Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка методу та програмних засобів вебсервісу керування короткими URL-посиланнями»

Виконав: студент IV курсу
групи ЗПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Коровай В. Г.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Черноволик Г. О.
(прізвище та ініціали)

Рецензент: к.т.н., доцент каф. ОТ Городецька О. С.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ _____
« 9 » червня 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Короваю Володимиру Григоровичу

1. Тема роботи – розробка методу та програмних засобів вебсервісу керування короткими URL-посиланнями.

Керівник роботи: Черноволик Галина Олександрівна, к.т.н., доц. каф. ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: середовище розробки – Visual Studio Code, мови розробки – PHP, JavaScript; фреймворки – Laravel; операційна система – Windows 8/10/11; браузер – на базі двигуна V8 і Chromium 90.

4. Зміст текстової частини: вступ, аналіз стану розвитку систем скорочення URL-посилань та постановка задач розробки, розробка методів, моделі та алгоритму роботи програми, розробка програмного забезпечення, тестування програми, висновки, список використаних джерел, додатки, ілюстративна частина.

5. Перелік ілюстративного матеріалу: титульний слайд – автор, науковий керівник бакалаврської кваліфікаційної роботи, тема; актуальність розробки; мета, об'єкт та предмет дослідження; постановка задачі; наукова новизна; практична цінність; огляд та аналіз аналогів; використані технології; модель роботи системи, загальний алгоритм роботи системи; блок-схема алгоритму роботи модуля скорочення посилань; блок-схема алгоритму роботи масового скорочення URL-посилань; блок-схема алгоритму роботи модуля переходів; діаграма діяльності системи; апробація та публікація результатів роботи; висновки; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Черноволик Г. О. к.т.н., доцент кафедри ПЗ	24.03.25	01.04.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку систем керування короткими URL-посиланнями та постановка задач розробки	25.03.25 – 31.03.25	Виконано
2	Розробка методів, моделі та алгоритмів роботи програми	01.04.25 – 14.04.25	Виконано
3	Розробка програмного забезпечення	15.04.25 – 12.05.25	Виконано
4	Тестування програми	13.05.25 – 15.05.25	Виконано
5	Оформлення матеріалів до захисту БКР	16.05.25 – 30.05.25	Виконано

Студент _____ **Коровай В. Г.**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **Черноволик Г. О.**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота містить 61 сторінку формату А4, на яких наведено 27 рисунки і 4 таблиці. Список використаних джерел складається з 23 найменувань.

У бакалаврській кваліфікаційній роботі проведено аналіз сучасного стану та тенденцій розвитку вебсервісів для керування короткими URL-посиланнями. У межах дослідження розглянуто існуючі програмні рішення, визначено їхні основні функціональні можливості, переваги та недоліки. На основі аналізу сформульовано вимоги до системи та обґрунтовано доцільність розробки власного вебзастосунку, орієнтованого на централізоване управління великою кількістю посилань.

Виконано обґрунтування вибору технологічного стеку, що включає мову програмування PHP, JavaScript, фреймворк Laravel для реалізації серверної логіки, а також jQuery і Bootstrap для побудови адаптивного та інтерактивного клієнтського інтерфейсу. У якості бази даних використано MySQL. У якості середовища розробки використано Visual Studio Code.

Отримані результати можуть бути використані для підвищення ефективності роботи з великими обсягами URL у сферах цифрового маркетингу, адміністрування контенту та обміну інформацією в інтернет-середовищі.

Ключові слова: вебсервіс, скорочення посилань, Laravel, PHP, JavaScript, аналітика, система керування URL.

ABSTRACT

The bachelor's thesis contains 61 pages of A4 format, which contain 27 figures and 4 tables. The list of used sources consists of 23 items.

The bachelor's thesis analyzes the current state and trends in the development of web services for managing short URL links. The study examines existing software solutions, identifies their main functionalities, advantages and disadvantages. Based on the analysis, the requirements for the system are formulated and the feasibility of developing an in-house web application focused on the centralized management of a large number of links is substantiated.

The choice of a technological stack, including the PHP programming language, JavaScript, the Laravel framework for implementing server logic, as well as jQuery and Bootstrap for building an adaptive and interactive client interface, is justified. MySQL is used as a database. Visual Studio Code is used as a development environment.

The obtained results can be used to increase the efficiency of working with large volumes of URLs in the areas of digital marketing, content administration, and information exchange in the Internet environment.

Keywords: web service, link shortening, Laravel, PHP, JavaScript, analytics, URL management system.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасні веб-технології відіграють визначальну роль у забезпеченні зручності та швидкості обміну інформацією в інтернеті. Зокрема, сервіси скорочення URL-посилань дозволяють оптимізувати представлення довгих та складних адрес, підвищуючи читабельність, спрощуючи поширення контенту в соціальних мережах і месенджерах, а також забезпечуючи аналітику переходів за короткими лінками [1]. Попит на подібні рішення зростає разом із розвитком цифрового маркетингу та мобільного інтернету, що зумовлює необхідність створення більш гнучких, безпечних та масштабованих платформ.

Розробка програмних модулів вебсистеми керування короткими URL-посиланнями є комплексним завданням, яке вимагає використання сучасних фреймворків, а також інтеграції з реляційними базами даних для зберігання інформації про створені посилання, статистику переходів і налаштування користувачів. Важливою складовою є вибір архітектурного підходу – моноліт чи мікросервіси – для забезпечення гнучкості при розширенні функціоналу та підтримці стабільності роботи системи [2].

Однією з ключових проблем у даній сфері є гарантування високої швидкості обробки запитів на перенаправлення та збору аналітичних даних без зниження продуктивності під час пікових навантажень. Необхідно передбачити ефективні алгоритми генерації унікальних і безпечних коротких ідентифікаторів, запобігання колізіям, а також захист від зловмисної активності, таких як спам-боти та DDoS-атаки [3].

Враховуючи зазначені проблеми, розробка власного вебсервісу з підтримкою адаптивного інтерфейсу, кросбраузерності та широких функціональних можливостей є актуальним і доцільним завданням.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є покращення процесу роботи зі скороченими посиланнями за рахунок впровадження розробленої вебсистеми, яка забезпечить користувачів зручним інструментом для керування URL-адресами та аналізу трафіку.

Відповідно до поставленої мети потрібно вирішити такі завдання:

– провести аналіз інформаційного забезпечення та існуючих сервісів скорочення URL;

– розробити метод, модель та алгоритм роботи системи;

– розробити модуль реєстрації та автентифікації для користувачів;

– розробити модуль генерації унікальних коротких посилань з автоматичною та кастомною ідентифікацією;

– розробити модуль масового скорочення URL за допомогою завантаження файлів;

– розробити модуль збору аналітики та статистики переходів;

– розробити механізм захисту посилань паролем та перевірки їхньої безпечності;

– розробити функціонал інтернаціоналізації і локалізації.

Об’єктом дослідження є процес розробки програмних модулів вебсистеми керування короткими URL-посиланнями.

Предметом дослідження є методи та інструменти, що використовуються для розробки програмних модулів, які забезпечують генерацію унікальних ідентифікаторів, організацію механізмів перенаправлення й збору аналітики.

Методи дослідження. У процесі дослідження використовувалися такі методи:

– методи розробки вебзастосунків на базі фреймворку Laravel для реалізації серверної логіки скорочення URL та забезпечення безпеки обробки запитів;

– методи проєктування та реалізації інтерактивного й адаптивного користувацького інтерфейсу;

– методи тестування для перевірки розроблених модулів вебсистеми керування короткими URL-посиланнями;

– методи проєктування та організації баз даних для збереження та обробки даних користувачів.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод генерації коротких URL-посилань, який, на відміну від існуючих рішень, включає механізми масового скорочення посилань з використанням завантаження файлів у форматі Excel та автоматичну перевірку посилань на безпечність через Google Safe Browsing API, що забезпечує підвищену ефективність роботи, гнучкість і додатковий захист користувачів від потенційно шкідливого контенту.

2. Подальшого розвитку отримав метод збору та візуалізації детальної аналітики переходів за короткими посиланнями, який, на відміну від існуючих методів, включає фіксацію додаткових метаданих (IP-адресу, тип браузера, пристрій, географічне розташування тощо), що дозволяє користувачам здійснювати глибокий аналіз ефективності поширення контенту, оптимізувати маркетингові кампанії та забезпечувати ефективне керування цифровими ресурсами.

Практичне значення проєкту полягає у можливості використання розробленого вебсервісу для організації ефективного скорочення та аналізу URL-посилань у різних сферах: цифровому маркетингу, корпоративних порталах, дистанційному навчанні, SMM-кампанії тощо.

Програмні модулі можуть бути адаптовані під потреби різних користувачів і систем — від внутрішніх сервісів компаній до публічних платформ. Також вебсервіс підтримує інтеграцію з іншими програмними рішеннями, що дозволяє

розширити його функціонал та підвищити ефективність взаємодії з кінцевими користувачами.

Особистий внесок здобувача. Усі результати, подані в бакалаврській кваліфікаційній роботі, були отримані автором особисто. У науковій праці [4], опублікованій у співавторстві, автору належать такі результати: таблиця порівняння функціональних характеристик аналогів, порівняння з аналогами, перелік розроблених функцій.

Апробація результатів роботи. Результати роботи доповідалися на Міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи 2025».

Публікації. Результати дослідження були опубліковані у матеріалах Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи 2025» [4].

Аналіз. У пояснювальній записці до бакалаврської кваліфікаційної роботи було розглянуто чотири розділи та було використано 23 літературних джерела.

1 АНАЛІЗ СТАНУ РОЗВИТКУ СИСТЕМ КЕРУВАННЯ КОРОТКИМИ URL-ПОСИЛАННЯМИ ТА ПОСТАНОВКА ЗАДАЧ РОЗРОБКИ

1.1 Аналіз предметної області

У сучасному світі короткі URL-посилання відіграють важливу роль у цифровому просторі, забезпечуючи користувачам зручність та ефективність при передачі посилань на різноманітні інформаційні ресурси. Зі зростанням популярності соціальних мереж, месенджерів та інших платформ онлайн-комунікації, використання коротких URL стало необхідним засобом оптимізації цифрових взаємодій.

Системи керування короткими URL-посиланнями не лише спрощують посилання, роблячи їх більш привабливими та зручними для поширення, але й забезпечують додатковий функціонал, такий як статистика переходів, управління часом дії посилань, налаштування доступу за певними критеріями тощо.

Серед найпопулярніших систем скорочення URL на сьогодні варто виділити сервіси такі як Bitly [5], TinyURL [6] та Rebrandly [7]. Кожен з цих сервісів пропонує базові функції скорочення та відслідковування посилань, однак також має свої особливості. Bitly, наприклад, надає детальну аналітику і можливість кастомізації посилань, тоді як Rebrandly орієнтований на брендування URL, що важливо для маркетингових кампаній та брендової ідентичності компаній.

Проте, незважаючи на широке використання цих сервісів, існують певні обмеження, які викликають потребу в розробці нових, більш гнучких і безпечних рішень. Основними недоліками існуючих платформ є обмеження у функціональності безкоштовних версій, недостатня безпека і приватність, а також обмежена можливість інтеграції з іншими програмними засобами.

Таким чином, актуальним завданням є розробка вебсистеми управління короткими URL, яка не лише забезпечуватиме основні функції скорочення посилань, але й пропонуватиме розширені можливості, такі як управління

доступом, глибока аналітика переходів, інтеграція з популярними вебсервісами та забезпечення високого рівня безпеки і конфіденційності даних користувачів.

1.2 Аналіз аналогів розроблюваного програмного застосунку

З метою визначення основних функціональних особливостей майбутньої системи керування короткими URL-посиланнями було здійснено порівняльний аналіз найпопулярніших аналогів: Bitly, TinyURL та Rebrandly.

Bitly – один із найбільш відомих сервісів для скорочення посилань, що користується популярністю завдяки широкому спектру функцій (див. рисунок 1.1). Він дозволяє створювати персоналізовані короткі URL, має вбудовані потужні інструменти аналітики для відстеження переходів за посиланнями, які включають географічний розподіл користувачів, джерела трафіку та інші корисні метрики. Проте, безкоштовна версія цього сервісу має суттєві обмеження щодо кількості посилань та глибини доступної аналітики, що ускладнює використання в комерційних масштабах. Bitly також має розширені можливості інтеграції зі сторонніми сервісами, що робить його популярним серед маркетологів і бізнес-користувачів.

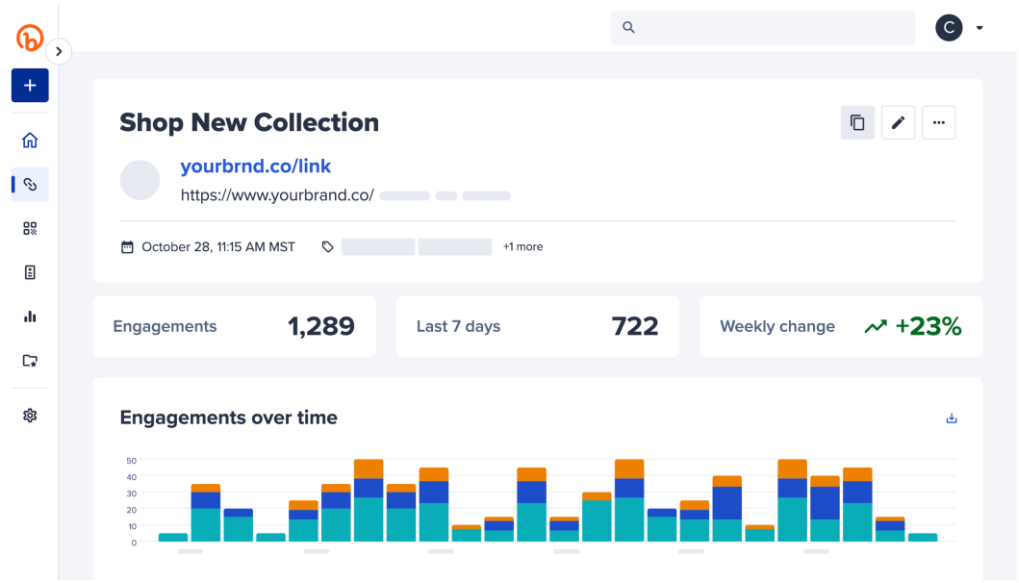

Рисунок 1.1 – Приклад скороченого посилання Bitly

TinyURL – простий та зручний сервіс скорочення URL, що не вимагає попередньої реєстрації і надає можливість створювати короткі посилання без термінів дії (див. рисунок 1.2). Його перевагою є швидкість та простота використання, що робить його популярним серед звичайних користувачів. Однак він не має можливостей для аналітики, що значно обмежує його застосування в бізнес-середовищі, де важливо відслідковувати ефективність посилань. Відсутність API для інтеграції також обмежує його функціональні можливості для корпоративних потреб.

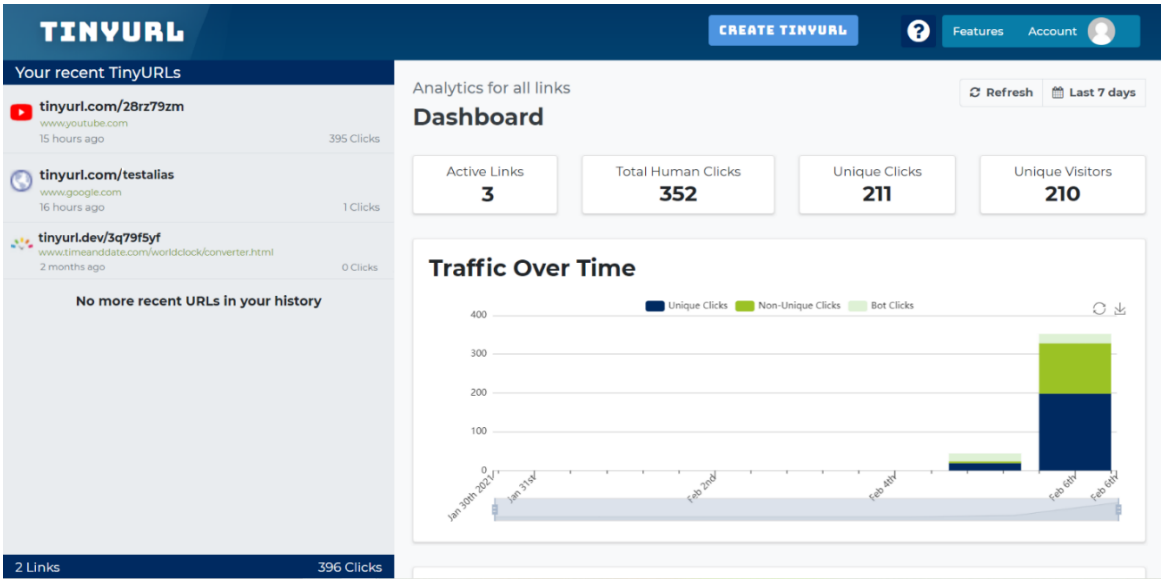

Рисунок 1.2 – Приклад використання панелі управління в TinyURL

Rebrandly – сервіс, який спеціалізується на створенні брендових коротких URL (див. рисунок 1.3). Він дозволяє використовувати власні домени, що є вагомою перевагою для компаній, які прагнуть підвищити впізнаваність свого бренду. Rebrandly надає розширені аналітичні можливості, включаючи детальний аналіз переходів за посиланнями, можливість встановлення термінів дії посилань та інтеграцію з популярними маркетинговими платформами. Однак більшість цих функцій доступна лише у платних тарифах, що може стати фінансовим бар'єром для малих підприємств або приватних користувачів.

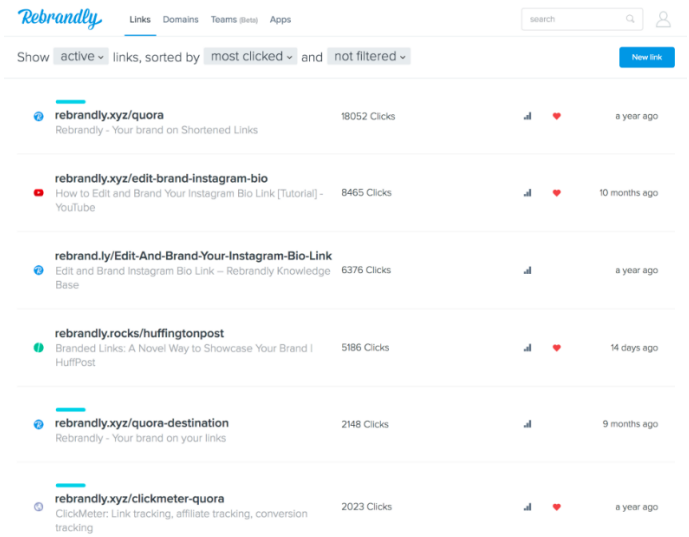

Рисунок 1.3 – Приклад використання панелі управління в Rebrandly

Для наочного порівняння можливостей вказаних сервісів та власної розробки було сформовано порівняльну таблицю (див. таблицю 1.1).

Таблиця 1.1 – Порівняльний аналіз аналогів вебсервісів керування короткими URL-посиланнями

Функціонал	Bitly	TinyURL	Rebrandly	Власна розробка
Базове скорочення посилань	+	+	+	+
Відстеження переходів	+	-	+	+
Масове скорочення посилань	+	+	-	+
API для інтеграції	+	+	+	+
Налаштування терміну дії посилань	-	-	+	+
Захист посилань паролем	-	-	+	+
Сумарний коефіцієнт	4	3	5	6

Аналізуючи дані таблиці 1.1, можна відзначити, що власна розробка демонструє на 50% кращі результати, ніж TinyURL (100% - 3/6 × 100% = 50%), на 33% ефективніша за Bitly (100% - 4/6 × 100% = 33%), і має 20% перевагу над Rebrandly (100% - 5/6 × 100% = 17%). Це підкреслює суттєві конкурентні переваги розроблюваної системи та її високий потенціал на ринку цифрових технологій.

Таким чином, розробка власної вебсистеми для керування короткими URL-посиланнями має значні переваги порівняно з існуючими аналогами. Запропонована система включатиме всі ключові функції конкурентів, при цьому забезпечуючи унікальні додаткові можливості, яких бракує в популярних сервісах. Зокрема, це налаштування терміну дії посилань, захист за паролем, підтримка масового скорочення та повноцінна інтеграція зі сторонніми сервісами через API. Такий підхід робить власну розробку універсальним інструментом як для пересічного користувача, так і для бізнесу.

1.3 Аналіз методів розв’язання задачі

Побудова вебсистеми для скорочення URL-посилань передбачає розв’язання ряду задач, пов’язаних із генерацією коротких адрес, зберіганням інформації про переходи, безпекою та зручністю взаємодії з інтерфейсом. Для цього необхідно обрати такі технічні рішення, які забезпечать оптимальний баланс між продуктивністю, простотою реалізації та можливістю подальшого розширення функціоналу.

У якості технологічної основи обрано архітектуру клієнт-сервер із використанням мови PHP і фреймворку Laravel на серверній стороні. Laravel надає зручні засоби для маршрутизації, валідації, організації контролерів і роботи з базами даних [8]. Для реалізації клієнтської частини буде використано jQuery і Bootstrap. jQuery забезпечить просту і швидку обробку подій, AJAX-запити [9], а Bootstrap – адаптивне оформлення інтерфейсу без потреби ручного налаштування стилів [10].

Одним з ключових викликів є генерація унікальних коротких ідентифікаторів для URL. Найпростішим і надійним методом є використання автозбільшуваного числового ID у базі даних із подальшим перетворенням у символьний рядок – наприклад, за допомогою хеш-функцій або кодування чисел у власну алфавітну систему. Такий підхід дозволяє уникнути колізій та зберігати просту структуру для зворотного розшифрування коротких посилань у первинні адреси.

Для зберігання даних буде використано реляційну базу даних MySQL, яка дозволяє зручно оперувати зв’язками між таблицями: короткими посиланнями, статистикою переходів, користувачами. Такий вибір спрощує побудову аналітики та звітності, а також забезпечує можливість подальшої оптимізації за допомогою індексів або кешування [11].

Важливою складовою є реалізація збирання статистики переходів – часу, IP-адресу, реферера, браузера, пристрою. Ця інформація дозволить користувачам відстежувати ефективність використання посилань. Збір даних буде реалізовано на серверній частині сайту при кожному запиті до короткого посилання перед перенаправленням.

Інтерфейс користувача повинен бути простим і зручним, що досягається завдяки використанню Bootstrap для стилізації сторінок та jQuery для реалізації взаємодії без перезавантаження – наприклад, обробка форм, динамічне оновлення статистики або підтверджень.

Також слід враховувати питання безпеки: система повинна захищати користувачів від фішингових посилань, автоматично відфільтровуючи шкідливий контент за допомогою Google Safe Browsing.

Загалом, обрані методи дозволяють створити систему, яка буде не лише функціональною, але й гнучкою в обслуговуванні та масштабуванні для подальшого вдосконалення системи й додавання нових можливостей. Простота використання jQuery і Bootstrap дає змогу зосередитися на логіці роботи, а Laravel забезпечує потужну серверну основу для реалізації бізнес-логіки.

1.4 Постановка задач дослідження

На основі проведеного аналізу аналогів, методів розв’язання задачі та визначених функціональних вимог, формулюються основні задачі бакалаврської кваліфікаційної роботи. Вони охоплюють як технічну, так і функціональну частину системи, спрямовану на створення ефективного інструменту для скорочення, керування та аналітики URL-посилань.

Перед системою ставляться наступні задачі:

– реалізувати генерацію коротких посилань із гарантією унікальності та можливістю обробки великої кількості запитів;

– створити механізм збереження асоціацій між короткими й оригінальними URL у базі даних із можливістю масштабування;

– забезпечити підтримку функції масового скорочення посилань, що особливо актуально для бізнес-користувачів та маркетингових кампаній;

– реалізувати збір детальної статистики щодо використання коротких URL з подальшою можливістю перегляду;

– впровадити обмеження терміну дії коротких посилань і механізм їх автоматичного видалення після завершення терміну;

– додати функціонал захисту посилань паролем для обмеження доступу до вмісту;

– розробити інтерфейс користувача, що забезпечить простоту роботи з системою та адаптивність до мобільних пристроїв;

– забезпечити захищену обробку запитів, валідацію вхідних даних та обмеження зловмисних дій.

Усі ці задачі спрямовані на створення повноцінної вебсистеми, яка дозволяє не лише комфортно працювати з URL-посиланнями, а й гарантує високий рівень інформаційної безпеки, зручності використання і гнучкого адміністрування для кінцевого користувача.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі бакалаврської кваліфікаційної роботи було проведено комплексний аналіз предметної області, пов’язаної з розробкою вебсистем для керування короткими URL-посиланнями. Розглянуто сучасний стан технологій у цій сфері, проаналізовано можливості існуючих сервісів, таких як Bitly, TinyURL та Rebrandly, і визначено їхні сильні сторони та обмеження.

Порівняльний аналіз показав, що власна розробка здатна перевершити популярні аналоги завдяки розширеному функціоналу, який включає масове скорочення, захист посилань паролем, налаштування терміну дії та повноцінну статистику переходів. У результаті, було визначено набір функцій, які роблять запропоновану систему конкурентоспроможною та зручною для різних категорій користувачів.

Окрему увагу приділено методам реалізації системи. На основі проведеного аналізу обрано оптимальний підхід: використання Laravel для серверної частини, а також jQuery і Bootstrap для реалізації динамічного і адаптивного інтерфейсу. Така архітектура дозволяє досягти гнучкості, безпеки та масштабованості проєкту.

Було описано задачі бакалаврської роботи, які охоплюють реалізацію генератора коротких URL, розробку модуля збирання статистики, побудову API та створення зручного інтерфейсу. Усі ці завдання є необхідними для створення надійного і функціонального вебсервісу, що може бути використаний як у персональному, так і в комерційному середовищі.

2 РОЗРОБКА МЕТОДІВ, МОДЕЛІ ТА АЛГОРИТМУ РОБОТИ СИСТЕМИ

2.1 Аналіз даних

Обробка та передача інформації є критично важливими аспектами роботи вебсервісу для керування короткими URL-посиланнями. Усі модулі системи взаємодіють між собою через клієнт-серверну архітектуру, де дані передаються по захищеному протоколу, що гарантує безпечну роботу з персональною інформацією користувачів та ефективне виконання основних функцій сервісу.

Модуль авторизації та реєстрації забезпечує створення та ідентифікацію користувачів. Вхідними даними є особиста інформація, така як ім’я, електронна пошта та пароль. У разі успішної авторизації користувачу відкривається доступ до функцій системи: створення посилань, перегляд статистики, масове скорочення та керування профілем. Вихідними даними є графічний інтерфейс із доступом до кабінету користувача та відповідних можливостей.

Модуль скорочення посилань відповідає за створення унікальних коротких URL. Користувач вводить оригінальну URL-адресу та, за потреби, задає додаткові параметри — бажаний ідентифікатор, термін дії, пароль. На сервері здійснюється перевірка безпечності посилання через Google Safe Browsing API, а також генерується або перевіряється унікальність ідентифікатора. Вихідними даними є коротке посилання, яке може бути скопійоване користувачем, а також збережене в базі даних для подальшого відстеження переходів.

Модуль масового скорочення дає змогу обробляти десятки або сотні посилань одночасно. Вхідними даними є файл у форматі Excel, що містить перелік URL-адрес. Після обробки кожне посилання проходить валідацію та скорочується із застосуванням тих самих правил, що й у звичайному режимі. Вихідними даними є новий Excel-файл зі списком створених коротких посилань, який користувач може одразу завантажити. Цей модуль значно спрощує роботу для маркетологів, адміністраторів і корпоративних користувачів.

Модуль переходів виконує перенаправлення користувачів, які переходять за короткими посиланнями. При запиті система перевіряє термін дії та наявність пароля (якщо такий заданий), після чого виконує редирект на оригінальну адресу. Вхідними даними є короткий код із URL-запиту. Вихідними — переадресації на відповідне оригінальне посилання, сторінка для введення пароля чи повідомлення про недійсність посилання.

Модуль збору статистики фіксує кожен перехід за коротким посиланням. Збираються такі метадані: IP-адреса, браузер, пристрій, країна, реферер та час кліку. Ці дані зберігаються в базі даних та використовуються для побудови аналітики. Користувач має можливість переглядати статистику по кожному посиланню у вигляді таблиць, графіків та звітів. Вихідними даними є інтерфейс перегляду статистичних даних з можливістю фільтрації, сортування та експорту.

Усі описані модулі взаємодіють через API, обробка даних здійснюється асинхронно, що забезпечує плавність та швидкодію системи навіть за високого навантаження. Це дає змогу ефективно керувати великою кількістю посилань, аналізувати трафік і підтримувати зручний користувацький досвід.

2.2 Розробка методу генерації коротких URL-посилань

У межах розробки вебсервісу було створено спеціальний метод генерації коротких URL-посилань, орієнтований передусім на масове скорочення великих обсягів адрес із урахуванням безпеки й коректності. Головна задача полягала в тому, щоб забезпечити автоматизовану обробку сотень URL-посилань за один цикл, водночас фільтруючи некоректні або потенційно небезпечні ресурси, та генерувати унікальні короткі коди без колізій. При простому послідовному обході списку URL без «проміжних» перевірок ризик перенавантаження системи або потрапляння шкідливих посилань у базу значно зростає.

Загальну структуру та послідовність реалізації цього підходу ілюструє блок-схема на рисунку 2.1.

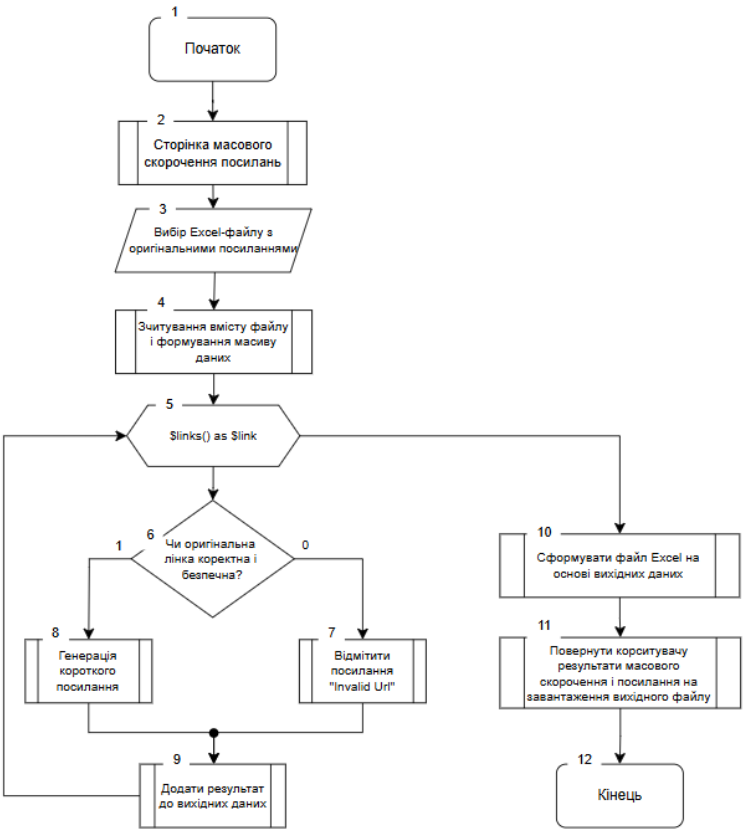

Рисунок 2.1 – Блок-схема алгоритму масового скорочення URL-посилань

1. Спершу користувач завантажує Excel-файл з оригінальними URL-адресами. На сервері файл одразу передається в модуль обробки, де бібліотека SimpleXLSX перетворює його в масив рядків. Далі кожна адреса по черзі проходить швидку перевірку синтаксису, щоб відсікти очевидно некоректні записи.

2. URL-адреси, що мають правильний формат, додатково аналізуються через Google Safe Browsing API. Такий подвійний фільтр дає змогу одразу виявляти потенційно небезпечні ресурси. Якщо посилання позначено сервісом Google як ризиковане, воно фіксується у вихідному наборі даних із поміткою «Invalid URL».

3. Безпечні й коректні адреси передаються в генератор коротких кодів. Сервіс створює випадковий алфавітно-цифровий рядок сталої довжини й перевіряє його унікальність. У разі колізії код генерується повторно. Після успішної генерації відбувається запис пари «оригінальний URL – короткий код» у вихідний масив.

4. Коли весь список оброблено, формується новий Excel-файл: у першому стовпці залишаються початкові оригінальні адреси, у другому — відповідні короткі посилання або відмітка «Invalid URL». Файл тимчасово зберігається на сервері, а у відповідь користувач отримує JSON-повідомлення з кількістю успішно скорочених записів та посиланням для завантаження результату.

5. Завершальний етап — прибирання тимчасових ресурсів. Планувальник Laravel видаляє згенерований файл після закінчення заданого часу зберігання, що підтримує порядок у файловій системі та усуває зайві витрати на дисковий простір.

Такий підхід поєднує перевірку безпеки, унікальну генерацію кодів і автоматичне формування звіту, забезпечуючи швидку та надійну обробку сотень URL-адрес одним запитом.

2.3 Розробка методу збору та візуалізації детальної аналітики переходів за короткими URL-посиланнями

На відміну від типових рішень, де зберігається лише лічильник кліків і кілька типових запропонований підхід фіксує повний набір метаданих: IP-адресу, реферер, рядок User-Agent, визначені з нього браузер і тип пристрою, а також країну, отриману через GeoLite2. У результаті користувач отримує не просто цифру «скільки переходів», а багатовимірний зріз аудиторії, що дозволяє глибоко аналізувати джерела трафіку й своєчасно оптимізувати рекламні активності.

Метод збору та візуалізації детальної аналітики переходів охоплює кілька ключових етапів:

1. Перш за все, у момент запиту до короткого коду викликається метод recordClick(). Він одержує IP-адресу, рядок User-Agent і реферер із заголовків HTTP-запиту, після чого визначає назву браузера та тип пристрою за допомогою бібліотеки Jenssegers Agent.

2. Через локальну базу GeoLite2-City.mmdb (MaxMind) виконується запит методу city(), який повертає код країни. У випадку збою ловиться виняток та

зберігається значення Unknown, а додатково логуються деталі помилки, що спрощує моніторинг працездатності геосервісу.

3. У таблицю clicks заносяться дата та час переходу, реферер, IP-адреса, повний User-Agent, визначені браузер, пристрій і країна. Така структура дає змогу формувати зрізи за вказаним критерієм.

4. На клієнтській частині при відкриті детальної інформації про коротке посилання за допомогою Chart.js відображається збережена статистика за цим посиланням у вигляді лінійних графіків (динаміка кліків), діаграм-пирогів (частка браузерів та типи пристроїв), а також списку IP-адрес та рефералів. Користувач може перемикати часові інтервали, фільтрувати дані за конкретним коротким кодом або кампанією та одразу бачити оновлені графіки.

Поєднання розширеної фіксації даних із інтерактивною візуалізацією надає користувачам інструмент глибокої аналітики: можна швидко визначати найефективніші канали розповсюдження, виявляти підозрілий трафік та адаптувати контент під конкретні аудиторії. Усе це робить вебсервіс повноцінною платформою для керування цифровими ресурсами та маркетинговою оптимізацією.

2.4 Розробка методу побудови вікон застосунку

Для реалізації графічного інтерфейсу користувача у вебзастосунку скорочення URL-посилань на базі Laravel та Blade-шаблонів було розроблено метод побудови вікон застосунку. Laravel – популярний фреймворк PHP, який забезпечує зручну архітектуру MVC, а Blade – вбудований шаблонізатор, що дозволяє створювати динамічні та адаптивні HTML-шаблони з високою гнучкістю.

Метод побудови вікон застосунку охоплює кілька ключових етапів:

1. Вхідною точкою застосунку є маршрут web.php, що визначає доступ до кожного представлення. Для кожної сторінки, зокрема сторінки створення або редагування посилань, реалізовані відповідні маршрути та методи контролерів. Базовий макет layouts/app.blade.php задає загальну структуру інтерфейсу.

2. Інтерфейс користувача побудовано з використанням Blade-шаблонів, які містять динамічні секції @section, що підключаються до основного шаблону за допомогою @extends. Це дозволяє централізовано налаштовувати заголовки, стилі, навігаційні панелі тощо.

3. Усі шаблони використовують принцип розділення на частини: повторювані елементи інтерфейсу (навігація, футер, повідомлення про статус) винесені у partials/. Це дозволяє забезпечити централізоване оновлення вигляду та спрощує підтримку інтерфейсу.

4. Глобальні стилі та компоненти інтерфейсу реалізовані за допомогою Bootstrap 5 та кастомних SCSS-файлів. Компоненти, такі як форми, кнопки, таблиці, адаптовані до різних розмірів екранів. Окремі елементи (наприклад, модальні вікна чи індикатори статусу) реалізовані як часткові шаблони та підключаються через @include.

5. Форма скорочення посилань є головним вікном застосунку для неавторизованих користувачів. Вона розташована на головній сторінці / і дозволяє створити коротке посилання без реєстрації. Обробка запиту виконується через контролер LinkController, що повертає результат у тому ж вікні.

6. Навігація між сторінками реалізована за допомогою системи маршрутизації Laravel. При програмному переході між сторінками користуються стандартними маршрутами із middleware auth, що забезпечує захист від несанкціонованого доступу.

7. Динамічний контент, зокрема форма масового завантаження, оновлення таблиць статистики, реалізований за допомогою JavaScript і бібліотек jQuery, Dropzone та DataTables, які забезпечують інтерактивність та зручність користування.

8. Для локалізації інтерфейсу використовується механізм @lang() і файли локалізації, що зберігаються у папці resources/lang. Поточна мова задається через middleware або вибирається вручну користувачем у профілі.

2.5 Розробка моделі системи

Для кращого розуміння архітектури розроблюваного застосунка побудовано модель системи у вигляді діаграми класів (див. рисунок 2.2). Вона відображає основні компоненти системи скорочення URL-посилань, їхні атрибути, методи та зв’язки між ними, що дозволяє структуровано уявити логіку взаємодії частин застосунку. Така модель є важливою складовою процесу проєктування, оскільки на її основі можна оптимізувати архітектуру системи, спростити розширення функціоналу та полегшити супровід коду в майбутньому.

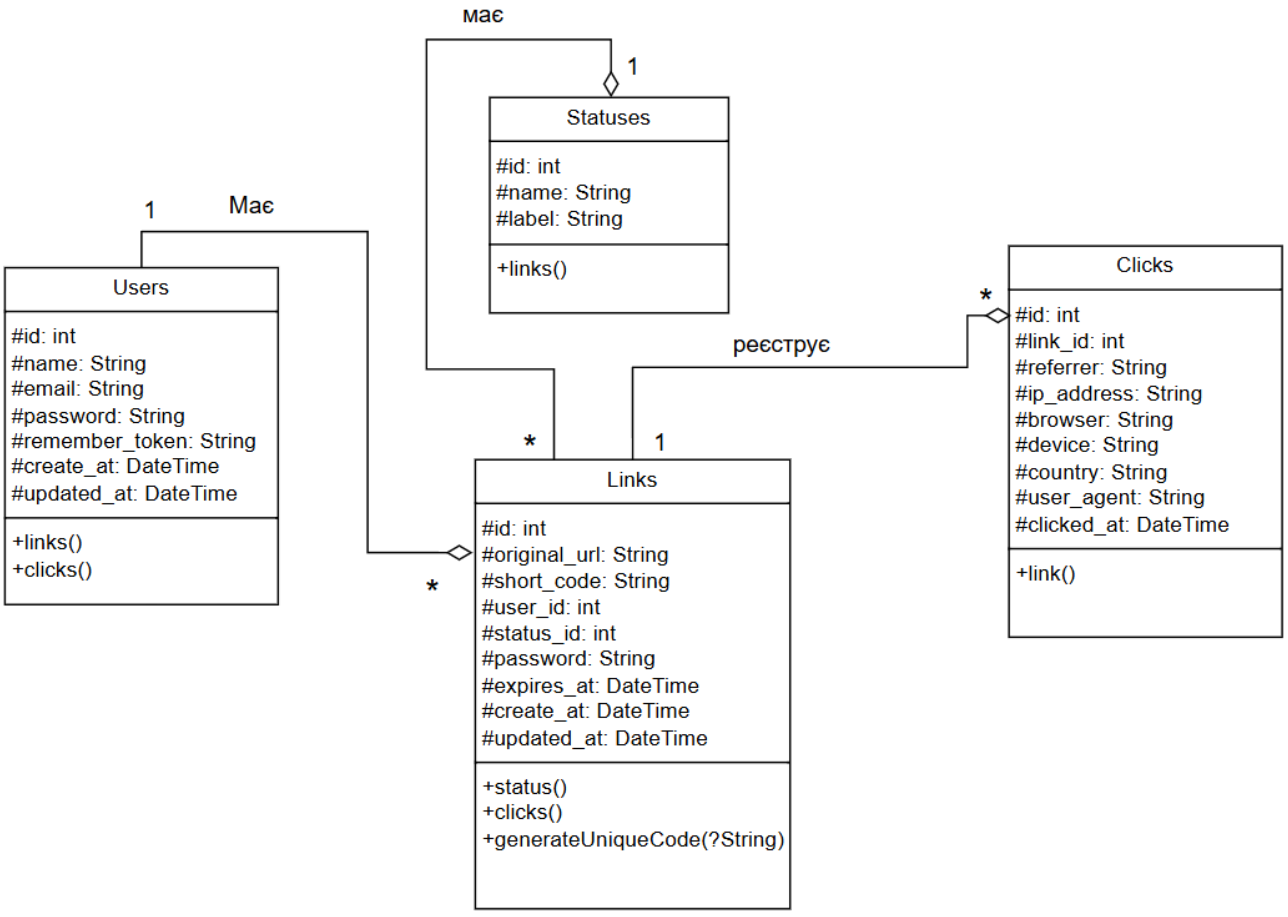

Рисунок 2.2 – Модель роботи системи з використанням діаграми класів

Основним класом системи є Users, який відповідає за зберігання даних про зареєстрованих користувачів вебсервісу. Атрибути цього класу включають id

(унікальний ідентифікатор користувача), name (ім’я), email (електронна адреса), password (зашифрований пароль), а також службові поля remember_token, created_at та updated_at.

Клас Links є центральною сутністю системи і відповідає за зберігання даних скорочених посилань. Атрибути включають original_url (повна URL-адреса), short_code (генерований унікальний ідентифікатор посилання), user_id (ідентифікатор автора), status_id (ідентифікатор статусу), password (опціональний пароль на посилання), expires_at (термін дії) та службову інформацію про дату створення і оновлення. Клас також містить метод generateUniqueCode(?String), який використовується для генерації унікального коду короткого посилання.

Клас Statuses реалізує перелік можливих статусів для посилань — наприклад, активне, тимчасово вимкнене або видалене. До його атрибутів належать id, name та label.

Клас Clicks зберігає інформацію про всі переходи за скороченими посиланнями. Його поля включають referrer (джерело переходу), ip_address (IP-адреса користувача), browser, device, country, user_agent, clicked_at (дата і час кліку), а також link_id — зв’язок із посиланням, за яким було здійснено перехід. Це дозволяє збирати детальну аналітику для кожного короткого URL.

Зв’язки між класами реалізовано згідно з принципами реляційного моделювання:

- Один користувач (Users) може створити багато посилань (Links) та мати багато переходів (Clicks);
- Одне посилання (Links) може бути пов'язане з багатьма переходами (Clicks) і має лише один статус (Statuses);
- Кожен запис про перехід (Clicks) належить одному конкретному посиланню.

Діаграма класів відображає логіку взаємозв’язку об’єктів у системі скорочення посилань, демонструючи, як дані проходять шлях від створення

короткого посилання до відстеження статистики його використання. Це забезпечує наочне уявлення про архітектуру системи, полегшує впровадження нових можливостей та сприяє забезпеченню цілісності даних.

2.6 Розробка загального алгоритму роботи системи та діаграм станів

Основний алгоритм застосунку служить фундаментом розробки вебсервісу. Він демонструє загальну логіку функціонування системи та взаємодію між її основними модулями (див рисунок 2.3).

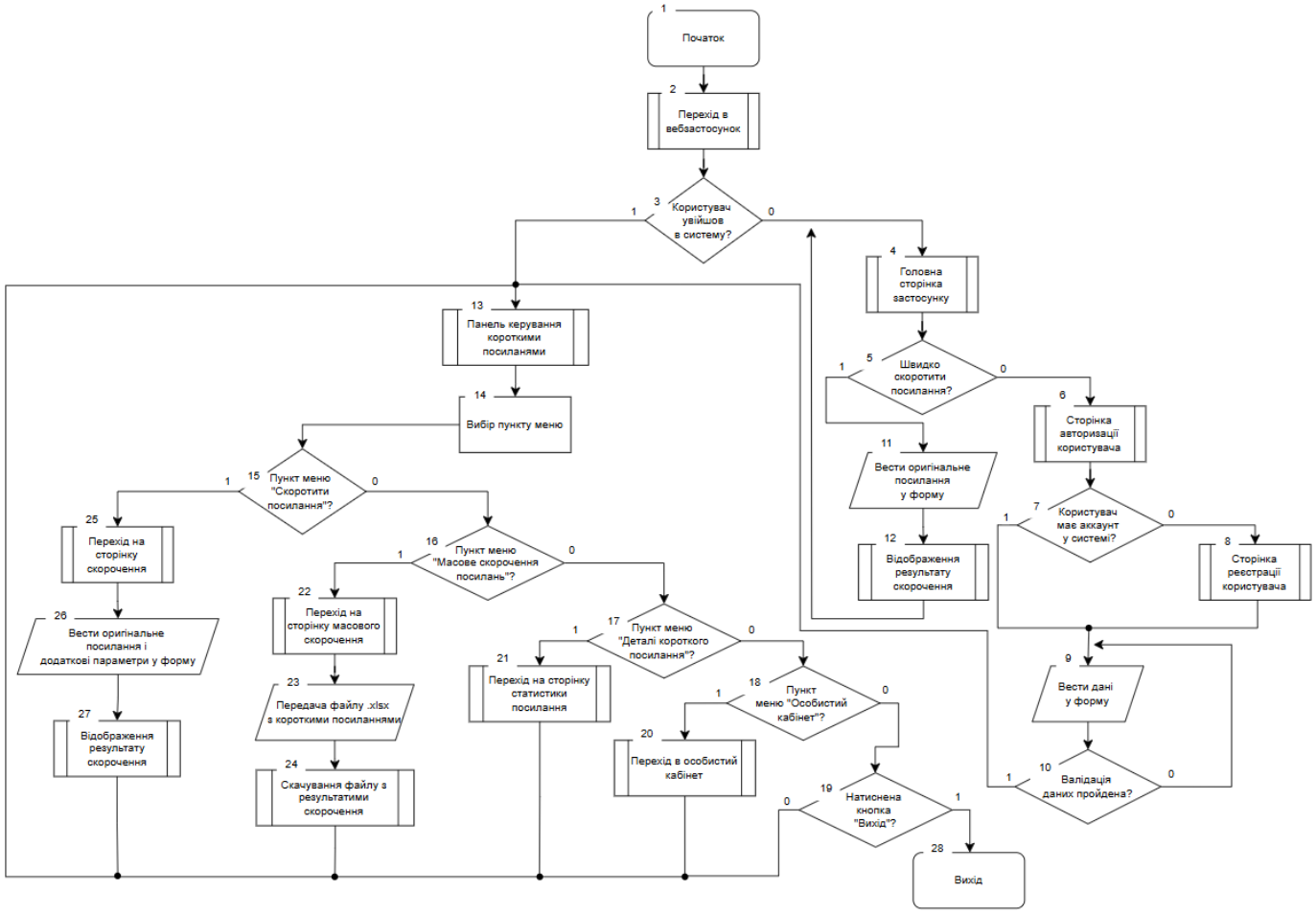

Рисунок 2.3 – Загальний алгоритм роботи системи

Після запуску програми користувач перевіряється на наявність активної сесії. Якщо система не виявляє автентифікованого користувача, відображається

інтерфейс переходу до сторінки входу або реєстрації. У випадку наявності облікового запису користувач вводить свої дані, які проходять валідацію. У разі помилки система повертає відповідне повідомлення з пропозицією повторити спробу. Після успішного входу відкривається головна панель, яка містить меню доступу до основних функцій застосунку.

Залежно від обраного пункту меню, система реагує відповідним чином: відкриває інтерфейс скорочення одного або кількох посилань, завантаження аналітичної інформації, статистики переходів або профілю користувача. У разі створення короткого посилання система перевіряє введену адресу на відповідність стандартам, формує унікальний код та зберігає запис у базі даних. Якщо користувач обирає масове скорочення, система дозволяє завантажити файл у форматі .xlsx, обробляє його вміст і створює відповідні записи для кожного з URL-адрес. Також передбачена можливість переходу до особистого кабінету або завершення сеансу, що супроводжується поверненням до головної сторінки.

Діаграми станів [12] є важливим інструментом моделювання, що дає змогу наочно відобразити поведінку системи шляхом фіксації її можливих станів і переходів між ними. Вони демонструють, у якому саме стані перебуває об’єкт на певному етапі взаємодії, які події спричиняють зміну стану, а також як система реагує на коректні або некоректні дії користувача.

Діаграми станів використовуються для опису життєвого циклу об’єктів і дозволяють визначити послідовність виконання процесів, зокрема у випадках перевірки, підтвердження, обробки помилок або завершення дій. Основними елементами діаграм станів є початковий і кінцевий стани, переходи між ними, а також умови, що визначають зміну стану.

На рисунку 2.4 зображено діаграму станів для модуля авторизації та реєстрації користувача.

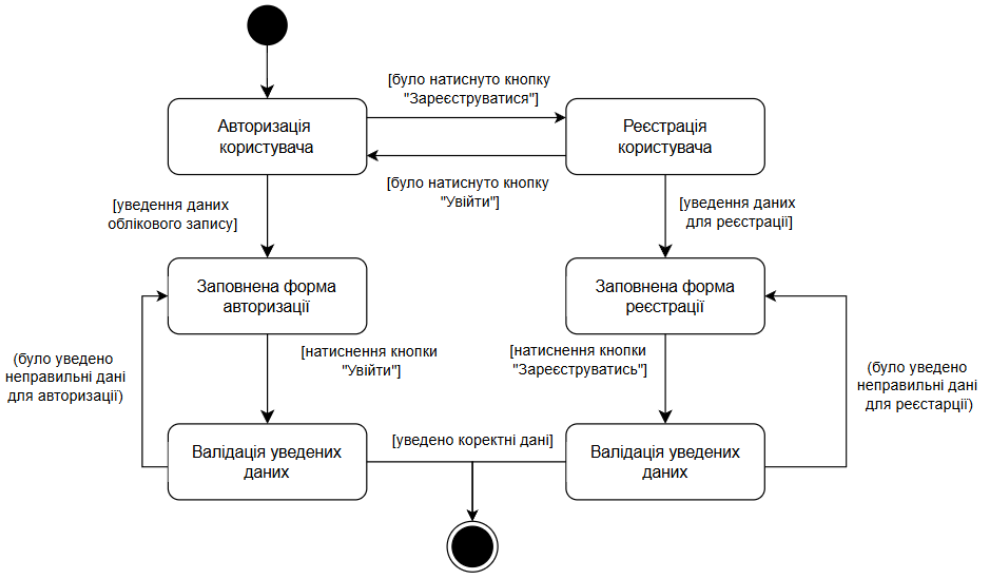

Рисунок 2.4 – Діаграма станів модуля авторизації та реєстрації користувача

Діаграма станів модуля скорочення посилання представлена на рисунку 2.5. Вона ілюструє процес обробки користувацького запиту на створення короткого посилання — від моменту введення URL-адреси до фінального збереження результату в базу даних або повідомлення про помилку.

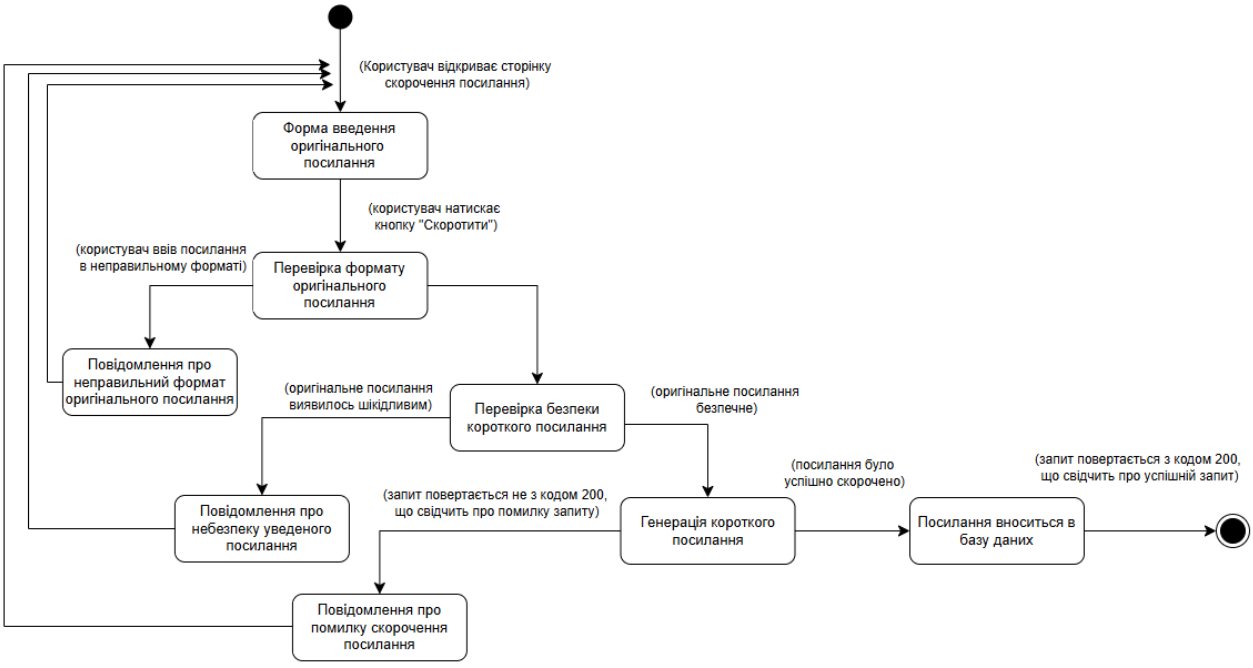

Рисунок 2.5 – Діаграма станів модуля скорочення посилання

Діаграму станів модуля масового скорочення наведено на рисунку 2.6.

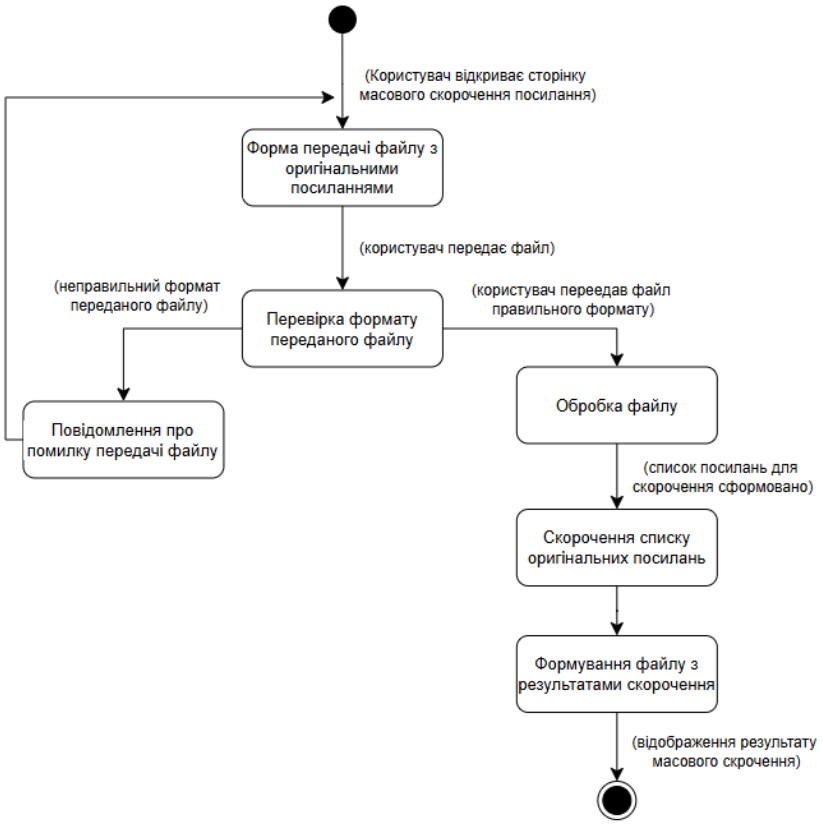

Рисунок 2.6 – Діаграма станів модуля масового скорочення

Діаграму станів модуля збору статистики наведено на рисунку 2.7.

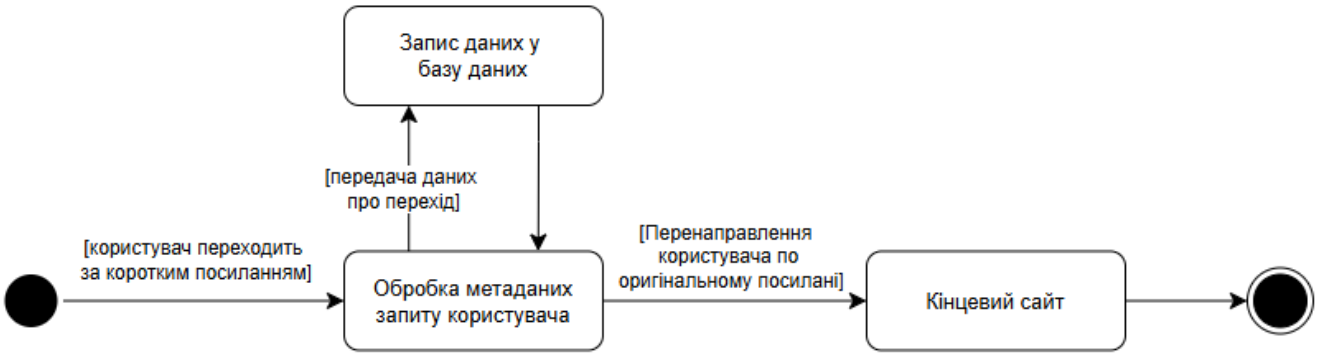

Рисунок 2.7 – Діаграма станів модуля збору статистики

2.7 Висновки

У другому розділі було детально розглянуто ключові аспекти розробки програмного забезпечення для вебсервісу керування короткими URL-посиланнями, зокрема організацію обробки та передачі даних, моделювання структури бази даних і реалізацію функціональних модулів системи. Значну увагу приділено реалізації таких складових, як модулі авторизації, створення та масового скорочення посилань, обробки переходів, збору статистики та локалізації інтерфейсу.

Метод побудови вікон застосунку на базі Laravel і Blade-шаблонів представлено через поетапну розробку графічного інтерфейсу користувача. Використання єдиного макету, часткових шаблонів і компонентів Bootstrap дозволяє створювати адаптивні сторінки з чіткою структурою та зручною навігацією. Динамічна поведінка вікон реалізована за допомогою jQuery, що забезпечує інтерактивність і зручність користування навіть при роботі з великим обсягом даних.

Загальний алгоритм роботи системи та діаграми станів чітко показують взаємозв’язки між модулями програми та їх поведінку.

Таким чином, результати цього розділу відображають послідовну реалізацію програмної частини платформи скорочення посилань з урахуванням вимог до зручності, надійності, масштабованості та безпеки. Проведений аналіз і реалізовані модулі формують цілісну архітектуру, придатну до подальшого розширення відповідно до потреб користувачів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

PHP – це широко використовувана мова програмування, яка вже понад два десятиліття залишається основою для створення динамічних вебзастосунків. Завдяки своїй простоті, гнучкості, підтримці великою кількістю хостинг-провайдерів, а також активній спільноті, PHP залишається одним з найкращих виборів для розробки серверної логіки вебзастосунків. Особливої популярності PHP набув у поєднанні з фреймворком Laravel, що забезпечує сучасну архітектуру, безпечну аутентифікацію, зручну маршрутизацію та підтримку шаблонізатора Blade [13].

JavaScript – це мова програмування, яка традиційно використовується для клієнтської частини вебзастосунків. Завдяки появі таких фреймворків, як React і Vue.js, JavaScript отримав широке застосування також у SPA-архітектурах, що дозволяє створювати динамічні та інтерактивні інтерфейси користувача. Хоча він демонструє високу гнучкість та продуктивність на фронтенді, його застосування на сервері менш поширене у класичних вебпроєктах, які реалізовані з використанням шаблонізаторів.

Python – універсальна мова програмування з чистим синтаксисом, що підходить як для веброзробки, так і для наукових обчислень. Тим не менш, у сфері високопродуктивної веброзробки Python (навіть із фреймворком Django) поступається PHP за швидкістю виконання, спрощеною інтеграцією з системами керування контентом та хостингом [14].

Java – це потужна строго типізована мова з високим рівнем безпеки, що часто використовується для створення великих корпоративних систем. Однак вона має складніший синтаксис, вимагає більше конфігурацій і ресурсів для розробки й запуску, що робить її менш зручною для реалізації легких вебзастосунків [15].

Порівняння функціональних характеристик наведених мов програмування подано в таблиці 3.1.

Таблиця 3.1 – Порівняння функціональних характеристик мов програмування

Характеристика	PHP	JavaScript	Python	Java
Об’єктно-орієнтованість	1	1	1	1
Простота у вивченні	1	1	1	0
Продуктивність у вебі	1	1	0	1
Підтримка сучасних фреймворків	1	1	1	1
Інтеграція з MySQL	1	1	1	1
Простота у розгортанні на сервері	1	0	0	0
Загальний коефіцієнт	6	5	4	4

Як видно з таблиці, PHP демонструє найвищу сумарну оцінку, що зумовлює його вибір як основної мови програмування для розробки серверної частини вебзастосунку скорочення URL-посилань. Його поєднання з фреймворком Laravel забезпечує високу продуктивність, зручність у використанні та розширюваність проєкту, що робить цей стек технологій ідеальним вибором для створення масштабованих і надійних вебсервісів.

Компанії-розробники програмного забезпечення пропонують широкий вибір сучасних середовищ для веброзробки. Розглянемо найпопулярніші з них: Visual Studio Code, PhpStorm, NetBeans і Eclipse.

Visual Studio Code (VS Code) – це безкоштовне, легке та розширюване інтегроване середовище розробки, створене компанією Microsoft [16]. VS Code підтримує велику кількість мов програмування, зокрема PHP, JavaScript, TypeScript, HTML та CSS. Для зручної роботи з Laravel існує низка розширень, як-от Laravel Blade Snippets, Laravel Artisan, Laravel Extra Intellisense, які забезпечують автодоповнення, маршрутизацію, роботу з командним рядком і відлагодження

коду. VS Code також має інтеграцію з Git, підтримує термінал, Docker і з'єднання з базами даних, що робить його універсальним інструментом для сучасного веброзробника.

PhpStorm – це потужне інтегроване середовище розробки (IDE), створене компанією JetBrains спеціально для роботи з PHP. Середовище підтримує широкий набір функціональностей, таких як інтелектуальний аналіз коду, автодоповнення, вбудована перевірка синтаксису, розширена підтримка Laravel та Blade, а також інтеграція з системами контролю версій, базами даних, Docker і вебсерверами [17]. PhpStorm також має інструменти для налагодження, тестування, візуалізації структури коду та рефакторингу.

NetBeans – це безкоштовне та відкрите IDE, що підтримує мови програмування Java, PHP, C/C++, HTML, JavaScript та інші. Воно розробляється під егідою Apache Software Foundation. NetBeans дозволяє створювати PHP-застосунки з базовою підтримкою автодоповнення, форматування коду, запуску сценаріїв і інтеграції з Git [18]. Однак, порівняно з сучасними середовищами, такими як VS Code чи PhpStorm, NetBeans поступається у швидкодії, гнучкості інтерфейсу, адаптації під сучасні фреймворки (зокрема Laravel) і загальному користувацькому досвіді.

Eclipse – це ще одне безкоштовне середовище розробки з відкритим кодом, яке найчастіше асоціюється з мовою програмування Java. Тим не менш, завдяки плагіну PDT (PHP Development Tools), Eclipse можна використовувати і для розробки PHP-застосунків [19]. Середовище забезпечує базову підтримку PHP, зокрема автодоповнення, навігацію по коду та налагодження. Проте процес налаштування середовища для PHP, а особливо для Laravel, є складнішим і потребує додаткових плагінів та конфігурацій. Через це Eclipse рідко обирають для сучасної PHP-розробки, особливо якщо проєкт використовує специфічні особливості Laravel (Blade, Artisan, Eloquent ORM тощо).

Порівняння середовищ наведено в таблиці 3.2.

Таблиця 3.2 – Порівняння функціональних характеристик середовищ програмування

Характеристика	VS Code	PhpStorm	NetBeans	Eclipse
Підтримка PHP	1	1	1	1
Легкість та швидкодія	1	0	0	0
Наявність вбудованого терміналу	1	1	0	0
Підтримка розширень та плагінів Laravel	1	1	0	0
Вбудована Git-інтеграція	1	1	1	1
Інструменти тестування	1	1	1	1
Загальний коефіцієнт	6	5	3	3

Згідно з порівняльним аналізом, Visual Studio Code було обрано як основне середовище розробки проєкту через його легкість, гнучкість, підтримку Laravel, високу швидкість роботи та повну безкоштовність.

Таким чином, комбінація PHP, фреймворку Laravel і Visual Studio Code дозволяє реалізувати функціональну, масштабовану та зручну в обслуговуванні систему керування короткими посиланнями, яка відповідає сучасним вимогам до продуктивності, безпеки та розширюваності.

3.2 Розробка модуля для авторизації та реєстрації користувача

Призначенням модуля автентифікації є створення безпечного та інтуїтивно зрозумілого механізму входу та реєстрації користувачів у вебзастосунку для керування короткими URL-посиланнями. При відкритті сторінки автентифікації користувачеві відображається один з варіантів взаємодії — форма входу або форма реєстрації, які реалізовано з урахуванням сучасних вимог до зручності та доступності інтерфейсів (див. рисунок 3.1).

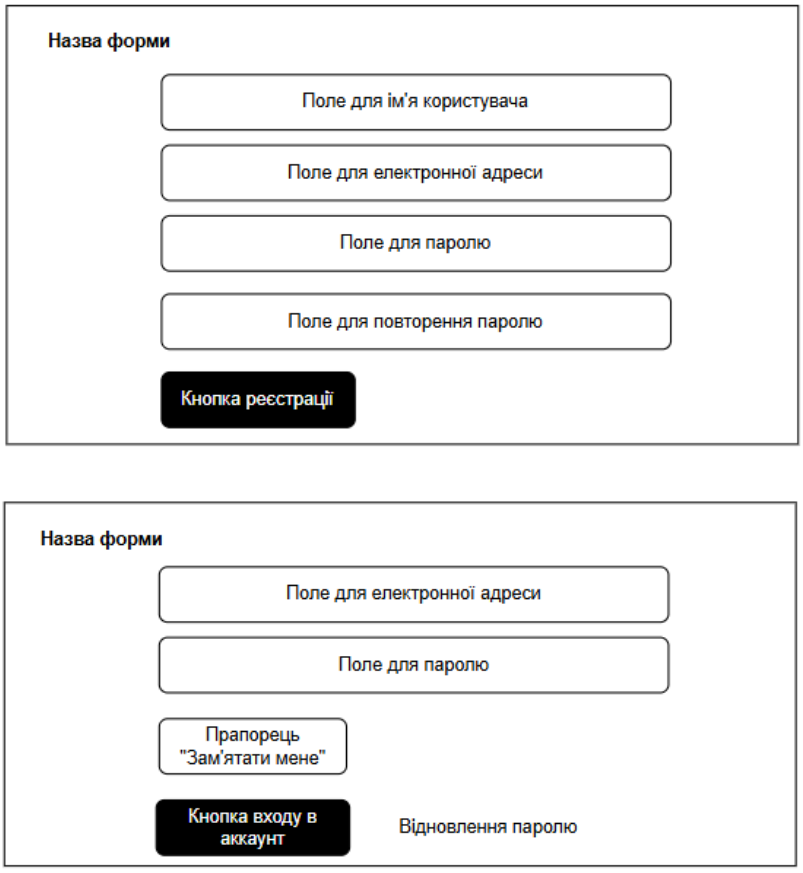

Рисунок 3.1 – Схема інтерфейсу сторінок авторизації та реєстрації

Форма реєстрації передбачає введення імені користувача, електронної адреси, пароля та його підтвердження. У свою чергу, форма входу дозволяє ввести електронну пошту, пароль та встановити прапорець «Запам’ятати мене» для збереження сесії. Також передбачена можливість відновлення пароля у разі його втрати. Вся валідація введених даних реалізована як на стороні клієнта, так і на сервері.

Механізм автентифікації побудовано на стандартних засобах Laravel Breeze, що включає використання Blade-шаблонів, вбудованої маршрутизації, middleware для захисту доступу, а також CSRF-захисту для форм. Захист автентифікаційного процесу забезпечується за допомогою хешування паролів за алгоритмом bcrypt, що реалізовано вбудованими засобами Laravel. Система сесій зберігає стан авторизованого користувача, що дозволяє виконувати доступ до захищених

сторінок без повторного входу.

Зовнішній вигляд інтерфейсу забезпечено за допомогою компонентів Blade та CSS-стилів, які адаптовані для коректного відображення на різних пристроях. Кнопки, поля вводу, повідомлення про помилки та інтерактивні елементи відповідають сучасним вимогам до UX/UI і зручні для сприйняття.

Загалом, архітектура модуля автентифікації забезпечує базовий, але надійний рівень безпеки й функціональності, необхідний для роботи з персоніфікованими даними користувачів у межах системи керування короткими посиланнями.

Було побудовано блок-схему для взаємодії з головною сторінкою застосунку, зображену на рисунку 3.2. Вона описує взаємодію користувача з основними компонентами модуля автентифікації і головною сторінкою.

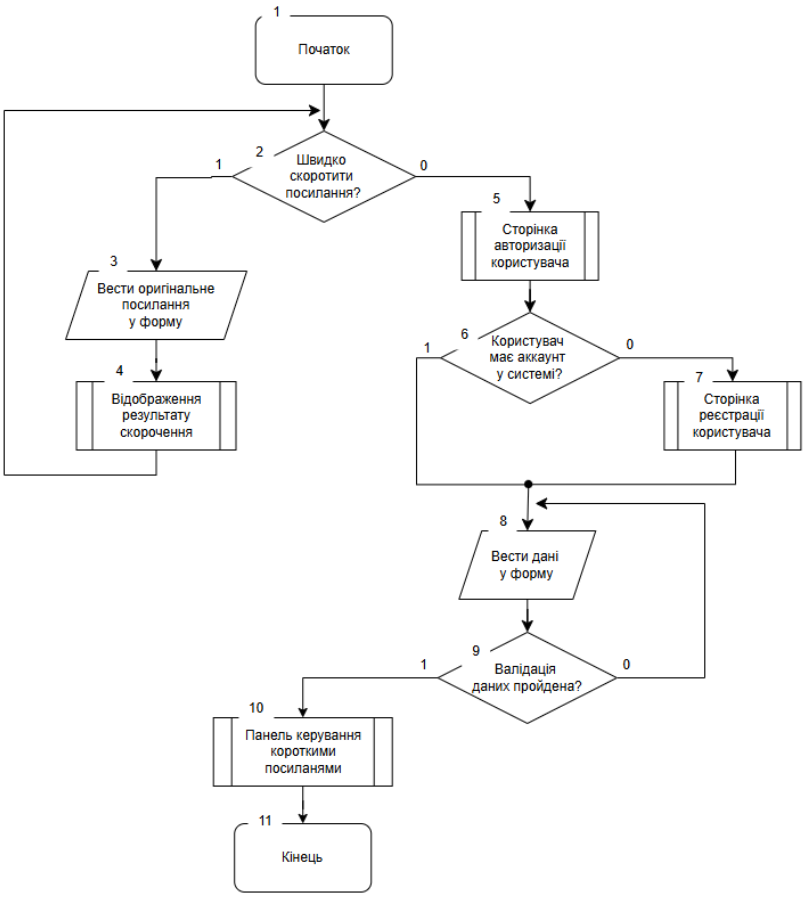

Рисунок 3.2 – Блок-схема взаємодії з головною сторінкою застосунку

3.3 Розробка модуля скорочення посилань

Модуль скорочення посилань є одним з ключових функціональних компонентів вебзастосунку, оскільки саме він реалізує основну бізнес-логіку — створення коротких URL-адрес на основі введеного користувачем оригінального посилання. Його головне завдання — забезпечити зручний, безпечний та надійний процес формування коротких посилань як для неавторизованих користувачів, так і для зареєстрованих. Алгоритм функціонування модуля представлено у вигляді блок-схеми на рисунку 3.3.

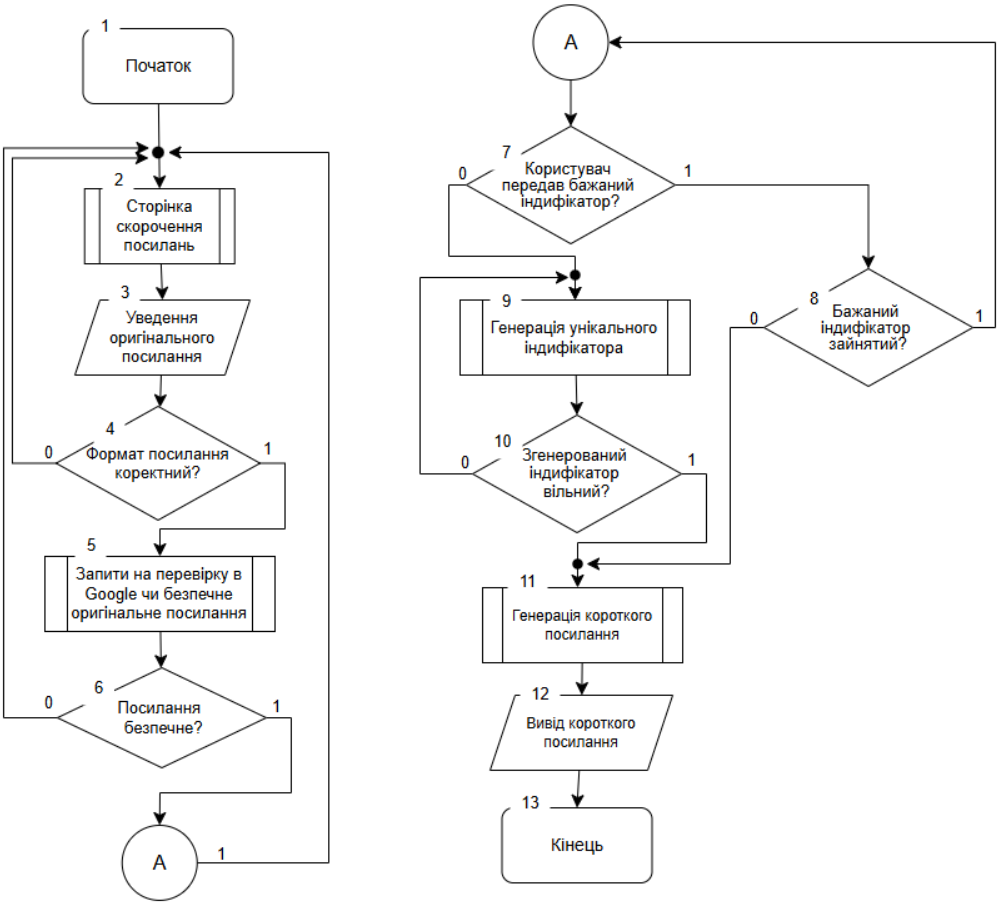

Рисунок 3.3 – Блок-схема алгоритму роботи модуля скорочення посилань

Робота модуля починається з того, що користувач переходить на головну сторінку застосунку, де розміщено форму для введення оригінального посилання.

Після натискання кнопки «Скоротити» система перевіряє правильність формату введеної адреси. У разі помилки користувач отримує повідомлення з поясненням. Якщо формат відповідає стандарту, система надсилає запит до Google Safe Browsing API для перевірки на безпечність посилання. Якщо посилання визначене як потенційно шкідливе, воно не допускається до подальшої обробки.

Після успішної перевірки безпечності система переходить до етапу формування ідентифікатора. Якщо користувач самостійно задає бажаний короткий код, система перевіряє його на унікальність у базі. Якщо код уже зайнятий, формується повідомлення про помилку. Якщо користувач не вказав код або заданий варіант зайнятий, система генерує випадковий ідентифікатор до тих пір, поки не знайде вільний.

У разі успішної генерації посилання створюється запис у базі даних, що включає оригінальний URL, згенерований або заданий короткий код, дату створення, термін дії та інші супровідні параметри. Користувач отримує готове коротке посилання, яке можна копіювати або ділитися ним. Цей процес є однаковим як для авторизованих, так і для гостьових користувачів.

Модуль реалізований за допомогою фреймворку Laravel із використанням Blade-шаблонів для побудови інтерфейсу, а також Eloquent ORM для роботи з базою даних. Це забезпечує швидкодію, захист від SQL-ін’єкцій та зручну підтримку коду в майбутньому. Отже, система є масштабованою та допускає розширення функціоналу, зокрема інтеграцію додаткових методів перевірки або варіантів генерації коду.

3.4 Розробка модуля переходів

Призначення модуля переходів полягає в забезпеченні надійного та безпечного перенаправлення користувача з короткого посилання на його оригінальну URL-адресу з урахуванням блокування, аутентифікації паролем та збору статистичних даних. Алгоритм роботи модуля представлено у вигляді блок-

схеми на рисунку 3.4. Він відповідає за послідовність дій від моменту отримання запиту за скороченим кодом до перенаправлення на цільовий ресурс або відображення повідомлення-попередження.

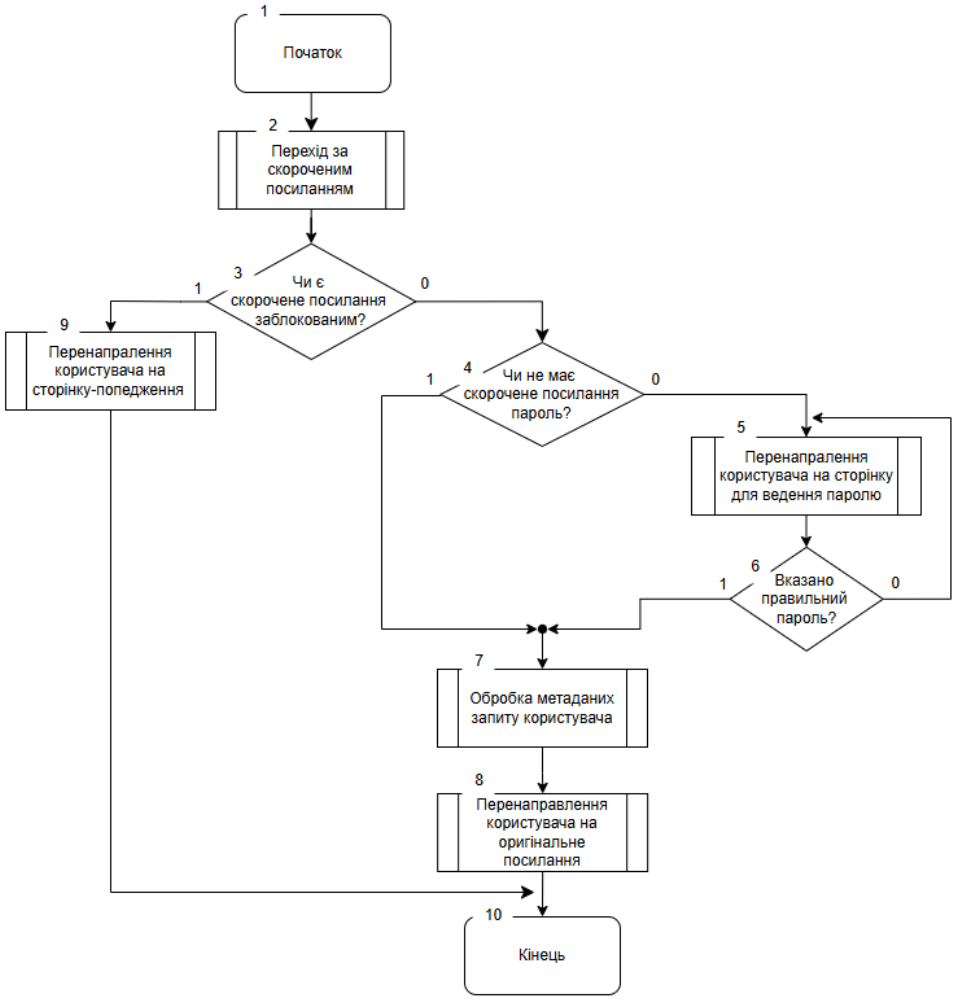

Рисунок 3.4 – Блок-схема алгоритму роботи модуля переходів

Алгоритм починається з отримання HTTP-запиту, коли користувач натискає на скорочене посилання. На цьому етапі система шукає відповідний запис у базі даних і перевіряє, чи не заблоковано посилання. Якщо посилання має статус «Banned», користувача негайно перенаправляють на сторінку-попередження про недоступність ресурсу. У протилежному випадку, коли посилання активне, здійснюється перевірка, чи захищене воно паролем.

Якщо скорочене посилання має встановлений пароль, користувач переходить на окрему сторінку для введення цього пароля, де йому пропонують ввести пароль доступу до цільового ресурсу. Після відправлення форми система порівнює введений пароль із захешованим значенням у базі. У разі невідповідності введення користувач знову бачить форму та отримує повідомлення про помилку. Якщо пароль введено коректно, відбувається перехід до наступного етапу обробки.

Наступним кроком система збирає детальні метадані запиту: IP-адресу користувача, дані про браузер і пристрій (user-agent), час кліку та сторінку з якої було зроблено запит. Ці дані фіксуються в таблиці переходів у базі даних, що дозволяє надалі будувати звіти з аналітики та статистики короткого посилання.

Після коректного збереження всіх необхідних даних сервер повертає відповідь із HTTP-редиректом на повну URL-адресу, яку користувач вказав при створенні скороченого посилання. Усі виняткові ситуації фіксуються через механізм журналювання Laravel Log, що спрощує подальшу діагностику та забезпечує безперервність роботи сервісу.

Таким чином, модуль переходів забезпечує послідовну обробку кожного запиту: від перевірки доступності короткого посилання та пароля до збору статистики й перенаправлення на оригінальне посилання. Завдяки цьому підходу гарантується, що користувач ніколи не потрапить на заблоковані або ненадійні ресурси, а власник сервісу отримає точну інформацію про всі згенеровані переходи.

3.5 Розробка моделі системи

Під час розробки вебзастосунку для скорочення URL-посилань важливим етапом є проєктування архітектури взаємодії користувача з системою. Для цього була побудована UML-діаграма діяльності, яка відображає основні сценарії використання функціоналу платформи та порядок дій у рамках сеансу користувача (див. рисунок 3.5).

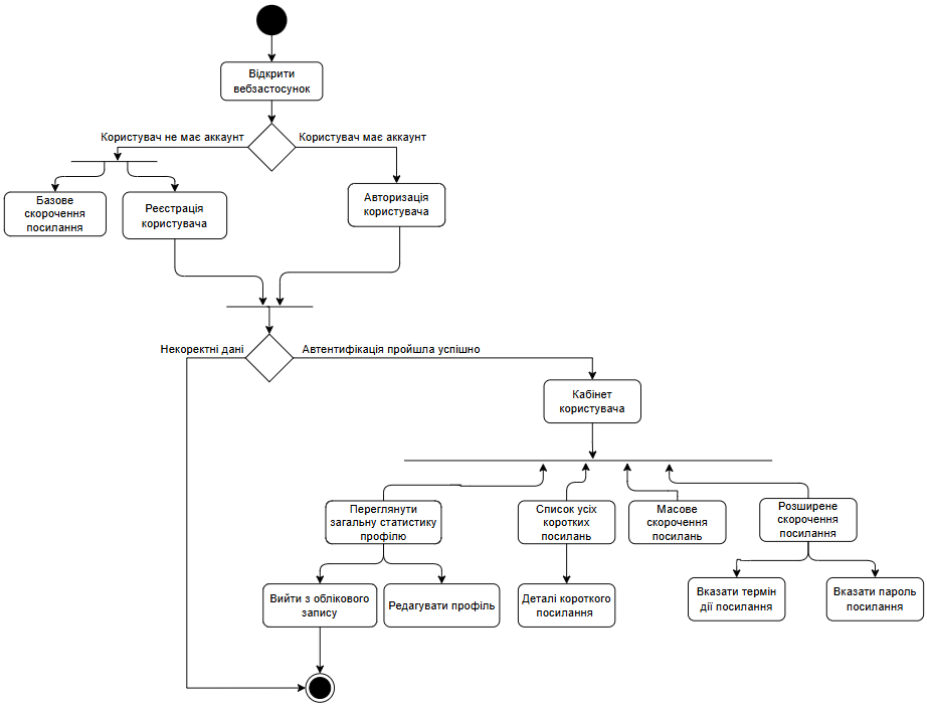

Рисунок 3.5 – Діаграма діяльності системи

На діаграмі зображено повний цикл взаємодії користувача з вебінтерфейсом. Початком процесу є відкриття вебзастосунку. Якщо користувач ще не має облікового запису, він може або зареєструватися, або скористатися базовою функцією створення короткого посилання без входу в систему. За наявності облікового запису користувач проходить автентифікацію, після чого отримує доступ до особистого кабінету. У випадку некоректного введення даних автентифікація переривається.

Усередині кабінету користувача доступні такі функції, як перегляд загальної статистики, редагування профілю, список усіх скорочених посилань з можливістю перегляду детальної інформації про кожне з них, функції масового або розширеного скорочення, включно з можливістю вказати термін дії або пароль до посилання. Крім того, користувач може вийти з облікового запису в будь-який момент, що завершує його сесію в системі.

Такий підхід до моделювання дозволяє чітко описати логіку навігації між основними функціями застосунку, допомагає структурувати взаємодію між

інтерфейсом та користувачем і є основою для побудови інтуїтивного та зрозумілого UX.

3.6 Розробка структури інтерфейсу програмного застосунку

Інтерфейс користувача є одним із ключових елементів системи скорочення URL-посилань, оскільки саме він визначає перше враження користувачів про сервіс, зручність його використання та ефективність взаємодії. Для побудови інтерфейсу було обрано бібліотеку Bootstrap для створення адаптивної розмітки та jQuery для обробки взаємодії з користувачем у реальному часі.

Головна сторінка вебсистеми є центральним елементом взаємодії користувача із сервісом. Вона реалізована таким чином, щоб забезпечити швидке, зручне та інтуїтивно зрозуміле скорочення довгих URL-адрес без необхідності авторизації чи реєстрації. Це дозволяє навіть новим відвідувачам одразу скористатися основною функцією системи – створенням короткого посилання – без зайвих кроків. Такий підхід значно знижує поріг входу для користувача і робить сервіс привабливим для широкої аудиторії (див. рисунок 3.6).

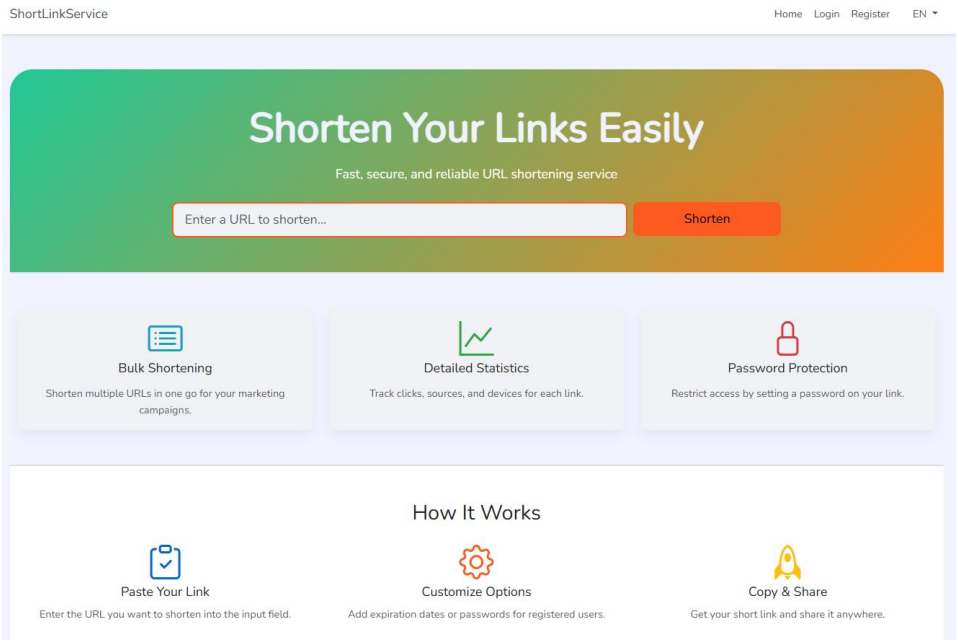

Рисунок 3.6 – Головна сторінка з формою для скорочення посилань

Основним функціональним блоком є форма для введення URL-адреси, яка містить велике текстове поле для вставки довгого посилання та яскраву кнопку «Shorten». Поле введення виділено червоною рамкою в разі валідаційної помилки, що дозволяє швидко зорієнтуватись у разі помилкового заповнення. Завдяки інтеграції бібліотеки jQuery з механізмом обробки подій, форма надсилає запит до сервера асинхронно, без перезавантаження сторінки, а результат – коротке посилання – одразу відображається нижче у відповідному блоці.

Сформоване посилання виводиться у вигляді активного гіперпосилання зеленого кольору. Крім того, поруч розміщена кнопка з іконкою буфера обміну, яка дозволяє одним кліком скопіювати посилання для вставки в інші ресурси.

Для користувачів, які бажають мати розширений функціонал (наприклад, керування посиланнями, масове скорочення посилань, аналітика тощо), передбачена сторінка реєстрації (див. рисунок 3.7).

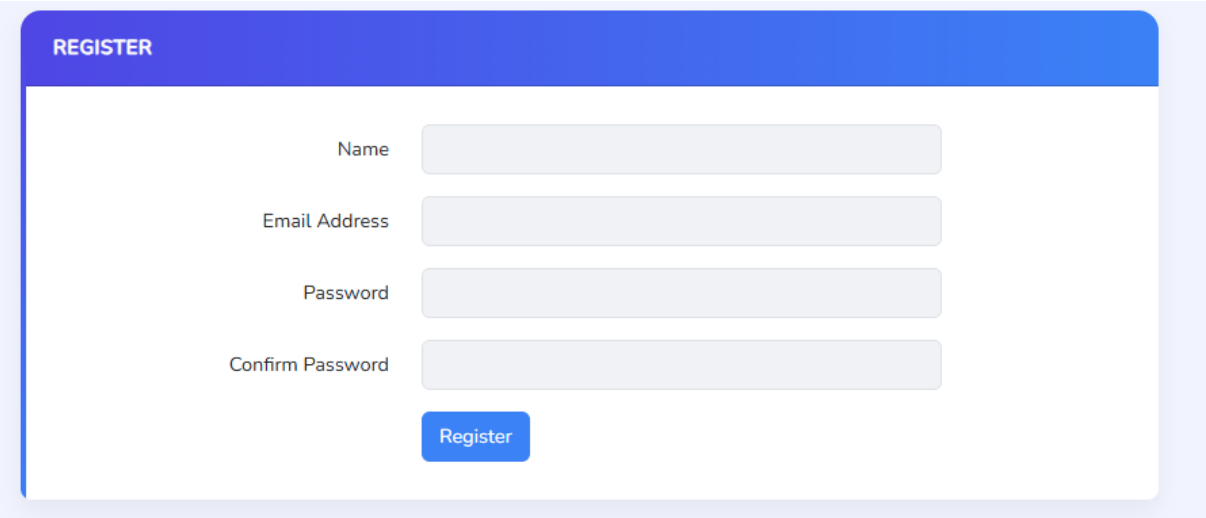

Рисунок 3.7 – Сторінка реєстрації користувача

Інтерфейс сторінки побудовано з урахуванням принципів UX-дизайну: чотири стандартні поля (ім’я, електронна адреса, пароль і підтвердження пароля), а також одна кнопка для завершення реєстрації. Форма має просту структуру, чітку

ієрархію та достатній відступ між елементами, що робить її зручною для заповнення навіть на мобільних пристроях.

Сторінка реалізована з використанням перевірки коректності введення на клієнтському рівні, а також обробки типових помилок, таких як невірно підтверджений пароль або вже зареєстрований email.

Після реєстрації або у випадку повторного входу користувач потрапляє на сторінку логіну (див. рисунок 3.8).

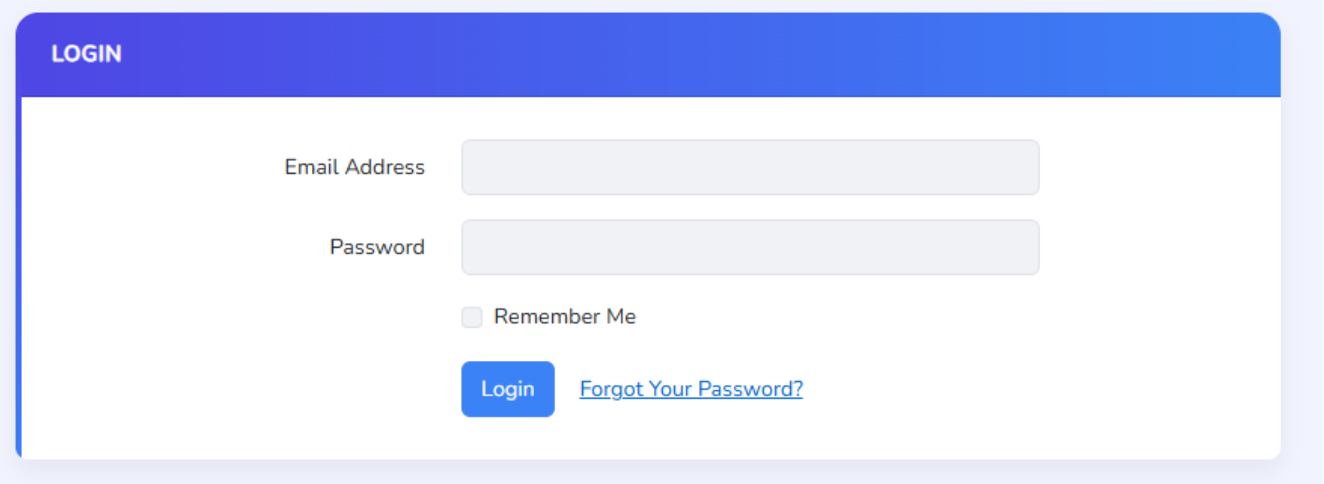

Рисунок 3.8 – Сторінка авторизації користувача

Інтерфейс логіну містить два поля: для електронної адреси та пароля, а також чекбокс «Remember Me», який дозволяє зберігати сесію користувача.

Нижче розміщено посилання «Forgot Your Password?», яке веде до функції відновлення пароля – це стандартний елемент для покращення доступності та безпеки.

Форма логіну оформлена в єдиному стилі з іншими елементами системи, що підтримує цілісність візуального дизайну.

Після авторизації користувач потрапляє до особистої панелі керування, яка є центральною точкою адміністрування скорочених посилань. Інтерфейс цієї

сторінки поєднує аналітичні віджети, інструменти керування посиланнями та навігацію до іншого функціоналу вебсервісу (див. рисунок 3.9).

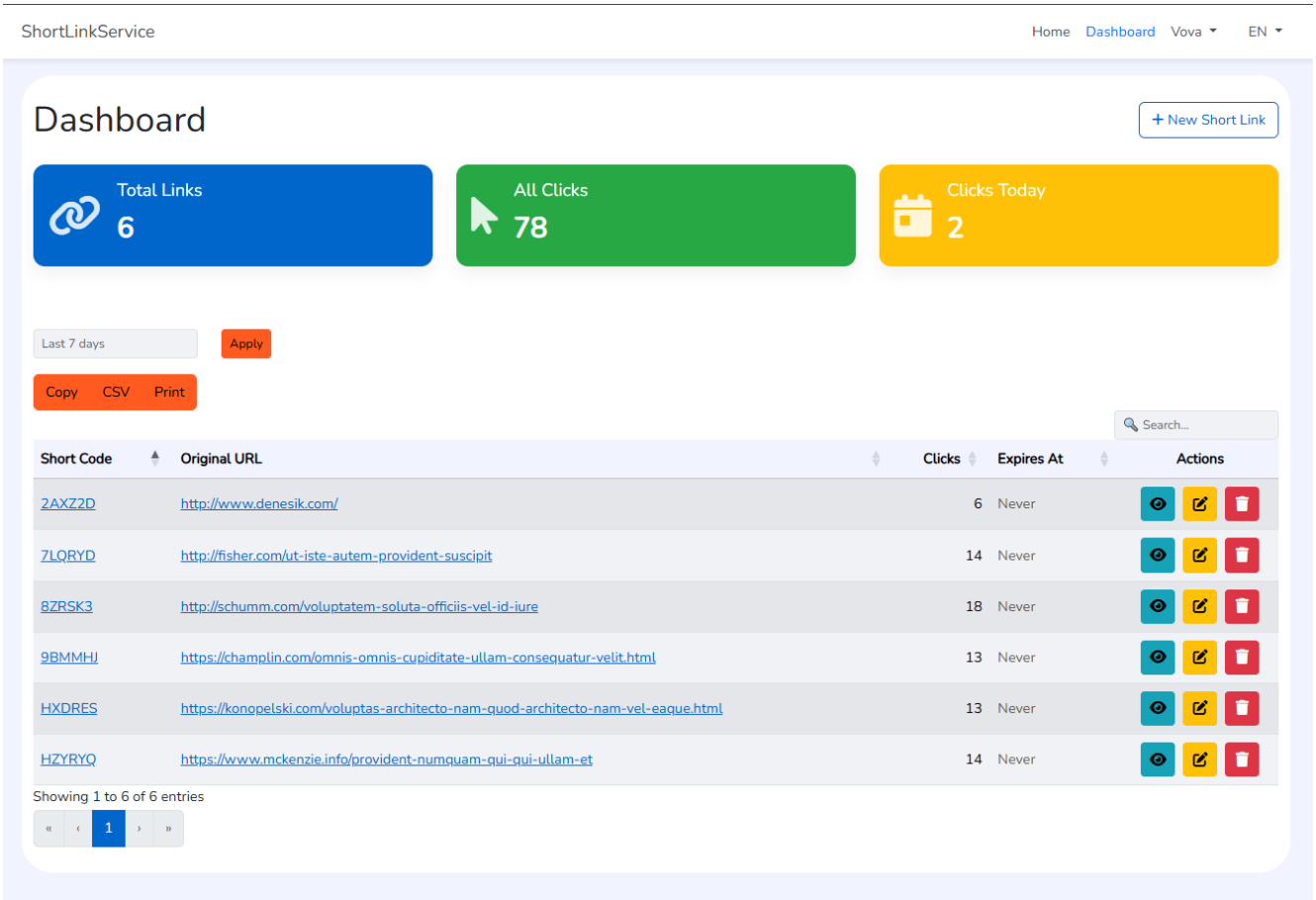

Рисунок 3.9 – Панель керування користувача

Основний табличний блок побудовано на стороні клієнта за допомогою плагіну jQuery DataTables із розширенням Buttons. Таблиця підтримує сортування за будь-якою колонкою, живий пошук у верхній частині й пагінацію [20]. Також поряд розміщено яскраві помаранчеві кнопки Copy, CSV та Print, які дозволяють одразу експортувати або скопіювати дані таблиці.

Кожен рядок таблиці містить клікабельний короткий код, що веде на перенаправляє на оригінальну URL, оригінальний URL, кількість кліків, дату завершення терміну дії та набір кнопок-дій: перегляд, редагування та видалення. Дії реалізовані як кнопки Bootstrap з відповідними піктограмами.

Завдяки інтеграції DataTables із Bootstrap та jQuery панель керування гарантує високу інтерактивність: зміни фільтрів чи пошук миттєво оновлюють вміст без перезавантаження сторінки, натискання на сторінкову навігацію плавно переходить між частинами таблиці, а адаптивний дизайн забезпечує комфортну роботу навіть на вузьких екранах.

Інтерфейс цієї сторінки розроблено з фокусом на зручність і мінімалізм. Основним елементом є форма, яка включає обов'язкове поле для введення оригінального URL, а також декілька опціональних налаштувань: ручне встановлення бажаного короткого коду, захист паролем і вибір дати завершення дії посилання через інтегрований календар (див. рисунок 3.10).

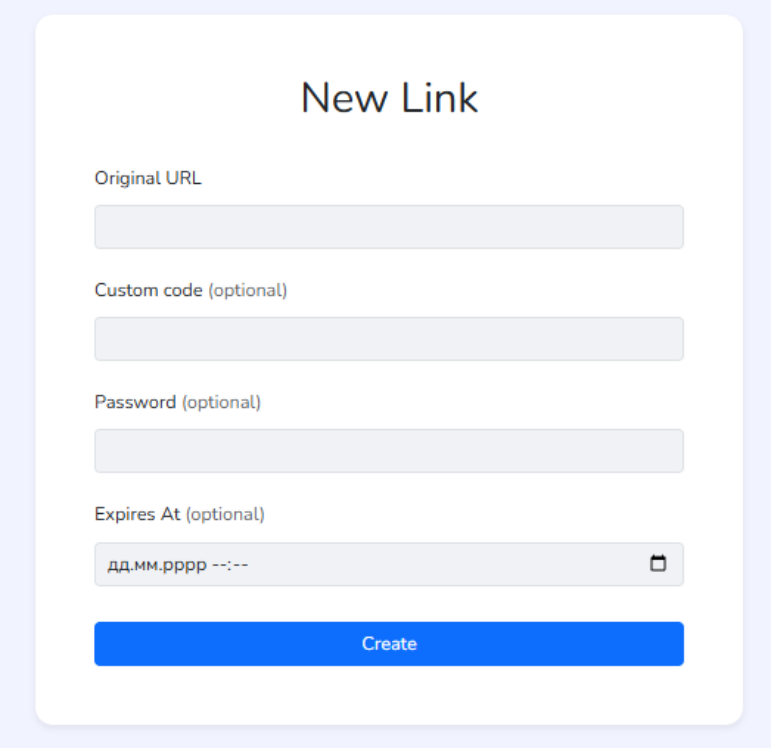

Рисунок 3.10 – Сторінка створення нового короткого посилання

Усі поля форми проходять перевірку на коректність за допомогою JavaScript та jQuery. У разі некоректного введення – наприклад, неправильний формат URL або дублювання короткого коду – користувач бачить модальне вікно з

повідомленням про помилку, оформлене в стилі Bootstrap із підсвічуванням проблемних полів.

У випадку успішного заповнення і надсилання форми, запит відправляється на сервер асинхронно. Після обробки, користувач автоматично перенаправляється на сторінку перегляду створеного посилання, де представлена детальна інформація про нього: сам короткий код, кількість переходів, термін дії та інші параметри. Такий підхід дозволяє зберегти плавність взаємодії та зменшує затримки у користувацькому досвіді.

Для випадків, коли потрібно обробити одразу багато посилань, у вебінтерфейсі реалізовано функціонал масового скорочення, який працює з файлами у форматі .xlsx (див. рисунок 3.11).

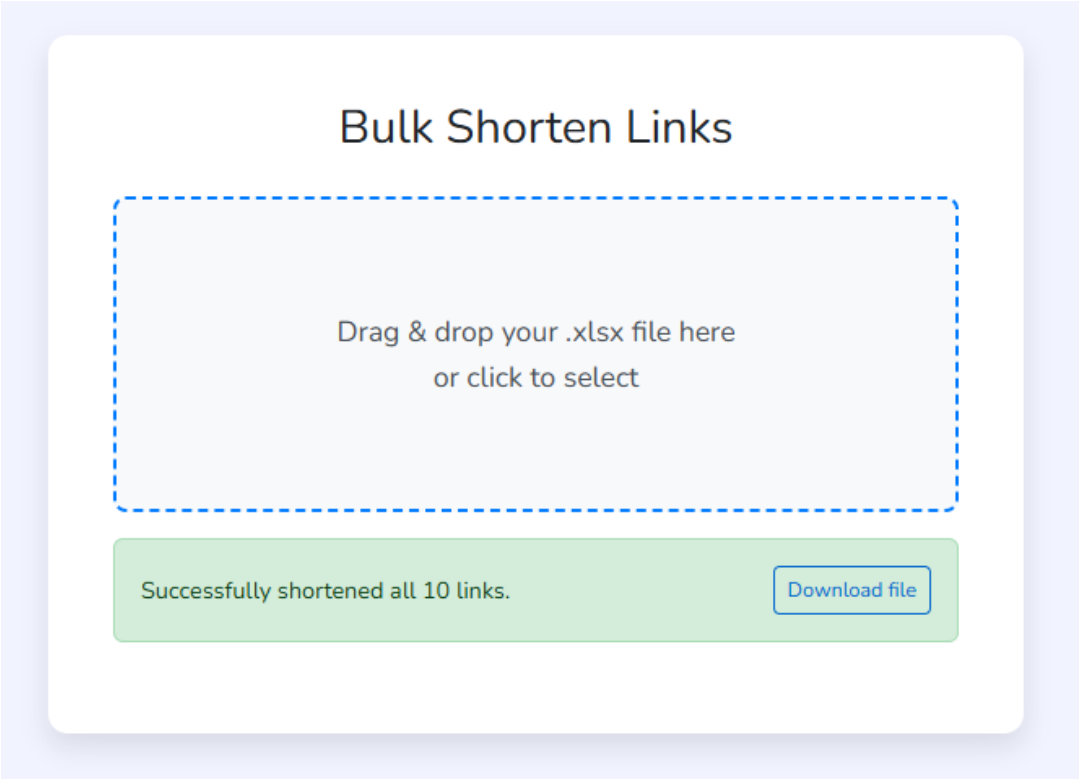

Рисунок 3.11 – Сторінка масового скорочення посилань

Ця сторінка призначена для обробки великої кількості URL-адрес за один запит. Основу складає інтерактивна область, реалізована за допомогою бібліотеки

Dropzone.js, яка дозволяє завантажувати Excel-файли з переліком посилань шляхом простого перетягування або через стандартний файловий діалог.

Після завантаження відбувається асинхронна обробка даних на стороні сервера. Кожне посилання проходить валідацію, після чого система формує відповідні короткі URL. У разі успіху користувач отримує повідомлення із кнопкою для завантаження нового Excel-файлу з результатами.

Загалом інтерфейс є максимально адаптивним і зберігає однорідну візуальну структуру завдяки використанню Bootstrap. Кнопки, повідомлення, поля – усе реалізовано відповідно до єдиного стилю вебсистеми.

3.7 Висновки

У третьому розділі було розглянуто структуру та реалізацію основних модулів вебсистеми скорочення посилань. Здійснено проєктування функціональних компонентів, зокрема модуля реєстрації та авторизації користувачів, інтерфейсу взаємодії, перенаправлення за скороченими посиланнями, збору статистики переходів, а також інструментів адміністрування та обслуговування системи.

Особливу увагу приділено створенню зручного користувацького інтерфейсу, реалізації механізмів безпечного доступу, автоматизації технічних процесів і фіксації аналітичних даних. Усі модулі були реалізовані з використанням сучасного стеку технологій, що забезпечило адаптивність, продуктивність і масштабованість системи. Результатом стала функціональна, стабільна та готова до використання вебсистема, яка відповідає актуальним вимогам до сервісів цього типу..

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Аналіз методів тестування програмного забезпечення

Тестування програмного забезпечення є важливим етапом розробки, який забезпечує її функціональність, зручність та надійність. Цей процес допомагає виявити можливі недоліки на різних етапах створення системи, що дозволяє покращити якість продукту перед його запуском. Основними завданнями тестування є перевірка коректності роботи програми, оцінка її швидкості та стабільності, а також мінімізація ризиків, пов’язаних з помилками, які можуть виникнути під час реального використання.

Методи тестування можна розділити за рівнями та підходами до перевірки. Зокрема, виділяють модульне тестування, яке перевіряє окремі частини системи незалежно одна від одної; інтеграційне тестування, спрямоване на аналіз взаємодії між окремими компонентами, з метою виявлення можливих помилок на межах їх інтеграції; системне тестування, що оцінює роботу програмного забезпечення в цілому, враховуючи взаємодію усіх його компонентів; та приймальне тестування, яке проводиться для підтвердження відповідності кінцевого продукту очікуванням і вимогам користувачів. Кожен із цих рівнів є важливим і дозволяє забезпечити всебічну перевірку розробленого програмного рішення [21].

Статичні методи тестування включають аналіз коду без його виконання. До таких методів належать огляд коду, коли інші розробники перевіряють написаний код на наявність помилок, та використання спеціальних інструментів для автоматичного аналізу. Важливими також є формальні перевірки документації, специфікацій, технічних завдань та інших матеріалів, що супроводжують процес розробки.

Динамічні методи тестування передбачають виконання програмного коду та перевірку системи під час її безпосередньої роботи. Серед цих методів виділяють метод «білої скриньки», який передбачає тестування з урахуванням внутрішньої

структури програми, аналізуючи логіку і шляхи виконання коду; метод «чорної скриньки», при якому програмне забезпечення перевіряється виключно з точки зору користувача, без знання внутрішніх деталей реалізації; та метод «сірої скриньки», що поєднує обидва підходи та дозволяє оцінити як внутрішню логіку системи, так і її зовнішню поведінку.

Проте, процес тестування супроводжується певними викликами. Зокрема, складно виявити приховані помилки, які проявляються лише за специфічних умов або при значному навантаженні. Також процес тестування може вимагати суттєвих витрат часу і ресурсів. Для подолання цих проблем активно використовуються автоматизовані інструменти тестування, які дозволяють прискорити та стандартизувати процес перевірки, підвищуючи ефективність виявлення дефектів. Важливим аспектом є також розробка комплексної стратегії тестування, яка включає різноманітні методи та підходи для забезпечення максимально повного покриття перевірок.

Аналіз методів тестування показує, що правильний вибір підходів, таких як метод «чорної скриньки», дозволяє оптимізувати процес перевірки та виявляти більшість недоліків до того, як програмне забезпечення потрапить до користувачів. Це забезпечує високу якість продукту та задоволення потреб кінцевих користувачів.

4.2 Тестування розробленого програмного застосунку

Системне тестування – це процедура перевірки повністю інтегрованого програмного забезпечення, яка оцінює його відповідність як функціональним, так і нефункціональним вимогам. На відміну від модульного тестування, що зосереджується на ізольованій перевірці окремих компонентів, системне тестування досліджує взаємодію між цими компонентами, комунікаційні протоколи та загальну поведінку системи в різних робочих середовищах [22].

Тестування вебсистеми скорочення URL є важливим етапом для перевірки її функціональності, безпеки та відповідності вимогам. Було протестовано роботу інтерфейсу, логіку серверної взаємодії, обробку помилок і валідацію введених даних. Основну увагу приділено перевірці головної форми скорочення посилань.

Перший тестовий сценарій перевіряв стандартну поведінку системи у разі введення коректного, безпечного та активного URL. На рисунку 4.1 показано результат успішного створення скороченого посилання.

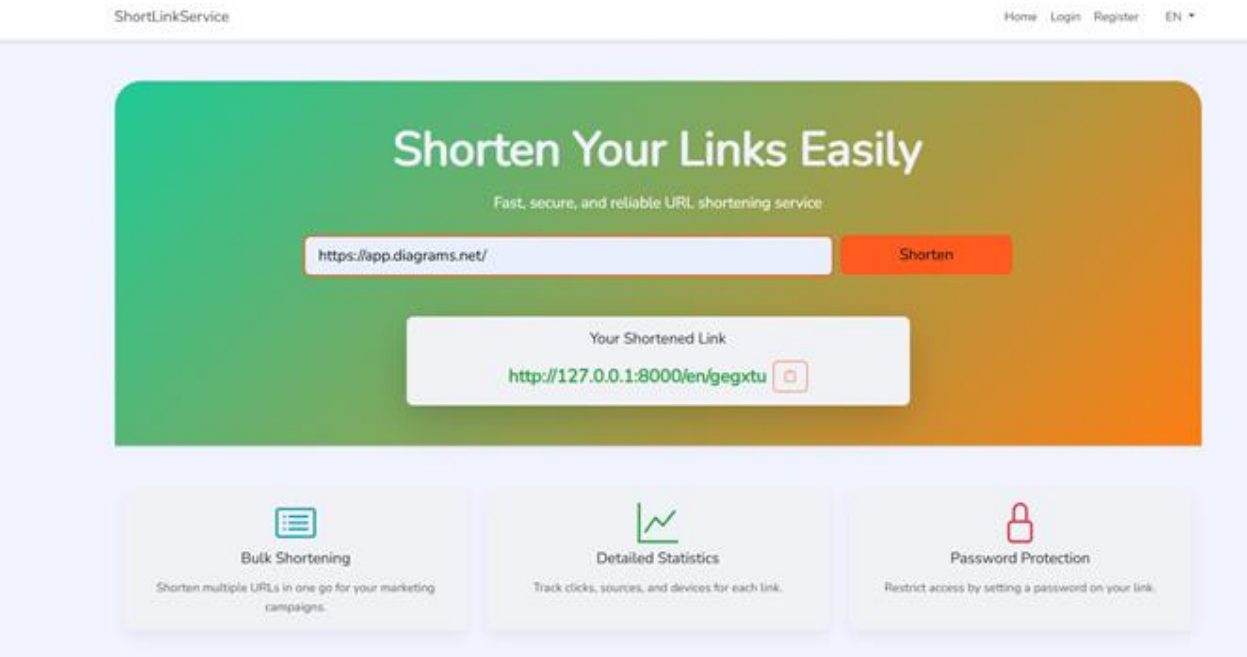

Рисунок 4.1 – Результат успішного скорочення URL-посилання

Після введення повної та дійсної адреси та натискання кнопки «Shorten», система автоматично згенерувала коротке посилання і відобразила його нижче у вигляді активного гіперпосилання. Поруч знаходиться кнопка для копіювання скороченого URL у буфер обміну.

Цей функціонал реалізований з використанням AJAX-запиту до серверного контролера. У разі позитивної відповіді система миттєво оновлює інтерфейс без

перезавантаження сторінки. Це підтверджує коректну роботу фронтенду, взаємодії з бекендом та генерації коротких посилань відповідно до вимог проєкту.

Окремо було перевірено, як система реагує на небезпечні або потенційно шкідливі посилання. У таких випадках користувач має отримати відповідне повідомлення про ризик (див. рисунок 4.2).

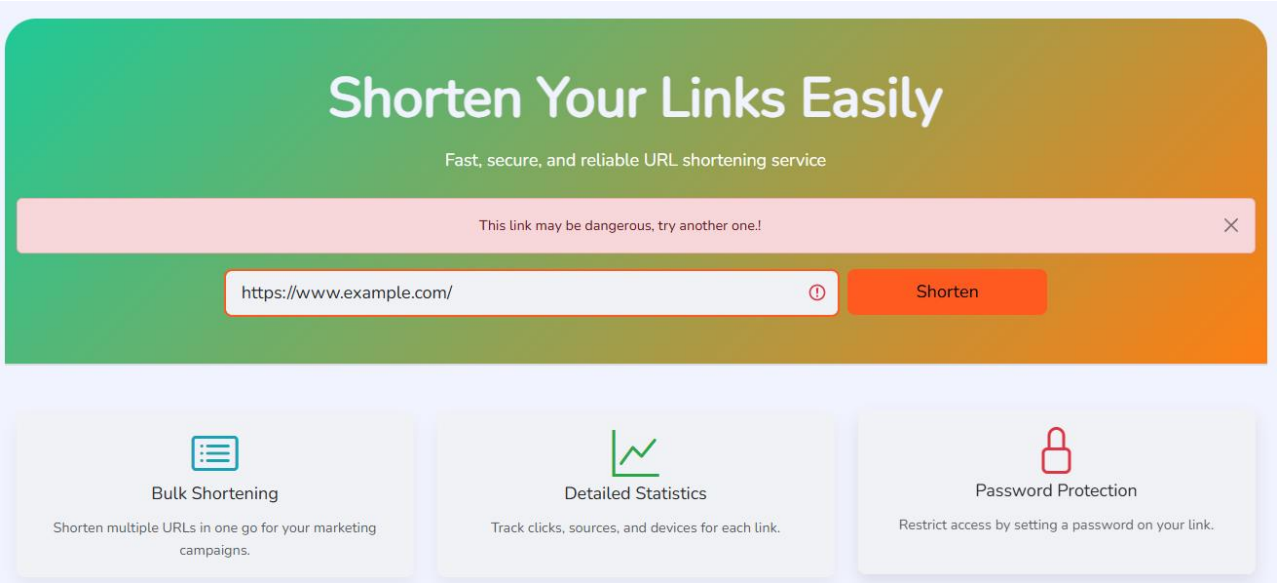

Рисунок 4.2 – Повідомлення про небезпечне посилання

У разі, якщо введена URL-адреса не проходить перевірку безпеки, система попереджає користувача, що посилання може бути небезпечним. Повідомлення оформлене у вигляді інформативного червоного блоку над полем вводу. Додатково вхідне поле підсвічується червоною рамкою для візуального акценту.

Такий механізм дозволяє уникнути реєстрації небажаного чи потенційно зловмисного вмісту в базі даних сервісу, а також підвищує рівень довіри з боку кінцевих користувачів.

Наступним етапом стало тестування можливості реєстрації користувача. Було перевірено поведінку системи у разі помилок у заповненні форми. На рисунку 4.3 представлено ситуацію, коли користувач ввів некоректну адресу електронної пошти та занадто короткий пароль.

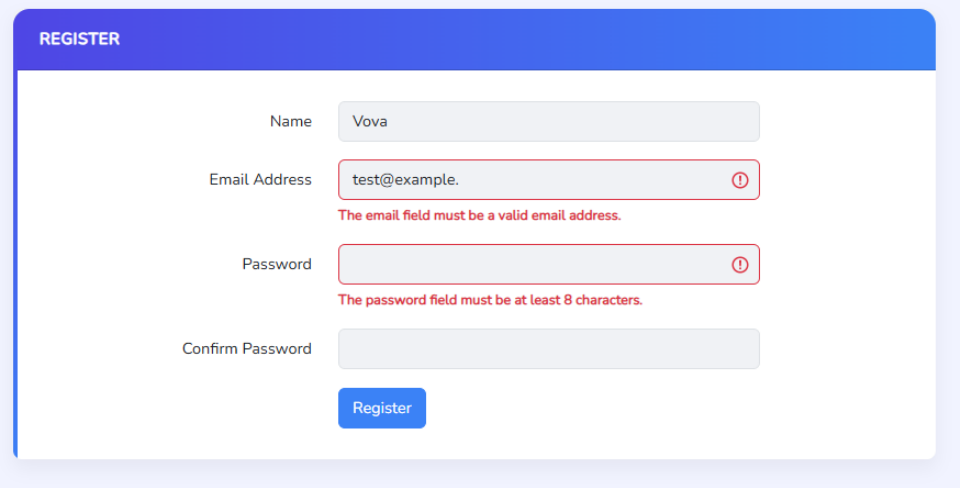

Рисунок 4.3 – Помилки при реєстрації

Усі помилки були чітко вказані поруч із відповідними полями, а також виділені червоним кольором. Це забезпечує швидке реагування з боку користувача та покращує загальну зручність користування. Система перевіряє валідність email-адреси та вимагає, щоб пароль містив щонайменше 8 символів. У випадку коректного введення даних обліковий запис додається в систему.

Для перевірки обробки логіну було протестовано ситуацію з введенням неправильних облікових даних (див. рисунок 4.4). У цьому випадку система коректно відобразила повідомлення про те, що облікові дані не збігаються з наявними у базі.

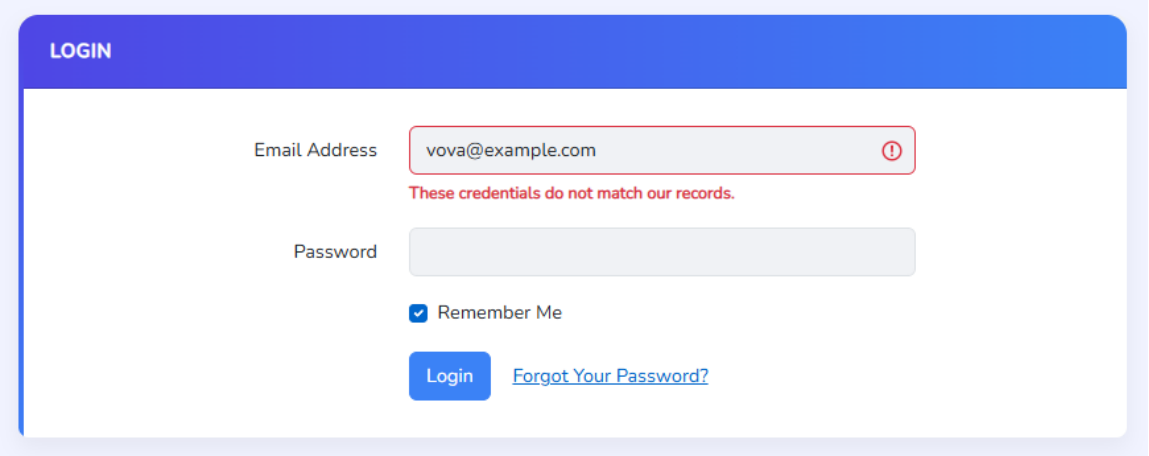

Рисунок 4.4 – Повідомлення про невірні облікові дані при авторизації

Уведене значення email залишилось у полі, щоб користувач міг легко змінити тільки пароль або скористатися функцією відновлення доступу. Такий підхід дозволяє мінімізувати втрати часу при повторних спробах входу.

Окрему увагу під час тестування було приділено функціоналу масового скорочення посилань. На рисунку 4.5 представлено результат завантаження Excel-файлу з 9 записами, з яких 7 не були скорочені через помилки у форматі вхідних URL.

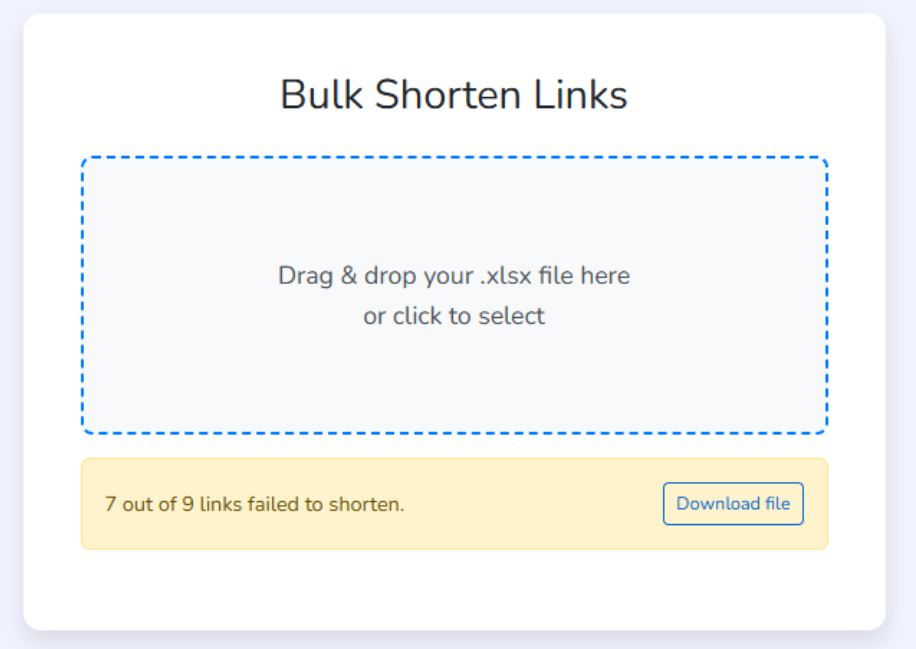

Рисунок 4.5 – Результат масового скорочення з помилками у вхідних даних

Після обробки з’являється повідомлення про часткову невдачу, у якому зазначено точну кількість некоректно опрацьованих рядків. Також надається можливість завантажити зворотний файл із результатами – у форматі Excel – для подальшого аналізу.

На рисунку 4.6 продемонстровано вміст завантаженого звіту: у колонці Short URL вказується або створене коротке посилання, або повідомлення «Invalid URL», якщо вхідне значення не відповідало очікуваному шаблону.

	A	B
1	Original URL	Short URL
2	not a url	Invalid URL
3	https://www.example.com/	http://127.0.0.1:8000/en/tckdiy
4	://missing.scheme.com	Invalid URL
5	http//missingcolon.com	Invalid URL
6	www.without-scheme.com	Invalid URL
7	http:/one-slash.com	Invalid URL
8	https://www.example.com/	http://127.0.0.1:8000/en/bzohyv
9	http://	Invalid URL
10	example.com/no-scheme	Invalid URL

Рисунок 4.6 – Файл із результатами обробки масового скорочення посилань.

Як видно з прикладу, система успішно обробила лише 2 з 9 рядків, тоді як у решті випадків вказані типові помилки користувачів: відсутній протокол, зайві символи на початку, або загальна некоректність структури посилання. Така деталізація допомагає користувачу швидко проаналізувати помилки та виправити їх для повторного завантаження.

На всіх протестованих сторінках було перевірено відповідність елементів інтерфейсу вимогам адаптивності: відображення на мобільних пристроях, доступність елементів, поведінка при різних розмірах екранів.

Загалом, результати тестування підтвердили, що відповідні модулі вебсервісу скорочення URL-посилань працюють стабільно, забезпечують коректну роботу і обробку виняткових ситуацій.

4.3 Розробка інструкції користувача вебзастосунку

Інструкція користувача містить основні технічні вимоги для запуску вебсервісу скорочення посилань, а також покроковий опис взаємодії з інтерфейсом. Для використання системи не потрібно встановлювати додаткове програмне забезпечення – сервіс повністю функціонує у сучасному браузері.

Мінімальні та рекомендовані вимоги до клієнтського пристрою наведено в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Параметр	Мінімальна конфігурація	Рекомендована конфігурація
Тип процесора	Intel i3-5000	Intel i5-7000
Оперативна пам’ять	2 ГБ	4 ГБ
Місце на жорсткому диску	500 МБ	1 ГБ
Операційна система	Windows 10, Linux, Mac	Windows 11, Linux, Mac
Браузер	Google Chrome 90+ версії / Firefox 80+	Google Chrome (остання версія)

Після відкриття вебдодатку користувач одразу бачить головну сторінку з формою скорочення URL-посилання. Якщо ввести коректну адресу у відповідне поле та натиснути кнопку «Shorten», система створює коротке посилання, яке з’являється нижче разом із кнопкою копіювання. Цей функціонал доступний без авторизації, що дозволяє швидко скористатися сервісом для базових потреб.

У разі, якщо користувач бажає отримати розширені можливості – перегляд статистики переходів, редагування або видалення власних посилань, а також пакетне завантаження через Excel-файл – необхідно пройти реєстрацію або авторизацію. Після входу до особистого кабінету відкривається панель керування, у якій відображається список усіх створених користувачем посилань, кількість кліків, статус та дата створення. У цій же панелі доступні функції фільтрації, пошуку, експорту таблиці у CSV та масової обробки посилань.

Інтерфейс системи також дозволяє завантажити Excel-файл із довгими URL-адресами. Після обробки даних користувач отримує відповідь із зазначенням успішно скорочених посилань, а у випадку помилок – пояснення та згенерований файл із розподілом результатів. Це значно спрощує роботу з великим обсягом URL-посилань.

4.4 Висновки

У межах четвертого розділу було проведено тестування ключових компонентів вебсистеми скорочення посилань та підготовлено інструкцію користувача. Перевірено як коректну роботу форм створення, реєстрації та входу, так і реакцію системи на помилкові та небезпечні дані. Всі модулі показали стабільність, виявлені недоліки виправлено, а механізми обробки помилок удосконалено. Інструкція дозволяє легко ознайомитися з функціоналом системи, а результати тестування підтвердили її готовність до впровадження як надійного та зручного сервісу для роботи з короткими посиланнями.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено методи та програмні засоби вебсервісу керування короткими URL-посиланнями, що включає наступні основні функціональні модулі:

– модуль реєстрації та автентифікації користувачів;

– модуль генерації унікальних коротких посилань з автоматичною та кастомною ідентифікацією;

– модуль масового скорочення URL за допомогою завантаження файлів;

– модуль збору аналітики та статистики переходів;

– реалізацію механізму захисту посилань паролем та перевірки їхньої безпечності;

– функціонал інтернаціоналізації та локалізації інтерфейсу.

Для реалізації цієї системи було розроблено метод, модель та алгоритми її роботи. Використано мову програмування PHP у поєднанні з фреймворком Laravel для реалізації серверної частини, а також середовище розробки Visual Studio Code. Робота оформлена з урахуванням рекомендацій [23].

У першому розділі було проведено аналіз сучасних систем для керування короткими URL-посиланнями. Було виявлено їхні недоліки, що стало основою для розробки власного вебсервісу. Другий розділ охоплював аналіз даних, розробку методу та моделі роботи системи, а також побудову алгоритмів роботи застосунку та його окремих модулів. У третьому розділі було обґрунтовано вибір засобів для реалізації програмного забезпечення, проведено варіантний аналіз та розроблено програмні модулі й графічний інтерфейс користувача. Четвертий розділ описує тестування розробленої системи, яке підтвердило її працездатність і відповідність очікуванням. Крім того, визначено мінімальні та рекомендовані вимоги до пристроїв для використання вебсервісу, а також розроблено інструкцію з користування програмним забезпеченням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SocialPilot. Importance of a URL Shortener in Social Media Marketing [Електронний ресурс] – Режим доступу: https://www.socialpilot.co/blog/importance-url-shortener-social-media-marketing (дата звернення: 26.03.2025).

2. M. Richards, N. Ford. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, 2020. – 432 с.

3. Medium: Collision handling in our URL shortener service [Електронний ресурс] – Режим доступу: https://medium.com/homeday/collision-handling-in-our-url-shortener-service-6612e3e82eae (дата звернення: 27.03.2025).

4. Черноволик Г.О. Розробка методу і засобів реалізації вебсистеми скорочення посилань / Г.О. Черноволик, С.В. Бевз, В.Г. Коровай // Матеріали Міжнародної науково-практичної інтернет-конференції "Молодь в науці: дослідження, проблеми, перспективи (МН-2025)". [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/24136/19976

5. Bitly [Електронний ресурс] – Режим доступу: https://bitly.com/ (дата звернення: 28.03.2025).

6. TinyURL [Електронний ресурс] – Режим доступу: https://tinyurl.com/ (дата звернення: 28.03.2025).

7. Rebrandly [Електронний ресурс] – Режим доступу до ресурсу: https://www.rebrandly.com/ (дата звернення: 28.03.2025).

8. ScaleupAlly. 21 Benefits of Using Laravel [Електронний ресурс] – Режим доступу: https://scaleupally.io/blog/benefits-of-laravel-framework/ (дата звернення: 28.03.2025).

9. WebFX. The Power of jQuery with Ajax [Електронний ресурс] – Режим доступу: https://www.webfx.com/blog/web-design/the-power-of-jquery-with-ajax/ (дата звернення: 29.03.2025).

10. OWDT. The pros and cons of using Bootstrap for front-end development [Електронний ресурс] – Режим доступу: https://owdt.com/article/the-pros-and-cons-of-using-bootstrap-for-front-end-development/ (дата звернення: 30.03.2025)

11. Datamation. 8 Major Advantages of Using MySQL [Електронний ресурс] – Режим доступу: https://www.datamation.com/storage/8-major-advantages-of-using-mysql/ (дата звернення: 31.03.2025).

12. All You Need to Know about State Diagrams [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/about-state-diagrams. (дата звернення: 09.03.2025).

13. Laravel Documentation [Електронний ресурс] – https://laravel.com/docs (дата звернення: 16.04.2025).

14. What is Python? - Python Programming Language Explained [Електронний ресурс] – https://aws.amazon.com/what-is/python/ (дата звернення: 16.04.2025).

15. Rahul Bisht. Java Programming: A Comprehensive Guide for Beginners. 2023. 140 p.

16. Visual Studio Code - Code Editing. Redefined [Електронний ресурс] – https://code.visualstudio.com/ (дата звернення: 20.04.2025).

17. PhpStorm: The PHP IDE by JetBrains [Електронний ресурс] – https://www.jetbrains.com/phpstorm/ (дата звернення: 20.04.2025).

18. Welcome to Apache NetBeans [Електронний ресурс] – https://netbeans.apache.org/front/main/index.html (дата звернення: 24.04.2025).

19. About the Eclipse Foundation | The Eclipse Foundation. [Електронний ресурс] – https://www.eclipse.org/org/ (дата звернення: 27.04.2025).

20. DataTables. Manual - DataTables example [Електронний ресурс] – Режим доступу: https://datatables.net/manual/ (дата звернення: 03.05.2025).

21. Рівні тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalearning.com.ua/theory/lectures/material/testing-levels. (дата звернення: 13.05.2025).

22. G. Blokdyk. System Testing A Complete Guide - 2020 Edition. 5STARCooks, 2021. – 304 с.

23. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

61

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
«_21_» __березня__ 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка методу та програмних засобів вебсервісу керування короткими URL-посиланнями» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____ к.т.н., доцент Г. О. Черноволик
«_21_» __березня__ 2025 р.

Виконав:
_____ студент гр. ЗПІ-21б В. Г. Коровай
«_21_» __березня__ 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка методу та програмних засобів вебсервісу керування короткими URL-посиланнями».

Галузь застосування – сфера онлайн-спілкування: освітні заклади, бізнес-організації та соціальні платформи.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

3. Мета та призначення розробки.

Метою дослідження є покращення процесу роботи зі скороченими посиланнями за рахунок впровадження розробленої вебсистеми, яка забезпечить користувачів зручним інструментом для керування URL-адресами та аналізу трафіку.

Призначення роботи – розробка методу та програмних засобів вебсервісу керування короткими URL-посиланнями.

4. Вихідні дані для проведення НДР

1. Сучасна комунікація: особливості мовлення в мережі інтернет [Електронний ресурс] – Режим доступу до ресурсу: http://dspace.nbuv.gov.ua/handle/123456789/178612.

2. Baumgartner S. (2023). TypeScript Cookbook: Real World Type-Level Programming: O’Reilly Media.

3. Рівні тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalearning.com.ua/theory/lectures/material/testing-levels.

5. Технічні вимоги

Комп’ютер з 64-розрядним (x64) процесором та тактовою частотою 2 ГГц. Об’єм оперативної пам’яті 8 ГБ, розмір жорсткого диску 256 ГБ. Операційна система Windows 10. ПК з будь-якою операційною системою та будь-який браузер.

6. Конструктивні вимоги

Вебзастосунок та повинен відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

- пояснювальна записка до бакалаврської кваліфікаційної роботи;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку систем керування короткими URL-посиланнями та постановка задач розробки	25.03.25 – 31.03.25
2	Розробка методів, моделі та алгоритмів роботи програми	01.04.25 – 14.04.25
3	Розробка програмного забезпечення	15.04.25 – 12.05.25
4	Тестування програми	13.05.25 – 15.05.25
5	Оформлення матеріалів до захисту БКР	16.05.25 – 30.05.25

10. Порядок контролю та прийняття

Всі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

65

Додаток Б. Протокол перевірки бакалаврської кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка методу та програмних засобів вебсервісу керування короткими URL-посиланнями.

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Черноволик Г. О.

Показники звіту подібності

Оригінальність	96.8%
Загальна схожість	3.2%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

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

Автор роботи _____ Коровай В. Г.

Керівник роботи _____ Черноволик Г. О.

Додаток В. Лістинг програми

```
// app/Http/Controllers/DashboardController.php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\Auth;
use Carbon\Carbon;

class DashboardController extends Controller
{
    public function index()
    {
        $user = Auth::user();

        $totalLinks = $user->links()->count();


        $totalClicks = $user->links()
            ->withCount('clicks')
            ->get()
            ->sum('clicks_count');

        $linkIds = $user->links()->pluck('id');
        $todayClicks = \App\Models\Click::whereIn('link_id', $linkIds)
            ->whereDate('clicked_at', Carbon::today())
            ->count();

        $links = $user->links()
            ->withCount('clicks')
            ->latest()
            ->paginate(10);

        return view('dashboard.index', compact(
            'totalLinks',
            'totalClicks',
            'todayClicks',
            'links'
        ));
    }
}
// app/Http/Controllers/HomeController.php
<?php

namespace App\Http\Controllers;
```

```
use GuzzleHttp\Exception\GuzzleException;
use Illuminate\Http\Request;
use App\Models\Link;
use Illuminate\Support\Str;
use GuzzleHttp\Client;

class HomeController extends Controller
{
    public function index()
    {
        return view('home');
    }

    public function shorten(Request $request)
    {
        $data = $request->validate([
            'original_url' => 'required|url|max:2048',
            'custom_code' => 'nullable|alpha_num|min:4|max:20',
            'password' => 'nullable|string|min:4|max:100',
            'expires_at' => 'nullable|date|after:now',
        ]);

        if ($this->isSafeUrl($data['original_url'])) {
            return response()->json([
                'message' => __('home.error_unsafe_url')
            ], 422);
        }

        $code = Link::generateUniqueCode($data['custom_code'] ?? null);

        $link = Link::create([
            'original_url' => $data['original_url'],
            'short_code' => $code,
            'password' => $data['password'] ?? null,
            'expires_at' => $data['expires_at'] ?? null,
            'user_id' => optional($request->user())->id,
        ]);

        return response()->json([
            'short_code' => $code,
            'short_url' => route('links.redirect', $code),
            'expires_at' => $link->expires_at,
            'link_id' => $link->id,
        ]);
    }

    protected function isSafeUrl(string $url): bool
    {
```

```
            $apiKey = config('services.safebrowsing.key');

            if (!$apiKey) {
                \Log::warning('Safe Browsing: API key not set, skipping URL check.');
                return true;
            }

            $client = new Client(['base_uri' => 'https://safebrowsing.googleapis.com']);

            try {
                $response = $client->post("/v4/threatMatches:find?key={$apiKey}", [
                    'json' => [
                        'client' => [
                            'clientId' => 'short-url-service',
                            'clientVersion' => '1.0.0',
                        ],
                        'threatInfo' => [
                            'threatTypes' => ['MALWARE', 'SOCIAL_ENGINEERING',
'UNWANTED_SOFTWARE'],
                            'platformTypes' => ['ANY_PLATFORM'],
                            'threatEntryTypes' => ['URL'],
                            'threatEntries' => [['url' => $url]],
                        ],
                    ],
                ]);
            } catch (GuzzleException $e) {
                \Log::error('Safe Browsing API error: ' . $e->getMessage());
                return true;
            }

            $body = json_decode($response->getBody(), true);
            return empty($body['matches'] ?? []);
        }
    }
    // app/Http/Controllers/LinkBulkController.php
    <?php

    namespace App\Http\Controllers;

    use Illuminate\Http\Request;
    use App\Models\Link;

    class LinkBulkController extends Controller
    {
        public function bulkForm()
        {
            return view('links.bulk');
        }
```

```
public function bulkProcess(Request $request)
{
    $request->validate([
        'file' => 'required|mimes:xlsx'
    ]);

    [$pairs, $invalidCount] = $this->parseBulkFile($request->file('file'));

    $filename = 'bulk_shortened_' . time() . '.xlsx';
    $path = storage_path('app/public/' . $filename);
    $data = array_merge([['Original URL', 'Short URL']], $pairs);
    \Shuchkin\SimpleXLSXGen::fromArray($data)->saveAs($path);

    return response()->json([
        'download_url' => route('links.bulk.download', $filename),
        'total' => count($pairs),
        'invalid' => $invalidCount,
    ]);
}

protected function parseBulkFile($file): array
{
    $xlsx = \Shuchkin\SimpleXLSX::parse($file->getRealPath());
    $pairs = [];
    $invalidCount = 0;

    foreach ($xlsx->rows() as $row) {
        $orig = trim($row[0] ?? '');
        if ($orig === '') continue;

        if (filter_var($orig, FILTER_VALIDATE_URL)) {
            try {
                $code = Link::generateUniqueCode();
                Link::create([
                    'original_url' => $orig,
                    'short_code' => $code,
                    'user_id' => optional(request()->user())->id,
                ]);
                $short = route('links.redirect', $code);
            } catch (\Throwable $e) {
                $short = __('links.bulk.error');
                $invalidCount++;
            }
        } else {
            $short = 'Invalid URL';
            $invalidCount++;
        }
        $pairs[] = [$orig, $short];
    }
```

```
        return [$pairs, $invalidCount];
    }


    public function bulkDownload(string $filename)
    {
        $path = storage_path("app/public/{$filename}");

        if (!file_exists($path)) {
            abort(404, __('links.bulk.error_file_not_found'));
        }

        return response()
            ->download($path, $filename)
            ->deleteFileAfterSend(true);
    }
}
// app/Http/Controllers/LinkRedirectController.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Link;
use App\Http\Controllers\Traits\RecordsClicks;
use Illuminate\Support\Facades\Hash;

class LinkRedirectController extends Controller
{
    use RecordsClicks;

    public function redirect($code)
    {
        $link = Link::where('short_code', $code)
            ->where(function ($q) {
                $q->whereNull('expires_at')
                    ->orWhere('expires_at', '>', now());
            })
            ->firstOrFail();

        if ($link->password) {
            session()->put('link.password_required', $link->id);
            return view('links.password', compact('link'));
        }

        $this->recordClick($link);
```

```
        return redirect()->away($link->original_url);
    }

    public function unlock(Request $request, $code)
    {
        $link = Link::where('short_code', $code)->firstOrFail();

        $request->validate([
            'password' => 'required|string',
        ]);

        if (!Hash::check($request->password, $link->password)) {
            return back()->withErrors(__('messages.password_incorrect'));
        }

        session()->forget('link.password_required');

        $this->recordClick($link);

        return redirect()->away($link->original_url);
    }
}
// app/Http/Controllers/ShortLinkController.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Link;
use App\Http\Controllers\Traits\RecordsClicks;

class ShortLinkController extends Controller
{
    use RecordsClicks;

    public function create()
    {
        return view('links.create');
    }

    public function show(Link $link)
    {
        $this->authorize('view', $link);
        $clicks = $link->clicks()->latest('clicked_at')->get();

        return view('links.show', compact('link', 'clicks'));
    }
```

```
    public function edit(Link $link)
    {
        $this->authorize('update', $link);

        return view('links.edit', compact('link'));
    }

    public function update(Request $request, Link $link)
    {
        $this->authorize('update', $link);

        $data = $request->validate([
            'original_url' => 'required|url|max:2048',
            'expires_at' => 'nullable|date|after:now',
            'password' => 'nullable|string|min:6|max:60',
        ]);

        if ($data['password']) {
            $data['password'] = bcrypt($data['password']);
        } else {
            unset($data['password']);
        }

        $link->update($data);

        return back()->with('status', __('messages.link_updated'));
    }

    public function destroy(Link $link)
    {
        $this->authorize('delete', $link);
        $link->delete();

        return redirect()
            ->route('dashboard')
            ->with('status', __('messages.link_deleted'));
    }
}

// app/Http/Controllers/Traits/RecordsClicks.php
<?php

namespace App\Http\Controllers\Traits;

use App\Models\Link;
use Http;
use Log;

trait RecordsClicks
```

```
{

    protected function recordClick(Link $link): void
    {
        $request = request();
        $ipAddress = $request->ip();
        $referrer = $request->headers->get('referer', '');
        $userAgentString = $request->headers->get('User-Agent', '');

        $agent = new Agent();
        $agent->setUserAgent($userAgentString);
        $browser = $agent->browser();

        try {
            $reader = new \GeoIp2\Database\Reader(storage_path('app/GeoLite2-City.mmdb'));
            $record = $reader->city($ipAddress);
            $country = $record->country->isoCode ?: 'Unknown';
        } catch (\Throwable $e) {
            \Log::warning("GeoIP lookup failed for IP {$ipAddress}: " . $e->getMessage());
            $country = 'Unknown';
        }

        $link->clicks()->create([
            'clicked_at' => now()->toDateTimeString(),
            'referrer' => $referrer,
            'ip_address' => $ipAddress,
            'user_agent' => $userAgentString,
            'browser' => $browser,
            'country' => $country,
        ]);
    }
}
// app/Models/Click.php

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Click extends Model
{
    use HasFactory;
    public $timestamps = false;

    protected $fillable = [
        'link_id','clicked_at','referrer','ip_address'
    ];
```

```
    protected $casts = [
        'clicked_at' => 'datetime',
    ];

    public function link()
    {
        return $this->belongsTo(Link::class);
    }
}
// app/Models/Link.php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Link extends Model
{
    use HasFactory;

    protected $fillable = [
        'original_url',
        'short_code',
        'user_id',
        'password',
        'expires_at'
    ];

    protected $casts = [
        'expires_at' => 'datetime',
    ];

    public function clicks()
    {
        return $this->hasMany(Click::class);
    }

    /**
     *
     * @param  string|null  $custom
     * @return string
     *
     * @throws \Illuminate\Validation\ValidationException
     */
    public static function generateUniqueCode(?string $custom = null): string
    {
        $cfg = config('links.generator');
```

```
        $blacklist = config('links.blacklist', []);

        $alphabet = implode('abcdefghijklmnopqrstuvwxyz', array_filter([
            $cfg['lowercase'] ? 'abcdefghijklmnopqrstuvwxyz' : null,
            $cfg['uppercase'] ? 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' : null,
            $cfg['digits'] ? '0123456789' : null,
            $cfg['symbols'] ?? null,
        ]));

        if ($custom) {
            if (in_array($custom, $blacklist, true)
                || static::where('short_code', $custom)->exists()
            ) {
                abort(422, __('home.error_code_taken'));
            }
            return $custom;
        }

        $length = (int) $cfg['length'];

        do {
            $code = '';
            for ($i = 0; $i < $length; $i++) {
                $code .= $alphabet[random_int(0, strlen($alphabet) - 1)];
            }
        } while (
            in_array($code, $blacklist, true)
            || static::where('short_code', $code)->exists()
        );

        return $code;
    }
}
// app/Models/User.php
<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    /** @use HasFactory<\Database\Factories\UserFactory> */
    use HasFactory, Notifiable;

    /**
```

```
     * The attributes that are mass assignable.
     *
     * @var list<string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var list<string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * Get the attributes that should be cast.
     *
     * @return array<string, string>
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }

    public function links()
    {
        return $this->hasMany(Link::class);
    }
}
// app/Policies/LinkPolicy.php
<?php

namespace App\Policies;

use App\Models\User;
use App\Models\Link;
use Illuminate\Auth\Access\HandlesAuthorization;

class LinkPolicy
{
```

```
use HandlesAuthorization;

/**
 * Determine whether the user can view any links (e.g. in index).
 */
public function viewAny(User $user)
{
    return true;
}

/**
 * Determine whether the user can view the link.
 */
public function view(User $user, Link $link)
{
    return $user->id === $link->user_id;
}

/**
 * Determine whether the user can create links.
 */
public function create(User $user)
{
    return true;
}

/**
 * Determine whether the user can update the link.
 */
public function update(User $user, Link $link)
{
    return $user->id === $link->user_id;
}

/**
 * Determine whether the user can delete the link.
 */
public function delete(User $user, Link $link)
{
    return $user->id === $link->user_id;
}

/**
 * Determine whether the user can restore the link.
 */
public function restore(User $user, Link $link)
{
    return $user->id === $link->user_id;
}
```

```
        /**
         * Determine whether the user can permanently delete the link.
         */
        public function forceDelete(User $user, Link $link)
        {
            return $user->id === $link->user_id;
        }
    }
    // routes/web.php
    <?php

    use App\Http\Controllers\DashboardController;
    use App\Http\Controllers\HomeController;
    use App\Http\Controllers\LinkBulkController;
    use App\Http\Controllers\ShortLinkController;
    use App\Http\Controllers\LinkRedirectController;
    use Mcamara\LaravelLocalization\Facades\LaravelLocalization;
    use Mcamara\LaravelLocalization\Middleware\LocaleSessionRedirect;
    use Mcamara\LaravelLocalization\Middleware\LaravelLocalizationRedirectFilter;

    Route::group([
        'prefix' => LaravelLocalization::setLocale(),
        'middleware' => [
            LocaleSessionRedirect::class,
            LaravelLocalizationRedirectFilter::class,
        ],
    ], function () {
        Route::get('/', [HomeController::class, 'index'])->name('home');
        Route::get('/home', [HomeController::class, 'index']);
        Route::post('/shorten', [HomeController::class, 'shorten'])->name('home.shorten');

        Auth::routes();

        Route::middleware('auth')->group(function () {
            Route::get('/dashboard', [ShortLinkController::class, 'index'])->name('dashboard');
            Route::get('/links/bulk', [LinkBulkController::class, 'bulkForm'])->name('links.bulk.form');
            Route::post('links/bulk/process', [LinkBulkController::class, 'bulkProcess'])-
>name('links.bulk.process');
            Route::get('links/bulk/download/{filename}', [LinkBulkController::class,
'bulkDownload'])->name('links.bulk.download');
            Route::resource('links', ShortLinkController::class)->except('index');
            Route::get('/dashboard', [DashboardController::class, 'index'])->name('dashboard');
        });

        Route::post('/{code}/unlock', [LinkRedirectController::class, 'unlock'])
            ->name('links.unlock');
        Route::get('/{code}', [LinkRedirectController::class, 'redirect'])
            ->name('links.redirect');
```

```
});
// resources/views/dashboard/index.blade.php
{{-- resources/views/dashboard/index.blade.php --}}
@extends('layouts.app')

@push('styles')
    @vite('resources/sass/pages/dashboard.scss')
@endpush

@section('content')
    <div class="bg-white rounded-5 container py-4">

        <div class="d-flex justify-content-between align-items-center mb-4">
            <h1 class="mb-0">{{ __('Dashboard') }}</h1>
            <a href="{{ route('links.create') }}" class="btn btn-outline-primary">
                <i class="fas fa-plus"></i> {{ __('New Short Link') }}
            </a>
        </div>

        <div class="row mb-5 stats-overview">
            @foreach ([['color' => 'primary', 'icon' => 'link', 'title' => __('Total Links'), 'value' =>
$totalLinks], ['color' => 'success', 'icon' => 'mouse-pointer', 'title' => __('All Clicks'), 'value' =>
$totalClicks], ['color' => 'warning', 'icon' => 'calendar-day', 'title' => __('Clicks Today'), 'value' =>
$todayClicks]] as $stat)
                <div class="col-md-4 mb-3">
                    <div class="card text-white bg-{{ $stat['color'] }} h-100 stat-card">
                        <div class="card-body d-flex align-items-center">
                            <i class="fas fa-{{ $stat['icon'] }} fa-3x me-3"></i>
                            <div>
                                <h5 class="card-title mb-1">{{ $stat['title'] }}</h5>
                                <p class="card-text fs-2">{{ $stat['value'] }}</p>
                            </div>
                        </div>
                    </div>
                </div>
            @endforeach
        </div>

        <div class="row mb-3">
            <div class="col-auto">
                <input type="text" id="dateRange" class="form-control form-control-sm"
                    placeholder="{{ __('Last 7 days') }}">
            </div>
            <div class="col">
                <button id="applyFilter" class="btn btn-sm btn-secondary">{{ __('Apply')
}}</button>
            </div>
        </div>
```

```
            <div class="row">
                <div class="col">
                    @include('dashboard.partials.links-table')
                </div>
            </div>
        </div>
    @endsection

    @push('scripts')
        @vite('resources/js/pages/dashboard.js')
    @endpush

    // resources/views/dashboard/partials/links-table.blade.php
    {{-- resources/views/dashboard/partials/links-table.blade.php --}}
    <table id="linksTable" class="table table-striped table-hover nowrap w-100">
        <thead class="table-light">
            <tr>
                <th>{{ __('Short Code') }}</th>
                <th>{{ __('Original URL') }}</th>
                <th>{{ __('Clicks') }}</th>
                <th>{{ __('Expires At') }}</th>
                <th class="text-center">{{ __('Actions') }}</th>
            </tr>
        </thead>
        <tbody>
            @foreach ($links as $link)
                <tr>
                    <td>
                        <a href="{{ route('links.show', $link) }}" class="link-primary">
                            {{ $link->short_code }}
                        </a>
                    </td>
                    <td class="text-truncate" style="max-width:220px;">
                        <a href="{{ $link->original_url }}" target="_blank">{{ $link->original_url }}</a>
                    </td>
                    <td>{{ $link->clicks_count }}</td>
                    <td>
                        @if ($link->expires_at)
                            {{ $link->expires_at->format('Y-m-d') }}
                        @else
                            <span class="text-muted">{{ __('Never') }}</span>
                        @endif
                    </td>
                    <td class="text-center">
                        <a href="{{ route('links.show', $link) }}" class="btn btn-sm align-content-center
btn-info me-1"><i
                                class="fas fa-eye"></i></a>
                        <a href="{{ route('links.edit', $link) }}" class="btn btn-sm align-content-center
btn-warning me-1"><i
```

```
                        class="fas fa-edit"></i></a>
                    <button class="btn btn-sm btn-danger delete-btn"><i class="fas fa-
trash"></i></button>
                    <form action="{{ route('links.destroy', $link) }}" method="POST" class="d-none
delete-form">
                        @csrf @method('DELETE')
                    </form>
                </td>
            </tr>
        @endforeach
    </tbody>
</table>
// resources/views/layouts/app.blade.php
<!doctype html>
<html lang="{{ app()->getLocale() }}">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <title>@yield('title', __('messages.app_name'))</title>

    {{-- Vite --}}
    @vite(['resources/js/app.js', 'resources/sass/app.scss'])
    @stack('styles')
</head>

<body class="d-flex flex-column min-vh-100 bg-light">

    {{-- Navbar --}}
    @include('partials.navbar')

    <main class="flex-fill container my-5">
        @if (session('status'))
            <div class="alert alert-success alert-dismissible fade show" role="alert">
                {{ session('status') }}
                <button type="button" class="btn-close" data-bs-dismiss="alert"
                    aria-label="{{ __('messages.close') }}"></button>
            </div>
        @endif

        @yield('content')
    </main>

    <footer class="footer mt-auto text-center py-3">
        &copy; {{ date('Y') }} ShortLink {{ __('messages.all_rights_reserved') }}
    </footer>
```

```
    @stack('scripts')
</body>

</html>
// resources/views/links/bulk.blade.php
@extends('layouts.app')

@push('styles')
    <link href="https://cdnjs.cloudflare.com/ajax/libs/dropzone/5.9.3/dropzone.min.css"
rel="stylesheet" />
    <style>
        .dropzone {
            border: 2px dashed #007bff;
            border-radius: .5rem;
            background: #f8f9fa;
            padding: 2rem;
        }

        .dz-message {
            font-size: 1.1rem;
            color: #495057;
        }

        .bulk-container {
            max-width: 600px;
            margin: 4rem auto;
            background: #fff;
            padding: 2.5rem;
            border-radius: .75rem;
            box-shadow: 0 0.5rem 1rem rgba(0, 0, 0, 0.1);
        }

        #bulkResult .alert {
            margin-top: 1rem;
            display: flex;
            align-items: center;
            justify-content: space-between;
        }
    </style>
@endpush

@section('content')
    <div class="bulk-container">
        <h2 class="text-center mb-4">@lang('links.bulk.title')</h2>

        <form action="{{ route('links.bulk.process') }}" class="dropzone" id="bulkDropzone"
enctype="multipart/form-data">
            @csrf
            <div class="dz-message">
```

```
                {!! trans('links.bulk.dropzone_hint') !!}
            </div>
        </form>

        <div id="bulkResult"></div>
    </div>
@endsection

@push('scripts')
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/dropzone/5.9.3/dropzone.min.js"></script>
    <script>
        const tSuccessAll = "@lang('links.bulk.success_all', ['total' => ':total'])";
        const tPartial = "@lang('links.bulk.partial', ['invalid' => ':invalid', 'total' => ':total'])";
        const tError = "@lang('links.bulk.error')";
        const tDownload = "Download file";

        Dropzone.options.bulkDropzone = {
            paramName: "file",
            maxFiles: 1,
            acceptedFiles: ".xlsx",
            init: function() {
                this.on("success", (file, resp) => {
                    let msg, cls;
                    if (resp.invalid === 0) {
                        msg = tSuccessAll.replace(':total', resp.total);
                        cls = 'alert-success';
                    } else {
                        msg = tPartial
                            .replace(':invalid', resp.invalid)
                            .replace(':total', resp.total);
                        cls = 'alert-warning';
                    }

                    document.getElementById('bulkResult').innerHTML =
                        `<div class="alert ${cls}" role="alert">
                     <span>${msg}</span>
                     <a href="${resp.download_url}" download class="btn btn-sm btn-outline-
primary">${tDownload}</a>
                    </div>`;

                    this.removeAllFiles(true);
                });

                this.on("error", (file, err) => {
                    document.getElementById('bulkResult').innerHTML =
                        `<div class="alert alert-danger" role="alert">${tError}</div>`;
                    this.removeFile(file);
                });
```

```
                }
            };
        </script>
    @endpush
    // resources/views/links/create.blade.php
    @extends('layouts.app')

    @push('styles')
        @vite('resources/sass/pages/home.scss')
    @endpush

    @section('content')
        <div class="create-page d-flex justify-content-center align-items-center">
            <div class="card shadow-sm bg-white p-4" style="max-width: 480px; width: 100%;
border-radius: .75rem;">
                <div class="card-body">
                    <h2 class="card-title mb-4 text-center">New Link</h2>
                    <form id="newLinkForm" action="{{ route('home.shorten') }}" method="POST">
                        @csrf

                        <div class="mb-3">
                            <label for="original_url" class="form-label small">Original URL</label>
                            <input type="url" class="form-control form-control-sm" id="original_url"
name="original_url"
                                required>
                            <div class="invalid-feedback small" id="error-original_url"></div>
                        </div>

                        <div class="mb-3">
                            <label for="custom_code" class="form-label small">Custom code <span
                                    class="text-muted">(optional)</span></label>
                            <input type="text" class="form-control form-control-sm" id="custom_code"
name="custom_code">
                            <div class="invalid-feedback small" id="error-custom_code"></div>
                        </div>

                        <div class="mb-3">
                            <label for="password" class="form-label small">Password <span
                                    class="text-muted">(optional)</span></label>
                            <input type="password" class="form-control form-control-sm" id="password"
name="password">
                            <div class="invalid-feedback small" id="error-password"></div>
                        </div>

                        <div class="mb-4">
                            <label for="expires_at" class="form-label small">Expires At <span
                                    class="text-muted">(optional)</span></label>
                            <input type="datetime-local" class="form-control form-control-sm"
id="expires_at" name="expires_at">
```

```
                        <div class="invalid-feedback small" id="error-expires_at"></div>
                    </div>

                    <button type="submit" class="btn btn-primary btn-sm w-100">Create</button>
                </form>
            </div>
        </div>
    </div>
@endsection

@push('scripts')
    @vite('resources/js/shorten.js')
@endpush

// resources/views/home.blade.php

@extends('layouts.app')

@push('styles')
    @vite('resources/sass/pages/home.scss')
@endpush

@section('content')

    <section class="home-hero text-center text-white rounded-top-5 py-5" style="background:
linear-gradient(135deg, #20c997 0%, #fd7e14 100%);">
        <div class="container">
            <h1 class="display-4 fw-bold">{{ __('home.hero_title') }}</h1>
            <p class="lead mb-4">{{ __('home.hero_subtitle') }}</p>

            <form id="shorten-form" action="{{ route('home.shorten') }}" method="POST" class="row
g-2 justify-content-center">
                <div class="col-md-6">
                    <input type="url"
                           class="form-control form-control-lg"
                           id="originalUrl"
                           placeholder="{{ __('home.url_placeholder') }}"
                           required>
                </div>
                <div class="col-md-2">
                    <button type="submit" class="btn btn-warning btn-lg w-100">{{
__('home.shorten_button') }}</button>
                </div>
            </form>

            <div id="shortenedResult" class="mt-5 d-none">
                <div class="card border-0 shadow-lg mx-auto" style="max-width: 600px;">
                    <div class="card-body text-center">
                        <h5 class="card-title mb-3">{{ __('home.result_title') }}</h5>
```

```
                <div class="d-flex justify-content-center align-items-center">
                  <a href="#" id="shortUrl" target="_blank" class="h4 mb-0 text-success text-
decoration-none"></a>
                  <button id="copyBtn"
                        class="btn btn-outline-secondary ms-2"
                        title="{{ __('home.copy') }}">
                    <i class="bi bi-clipboard"></i>
                  </button>
                </div>
              </div>
            </div>
          </div>
        </section>

        <div class="section-divider"></div>

        <section class="features py-5 bg-light">
          <div class="container">
            <div class="row g-4 text-center">
              <div class="col-md-4">
                <div class="card h-100 border-0">
                  <div class="card-body">
                    <i class="bi bi-card-list display-4 mb-3 text-info"></i>
                    <h5 class="card-title">{{ __('home.feature_bulk_title') }}</h5>
                    <p class="card-text text-muted">{{ __('home.feature_bulk_desc') }}</p>
                  </div>
                </div>
              </div>
              <div class="col-md-4">
                <div class="card h-100 border-0">
                  <div class="card-body">
                    <i class="bi bi-graph-up display-4 mb-3 text-success"></i>
                    <h5 class="card-title">{{ __('home.feature_stats_title') }}</h5>
                    <p class="card-text text-muted">{{ __('home.feature_stats_desc') }}</p>
                  </div>
                </div>
              </div>
              <div class="col-md-4">
                <div class="card h-100 border-0">
                  <div class="card-body">
                    <i class="bi bi-lock display-4 mb-3 text-danger"></i>
                    <h5 class="card-title">{{ __('home.feature_protected_title') }}</h5>
                    <p class="card-text text-muted">{{ __('home.feature_protected_desc') }}</p>
                  </div>
                </div>
              </div>
            </div>
          </div>
```

```
</section>

<div class="section-divider"></div>

<section class="how-it-works py-5">
  <div class="container">
    <h2 class="text-center mb-4">{{ __('home.how_title') }}</h2>
    <div class="row text-center">
      <div class="col-md-4 mb-4">
        <i class="bi bi-clipboard-check display-4 mb-3 text-primary"></i>
        <h5>{{ __('home.how_step1_title') }}</h5>
        <p class="text-muted">{{ __('home.how_step1_desc') }}</p>
      </div>
      <div class="col-md-4 mb-4">
        <i class="bi bi-gear display-4 mb-3 text-secondary"></i>
        <h5>{{ __('home.how_step2_title') }}</h5>
        <p class="text-muted">{{ __('home.how_step2_desc') }}</p>
      </div>
      <div class="col-md-4 mb-4">
        <i class="bi bi-rocket display-4 mb-3 text-warning"></i>
        <h5>{{ __('home.how_step3_title') }}</h5>
        <p class="text-muted">{{ __('home.how_step3_desc') }}</p>
      </div>
    </div>
  </div>
</section>

<div class="section-divider"></div>

<section class="testimonials bg-light py-5">
  <div class="container">
    <h2 class="text-center mb-4">{{ __('home.testimonials_title') }}</h2>
    <div class="row justify-content-center">
      <div class="col-md-8">
        <div id="testimonialCarousel" class="carousel slide" data-bs-ride="carousel">
          <div class="carousel-inner">
            <div class="carousel-item active">
              <blockquote class="blockquote text-center">
                <p class="mb-4">“{{ __('home.testimonial1_text') }}”</p>
                <footer class="blockquote-footer">{{ __('home.testimonial1_author') }}</footer>
              </blockquote>
            </div>
            <div class="carousel-item">
              <blockquote class="blockquote text-center">
                <p class="mb-4">“{{ __('home.testimonial2_text') }}”</p>
                <footer class="blockquote-footer">{{ __('home.testimonial2_author') }}</footer>
              </blockquote>
            </div>
          </div>
```

```
                <button class="carousel-control-prev" type="button" data-bs-
target="#testimonialCarousel" data-bs-slide="prev">
                  <span class="carousel-control-prev-icon"></span>
                </button>
                <button class="carousel-control-next" type="button" data-bs-
target="#testimonialCarousel" data-bs-slide="next">
                  <span class="carousel-control-next-icon"></span>
                </button>
              </div>
            </div>
          </div>
        </div>
      </section>

      @endsection

      @push('scripts')
        @vite('resources/js/pages/home.js')
      @endpush

      // vite.config.js
      import { defineConfig } from 'vite';
      import laravel from 'laravel-vite-plugin';

      export default defineConfig({
        plugins: [
          laravel({
            input: [
              'resources/js/app.js',
              'resources/sass/app.scss',
              'resources/sass/pages/dashboard.scss',
              'resources/sass/pages/create.scss',
              'resources/js/pages/dashboard.js',
              'resources/js/shorten.js',
            ],
            refresh: true,
          }),
        ],
      });
```

Додаток Г. Ілюстративна частина

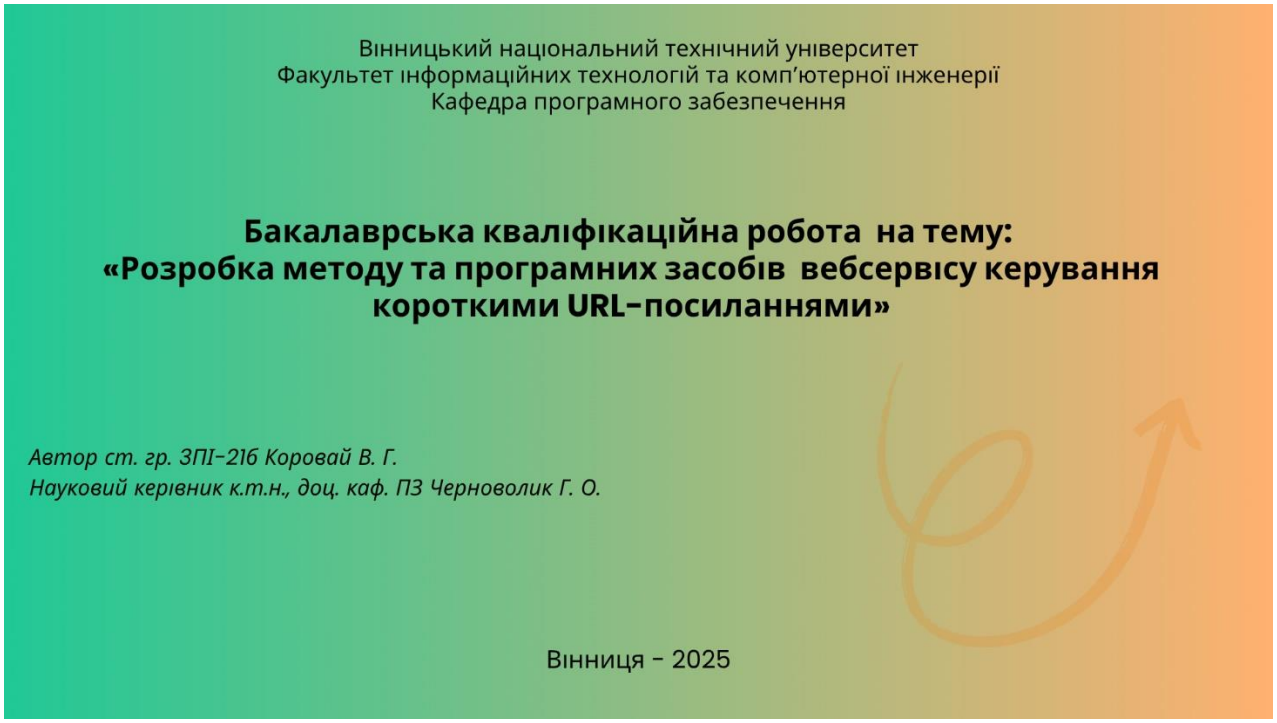

Рисунок Г.1 – Титульний слайд

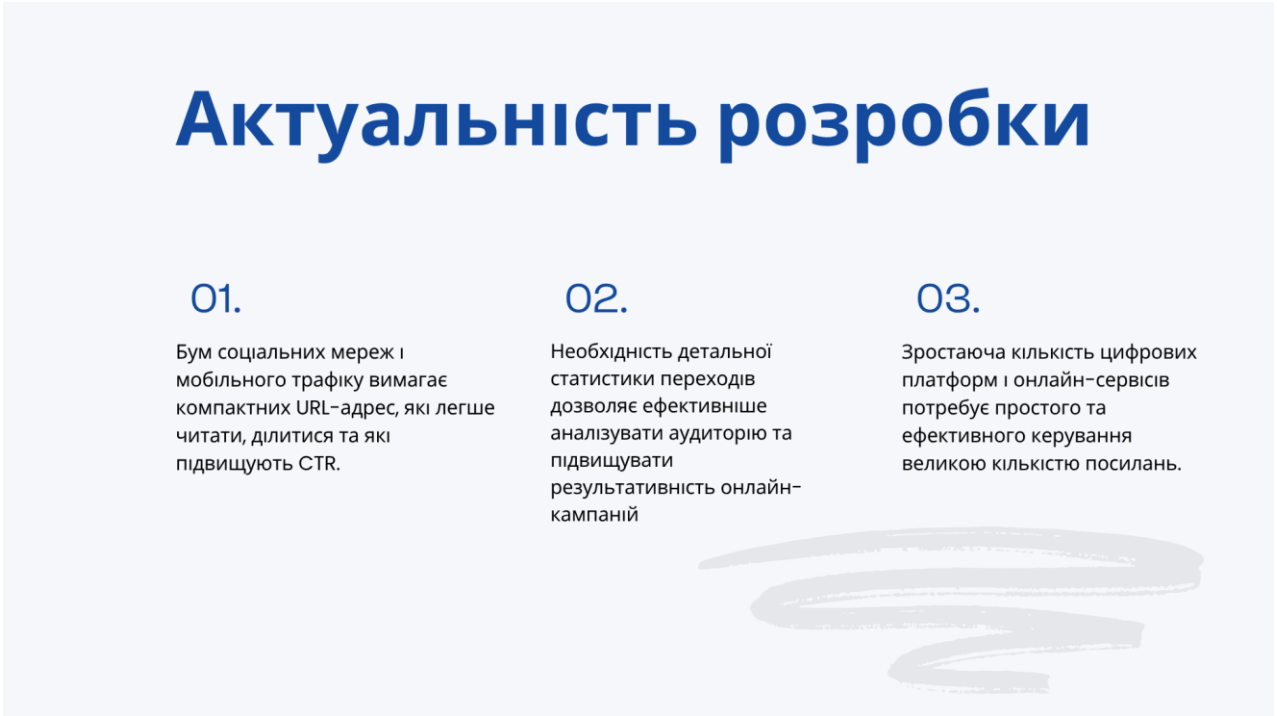

Рисунок Г.2 – Актуальність розробки

Рисунок Г.3 – Мета дослідження

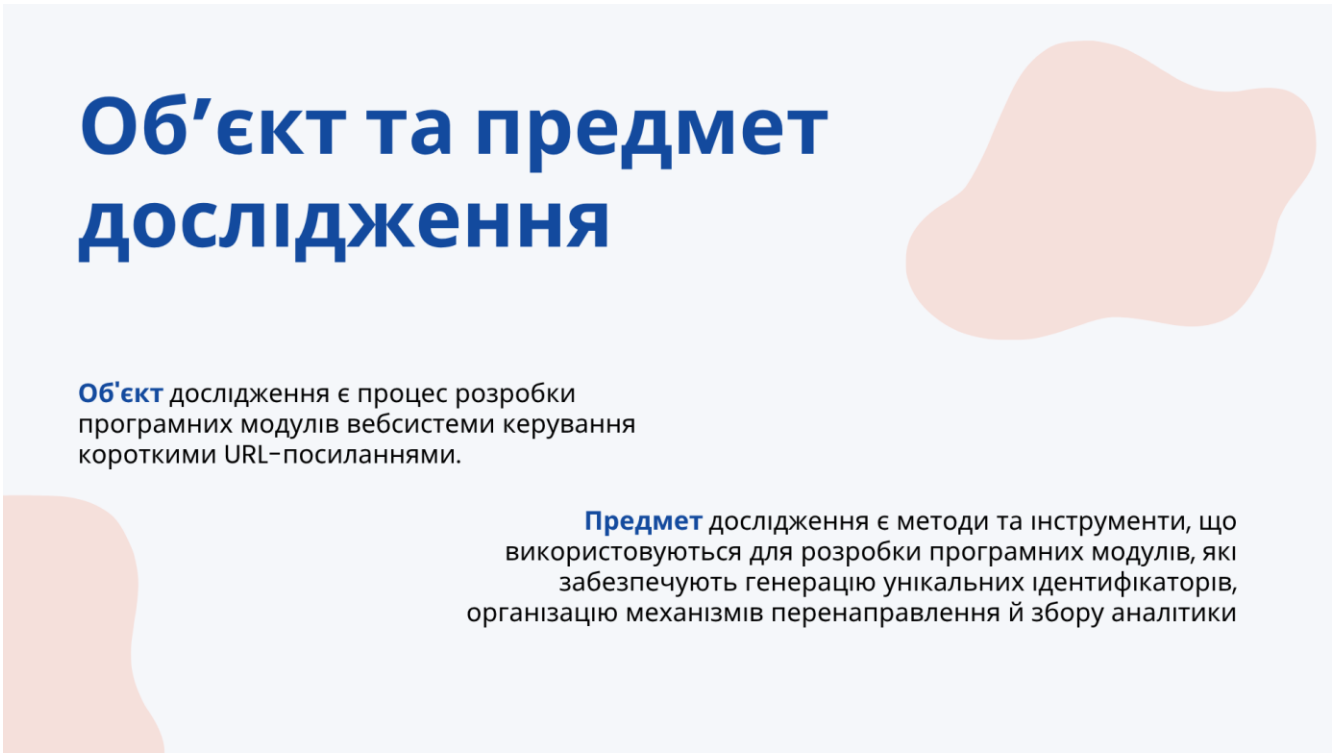

Рисунок Г.4 – Об’єкт та предмет дослідження

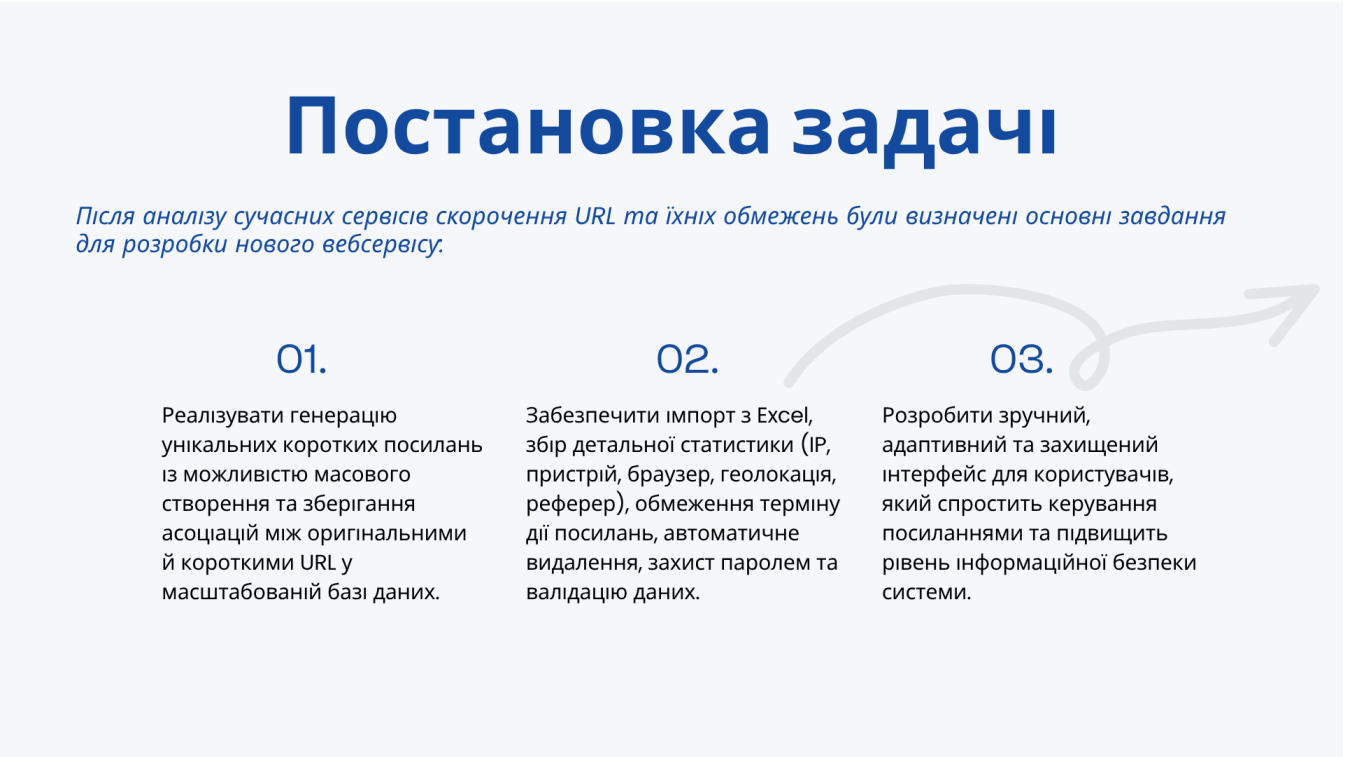

Рисунок Г.5 – Постановка задачі

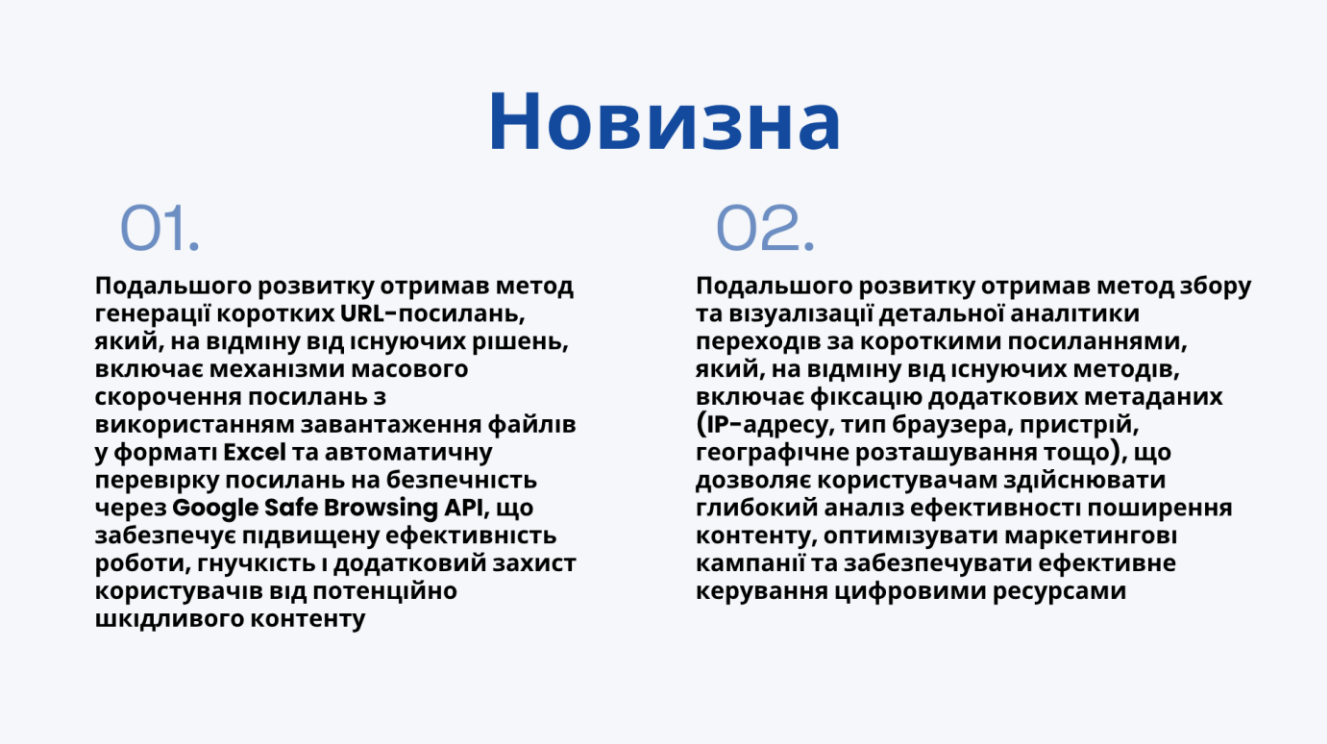

Рисунок Г.6 – Новизна розробленого застосунку

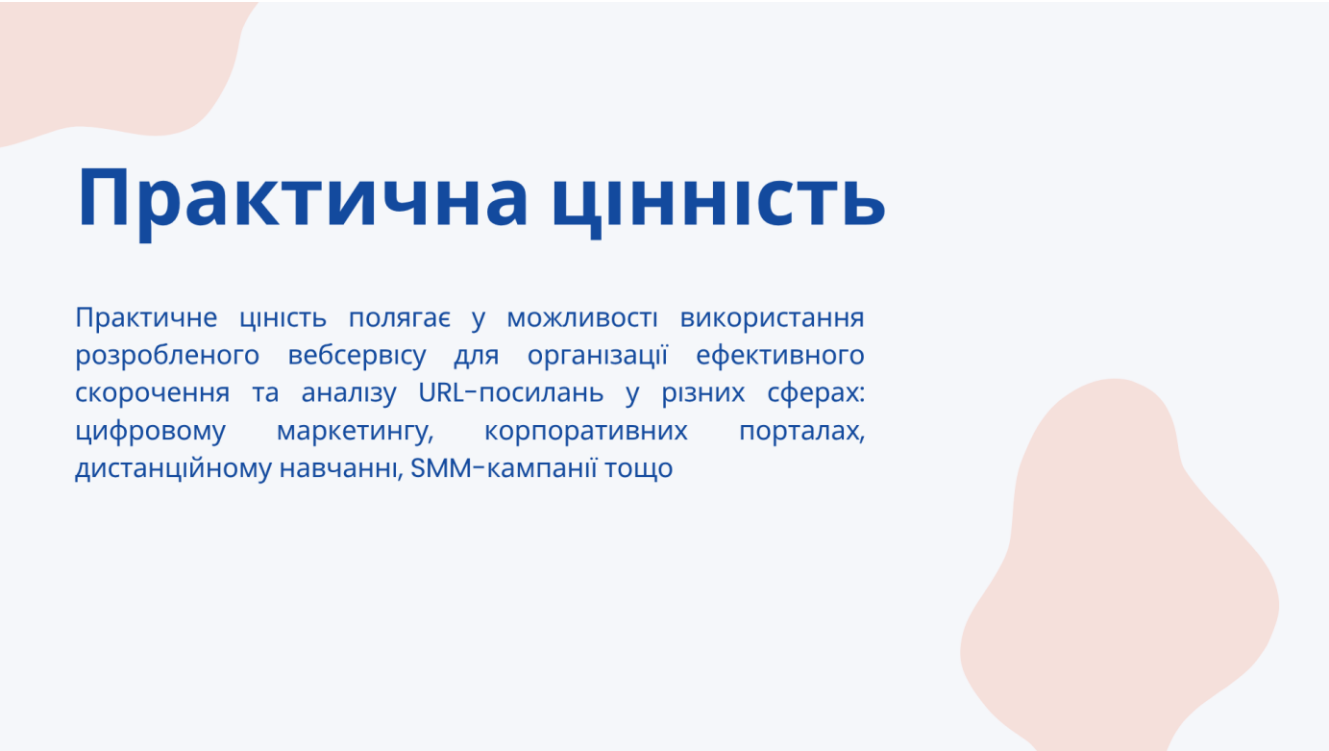

Рисунок Г.7 – Практична цінність

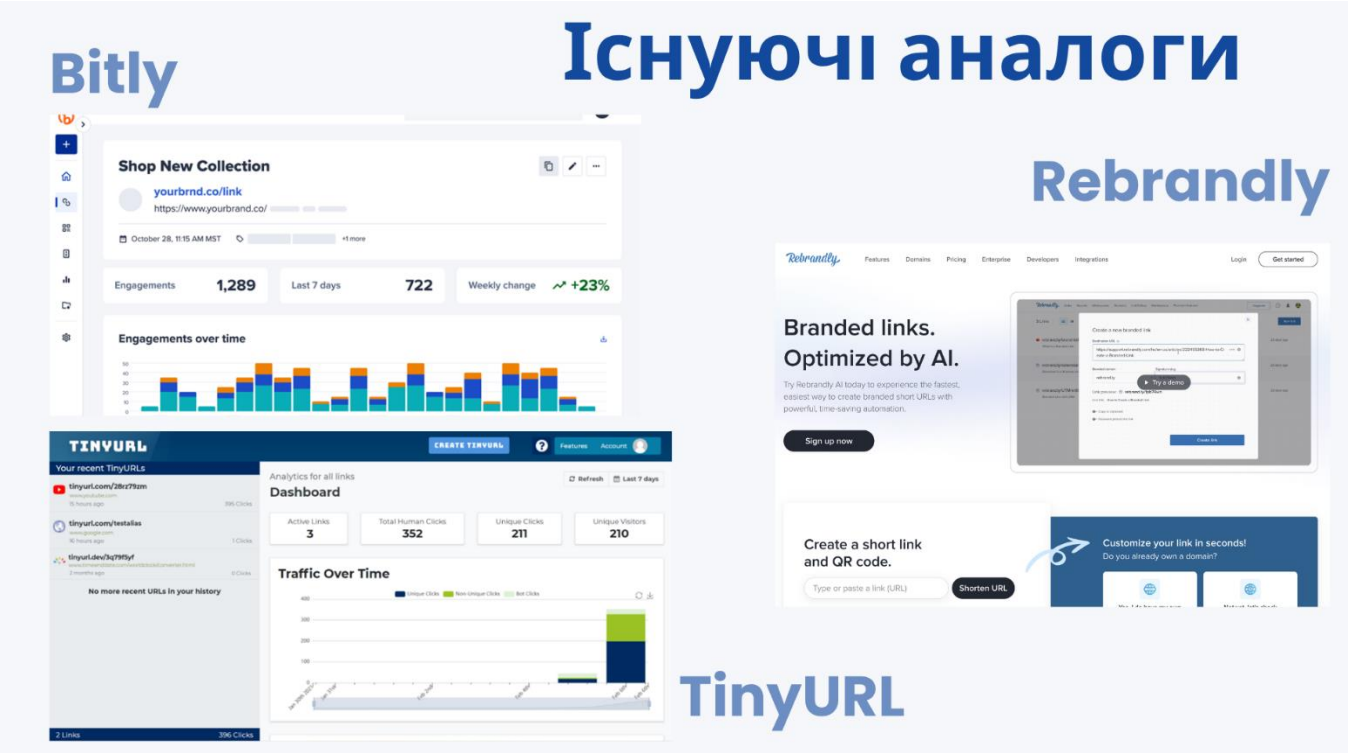

Рисунок Г.8 – Існуючі аналоги

Функціонал	Bitly	TinyURL	Rebrandly	Власна розробка
Базове скорочення посилань	+	+	+	+
Відстеження переходів	+	-	+	+
Масове скорочення посилань	+	+	-	+
API для інтеграції	+	+	+	+
Налаштування терміну дії посилань	-	-	+	+
Захист посилань паролем	-	-	+	+
Сумарний коефіцієнт	4	3	5	6

Рисунок Г.9 – Аналіз аналогів

Рисунок Г.10 – Використані технології

Рисунок Г.11 – Модель роботи системи

Рисунок Г.12 – Загальний алгоритм роботи системи

Рисунок Г.13 – Алгоритм роботи модуля скорочення посилань

Рисунок Г.14 – Алгоритм роботи масового скорочення URL-посилань

Рисунок Г.15 – Алгоритм роботи модуля переходів

Рисунок Г.16 – Діаграма діяльності системи

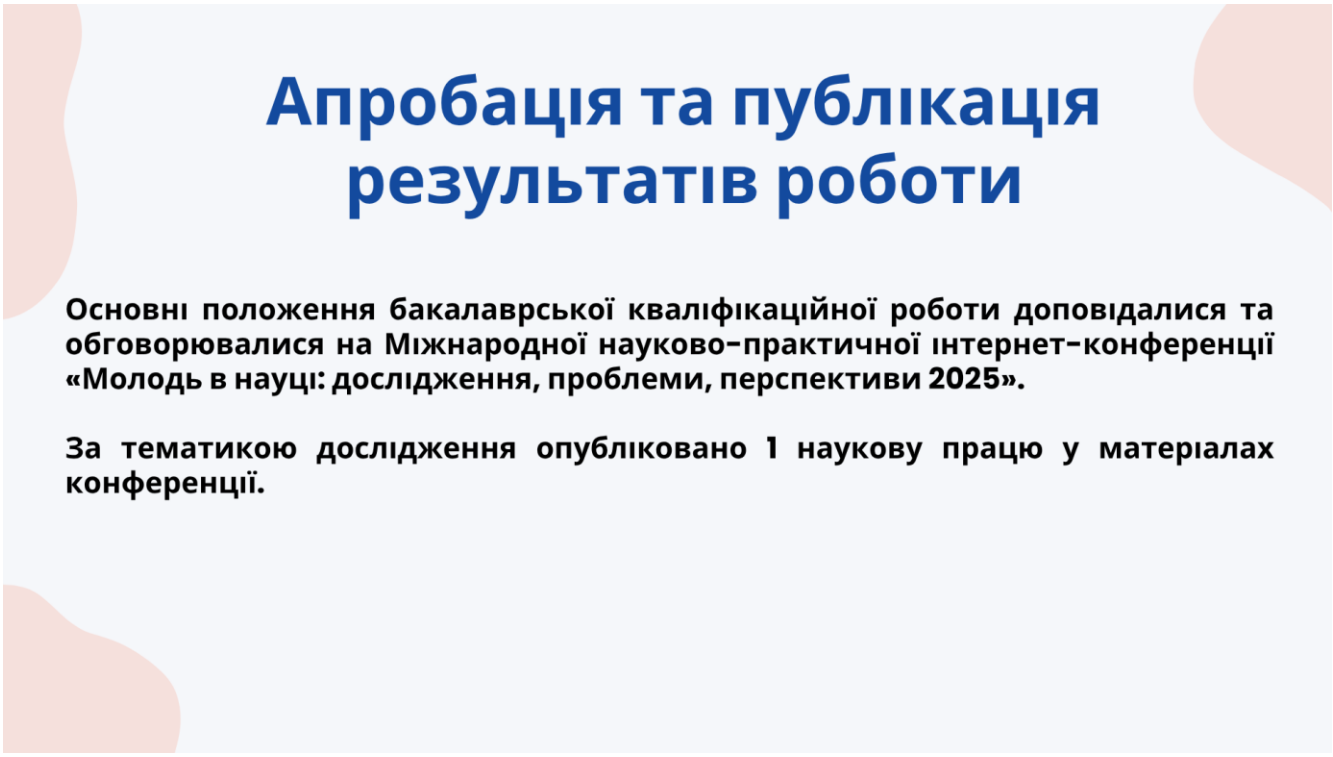

Рисунок Г.17 – Апробація та публікація результатів роботи

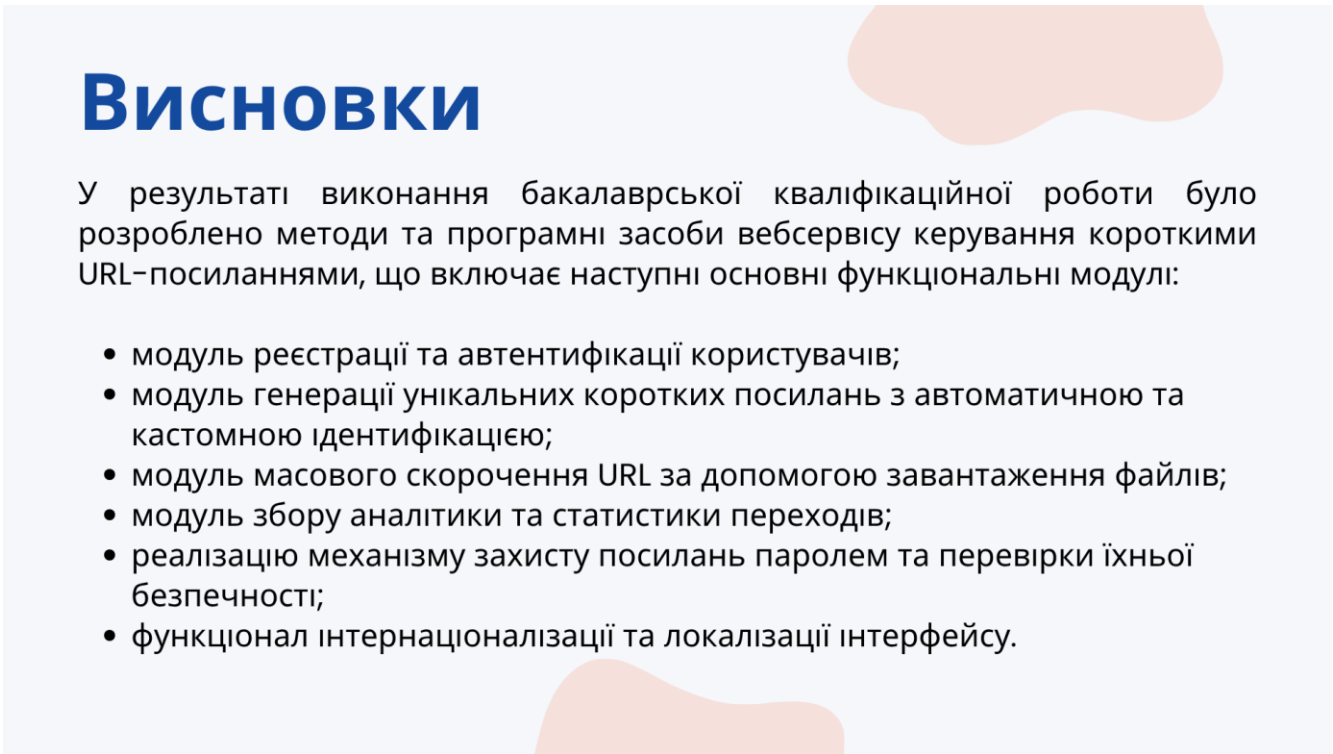

Рисунок Г.18 – Висновки

Рисунок Г.19 – Фінальний слайд