префіксний код, підготовлений за спеціальним алгоритмом. Тут замість того, щоб кожен код був серією чисел від 0 до 9, кожен код є серією бітів, або 0, або 1. Замість того, щоб кожен код представляв телефон, кожен код представляє елемент певного «алфавіту» (наприклад, набір символів ASCII, що є основним, але не єдиним застосуванням кодування Хаффмана в DEFLATE).

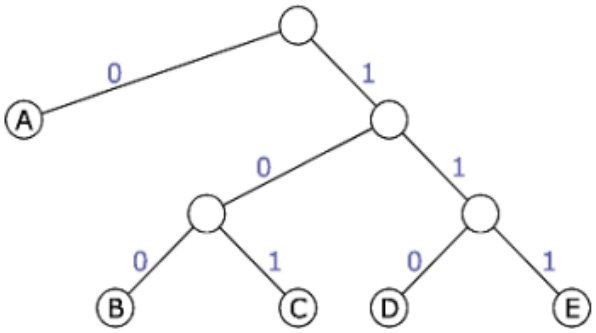

Рисунок 5 – Префіксне дерево Хаффмана

Алгоритм Хаффмана починається зі складання елементів «алфавіту», кожному з яких присвоюється «вага» – число, що відображає його відносну частоту в даних, які потрібно стиснути. Ці ваги можна вгадати заздалегідь, або ж точно, виміряти при проходженні через дані, або ж це може бути комбінація обох способів. У будь-якому випадку, елементи вибираються по два за раз, причому вибираються елементи з найменшою вагою. Ці два елементи стають листками вузла з двома гілками. Спочатку ми виберемо D і E на рисунку 5, і зробимо їх гілками одного вузла – одну гілку «0», а іншу «1».

На даний момент жоден елемент ще не отримав свого повного коду, але ми знаємо, що коди для D і E будуть абсолютно однаковими, за винятком останньої двійкової цифри: D закінчується на 0, а E на 1. Об'єднану вершину D і E поміщаємо назад до інших (ще не об'єднаних) елементів і надаємо їй вагу, що дорівнює сумі

ваг її листків. Знову беремо дві вершини з найменшою вагою, тобто A та D і E, і об'єднуємо їх у більшу вершину.

Коли всі вузли рекомбіновані в єдине «дерево Хаффмана», то, починаючи з кореня і вибираючи 0 або 1 на кожному кроці, ви можете дістатися до будь-якого елемента в дереві. Кожен елемент тепер має код Хаффмана, тобто послідовність 0 та 1, яка представляє цей шлях по дереву.

Тепер досить легко зрозуміти, як таке дерево і такий набір кодів можна використовувати для стиснення. Якщо стискати, наприклад, звичайний текст, то, ймовірно, більше половини набору символів ASCII можна взагалі не включати в дерево. Часто використовувані символи, такі як «E», «T» і «A», ймовірно, отримають набагато коротші коди, і навіть якщо деякі коди будуть зроблені довшими, це будуть саме ті, які використовуються рідше.

У класичному алгоритмі Хаффмана з одного набору елементів і ваг можна згенерувати кілька дерев. У варіації, що використовується стандартом Deflate, є два додаткових правила: елементи з коротшими кодами розміщуються лівіше від елементів з довгими кодами. (У нашому попередньому прикладі елементи D та E мають найдовші коди, тому вони будуть розміщені праворуч). Серед елементів з кодами однакової довжини ліворуч розміщуються ті, що йдуть першими у наборі елементів. (Якщо D і E виявляться єдиними елементами з кодами такої довжини, то D отримає гілку 0, а E – гілку 1, оскільки D стоїть перед E).

Наступною головною частиною Deflete є LZ77. Стиснення LZ77 працює шляхом пошуку послідовностей даних, які повторюються. Використовується термін «ковзаюче вікно», який насправді означає, що в кожній точці даних є запис про те, які символи йшли раніше. Ковзаюче вікно означає, що компресор (і декомпресор) мають запис про те, якими були останні 32768 (32 * 1024) символів. Коли наступна послідовність символів, що підлягає стисненню, ідентична тій, яку можна знайти в ковзаючому вікні, послідовність символів замінюється двома числами: відстанню, що представляє, наскільки далеко назад у вікні починається послідовність, і довжиною, що представляє кількість символів, для яких

послідовність ідентична. LZ77 [14] послідовно переглядає вхідний рядок і зберігає кожен новий збіг у пошуковому буфері.

Процес стиснення можна розділити на 5 кроків.

Крок 1. Встановіть позицію кодування на початок вхідного потоку.

Крок 2. Знайдіть найдовший збіг у вікні для буфера очікування.

Крок 3. Якщо збіг знайдено, виведіть вказівник P. Пересуньте позицію кодування (і вікно) на L байт вперед.

Крок 4. Якщо співпадіння не знайдено, виведіть нульовий вказівник і перший байт у буфері очікування. Пересунути позицію кодування (і вікно) на один байт вперед.

Крок 5. Якщо буфер очікування не порожній, поверніться до Кроку 2.

Алгоритм стиснення шукає у вікні найдовший збіг з початком буфера очікування, а потім виводить вказівник на цей збіг. Оскільки навіть 1-байтовий збіг може бути не знайдено, виведення не може містити лише вказівники. Алгоритм стиснення вирішує цю проблему, виводячи після вказівника перший байт у буфері очікування після збігу. Якщо збігу не знайдено, алгоритм виводить нульовий вказівник і байт у позиції кодування.

Розглянемо приклад для строки «AABCBBABC». В таблиці 1 наведено номер кроку кодування. Крок у таблиці завершується кожного разу, коли алгоритм кодування робить висновок. В алгоритмі стиснення цей процес відбувається при кожному проходженні кроку 3.

Таблиця 1 – Таблиця позицій байтів у вхідному значенні строки

Позиція	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Байт	A	A	B	C	B	B	A	B	C

Перед явним байтом, який показано у стовпчику Byte, може міститися один або декілька вказівників. Тобто, вказівник метаданих не завжди повинен

супроводжуватися явним байтом. Вхідна строка «ABCABCCCC», наприклад, можна подати у вигляді «(0,0)A(0,0)B(0,0)C(3,3)(1,3)», використовуючи нотацію (B,L)C, де останні два елементи є вказівниками без явних байтів. Стислий вивід може бути будь-якою комбінацією вказівників і явних байт.

Результатом стиснення, концептуально, є вихідний стовпчик – тобто серія байт і необов'язкові метадані, які вказують, чи передує цьому байту деяка послідовність байт, яка вже є у вихідному потоці. Перший байт у вхідному потоці має позицію кодування 1. Колонка збіг показує найдовший збіг, знайдений у вікні. Колонка байт показує перший байт у буфері очікування після збігу. Колонка вихідні дані представляє вивід у форматі (B,L)C, де (B,L) – це вказівник (P) на збіг. Це дає наступні інструкції декодеру – повернутися на B байт назад у вікні і скопіювати L байт на вивід. C – це явний байт.

Таблиця 2 – Таблиця кроків для байтів у LZ77

Крок	**Позиція**	**Співпадіння**	**Байт**	**Вихідне значення**
1.	1	--	A	(0,0)A
2.	2	A	--	(1,1)
3.	3	--	B	(0,0)B
4.	4	--	C	(0,0)C
5.	5	B	--	(2,1)
6.	6	B	--	(1,1)
7.	7	A B C	--	(5,3)

Оскільки для представлення самих метаданих потрібні байти у вихідному потоці, неефективно представляти один байт, який раніше був закодований двома байтами метаданих (зсув і довжина). Накладні витрати на байти метаданих дорівнюють або перевищують вартість безпосереднього виведення байт. Тому

протокол вважає послідовності байтів збігом, тільки якщо послідовності мають три або більше спільних байтів.

Загальний алгоритм стиснення для варіанту Хаффмана може обробляти довільну кількість даних. Дані обробляються блоками по 64 кбайт, а закодовані результати зберігаються впорядковано. Після останнього блоку кодується символ кінця файлу (EOF).

Кожен 64-кб блок проходить через три етапи, які виконуються в такому порядку:

1. Виконати стиснення LZ77 для створення проміжного стисненого буфера.
2. Побудувати канонічні коди Хаффмана.
3. Обробити проміжні дані LZ77 та перекодувати їх у бітовий потік на основі Хаффмана.

Алгоритм не може почати кодування Хаффмана, поки не обчислить коди Хаффмана, а він не може обчислити коди Хаффмана, поки не знає частоту кожного символу в алфавіті Хаффмана. Щоб обчислити ці частоти, алгоритм спочатку виконує фазу LZ77. Для ефективності алгоритм повинен зберігати результат LZ77, щоб на фінальній фазі не довелося його перераховувати.

Остаточний формат стиснення складається з двох частин:

– перші 256 байт вказують довжину біта кожного з 512 символів Хаффмана;

– решта даних – це послідовність символів Хаффмана разом з довжинами та відстанями між ними.

Алгоритм Deflate може бути хорошим варіантом для скорочення посилань. Варто зазначити, що алгоритм може використовуватися у сферах, пов’язаних зі штучним інтелектом для розпізнавання мови [15] або зображень [16]. Проте процес скорочення може займати трохи довше часу чим знаходження хеш-коду або просто кодування за допомогою Base. Deflate [17] може бути ефективним якраз для стиснення мультимедійних файлів таких як графічні файли, відео та аудіо файли.

2.1.3 Огляд алгоритму SHA-256 хешування для скорочення посилань

SHA-256 – це безпечна криптографічна хеш-функція. Як така, що її вихідні дані не повинні мати жодних властивостей, які можна було б виявити. Опишемо три бітові рядки, хеші яких за допомогою SHA-256, тим не менш, корелюються нетривіальним чином: перша половина їх хешів XOR з нуль. Вони були знайдені методом «грубої сили» [18], без використання криптографічної вразливості самої хеш-функції. Це не загрожує безпеці хеш-функції і не має жодних криптографічних наслідків.

Це приклад великого «комбінаторного» обчислення, в якому було виконано щонайменше 8,7 × 1022 ціло-чисельних операцій.

Це стало можливим завдяки поєднанню:

– нещодавнього прогресу в алгоритмах для основної проблеми;

– творчого використання «виділених» апаратних прискорювачів;

– адаптованих реалізації відповідних алгоритмів, які могли б працювати на масово паралельних машинах.

В основному для скорочення посиль використовують SHA-256 разом з Base62, оскільки результатом SHA-256 [19] є хеш довжиною 64. Хоча можна скоротити посилання тільки за допомогою алгоритму SHA-256, 64 символи в такому посиланні можуть бути важкими для візуального сприйняття користувачем. Тому можна розглянути використання більш коротких і зрозумілих для людей URL-адрес з кількома словами, що відображають зміст посилання. Тому Base62 є своєрідним алгоритмом, що дозволяє обмежити довжину хешу, тим самим розширює діапазон унікальності посилань та зменшує навантаження на фізичні сховища відносно вже скорочених посилань.

Процес створення хешу складається з 5 кроків:

– наповнення повідомлення;

– визначення змінної;

– робота функції стиснення;

– підрахунок проміжних хеш-значень;

– компоновка значення хешу.

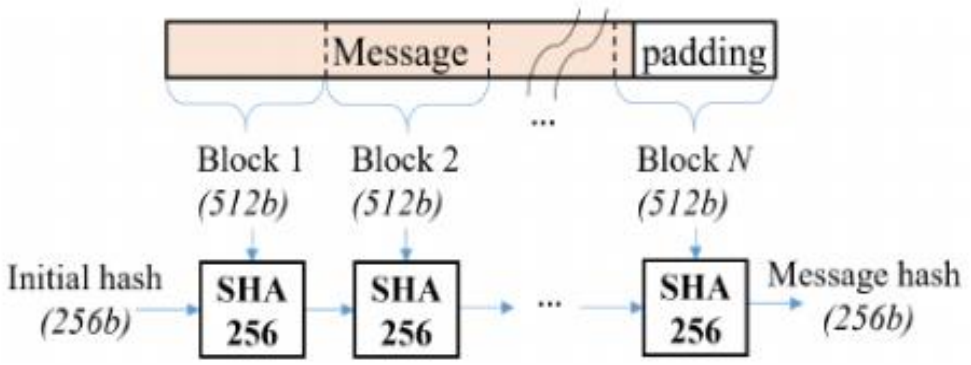

Рисунок 6 – Генерація хеш-значення SHA-256 для довгого повідомлення

Генерація хеш-значення SHA-256 для довгого повідомлення. Як правило, ці методи просто переставляють місцями оператори, необхідні для обчислення раунду SHA-256, такі як суматори, логічні вентилі та обертання, у відповідний та ефективний спосіб, щоб досягти високої продуктивності з точки зору швидкості обробки, площі схеми або затримки критичного шляху. Час роботи схеми SHA-256 включає в себе не тільки час обчислення хеш-значення, але і час передачі даних між зовнішньою пам'яттю і схемою SHA-256. SHA-256 обчислює 256-бітове хеш-значення для вхідного повідомлення довжиною 512 біт. У реальній програмі може знадобитися обчислити хеш-значення для дуже довгого повідомлення. У таких випадках повідомлення розбивається на багато 512-бітних блоків даних. Якщо останній блок менший за 512 біт, додається заповнення. Обчислення хешу для послідовного повідомлення показано на рис. 6. Алгоритм SHA-256 обчислює проміжні хеш-значення для блоків даних один за одним, при цьому результат хешування поточного блоку стає вхідним початковим хешем для обчислення хешу наступного блоку даних. Результат останнього блоку даних вважається хеш-значенням всього повідомлення.

На рисунку 7 показано загальну схему роботи алгоритму SHA-256. Він включає в себе два процеси, які називаються розширювач повідомлення (ME) і стискач повідомлення (MC). Процес ME розширює 512-бітне вхідне повідомлення до 64 частин 32-бітних даних $W_j (0 \leq j \leq 63)$. За перші 16 раундів ME розбирає 512-бітне повідомлення на 16 32-бітних блоків даних (позначаються як $W_{j,j}$= від 0

до 15, де j – індекс раунду). За останні 48 раундів ME обчислює 48 блоків 32-бітних даних $W_j(16 \leq j \leq 63)$ відповідно до рівняння (2). Для обчислення $W_j(16 \leq j \leq 63)$ потрібні три 32-бітових суматори та дві логічні функції $\sigma_0(x)$ та $\sigma_1(x)$. $\sigma_0(x)$ та $\sigma_1(x)$ обчислюються за формулами:

$$\sigma_0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x), \quad (2)$$

$$\sigma_1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x), \quad (3)$$

де $\sigma_i(x)$ – логічна функція хешування,

S^n – перетворення, які змінюють біти вхідного слова згідно з певними таблицями заміни,

R^n – перетворення, які здійснюють циклічні зсуви бітів вхідного слова на певну кількість позицій вправо.

Звертаємо увагу, що $S^n(x)$ та $R^n(x)$ позначають поворот вправо та зсув вправо даних *x* на *n* біт.

Процес *MC* обчислює 256-бітне хеш-значення з бітове хеш-значення з виходів процесу ME (64 частини $W_j(0 \leq j \leq 63)$). Процес складається з двох основних етапів: цикли та оновлення хешу. На етапі циклів вісім хеш-значень циклу (позначені *a, b, c, d, e, f, g, h*) ініціалізуються початковими хеш-значеннями *H0,H1,...,H7*. Потім обчислюються та оновлюються хеш-значення *a, b, c, d, e, f, g, h* у 64 циклах. У кожному циклі (цикл j, $0 \leq j \leq 63$) застосовуються рівняння 4-8.

$$T_1 = h + \sum_1(e) + Ch(e, f, g) + K_j + W_j, \quad (4)$$

$$T_2 = \sum_0(e) + Maj(a, b, c), \quad (5)$$

$$a = T_1 + T_2, \quad (6)$$

$$e = d + T_1, \quad (7)$$

де T_i – проміжні значення розрахування хеш-функцій,

h, e, f, g – 32-бітні слова, що представляють стани регістрів внутрішнього стану хеш-функції,

$\Sigma_i(e)$ – функція, яка виконує логічне доповнення, циклічне зсув та логічне «або» над 32-бітним словом e,

Ch – функція, яка виконує логічне «і», логічне «або» та логічне доповнення над 32-бітними словами e, f та g,

$Maj(a,b,c)$ – функція, яка використовується в алгоритмі SHA-256 для обчислення значення T_2 в кожному з 64 раундів шифрування,

Kj – константа, яка залежить від номеру ітерації (j),

Wj – 32-бітне слово, яке представляє j-те слово вхідного повідомлення (або вихід з попередньої ітерації),

a – змінна суми проміжних значень хеш-функції,

e – змінна суми проміжного значення.

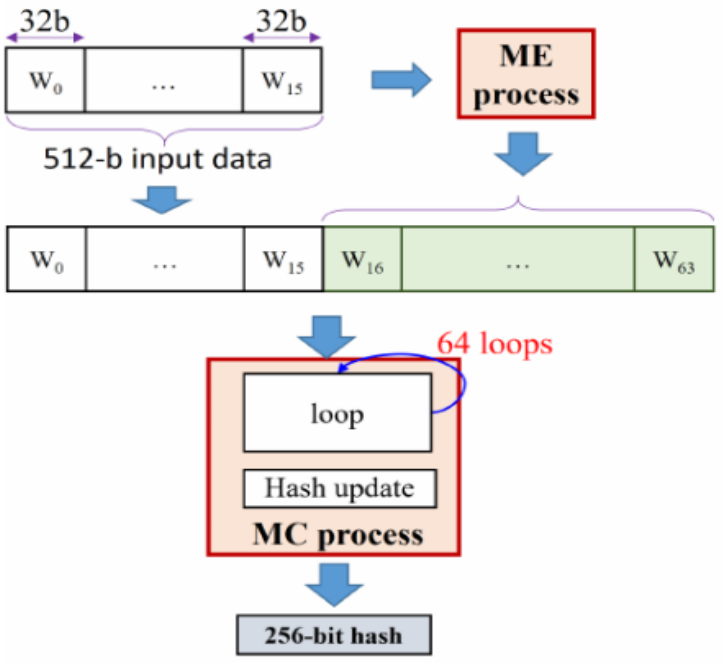

Рисунок 7 – Огляд роботи алгоритму SHA-256

Де логічні функції, такі як $\sum_0(x)$, $\sum_1(x)$, *Ch(x,y,z)* та *Maj(x,y, z)* обчислюються за наступними рівняннями (9–10):

$$\sum_0(x) = S^2(x) \oplus S^{13}(x) \oplus S^{22}(x) \ , \qquad (9)$$

$$\sum_1(x) = S^6(x) \oplus S^{11}(x) \oplus S^{25}(x), \quad (10)$$

де S^n – перетворення, які змінюють біти вхідного слова згідно з певними таблицями заміни,

$\Sigma_i(x)$ – результат логічного виключення над 32-бітними значеннями.

На етапі оновлення хешу остаточне 256-бітове значення хешу, яке розбивається на 8 фрагментів 32-бітових даних $HO_0, HO_1, ..., HO_7$, обчислюється шляхом додавання початкових хешів $H_0, H_1, ..., H_7$ до циклічних хешів *a, b, c, d, e, f, g, h*, як показано у рівнянні (11):

$$HO_0 = H_0 + a; \ldots; HO_7 = H_7 + h, \quad (11)$$

де H_n – значення станів хеш-функції після застосування функції в кожному кроці алгоритму.

Комбінація SHA-256 та Base62 зображена на рисунку 8. У сервісі скорочення посилань Base62 виступає як обмежувач хешу. На схемі зображено, що Base62 з результату хешу обирає 7 символів, а потім зберігає результат скорочення у базу даних

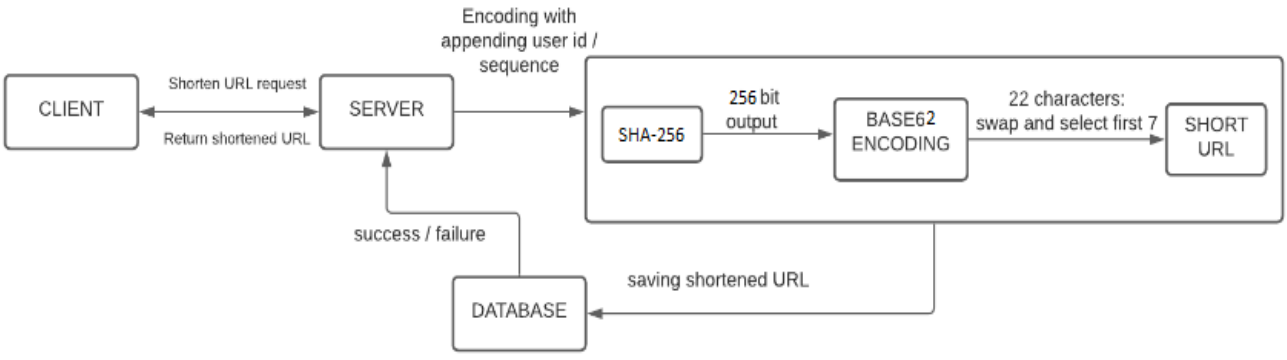

Рисунок 8 – Огляд схеми роботи сервісу скорочення посилань

Сервіс функції хешування повинен бути бездержавним, щоб його можна було легко реплікувати для масштабування. Вхідний трафік розподіляється на сервіс

функції хешування за допомогою балансувальника навантаження. Потенційні алгоритми балансування навантаження для маршрутизації трафіку наведені нижче:

– хешування IP-адреси;
– функція хешування по модулю;
– послідовне хешування.

Однак, ймовірність колізії є вищою через довжину вихідних даних 256 біт. Компроміси рішення з хеш-функцією полягають у наступному:

– передбачуваний результат завдяки хеш-функції;
– вища ймовірність колізії.

2.2 Визначення методики проведення дослідження

Вплив обраного підходу відносно скорочення посилання у високонавантажених системах можна спостерігати при достатньо великому трафіку користувачів.

Важливо дбати про продуктивність на всіх етапах розробки системи, а не тільки у певний критичний момент в разі появи проблем.

Важливо пам'ятати, що продуктивність є ключовою складовою успіху будь-якої програми. Це стосується не тільки критичних моментів, але і всіх етапів розробки. Оцінка програмного забезпечення на стадії проектування є надзвичайно важливою, оскільки дозволяє заздалегідь оцінити продуктивність програми та врахувати можливі ризики проекту та етапів розробки. Це також дозволяє розробникам порівняти необхідні витрати та майбутню вартість проекту. Таким чином, при проектуванні програмного забезпечення слід враховувати продуктивність на всіх етапах розробки, а не тільки в критичний момент. Попередня оцінка програмного забезпечення дозволяє зменшити ризики проекту та збільшити його шанси на успіх.

Швидкість роботи системи може легко погіршитися, якщо використовуються неправильні бібліотеки, не тільки для управління станом. Крім того, метрики та їх граничні значення зазвичай визначаються індивідуально для кожного проекту, якщо такі є. Таким чином, важливо уважно підходити до вибору бібліотек та інших

інструментів, щоб забезпечити максимальну продуктивність додатка. Крім того, слід заздалегідь встановлювати метрики та граничні значення, щоб забезпечити якісну роботу додатка та підтримувати його швидкість.

Для виміру швидкодії у .NET існує BenchmarkDotNet – це потужна бібліотека для проведення бенчмарків, тобто вимірювання продуктивності коду в різних умовах. Ця бібліотека дозволяє легко створювати тестові набори, визначати параметри тестування, проводити тестування в автоматичному режимі та отримувати звіти з результатами тестування в різних форматах. BenchmarkDotNet дозволяє проводити порівняльний аналіз різних варіантів реалізації алгоритмів, що допомагає знайти оптимальний варіант для конкретного випадку використання.

BenchmarkDotNet містить такі показники:

– середнє значення часу виконання для кожного методу;
– найменше і найбільше значення часу виконання для кожного методу;
– середнє значення часу виконання для одного виклику методу;
– графік, що показує, як різні виклики методів розподіляються в часі;
– кількість пам'яті, використаної під час виконання методу;
– точність вимірювання часу виконання;
– статистичні значення – такі як середнє значення, медіана, стандартне відхилення та коефіцієнт варіації для кожного методу.

Ці показники допомагають проводити детальний аналіз продуктивності програмного коду та знаходити шляхи для її оптимізації.

Крім показників бенчмаркінгу, існують також інші критерії відносно скороченого посилання, такі як ступінь унікальності скороченого посилання, стійкість до атак, вартість ресурсів для зберігання скорочених посилань тощо.

Тим не менш, метрики, створені BenchmarkDotNet, були враховані. Передбачалося, що найбільший вплив методу скорочення посилань на розмірі вхідних даних.

Для проведення дослідження та порівняння методів скорочення URL посилань були визначені наступні показники:

– дельта відсотка швидкодії функції, необхідного для скорочення URL посилання;

– відсоток колізій довільних URL посилань;

– коефіцієнт унікальності даних;

– відсоток зменшення початкового посилання;

– дельта масштабованості функції в залежності від вихідних даних;

– дельта пам’яті від збереження скорочених даних;

– метрика відносно пам’яті зберігання скорочених посилань.

Чим більше буде поступати даних до системи, тим більше потрібно часу для скорочення даних. Існують певні обмеження для методів скорочення URL посилань. Більшість сервісів скорочення URL мають обмеження довжини URL.

Також існують мовні обмеження, оскільки деякі сервіси скорочення URL можуть не підтримувати певні мови або символи. Це може стати проблемою для користувачів, які хочуть скоротити посилання з використанням символів, що не підтримуються сервісом.

Скорочення URL-адресів є дуже корисною функцією в інтернеті. Однак, якщо вихідний URL занадто довгий, то сервіс може відмовитися його скорочувати.

Навіть якщо URL було успішно скорочено, важливо забезпечити відповідні показники взаємодії з користувачами, такі як кількість кліків на скорочене посилання, час збереження посилання, кількість використань збереженого посилання та інші. Доводити їх важливість немає потреби, оскільки це є основою для методів скорочення. Неправильно обраний метод для скорочення посилання у системі може призвести до вповільнення відносно скорочення, а також надлишкового збільшення скороченого посилання. Тому необхідно ретельно обирати методи скорочення та забезпечувати відповідну взаємодію з користувачами, щоб забезпечити ефективне та зручне використання скороченого посилання.

3 СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ ДЛЯ ДОСЛІДЖЕННЯ

3.1 Функціональні вимоги

Для того щоб вивчити вплив різних підходів для скорочення URL посилань для дослідження швидкодії, було створено веб серверний застосунок на платформі .NET 6 з використанням мови програмування C#.

До розробленого серверного застосунку було визначено наступні функціональні вимоги:

– програма дозволяє ввести URL адресу посилання;

– програма скорочує адресу;

– програма дозволяє обрати алгоритм для скорочення посилання;

– програма відображає спектр статистичних даних відносно спроби скорочення URL посилання;

Усі скорочені посилання будуть зберігатися у оперативній пам’яті додатку під час роботи процесу програми.

Згідно з перерахованими функціональними вимогами, у тестовому застосунку має бути організовано роботу з алгоритмами, які користувач може обрати для скорочення посилання.

Реалізація функціональних вимог за допомогою різних методів для URL-адрес дозволяє порівнювати їх без будь-яких припущень або змін в даних результату, тому що експеримент буде проводитися на одних і тих же даних. Це дає змогу отримати точні і порівнянні результати.

Відповідно до функціональних вимог, зазначених вище, сутності в додатку будуть відсутні. Тестова програмна реалізація зосереджена тільки на скорочення посилань, основний функціонал описано нижче:

– вибір алгоритму скорочення посилання;

– скорочення URL посилання;

– відображення статистичних даних.

Презентаційна частина програми має бути розроблена за платформи .NET за допомогою мови програмування C# та має залишатися незмінною під час

дослідження, тоді як алгоритми скорочення будуть реалізовані окремо за допомогою обраних бібліотек, які також розроблені на базі .NET.

3.2 Розробка презентаційної частини

Для створення користувацького інтерфейсу було використано фреймворк ASP.NET Core на базі .NET 6. Ця бібліотека передбачає, що інтерфейс формується як композиція із різних компонентів. ASP.NET Core – це кросплатформовий фреймворк з відкритим вихідним кодом, призначений для розробки веб-додатків на платформі .NET. Його розробляє компанія Майкрософт у співпраці зі спільнотою. Завдяки високій продуктивності, яка є більш високою, ніж у ASP.NET, він дуже популярний. ASP.NET Core має модульну структуру та підтримує операційні системи, такі як Windows, Linux та macOS. Запуск тестового додатку відбувається на локальній машині за допомогою вбудованого серверу IIS.

Для зручної взаємодії з API було використано специфікацію Swagger, яка є стандартом OpenAPI для REST додатків. Цей інструмент допомагає зрозуміти комп'ютерам та користувачам можливості REST API, не звертаючись безпосередньо до вихідного коду. Його головні цілі полягають у зменшенні обсягу робіт, що потрібні для з'єднання окремих сервісів, та скороченні часу, необхідного для опису документації цих сервісів.

Swagger – це графічний інтерфейс, який надає користувачеві веб-інтерфейс з інформацією про службу, використовуючи специфікацію OpenAPI. Swashbuckle та NSwag містять вбудовану версію інтерфейсу Swagger, який може бути включений в додаток ASP.NET Core за допомогою реєстрації ПЗ проміжного рівня.

Основний екран веб-застосунку передбачає відображення списку методів обробників запитів. Кожен метод взаємодії з API має назву та назву HTTP-методу GET, що запитує представлення вказаного ресурсу. На рисунку 9 відображено методи контролерів для скорочення посилань.

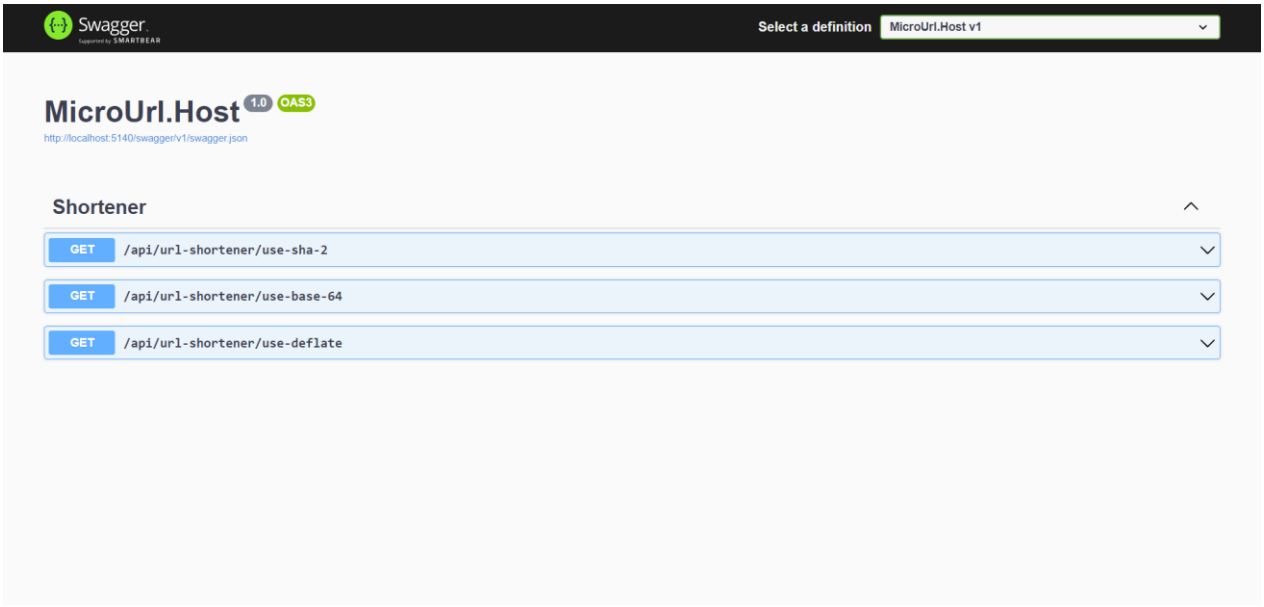

Рисунок 9 – Головний екран застосунку

Після створення веб-додатку стало можливо виділення системи компонентів. Таким чином, наступні компоненти були визначені як необхідні для реалізації:

– IShortener;
– ShortneterController;
– Base64Shortener;
– DeflateShortener;
– Sha2Shortener.

Згідно діаграми класів інтерфейс IShortener містить сигнатуру методу, в параметрах якого на вхід приходить параметр строкового типу – це і є вхідна строка URL посилання. Інтерфейс виділяє тільки один метод, який сервісі будуть імплементувати для реалізації власного підходу – відносно скорочення URL посилань. На діаграмі класів (рис. 10) видно, що сервіси Base64Shortener, DeflateShortener, Sha2Shortener реалізують інтерфейс IShortener. Реалізація інтерфейсів є важливою частиною розробки програмного забезпечення, оскільки дозволяє забезпечити більш гнучкий та простий в обслуговуванні код. Реалізація інтерфейсів є важливою частиною розробки програмного забезпечення, оскільки дозволяє забезпечити більш гнучкий та простий в обслуговуванні код.

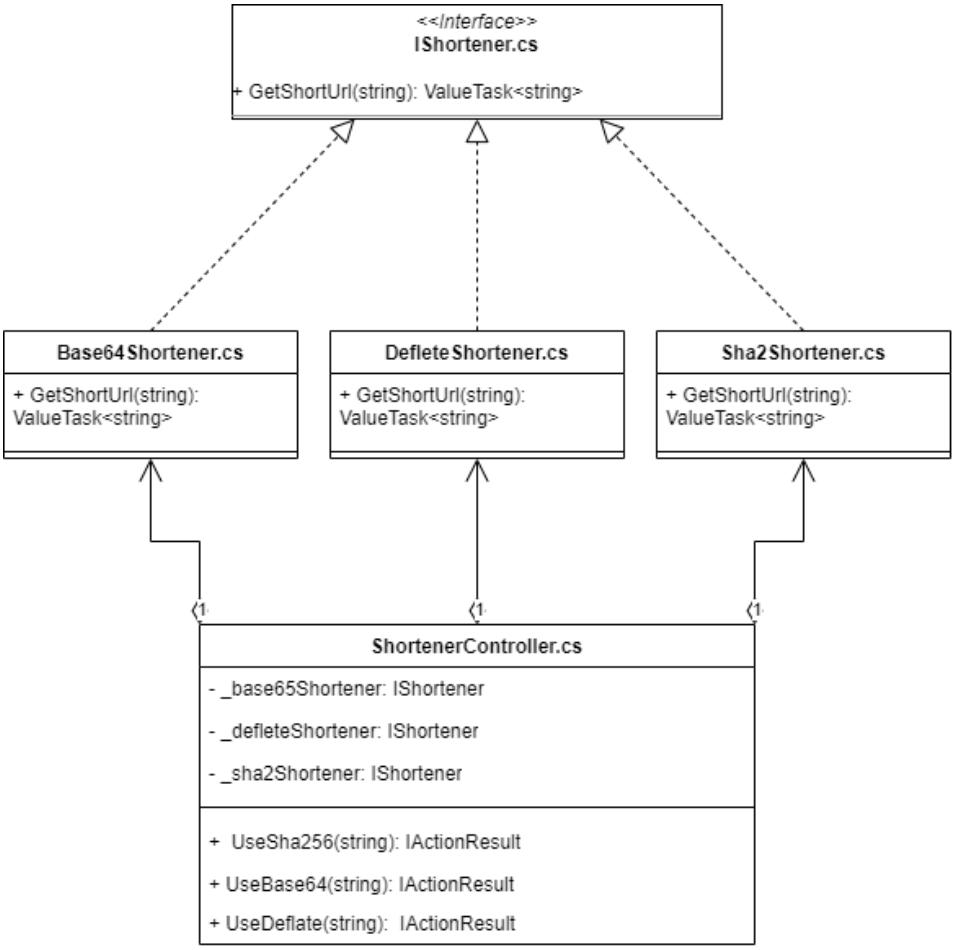

Рисунок 10 – Діаграма класів тестового додатку

Інтерфейси дозволяють досягти слабкої зв'язності між різними компонентами програми. Завдяки використанню інтерфейсів, класи можуть спілкуватися між собою через обмін повідомленнями, не знаючи деталей реалізації один одного. Це дозволяє збільшити гнучкість та можливості модифікації системи, оскільки різні компоненти можуть замінюватися один на одного, не порушуючи роботу інших компонентів. Таким чином, використання інтерфейсів підвищує модульність програми і полегшує її розширення та підтримку. Контролер ShortenerController містить у конструкторі залежності від наших сервісів для скорочення посилання. За допомогою вбудованого контейнера ін’єкції залежностей можна досягти склабкої зв’язності. Оскільки ShortenerController контролер в ASP.NET Core – це клас, який обробляє вхідні запити веб-додатку. Він приймає запити від користувача, виконує необхідні дії та повертає відповідь. Контролери використовуються для реалізації логіки додатку та розподілу

відповідальності між різними компонентами. На рисунку 10 ShortenerController агрегує сервіси Base64Shortener, DefleteSHortener, Sha2Shortener, які релізують інтерфейс IShortener, використовуючи їх функціональність для скорочення URL посилань. В контейнері залежностей ми реєструємо інтерфейс та реалізацію наших сервісів:

```
public static void Register(this IServiceCollection services)
{
   services.AddSingleton<IShortener, Sha2Shortener>();
   services.AddSingleton<IShortener, Base64Shortener>();
   services.AddSingleton<IShortener, DefleteShortener>();
}
```

ASP.NET Core підтримує патерн проектування програмного забезпечення із використанням впровадження залежностей (DI), що є технікою для досягнення Інверсії Керування між класами та їх залежностями. Реєстрація залежностей відбувається за допомогою методу AddSingleton, основна властивість якого контролювати об’єктом залежності протягом життєвого циклу додатку. Метод життєвого циклу залежностей AddSingleton створює тільки один об’єкт протягом усього життєвого циклу додатку, тобто для кожного запиту в місцях де залежність потребує реалізації певного сервісу буде створено тільки один об’єкт.

AddSingleton є корисний в випадках коли нам потрібно обмежити алокацію об'єктів в оперативній пам’яті (купі).

У програмі були реалізовані всі функції, які необхідні для успішного проведення дослідження.

3.3 Реалізація скорочення URL посилань

3.3.1 Реалізація скорочення URL посилань за допомогою Base64

Для створення функціоналу зі скорочення URL-адрес було використано бібліотеку UrlBase64. UrlBase64 – це пакет для всіх версій .NET Framework та .NET Core, який підтримує створення безпечного для використання в інтернеті base64 кодування URL-адресів. UrlBase64 надає симетричні функції для кодування та декодування даних таким чином, щоб його можна було безпечно використовувати

в загальному вебі та як вхідні дані до веб-додатків ASP.NET (під IIS або подібному).

Всі функції містяться в статичному класі UrlBase64, який знаходиться в просторі імен NeoSmart.Utils. UrlBase64 підтримує два різних режими доповнення (внутрішньо: PaddingPolicy) для генерації кодування base64:

– PaddingPolicy;
– DiscardPolicy.

Параметр PaddingPolicy керує поведінкою UrlBase64 під час кодування вмісту, який не попадає на границю з 4 символами (виводу). Згідно з RFC 4648, base64 вказує на те, що необов'язковий, залежно від обставин знак «=» використовується для доповнення виводу до кратного 4 символам довжини. UrlBase64 підтримує як вивід з доповненням, так і без нього за допомогою необов'язкового параметра PaddingPolicy для методу UrlBase64.

Encode, який керує цією поведінкою. За замовчуванням на даний час поведінка полягає в опусканні знаку доповнення, оскільки він може бути автоматично визначений при опусканні з закодованого виводу, і використовує символ, який потрібно кодувати при використанні в URL.

Для того, щоб отримати дані з посилання, було використано метод Encode, який надається бібліотекою UrlBase64. Нижче наведено приклад коду, необхідного для конвертування посилання у строку формату Base64:

```
var converted = UrlBase64.Encode(bytes,PaddingPolicy.Preserve);
```

Таким чином відбувається кодування вхідних даних компоненту до глобального стану. Розглянемо метод UrlBase64 більш детально:

```
public static string Encode(byte[] bytes, PaddingPolicy padding = 
PaddingPolicy.Discard)
{
string text = Convert.ToBase64String(bytes).Replace('+', '-').Replace('/', 
'_');
if (padding == PaddingPolicy.Discard)
{
text = text.TrimEnd(new char[1] { '=' });
```

```
}
return text;
}
```

Проаналізувавши код бібліотеки можна побачити що для конвертації даних використовується нативний метод Convert.ToBase64String. Особливістю цього методу є те, що строка очищається від символів «+», «/», «_», «–». Варто звернути увагу на умову, яка перевіє відступ, який представляє символ «=» або «==».

3.3.2 Реалізація скорочення URL посилань за допомогою Deflate

Для реалізації скорочення посилання за допомогою Deflate були використана бібліотека. Shuttle.Core.Compression – це бібліотека з відкритим вихідним кодом для стиснення та розпакування даних в .NET-програмах. Вона розроблена компанією Shuttleworth для використання в проектах з великою кількістю даних, які потребують зберігання в ефективному форматі.

Бібліотека Shuttle.Core.Compression надає API для роботи з різними форматами стиснення даних, такими як GZip, Deflate, BZip2 та інші. Вона дозволяє програмістам легко стискувати та розпаковувати файли та потоки даних без необхідності писати власний код для цього.

Shuttle.Core.Compression також має підтримку декомпозиції стиснених файлів, що дозволяє розбивати стиснений файл на частини для зберігання в різних місцях або передачі по мережі.

Нижче наведено частину коду для встановлення Deflate, що виконує стиснення даних. Щоб додати конкретні алгоритми стиснення, скористайтеся відповідними викликами будівельника (builder calls):

```
services.AddCompression(builder => {
            builder.AddDeflate();
});
```

Цей код спробує додати один екземпляр (singleton) CompressionService:

```
services.AddCompression();
```

Надає адаптер стиснення через інтерфейс ICompressionAlgorithm.

Реалізації, доступні в цьому пакеті:

– DeflateCompressionAlgorithm;
– GZipCompressionAlgorithm;
– NullCompressionAlgorithm.

Розглянемо DeflateCompressionAlgorithm. Клас DeflateCompressionAlgorithm реалізує інтерфейс ICompressionAlgorithm, який містить методи для стиснення та розпакування даних.

Метод Compress використовує клас DeflateStream з простору імен System.IO.Compression для стиснення переданих йому байтів. Спочатку він створює об'єкт MemoryStream з використанням пула MemoryStreamCache.Manager, потім створює об'єкт DeflateStream, який записує стислі дані в створений MemoryStream. Нарешті, метод повертає байти з MemoryStream у вигляді масиву byte[]. Приклад реалізації стиснення вхідних даних.

```
public byte[] Compress(byte[] bytes)
{
Guard.AgainstNull(bytes, nameof(bytes));
using (var compressed = MemoryStreamCache.Manager.GetStream())
{
using (var gzip = new DeflateStream(compressed, CompressionMode.Compress,
true))
{
gzip.Write(bytes, 0, bytes.Length);
}
   return compressed.ToArray();
  }
}
```

Клас Guard надає методи для перевірки аргументів на null, щоб уникнути помилок під час виконання.

Усі інші функції щодо скорочення посилань, які необхідні для взаємодії з прикладним програмним інтерфейсом, були створені за прикладом вищенаведених.

3.3.3 Реалізація скорочення URL посилань за допомогою SHA-2

Реалізація методу скорочення посилань дуже схожа на Deflate та Base64. Так, щоб використати SHA-2 [20], необхідно підключити просторі імен

System.Security.Cryptography. Класс Sha256 є частиною ядра .NET і надає хеш, який використовується як унікальне значення фіксованого розміру, що представляє велику кількість даних. Хеші двох наборів даних повинні збігатися тоді і тільки тоді, коли відповідні дані також збігаються. Невеликі зміни в даних призводять до великих непередбачуваних змін у хеші. Розмір хешу для алгоритму SHA256 становить 256 біт. Базова реалізація скорочення посилання буде виглядати наступним чином:

```
public ReadOnlySpan<char> GetShortUrl(ReadOnlySpan<char> url, int limit = 8)
{
   using (var sha2 = SHA256.Create())
{
   var bytes = Encoding.UTF8.GetBytes(url.ToString());
   var hash = sha2.ComputeHash(bytes);
   var base62Span = hash.ToBase62().AsSpan();
   return base62Span[..limit];
   }
}
```

Відмінність полягає у тому, що для Recoil немає необхідності створювати єдиний об’єкт глобального стану та передавати його напряму.

На вхід метод скорочення приймає структуру ReadOnlySpan<char> що забороняє алокацію строк, оскільки строка є посилання, тому для зменшення навантаження на пам’ять використовується структурний тип ReadOnlySpan обмежений типом char, оскільки строка є масивом з символів типу char.

У першому рядку створюється новий екземпляр класу SHA256 за допомогою методу SHA256.Create(). Далі рядок url перетворюється у масив байтів bytes за допомогою методу Encoding.UTF8.GetBytes(url.ToString()). За допомогою методу ComputeHash об'єкту sha2 обчислюється хеш hash від масиву байтів bytes. Далі, за допомогою розширення методу ToBase62(), який є методом розширенням та конвертує масив байтів хешу у строку. Для того щоб очистити хеш від символів «+», «/» та «=», було використано бібліотеку UrlBase62. В результаті використання отриманий хеш перетворюється в рядок з базовими 62 символами .

Об'єкт Span<char>, який забезпечує доступ до ділянки пам'яті, де знаходиться перетворений хеш, за допомогою методу AsSpan(). З допомогою синтаксису зрізів

виконується обмеження довжини хешу до значення limit. На виході повертається відрізок Span<char>, який містить скорочену URL.

Для обмеження довжини було використано оператор ранжування. В результатів виконання функції хешування та виконання над ним кодування за допомогою 62 символів буде створюватися строка з 43 символів. Звичайно, для того щоб обмежити символьне посилання використовується певний ліміт, який за замовчуванням дорівнює 8:

```
base62Span[..limit]
```

Такий підхід може бути не дуже ефективним, оскільки хеш-функції можуть повертати для різних значень один й той самий хеш.

Як можна побачити з наведеного коду, платформа .NET надає значну перевагу відносно розрахунку хешу та його перетворення до формату Base62.

4 ОПИС ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ

У межах даної роботи було проведено два типи вимірювань:

– вимірювання швидкодії;

– вимірювання колізій.

Для вимірювання швидкодії коду проекту було використано бібліотеку BenckmarkDotNet [21].

Результати, що були отримані при вимірюванні коду, наведено в таблиці 3.

Таблиця 3 – Вимірювання швидкості коду відносно кількості ітерацій

№	Кількість ітерацій	Вимірювання з UrlBase64 у (сек.)	Вимірювання з Deflate у (сек.)	Вимірювання з Sha256 + Base62 у (сек.)
1	10	0,12	0,25	0,07
2	1000	0,43	1,5	0,25
3	10000	1,02	1,44	0,54
4	100000	2,75	5,19	1,49

Наступна формула (12) використовується для розрахунку відсотка дельти в швидкості виконання коду на різних ітераціях, необхідного для скорочення посилань.

$$V_{\Delta bm} = \frac{V_1 - V_2}{V_1} * 100\%, \quad (12)$$

де V_1 – це найгірший час обробки функції даних, який був виміряний під час ітерацій,

V_2 – це найкращий час обробки функції даних, який був виміряний під час ітерацій.

Для отримання найкращого і найгіршого часу було проведено 10 вимірювань і отримані результати були об'єднані. Це було зроблено для отримання більш точного результату.

Результати розрахунків представлені в таблиці 4.

Таблиця 4 – Вимірювання дельти швидкості коду в залежності від ітерацій

№	Кількість ітерацій	Вимірювання з UrlBase64	Вимірювання з Deflate	Вимірювання з Sha256 + Base62
1	10	75%	28%	57%
2	1000	25%	32%	47%
3	10000	51%	64%	46%
4	100000	45%	21%	53%

Кількість вхідних даних має важливе значення для процесу скорочення URL-адреси, оскільки зі збільшенням обсягу даних збільшується час виконання функції (див. табл. 5).

Таблиця 5 – Вимірювання швидкості коду відносно довжини вхідних даних

№	Розмір масиву вхідних даних	Вимірювання з UrlBase64 у (наносек.)	Вимірювання з Deflate у (наносек.)	Вимірювання з Sha256 + Base62 у (наносек.)
1	16	311	8059	9245
2	32	514	9057	9386
3	64	841	11770	9530
4	128	1000	14000	10000

Розглянемо результати бенчмаркінгу для посилання довжиною 32. Бібліотека розрахує середнє значення, стандартне відхилення, а також загальну алокацію даних та алокацію поколінь об’єктів на купі.

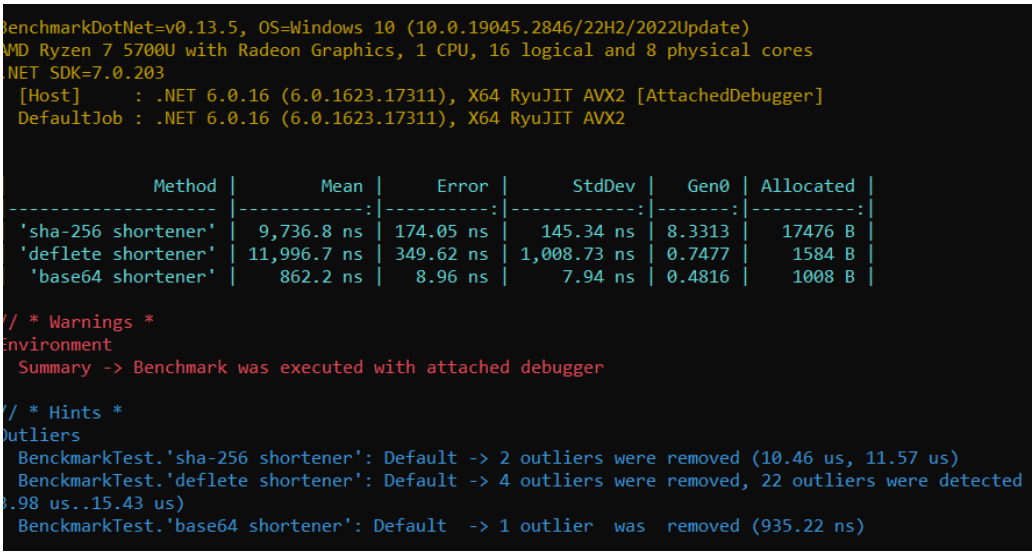

Рисунок 11 – Результати таблиці бенчмаркінгу

Швидкість обробки посилань зменшується пропорційно розміру вхідних даних. Орієнтуючись на швидкість, ми можемо розрахувати скільки даних обробляється за одиницю часу (див. табл. 6). Для обчислення скільки даних обробляється за одиницю використовується така формула (13):

$$V = \frac{L}{T}\ , \tag{13}$$

де V – швидкість обробки даних за одиницю часу в Б/С,

L – розмір вхідних даних в Б,

T – загальний час на обробку.

Таблиця 6 – Вимірювання швидкості коду відносно довжини вхідних даних

№	Розмір масиву вхідних даних	Вимірювання з UrlBase64 у (байт/сек)	Вимірювання з Deflatey у (байт/сек)	Вимірювання з Sha256 + Base62 у (байт/сек)
1	16	51402	8059	1732,63
2	32	62345,99	3530,17	3411,41
3	64	76142,01	5439,05	6722,14
4	128	128000	9142,86	12800

Наступним дослідженням ми зробимо вимірювання колізій у даних. Для вимірювання колізій ми будемо використовувати наступні кроки:

1. Створіть словник, де ключем буде хеш, а значенням буде список початкових даних, які хешуються в цей ключ.

2. Для кожного початкового значення обчисліть його хеш та додайте його до словника.

3. Якщо ключ вже є в словнику, це означає, що відбулась колізія, і потрібно додати поточне значення до списку значень для цього ключа.

Кількість ітерацій в такому алгоритмі залежить від обраного діапазону символів та довжини скороченого URL. Чим довший скорочений URL та більший діапазон символів, тим менша ймовірність колізії, але при цьому збільшується кількість ітерацій для генерації унікального скороченого URL. В таблиці 7 наведено кількість колізій в методах скорочення відносно ітерацій.

Таблиця 7 – Пошук колізій

Кількість ітерацій	Вимірювання з UrlBase64	Вимірювання з Deflate	Вимірювання з Sha256 + Base62
500 000	4	0	0
1500 000	48	18	2
1000 0000	424	37	14
100000000	2424	84	29

Процент колізій відносно кількості даних розраховується за формулою (14):

$$h_{\%} = \frac{h_c}{V} * 100\%, \tag{14}$$

де $h_{\%}$– процент колізії,

h_c– кількість колізій знайдена за всі ітерації,

V – загальний об’єм даних .

Процент колізій відносно кількості даних є важливим показником ефективності скорочення посилань. Цей показник визначається як відношення кількості скорочених посилань, які мають однаковий хеш, до загальної кількості скорочених посилань. Якщо процент колізій дуже високий, то це означає, що велика кількість посилань можуть мати один і той же скорочений варіант, що зменшує ефективність скорочення посилань. У такому випадку можна розглянути застосування більш складних алгоритмів хешування або збільшення довжини скороченого посилання. Навпаки, якщо процент колізій низький, то це свідчить про високу ефективність скорочення посилань та мінімальну ймовірність конфліктів. Однак в такому випадку можуть виникнути питання щодо використання складних алгоритмів хешування, які можуть займати більше часу на обробку даних.

Результати вимірювань представлені в таблиці 8 зобразимо значення проценту колізій відносно кількість даних.

Таблиця 8 – Середнє значення проценту колізій

Значення для UrlBase64	Значення для Deflate	Значення для версії Sha256 + Base62
14%	5%	2%

Для отримання значення було проведено 10 вимірювань і отримані результати були об'єднані, після чого їх було підраховано і знайдено середнє значення.

5 АНАЛІЗ ПРОВЕДЕНОГО ДОСЛІДЖЕННЯ

Три методи для скорочення URL посилань, створених за допомогою .NET, порівнювалися з точки зору швидкодії. Для отримання повної інформації про обрані методи було розглянуто та виміряно декілька показників. Це означає, що були виконані дослідження, під час яких було вивчено обрані методи, та були виміряні деякі параметри цих методів для отримання повної інформації про їхню ефективність та продуктивність.

В ході аналізу предметної області було розглянуто та розкриту тему відносно скорочення URL посилань. Основною метою аналітичного огляду було зображення проблематики скорочення посилань. Було розглянуто архітектурний підхід для проектування сервісів скорочення посилань. Основними модулями в таких системах виступає серверна та сховище зберігання даних. Крім того, варто зазначити, що також і правильно вибраний підхід щодо скорочення посилань також впливає на роботу системи в цілому. Проміжні сервери, які дозволяють хешувати скорочені посилання, дозволяють зменшити навантаження на доступ до сховища і також збільшити відгук сервера на запити клієнта.

В аналітичній частині дослідження було розглянута основна проблематика скорочення посилань, на яку потрібно обов'язково звертати увагу при розробці подібних систем. Також були визначені основні цілі, які дозволили дослідженню розкритися в повній мірі. Були поставлені задачі дослідження основних властивостей методів та їх вплив на швидкодію роботи алгоритмів.

У розділі 2 були розглянуті основні методи та алгоритми, включаючи популярні сервіси, які використовуються для скорочення посилань. Слід звернути увагу, що дослідження було зосереджено на розумінні принципу роботи трьох алгоритмів, що можуть використовуватися у реальній системі для скорочення посилань.

В ході проведення дослідження було створений веб-серверний додаток за допомогою фреймворку ASP.NET Core Web API [22]. За допомогою OpenAPI специфікації розроблено основні методи обробки запитів користувача щодо методів скорочення посилання. Для того щоб забезпечити тестовий додаток

можливістю взаємодії з методами скорочення, було створено три окремих обробки вхідних запитів, які містяться у собі сервіс, що надають користувачу можливість скоротити бажане посилання.

В ході виконання роботи були розглянуті основні можливості алгоритмів Base64, Deflate та SHA256. В дослідженні детально описано фундаментальний принцип роботи для кожного алгоритму. Для проведення дослідження вирішено використати бібліотеки, які вже містять реалізацію цих алгоритмів. Платформа .NET надає переваги відносно використання бібліотек, написаних за допомогою мови програмування C#. В якості бібліотек, які реалізують алгоритми, що потрібні для скорочення посилань, було використано наступні бібліотеки: UrlBase64, Shuttle.Core.Compression та нативна реалізація SHA256 плюс реалізація алгоритму кодування Base62.

В експериментальній частині дослідження за мету було поставлено дослідити два типи метрик:

– швидкодія алгоритму;
– вимір колізій.

Вимір швидкодії алгоритмів для скорочення посилань полягає у вимірі швидкості обробки відносно кількості ітерацій.

Дивлячись на таблиці вимірювання можна зробити висновки, що алгоритм SHA256 + Base62 показав себе найбільш стабільним при зростанні кількості ітерацій. Проте не менш ефективним є Base64. Deflate є найбільш нестабільним при зрості ітерацій, оскільки стиснення даних розбивається на два інших алгоритми і через це виникають певні проблеми для скорочення даних.

У експерименті для вивчення швидкодії скорочення посилань була розрахована дельта швидкодії алгоритмів. Ідея полягала в тому, щоб визначити яка буде дельта при найкращому та найгіршому часу виконання скорочення при збільшенні ітерацій. Для кожного з алгоритмів була розрахована дельта швидкості у процентному співвідношенні.

Наступним кроком у дослідженні швидкодії алгоритмів був розрахунок часу виконання в залежні від розміру вхідних даних. Гарний показник отримано у

Base64 при всіх розмірах даних, на відміну від Sha256 + Base62. Оскільки сам процес скорочення посилання у Sha256 та Base62 складається з обрахування хешу 265 байтів, а потім перетворення його до Base62 формату, займає певний час. Крім того, у Deflate була певна перевага у скороченні посилань при довжинах 16 та 32, однак при 64 та 128 розмірах Sha256 показав себе найкращим порівняно з Deflate.

Відносно розрахованих даних, а саме часу обробки відносно довжини вхідних даних, було обраховано і пропускну здатність скорочення посилання. Для дослідження швидкодії алгоритмів була використана бібліотека BenchmarkDotNet, яка дозволяла виміряти середнє значення роботи алгоритмів, їх стандартне відхилення, відслідкувати алокацію об’єктів у поколіннях купи, а також загальну алокацію даних.

Слід вказати, що наступною не менш важливою метрикою у дослідженні було підрахування колізій у алгоритмах. Дана метрика відображала при яких кількостях даних алгоритм скорочення даних може видавати однакові результати. Даний метод є дуже простим у реалізації, проте для дослідження на більш великих даних, ніж ті що були заявлені у дослідженні, потрібно вертикально масштабувати фізичний пристрій, що збільшує пропускну здатність скорочення URL посилань. Відносно дослідження показників було прийнято рішення дослідити проценте відношення колізій до кількості експериментальних даних. В результаті дослідження колізій алгоритмів з'ясувалося, що найбільший процент колізій випадає на Base64, оскільки цей метод є самим простим і використовує тільки 64 символи для кодування вхідного посилання, в той час як Sha256 та Defalte оперують набагато більшою кількість символів для скорочення URL посилання та роблять відповідні перестановки символів для мінімізації утворення колізій у даних.

Узагальнений опис проведеного дослідження показав, що всі три методи є ефективними при скороченні посилань, але існують обмеження щодо швидкості методів при збільшенні кількості та довжини даних. Для подальшого дослідження рекомендується використовувати реальні навантаження на систему з функціональністю скорочення посилань, щоб отримати реальну швидкість методів.

Крім того, варто додати можливість користувачам обирати метод скорочення, який вони хочуть використовувати для своїх посилань. У разі проблем з швидкодією алгоритмів, можна використовувати підходи горизонтального масштабування системи та балансування запитів між серверами, що значно поліпшить швидкість обробки скорочень посилань.

Отже, дослідження показало, що скорочення посилань – це ефективний спосіб зменшення довжини посилань, але він ще потребує додаткових досліджень та оптимізації для досягнення максимальної швидкості та ефективності.

Важливо враховувати захист від можливих атак на систему скорочення посилань. Для цього рекомендовано використовувати захисні механізми, такі як валідація введеного користувачем URL, фільтрація введення на наявність шкідливих елементів та застосування механізмів обмеження кількості запитів від одного користувача.

Також варто звернути увагу на масштабованість системи, тобто здатність до збільшення кількості запитів і користувачів. Необхідно використовувати технології та підходи, які дозволяють легко розширювати обсяги обробки даних та зберігання скорочених посилань.

Потрібно стежити за швидкістю роботи системи та забезпечувати легку навігацію для користувачів. Додатково, можна розглянути можливість додавання функціональності, такої як аналітика переходів за скороченими посиланнями для поліпшення користувацького досвіду та збільшення цінності системи скорочення посилань.

ВИСНОВКИ

На даний момент існує велика низка сервісів та бібліотек для скорочення URL посилань. Деякі з них надають різні особливості щодо скорочення посилань, такі як: період часу по якому можна переходити на посилання, регіональні обмеження щодо переходу на посилання, відслідковування кількості кліків, оптимізація роботи SEO та інше.

У ході виконання кваліфікаційної роботи було виконано аналіз предметної області та досліджена швидкодія методів скорочення URL посилань в залежності від різних вхідних даних за допомогою платформи .NET.

Для цього було:

– розглянуто сервіси та методи скорочення посилань;

– проаналізовано та обрано алгоритми для скорочення URL посилань;

– проаналізовано існуючі підходи до вимірювання швидкодії у веб-серверному застосунку;

– визначено можливість впливу алгоритмів скорочення посилань на швидкодію веб-застосунків;

– для оцінювання переробки речень в експерименті, були визначені та використані різноманітні метрики, які дозволили зробити об'єктивну оцінку результатів;

– визначено функціональні вимоги та розроблено веб-серверний застосунок, який було використано в цілях проведення практичної частини експерименту;

– виміряно та підраховано значення обраних метрик для кожного з методів управління станом за допомогою біліотеки BenchmarkDotNet;

– порівняно та проаналізовано отримані дані;

– зроблено висновки відносно описаних результатів дослідження.

Під час дослідження було проведено ряд тестів та вимірювань для оцінки впливу кожного методу скорочення посилань на швидкодію. Отримані результати дозволили зробити висновки про те, який метод дає найбільшу швидкість обробки посилань, а також про те, які чинники впливають на ефективність цих методів.

Отже, для проведення дослідження було використано три основні бібліотеки:

– UrlBase64;
– Shuttle.Core.Defalte;
– SHA-2 з ядра .NET та Base62.

У результаті дослідження можна зробити висновок, що оптимальною бібліотекою для скорочення URL посилань, з точки зору швидкодії, є Url Base64, проте в неї є недоліки відносно колізій при збільшенні кількості вхідних даних. Найкращим рішенням для скорочення посилання показав себе підхід Sha256 з комбінацією алгоритму Base62. Звичайно він показав певні недоліки відносно швидкодії для посилань що мають довжину менше 64 символи, проте надав дуже хороші показники при зростанні даних.

При врахуванні того, що дослідження було проведено з використанням обмеженого функціонально веб-серверного додатку, який був створений спеціально для цього дослідження, не можна зробити однозначних висновків щодо можливих результатів в подібному дослідженні, якщо використовуватиметься більш складний веб-додаток. Різні технічні особливості та обмеження програмного забезпечення можуть призвести до отримання різних результатів в подібних дослідженнях.

У разі потреби швидкодії програми не рекомендовано використовувати Deflate для скорочення посилань, оскільки основна ідея алгоритму має ціль зробити компактні дані, а не перетворити дані у певну послідовність байтів, яку вже неможливо буде перетворити назад у потрібні дані.

Отже можна зробити висновок що використання Sha-256 з комбінацією Base64 або Base62, буде оптимальним варіантом для створення ядра системи скорочення посилань. Даний підхід відносно досліджень має переваги у швидкодії обробки даних, а саме їх безперервне зростання, а також мінімізує виникнення колізій при скорочені URL посилання.

Для того, щоб підтвердити або спростувати отримані результати та висновки, потрібні додаткові дослідження. Для цього можна розглянути використання більш складних веб-додатків з можливостями хешування [23] запитів користувачів.

Необхідно порівняти різні методи та їх комбінації, щоб отримати більш точне уявлення про швидкість та ефективність скорочення посилань.

Також для досліджень можна використовувати різні типи даних, наприклад, зберігати додаткову інформацію про користувачів, які використовують скорочення посилань. Це може допомогти зрозуміти поведінку користувачів та їхні потреби, а також допоможе у поліпшенні функціоналу додатку.

Важливо забезпечувати безпеку при скороченні посилань. Можна використовувати різні методи аутентифікації користувачів та захисту від злома. Наприклад, двохфакторна аутентифікація, використання SSL-шифрування та інших методів захисту допоможуть у запобіганні несанкціонованому доступу до скорочених посилань.

Також, важливо розглянути використання різних мов програмування та фреймворків для розробки додатку скорочення посилань. Це може допомогти у покращенні швидкодії та ефективності роботи додатку.

Кваліфікаційна робота пройшла апробацію у журналі «Біоніка інтелекту», який включено до Переліку наукових фахових видань України, категорія «Б», технічні науки [24], матеріали статті наведені у додатку Г.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is a URL?. URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_URL (дата звернення: 14.01.2023).

2. Smelyakov, K., Chupryna, A., Bohomolov, O., Vakulik, E. Lung X-Ray Images Preprocessing Algorithms for COVID-19 Diagnosing Intelligent Systems CEUR Workshop Proceedings, 2022, 3171, pp. 1233–1250.

3. Yuan, H.; Wun, B.; Crowley, P.; Beckett, et al. (2010). Software-based implementations of updateable data structures for high-speed URL matching. IEEE Xplore. URL: https://ieeexplore.ieee.org/document/5623837(дата звернення: 14.01.2023).

4. Yan, K., Chen, J., Cao, B., Zheng, Y., & Hong, T. (2014). Research on a low conflict flow matching hash algorithm. IEEE Xplore. doi: 10.1049/cp.2013.2017. URL: https://ieeexplore.ieee.org/document/6737829 (дата звернення: 14.01.2023).

5. Shortened URL Security Tips [Електронний ресурс] / Carnegie Mellon University. – URL: https://www.cmu.edu/iso/aware/dont-take-the-bait/shortened-url-security.html (дата звернення: 16.01.2023).

6. What Is An API (Application Programming Interface)? [Електронний ресурс]. URL: https://aws.amazon.com/what-is/api/?nc1=h_ls (дата звернення: 16.01.2022).

7. What Is TinyURL and Should You Use It? [Електронний ресурс]. URL: https://www.makeuseof.com/what-is-tinyurl/ (дата звернення: 16.01.2023).

8. URL Expiration [Електронний ресурс]. URL: expiration.https://help.salesforce.com/s/articleView?id=sf.mc_es_url_expiration.htm&type=5 (дата звернення: 16.02.2023).

9. Wen, S.; Dang, W.; Beckett, D.; Sezer, S.; et al. (2018). Research on Base64 Encoding Algorithm and PHP Implementation. IEEE Xplore. doi: 10.1109/GEOINFORMATICS.2018.8557068. URL: https://ieeexplore.ieee.org/document/8557068.

10. UTF-7.A Mail-Safe Transformation Format of Unicode [Електронний ресурс]. URL: https://www.rfc-editor.org/rfc/rfc2152.html (дата звернення: 18.02.2022).

11. RFC 1521 MIME (Multipurpose Internet Mail Extensions) [Електронний ресурс] / N. Borenstein. URL: https://www.ietf.org/rfc/rfc1521.txt (дата звернення: 18.02.2022).

12. ASCII code: benefits and areas of application [Електронний ресурс] / Rajab Mohammadi – URL: https://www.ionos.com/digitalguide/server/know-how/ascii-codes-overview-of-all-characters-on-the-ascii-table/ (дата звернення: 18.02.2022).

13. The MD5 Message-Digest Algorithm [Електронний ресурс] / R. Rivest // 1992
– URL: https://www.ietf.org/rfc/rfc1321.txt (дата звернення: 18.02.2022).

14. LZ77 Compression Algorithm [Електронний ресурс]. URL - https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-wusp/fb98aa28-5cd7-407f-8869-a6cef1ff1ccb (дата звернення: 15.03.2023).

15. Sharonova, N., Kyrychenko, I., Gruzdo, I., Tereshchenko, G. Generalized Semantic Analysis Algorithm of Natural Language Texts for Various Functional Style Types CEUR Workshop Proceedings, 2022, 3171, pp. 16–26.

16. Smelyakov, K., Honchar, Y., Bohomolov, O., Chupryna, A. Machine Learning Models Efficiency Analysis for Image Classification Problem CEUR Workshop Proceedings, 2022, 3171, pp. 942–959.

17. DEFLATE Compressed Data Format Specification version 1.3 [Електронний ресурс]. URL: https://www.w3.org/Graphics/PNG/RFC-1951 (дата звернення: 21.03.2023).

18. Laatansa, R., Saputra, R., Noranita, B., Yan, K., Chen, J., Cao, B., Zheng, Y., & Hong, T. (2020). Analysis of GPGPU-Based Brute-Force and Dictionary Attack on SHA-1 Password Hash: An Improved Low Conflict Flow Matching Hash Algorithm. IEEE Xplore. doi: 10.1049/cp.2020.8982390. URL: https://ieeexplore.ieee.org/document/8982390.

19. Lee, S.-H., & Shin, K.-W. (2018). An efficient implementation of SHA processor including three hash algorithms (SHA-512, SHA-512/224, SHA-512/256). IEEE International Conference on Computer Communications Workshops (INFOCOM WKSHPS). doi: 10.23919/ELINFOCOM.2018.8330578. URL: https://ieeexplore.ieee.org/document/8330578.

20. Pham H. L., Tran T. H., Le V. T. D., Nakashima Y., & Yan K. (2022). A Coarse Grained Reconfigurable Architecture for SHA-2 Acceleration: An Improved Low Conflict Flow Matching Hash Algorithm. IEEE Xplore. doi: 10.1109/IPDPSW55747.2022.00117. URL: https://ieeexplore.ieee.org/document/9835347.

21. Benchmarking .NET 6 Applications Using BenchmarkDotNet: A Deep Dive [Електронний ресурс] / Joydip Kanjilal, August 31, 2022. URL: https://www.codemag.com/Article/2209061/Benchmarking-.NET-6-Applications-Using-BenchmarkDotNet-A-Deep-Dive (дата звернення: 28.04.2023).

22. ASP.NET MVC Controller Overview (C#) / Stephen Walther, 11/07/2022. URL: https://learn.microsoft.com/en-gb/aspnet/mvc/overview/older-versions-1/controllers-and-routing/aspnet-mvc-controllers-overview-cs (дата звернення: 29.04.2023).

23. Building a simple URL shorten service with Redis [Електронний ресурс] / Henry Chu, 16 Sep 2014. URL: https://www.codeproject.com/Articles/819235/Building-a-simple-URL-shorten-service-with-Redis (дата звернення: 29.042023)

24. Comparative analysis of url shortening algorithms: performance considerations in the application domain of web search and information retrieval technology / I. Kyrychenko, Y. Shamrai// Bionics of Intelligence. – Kharkiv : KhNURE. – 2022. – № 2 (99). – С. 15-22.