Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Чернова Сергія Володимировича

на здобуття ступеня вищої освіти Бакалавра

Система генерації пошукових запитів для виявлення вразливостей та витоків даних інтернет-ресурсів

Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітня програма Кібербезпека

Шифр КРБКБ.220165.22.01.08 ПЗ

Виконав студент 3 курсу група КБс-22-1 Сергій ЧЕРНОВ
Керівник канд. техн. наук, доцент Віктор ЧЕШУН
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ
8 06 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ _____

15 лютого 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Чернову Сергію Володимировичу

1 Тема роботи Система генерації пошукових запитів для виявлення вразливостей та витоків даних інтернет-ресурсів
Керівник роботи канд. техн. наук, доцент, Чешун Віктор Миколайович
Затверджено наказом ректора університету від 15 лютого 2025 № 8

2 Строк подання студентом кваліфікаційної роботи на кафедру 1 червня 2025р.

3 Вихідні дані до роботи Проєкт розроблявся у вигляді веб-додатку з використанням ASP.NET Core MVC. Система працює з JSON файлами, що містять шаблони пошукових запитів, призначених для виявлення конфіденційної або критично важливої інформації. Система включає механізми побудови, фільтрації, автоматизації запитів та класифікації результатів із використанням машинного навчання.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області, Порівняння існуючих рішень, Вивчення особливостей пошукових операторів Google, Формування та класифікація шаблонів запитів, Проєктування архітектури системи, Побудова інтерфейсу користувача, Реалізація сервісів пошуку, генерації та фільтрації, Розробка модуля класифікації результатів, Побудова датасету і навчання моделі.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Схема алгоритму взаємодії користувача із системою. Схема алгоритму пошуку через Google API. Схема застосованих технологій у вигляді архітектури.

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7 Дата видачі завдання 16 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	виконав
Ознайомлення з предметною областю	Лютий	виконав
Дослідження існуючих рішень	Лютий	виконав
Постановка задачі	Березень	виконав
Визначення загальних принципів рішення задачі	Березень	виконав
Деталізація принципів рішення задачі	Квітень	виконав
Розробка проєктних рішень	Квітень	виконав
Апробація проєктних рішень	Травень	виконав
Оформлення пояснювальної записки згідно вимог	Травень	виконав
Оформлення графічної частини	Червень	виконав
Захист КР	Червень	виконав

Студент Сергій ЧЕРНОВ

Керівник кваліфікаційної роботи Віктор ЧЕШУН

АНОТАЦІЯ

Тема кваліфікаційної роботи: Система генерації пошукових запитів для виявлення вразливостей або витоків даних інтернет-ресурсів.

Автор роботи: Чернов Сергій Володимирович.

Керівник роботи: Чешун Віктор Миколайович.

Пояснювальна записка: 69 с., 2 додатки, 38 рисунків, 3 таблиць, 40 джерел.

Графічна частина: 3 плакати, 12 презентаційних слайдів.

OSINT, Google Dorks, пошукові запити, класифікація результатів, C#, ASP.NET, JSON, API.

Кваліфікаційна робота бакалавра присвячена розробці програмного засобу AbyssDorks – системи автоматизованої генерації пошукових запитів, орієнтованої на виявлення потенційних витоків даних або вразливостей, які знаходяться у вільному доступі в інтернеті.

У роботі проведено аналіз сучасних підходів до OSINT-моніторингу, досліджено можливості Google Dorks, інструментів типу Shodan, FOCA, Censys, а також описано типові загрози, пов'язані з відкритими конфігураційними файлами, документами, паролями та метаданими. Виявлено основні обмеження існуючих рішень і сформульовану технічні вимоги до власного програмного засобу. Реалізовано веб-додаток на платформі ASP.NET Core із підтримкою Google Custom Search API, системою тегів, шаблонів і модулів, що дозволяє користувачеві швидко сформувати релевантні dork-запити. Система підтримує збереження шаблонів, класифікацію результатів за критичністю, обробку API-ключів, а також інтерфейс із зручним дизайном.

01.06.2025

ABSTRACT

Subject of qualification work: Search query generation system for detecting vulnerabilities or data leaks on internet resources.

Author: Chernov Serhii Volodymyrovych.

Head of work: Cheshun Viktor Mykolaiovych.

Explanatory note: 69 p., 2 appendices, 38 figures, 3 tables, 40 sources

Graphic part: 3 posters, 12 presentation slides.

OSINT, Google Dorks, search queries, result classification, C#, ASP.NET, JSON, API.

The bachelor's qualification work is dedicated to the development of AbyssDorks, a software system for the automated generation of search queries aimed at identifying potential data leaks or vulnerabilities publicly accessible on the Internet.

The work analyzes modern OSINT monitoring methods, explores the capabilities of Google Dorks and tools such as Shodan, FOCA and Censys, and describes typical threats related to explores configuration files, documents, passwords, and metadata. Based on this the limitations of existing solutions were identified and technical requirements for a custom system were formulated. The web application was implemented using the ASP.NET Core framework, with integrated support for Google Custom Search API a system of tags, templates and modules that enables users to quickly generate relevant dork queries. The system support saving templates, classifying results by severity, processing API keys and offers a user-friendly interface.

01.06.2025

ЗМІСТ

ВСТУП

Сучасний світ дедалі більше покладається на цифрові технології, що породжує нові можливості для розвитку але й в той же час створює серйозні загрози для кібербезпеки. З кожним днем кількість витоків даних, кібератак та експлуатації вразливостей у системах стрімко зростає. За даними аналітичних агенств, щороку фіксується мільйони інцидентів, пов’язаних з несанкціонованим доступом до конфіденційної інформації, що ставить під загрозу як окремих осіб так і великі компанії.

Одним із важливих аспектів забезпечення інформаційної безпеки є виявлення вразливостей у публічно доступних ресурсах, таких як вебсайти, сервери, бази даних та інші об’єкти цифрової інфраструктури. При цьому значна частина вразливостей може бути виявлена за допомогою правильно складених пошукових запитів у таких пошукових системах як от Google. Такі запити, відомі як Google Dorks. Вони дозволяють знаходити відкриті адмін-панелі, незахищені файли, бази даних, конфігураційні файли тощо.

Однак ефективне використання Google Dorks вимагає спеціальних знань та досвіду, що обмежує їх застосування для широкого кола фахівців. Більше того, автоматизація процесу складання пошукових запитів може значно підвищити ефективність виявлення вразливостей. Звісно, варто згадати про враховування етичних і правових аспектів використання таких технологій.

Метою даної роботи є розробка системи автоматизації генерації пошукових запитів для виявлення вразливостей або витоків даних в інтернеті. Така система дозволить спростити та пришвидшити процес складання запитів, підвищуючи їх точність та релевантність.

Об’єктом дослідження є технології автоматизації пошуку вразливостей у публічно доступних цифрових ресурсах. Предметом дослідження виступає використання пошукових операторів Google для складання запитів, спрямованих на виявлення конфіденційної інформації що може бути неправомірно доступною.

Для досягнення поставлених цілей було використано такі методи дослідження, як аналіз існуючої літератури та наукових джерел, для вивчення підходів та інструментів у сфері кібербезпеки. Експериментальним методом виступить тестування розробленої системи на реальних сценаріях її використання. Також скористаємось моделюванням для створення алгоритмів автоматизації пошукових запитів, а потім займемось програмною реалізацією для розробки функціональних модулів системи.

Розроблена система може бути корисною для фахівців з кібербезпеки, пентестерів, дослідників та організацій які займаються захистом своїх цифрових ресурсів. Вона також може стати платформою для подальших досліджень у сфері автоматизації та інтелектуального аналізу даних.

1 ТЕОРЕТИЧНИЙ АНАЛІЗ ПРОБЛЕМИ ВИТОКІВ ДАНИХ ТА МЕТОДІВ ЇХ ПОШУКУ

1.1 Витоки даних і пов’язані з ними загрози

Вплив витоків даних на людей, компанії або цілі організації набуває дедалі більшої актуальності в сучасному світі. У добу цифровізації майже всі послуги, товари та транзакції пов’язані з використанням інтернету, що значно збільшує ризик втрати конфіденційної інформації. Зокрема, користувачі стикаються з можливістю розкриття своїх персональних даних, таких як ім’я, прізвище, дата народження, номери телефонів, електронні пошти, адреси проживання та навіть банківські реквізити. Для компаній і організацій ризики ще більші, оскільки витоки можуть включати комерційну інформацію, інтелектуальну власність, фінансові звіти та інформацію про клієнтів, створюючи загрози на багатьох рівнях. Витоки даних слугують каталізатором шантажу, фінансових шахрайств або крадіжок особистості, особливо коли втрачені дані потрапляють на чорні ринки. Крім фінансових втрат, такі інциденти часто наносять серйозний удар по довірі користувачів до цифрових сервісів. У результаті, компанії вимушені не лише інвестувати в захист, але й відновлювати свою репутацію через різні маркетингові кампанії та покращення сервісів підтримки.

Фінансові втрати є одним із найбільш відчутних наслідків витоків даних. Згідно з даними IBM Cost of Data Breach Report 2023 [1], середня вартість одного витоку даних досягла 4,45 мільйона доларів США, що на 2,3% більше ніж у 2022 році [2]. Ці втрати включають широкий спектр витрат, сюди відносяться витрати на компенсації постраждалим клієнтам, витрати на інцидент-реагування, зокрема створення спеціалізованих команд для розслідування інциденту та інвестиції в нові засоби кіберзахисту, що включають впровадження систем виявлення загроз, багатофакторну автентифікацію та оновлення інфраструктури безпеки. Також існують юридичні витрати та штрафи за порушення таких нормативів, як GDPR (Загальний регламент захисту даних) [3], CCPA (Каліфорнійський закон про конфіденційність споживачів) [4] та інших. Як приклад, у 2023 році ірландська

Комісія із захисту даних наклала рекордний штраф у розмірі 1,2 мільярда євро на технологічного гіганта Meta за неналежне управління персональними даними [5]. Ще один приклад – компанія Yahoo, чий витік даних у 2013 році став відомий у 2016 році та призвів до зменшення ціни її придбання на $350 мільйонів [6]. Витрати на інцидент-менеджмент можуть складати до 30% загальної суми втрат, особливо в компаніях із високою цифровою залежністю. Крім штрафів, організації зіштовхуються з необхідністю проходження повторних аудитів та перегляду політик безпеки. У багатьох випадках втрата даних змушує компанії призупиняти бізнес-процеси, що призводить до додаткових збитків та втрати ринку.

Репутаційні збитки після витоку даних можуть бути довготривалий період руйнівними для бізнесу, незалежно від його масштабу чи галузі. Дослідження демонструють, що до третини клієнтів у таких секторах, як роздрібна торгівля, фінанси та охорона здоров'я, припиняють співпрацю з компаніями, які стали жертвами витоку даних. Це свідчить про високий рівень недовіри споживачів до організацій, які не змогли забезпечити належний захист їхньої конфіденційної інформації. Для бізнесу це означає не лише втрату доходів, але й значне зменшення клієнтської бази. Ще більш тривожним є той факт, що 85% постраждалих клієнтів діляться своїм негативним досвідом із друзями, колегами або в соціальних мережах. У цифрову епоху такі відгуки здатні миттєво поширюватися, завдаючи шкоди репутації компанії на глобальному рівні. Близько 33,5% клієнтів активно висловлюють своє обурення у соціальних мережах, створюючи додатковий тиск на організацію та посилюючи кризу. Наслідки репутаційного характеру не обмежуються лише втратою поточних клієнтів. Вони також впливають на здатність компанії залучати інвесторів та наймати кваліфікованих співробітників. Інвестори можуть сумніватися у фінансовій стабільності бізнесу, а потенційні співробітники – у його професійній етиці та безпеці робочого середовища. Відновлення довіри клієнтів зазвичай потребує тривалого часу та значних зусиль у сфері PR та комунікацій. Компаніям доводиться в такому разі запускати додаткові кампанії з прозорості,

запроваджувати політики відкритості та залучати зовнішніх експертів для аудиту безпеки. У деяких випадках, особливо у сфері фінансових послуг, репутаційні наслідки можуть бути настільки серйозними, що призводять до втрати ліцензії або регуляторного тиску.

Витік даних також значно порушує бізнес-операції. Організаціям необхідно проводити масштабні розслідування щоб з’ясувати як саме відбувся витік. Згідно з дослідженнями IBM, середній час виявлення витоку даних становить 277 днів. Це означає, що організація може не знати про проблему майже дев'ять місяців, поки інформація не стане доступною для сторонніх осіб. Протягом цього часу зловмисники можуть використовувати отримані дані для шахрайства, що завдає ще більшої шкоди репутації компанії. Процес реагування на витік починається з масштабного розслідування. Організаціям необхідно визначити джерело проблеми, виявити, які саме дані були скомпрометовані, і оцінити масштаби загрози. Цей етап часто супроводжується частковим або повним призупиненням бізнес-операцій для запобігання подальшому поширенню витоку. Після виявлення причин інциденту компанія повинна впровадити заходи відновлення. Це може включати оновлення програмного забезпечення, посилення системи кібербезпеки, навчання працівників та перегляд політик зберігання даних. На реалізацію цих заходів може знадобитися кілька тижнів або навіть місяців. Деякі компанії змушені тимчасово відключати частину своїх сервісів, щоб уникнути повторного витоку або доступу до скомпроментованих систем. Такий простій бізнес-процесів часто веде до утрати клієнтів, порушення договорів і навіть до штрафних санкцій з боку партнерів. У довгостроковій перспективі організація може втратити конкурентну перевагу, якщо її основні сервіси вийдуть з ладу на тривалий час.

Так як існує закон, організації повинні дотримуватись норм щодо захисту даних. З юридичної точки зору постраждалі особи мають право звертатися до суду з метою захисту своїх прав. Прецеденти свідчать про те, що кількість судових позовів у справах, пов’язаних із витоками даних, стрімко зростає. Наприклад, витік медичних записів не лише порушує право пацієнта на

конфіденційність, але й може негативно вплинути на процес його лікування. Розголошення такої інформації може стати причиною дискримінації або навіть помилок у медичних діагнозах. Окрім того, втрата біометричних даних, таких як відбитки пальців або скани обличчя, становить особливу небезпеку. Ці дані мають високу цінність на чорному ринку і можуть бути використані у злочинних схемах, зокрема для крадіжки особистості чи доступу до фінансових ресурсів [7]. Витрати на судові процеси та юридичний супровід можуть сягати мільйонів доларів, особливо у випадках колективних позовів. У деяких країнах регулятори навіть мають право тимчасово призупинити діяльність компанії або відкликати її ліцензію. Таким чином, недотримання вимог законодавства у сфері захисту даних становить не лише фінансову, але й екзистенційну загрозу для бізнесу.

Усе вище перераховане робить проблему витоків однією з найбільш нагальних у сфері кібербезпеки. Щоб ефективно протидіяти цим загрозам, необхідно впроваджувати сучасні системи виявлення та запобігання витокам даних. Одним із ефективних інструментів що дозволяє аналізувати відкриті джерела в інтернеті для ідентифікації вразливостей є Google dorks [8]. Ця методика дозволяє автоматизувати процес пошуку відкритих даних, надаючи дослідникам та адміністраторам потужний інструмент для моніторингу інформаційної безпеки. Саме через це, на мою думку, і виникає потреба у розробці спеціалізованих систем, які б використовували сучасні технології для генерації релевантних пошукових запитів та забезпечували високий рівень ефективності в процесах кіберзахисту. Така система може стати ключовим елементом у стратегічному підході до запобігання витокам даних та забезпечення інформаційної безпеки як для окремих організацій так і для суспільства у цілому. Зокрема, автоматизація таких процесів дозволятиме зменшити вплив людського фактору та уникнути помилок при формуванні запитів. У поєднанні з машинним навчанням, така система зможе аналізувати й класифікувати результати пошуку, що дає змогу виділяти дійсно критичні інциденти. Це особливо важливо в умовах зростаючого інформаційного шуму та обмежених ресурсів кіберкоманд. У майбутньому подібні платформи можуть

стати основою для інтегрованих систем кіберзахисту в рамках національної або корпоративної стратегії.

1.2 Google Dorks як інструмент розвідки

Багато користувачів інтернету щодня звертаються до пошукових систем для отримання необхідної інформації. Проте, більшість із них обмежуються введенням простих ключових слів, не використовуючи додаткові інструменти для оптимізації пошуку. Одним із таких потужних інструментів є Google Dorks – набір спеціальних команд, які дозволяють значно покращити точність та релевантність результатів. Google Dorks базується на використанні операторів – ключових слів або символів, які модифікують запит і впливають на спосіб обробки інформації пошуковою системою. Наприклад, якщо користувач шукає інформацію про університети в Україні, він може ввести запит “ХНУ AND КНУ”. У цьому випадку Google покаже лише ті результати, які містять обидва ключові слова одночасно. Важливо зазначити, що Google Dorks є чутливим до регістру, тому написання запиту як “ХНУ and КНУ” може дати зовсім інші результати.

Розглянемо на рисунку 1.1 як працюють ці дорки на кількох базових операторах, щоб побачити як досягаються конкретні результати:

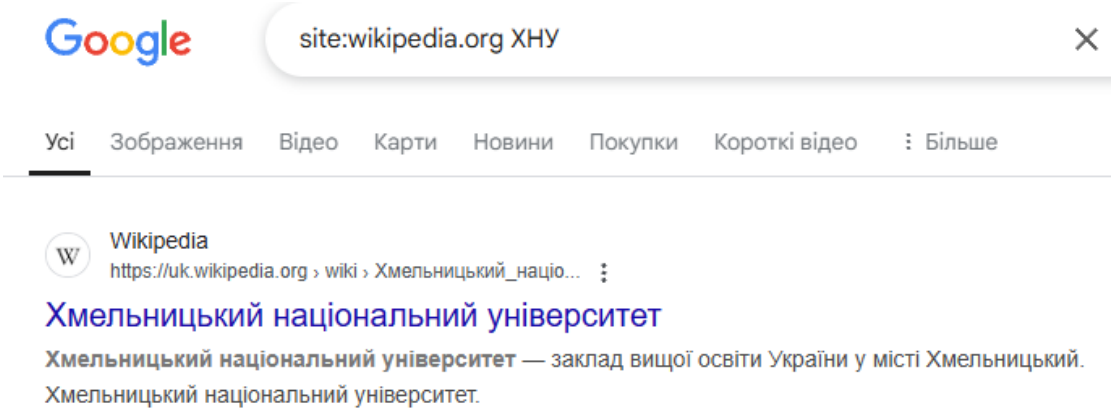


Рисунок 1.1 – Приклад оператора “site”

Оператор "site" є потужним інструментом для оптимізації пошуку в Google, який дозволяє отримувати результати виключно з конкретного вебсайту. Це особливо корисно для тих, хто шукає інформацію на певному ресурсі, не витрачаючи час на перегляд нерелевантних сторінок. Синтаксис використання цього оператора досить простий: у пошуковому рядку Google необхідно ввести слово "site:", за яким слідує адреса потрібного сайту, а далі – ключові слова вашого запиту. У сфері кібербезпеки цей оператор один із ключових при ручному аудиті сайту на наявність вразливих або відкритих документів. Наприклад, через нього можна легко виявити відкриті конфігураційні файли або резервні копії, що випадково залишились на сервері.

Один із базових операторів – це “-” (мінус). Його функція полягає у виключенні певних результатів із пошукової видачі. Наприклад, якщо потрібно знайти інформацію на тему, але виключити посилання на конкретний сайт, цей оператор стане незамінним. На рисунку 1.2 розглянемо ситуацію, коли необхідно знайти матеріали в Інтернеті, але виключити будь-які результати з сайту Вікіпедія. Для цього у пошуковому запиті можна використати комбінацію операторів. Такий запит дозволяє отримати інформацію з інших джерел, виключивши дані з Вікіпедії. Це може бути корисним, коли потрібні більш спеціалізовані або першоджерельні матеріали. [9]

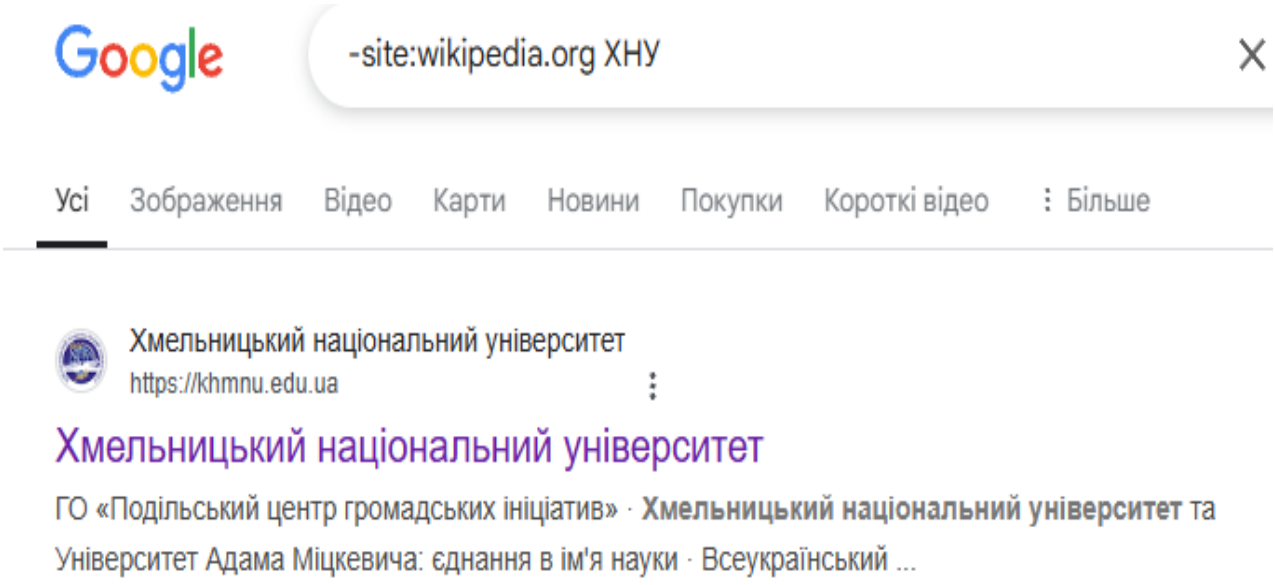


Рисунок 1.2 – Приклад комбінації двох операторів

Хоча Google dorks можуть бути корисним для будь якого користувача, його основна цінність полягає у відкритих розслідуваннях, особливо у пошуку інформації про конкретних осіб або організації. Одним із прикладів є пошук за іменем користувача. Наприклад, введення запиту “BadGuy1” у пошукову систему Google може показати згадки цього псевдоніма на різних вебресурсах. Якщо потрібно знайти електронні адреси, які використовують цей псевдонім, можна застосувати запит із символом підстановки, наприклад “BadGuy1*com” [10]. Це дозволить ідентифікувати адреси електронної пошти, пов’язані з цим ім’ям.

Для пошуку інформації про конкретну особу, наприклад, людину з іменем Петро Петрович, можна використати запит “Петро Петрович filetype:pdf | filetype:xlsx” (рисунок 1.3). Цей запит обмежує результати лише документами у форматах PDF та Excel, які містять точне ім’я та по батькові. Комбінація таких операторів може допомогти виявити критичну інформацію в різних OSINT-дослідженнях. Наприклад, можна не лише знайти наявність імені у файлах, а й оцінити його контекст: участь в проектах, публічні виступи або згадування в офіційних звітах. Ще такий метод є особливо корисним для виявлення офіційних документів, таких як судові записи, резюме чи звіти. Вони можуть містити важливі деталі про людину або її діяльність.

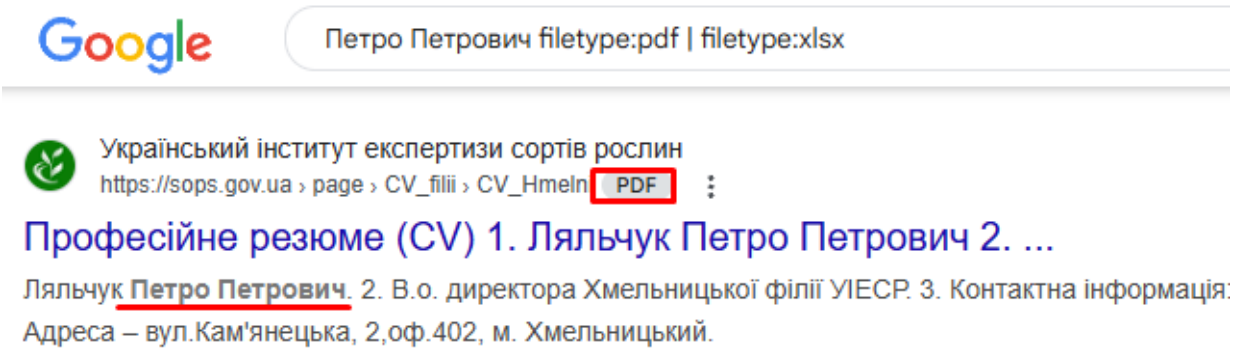


Рисунок 1.3 – Результати пошуку з операторами

Соціальні мережі є в тому числі багатим джерелом інформації про людей, компанії або мережі. Одним із ефективних способів пошуку інформації є

використання спеціальних пошукових запитів. Наприклад, щоб знайти дані, пов’язані з Хмельницьким національним університетом, можна скористатися запитом: “ХНУ site:Instagram.com”. Цей оператор обмежує пошук лише Інстаграмом і допомагає швидко знаходити релевантний контент, пов’язаний із нашим закладом. Для розширення пошуку на кілька платформ одночасно застосовується логічний оператор “OR”. Наприклад, запит “ХНУ site:Instagram.com OR site:Facebook.com” дозволить отримати результати як з Instagram, так і з Facebook (рисунок 1.4). Це значно підвищує ефективність пошуку, особливо якщо потрібна інформація може бути розміщена на різних соціальних мережах. І найголовніше, що такий підхід відкриває можливість моніторингу репутації установ, збору публічної інформації про події, заходи та їх учасників. А це особливо корисно, якщо ми до прикладу хочемо виявити неофіційні згадки про компанії або персони, які не фігурують у формальних звітах.

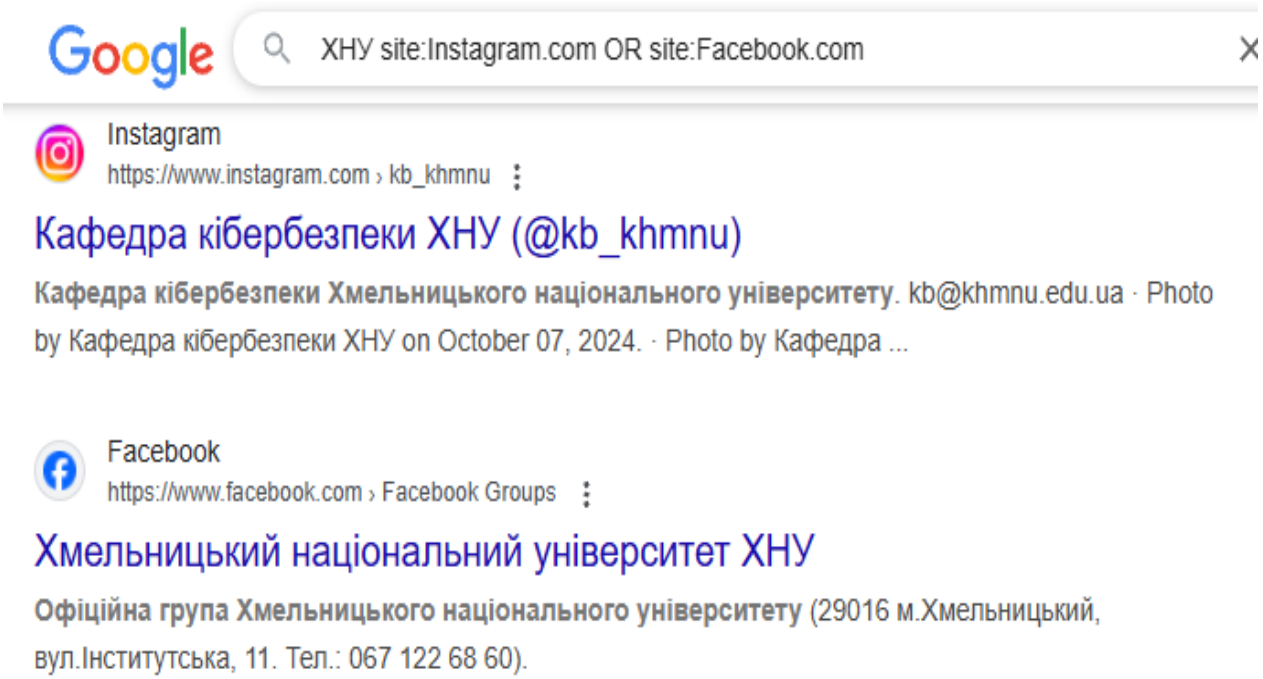


Рисунок 1.4 – Пошук інформації про ХНУ через соціальні мережі

Google Dorks є потужним інструментом який дозволяє суттєво покращити процес розслідування як у кібербезпеці так і в інших сферах. Завдяки

використанню комбінацій операторів можна отримати доступ до інформації різного характеру, яку важко віднайти традиційними способами. Так звані “дорки” були вперше популяризовані Джонні Лонгом, відомим дослідником з кібербезпеки. У 2002 році він почав досліджувати як можна використовувати пошукові оператори Google для ефективнішого збору інформації. З часом цей чоловік заснував нову концепцію – “Google Hacking”. Простими словами, це техніка яка дозволяє шукати і знаходити конфіденційну інформацію, неправильно налаштовані сервери, відкриті БД та інші вразливості за допомогою пошукової системи Google.

Але існує і “темна” сторона цього методу пошуку, яку використовують хакери. Ще у 2011 році дослідники з безпеки почали помічати, як хакери обходять ліміт Google на запити. Тоді центр захисту додатків (ADC) від Imperva відстежив конкретну бот-мережу, яка була використана проти відомого провайдера пошукових систем. Спостереження показали, як нападники закладають основу для спрощення та автоматизації наступних етапів атаки на веб-додатки. У своєму звіті дослідники зазначили, що хакери можуть генерувати понад 80 000 запитів щодня для сканування Інтернету в пошуку вразливих веб-додатків. Це відбувається за рахунок ботів для автоматизації процесу та обходу методів виявлення автоматизації, які зазвичай використовують провайдери пошукових мереж. Хакери створили ілюзію того, що індивідуальні користувачі виконують звичайні пошукові запити. Реальність ж полягає в тому, що вони проводять масштабну кіберрозвідку. Збір інформації про організацію може підготувати хакерів до створення атаки, орієнтованої на конкретний додаток. Спеціалізована експлуатація вразливостей може привести до зараження веб-сайтів, крадіжки даних, їх модифікації або навіть компрометації серверів компанії. Вище ми вже згадали про деякі оператори; хакери їх також активно використовують. Крім них, ще є безліч інших, наприклад, для атаки орієнтованої на всіх потенційних жертв у конкретній географічній локації (наприклад, за країною) запит включатиме оператор пошуку за локацією. Інший сценарій: хакер може захотіти націлитись на всі вразливості або витоки конкретного вебсайту,

використовуючи вище згаданий “site”. Були відомі випадки, коли хакери за допомогою Google Dorks знаходили не лише вразливості в маловідомих веб-ресурсах, але й у державних платформах та міжнародних корпораціях. Цей факт остаточно підкреслює важливість належного налаштування доступу до даних і видимості конфіденційної інформації в індексі Google. Окремі держави навіть забороняють використання Google Dorks без спеціального дозволу в рамках закону про комп’ютерні злочини.

Автоматизація запитів і аналіз результатів дозволяє зловмиснику виконувати велику кількість запитів, перевіряти всі результати і отримувати відфільтрований список потенційно експлуатованих сайтів дуже швидко та з мінімальними зусиллями. Для блокування автоматизованих пошукових кампаній, сучасні пошукові системи впроваджують механізми виявлення, які базуються на IP-адресі запиту. З дослідження Imperva [11] можна зрозуміти які етапи має атака хакерів. З початку вони орендують бот-мережі у ферм ботів, які мають глобальну мережу скомпрометованих пристроїв, потім завантажуються інструменти для координованого розподіленого пошуку на пристрої бот-мережі (зазвичай, саме вони містять базу даних “дорків”). Автоматизована інфраструктура хакерів контролює розподіл запитів і аналіз результатів серед частин бот-мережі. Після отримання списку потенційних цілей хакери можуть використовувати готові експлойти або створювати скрипти для атаки, спрямовані на вразливі ресурси.

1.3 Аналіз існуючих рішень

Щоб знаходити вразливості або потенційні місця витоків даних дослідники безпеки використовують різні пошукові системи в Інтернеті. Серед існуючих я б виділив три найпопулярніших – це Shodan, FOCA і Censys [12]. Такі сервіси – “must have” у портфелі інструментів кожного фахівця з інформаційної безпеки, адже забезпечують можливість аналізу публічно доступної інформації на

глибшому рівні аніж традиційні пошукові системи. Кожна з них має свої переваги, залежно від цілей розвідки – від технічного аналізу мережевих пристроїв до витягування метаданих із документів.

Конкретно ці інструменти займаються скануванням і індексацією живих хостів в інтернеті, також надають можливість шукати любі сервіси які запущені на цих хостах. Наприклад, можна виявити нові хости, які недавно додані в мережу організації або випадково виставлені хости, котрі не повинні бути доступні в Інтернеті.

Оператори пошуку дозволяють фільтрувати результати, зосереджуючи увагу лише на релевантних хостах і пристроях. У випадку, якщо дослідник прагне швидко ідентифікувати критичні вузли інфраструктури серед значного обсягу сирих даних це особливо корисно, до того ж, у комбінації з візуалізацією мережевих зв'язків це дає змогу оперативно оцінити потенційну площу атаки. Це дозволяє зменшити кількість нерелевантної інформації й зосередитися на цільових організаціях, системах або пристроях. Оператор "org" повертає результати, що відповідають конкретній організації, цей підхід корисний для виявлення пристроїв які належать до визначеної організації але вимагає ретельної перевірки релевантності знайдених хостів. Метод фільтрації за номером автономної системи (ASN) надає можливість шукати хости за їхньою належністю до автономної системи, а фільтрація за HTTP-кодом уточняє результати пошуку за статусом сторінки. Також інструменти можуть шукати піддомени через SSL-сертифікати. Так як останні часто містять список піддоменів, пов'язаних із основним. А пошук за ключовими словами, такі як авторські права або назви компаній можуть використовуватись для ідентифікації пов'язаних активів. [13]

Ці розглянуті оператори можуть бути використані в різних сценаріях. Пошук хостів з простроченими сертифікатами може знайти ресурси, які більше не обслуговуються. В випадку ж, якщо пентестер захоче, до прикладу, атакувати брутфорсом точку входу в ресурс, він може використати пошук панелей аутентифікації додавши оператор "http.title:login, Log in". За допомогою цих

запитів можна оперативно виявити слабко захищені панелі входу до внутрішніх систем, особливо якщо не застосована багатофакторна автентифікація. У разі відкритих логін-форм, зловмисники можуть проводити брут-форс атаки або тестувати відомі дефолтні паролі. Серед сценаріїв використання можна також виділити пошук публічних файлів, котрі можуть включати резервні копії, конфігураційні файли та інші чутливі дані; приклад використання оператора - “http.title’Index of’”.

Крім розглянутих Shodan і Censys вартує згадати про не мало потужний інструмент для розвідки – FOCA [14]. Він аналізує метадані документів, використовується для збору інформації про організації через документи, розміщені в публічному доступі. Цей інструмент здатний аналізувати файли різних форматів, включаючи PDF, DOC, XLS, PPT, а також витягувати із них метадані, які можуть містити критичну інформацію, таку як імена авторів документів, шляхи до локальних файлів на комп’ютері, типи використовуваних ПЗ та інші технічні деталі (версії програм, дати редагування тощо). Його перевага заключається в тому, що FOCA може збирати інформацію про DNS-записи, IP-адреси, імена серверів і навіть фізичне розташуваня організації; може автоматично завантажувати документи через Google, Bing та інші пошукові системи для подальшого аналізу. Але також є і обмеження, наприклад, він не підходить для реального часу через те що зосереджений на статичних даних і не може ефективно працювати з динамічним контентом або мережевими пристроями, до того ж необхідний попередній збор даних.

Інструмент FOCA часто використовується на підготовчому етапі тестування безпеки (reconnaissance phase), дозволяючи збирати профіль організації до моменту активних дій. Він також може бути інтегрований у більш розвідницькі фреймворки, автоматизуючи ланцюжки витягування інформації та побудову карти активів.

З наведеної нижче таблиці 1.1 із порівнянням видно, що кожен з інструментів має свою сферу ефективного застосування, але й жоден не є універсальним.

Таблиця 1.1 – Переваги та обмеження аналізованих рішень

Інструмент	Переваги	Обмеження
Shodan	Зручність фільтрації, багато операторів	Обмежена кількість запитів у безкоштовній версії
Censys	Більша точність для аналізу SSL/TLS	Складний інтерфейс, обмеження на API
FOCA	Аналіз метаданих документів	Не підходить для динамічного контенту
Google Dorks	Простота, доступність	Ризик CAPTCHA та блокування

Зловмисники ж пішли далі і почали створювати нові, свої інструменти для автоматизації пошуку інформації у відкритих джерелах (рисунок 1.5). Деякі з цих інструментів є настільними додатками, а деякі доступні у виді онлайн-сервісів. Одні з них автоматизують лише збір цільових ресурсів, тоді як інші автоматизують побудову векторів експлуатації та саму атаку. У зв’язку з цією загрозою більшість пошукових систем запровадили заходи для боротьби з автоматизацією, які в основному базуються на таких параметрах:

– кількість пошукових запитів із одного джерела (IP/сесія);

– частота запитів із одного джерела;

– масове отримання результатів для одного запиту.

Рисунок 1.5 – Приклад сервісу для кампаній автоматизованого пошуку і атак

Однак ці заходи змусили зловмисників шукати нові альтернативи для автоматизації атак через пошукові системи. Вони знайшли рішення у вигляді використання ботнетів для майнінгу даних пошукових систем. Завдяки потужності ботнетів зловмисники можуть запускати розподілені скоординовані кампанії пошуку, які обходять стандартні механізми протидії автоматизації. Інтегрована розподілена природа цих атак допомагає уникнути проблем із єдиним джерелом запитів. Використання спеціальних пошукових операторів, які штучно розділяють пошуковий простір (наприклад, за країнами чи частинами домену), дозволяє обійти обмеження, встановлені пошуковими системами на кількість результатів, які можна отримати за один запит. Крім того, зловмисники додають ще один рівень не прямого зв’язку через використання так званих “пошукових проксі”. Цей додатковий шар робить ідентифікацію справжнього джерела атаки та місцезнаходження зловмисника ще складнішою.

1.4 Постановка задачі

Описаний у попередніх підрозділах теоретичний та практичний контекст вказує на потребу створення не просто “інтерфейсу до Google”, а повноцінного, спеціалізованого інструменту, який дозволяє шукати, виявляти та аналізувати витоки або вразливості за допомогою пошукових механізмів. Сучасні реалії підтверджують актуальність такого рішення: зростає кількість інцидентів, пов’язаних із випадковим або зловмисним відкриттям конфіденційної інформації в інтернеті, а користувачі, особливо з обмеженим технічним досвідом, не мають простих інструментів для оперативної перевірки ресурсів. Саме тому доцільно розробити платформу, яка б дозволила фахівцям так і початківцям швидко та ефективно виявляти проблеми доступності даних у відкритому доступі.

Метою даного дослідження є розробка архітектурно і функціонально продуманої веб-системи, яка буде автоматизовано формувати Google Dorks запити на основі модулів і тегів, здійснювати запити через Google API,

обробляти отримані результати, а також класифікувати їх відповідно до рівня критичності знайденої інформації.

Об’єктом дослідження виступають системи й сервіси, які надають можливість автоматизованої перевірки інформаційних ресурсів з метою виявлення витоків, неправильних конфігурацій або відкритих документів, включаючи OSINT інструменти, пошукові механізми та класифікаційні алгоритми.

Предметом дослідження будуть методи генерації dork-запитів, підходи до фільтрації і класифікації результатів пошуку, а також архітектура клієнт-серверної веб-системи, побудованої з урахуванням UX-дизайну та інтеграції із зовнішніми API.

Як підсумок – маємо такі завдання, яких будемо дотримуватись у ході проведення дослідження у рамках цієї дипломної роботи. До основних завдань належать такі: проведення аналітичного огляду існуючих рішень у сфері автоматизованого OSINT та Google Hacking, дослідження роботи пошукових операторів Google і формування бази релевантних шаблонів запитів. До програмно-технічних завдань, належать відповідно наступні: проєктування архітектури веб-додатку з чітким розділенням логіки, використовуючи сучасні практики модульного та сервіс-орієнтованого програмування; інтегрування Google API як основний механізм пошуку задля уникнення блокування зі сторони компанії; розробка механізму класифікації знайденої інформації за допомогою евристичних правил або методів машинного навчання. Логічним завершенням після виконання попередніх завдань буде повноцінне тестування інтерфейсу за принципами UX та реалізація сценарного тестування для оцінки досвіду кінцевого користувача.

В ідеалі, очікуваним результатом буде створення гнучкого та надійного веб-додатку, здатного допомогти у виявленні критичних витоків та вразливостей за допомогою пошукових шаблонів та API Google.

2 РОЗРОБКА СИСТЕМИ АВТОМАТИЗАЦІЇ ГЕНЕРАЦІЇ ПОШУКОВИХ ЗАПИТІВ

2.1 Уточнення вихідних положень для розробки системи

Для проектування задачі, варто розуміти з чим стикаються основні цільові групи користувачів нашої системи у своїй рутинній роботі. Фахівці з кібербезпеки, наприклад, можуть моделювати кібератаки на компанії, державні установи та інші організації з метою пошуку вразливостей. Їхнє завдання – імітувати дії потенційного зловмисника та надавати замовнику звіт про знайдені недоліки в безпеці. Ці фахівці, до речі, досить часто застосовують Google dorks для ефективного пошуку інформації про об’єкт захисту в інтернеті. Звісно, що у арсеналі таких фахівців є й інші методи збору відкритих даних, проте саме дорки дозволяють швидко локалізувати потенційно вразливі точки без глибокого сканування або залучення важкої інфраструктури. Це доводить, що дорки слугують ефективним інструментом попереднього етапу оцінки загроз і є невід’ємною частиною будь-якої OSINT-кампанії.

Дослідники безпеки – експерти які аналізують вразливості, моніторять нові варіанти атак, беруть участь у програмах Bug Bounty або підвищують рівень обізнаності в кібербезпеці.

DevOps інженери впроваджують безпеку на ранніх стадіях розробки ПЗ, вони відповідають за постійну інтеграцію, а також за автоматизоване сканування безпеки коду, сервісів і середовищ.

Розробники програмного забезпечення, яким не байдуже, на скільки безпечною є їхня програма чи інфраструктура, зазвичай перевіряють за допомогою пошукових систем чи не потрапили їхні ключі, паролі, токени випадково у відкритий доступ. І на кінець спеціалісти з комплаєнсу (Compliance Officers) відповідають за дотримання регуляторних вимог типу GDPR, PCI DSS тощо. Вони контролюють, щоб в організації не було витоку конфіденційних даних у відкриті джерела.

Наша майбутня система має на меті автоматизувати їх рутинну роботу. Розглянемо випадок: коли пентестер має обмежений час для проведення тесту безпеки, інструмент скорочує цикл пошуку та підвищує точність результатів. Це допомагає швидше отримувати релевантні результати про можливі вразливості або витоки. Ще одним плюсом є те, що система дає змогу зберігати та повторно використовувати шаблони пошукових запитів під час переходу від одного пентест-проекту до іншого. DevOps інженери можуть адаптувати нашу систему під себе, налаштувавши її так, щоб вона автоматично відстежувала появу нових файлів чи згадок про організацію в публічному доступі (наприклад, “.env” файли, які містять секрети). Та й оскільки професія цих спеціалістів вимагає постійного моніторингу, наш веб-додаток зможе спростити налаштування постійних пошукових завдань для контролю витоків.

Програмісти-джуніори в основному мають за звичку не робити свої репозиторії з проектами на таких платформах як “github” або “gitlab” приватними, так вони лише винагороджують зловмисника, який знайде відкритий пароль або конфігураційних файл. Тому розробники за допомогою нашого проекту зможуть швидко дізнатись, чи не були викладені приватні репозиторії, паролі або токени; навіть якщо розробник не є експертом з безпеки. У великих компаніях ці ризики часто компенсуються процесами code review та автоматичними сканерами, але у стартапах або невеликих командах такі механізми відсутні. Через це цей додаток може стати своєрідним “гарантом контролю” навіть для команд, які не мають виділених фахівців з безпеки. Спеціалісти ж з комплаєнсу при ревізії якоїсь організації чи у разі інциденту зможуть оперативно з’ясувати, чи не потрапили конфіденційні файли до результату пошуку Google.

Восени 2024 року команда Sysdig Threat Research Team виявила глобальну операцію “Emeraldwhale” [15], яка була спрямована на викриті конфігураційна файли Git, у результаті чого було викрадено понад п’ятнадцять тисяч облікових даних хмарних сервісів [16]. По ходу кампанії хакери використовували приватні інструменти для зловживання неправильно налаштованими вебсервісами, саме

це дозволило їм викрадати облікові дані, клонувати приватні репозиторії та вилучати хмарні облікові дані з відкритого коду. У ході операції вони зібрали більше десяти тисяч приватних репозиторіїв. Основна мета викрадення – фішинг та спам. Подальші логи зловмисників, які виявили дослідники, показали масштабну кампанію сканування спрямовану на сервери з відкритими конфігураційними файлами Git. Але чому саме Git? Git – це система керування версіями [17] і вона дозволяє розробникам працювати над одним проєктом. Репозиторії Git містять конфігураційна файли, які можуть зберігати налаштування й облікові дані для доступу до інших ресурсів. Якщо директорія ".git" відкрита через неправильні налаштування вебсервера, хакери можуть отримувати доступ до історії репозиторія, конфігурацій, повідомлень комітів, імен користувачів, паролів або ключів API. Окрім цього, ці файли можуть містити сліди інфраструктурних рішень компанії – IP-адреси серверів, ключі до сховищ, URL-адреси внутрішніх сервісів. Тому навіть один відкритий ".git" каталог може дати хакеру достатньо інформації для масштабної атаки або руху всередині мережі. [18]

Серед результатів атак – понад п'ятнадцять тисяч облікових даних для хмарних сервісів, більше за шістдесят тисяч URL-адрес із відкритими файлами ".git"/"config", та саме цікаве те, що близько половини із шести тисяч виявлених токенів були дійсними. Це дослідження показує, на скільки серйозними можуть бути наслідки простих помилок конфігурацій, які коштуватимуть комусь фінансів.

Стає зрозумілим, що експерти з безпеки та особливо зловмисники стараються використовувати автоматизовані сканери такого типу. Якщо ви хочете швидко отримати релевантні результати на свій запит, то ручний метод пошуку буде не ефективним. Як мінімум через блокування з боку пошукової системи – в нашому випадку Google. Особистий досвід показав, що Google виставляє користувачеві "капчу" вже на сьомий раз відправки запиту. В масштабах кампаній сканування цього дуже мало. Підтримка Google повідомляє, це може відбуватись через виявлення автоматичного трафіку, тобто все що

торкається комп'ютерних програм, роботів, автоматизованих сервісів і засобів скрейпінгу в пошукових системах буде блокуватись. Тому постає логічне питання "як обійти зауваження від Google і не бути заблокованим?". В цьому випадку гарним інструментом буде Google Custom Search – це засіб, API якого надає Google для вільного пошуку без блокування. Таке рішення надає змогу вбудовувати функцію пошуку напряму в додаток, зберігаючи контроль над запитами та результатами без залучення браузера або взаємодії з CAPTCHA. Правда в тому, що безкоштовна версія надає лише сто запитів на день та лише десять результатів на один запит.

Через цей факт з'являється задача організувати такий функціонал, щоб будь який користувач зміг користуватись веб-додатком без можливості отримати блокування зі сторони Google. Ми повині змоделювати ситуацію, щоб краще розуміти ризики кінцевого користувача спіймати капчу або бан: користувач отримує капчу в середньому після 6-7 запитів з інтервалом між ними в 10 секунд; нормальною поведінкою вважається 1-2 запита на хвилину; це все заважає концепції автоматизації у нашому проекті, тому такий підхід буде виключений.

Тепер подивимось на алгоритм (рисунок 2.1) ситуації, коли запити до Google надходять через його API (випадок з безкоштовним тарифом, без підписки).

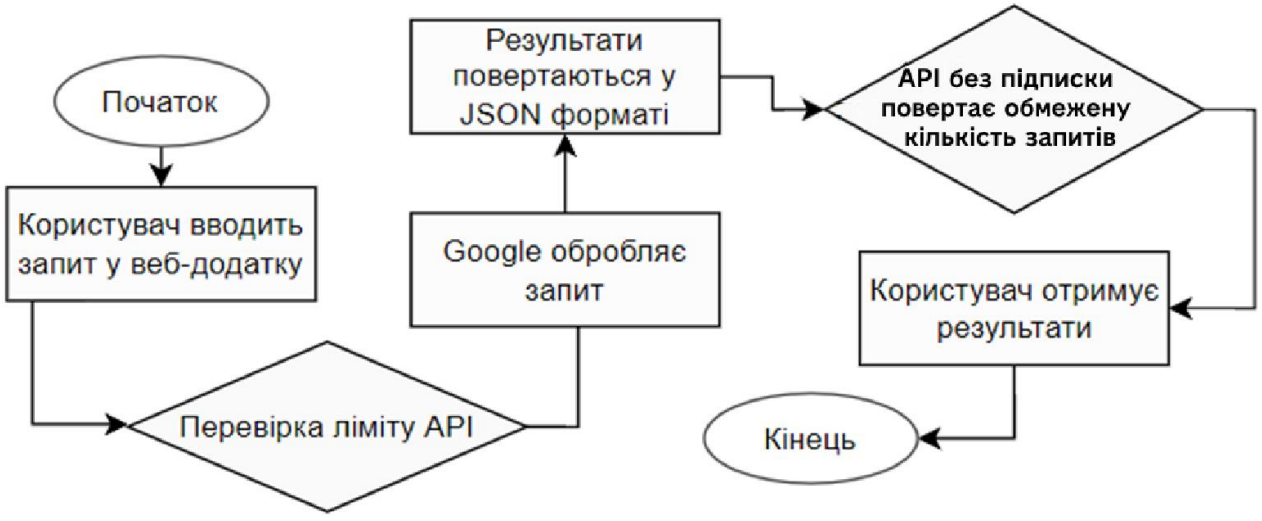


Рисунок 2.1 – Алгоритм пошуку через Google API

Виходячи з висновків вище, було прийнято рішення використовувати Google Custom Search API у веб-додатку цього проекту через наступні причини:

– використання API повністю відповідає політиці Google, а це гарантує стабільність процесу та відсутність блокувань;

– навіть безкоштовний тариф дозволяє виконувати до 100 запитів на день, а платний тариф розширює можливість до 1000 і більше запитів;

– ми можемо налаштувати обмеження кількості запитів на день залежно від потреб проєкту, що забезпечує економічну ефективність;

– API повертає дані у зручному форматі (JSON або XML), це значно спрощує обробку та інтеграцію відповіді у вивід результатів. [19]

Такий підхід також відкриває можливість для обробки відповідей через машинне навчання або інші евристичні алгоритми для подальшої класифікації результатів за рівнем важливості чи загрози.

Але зарано говорити, що проблему автоматизованого підходу для пошуку вирішено тільки за допомогою API. З точки зору користувача при роботі з Google dorks запитами не все може бути відразу зрозуміло, тому потрібно впровадити запити у виді шаблонів, які вже є в відкритому доступі або їх можна створити самотужки в процесі створення веб-додатку – такий підхід із шаблонами може навіть послужити своєрідним тренажером, для людей котрі мало що розуміють в техніці Google hacking; а для людей, котрі не від сьогодні знайомі із цією темою буде просто широкий вибір шаблонів, які можна легко редагувати або модифікувати в залежності від своїх цілей. А це в свою чергу економить час.

Google dorks запити в готовому виді можна знайти у відкритих джерелах, наприклад, найбанальніший варіант – це сайт Exploit-db. В категорії “google hacking database” [20] можна знайти запити “на будь який смак”: від запитів, котрі націлені на виявлення відкритих веб-серверів до запитів, які зможуть знайти файли вміщуючі паролі (рисунок 2.2):

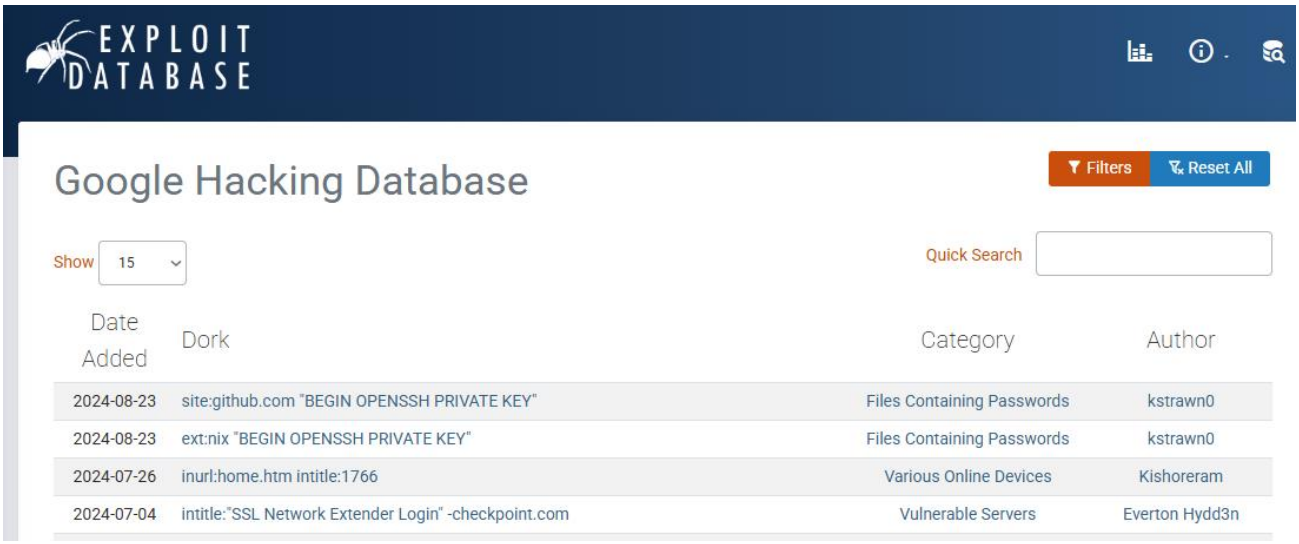


Рисунок 2.2 – На сайті explot-db ентузіасти діляться з Інтернетом dork-запитами

Щоб ще більше полегшити сприйняття процесу для користувача, скористаємось принципом “розділяй і володарюй”. Є багато різних запитів, які ми використаємо як шаблони але всі вони розділені на дуже абстрактні категорії. Для цього проекту ж постає задача ще більше категоріїзувати шаблони, щоб кінцевий користувач міг з легкістю обрати собі найбільш потрібний. Створимо дев’ять модулів за такими задачами:

– пошук адмін-панелей;

– пошук електронних пошт;

– пошук файлів та документів;

– мета-дані різного типу;

– мережеві пристрої;

– пошук особистостей в Інтернеті;

– потенційні вразливості;

– веб-сторінки;

– веб-сервіси.

Таке розбиття на модулі забезпечить уникнення дубльованих запитів, структуруватиме логіку пошуку та спростить підтримку системи у майбутньому.

Підемо ще далі. Кожному запиту потрібно буде дати крім модуля, теги. Наприклад, маємо запит “filetype:bak inurl:"backup"”. Цей запит виконує пошук резервних копій файлів в Інтернеті. Теги можна давати за асоціаціями або за

прямими відповідями на нього; якщо візьмемо такий запит і введемо в пошук, то отримаємо результат як на рисунку 2.3. На цьому рисунку видно різні веб-ресурси, які тримають файл з бекапами у відкритому доступі. Фундаментуючись на результатах, можемо дати назви першим тегам – це “backup ” та “files”. Так потрібно вчинити зі всіма шаблонами.

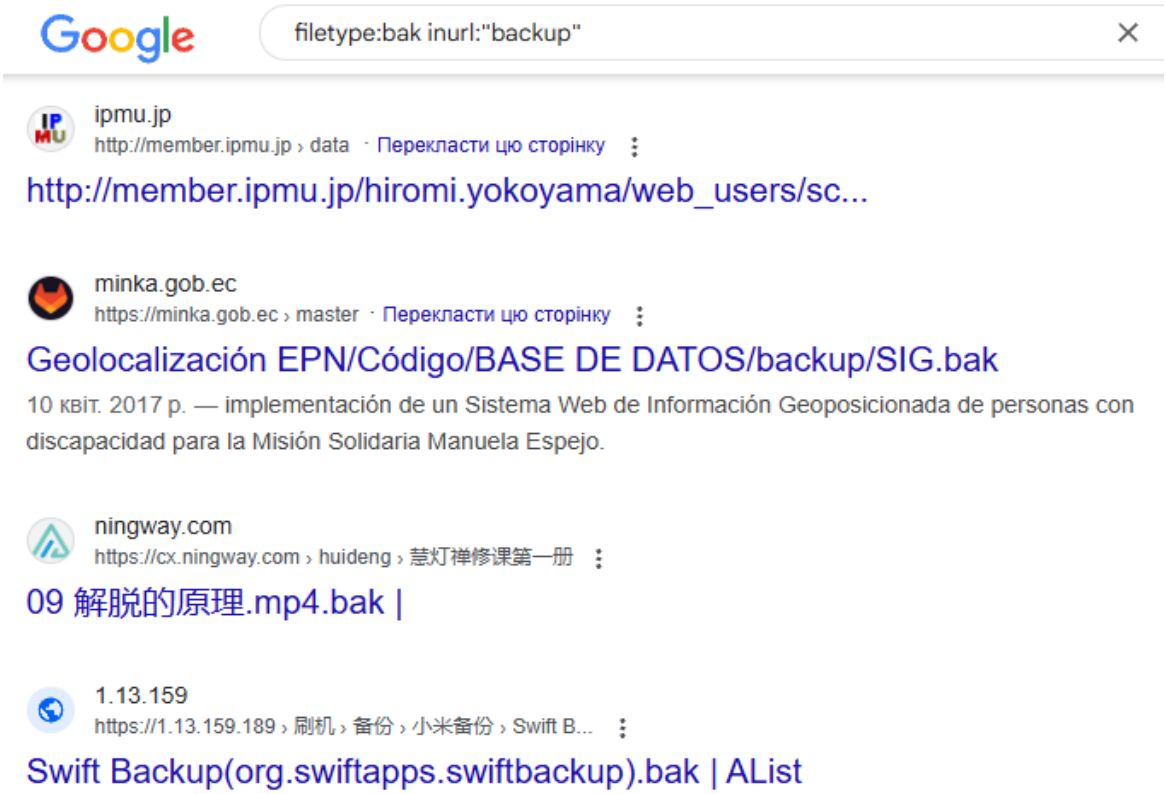


Рисунок 2.3 – Результат на запит

Теги можуть переплітатись між модулями, тобто два різних за модулями шаблона можуть мати схожі теги. Це спроектовано по таким причинам: строга типізація тегів в рамках проекту (не може бути синонімів для тегів, лише точний вказівник); один шаблон модуля А може знаходити схожі за суттю результати з модулем В (рисунок 2.4). Наприклад, тег “password” може бути присутній як у модулі “пошук файлів”, так і в модулі “вразливості”, бо файл із паролями – це одночасно і документ і потенційна загроза. Цей прекрасний підхід дозволяє будувати більш гнучку систему класифіккації, де кожен запит не ізольований а включений у загальний контекст пошукових сценаріїв.

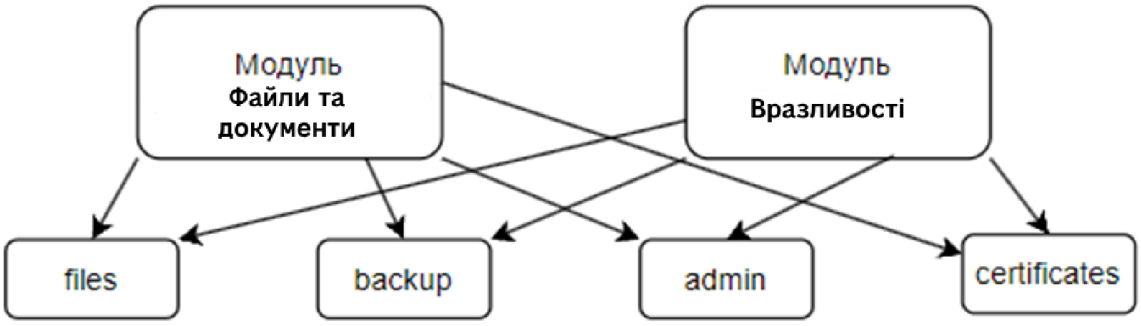


Рисунок 2.4 – Приклад, як теги можуть пересікатись між модулями

Таким чином, керуючись принципом “розділяй та володарюй”, ми задаємо користувачеві на старті користування нашим додатком уточнюючі два питання: “де шукати?” та “що саме шукати?”.

Ще одна проблема з якою зіткнеться користувач при отримані результатів Google – це їх можлива нерелевантність (рисунок 2.5). Наприклад, ми хочемо знайти адмін-панелі для користувачів “php”. Отож, ми вводимо запит “inurl:/phpmyadmin/” але в результатах бачимо не те, чого бажали.

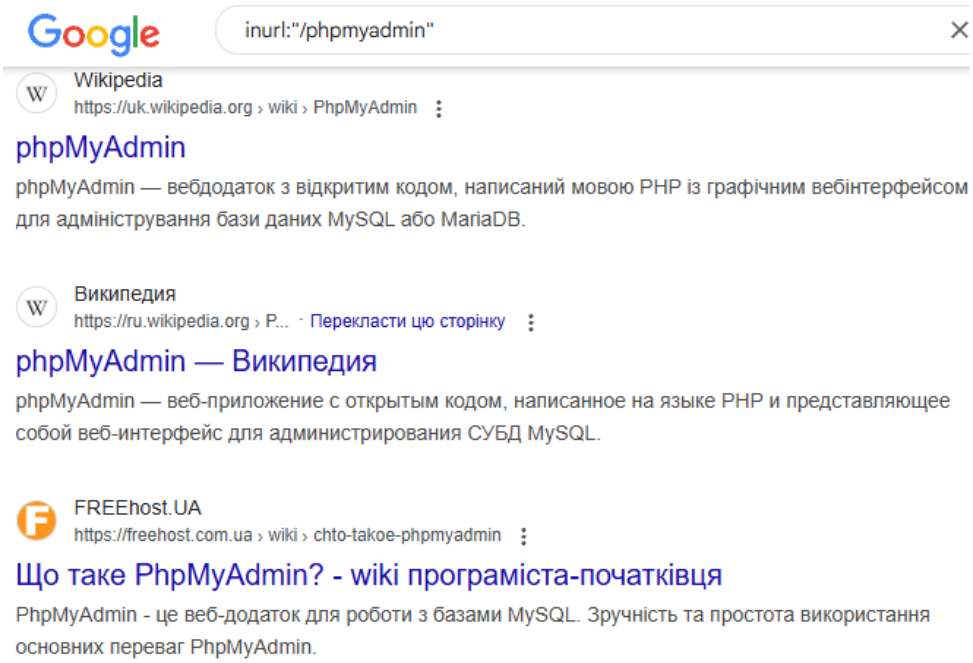


Рисунок 2.5 – Приклад не релевантних результатів

Серед них можуть бути не потрібні нам ресурси, наприклад, якісь статті або навчальні матеріали. Звісно, користувач може уточнити більш коректно свій

запит але для цього треба розуміти специфіку dork-запитів; якщо наш додаток прагне надати кінцевому користувачеві автоматизоване рішення, краще б було віддати проблему нерелевантних результатів самій системі аби надати комфортні умови роботи користувачеві і полегшити його процес. Нехай, ми маємо число N всієї суми результатів, які отримаємо від Google API, з числа N нехай 60% непотрібні, а інші ті які нам цікаві. В такому разі, дані повинні обробляти та класифікувати на релевантні та нерелевантні. Таким чином, для кожного модуля окремо буде розроблений механізм класифікації результатів.

2.2 Архітектура системи

Архітектура будь якого додатку – це план або структура, тобто скелет, який описує, як система організована, як різні її частини взаємодіють між собою. В цьому випадку можна провести аналогію з будівництвом будинку, де є стіни, кімнати, двері або вікна, але замість будівельних матеріалів ми працюєм з кодом, базами даних та серверами. Так само як і в будівництві, де неякісний фундамент може спричинити обвал, неправильна архітектура додатку може призвести до втрати продуктивності, вразливостей або повного краху системи при масштабуванні. Саме тому й на початковому етапі розробки важливо закласти правильну структуру, яка дозволить проєкту розвиватись без необхідності повної перебудови. Архітектура це важлива складова у проектуванні любої системи і від неї залежить як система працює, на скільки вона гнучка, безпечна і чи готова до масштабування у майбутньому. [21]

Архітектури проектів бувають різних типів. Розглянемо деякі з них. Монолітна архітектура [22] (рисунок 2.6) – це єдиний блок коду, де всі частини взаємодіють прямо одна з одною. Якщо це великий додаток, в ньому є всі можливості та функції в одному місці. Серед плюсів такої архітектури: простота її розробки, легкість тестування, чітке представлення про всі зв’язки у програмі. Але є і мінуси: з ростом системи код стає великим, а це додає складності у його

підтримку; зміна однієї частини може вплинути на інші частини, що може привести до помилок.

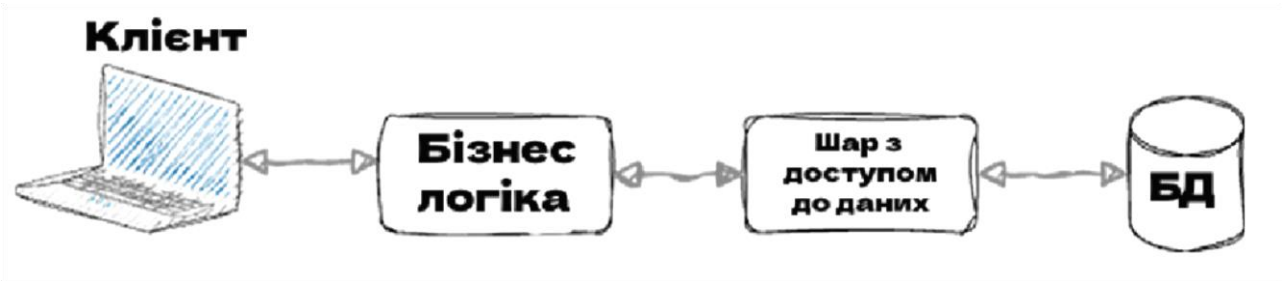


Рисунок 2.6 – Монолітна архітектура

Мікросервісна архітектура [23] роздроблює систему на декілька автономних частин, які виконують кожна окрему функцію. Це практично підпрограми. Така архітектура часто використовується в інтернет-магазинах тощо, наприклад, може бути поділ на мікросервіси за бізнес-логікою, як на рисунку 2.7. Крім легкого масштабування окремих частин системи, також можна оновлювати або змінювати один мікросервіс не займаючи інші. Але серед мінусів: проблематичність в налаштуванні взаємодії між мікросервісами; необхідно тонко підходити до тестування і безпеки, оскільки система складається із багатьох частин. Для успішної реалізації такої архітектури необхідне впровадження систем логування, моніторингу стану кожного сервісу, а також централізованого управління конфігураціями.

Часто мікросервіси використовують брокери повідомлень, такі як RabbitMQ або Kafka для організації асинхронної взаємодії між сервісами, що ще більше ускладнює інфраструктуру. Однак завдяки й цим механізмам мікросервісні системи демонструють високу стійкість до збоїв: наприклад, якщо один сервіс вийде з ладу – тоді інші можуть продовжувати роботу, що критично для систем із високими вимогами до безперервної доступності. Такі архітектури активно використовують великі компанії – такі як Netflix, Amazon, Uber – для обслуговування мільйонів користувачів у реальному часі.

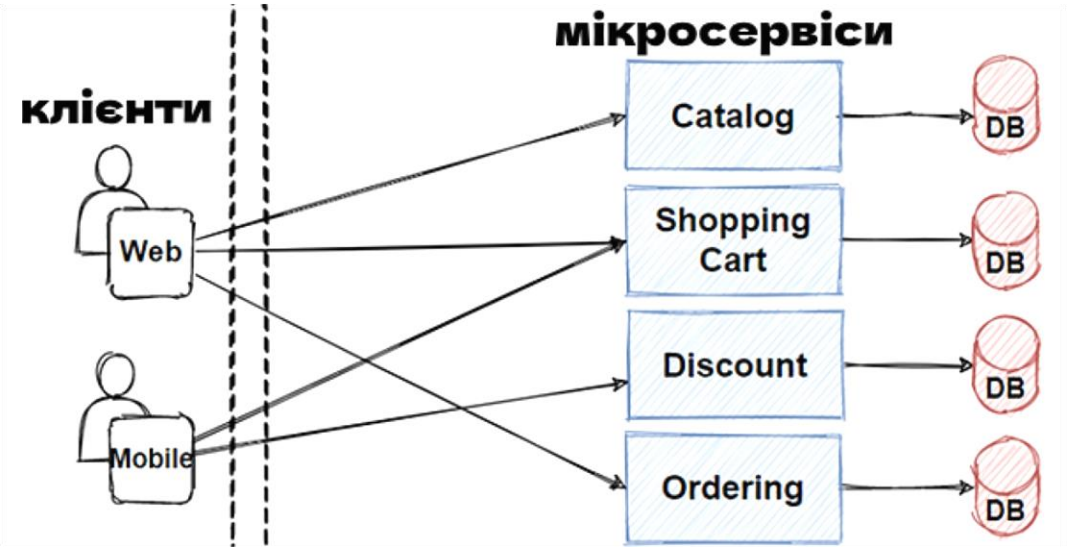


Рисунок 2.7 – Мікросервісна архітектура.

Сервіс-орієнтовна архітектура або SOA (Service-oriented architecture) – це архітектура де всі частини системи представлені у виді незалежних сервісів, які можуть обмінюватись даними один з одним. [24]

Виходячи з розуміння процесів роботи нашої системи, можна з впевненістю сказати, що сервісно-орієнтовна у поєднанні з архітектурою “Model View Controller” нам підходить. Тому що така архітектура розділить обов’язки між частинами системи, спростить підтримку та зробить код більш читабельним. Навіть у майбутньому завдяки такому підходу можна буде легко інтегрувати нові модулі без порушення логіки існуючих частин. Model View Controller (або ж MVC) – це набір архітектурних принципів для побудови систем з інтерфейсом користувача, що в сумі ділиться на три основних складових: моделі – бізнес логіка програми, представлення – все що бачить користувач, контролер – займається обробкою дій користувача у взаємодії із системою. [25]

Кістяк нашого додатку – це сервіс “JsonDataService”. Він виконує роль менеджера конфігурацій, відповідального за завантаження та обробку інформації з структурованих JSON-файлів, які містять конфігурацію модулів і операторів для запитів. Основна мета цього сервісу – забезпечення доступу до даних в JSON форматі, які зберігають в собі шаблони для генерації пошукових запитів. Він дає можливість завантажувати шаблони пошукових запитів та їхні параметри. В коді

сервіса зберігається шлях до JSON файлів, а також два методи: для десеріалізації файлів з шаблонами для різних модулів, та для десеріалізації файлу з операторами пошуку. Такий підхід дозволяє легко додавати нові модулі та оператори без зміни основної логіки програми.

Наступний сервіс управляє тегами. “TagsService” виконує аналітичну та категоризаційну функцію, дозволяючи витягувати унікальні теги, які прив’язані до конкретних модулів пошуку. Він допомагає класифікувати та фільтрувати пошукові запити, забезпечуючи швидкий доступ до ключових тегів, що характеризують певний модуль або групу запитів. Також цей сервіс відіграє важливу роль у побудові інтерфейсу користувача, дозволяючи створювати динамічні фільтри. Це дасть користувачеві швидке орієнтування у великій кількості запитів, що особливо корисно при роботі з масштабними кампаніями розвідки.

Ядро системи – сервіс “DorkGeneratorService”, який відповідає за генерацію пошукових запитів Google Dorks на основі заданих модулів, тегів чи параметрів. Він виконує ключову функцію – генерує пошукові запити, використовуючи шаблони та користувацькі параметри. Завдяки гнучкому підходу він дозволяє формувати специфічні Dork-запити для пошуку інформації. Короткий опис функціональності сервісу “DorkGeneratorService”:

– завантажує модуль пошуку через JsonDataService;

– аналізує всі доступні шаблони запитів у модулі;

– фільтрує шаблони за списком заданих тегів;

– повертає відфільтровані шаблони, що містять хоча б один із зазначених тегів;

– виконує перевірку відповідності параметрів (через сервіс-валідатор);

– динамічно підставляє значення користувача у змінні шаблону;

– перевіряє синтаксис сформованого пошукового запиту перед поверненням.

Серед ключових переваг такого сервісу є автоматизований підхід, який формує запит без необхідності ручного складання, перевірка коректності та гнучкість для користувача для зміни параметрів під особисті потреби.

Ще один критичний сервіс, який виконує автоматизовані пошукові запити та повертає структуровані результати – “SearchService”. Забезпечує пошук через Google API, обробку результатів та їх структурування у форматі зручному для подальшої роботи. Цей сервіс по суті будує запит для відправки, а потім відправляє його через “HttpClient”. Серед переваг цього сервісу, це вперше чергу безпечне зберігання ключів API користувача – він використовує сесію для динамічного отримання конфігурації.

Як згадувалось вище – наш додаток працює з JSON файлами. Це своєрідна заміна базі даних, адже таким чином, система може запускатись і працювати локально. По-друге, це підходить не лише для локального запуску, а й для швидкого прототипування та розробки MVP (minimum viable product). У разі потреби, JSON-структуру можна легко перенести на повноцінну реляційну або документну базу даних (наприклад, PostgreSQL або MongoDB) з мінімальними витратами на адаптацію коду. JSON – це простий формат обміну даними, який заснований на двох структурах даних – колекція пар “ключ = значення” [26]. У додатку такий формат використовується для зберігання шаблонів запитів та переліку операторів. На рисунку 2.8 зображено шаблон, який по суті є JSON-об’єктом із ключами та їх значеннями.

```
"templates": [
  {
    "name": "Cisco Панелі",
    "description": "Знайти панелі управління Cisco пристроями.",
    "template": "intitle:\"Cisco Router\" inurl:main",
    "hint": "Вкажіть модель пристрою або IP-адресу.",
    "tags": [ "network", "admin" ]
  },
```

Рисунок 2.8 – Приклад шаблону в JSON-файлі

В цілому архітектура нашої системи ділиться на декілька шарів. Це можна уявити у вигляді піраміди, яку зображено на рисунку 2.9. Ця структура демонструє, як логічні рівні – від даних і сервісів до інтерфейсу та сесій користувача – поєднуються у цілісну систему. Кожен рівень взаємодіє лише з суміжними шарами, що мінімізує ризики “протікання” логіки або неправильного використання даних.

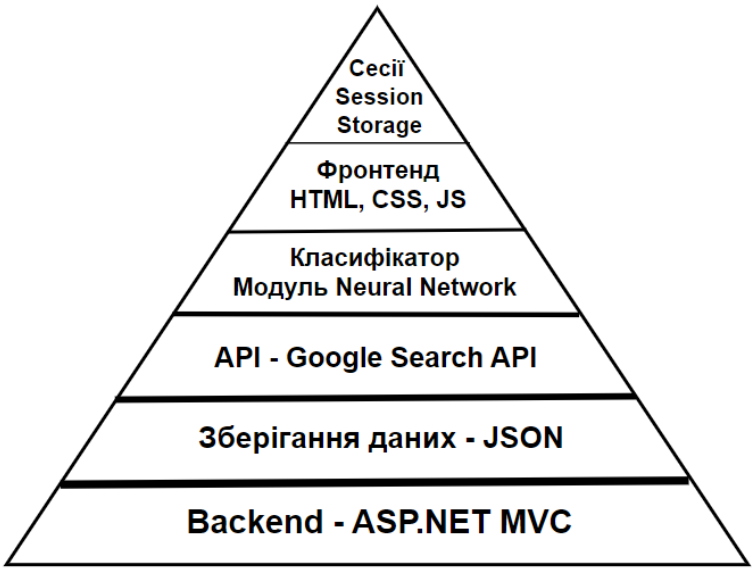


Рисунок 2.9 – Набір технологій у вигляді архітектури

Як результат, можна сказати, що наша архітектура поєднує в собі сервісно-орієнтовний підхід (SOA) та шаблон MVC, що дозволяє ефективно розділити бізнес-логіку, шар представлення й механізми доступу до даних. Ключові сервіси – “JsonDataService”, “TagsService”, “DorkGeneratorService” та “SearchService” – забезпечують модульність, легку розширюваність та гнучке керування шаблонами і запитами. Використання JSON як міні-БД спрощує розгортання адже система може запускатись локально без потреби в окремому сервері баз даних. У комплексі це надає проєкту масштабованість, простоту додавання нових модулів і швидку адаптацію під змінні потреби, що є визначальними факторами для сучасного веб-додатку. Отже, архітектура цього додатку не лише надійна, але й зручна для подальшого розширення. В архітектуру з таким стилем можна легко впровадити CI/CD процеси,

налаштовувати автоматизоване тестування та впевнено масштабуватись при зростанні навантаження.

2.3 Класифікація результатів

Проблематика результатів, як було розглянуто у минулих підрозділах заключається у їх різноманітності, що логічно. Google видає окрім потрібних об’єктів ще й непотрібні, а це створює шум. Тому для боротьби з таким феноменом було прийняте рішення класифікувати виводимі результати; такий підхід полегшить і прискорить роботу кінцевого користувача, тоді йому не доведеться передивлятись кожен об’єкт в надії знайти те, що йому дійсно потрібно: він просто буде бачити перед собою відсортовану за критичністю колекцію результатів. Як результат, сортування за критичністю дозволятиме ефективніше розподіляти ресурси команди безпеки, фокусуючись насамперед на загрозах з високим потенціалом ризику. Це в свою чергу особливо актуально у великих організаціях, де кількість інцидентів перевищує можливості ручної перевірки. Фільтрація результатів також покращить якість аналітики та звітності, зменшуючи ймовірність людських помилок при аналізі великих масивів даних.

Предметом дослідження тут виступлять результати як такі, на рисунку 2.10 їх, поки, сирий вигляд у нашому додатку.

Рисунок 2.10 – Вивід некласифікованих результатів

Для шаблону запиту “inurl:login intext:admin site:.ru” було отримано близько десяти результатів, але після перевірки виявилося, що лише три з них

дійсно ведуть на адміністративні панелі. Це свідчить про те, що навіть при вузькоспеціалізованих запитах частина знайдених сторінок залишається нерелевантною або другорядною. Таким чином, перед нами стоїть завдання: розробити метод класифікації, який дозволить автоматично розподіляти отримані результати за рівнем їх критичності. Отже, нам дано: назву, посилання, короткий опис та назву модуля, через який проходить пошук.

При класичних підходах до класифікації даних часто використовують евристичні алгоритми, такі як аналіз ключових слів або шаблонне розпізнавання. Суть цього метода складається з вирішення проблеми основуючись на певних істинних гіпотезах, а не на суворих висновках з наявних фактів і припущень. Евристичні алгоримти використовуються у антивірусах, наприклад, у детекції шкідливого ПЗ. Евристичний аналіз дозволяє виявити нові віруси ще до того, як для них будуть створені сигнатури. На відміну від сигнатурного методу, який шукає точні відповідності за відомими зразками, евристичний підхід аналізує схожість поведінки файлів із вже відовими шкідливими об’єктами. Однак тут є недоліки:

– алгоритм може помилково вважати безпечний файл вірусом;

– система може не знати як правильно його знешкодити, якщо вірус не має точної сигнатури;

– при новаторських вірусах, евристичний аналіз може не спрацювати.

Цю аналогію можна підвести й до нашої ситуації. При евристичних алгоритмах потрібно вручну створювати правила, які можуть не врахувати всіх можливих варіантів; невеликі зміни, наприклад, у “Title” або “Snippet” можуть зробити систему неефективною; також потрібно буде постійно оновлювати правила [27]. У практиці розробки безпечних рішень це значитиме додаткове навантаження на підтримку проекту, оскільки кожне оновлення логіки вимагатиме повторного тестування тощо. Відсутність адаптивності у жорстких правилах робить систему вразливою до нових, ще не відомих патернів або контекстів витоків. Аналогічно, при спробі масштабування проєкту на інші мови чи регіони доведеться переписувати правила майже з нуля.

Альтернативою є використання методів машинного навчання, які дозволяють автоматично виявляти закономірності в даних. Серед можливих варіантів розглянемо:

– логістична регресія;

– дерева рішень;

– глибокі нейромережі;

– наївний байєсівський класифікатор.

Логістична регресія – це статистичний метод, котрий використовується для прогнозування ймовірності належності об’єкта до одного з двох класів (бінарна класифікація). Він працює таким чином, що використовує лінійну функцію для обчислення певного значення (логіт-функція), яке потім перетворюється у ймовірність (від 0 до 1). Якщо ця ймовірність вище певного порогу, наприклад 0.7, тоді об’єкт відноситься до першого класу, якщо вище – до другого [28]. І хоча цей метод є лінійним, у поєднанні з техніками попередньої обробки (наприклад, розширенням ознак або інженерією атрибутів), він здатен давати конкурентні результати у задачах класифікації тексту. Однак варто враховувати, що логістична регресія не дуже добре справляється з багатокласовими задачами без модифікацій (наприклад, через підхід “один проти всіх”). Цей підхід полягає в тому, щоб для кожного класу створювалась окрема модель, яка навчається розрізняти: “цей клас” проти “всі інші”. Наприклад, при класифікації результатів на три типи – критичні, середні та інформаційні – буде створено три окремі моделі і кожна з них б прогнозувала ймовірність, що об’єкт належить до свого класу. Остаточний вибір робився б на основі найвищої ймовірності серед трьох. Гарний спосіб класифікації але іноді потребує балансування класів або додаткової валідації. [29]

Другий варіант – це дерева рішень. Метод машинного навчання, який розділяє дані по групах, грунтуючись на правилах “if-else”. Ці дерева організовані у виді ієрархічної структури, яка складається з вузлів прийняття рішень за оцінкою значень певних змінних для прогнозування кінцевого значення. Використання класифікаційних дерев призводить до символічного

позначення класу. Насамперед цей метод вимагає створення навчальної збірки даних, яка використовуватиметься у навчанні самих дерев рішень.Унікальністю дерева є можливість отримати інтерпретований результат – логіку прийняття рішення можна візуалізувати, що як мінімум зручно для дебагу. Однак навіть одне неправильно підібране правило на початкових рівнях дерева може запустити “ефект доміно”, іншими словами – воно спрямує аналіз у хибному напрямі, що потребує валідації глибини дерева та чистоти навчального датасету.

Нижче на рисунку 2.13 конкретний приклад, як міг би застосовуватись такий підхід у нашій системі: [30]

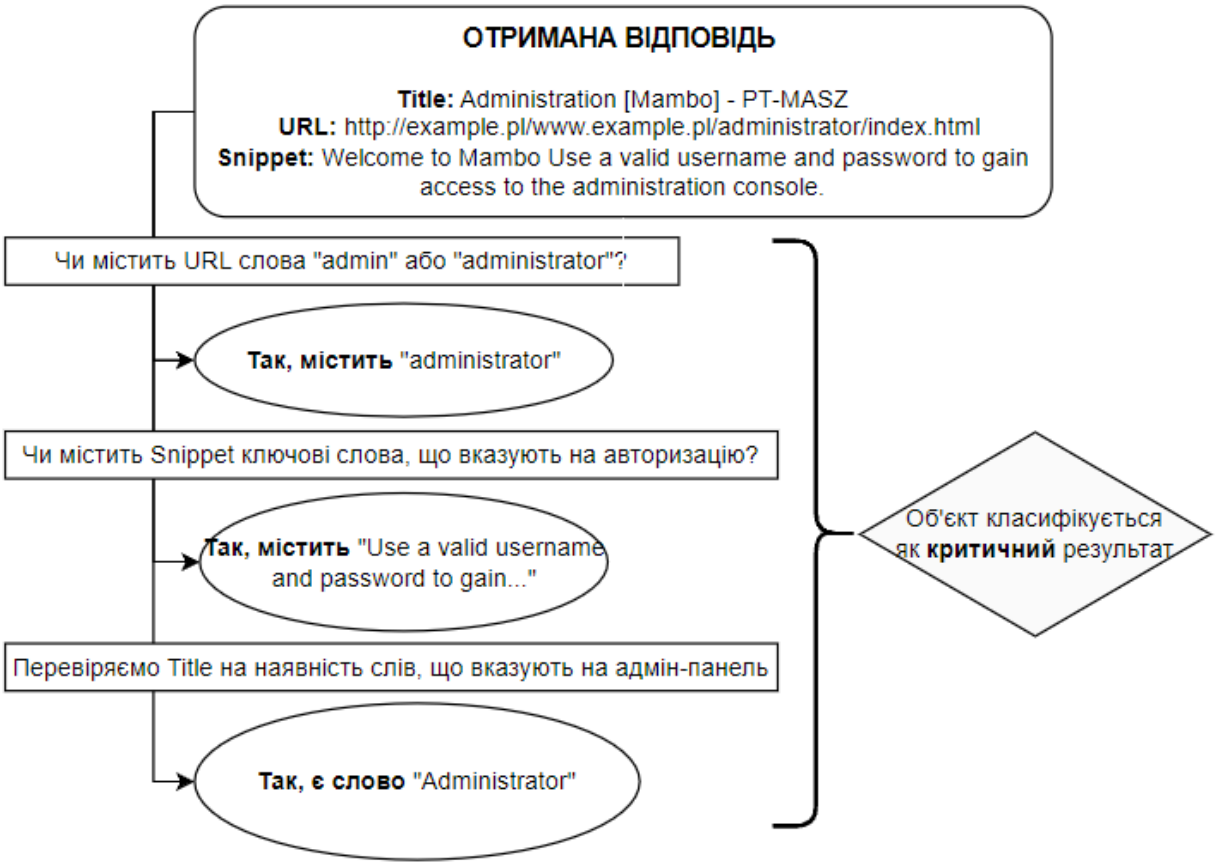


Рисунок 2.13 – Приклад використання метода дерев рішень у нашій системі

Ще один варіант класифікації результатів – це глибокі нейронні мережі (Deep Learning). Метод, котрий використовує багатошарові штучні нейрони, щоб визначати складні закономірності у великих даних. В результаті виходить модель глибокого навчання, яка після завершення навчання вміє обробляти нові

дані. Одне з головних переваг використання цього методу є вміння нейронних мереж знаходити в даних приховані закономірності і зв’язки, які раніше були не такі очевидні. [31]

Останній потенційний метод базується на теоремі Байєса та примущенні, що всі ознаки незалежні одна від одної (навіть якщо це не так насправді). Формула Байєса дозволяє вияснити ймовірність події при умові, що відбулась пов’язана з ним інша подія. [32]

$$P(A|B) = \frac{P(A) \times P(B|A)}{P(B)} \tag{2.1}$$

Щоб довести формулу 2.1, можна взяти задачу класифікації пошукових результатів (рисунок 2.15).

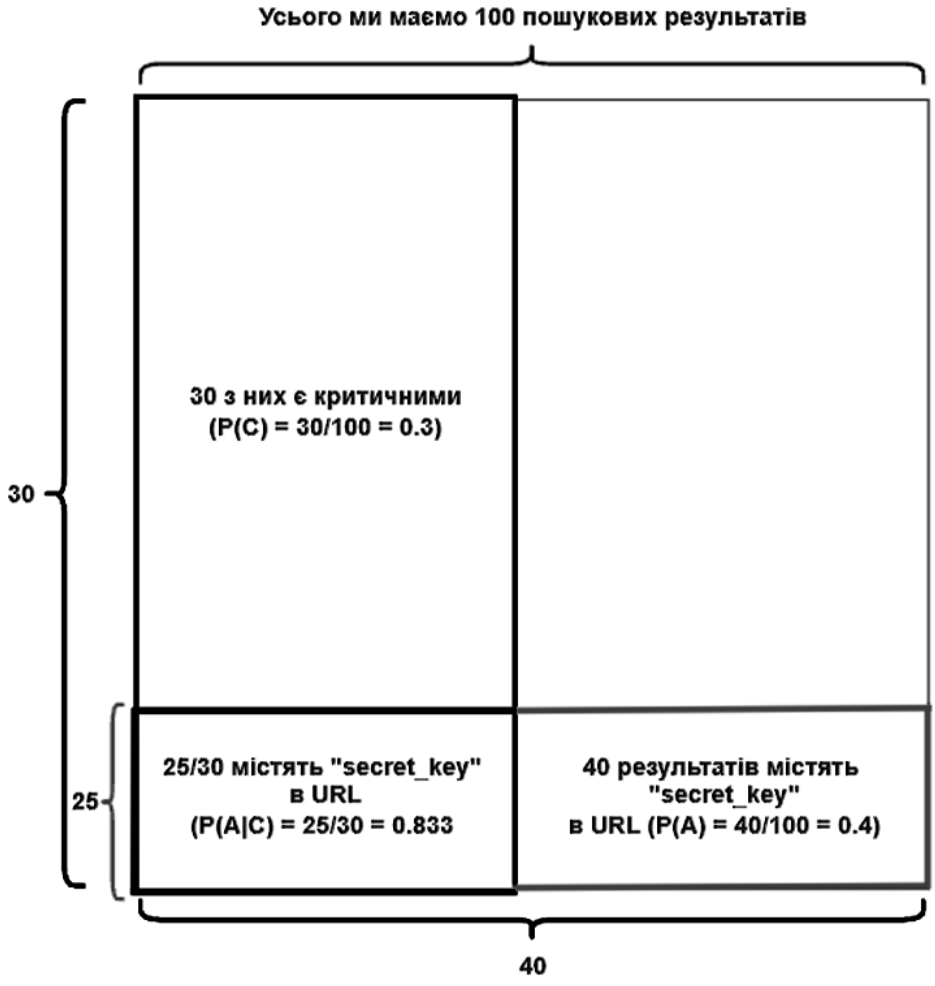


Рисунок 2.15 – Графічний вигляд умов задачі

Припустимо, що у нас є база даних пошукових результатів і ми аналізуємо їхню критичність. Ми розглядаємо дві події: результат є критичним (означає, що сторінка несе високий ризик) або результат містить якесь слово в посилані, яке вказує на критичність. Візьмемо емпіричні дані: маємо 100 сторінок, 30 з яких є критичними, 40 результатів містить слово “secret_key” в посиланні, а з тих 30 критичних результатів – 25 входять в перелік посилань із критичним словом. Таким чином, використовуючи формулу проведемо розрахунки:

$$P(C|A) = \frac{0{,}3 \times 0{,}833}{0{,}4} = \frac{0{,}249}{0{,}4} = 0{,}625 \quad (2.2)$$

Відповідь на формулі 2.2 означає, що якщо в посиланні є слово “secret_key”, то ймовірність того, що результат є критичним становить 62,5%. Звичайно, подібні обчислення у системі будуть виконуватись автоматично але розуміння механізму прийняття рішення залишається важливою частиною процесу розробки.

Звісно, це лише поверхневий приклад, адже цей принцип можна розширити, наприклад, замість одного фактора “secret_key” в посиланні можна враховувати також вміст “Snippet” або “Title”. Або взагалі комбінувати такий підхід з іншими методами, такими як дерево рішень.

Таблиця 2.1 – Порівняння методів для автоматичної класифікації даних

Метод	Основна ідея	Переваги	Недоліки
1	2	3	4
Логістична регресія	Прогнозує ймовірність класу	Проста, швидка	Погано працює з нелінійними даними
Дерева рішень	Створює ієрархію правил для класифікації	Інтуїтивно зрозумілий	Може перенавчуватись

Кінець таблиці 2.1

1	2	3	4
Глибокі нейромережі	Використовує багатошарові нейрони	Висока точність	Потребує багато ресурсів і даних
Наївний байєс	Обчислює ймовірності класів за ознаками	Швидкий, ефективний	Припускає незалежність ознак

Щоб знаходити ймовірності принадлежності об’єкта до певного класу, потрібно для початку підготувати дані для їх подальшого аналізу і виявлення закономірності, а на її основі створити алгоритм розрахунку.

Ключовою проблемою, яка виникає при формуванні збірки навчальних даних – є їх потенційна упередженість або спотвореність, а це може призвести до неправильних або несправедливих висновків. Тому потрібно оглянути та проаналізувати вплив систематичної упередженості в аналізі даних. Зокрема, існує фундаментальний вид систематичної упередженості – упередженість через усічення (truncation bias).

Усічена упередженість часто виникає, коли вибірки які знаходяться за певною частиною генеральної сукупності не спостерігаються; простіше кажучи – не враховуються всі можливі дані. Так було, наприклад, коли в США оцінювали освіту та заробітки осіб із низькими результатами тестів: у цьому випадку було зібрано вибірку людей, яких не взяли у армію через низькі бали на тесті AFQT (Armed forces qualification test). Проблема полягала у тому, що доступні дані були усіченими, оскільки включали лише осіб із низькими результатами тесту, а це в свою чергу призвело до упередженості у висновках та неточних оцінок; щоб отримати неупереджену оцінку, потрібно було зібрати вибірку людей з усіма балами – як з низькими, так і високими. У нашому ж випадку, для того, щоб класифікувати дані за критичністю – потрібно звертати увагу на всі результати, як і ті, які є апріорі критичними, так і ті, які носять інформаційний характер. [33]

Після того як ми зібрали близько ста двадцяти рядків з результатами (спочатку для модуля з адмін-панелями) їх було оцінено в ручну, тобто до кожного об’єкта який повернув пошук було додано мітку – “Critically”, “Medium critical” або “Information”. Щоб алгоритм працював правильно, дані були приведені до єдиного формату:

– URL, Snippet, Title були об’єднані в одне ціле;

– текст був приведений до нижнього регістру;

– видалили зайві символи, тобто лапки, пунктуацію тощо;

– провели токенізацію текста (слова в тексті розбиваються на токени), крім того ми генеруємо двограми (пари слів, наприклад “admin login”, що дозволить врахувати більше контексту). Далі після розбиття тексту ми збираємо словниковий запас – набір усіх унікальних слів, які зустрічаються у всіх об’єктах навчальних даних. Цей список потрібен для обчислення умовних ймовірностей.

Тепер можна приступити до налаштування алгоритму класифікації, який базується на теоремі Байєса і примущенні незалежності токенів. Він визначає, до якого класу (наприклад, “Critically” або “Information”) належить новий результат, на основі обчислення ймовірності кожного класу для заданого тексту. Розглянемо конкретну формулу 2.3, з якою працюємо:

$$P(C) = \frac{N(C)}{N} \tag{2.3}$$

На формулі 2.3 де “N(C)” – кількість об’єктів у класі “C”; де “N” – загальна кількість документів. Іншими словами, якщо у нас 40 з 120 об’єктів мають оцінку “Critically”, то апріорна ймовірність для цієї оцінки становить 40/120.

$$P(w|C) = \frac{\text{кількість}(w, C) + 1}{T(C) + |V|} \tag{2.4}$$

На формулі 2.4 де “кількість(w, C)” – кількість входжень окремого токену в об’ єктах класу C; де T(C) – загальна кількість токенів у класі C; “|V|” – розмір

словникового запасу (кількість унікальних токенів). Тобто, якщо слово “admin” зустрічається тридцять разів у всіх об’єктах класу з оцінкою “критично”, а всього у класі триста токенів і в словнику п’ятсот унікальних слів, тоді “P(‘admin’ | ‘Critically’)” = 30 + 1 / 300 + 500. Лапласове згладжування, це константа яка додана спеціально для випадків, якщо токен не входить в перелік вже відомого словника.

$$logP(C|d) \propto logP(C) + \sum_{i=1}^{n} logP(w_i|C) \tag{2.5}$$

На формулі 2.5 для нового об’єкта, в який входять певні токени (w) обчислюємо логарифмічну ймовірність (щоб уникнути числового переповнення при множенні малих чисел). Ми обчислюємо логарифм апріорної ймовірності для класу, а потім додаємо логарифми умовних ймовірностей для кожного токену. Клас із найвищою сумою логарифмічних ймовірностей – це і є ймовірно правильний клас для об’єкта результатів.

$$C = \arg\_\max(logP(C) + \sum_{i=1}^{n} logP(w_i|C) \tag{2.6}$$

В кінці обчислення алгоритм на основі формули 2.6 приймає рішення вибираючи клас, для якого “log P(C|d)” є максимальним. Для інтеграції цього механізму в систему було створено сервіс “ClassifierManager”, який завантажує навчальні дані, тренує баєсівський класифікатор і створює об’єкт для кожного модуля. У контролері система приймає запити користувача, після отримання результатів пошуку, для кожного результату формується комбінований текст, який передається в класифікатор. Отримані мітки додаються до результатів та повертаються у форматі JSON. На стороні клієнта результати відображаються як: назва, посилання, опис, модуль пошуку, мітка (оцінка).

2.4 Висновки до другого розділу

У конкретному розділі було проведено комплексне опрацювання етапів розробки системи автоматизовації генерації пошукових запитів з метою виявлення вразливостей та витоків даних в інтернеті. Паралельно з цим, в реальному часі проводилась розробка веб-додатку у середовищі для розробників Visual Studio. З початку були уточнені вихідні положення для створення системи, тобто ми охарактеризували основну потенційну аудиторію (пентестери, DevOps-фахівці, анатілитики комплаєнсу, розробники), описали їхні потреби та способи взаємодії з системою. Це дало контур функціональних вимог та логіку поведінки користувачів у контексті реальних сценаріїв безпеки.

Важливий крок який був виконаний далі – проектування архітектури системи. Було обгрунтовано вибір сервісно-орієнтованої моделі з використанням архітектурного шаблону MVC, який забезпечує модульність, масштабованість та простоту підтримки. Детально описано ключові сервіси, зокрема “JsonDataService”, “TagsService”, “DorkGeneratorService” та “SearchService”, а також підхід до зберігання шаблонів у форматі JSON. Так як це повністю забезпечує нас, як розробників, гнучкістю при оновленні додатку у майбутньому та додаванні нових функцій, зберігаючи стабільність основного функціоналу.

Не менш особливу увагу отримало питання про класифікацію результатів пошуку, що є критичним для зручності аналізу даних. Розглянули два основні напрямки – евристичні алгоритми та методи машинного навчання. В результаті аналізу переваг і недоліків кожного методу було обрано наївний баєсівський класифікатор як компромісне рішення між складністю реалізації та ефективністю роботи із текстовими даними. Цей класифікатор отримує дані після попередньої обробки результатів, які потім розбиваються на ключові токени.

3 ТЕСТУВАННЯ, РЕЗУЛЬТАТИ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

3.1 Огляд логіки інтерфейсу

Коли розробка проекта перетинає фінішну пряму, приходить час перевірки системи на дієздатність у рамках своєї спеціалізації. Для цього впершу чергу перевіряється інтерфейс системи, так як це те, з чим безпосередньо працює користувач при роботі із додатком. Крім естетичного і гарного вигляду, інтерфейс також має свою логіку, яка включає в себе інформаційну архітектуру, взаємодію між людиною і системою та інтерактивний дизайн – це все називається “user experience” або “UX”. Тобто, це користувацький досвід [34], який отримує людина при взаємодії із нашим продуктом. Отже, наша задача перевірити, чи наданий користувачеві легкий і інтуїтивно зрозумілий досвід. Ця перевірка буде проводитись за допомогою так званих “евристик Якоба Нільсона” [35]. Взагалі цих принципів проектування є декілька і у цьому підрозділі ми пройдемось по ним, щоб взнати чи всі ці правила дотримані у нашому проекті. Ба більше – сучасні підходи до UX враховують не лише зручність але й емоційне сприйняття – чи почувається користувач впевнено, чи викликає система роздратування. Важливу роль відіграє і узгодженість елементів – чи відповідає оформлення очікуванням користувача та загальним патернам поведінки в подібних веб-системах. Навіть такі “дрібниці” як порядок елементів у формі чи шрифт на кнопках впливають на швидкість взаємодії. У якісному UX-проєктуванні немає випадкових рішень – все повинно мати логічне пояснення. Саме тому перевірка за евристиками дозволяє виявити як критичні помилки, так і неочевидні недоліки, які можуть знизити загальну ефективність системи.

Веб-додаток повинен надавати зворотній зв’язок користувачам про те, що відбувається в системі упродовж часу її експлуатації. Цей принцип названий, як “Відображення стану системи” [36]. В першу чергу, це правило дотримується через стан збереженого ключа: коли користувач вводить свій Google API-ключ і зберігає його в сесії, інтерфейс може відображати повідомлення як на

рисунку 3.1. Це інформує, що ключ справді збережено і більше не треба вводити його повторно.

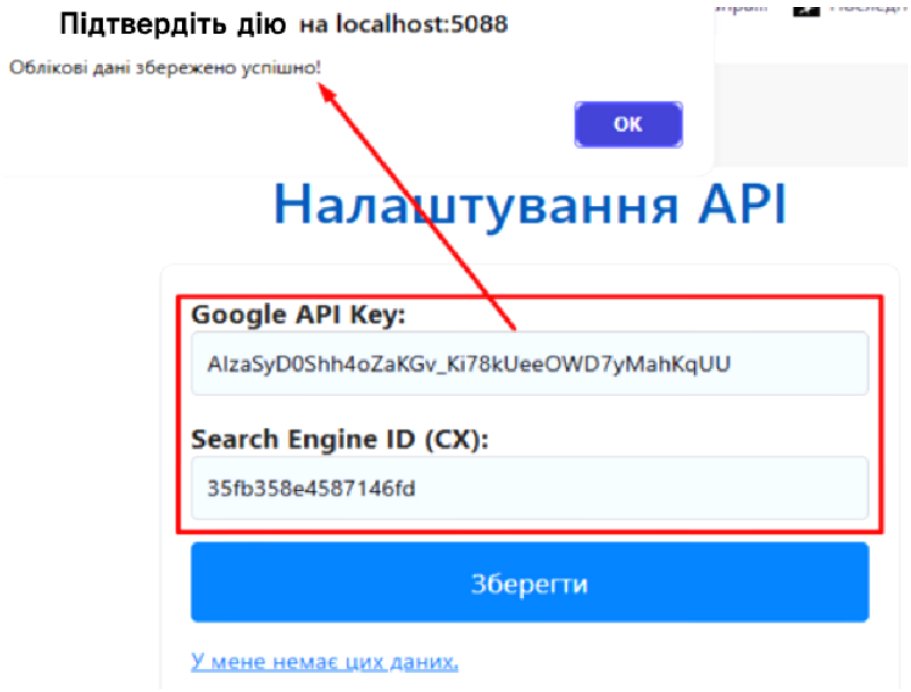


Рисунок 3.1 – Процес заповнення даних

Другий момент – це інформування того, який модуль зараз активний. На рисунку 3.2 видно, що після того як ми вибрали будь-який модуль, кнопка актуального модуля міняє свій колір – це спрощує розуміння поточної конфігурації:

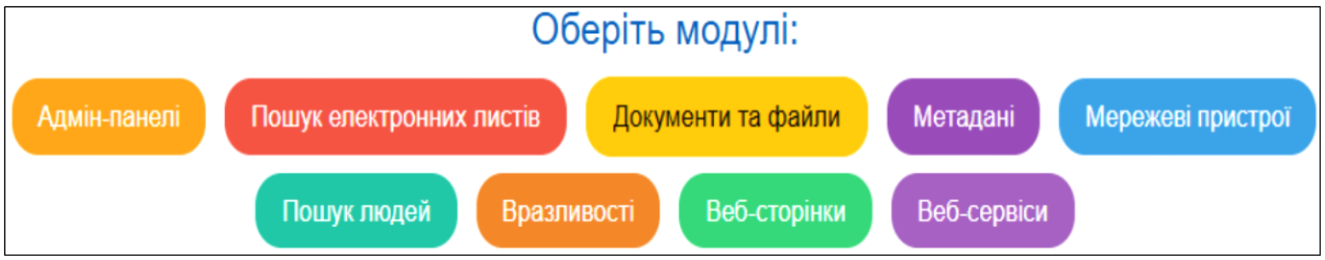


Рисунок 3.2 – Обрали модуль

Після того, як ми вибрали якийсь шаблон – ми отримуємо знову ж таки повідомлення, про те, що ця подія пройшла успішно (рисунок 3.3). Це реалізовано через виклик функції на фронт-енді – за допомогою мови JS.

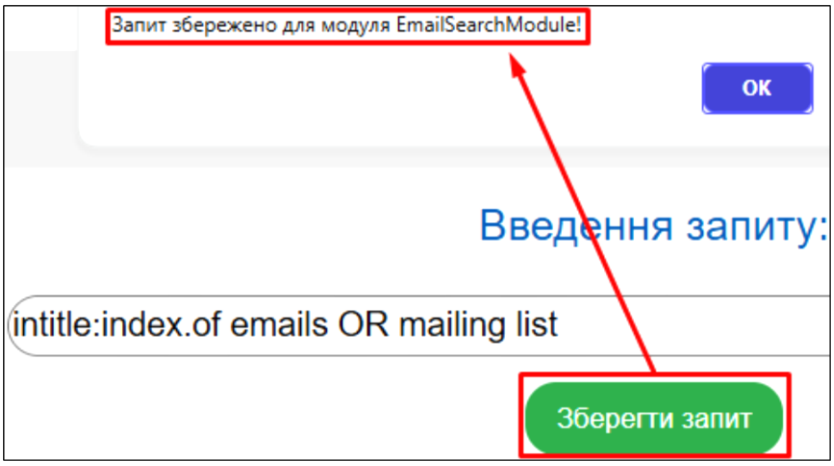


Рисунок 3.3 – Процес додавання запиту

Візуальне підтвердження дій користувача допомагає зменшити когнітивне навантаження – тобто користувачеві не потрібно гадати, чи його дія була врахована системою. Така дрібниця як коротке повідомлення “запит збережено для модуля” формує довіру до системи та створює почуття контролю.

Крім цього, внизу екрана є лічильник, який показує перелік актуальних доданих шаблонів (рисунок 3.4). Коли шаблон додається – лічильник інкремінує нуль і таким чином показує загальну кількість всіх доданих шаблонів. Також, якщо навести на нього вказівник миші, ми побачимо інформацію про шаблони. Цей компонент слугую своєрідною точкою навігації – користувач у будь-який момент може переглянути які саме запити вже збережено. По суті, він працює як буфер для швидкого доступу до результатів, не вимагаючи повторного введення параметрів.

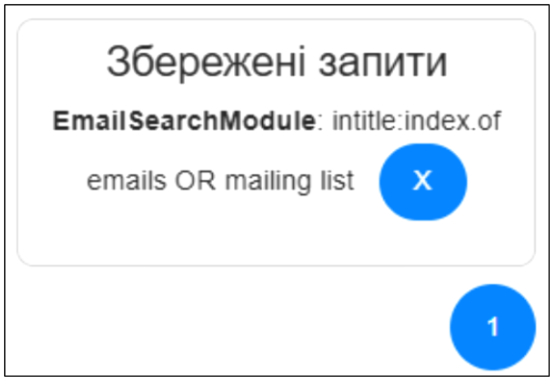


Рисунок 3.4 – Тамбнейл для відображення збережених запитів

Як результат, завдяки цим візуальним підказкам та повідомленням ми подбали про те, щоб користувач завжди розумів актуальний стан в системі. Такі рішення окрім того що підвищують довіру користувача до додатку, тому що він буквально бачить, що його дії не лишаються “в повітрі”, а мають результат, але ще й покращує загальне враження від продукту та прибирають сумніви у користувача, наприклад, чи дійсно шаблон додано. Чим більше таких зрозумілих реакцій у додатку, тим меншою є ймовірність виникнення критичних помилок у процесі взаємодії.

Друге правило “евристик юзабіліті” дбає про те, щоб система “говорила” із користувачем на зрозумілій йому мові, тобто їй потрібно оперувати знайомими йому термінами і логікою. Воно називається “відповідність системи реальному світу” [37]. В контексті цього веб-додатку, цей принцип використовується двічі – замість внутрішньо-технічного жаргону, ми використали зрозумілі назви для назв модулей та шаблонів. Приклади на рисунку 3.5. Через цей підхід ми зробили інтерфейс менш відштовхуючим для новачків та знизили вхідний бар’єр. У багатьох аналогічних системах термінологія надто технічна, що змушує користувача шукати додаткові пояснення.

Рисунок 3.5 – Зрозумілі назви модулів та шаблонів

Отже, якщо термінологія й логіка інтерфейсу будуть близькі до реальних завдань користувача, він легше засвоїть, як формувати потрібні Dork-запити.

Наступний принцип – це “контроль і свобода користувача” [38], він ставить правило, яке каже, що користувачі повинні легко виправляти помилки й мати можливість відмінити якісь свої минулі дії. Так людина відчуває, що вона контролює процес пошуку, а не система диктує їй свої жорсткі правила.

Адаптували для нашого проекту цей принцип через кнопку “скасувати”, яка знаходиться в тамбнейлі, котрий відображає збережені шаблони (рисунок 3.4). Також є можливість редагування сформованого запиту в ручну. Навіть якщо шаблони генеруються автоматично, користувач може самотужки виправляти або розширювати запит без примусового початку “з нуля”.

Принцип, який запобігає помилкам зі сторони користувача і виключає реагування на них – це принцип “Error prevention”. Він адаптований у нашому проекті через виводимі повідомлення в разі пустих запитів, які відправляє користувач. На рисунку 3.6 повідомлення, яке реагує на відправку пустого запиту в Google:

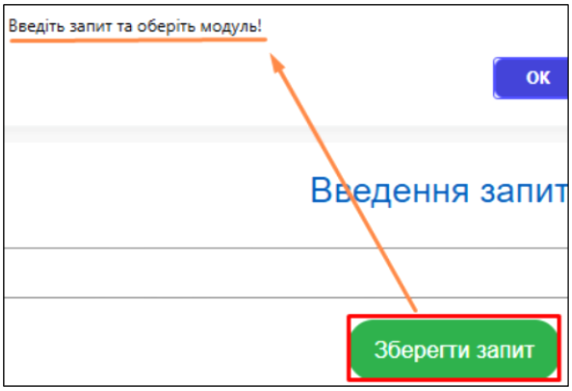


Рисунок 3.6 – Відправили пустий шаблон

На рисунку 3.7 бачимо результат, що буде якщо відправити Google API ключ в неправильному форматі. Завдяки таким превентивним крокам менше ризику, що людина зламає запит або введе не ті дані.

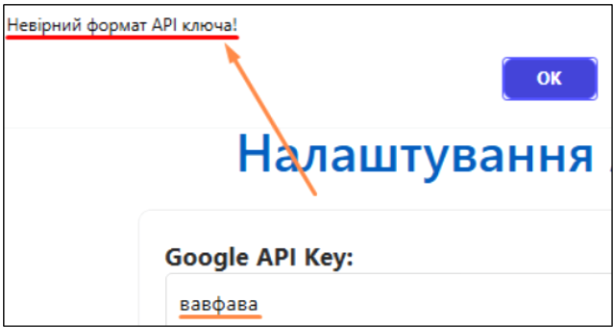


Рисунок 3.7 – Відправка некоректних даних

Також маємо (рисунок 3.8) зручний список доменів і дат, представленні в звичному форматі, який дозволяє людині зорієнтуватись і швидко вибрати потрібний їй параметр. Це виключає людський фактор з боку користувача, так як вводити самому нічого не потрібно.

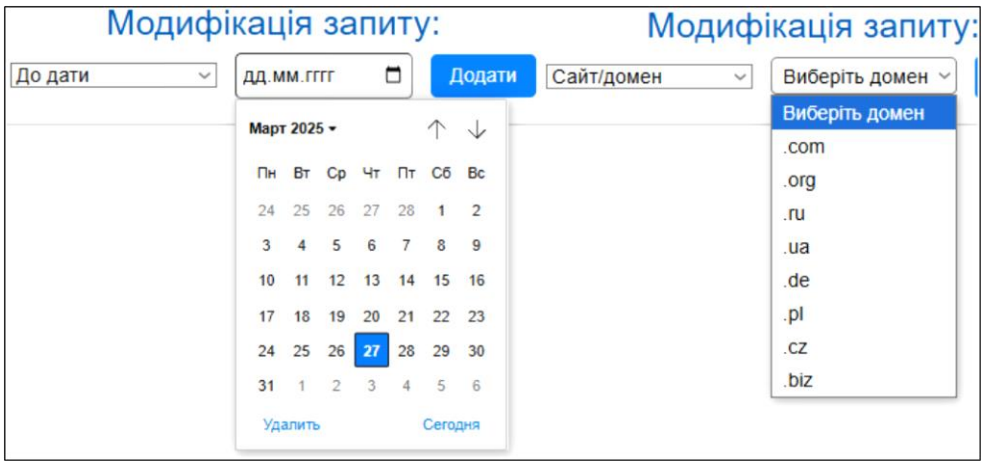


Рисунок 3.8 – Комфортний вибір параметрів

П’ятий принцип відповідає за гнучкість і ефективність використання нашого додатку, тобто ми повинні задовольняти потреби як новачків так і досвідчених користувачів, пропонуючи скорочені шляхи та доступність для кожного [39]. Перше, що приніс в проект цей принцип – це документація, тобто гайд для недосвідчених користувачів по використанню (рисунок 3.9). До речі, ця документація також стосується принципу “help and documentation”.

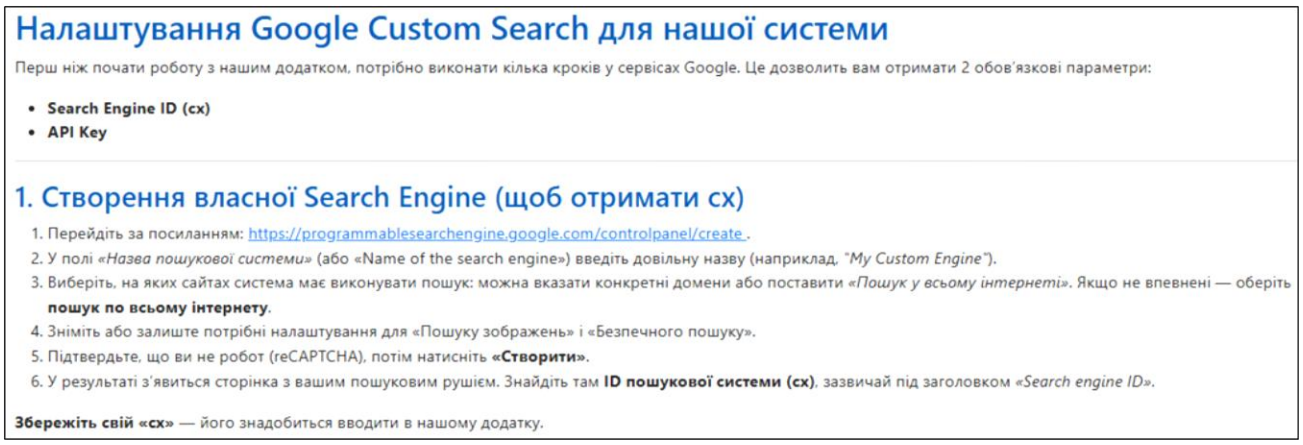


Рисунок 3.9 – Вигляд сторінки гайду

На рисунку 3.10 показано другий приклад правила про короткі шляхи і гнучкість системи. На ньому зображено тамбнейл, який показує збережені людиною запити. Тут можна або видалити їх або запустити пошук результатів відразу за двома чи більше вибраними шаблонами. Це робить роботу більш ефективною та швидкою.

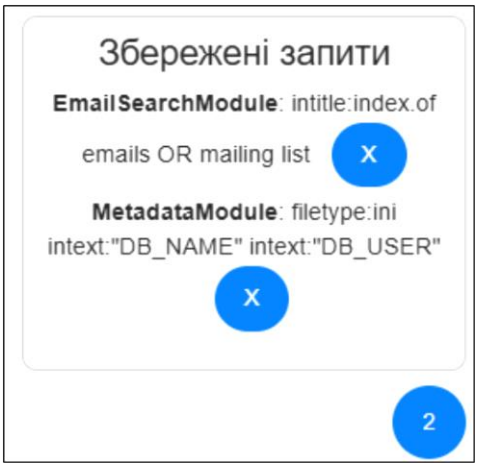


Рисунок 3.10 – Тамбнейл з двома запитами

Завдяки останньому евристичного принципу Якоба Нільсона, який називається “естетичний і мінімалістичний дизайн” ми проектуємо інтерфейс так аби він не містив зайвої інформації та візуального шуму. На сторінці де людина вибирає модуль і теги, а потім конструює свій запит (рисунок 3.11) мінімум полів і кнопок, ми виводимо лише ключову інформацію і не перевантажуємо користувача інтерфейсом.

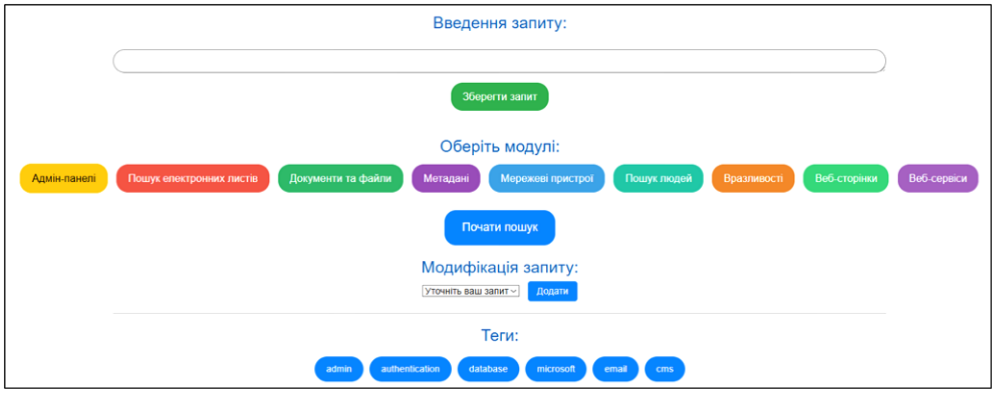


Рисунок 3.11 – Вигляд сторінки запуску пошуку

Також, як можна помітити у нас присутня візуальна ієрархія. Інформація перетікає зверху вниз; спочатку користувач бачить модулі, а потім вибирає потрібний тег, а далі отримує список шаблонів. Це створює звичний для користувача логічний зв'язок і допомагає швидше зорієнтуватись у інтерфейсі.

Отож, переглянуті евристики дозволяють поглянути на інтерфейс цього продукту з позиції зручності та ефективності для користувача. Застосування цих правил допомогло зробити систему передбачуваною і прозорою завдяки візуальним підказкам і статусам; запобігати помилкам та давати свободу маніпулювання запитами аби людина відчувала контроль.

3.2 Сценарне тестування системи

Коли інтерфейс оцінений з точки зору комфорту і ефективності роботи для користувача, потрібно також протестувати сценарії, щоб зрозуміти, як саме працюватиме програмне забезпечення, коли його використовуватиме кінцевий клієнт. У цьому підрозділі ми розглянемо характеристики, процеси та методи тестування сценаріїв імітуючи реальне використання веб-додатку. Тестування сценаріїв [40] – це метод тестування ПЗ, що використовує умовні історії. Його проводять щоб переконатись чи система коректно працює від початку і до кінця, а всі її процеси функціонують належним чином. Іншими словами, нам, як тестувальникам, потрібно уявити себе кінцевими користувачами і змоделювати реальні сценарії або варіанти використання, які можуть бути здійснені людьми у нашому ПЗ.

Ситуація, яка буде розігруватись у цьому тестуванні побудована навколо одного уявного користувача: фахівець з безпеки, який вперше чує про наш додаток. У нього буде певна ціль (знайти файли з паролями), яку він буде переслідувати за допомогою системи AbyssDorks. Методика дослідження полягає у тому, щоб пройти самотужки шлях від початку до знаходження поставленої цілі, а потім описати кроки користувача та побудувати їх графічний

алгоритм з підсумковою таблицею. Так ми переймемо весь реальний досвід на себе і дізнаємось де можуть виникнути питання.

Перше, що зустрічає користувач на екрані перед собою – це сторінка з введенням своїх даних (рисунок 3.12). Тут він розуміє що він не має Google API ключа та Search Engine ID. Логічно, що він натисне на відповідну кнопку “у мене немає цих даних”.

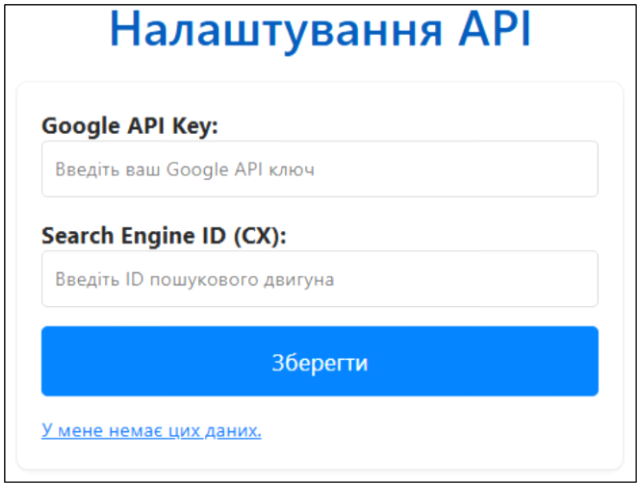


Рисунок 3.12 – Сторінка введення даних

Потім він переходить на сторінку з гайдом (рисунок 3.13) і бачить кроки, як отримати Search Engine ID та API ключ. За інструкціями він створює новий пошуковий рушій і отримує CX; переходить до Google Cloud Console та створює API Key.

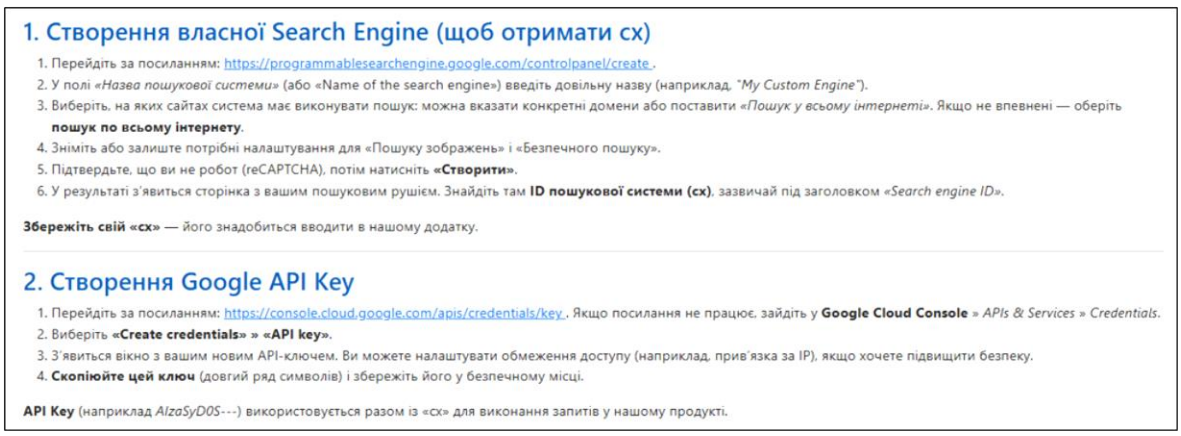


Рисунок 3.13 – Гайд як отримати потрібні дані

Повертається в інтерфейс нашого додатку й у полі введення прописує вже отримані дані (рисунок 3.14). Натискає кнопку зберегти, і тоді система повідомляє його, що “ключі збережено” й перенаправляє його на головну сторінку.

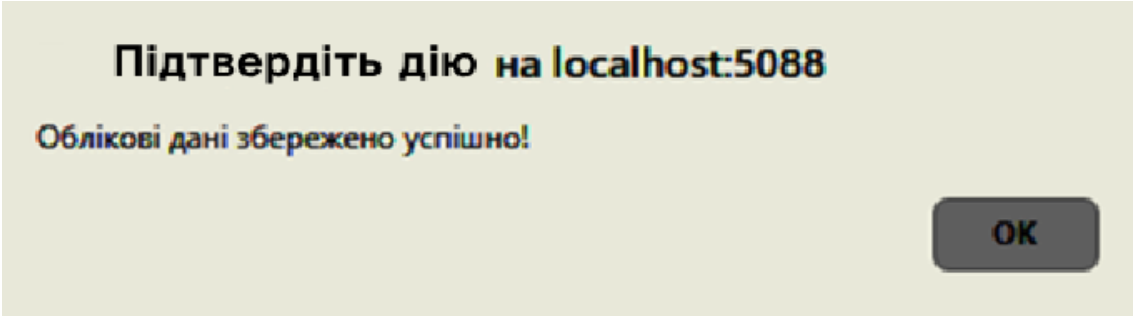


Рисунок 3.14 – Ключі збережено

Нарешті користувач попадає на основну сторінку та бачить перед собою список модулів (рисунок 3.15). Так як його ціль – це знайти файли із паролями, логічно, що він вибере кнопку з текстом “файли та документи”.

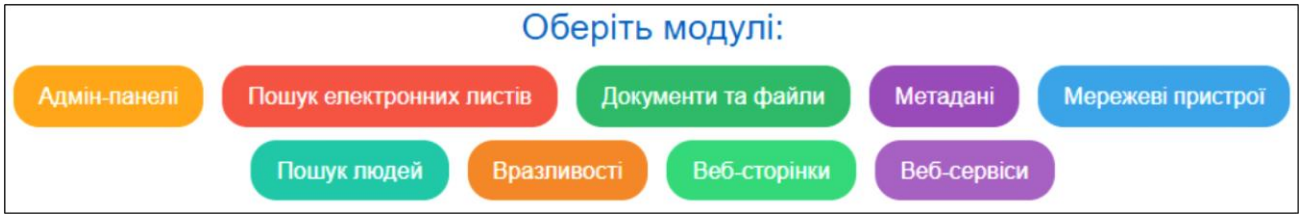


Рисунок 3.15 – Список модулів

Далі він отримує список тегів, які належать вибраному модулю (рисунок 3.16). Вибирає тег “files” (файли) і отримує список шаблонів цього тегу. Табличний вид шаблонів з колонками “назва, опис, підказка та шаблон” особливо зручний, тому що користувач таким чином бачить всю інформацію про шаблони. Так як йому потрібні файли з паролями, користувач проходиться по назвам і помічає назву пов’язану з паролями. Вибирається шаблон “filetype:txt intext:"password" – він шукає файли в форматі “.txt” який містить рядок “password”.

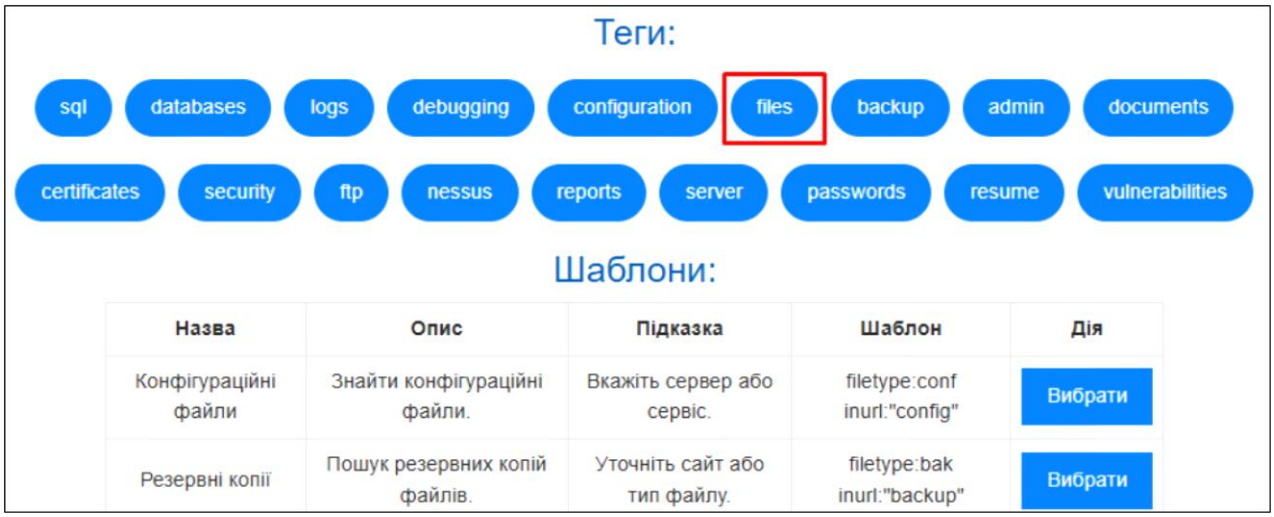


Рисунок 3.16 – Теги та список шаблонів

Тепер користувач бачить шаблон, який він вибрав у полі вводу та нажимає з початку на кнопку “зберегти запит”, а потім “почати пошук”. Після цього виконується відправка шаблону через Google API та повертаються результати в систему; результати виводяться (рисунок 3.17):

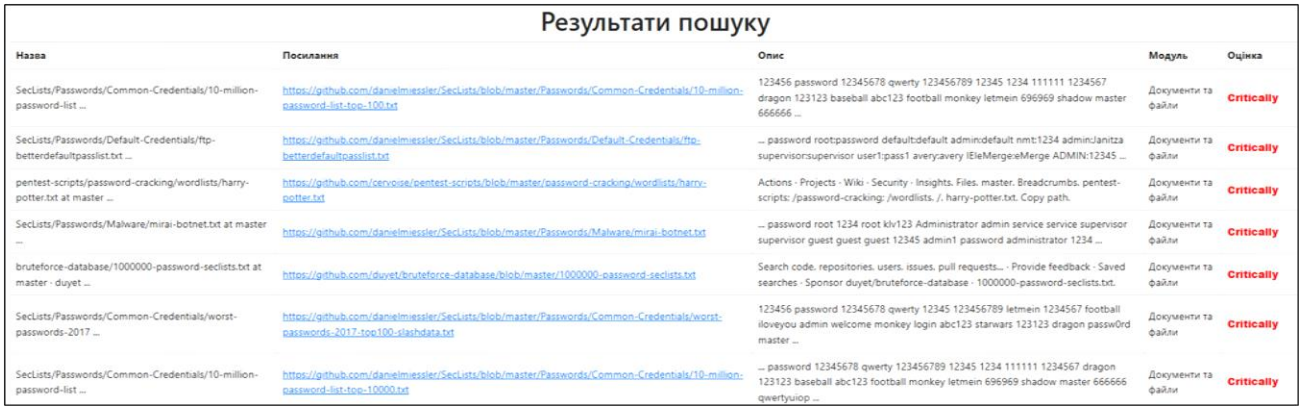


Рисунок 3.17 – Отримали перші результати

Користувач отримує результати, помічає оцінку “критично”, переходить по знайденим об’єктам і бачить дійсно паролі. Але проблема у тому, що це словники для брутфорсу від GitHub, а не реальні паролі, які хотілося б бачити. Тоді користувач повертається сторінкою назад і бачить блок модифікації запиту (рисунок 3.18).

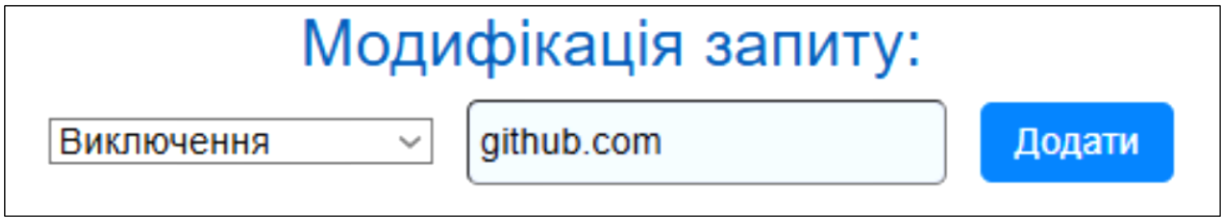


Рисунок 3.18 – Модифікація запиту

Цей випадок ідеальний, щоб скористуватись цією функцією. Користувач бачить список з дій, серед який є дія “виключення” – він вводить в поле сайт, який треба виключити з пошуку, нажимає “додати” і отримує в кінці готовий запит “filetype:txt intext:"password" -github.com”.

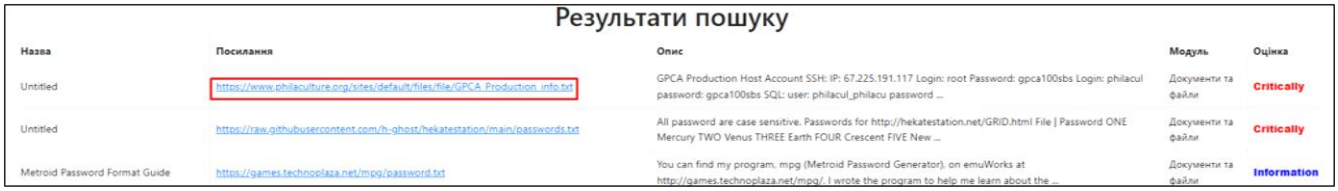


Рисунок 3.19 – Отримали нові результати

Після того, як користувач сформував новий запит і виконав по ньому пошук – отримав нові результати (рисунок 3.19). Серед них помітно один об’єкт з критичною оцінкою, саме по ньому він переходить і бачить реальний пароль (рисунок 3.20) та інші конфіденційні дані, які чомусь не захищені і знаходяться у відкритому доступі:

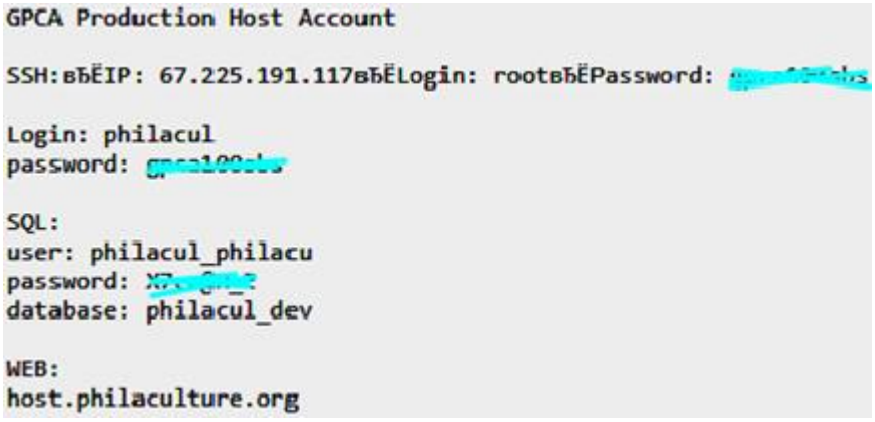


Рисунок 3.20 – Знайдена ціль з паролем

Коли сценарне тестування завершене, побудуємо підсумкову таблицю 3.1 для відображення кроків, очікуваних та фактичних результатів, а також статусу тесту.

Таблиця 3.1 – підсумкова таблиця тестування

Крок	Очікуваний результат	Фактичний результат	Статус
1	2	3	4
Користувач відкриває додаток і бачить форму заповнення даними	Є форма і поля для вводу ключів; напис “у мене немає цих даних”	Поля для ключів є, посилання на гайд відкриває інструкцію	ОК
Користувач переходить до гайду, створює ключі, повертається і вводить у форму. Натискає “зберегти”	Після натискання “зберегти” система інформує “ключі збережено”, перенаправляє на головну.	Після збереження ключів бачимо сповіщення “ключі збережено”, відбувається редирект.	ОК
На головній користувач бачить список модулів	Система відобразить кнопки на всі модулі	Всі 9 модулів відображаються	ОК
Користувач обирає модуль “FilesDocumentModule”	Появляється список тегів, які належать до модуля	Теги виведені	ОК

Кінець таблиці 3.1

1	2	3	4
Натискає “зберегти запит”, потім “почати пошук”	Запит повинен з’явитись у “збережених”. Відбувається виклик Google API. Повертаються перші результати	Вибраний запит збережений. Система показала список із 10-12 посилань.	ОК
Користувач знаходить потрібні файли (.txt з реальними паролями), перевіряє їх в браузері	Користувач бачить, що файл відкритий, є паролі всередині	Все успішно. Ціль досягнута	ОК

Нижче, на рисунку 3.21 зображено графічний алгоритм, що дає огляд послідовності проведеного тестування у стилі блок-схеми.

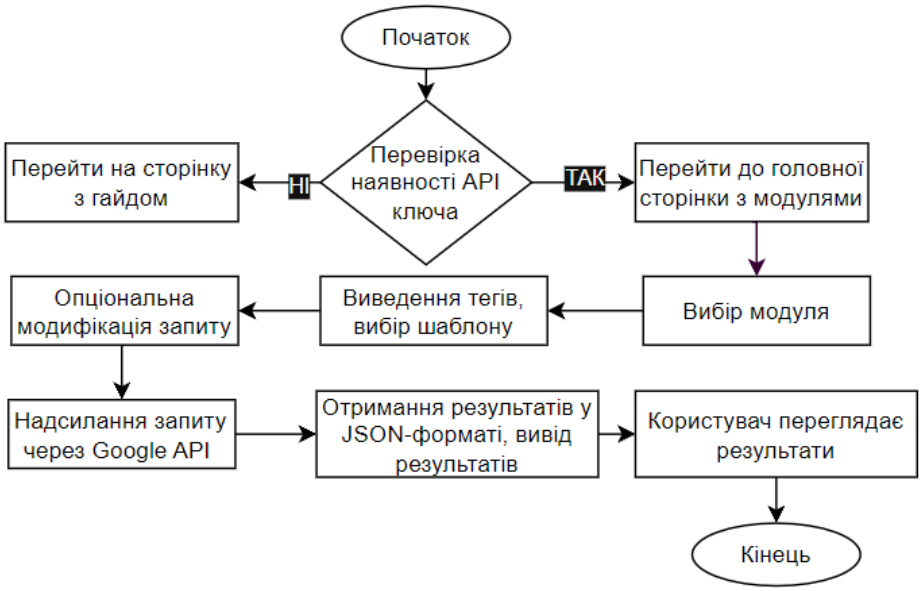


Рисунок 3.21 – Алгоритм кроків користувача у системі

Проведене і описане у цьому підрозділі сценарне тестування, яке передбачило симуляцію реальних умов використання системи та моделювання дій потенційного користувача пройшло успішно. Завдяки такому підходу, було досягнуто декількох цілей, а саме перевірки працездатності додатку від початку і до кінця та оцінки логіки і зручності (UX) в “бойових умовах”. У результаті підтверджено, що AbyssDorks успішно працює “end-to-end” для пошуку цікавих (у тому числі критичних) файлів в інтернеті. Сценарне тестування продемонструвало узгодженість усіх компонентів: від введення ключів і вибору модуля до відображення релевантних посилань. Більше того, у рамках сценарію умовному користувачеві дійсно вдалось знайти реальний файл із паролями, що символізує коректність, потенціал і цільову ефективність додатку.

3.3 Висновки до третього розділу

У межах третього розділу ми здійснили всебічну перевірку функціональності веб-додатку як з точки зору інтерфейсної взаємодії, так і з погляду практичного використання у реальному сценарії. Перевірка інтерфейсу базувалась на відомих UX-принципах, зокрема евристиках Якоба Нільсона. Таким способом оцінено не лише зовнішній вигляд, а й логіку, зручність та послідовність взаємодії користувача із системою. Результати аналізу довели, що інтерфейс додатку дотримується ключових принципів зручності: система забезпечує чіткий зворотний зв’язок, використовує мову зрозумілу користувачеві тощо. Сценарне тестування на прикладі змодельованого випадку використання (з пошуком файлів із паролями) показало, що система ефективно виконує покладені на неї функції. Більше того, ми, як користувач, змогли пройти шлях від ініціалізації налаштувань до знаходження потрібної інформації швидко та без перешкод, та з логічною послідовністю кроків. Гарним закінченням тестування було досягнення практичного результату – виявлення чутливої інформації у відкритому доступі.

ВИСНОВКИ

Головною метою цієї дипломної роботи стала розробка системи генерації пошукових запитів для виявлення вразливостей та витоків конфіденційних даних у відкритому доступі. Було реалізовано повний життєвий цикл створення програмного забезпечення – від зародження ідеї до її практичної реалізації й апробації у одного кінцевого користувача. Результатом стала веб-платформа AbyssDorks, що об’єднала сучасні підходи до автоматизації пошуку, елементи машинного навчання та концепцію open-source-intilligence (OSINT).

На початковому етапі було абстрактно змодельовано ідею системи, визначено її функціональні межі, цілі та обрано цільову аудиторію. Передбачалось, що основними користувачами додатку стануть фахівці з інформаційної безпеки, пентестери, DevOps-інженери, розробники та інші представники IT-сфери, які мають потребу у швидкому та точному виявленні потенційно відкритих або уразливих ресурсів в публічному доступі.

У ході аналітичного дослідження було розглянуто феномен витоку даних в інтернеті, його природу та найбільш поширені джерела витоку. Сучасні IT-інфраструктури дедалі частіше стикаються з несанкціонованими витоками даних: помилкова конфігурація хмарних бакетів, залишені у публічних репозиторіях ключі доступу, відкриті резервні копії тощо. Для аналізу таких інцидентів використовується широкий спектр OSINT-технік. У рамках роботи, як ключовий інструмент для доступу до таких ресурсів було обрано немаловідомий пошуковик Google, зокрема їх потужний механізм – Google Dorks. Це спеціалізовані оператори, котрі дозволяють здійснювати високоточні пошукові запити для виявлення в тому ж числі публічно доступних але потенційно конфіденційних даних.

Було виділено дев’ять категорій, що найчастіше містять критичні дані: некоректно захищені адміністраторські веб-консолі; резервні копії, конфіги, службові документи; SQL-дампи, відкриті Mongo/Elastic кластери; пошук осіб; індексація S3, Azure, GCS; сервісні журнали, build-логи тощо; бекап директорії;

файли з паролями, токенами, ключами API; відкриті інтерфейси мережевих пристроїв. Саме на цій класифікації базується модульна структура AbyssDorks. Кожен модуль містить набор шаблонів dork-запитів, оформлених у вигляді JSON-файлів: назва, опис, підказка щодо вживання, сам запит. Такий підхід робить систему відкритою до розширення.

Щоб результати пошуку були максимально інформативними та не перевантажували користувача “сирими” даними, було розроблено власний механізм оцінки релевантності. Кінцева мета – класифікувати знайдену інформацію за ступенем важливості. Для цього була обрана і реалізована базова модель з області машинного навчання – метод наївного Баєса. Даний підхід дозволив реалізувати просту але ефективну класифікацію результатів на три категорії: “інформаційний”, “середня критичність” та “критичний”.

Для навчання цієї моделі був створений власний набір даних (дата-сет), який вручну наповнювався реальними результатами Google-пошуку. Кожен запис був оцінений за згаданою шкалою, що дало змогу в подальшому використовувати цей набір як основу для порівняння нових результатів. Таким чином, кожен новий пошуковий результат, який система отримує через API автоматично проходить класифікацію за допомогою моделі, що значно підвищує ефективність і точність інтерпретації знайденої інформації.

Центральною частиною дослідження стало сценарне тестування. У ролі персонажа виступив автор цієї записки, якому потрібно було за короткий час з’ясувати чи не знаходяться випадково у вільному доступі списки чужих паролей. Послідовність дій відобразилась у шести кроках: від заходу у веб-додаток до знаходження самої цілі – це доказ повної ефективності і реальної користі цього веб-додатку.

Цей проект позиціонується як middleware – проміжна ланка між пошуковими рушіями та експертом-аналітиком. Це надає низку переваг: додаток замінює ручне складання dork-ланцюжків і однотипний перегляд сотень URL; кольорова індикація допомагає миттєво фокусуватись на критичних знахідках, економлячи до 60% робочого часу; платформа несе навіть і освітню цінність –

може бути використана як приклад практичних аспектів OSINT. Перспективою цього проекту може бути інтеграція інших пошукових рушіїв, таких як Bing, DuckDuckGo, Shodan або Censys; перехід із баєсівської моделі на потужні класифікатори з контекстним розумінням; для DevOps-інженерів як перспектива може бути інтеграція API додатку для CI-конвеєрів, котрий буде запускати скан під час збірки та блокувати реліз при виявленні витоків у вихідному коді.

У результаті виконання цього дипломного проєкту здійснено комплексне моделювання загроз витоку даних та синтезовано модульну бібліотеку dork-шаблонів (більше 200 одиниць), розроблено веб-додаток із кросплатформеною архітектурою та впроваджено класифікатор подібний до справжніх ML, який забезпечує автоматичну пріоритизацію результатів із точністю близько 80%. Шляхом сценарного тесту доведено практичну цінність системи – виконано успішну ідентифікацію критичного витоку. Отже, поставлені в роботі цілі досягнуто повністю. Створене рішення не лише має навчальну та дослідницьку, а й прикладну вартість – воно може використовуватись фахівцями різних IT-сфер для оперативного виявлення вразливих ресурсів. Це в свою чергу буде сприяти підвищенню загальної культури інформаційної безпеки у цифровому суспільстві.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Cost of a Data breach report. IBM Security. URL: https://d110erj175o600.cloudfront.net/wp-content/uploads/2023/07/25111651/Cost-of-a-Data-Breach-Report-2023.pdf (дата звернення: 09.03.2025)
2. Cost of a Data breach report 2022. Krontech. URL: https://krontech.com/highlights-of-the-2022-cost-of-a-data-breach-report (дата звернення: 09.03.2025)
3. Загальний регламент про захист даних. GDPR. URL: https://www.gdpr.org.ua/ (дата звернення: 09.03.2025)
4. Каліфорнійський закон про захист прав споживачів. eSputnik. URL: https://esputnik.com/blog/california-consumer-privacy-act (дата звернення: 10.03.2025)
5. Штраф Фейсбуку зі сторони EDPB. Edpb. URL: https://www.edpb.europa.eu/news/news/2023/12-billion-euro-fine-facebook-result-edpb-binding-decision_en (дата звернення: 10.03.2025)
6. Yahoo agree to lowered $4.48 billion deal following cyber attacks. Reuters. URL: https://www.reuters.com/article/business/verizon-yahoo-agree-to-lowered-448-billion-deal-following-cyber-attacks-idUSKBN1601EK/ (дата звернення: 10.03.2025)
7. Наслідки витоків даних. MetaCompliance. URL: https://www.metacompliance.com/blog/data-breaches/5-damaging-consequences-of-a-data-breach (дата звернення: 10.03.2025)
8. Як працюють Google Hacking. Imperva. URL: https://www.imperva.com/learn/application-security/google-dorking-hacking/ (дата звернення: 10.03.2025)
9. Використання Google-dorking в OSINT. Maltego. URL: https://www.maltego.com/blog/using-google-dorks-to-support-your-open-source-intelligence-investigations/ (дата звернення: 10.03.2025)

10. Оператори в Google Dorks. RecordedFuture. URL: https://www.recordedfuture.com/threat-intelligence-101/threat-analysis-techniques/google-dorks (дата звернення: 10.03.2025)

11. Пошук вразливостей безпеки за допомогою автоматизованих ботнетів. Imperva. URL: https://www.imperva.com/docs/HII_The_Convergence_of_Google_and_Bots_-_Searching_for_Security_Vulnerabilities_using_Automated_Botnets.pdf (дата звернення: 10.03.2025)

12. Топ 5 пошуковиків для дослідників безпеки. SOCRadar. URL: https://socradar.io/top-5-cyberspace-search-engines-used-by-security-researchers/ (дата звернення: 11.03.2025)

13. Інструменти пасивної розвідки в кібербезпеці. ZTM. URL: https://zerotomastery.io/blog/passive-reconnaissance-tools/ (дата звернення: 11.03.2025)

14. Інструмент FOCA. HackersArise. URL: https://hackers-arise.com/using-foca-to-gather-website-metadata/ (дата звернення: 11.03.2025)

15. EmeraldWhale: глобальна операція крадіжки хмарних облікових даних з конфігурацій git. Sysdig Blog. URL: https://sysdig.com/blog/emeraldwhale/ (дата звернення: 11.03.2025)

16. Витік креденшалів через GIT конфігурацію. CyberScoop. URL: https://cyberscoop.com/sysdig-git-credentials-cloud-service-emeraldwhale/ (дата звернення: 11.03.2025)

17. Що таке Git. Git-scm. URL: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F (дата звернення: 12.03.2025)

18. Захист Git-репозиторіїв. Medium. URL: https://medium.com/@cuncis/protecting-your-git-repositories-a-comprehensive-guide-to-using-gitleaks-for-securing-sensitive-323fd5fc9638 (дата звернення: 12.03.2025)

19. Custom Search JSON API. Google developers. URL: https://developers.google.com/custom-search?csw=1&hl=ru (дата звернення: 12.03.2025)

20. База даних google-dorking. Exploit-DB. URL: https://www.exploit-db.com/google-hacking-database (дата звернення: 12.03.2025)

21. Основні види архітектури програмного забезпечення. Art of BA. URL: https://www.artofba.com/post/main-types-of-software-architecture-ru (дата звернення: 14.03.2025)

22. Що таке монолітна архітектура. IBM. URL: https://www.ibm.com/think/topics/monolithic-architecture (дата звернення: 14.03.2025)

23. Що таке мікросервісна архітектура. GeeksForGeeks. URL: https://www.geeksforgeeks.org/what-are-the-advantages-and-disadvantages-of-microservices-architecture/ (дата звернення: 14.03.2025)

24. What is SOA. AWS. URL: https://aws.amazon.com/what-is/service-oriented-architecture/ (дата звернення: 15.03.2025)

25. Ознайомлення з патерном MVC (Model-View-Controller). JavaRush. URL: https://javarush.com/ua/groups/posts/uk.2536.chastina-7-oznayomlennja-z-paternom-mvc-model-view-controller (дата звернення: 17.03.2025)

26. Що таке JSON. W3Schools. URL: https://www.w3schools.com/whatis/whatis_json.asp (дата звернення: 19.03.2025)

27. Евристичний аналіз. Studfile. URL: https://studfile.net/preview/3894673/page:2/ (дата звернення: 22.03.2025)

28. Логістична регресія. Habr. URL: https://habr.com/ru/companies/io/articles/265007/ (дата звернення: 22.03.2025)

29. Орел’єн Жерон. Практичне машинне навчання – концепції, інструменти та техніки. Канада: O’Reilly Media, 2019. 172 с.

30. Дерева рішень . Кафедра ПЗС ДНУ. URL: https://pzs.dstu.dp.ua/DataMining/tree/index.html (дата звернення: 26.03.2025)

31. Що таке глибинне навчання. Oracle. URL: https://www.oracle.com/cis/artificial-intelligence/machine-learning/what-is-deep-learning/ (дата звернення: 12.04.2025)

32. Педро Домінгос. Верховний алгоритм. Київ: Фербер, 2015. 167 с.

33. On the Complexity of Learning Decision Trees. SIGecom Exchanges. URL: https://www.sigecom.org/exchanges/volume_20/1/ZAMPETAKIS.pdf (дата звернення: 14.04.2025)

34. Що таке UX. Goldweb Solutions. URL: https://goldwebsolutions.com/uk/blog/shho-take-ux/ (дата звернення: 16.04.2025)

35. Ten Usability Heuristics for User Interface Design. Nielsen Norman Group. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 20.04.2025)

36. Принцип відображення стану системи в UX. CareerFoundry. URL: https://careerfoundry.com/en/blog/ux-design/usability-heuristics/ (дата звернення: 20.04.2025)

37. Принцип “Відповідність системи реальному світу” в евристиках Нільсена. Medium. URL: https://medium.com/@yahiazakaria445/nielsens-10-usability-heuristics-with-examples-4d4cc5e3fe18 (дата звернення: 20.04.2025)

38. Принцип “контроль та свобода користувача” в UX. UX247. URL: https://ux247.com/usability-principles/ (дата звернення: 20.04.2025)

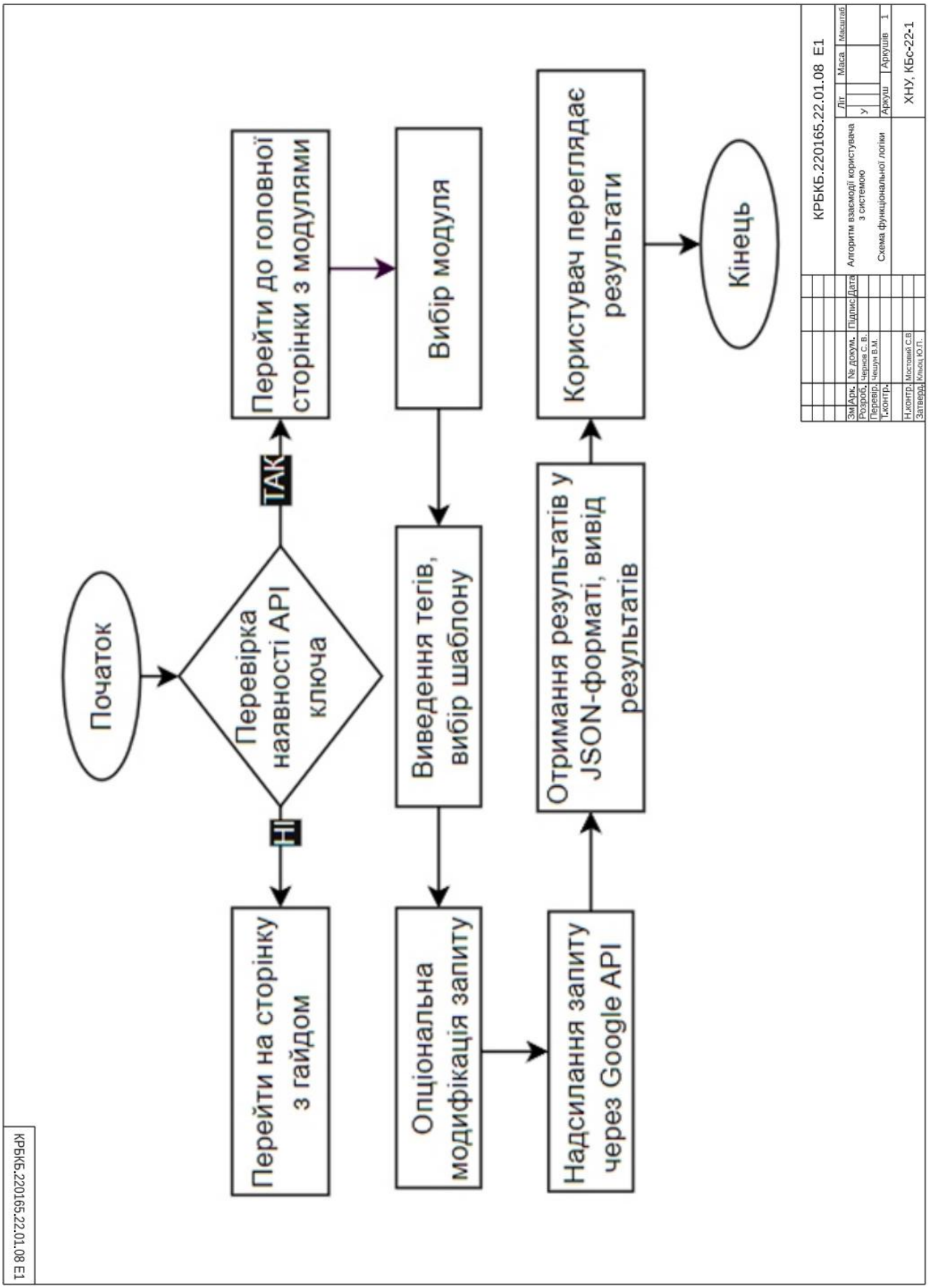
39. Гнучкість та ефективність в UX. UXTigers. URL: https://www.uxtigers.com/post/usability-heuristics-history (дата звернення: 23.04.2025)

40. Software Testing – Scenario Testing. GeeksForGeeks. URL: https://www.geeksforgeeks.org/software-testing-scenario-testing/ (дата звернення: 25.04.2025)

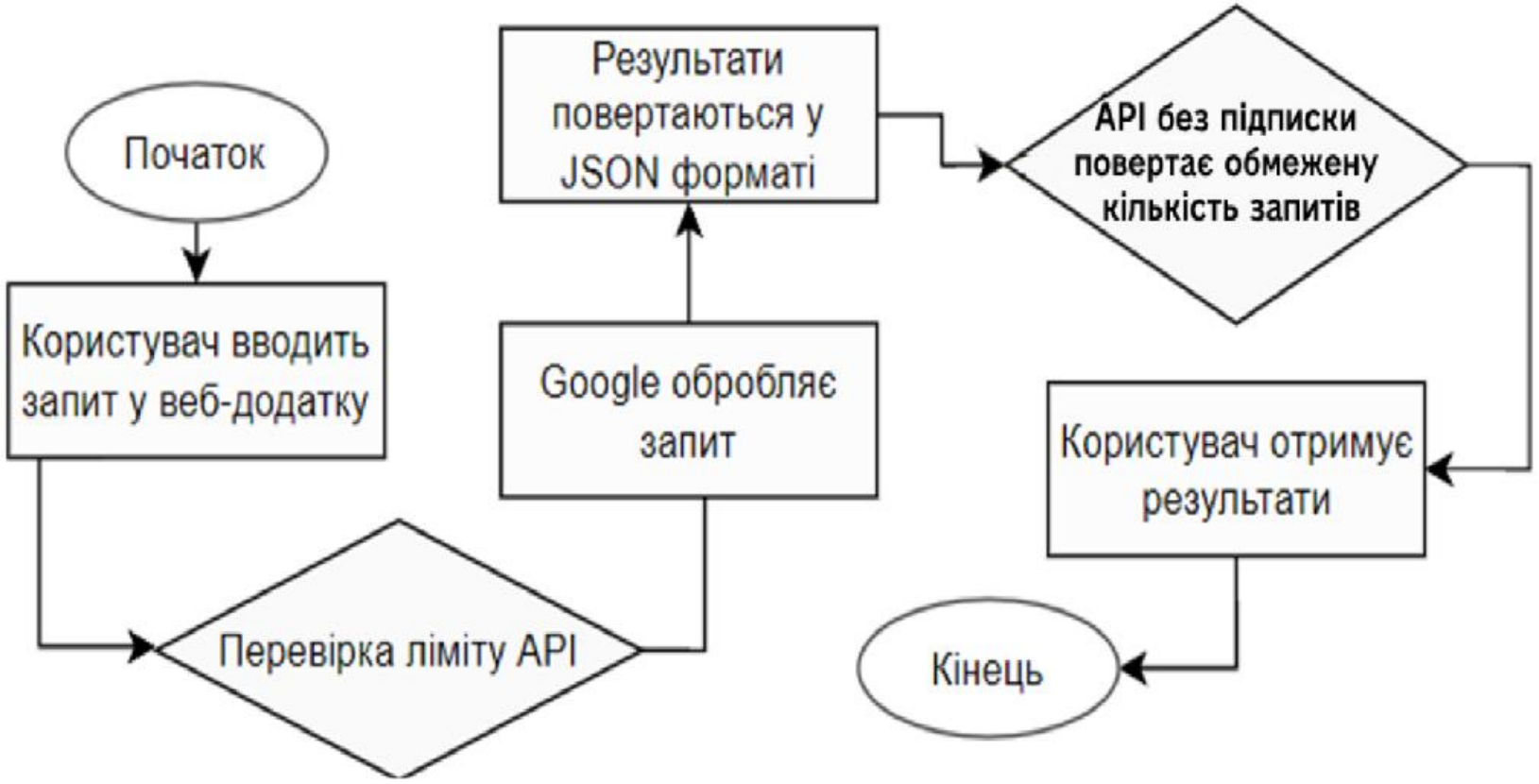
ДОДАТОК А

(обов’язковий)

Копія графічної частини

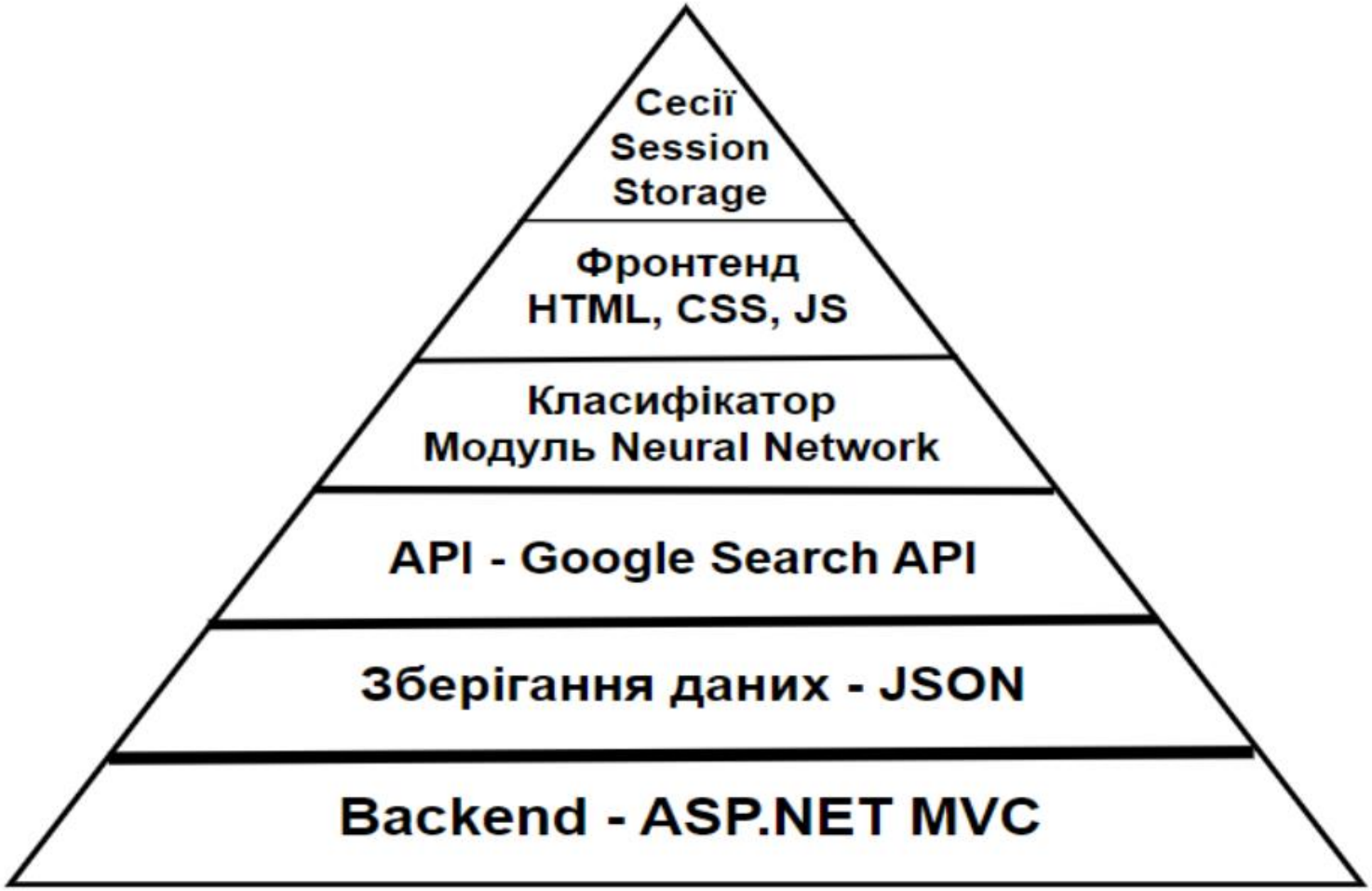


КРБКБ.220165.22.01.08 Е1



					КРБКБ.220165.22.01.08 Е1		
Зм.	Арк.	№докум.	Підпис	Дата	Алгоритм пошуку через Google API Діаграма послідовності дій	Літ	Маса	Масштаб
Розроб.		Чернов С.В.				У		
Перевір.		Чешун В.М.						
Т.контр.						Аркуш	Аркушів 1	
Н.контр.		Мостовий С.В.				ХНУ,КБс-22-1		
Затверд.		Кльоц Ю.П.						

КРБКБ.220165.22.01.08 Е1



					КРБКБ.220165.22.01.08 Е1		
Зм.	Арк.	№докум.	Підпис	Дата	Набір технологій у вигляді архітектури	Літ	Маса / Масштаб
Розроб.		Чернов С.В.				У	
Перевір.		Чешун В.М.				Аркуш	Аркушів 1
Т.контр.							
Н.контр.		Мостовий С.В.				ХНУ, КБс-22-1	
Затверд.		Кльоц Ю.П.					

ДОДАТОК Б

(обов’язковий)

Лістинг коду

```
public class AdminPanelsModuleClassifier : IClassifier
{
   private readonly NaiveBayesClassifier _classifier;
   private readonly ValidationService _validationService;

   public AdminPanelsModuleClassifier(NaiveBayesClassifier classifier, ValidationService validationService)
   {
      _classifier = classifier;
      _validationService = validationService;
   }

   public Task<ClassificationResult> ClassifyAsync(string text)
   {
      string cleanedText = _validationService.NormalizerTextForClassification(text);
      var logProbs = _classifier.GetClassLogProbabilities(cleanedText);
      string predictedLabel = _classifier.Predict(cleanedText);
      return Task.FromResult(new ClassificationResult
      {
         PredictedLabel = predictedLabel,
         LogProbabilities = logProbs
      });
   }
}

public class ClassificationResult
 {
    public string? PredictedLabel {  get; set; }
    public Dictionary<string, double>? LogProbabilities { get; set; }
 }

public class ClassifierManager
{
   readonly ValidationService _validationService;
   private readonly Dictionary<string, IClassifier> _classifiers;
   private readonly string _learningDataPath = Path.Combine(AppContext.BaseDirectory, "BayesClassifier",
"LearningData");

   public ClassifierManager(ValidationService validationService)
   {
      _validationService = validationService;
      _classifiers = new Dictionary<string, IClassifier>();
      string adminDataFile = Path.Combine(_learningDataPath, "data_AdminPanels.txt");
      var naiveBayesAdmin = new NaiveBayesClassifier(_validationService);
      List<Models.Document> adminDocs = ReadTrainingData(adminDataFile);
      naiveBayesAdmin.Train(adminDocs);
      _classifiers["AdminPanelsModule"] = new AdminPanelsModuleClassifier(naiveBayesAdmin, _validationService);
      string fileDocDataFile = Path.Combine(_learningDataPath, "data_FilesDocument.txt");
      List<Models.Document> fileDocs = ReadTrainingData(fileDocDataFile);
      var naiveBayesFiles = new NaiveBayesClassifier(_validationService);
      naiveBayesFiles.Train(fileDocs);
      _classifiers["FilesDocumentModule"] = new AdminPanelsModuleClassifier(naiveBayesFiles, _validationService);
      string metaDataFile = Path.Combine(_learningDataPath, "data_Metadata.txt");
      List<Models.Document> metaDocs = ReadTrainingData(metaDataFile);
      var naiveBayesMetaData = new NaiveBayesClassifier(_validationService);
```

```
        naiveBayesMetaData.Train(metaDocs);
        _classifiers["MetadataModule"] = new AdminPanelsModuleClassifier(naiveBayesMetaData, _validationService);
        string networkDevicesFile = Path.Combine(_learningDataPath, "data_NetworkDevices.txt");
        List<Models.Document> networkDocs = ReadTrainingData(networkDevicesFile);
        var naiveBayesNetwork = new NaiveBayesClassifier(_validationService);
        naiveBayesNetwork.Train(networkDocs);
        _classifiers["NetworkDevicesModule"] = new AdminPanelsModuleClassifier(naiveBayesNetwork,
_validationService);
        string vulnFile = Path.Combine(_learningDataPath, "data_Vulnerabilities.txt");
        List<Models.Document> vulnDocs = ReadTrainingData(vulnFile);
        var naiveBayesVuln = new NaiveBayesClassifier(_validationService);
        naiveBayesVuln.Train(vulnDocs);
        _classifiers["VulnerabilitiesModule"] = new AdminPanelsModuleClassifier(naiveBayesVuln, _validationService);
        string webPageFile = Path.Combine(_learningDataPath, "data_WebPage.txt");
        List<Models.Document> webPageDocs = ReadTrainingData(webPageFile);
        var naiveBayesWebPage = new NaiveBayesClassifier(_validationService);
        naiveBayesWebPage.Train(webPageDocs);
        _classifiers["WebPageModule"] = new AdminPanelsModuleClassifier(naiveBayesWebPage, _validationService);
        string webSerFile = Path.Combine(_learningDataPath, "data_WebServices.txt");
        List<Models.Document> webSerDocs = ReadTrainingData(webSerFile);
        var naiveBayesWebSer = new NaiveBayesClassifier(_validationService);
        naiveBayesVuln.Train(webSerDocs);
        _classifiers["WebServicesModule"] = new AdminPanelsModuleClassifier(naiveBayesWebSer, _validationService);

        _classifiers["PeopleSearchModule"] = new NoClassificationClassifier();
        _classifiers["EmailSearchModule"] = new NoClassificationClassifier();
    }

    public IClassifier GetClassifier(string moduleName)
    {
        if (_classifiers.TryGetValue(moduleName, out var classifier))
        {
            return classifier;
        }
        throw new Exception($"Класифікатор для модуля {moduleName} не знайдений");
    }



    private List<Models.Document> ReadTrainingData(string filePath)
    {
        List<Models.Document> docs = new List<Models.Document>();
        var lines = File.ReadAllLines(filePath);
        foreach (var line in lines)
        {
            if (string.IsNullOrWhiteSpace(line)) continue;
            string trimmed = line.Trim();
            if (trimmed.StartsWith("\"") && trimmed.EndsWith("\""))
                trimmed = trimmed.Substring(1, trimmed.Length - 2);
            var parts = trimmed.Split(new string[] { "\",\"" }, StringSplitOptions.None);
            if (parts.Length < 5) continue;

            Models.Document doc = new Models.Document
            {
                Title = parts[0],
                Url = parts[1],
                Snippet = parts[2],
                Module = parts[3],
                Label = parts[4]
            };
            // Об'єднуємо Title, Url і Snippet
            doc.CombinedText = _validationService.NormalizerTextForClassification(doc.Title + " " + doc.Url + " " +
(doc.Snippet ?? ""));
```

```
            docs.Add(doc);
        }
        return docs;
    }
}

public interface IClassifier
{
    Task<ClassificationResult> ClassifyAsync(string text);
}

public class NaiveBayesClassifier
{
    private Dictionary<string, int> docCountByClass = new Dictionary<string, int>();
    private Dictionary<string, Dictionary<string, int>> wordCountsByClass = new Dictionary<string, Dictionary<string,
int>>();
    private Dictionary<string, int> totalWordsInClass = new Dictionary<string, int>();
    private HashSet<string> vocabulary = new HashSet<string>();
    private int totalDocuments = 0;
    private readonly ValidationService _validationService;

    public NaiveBayesClassifier(ValidationService validationService)
    {
        _validationService = validationService;
    }

    public void Train(List<Models.Document> documents)
    {
        foreach (var doc in documents)
        {
            totalDocuments++;
            string label = doc.Label;
            if (!docCountByClass.ContainsKey(label))
            {
                docCountByClass[label] = 0;
                wordCountsByClass[label] = new Dictionary<string, int>();
                totalWordsInClass[label] = 0;
            }
            docCountByClass[label]++;

            var tokens = Tokenize(doc.CombinedText);
            foreach (var token in tokens)
            {
                vocabulary.Add(token);
                if (!wordCountsByClass[label].ContainsKey(token))
                    wordCountsByClass[label][token] = 0;
                wordCountsByClass[label][token]++;
                totalWordsInClass[label]++;
            }
        }


    }

    private IEnumerable<string> Tokenize(string text)
    {
        text = _validationService.NormalizerTextForClassification(text);
        var words = text.Split(' ')
            .Where(s => !string.IsNullOrWhiteSpace(s)).ToList();
        List<string> tokens = new List<string>(words);
        for (int i = 0; i < words.Count - 1; i++)
        {
            string bigram = words[i] + "_" + words[i + 1];
```

```
            tokens.Add(bigram);
        }
        return tokens;
    }

    private double CalculateLogProbability(string text, string label)
    {
        double logProb = Math.Log((double)docCountByClass[label] / totalDocuments);
        var tokens = Tokenize(text);
        int vocabSize = vocabulary.Count;
        int totalTokens = totalWordsInClass[label];

        foreach (var token in tokens)
        {
            int count = wordCountsByClass[label].ContainsKey(token) ? wordCountsByClass[label][token] : 0;

            double tokenProb = (double)(count + 1) / (totalTokens + vocabSize);
            logProb += Math.Log(tokenProb);
        }
        return logProb;
    }

    public Dictionary<string, double> GetClassLogProbabilities(string text)
    {
        var result = new Dictionary<string, double>();
        foreach (var label in docCountByClass.Keys)
        {
            result[label] = CalculateLogProbability(text, label);
        }
        return result;
    }

    public string Predict(string text)
    {
        string bestLabel = null;
        double bestLogProb = double.NegativeInfinity;
        foreach (var label in docCountByClass.Keys)
        {
            double logProb = CalculateLogProbability(text, label);
            if (logProb > bestLogProb)
            {
                bestLogProb = logProb;
                bestLabel = label;
            }

        }
        return bestLabel;
    }
}

public class NoClassificationClassifier : IClassifier
{
    public Task<ClassificationResult> ClassifyAsync(string text)
    {
        return Task.FromResult(new ClassificationResult
        {
            PredictedLabel = "Не оцінюється",
            LogProbabilities = new Dictionary<string, double>()
        });
    }
}

namespace AbyssDorks.Controllers
```

```
{
   public class DorkController : Controller
   {
      private readonly ClassifierManager _classifierManager;
      private readonly ILogger<DorkController> _logger;
      private readonly JsonDataService _jsonDataService;
      private readonly TagsService _tagsService;
      private readonly DorkGeneratorService _dorkGeneratorService;
      private readonly SearchService _searchService;
      private readonly ValidationService _validationService;

      public DorkController(ClassifierManager classifierManager, ILogger<DorkController> logger, JsonDataService
jsonDataService, TagsService tagsService, DorkGeneratorService dorkGeneratorService, SearchService searchService,
ValidationService validationService)
      {
         _classifierManager = classifierManager;
         _logger = logger;
         _jsonDataService = jsonDataService;
         _tagsService = tagsService;
         _dorkGeneratorService = dorkGeneratorService;
         _searchService = searchService;
         _validationService = validationService;
      }

      public IActionResult Index()
      {

         var modules = new List<string>
         {
            "AdminPanelsModule",
            "EmailSearchModule",
            "FilesDocumentModule",
            "MetadataModule",
            "NetworkDevicesModule",
            "PeopleSearchModule",
            "VulnerabilitiesModule",
            "WebPageModule",
            "WebServicesModule"
         };

         return View(modules);
      }

      public IActionResult SetCredentials()
      {
         return View();
      }

      public IActionResult Guide()
      {
         return View();
      }

      [HttpPost]
      public IActionResult SaveCredentials(string apiKey, string cx)
      {
         if (string.IsNullOrEmpty(apiKey) || string.IsNullOrEmpty(cx))
         {
            return BadRequest("API-ключ або CX не може бути пустим.");
         }

         HttpContext.Session.SetString("GoogleApiKey", apiKey);
         HttpContext.Session.SetString("GoogleCx", cx);
```

```
            Console.WriteLine($"API: {apiKey}");
            Console.WriteLine($"CX: {cx}");

            return Ok("Дані успішно збережено");
        }

        public IActionResult GetTags(string moduleName)
        {
            var tags = _tagsService.GetTagsForModule(moduleName);
            return Json(tags);
        }

        public IActionResult GetTemplates(string moduleName, List<string> tags)
        {
            var templates = _dorkGeneratorService.GetTemplateByTags(moduleName, tags);
            return Json(templates);
        }

        [HttpPost]
        public IActionResult GenerateQuery(string moduleName, string templateName, Dictionary<string, string>
parameters)
        {
            var module = _jsonDataService.LoadModule(moduleName);
            var template = module.Templates.FirstOrDefault(t => t.Name == templateName);

            if (template == null)
            {
                return BadRequest("шаблон не знайдено");
            }

            if (!_validationService.ValidateParameters(template, parameters))
            {
                return BadRequest("Некоректні параметри шаблону");
            }

            var query = _dorkGeneratorService.GenerateSearchQuery(template, parameters);

            if (!_validationService.ValidateSyntax(query))
            {
                return BadRequest("Некоректний синтаксис запиту");
            }

            return Json(new { Query =  query });
        }

        [HttpPost]
        public async Task<IActionResult> ExecuteSearch([FromBody] List<QueryModulePair> queryModulePairs)
        {
            try
            {
                var allResults = new List<SearchResult>();
                foreach (var pair in queryModulePairs)
                {
                    var decodeQuery = WebUtility.UrlDecode(pair.Query);
                    Console.WriteLine($"(контролер) запит: {pair.Query} для модуля {pair.Module}");
                    var results = await _searchService.ExecuteSearch(HttpContext, decodeQuery);

                    if (results == null || !results.Any())
                    {
                        _logger.LogWarning($"Для запиту {pair.Query} повернулося 0 результатів, пропускаємо...");
                        continue;
                    }
```

```
                var classifier = _classifierManager.GetClassifier(pair.Module);
                foreach (var result in results)
                {
                    string combinedText = _validationService.NormalizerTextForClassification(result.Title + " " + result.Url +
" " + result.Snippet);
                    var classification = await classifier.ClassifyAsync(combinedText);
                    result.PredictedLabel = classification.PredictedLabel;
                    result.Module = pair.Module;
                }
                allResults.AddRange(results);
            }

            if (allResults.Any())
            {
                var sortedResults = allResults.OrderBy(r => r.PredictedLabel == "Critically" ? 0 : 1).ToList();
                return Json(new { Items = sortedResults });
            }
            else
            {
                return Json(new { Items = new List<SearchResult>() });
            }

        }
        catch (Exception ex)
        {
            return BadRequest(new { Error = ex.Message });
        }
    }

    [HttpGet]
    public IActionResult Results(string query)
    {
        ViewBag.Query = query;
        return View();
    }

    public IActionResult Privacy()
    {
        return View();
    }
}

public class Document
{
    public string Title {  get; set; }
    public string Url {  get; set; }
    public string Snippet { get; set; }
    public string Module {  get; set; }
    public string Label { get; set; }
    public string CombinedText {  get; set; }
}

public class DorkModules
{
    public Module AdminPanelsModule { get; set; }
    public Module EmailSearchModule {  get; set; }
    public Module FilesDocumentModule { get; set; }
    public Module MetadataModule {  get; set; }
    public Module NetworkDevicesModule {  get; set; }
    public Module PeopleSearchModule { get; set; }
    public Module VulnerabilitiesModule {  get; set; }
    public Module WebPageModule {  get; set; }
```

```
    public Module WebServicesModule { get; set; }
}

public class Module
 {
    [JsonPropertyName("templates")]
    public List<Template> Templates { get; set; }
 }

public class Operator
 {
    [JsonPropertyName("name")]
    public string Name { get; set; }
    [JsonPropertyName("description")]
    public string Description { get; set; }
    [JsonPropertyName("example")]
    public string Example {  get; set; }
    [JsonPropertyName("parameters")]
    public bool Parameters { get; set; }
    [JsonPropertyName("categories")]
    public List<string> Categories { get; set; }

 }

public class QueryModulePair
{
   public string Query {  get; set; }
   public string Module { get; set; }
}

public class SearchResult
{
   public string Title { get; set; }
   public string Url { get; set; }
   public string Snippet { get; set; }
   public string Module { get; set; }
   public string PredictedLabel {  get; set; }
}

public class Template
{
   [JsonPropertyName("name")]
   public string Name { get; set; }
   [JsonPropertyName("description")]
   public string Description { get; set; }
   [JsonPropertyName("template")]
   public string TemplateText { get; set; }
   [JsonPropertyName("hint")]
   public string Hint { get; set; }
   [JsonPropertyName("tags")]
   public List<string> Tags { get; set; }
}

public class DorkGeneratorService
 {
    private readonly JsonDataService _jsonDataService;
    private readonly ValidationService _validationService;

    public DorkGeneratorService(JsonDataService jsonDataService, ValidationService validationService)
    {
       _jsonDataService = jsonDataService;
       _validationService = validationService;
    }
```

```
    public List<Template> GetTemplateByTags(string moduleName, List<string> tags)
    {
        var module = _jsonDataService.LoadModule(moduleName);

        var normalizedTags = tags.Select(t => t.Trim().ToLower()).ToList();

        return module.Templates
            .Where(t => t.Tags.Select(tag => tag.Trim().ToLower()).Intersect(normalizedTags).Any()).ToList();
    }

    public string GenerateSearchQuery(Template template, Dictionary<string, string> parameters)
    {
        if (!_validationService.ValidateParameters(template, parameters))
            throw new Exception("Невідповідність параметрів у шаблоні");

        var query = template.TemplateText;
        foreach (var param in parameters)
        {
            query = query.Replace($"{{{param.Key}}}", param.Value);
        }

        if (!_validationService.ValidateSyntax(query))
            throw new Exception("Синтаксис запиту некоректний");

        return query;
    }
}

public class JsonDataService
{
    private readonly string _basePath = Path.Combine(AppContext.BaseDirectory, "JsonData");

    public Module LoadModule(string moduleName)
    {
        try
        {
            Console.WriteLine($"Шлях до файлу: {_basePath}\\{moduleName}.json");

            var filePath = Path.Combine(_basePath, $"{moduleName}.json");

            if (!File.Exists(filePath))
            {
                throw new FileNotFoundException($"JSON файл для модуля {moduleName} не існує за шляхом
{filePath}");
            }

            var json = File.ReadAllText(filePath);
            return JsonSerializer.Deserialize<Module>(json) ?? throw new Exception($"Не вдалося десеріалізувати модуль
{moduleName}.");
        }
        catch (Exception ex)
        {
            // Детальний лог помилки
            Console.WriteLine($"Помилка при завантаженні модуля {moduleName}: {ex.Message}");
            throw;
        }
    }

    public List<Operator> LoadOperators()
    {
        var filePath = Path.Combine(_basePath, "operators.json");
        if (!File.Exists(filePath))
```

```
            throw new FileNotFoundException("JSON файл з операторами не знайдений");

        var json = File.ReadAllText(filePath);
        return JsonSerializer.Deserialize<List<Operator>>(json);
    }
}

public class SearchService
{
    private readonly HttpClient _httpClient;

    public SearchService(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    public async Task<List<SearchResult>> ExecuteSearch(HttpContext httpContext, string query)
    {
        var apiKey = httpContext.Session.GetString("GoogleApiKey");
        var cx = httpContext.Session.GetString("GoogleCx");

        Console.WriteLine($"API-ключ із сесії: {apiKey}");
        Console.WriteLine($"CX із сесії: {cx}");

        if (string.IsNullOrEmpty(apiKey) || string.IsNullOrEmpty(cx))
            throw new Exception("API-ключ або CX не задано. Введіть їх у відповідне поле.");

        var allResults = new List<SearchResult>();
        int startIndex = 1;
        int maxIndex = 20;
        Console.WriteLine($"(сервіс) надійшов запит: {query}");

        while (startIndex <= maxIndex)
        {
            var decodeHtmlQuery = WebUtility.HtmlDecode(query);
            Console.WriteLine($"запит перед додаванням до урл: {decodeHtmlQuery}");
            var url =
$"https://www.googleapis.com/customsearch/v1?key={apiKey}&cx={cx}&q={Uri.EscapeDataString(decodeHtmlQuery
)}&start={startIndex}&safe=off&filter=0";
            Console.WriteLine($"URL-запиту: {url}");

            try
            {
                var response = await _httpClient.GetAsync(url);
                Console.WriteLine($"Статус відповіді: {response.StatusCode}");

                if (!response.IsSuccessStatusCode)
                    throw new Exception($"Помилка виконання запиту. Код: {response}");

                var googleResponse = await response.Content.ReadFromJsonAsync<GoogleSearchResponse>();
                Console.WriteLine($"Отримані дані: {JsonSerializer.Serialize(googleResponse)}");

                if (googleResponse == null || googleResponse.Items == null)
                    break;

                allResults.AddRange(googleResponse.Items.Select(item => new SearchResult
                {
                    Title = item.Title,
                    Url = item.Link,
                    Snippet = item.Snippet,
                    //Relevance = null //пізніше додати коли буде створюватись ШІ
                }));
```

```
            if (googleResponse.Items.Count < 10)
                break;

            startIndex += 10;
        }
        catch (HttpRequestException ex)
        {
            throw new Exception("Помилка HTTP-запиту: " + ex.Message);
        }
        catch (Exception ex)
        {
            throw new Exception("Помилка при обробці пошуку: " + ex.Message);
        }
    }

    return allResults;
}

public class GoogleSearchResponse
{
    public List<GoogleSearchItem> Items { get; set; }
}

public class GoogleSearchItem
{
    public string Title { get; set; }
    public string Link { get; set; }
    public string Snippet { get; set; }
}
}

public class TagsService
{
    private readonly JsonDataService _jsonDataService;

    public TagsService(JsonDataService jsonDataService)
    {
        _jsonDataService = jsonDataService;
    }

    public List<string> GetTagsForModule(string moduleName)
    {
        var module = _jsonDataService.LoadModule(moduleName);
        return module.Templates.SelectMany(t => t.Tags).Distinct().ToList();
    }
}

public class ValidationService
{
    private readonly JsonDataService _jsonDataService;

    public ValidationService(JsonDataService jsonDataService)
    {
        _jsonDataService = jsonDataService;
    }

    public bool ValidateOperators(string searchQuery)
    {
        var operators = _jsonDataService.LoadOperators();
        var operatorNames = operators.Select(op => op.Name).ToList();

        var regex = new Regex(@"(\w+:)");
        var matches = regex.Matches(searchQuery);
```

```
        foreach ( Match match in matches )
        {
            var operatorName = match.Value.TrimEnd(':');
            if (!operatorNames.Contains(operatorName) )
            {
                return false;
            }
        }
        return true;
    }

    public bool ValidateSyntax(string searchQuery)
    {
        if (searchQuery.Contains("\"\"") || searchQuery.Contains("::"))
            return false;

        return true;
    }

    public bool ValidateParameters(Template template, Dictionary<string, string> parameters)
    {
        foreach (var param in parameters.Keys)
        {
            if (!template.TemplateText.Contains($"{{{param}}}"))
            {
                return false;
            }
        }
        return true;
    }

    public string NormalizerTextForClassification(string input)
    {
        input = input.ToLower();
        input = Regex.Replace(input, @"[^\w\s]", " ");
        input = Regex.Replace(input, @"\s+", " ");
        return input.Trim();
    }
}

@{
    ViewData["Title"] = "Guide";
}

<div class="container mt-4">
    <h2>Налаштування Google Custom Search для нашої системи</h2>
    <p>
        Перш ніж почати роботу з нашим додатком, потрібно виконати кілька кроків у сервісах Google.
        Це дозволить вам отримати 2 обов’язкові параметри:
    </p>
    <ul>
        <li><strong>Search Engine ID (cx)</strong></li>
        <li><strong>API Key</strong></li>
    </ul>

    <hr />

    <h3>1. Створення власної Search Engine (щоб отримати cx)</h3>
    <ol>
        <li>
            Перейдіть за посиланням:
            <a href="https://programmablesearchengine.google.com/controlpanel/create" target="_blank">
```

```
            https://programmablesearchengine.google.com/controlpanel/create
          </a>.
        </li>
        <li>
          У полі <em>«Назва пошукової системи»</em> (або «Name of the search engine»)
          введіть довільну назву (наприклад, <em>"My Custom Engine"</em>).
        </li>
        <li>
          Виберіть, на яких сайтах система має виконувати пошук:
          можна вказати конкретні домени або поставити <em>«Пошук у всьому інтернеті»</em>.
          Якщо не впевнені — оберіть <strong>пошук по всьому інтернету</strong>.
        </li>
        <li>Зніміть або залиште потрібні налаштування для «Пошуку зображень» і «Безпечного пошуку».</li>
        <li>Підтвердьте, що ви не робот (reCAPTCHA), потім натисніть <strong>«Створити»</strong>.</li>
        <li>
          У результаті з’явиться сторінка з вашим пошуковим рушієм.
          Знайдіть там <strong>ID пошукової системи (cx)</strong>, зазвичай під заголовком <em>«Search engine ID»</em>.
        </li>
      </ol>
      <p><strong>Збережіть свій «cx»</strong> — його знадобиться вводити в нашому додатку.</p>

      <hr />

      <h3>2. Створення Google API Key</h3>
      <ol>
        <li>
          Перейдіть за посиланням:
          <a href="https://console.cloud.google.com/apis/credentials/key" target="_blank">
            https://console.cloud.google.com/apis/credentials/key
          </a>.
          Якщо посилання не працює, зайдіть у <strong>Google Cloud Console</strong> &amp;raquo;
          <em>APIs &amp;amp; Services</em> &amp;raquo; <em>Credentials</em>.
        </li>
        <li>Виберіть <strong>«Create credentials» &amp;raquo; «API key»</strong>.</li>
        <li>
          З’явиться вікно з вашим новим API-ключем. Ви можете налаштувати обмеження доступу
          (наприклад, прив’язка за IP), якщо хочете підвищити безпеку.
        </li>
        <li><strong>Скопіюйте цей ключ</strong> (довгий ряд символів) і збережіть його у безпечному місці.</li>
      </ol>
      <p>
        <strong>API Key</strong> (наприклад <em>AIzaSyD0S---</em>) використовується
        разом із «cx» для виконання запитів у нашому продукті.
      </p>

      <hr />

      <h3>3. Вставлення ключів у наш додаток</h3>
      <p>
        У головному вікні програми ви побачите форму для введення <strong>Google API Key</strong>
        і <strong>Search Engine ID</strong>. Введіть туди значення, які ви отримали вище,
        і натисніть <em>«Зберегти»</em>.
      </p>

      <h4>Приклад введення:</h4>
      <ul>
        <li>API Key: <code>AIzaSyD0S...</code></li>
        <li>Search Engine ID (cx): <code>0123456789abcdef:xyz123</code></li>
      </ul>

      <hr />
```

```
    <h3>4. Поради та важливі моменти</h3>
    <ul>
        <li>
            Якщо ви бажаєте обмежити доступ до свого API-ключа,
            рекомендується використовувати <em>Application restrictions</em> (Websites, IP тощо).
        </li>
        <li>
            Стежте за <strong>лімітом запитів</strong>, оскільки безкоштовна квота не безмежна.
            Зазвичай ~100 запитів на добу.
        </li>
        <li>
            Для перевірки, що все працює, спробуйте зробити найпростіший запит у полі пошуку (наприклад,
<em>"test"</em>).
            Якщо бачите результати — налаштування виконані успішно!
        </li>
    </ul>

    <hr />

    <h4>Все готово!</h4>
    <p>
        Тепер, маючи Google API Key та Search Engine ID, ви можете повноцінно користуватися нашим
        застосунком для автоматизованої генерації Google Dork-запитів.
    </p>
</div>

@{
    ViewData["Title"] = "Dork Generator";
    var moduleDictionary = new Dictionary<string, string>
    {
        { "AdminPanelsModule", "Адмін-панелі" },
        { "EmailSearchModule", "Пошук електронних листів" },
        { "FilesDocumentModule", "Документи та файли" },
        { "MetadataModule", "Метадані" },
        { "NetworkDevicesModule", "Мережеві пристрої" },
        { "PeopleSearchModule", "Пошук людей" },
        { "VulnerabilitiesModule", "Вразливості" },
        { "WebPageModule", "Веб-сторінки" },
        { "WebServicesModule", "Веб-сервіси" }
    };
}
<div id="query-section" style="text-align: center; margin-top: 20px;">
    <h3>Введення запиту:</h3>
    <textarea id="queryText" style="width: 80%; height: 40px; border-radius: 20px; margin-top: 20px; font-size:
22px;"></textarea>
    <div id="savedQueriesThumbnail"
         style="position: fixed; bottom: 20px; right: 20px; width: 50px; height: 50px; background: #007bff; color: white;
border-radius: 50%; text-align: center; line-height: 50px; cursor: pointer;">
        0
    </div>

    <div id="savedQueriesPopup"
         style="display: none; position: fixed; bottom: 80px; right: 20px; background: white; border-radius: 10px; border:
1px solid #ccc; padding: 10px; max-width: 300px;">
        <h4 style="margin-top:0;">Збережені запити</h4>
        <ul id="savedQueriesList" style="list-style: none; padding: 0;"></ul>
    </div>

    <br>
    <button id="saveQueryBtn" type="button" style="padding: 10px 20px; border-radius: 20px; font-size: 18px;
background-color: #28a745; color: white; margin-top:10px;">
        Зберегти запит
    </button>
```

```
    <br><br>
    <div id="module-selection" style="margin-top: 20px;">
        <h3>Оберіть модулі:</h3>
        @foreach (var module in moduleDictionary)
        {
            <button type="button" class="module-button" data-module="@module.Key" style="margin: 5px; padding: 10px
20px; border-radius: 20px; font-size: 18px;">
                @module.Value
            </button>
        }
    </div>
    <br>
    <button id="executeSearch" type="button" style="padding: 15px 30px; border-radius: 20px; font-size: 20px;">
        Почати пошук
    </button>
</div>

<div id="operator-section" style="margin-top: 20px; text-align: center;">
    <h3>Модифікація запиту:</h3>
    <select id="operatorSelect" style="margin-right: 10px;">
        <option>Уточніть ваш запит</option>
        <option value="before">До дати</option>
        <option value="after">Після дати</option>
        <option value="site">Сайт/домен</option>
        <option value="-">Виключення</option>
        <option value="loc">Локація</option>
    </select>
    <input id="operatorParameter" type="text" placeholder="Параметр" style="margin-right: 10px; display: none;
padding: 5px; border-radius: 5px;" />
    <input id="datePicker" type="date" style="margin-right: 10px; display: none; padding: 5px; border-radius: 5px;" />
    <select id="domainSelect" style="margin-right: 10px; display: none; padding: 5px; border-radius: 5px;">
        <option value="">Виберіть домен</option>
        <option value=".com">.com</option>
        <option value=".org">.org</option>
        <option value=".ru">.ru</option>
        <option value=".ua">.ua</option>
        <option value=".de">.de</option>
        <option value=".pl">.pl</option>
        <option value=".cz">.cz</option>
        <option value=".biz">.biz</option>
    </select>
    <select id="locationSelect" style="margin-right: 10px; display: none; padding: 5px; border-radius: 5px;">
        <option value="">Виберіть локацію</option>
        <option value="London">Лондон</option>
        <option value="Paris">Париж</option>
        <option value="Kyiv">Київ</option>
        <option value="New York">Нью-Йорк</option>
    </select>
    <button id="addOperatorBtn" style="padding: 5px 15px; border-radius: 5px;">Додати</button>
</div>

<div id="error-message" style="display: none; color: red; text-align: center; margin-top: 20px;"></div>

<hr style="width: 80%; margin: 20px auto" />

<div id="tags-section" style="display: none; text-align: center; margin-top: 20px;">
    <h3>Теги:</h3>
    <div id="tagsContainer" style="display: flex; flex-wrap: wrap; justify-content: center; gap: 10px;"></div>
</div>

<div id="templates-section" style="display: none; text-align: center; margin-top: 20px;">
    <h3>Шаблони:</h3>
    <table id="templateTable" style="width: 80%; margin: 0 auto; border-collapse: collapse;">
```

```
        <thead>
            <tr>
                <th style="border: 1px solid #ddd; padding: 8px;">Назва</th>
                <th style="border: 1px solid #ddd; padding: 8px;">Опис</th>
                <th style="border: 1px solid #ddd; padding: 8px;">Підказка</th>
                <th style="border: 1px solid #ddd; padding: 8px;">Шаблон</th>
                <th style="border: 1px solid #ddd; padding: 8px;">Дія</th>
            </tr>
        </thead>
        <tbody></tbody>
    </table>
</div>

<div id="results-section" style="display: none; text-align: center; margin-top: 20px;">
    <h3>Результати:</h3>
    <table id="resultsTable" style="width: 80%; margin: 0 auto; border-collapse: collapse;">
        <thead>
            <tr>
                <th style="border: 1px solid #ddd; padding: 8px;">Назва</th>
                <th style="border: 1px solid #ddd; padding: 8px;">Посилання</th>
                <th style="border: 1px solid #ddd; padding: 8px;">Опис</th>
                <th style="border: 1px solid #ddd; padding: 8px;">Модуль</th>
                <th style="border: 1px solid #ddd; padding: 8px;">Оцінка</th>
            </tr>
        </thead>
        <tbody id="resultsTableBody"></tbody>
    </table>
</div>

@section Scripts {
    <style>
        .module-button.active {
            border: 2px solid #ffc107;
            background-color: #ffc107 !important;
            color: black;
        }

        body {
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
            flex-direction: column;
            margin: 0;
            padding: 20px;
            font-family: Arial, sans-serif;
        }

        #moduleButtonContainer {
            display: flex;
            flex-wrap: wrap;
            justify-content: center;
            gap: 10px;
        }

        .module-button, .tag-button {
            background-color: #007bff;
            color: white;
            border: none;
            border-radius: 250px;
            padding: 10px 20px;
            font-size: 16px;
            cursor: pointer;
```

```
        transition: background-color 0.4s;
    }

        .module-button:hover, .tag-button:hover {
            background-color: #0056b3;
        }

        .module-button[data-module="AdminPanelsModule"] {
            background-color: #f39c12;
        }

        .module-button[data-module="EmailSearchModule"] {
            background-color: #e74c3c;
        }

        .module-button[data-module="FilesDocumentModule"] {
            background-color: #27ae60;
        }

        .module-button[data-module="MetadataModule"] {
            background-color: #8e44ad;
        }

        .module-button[data-module="NetworkDevicesModule"] {
            background-color: #3498db;
        }

        .module-button[data-module="PeopleSearchModule"] {
            background-color: #1abc9c;
        }

        .module-button[data-module="VulnerabilitiesModule"] {
            background-color: #e67e22;
        }

        .module-button[data-module="WebPageModule"] {
            background-color: #2ecc71;
        }

        .module-button[data-module="WebServicesModule"] {
            background-color: #9b59b6;
        }

        .module-button:active {
            background-color: #34495e;
        }
</style>
<script>
    $(document).ready(function () {
        let savedQueries = JSON.parse(sessionStorage.getItem("savedQueries") || "[]");

        function updateSavedQueriesDisplay() {
            let savedQueries = JSON.parse(sessionStorage.getItem("savedQueries") || "[]");
            $("#savedQueriesThumbnail").text(savedQueries.length);
            let list = $("#savedQueriesList");
            list.empty();
            if (savedQueries.length === 0) {
                list.append("<li>Немає збережених запитів</li>");
            } else {
                savedQueries.forEach(function(q, index) {
                    list.append(`<li style="margin-bottom:5px;">
                        <strong>${q.module}</strong>: ${q.query}
```

```
                    <button class="delete-query" data-index="${index}" style="margin-left:10px; border-radius:60px;">X</button>
                </li>`);
            });
        }
    }

    $(document).on("click", ".delete-query", function () {
        let index = $(this).data("index");
        let savedQueries = JSON.parse(sessionStorage.getItem("savedQueries") || "[]");
        savedQueries.splice(index, 1);
        sessionStorage.setItem("savedQueries", JSON.stringify(savedQueries));
        updateSavedQueriesDisplay();
    });

    updateSavedQueriesDisplay();

    $("#savedQueriesThumbnail").hover(
        function () {
            $("#savedQueriesPopup").fadeIn(200);
        },
        function () {
            $("#savedQueriesPopup").fadeOut(1000);
        }
    );

    $("#saveQueryBtn").on("click", function () {
        let currentQuery = $("#queryText").val().trim();
        let activeModule = $(".module-button.active").data("module");
        if (!currentQuery || !activeModule) {
            alert("Введіть запит та оберіть модуль!");
            return;
        }
        savedQueries.push({ query: currentQuery, module: activeModule });
        sessionStorage.setItem("savedQueries", JSON.stringify(savedQueries));
        alert("Запит збережено для модуля " + activeModule + "!");
        $("#queryText").val("");
        $(".module-button").removeClass("active");
        updateSavedQueriesDisplay();
    });
    $("#operatorSelect").on("change", function () {
        const selectedOperator = $(this).val();
        $("#operatorParameter, #datePicker, #domainSelect, #locationSelect").hide();
        if (selectedOperator === "before" || selectedOperator === "after") {
            $("#datePicker").show();
        } else if (selectedOperator === "site") {
            $("#domainSelect").show();
        } else if (selectedOperator === "loc") {
            $("#locationSelect").show();
        } else if (selectedOperator === "-") {
            $("#operatorParameter").show().attr("placeholder", "Введіть слово для виключення");
        }
    });

    $("#addOperatorBtn").on("click", function () {
        const baseQuery = $("#queryText").val();
        const operatorName = $("#operatorSelect").val();
        if (!operatorName) {
            alert("Будь ласка, виберіть оператор.");
            return;
        }
        let parameter = "";
        if (operatorName === "before" || operatorName === "after") {
```

```
            parameter = $("#datePicker").val();
            if (!parameter) { alert("Будь ласка, виберіть дату."); return; }
        } else if (operatorName === "site") {
            parameter = $("#domainSelect").val();
            if (!parameter) { alert("Будь ласка, виберіть домен."); return; }
        } else if (operatorName === "loc") {
            parameter = $("#locationSelect").val();
            if (!parameter) { alert("Будь ласка, виберіть локацію."); return; }
        } else if (operatorName === "-") {
            parameter = $("#operatorParameter").val();
            if (!parameter) { alert("Будь ласка, введіть слова для виключення."); return; }
        }
        const modifiedQuery = `${baseQuery} ${operatorName}:${parameter}`;
        $("#queryText").val(modifiedQuery);
        alert("Оператор успішно додано до запиту!");
    });

    $(document).on("click", ".module-button", function () {
        const moduleName = $(this).data("module");
        if (!moduleName) return;
        sessionStorage.setItem("selectedModule", moduleName);
        $.get("/Dork/GetTags", { moduleName })
            .done(function (tags) {
                $("#tagsContainer").empty();
                tags.forEach(function (tag) {
                    $("#tagsContainer").append(`<button style="margin-top: 12px;" class="tag-button" data-tag="${tag}">${tag}</button>`);
                });
                $("#tags-section").show();
                $(".module-button").removeClass("active");
                $(this).addClass("active");
            }.bind(this));
    });

    $(document).on("click", ".tag-button", function () {
        $(".tag-button").removeClass("active");
        $(this).addClass("active");
        const moduleName = $(".module-button.active").data("module");
        const selectedTag = $(this).data("tag");
        if (!selectedTag) {
            $("#templates-section").hide();
            return;
        }
        $.get("/Dork/GetTemplates", { moduleName, tags: selectedTag })
            .done(function (templates) {
                const tableBody = $("#templateTable tbody");
                tableBody.empty();
                templates.forEach(function (template) {
                    const safeJson = JSON.stringify(template.template)
                        .replace(/'/g, '&#39;')
                        .replace(/"/g, '&quot;');
                    const row = `
                        <tr>
                            <td style="border: 1px solid #ddd; padding: 8px;">${template.name}</td>
                            <td style="border: 1px solid #ddd; padding: 8px;">${template.description}</td>
                            <td style="border: 1px solid #ddd; padding: 8px;">${template.hint}</td>
                            <td style="border: 1px solid #ddd; padding: 8px;">${template.template}</td>
                            <td style="border: 1px solid #ddd; padding: 8px;">
                                <button class="select-template" data-template='${safeJson}'>Вибрати</button>
                            </td>
                        </tr>
                    `;
                    tableBody.append(row);
```

```
                });
                $("#templates-section").show();
            });
        });

        $(document).on("click", ".select-template", function () {
            try {
                const templateData = JSON.parse($(this).data("template").replace(/&quot;/g, ""));
                console.log("Вибраний шаблон: ", templateData);
                $("#queryText").val(templateData);
                $("#query-section").show();
            } catch (error) {
                alert("Неможливо вибрати шаблон. Перевірте структуру даних.");
            }
        });

        $("#saveQueryBtn").on("click", function () {
            let currentQuery = $("#queryText").val().trim();
            let activeModule = $(".module-button.active").data("module");
            if (!currentQuery || !activeModule) {
                alert("Введіть запит та оберіть модуль!");
                return;
            }
            savedQueries.push({ query: currentQuery, module: activeModule });
            sessionStorage.setItem("savedQueries", JSON.stringify(savedQueries));
            alert("Запит збережено для модуля " + activeModule + "!");
            $("#queryText").val("");
            $(".module-button").removeClass("active");
        });

        $("#executeSearch").on("click", function () {
            let savedQueries = JSON.parse(sessionStorage.getItem("savedQueries") || "[]");
            if (savedQueries.length === 0) {
                alert("Будь ласка, збережіть хоча б один запит!");
                return;
            }
            $.ajax({
                url: "/Dork/ExecuteSearch",
                method: "POST",
                contentType: "application/json",
                data: JSON.stringify(savedQueries),
                success: function(results) {
                    const encodedQuery = encodeURIComponent("");
                    window.location.href = `/Dork/Results?query=${encodedQuery}`;
                },
                error: function (err) {
                    $("#error-message").text("Нічого не знайдено.");
                    $("#error-message").show();
                }
            });
        });
    });
  </script>
}

@{
  ViewData["Title"] = "Результати пошуку";
}

<h1 style="text-align: center;">Результати пошуку</h1>

<table style="width: 80%; margin: 0 auto; border-collapse: collapse;">
  <thead>
```

```
        <tr>
            <th style="border: 1px solid #ddd; padding: 8px;">Назва</th>
            <th style="border: 1px solid #ddd; padding: 8px;">Посилання</th>
            <th style="border: 1px solid #ddd; padding: 8px;">Опис</th>
            <th style="border: 1px solid #ddd; padding: 8px;">Модуль</th>
            <th style="border: 1px solid #ddd; padding: 8px;">Оцінка</th>
        </tr>
    </thead>
    <tbody id="resultsTableBody"></tbody>
</table>

@section Scripts {
    <script>
        $(document).ready(function () {
            const moduleMapping = {
                "AdminPanelsModule": "Адмін-панелі",
                "EmailSearchModule": "Пошук електронних листів",
                "FilesDocumentModule": "Документи та файли",
                "MetadataModule": "Метадані",
                "NetworkDevicesModule": "Мережеві пристрої",
                "PeopleSearchModule": "Пошук людей",
                "VulnerabilitiesModule": "Вразливості",
                "WebPageModule": "Веб-сторінки",
                "WebServicesModule": "Веб-сервіси"
            };

            $.ajax({
                url: "/Dork/ExecuteSearch",
                method: "POST",
                contentType: "application/json",
                data: JSON.stringify(JSON.parse(sessionStorage.getItem("savedQueries") || "[]")),
                success: function (data) {
                    const tableBody = $("#resultsTableBody");
                    tableBody.empty();
                    let results = data.Items || data.items;
                    if (results && Array.isArray(results)) {
                        results.forEach(function (result) {
                            const predictedLabel = result.PredictedLabel || result.predictedLabel || "";
                            let labelStyle = "";
                            if (predictedLabel === "Critically") {
                                labelStyle = "color: red; font-weight: bold; font-family: 'Arial Black', sans-serif;";
                            } else if (predictedLabel === "Information") {
                                labelStyle = "color: blue; font-weight: bold; font-family: 'Arial Black', sans-serif;";
                            } else if (predictedLabel === "Не оцінюється") {
                                labelStyle = "color: grey; font-style: italic; font-family: 'Courier New', monospace;";
                            }
                            const moduleValue = result.Module || result.module || "Невідомий";
                            const moduleName = moduleMapping[moduleValue] || moduleValue;
                            const row = `
                                <tr>
                                    <td style="border: 1px solid #ddd; padding: 8px;">${result.Title || result.title || "Без назви"}</td>
                                    <td style="border: 1px solid #ddd; padding: 8px;">
                                        <a href="${result.Link || result.url}" target="_blank">${result.Link || result.url}</a>
                                    </td>
                                    <td style="border: 1px solid #ddd; padding: 8px;">${result.Snippet || result.snippet || "Опис
відсутній"}</td>
                                    <td style="border: 1px solid #ddd; padding: 8px;">${moduleName}</td>
                                    <td style="border: 1px solid #ddd; padding: 8px; ${labelStyle}">${predictedLabel}</td>
                                </tr>
                            `;
                            tableBody.append(row);
                        });
                    } else {
```

```
                tableBody.html("<tr><td colspan='5'>Результати не знайдено або сталася помилка.</td></tr>");
            }
        },
        error: function (err) {
            console.error("Помилка отримання результатів:", err);
            $("#resultsTableBody").html("<tr><td colspan='5'>Результати не знайдено або сталася
помилка.</td></tr>");
            alert("Помилка отримання результатів");
        }
    });
});
</script>
}

@{
    ViewData["Title"] = "Налаштування API";
}

<h1 style="text-align: center; font-size: 2.5rem; margin-bottom: 20px; color: #0056b3;">Налаштування API</h1>

<div id="credentials-section" style="max-width: 500px; margin: 0 auto; padding: 20px; border: 1px solid #ddd; border-
radius: 10px; background-color: #f8f9fa; box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);">
    <form style="display: flex; flex-direction: column; gap: 15px;">
        <div style="text-align: left;">
            <label for="apiKey" style="font-size: 1.2rem; font-weight: bold;">Google API Key:</label>
            <input type="text" id="apiKey" placeholder="Введіть ваш Google API ключ"
                style="width: 100%; padding: 10px; font-size: 1rem; border: 1px solid #ccc; border-radius: 5px;">
        </div>

        <div style="text-align: left;">
            <label for="cxKey" style="font-size: 1.2rem; font-weight: bold;">Search Engine ID (CX):</label>
            <input type="text" id="cxKey" placeholder="Введіть ID пошукового двигуна"
                style="width: 100%; padding: 10px; font-size: 1rem; border: 1px solid #ccc; border-radius: 5px;">
        </div>

        <button id="saveCredentials" type="button"
            style="padding: 15px; font-size: 1.2rem; background-color: #007bff; color: white; border: none; border-radius:
5px; cursor: pointer; transition: background-color 0.3s;">
            Зберегти
        </button>
        <div>
            <a asp-controller="Dork" asp-action="Guide">У мене немає цих даних.</a>
        </div>
    </form>
</div>

@section Scripts {
    <script>
        $(document).ready(function () {
            $("#saveCredentials").on("click", function () {
                const apiKey = $("#apiKey").val();
                const cxKey = $("#cxKey").val();
                const apiKeyPattern = /^[A-Za-z0-9_-]{39}$/;

                if (!apiKey || !cxKey) {
                    alert("Введіть API ключ та ID пошукового двигуна!");
                    return;
                }

                if (!apiKeyPattern.test(apiKey)) {
                    alert("Невірний формат API ключа!");
                    return;
                }
```

```
            $.post("/Dork/SaveCredentials", { apiKey, cx: cxKey })
                .done(() => {
                    alert("Облікові дані збережено успішно!");
                    window.location.href = "/Dork/Index";
                })
                .fail(() => alert("Помилка при збереженні облікових даних."));
        });
    });
  </script>
}

using AbyssDorks.BayesClassifier;
using AbyssDorks.Services;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllersWithViews();

builder.Services.AddSingleton<HttpClient>();
builder.Services.AddSingleton<JsonDataService>();
builder.Services.AddSingleton<DorkGeneratorService>();
builder.Services.AddSingleton<TagsService>();
builder.Services.AddSingleton<ValidationService>();
builder.Services.AddSingleton<SearchService>();
builder.Services.AddSingleton<ClassifierManager>();

builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
   options.Cookie.Name = ".AbyssDorks.Session";
   options.IdleTimeout = TimeSpan.FromMinutes(30);
   options.Cookie.HttpOnly = true;
   options.Cookie.IsEssential = true;
});

var app = builder.Build();
if (!app.Environment.IsDevelopment())
{
   app.UseExceptionHandler("/Home/Error");
}
app.UseStaticFiles();

app.UseRouting();
app.UseSession();
app.UseAuthorization();

app.MapControllerRoute(
   name: "default",
   pattern: "{controller=Dork}/{action=SetCredentials}/{id?}");

app.Run();
```

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Чернова Сергія Володимировича
ПІБ здобувача вищої освіти
Студента ФІТ, 3 курсу, групи КБс-22-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (StrikePlagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

02.06.25
дата

підпис

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 9%**

ID: 243249 Title: Система генерації пошукових запитів для виявлення вразливостей та витоків даних інтернет-ресурсів Added in a DB: 2025-06-03 Authors: Чернов Сергій Володимирович Heads: Чешун В.М. Consultants: Opponents:	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
	73758	1118	623 (1%)	7 (1%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Чернов Сергій Володимирович

Співавтор:

Назва: Система генерації пошукових запитів для виявлення вразливостей та витоків даних інтернет-ресурсів

Науковий керівник:

Підрозділ: Кафедра кібербезпеки

Коефіцієнт подібності 1:1%

Коефіцієнт подібності 2:0.3%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-04 06:18:42.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 04.06.2025 Керівник /Нечич В.М./

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Система генерації пошукових запитів для виявлення вразливостей та витоків даних інтернет-ресурсів

Автор: Чернов Сергій Володимирович

Спеціальність: 125 – Кібербезпека

Освітня програма:Кібербезпека

Науковий керівник: Віктор ЧЕШУН, канд. техн. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою StrikePlagiarism складає 99%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism складає 99%.

Згідно з правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 24.09.2024, авторська робота, обсяг оригінального тексту у відсотках до загального обсягу матеріалу в якій складає 90-100%, визначається роботою з високою унікальністю тексту і допускається до захисту.

Керівник роботи _____ Віктор ЧЕШУН

Гарант ОП _____ Віктор ЧЕШУН

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студен Чернов Сергій Володимирович

Тема : Система генерації пошукових запитів для виявлення вразливостей та витоків даних інтернет-ресурсів

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека та захист інформації

Обсяг кваліфікаційної роботи освітнього ступеня «бакалавр»:

кількість листів креслень 3 ; кількість сторінок записки 69

1. Короткий зміст роботи та прийнятих рішень Кваліфікаційна робота присвячена розробці веб-додатку AbyssDorks – інструменту автоматизованої генерації пошукових запитів (dorks), орієнтованого на виявлення потенційних витоків або вразливостей у відкритому доступі. У роботі виконано аналіз методів OSINT-моніторингу, Google Dorks, досліджено інструменти на кшталт Shodan, FOCA, Censys. На основі виявлених недоліків існуючих рішень сформульовано вимоги до розробки власної системи. Реалізовано модульну веб-систему з підтримкою Google-API, шаблонізації запитів, категоризації за модулями й тегами, а також алгоритмів класифікації результатів за критичністю.

2. Висновок про відповідність кваліфікаційної роботи завданню Кваліфікаційна робота відповідає поставленому завданню як в теоретичній, так і в практичній частині.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі окреслено актуальність теми, мету і завдання. У першому розділі виконано огляд інструментів OSINT, досконало проаналізовано принципи Google Hacking. У другому – реалізовано веб-додаток з багаторівневою архітектурою, описано сервіси, що відповідають за генерацію запитів, фільтрацію, збереження шаблонів і взаємодію з API Google. Особливу увагу приділено побудові логіки інтерфейсу користувача та класифікації результатів. У третьому розділі проведено UX-аналіз, сценарне тестування і верифікацію функціональності системи в умовах, наближених до реальних.

4. Позитивні сторони роботи Система має сучасний, інтуїтивно зрозумілий інтерфейс, гнучку модульну структуру, підтримку шаблонів, API, класифікації та адаптації під потреби різних користувачів. Реалізовано унікальну комбінацію OSINT-практик та автоматизації пошукових запитів, що підвищує ефективність перевірки витоків даних у відкритому доступі.

5. Негативні сторони роботи Доцільно було б окремо визначити обмеження веб-додатку щодо продуктивності при великій кількості запитів, а також надати рекомендації з інтеграціями з іншими OSINT-інструментами або API сканерів.

6. Оцінка графічного оформлення та пояснювальної записки роботи Оформлення всіх матеріалів кваліфікаційної роботи є якісним, здійснене з дотриманням актуальних стандартів та інституційних положень ХНУ. Пояснювальна записка відповідає нормам щодо її оформлення як за структурою, так і за представленням і форматуванням матеріалу.

7. Відгук про роботу в цілому Кваліфікаційна робота заслуговує на позитивну оцінку. Весь матеріал кваліфікаційної роботи структурований, чіткий та наскрізно пов'язаний. Усі розділи роботи послідовні та логічні, що дозволяє чітко розуміти викладений матеріал в рамках тематики кваліфікаційної роботи. Графічний та ілюстративний матеріал дозволяє наочно побачити доцільність та ефективність рішень, які були прийняті за основу для досягнення поставленої мети.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Враховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінку «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Підченко Сергій Костянтинович
завідувач кафедри ТМІТ, доктор технічних наук, професор

«6» 06 2025. (підпис)