

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки**

«На правах рукопису»
УДК 004.056.5:004.8

До захисту допущено:
В.О. Завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
«__» _____ 20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

зі спеціальності 123 «Комп'ютерна інженерія»

**на тему: «Система автоматичного виявлення фішингових сайтів на основі баз
відкритих загроз та поведінкових ознак»**

Виконала:

студентка VI курсу, групи ІО-41мп
Лавріненко Юлія Павлівна _____

Керівник:

Доц. каф. ОТ, к.т.н., доц.
Верба Олександр Андрійович _____

Консультант:

Проф. каф. ОТ, д.т.н., проф.
Кулаков Юрій Олексійович _____

Рецензент:

Доц. каф. ІСТ, к.т.н., доц.
Шимкович Володимир Миколайович _____

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць
інших авторів без відповідних
посилань.

Студент _____

Київ – 2025 рік

**Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»**

Факультет (інститут) Інформатики та обчислювальної техніки
(повна назва)

Кафедра Обчислювальної техніки
(повна назва)

Освітньо-кваліфікаційний ступінь магістр
(назва ОКР)

Спеціальність 123. Комп'ютерна інженерія
(код і назва)

Спеціалізація 123. Комп'ютерні системи та мережі
(код і назва)

ЗАТВЕРДЖУЮ

В.О. Завідувача кафедри

М.А. НОВОТАРСЬКИЙ
(підпис) (ініціали, прізвище)

« » _____ 2025 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Лавріненко Юлії Павлівні
(прізвище, ім'я, по батькові)

1. Тема дисертації Система автоматичного виявлення фішингових сайтів на основі баз відкритих загроз та поведінкових ознак

Науковий керівник дисертації Верба Олександр Андрійович, к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «06» 11 2025 р. № 4841-с

2. Строк подання студентом дисертації 05.12.2025

3. Об'єкт дослідження процес автоматичного виявлення фішингових сайтів у мережі Інтернет

4. Предмет дослідження методи комбінованого аналізу веб-ресурсів, що поєднують перевірку у базах відкритих загроз, аналіз контенту веб-сторінки та поведінкові ознаки взаємодії користувача.

5. Перелік завдань, які потрібно розробити: аналіз існуючих підходів і рішень для автоматичного виявлення фішингових сайтів; проєктування мікросервісної архітектури; реалізація модулів перевірки; розробка

поведінкової симуляції; організація зберігання результатів; створення веб-інтерфейсу; проведення тестування роботи системи.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормконтроль	Кулаков Ю. О.		
1	Кулаков Ю. О.		
2	Кулаков Ю. О.		
3	Кулаков Ю. О.		
4	Кулаков Ю. О.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Затвердження теми дослідження. Визначення предмету дослідження	02.09.25 - 08.09.25	
2	Аналіз наукових джерел і існуючих систем	09.09.25 - 15.09.25	
3	Порівняння підходів, вибір стеку	16.09.25 - 22.09.25	
4	Проектування архітектури та схем взаємодії мікросервісів	23.09.25 - 29.09.25	
5	Реалізація сервісу перевірки посилань та домену у відкритих базах загроз	30.09.25 - 06.10.25	
6	Реалізація контентного та поведінкового аналізатора	07.10.25 - 13.10.25	
7	Інтеграційне тестування та налагодження контейнерів	14.10.25 - 20.10.25	
8	Фінальне тестування сервісів, оформлення матеріалів і демонстрації	21.10.25 - 26.10.25	
9	Розробка стартап-проекту	27.10.25 - 03.11.25	
10	Оформлення магістерської дисертації	04.11.25 - 05.12.25	

Студент

(підпис)

Юлія ЛАВРІНЕНКО
(ініціали, прізвище)

Науковий керівник дисертації

(підпис)

Олександр ВЕРБА
(ініціали, прізвище)

РЕФЕРАТ

На магістерську дисертацію

Виконано на тему: Система автоматичного виявлення фішингових сайтів на основі баз відкритих загроз та поведінкових ознак

Студенткою: Лавріненко Юлією Павлівною

Робота містить вступ та чотири розділи. Загальний обсяг: 140 аркушів основного тексту, 39 рисунків, 35 таблиць, 1 формула, 6 додатків. При підготовці роботи використовувалась література з 33 джерел.

Актуальність. Фішингові вебсайти залишаються одним із наймасовіших інструментів соціальної інженерії та становлять значний ризик для користувачів і організацій. Зростання кількості шаблонних сервісів створення сайтів і автоматизації контенту ускладнює виявлення фішингу лише за статичними ознаками, тому актуальним є застосування комбінованого підходу з урахуванням репутаційних даних, вмісту сторінки та поведінкових індикаторів.

Фішингові кампанії швидко змінюють домени, структуру сторінок і сценарії взаємодії з користувачем, через що «чорні списки» та сигнатурні методи часто не встигають оновлюватися. Для бізнесу наслідками таких атак є прямі фінансові втрати, компрометація облікових записів, витоки даних і репутаційні збитки, а для команд кібербезпеки – зростання навантаження на виявлення інцидентів. У цьому контексті важливими стають рішення, що забезпечують не лише виявлення, а й пояснюваність результатів та можливість інтеграції в існуючі процеси реагування.

Мета та завдання дослідження. Метою роботи є розробка та реалізація системи автоматичного виявлення фішингових сайтів на основі відкритих баз загроз, контентного й поведінкового аналізу та симуляції дій користувача у вигляді набору мікросервісів у контейнерах.

Для досягнення мети виконано наступні завдання:

- проаналізовано існуючі підходи до виявлення фішингових сайтів та відкриті бази загроз;

- спроектовано мікросервісну архітектуру системи з використанням контейнеризації Docker;
- розроблено модулі перевірки у відкритих базах загроз, аналізу домену та вебконтенту;
- реалізовано механізм поведінкової симуляції дій користувача з фіксацією підозрілих індикаторів;
- створено базу даних для зберігання результатів перевірок та вебінтерфейс для роботи з системою;
- проведено тестування системи та аналіз отриманих результатів.

Об'єкт дослідження – процес автоматичного виявлення фішингових сайтів у мережі Інтернет.

Предмет дослідження – методи комбінованого аналізу вебресурсів, що поєднують перевірку у відкритих базах загроз, аналіз контенту сторінки та поведінкові ознаки користувача.

Методи дослідження. Для досягнення поставленої мети використано мікросервісний підхід, клієнт-серверну взаємодію та проектування REST API, засоби автоматизованого отримання та аналізу вебконтенту, методи виявлення поведінкових індикаторів, а також зберігання та обробку даних.

Наукова новизна одержаних результатів полягає у:

- запропоновано комбіновану мікросервісну систему аналізу фішингових сайтів, у якій результати з відкритих баз загроз, контентного аналізу та поведінкових ознак інтегруються в єдиному середовищі з централізованою агрегацією ризику;
- поведінковий аналіз доповнено симуляцією дій користувача з фіксацією індикаторів.

Особистий внесок здобувача. Магістерське дослідження є самостійно виконаною роботою, у якій відображено авторський підхід та особисто отримані теоретичні й прикладні результати, що стосуються розробки системи

автоматичного виявлення фішингових сайтів. Формулювання мети та завдань дослідження проводилось спільно з науковим керівником.

Практична цінність і сфера застосування. Результати можуть бути використані для автоматизованого моніторингу підозрілих вебресурсів у корпоративних або державних середовищах, у сценаріях попередньої перевірки URL та навчальних стендів з кібербезпеки.

Ключові слова: GOOGLE AI API, GOOGLE WEB RISK API, LARGE LANGUAGE MODEL, SELENIUM, VIRUSTOTAL API, ВЕБ-ФІШИНГ, ВИЯВЛЕННЯ ФІШИНГУ, ІНТЕГРАЛЬНА ОЦІНКА, МІКРОСЕРВІСНА АРХІТЕКТУРА, ПОВЕДІНКОВА СИМУЛЯЦІЯ.

ABSTRACT

For a master's thesis

Made on topic: Automatic detection system for phishing sites based on open threat intelligence databases and behavioral indicators

By student: Lavrinenko Yuliia Pavlivna

The study consists of an introduction and four chapters. Total volume: 140 pages of main text, 39 figures, 35 tables, 1 formula, 6 appendices. Literature from 33 sources was used in the preparation of the work.

Relevance. Phishing websites remain one of the most widespread social engineering tools and pose a significant risk to users and organizations. The growth in the number of template-based website creation and content automation services makes it difficult to detect phishing based on static characteristics alone, so it is important to use a combined approach that takes into account reputation data, page content, and behavioral indicators.

Phishing campaigns quickly change domains, page structures, and user interaction scenarios, which means that blacklists and signature-based methods often cannot keep up with updates. For businesses, the consequences of such attacks include direct financial losses, compromised accounts, data leaks, and reputational damage, while for cybersecurity teams, they mean an increased workload in detecting incidents. In this context, solutions that not only detect but also explain the results and can be integrated into existing response processes are becoming increasingly important.

Research objectives and tasks. The objective of this work is to develop and implement a system for automatic detection of phishing sites based on open threat databases, content and behavioral analysis, and user action simulation in the form of a set of microservices in containers.

To achieve this goal, the following tasks were completed:

- existing approaches to detecting phishing sites and open threat databases were analyzed;

- a microservice architecture for the system was designed using Docker containerization;
- modules for checking open threat databases, domain analysis, and web content were developed;
- a mechanism for behavioral simulation of user actions with the recording of suspicious indicators was implemented;
- a database for storing check results and a web interface for working with the system were created;
- The system was tested and the results were analyzed.

The object of the study is the process of automatically detecting phishing sites on the Internet.

The subject of the study is methods of combined analysis of web resources, combining verification in open threat databases, analysis of page content, and user behavior patterns.

Research methods. To achieve the set goal, a microservice approach, client-server interaction and REST API design, means of automated web content retrieval and analysis, methods of detecting behavioral indicators, as well as data storage and processing were used.

The scientific novelty of the results obtained lies in:

- a combined microservice system for analyzing phishing sites has been proposed, in which results from open threat databases, content analysis, and behavioral characteristics are integrated into a single environment with centralized risk aggregation;
- behavioral analysis is supplemented by simulation of user actions with indicator recording.

Personal contribution of the applicant. The master's research is an independently completed work that reflects the author's approach and personally obtained theoretical and applied results related to the development of a system for automatic detection of

phishing sites. The formulation of the research objectives and tasks was carried out jointly with the scientific supervisor.

Practical value and scope of application. The results can be used for automated monitoring of suspicious web resources in corporate or government environments, in URL pre-checking scenarios, and in cybersecurity training stands.

Keywords: GOOGLE AI API, GOOGLE WEB RISK API, LARGE LANGUAGE MODEL, SELENIUM, VIRUSTOTAL API, WEB PHISHING, PHISHING DETECTION, INTEGRAL EVALUATION, MICROSERVICE ARCHITECTURE, BEHAVIORAL SIMULATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ	13
ВСТУП.....	15
РОЗДІЛ 1 АНАЛІЗ ВІДОМИХ ПІДХОДІВ АВТОМАТИЧНОГО ВИЯВЛЕННЯ ФІШИНГОВИХ САЙТІВ НА ОСНОВІ БАЗ ВІДКРИТИХ ЗАГРОЗ ТА ПОВЕДІНКОВИХ ОЗНАК.....	18
1.1. Огляд загальних відомостей про фішинг.....	18
1.1.1. Класифікація типів фішингу	18
1.1.2. Життєвий цикл фішингової атаки	20
1.1.3. Характерні технічні індикатори.....	22
1.1.4. Бізнес-наслідки та мотивація для детекції	24
1.2. Аналіз існуючих технічних рішень і баз загроз для виявлення фішингу.....	25
1.2.1. VirusTotal	25
1.2.2. Google Web Risk та Safe Browsing	28
1.2.3. Спільні відкриті репозиторії та сервіси	29
1.2.4. Браузерні захисти та комерційні рішення.....	31
1.2.5. Висновки до аналізу існуючих технічних рішень і баз загроз	35
1.3. Аналіз методологічних підходів до детекції фішингу.....	36
1.3.1. Перевірка по базах даних	36
1.3.2. Статичний аналіз контенту сторінки.....	37
1.3.3. Поведінковий аналіз.....	38
1.3.4. ML/LLM-підходи до аналізу контенту та візуальної подібності.....	39
1.3.5. Гібридні стратегії	41
1.3.6. Висновки до аналізу методологічних підходів.....	42
1.4. Формалізація задачі та постановка часткових задач досліджень	43

ВИСНОВОК ДО РОЗДІЛУ 1	46
РОЗДІЛ 2 РОЗРОБКА СИСТЕМИ АВТОМАТИЧНОГО ВИЯВЛЕННЯ ФІШИНГОВИХ САЙТІВ НА ОСНОВІ БАЗ ВІДКРИТИХ ЗАГРОЗ ТА ПОВЕДІНКОВИХ ОЗНАК.....	47
2.1. Розробка структурної схеми системи автоматичного виявлення фішингових сайтів.....	47
2.1.1. Загальна архітектура системи та взаємодія мікросервісів.....	50
2.1.2. Опис основних компонентів системи.....	52
2.1.3. Використання контейнеризації Docker	56
2.2. Вибір технологій для реалізації мікросервісів системи.....	58
2.3. Розробка програмної компоненти системи автоматичного виявлення фішингових сайтів.....	61
2.3.1. Інженерія вимог до функціональних компонентів системи	61
2.3.2. Архітектурне проєктування програмного забезпечення	64
2.3.3. Реалізація сервісу перевірки посилань	66
2.3.4. Реалізація сервісу перевірки доменів.....	69
2.3.5. Реалізація сервісу аналізу контенту сторінки.....	72
2.3.6. Реалізація сервісу симуляції дій користувача.....	75
2.3.7. Реалізація сервісу агрегатора результатів.....	79
2.3.8. Реалізація бази даних та сервісу-інтерфейсу	82
2.3.9. Реалізація демонстраційного веб-інтерфейсу	84
ВИСНОВОК ДО РОЗДІЛУ 2	87
РОЗДІЛ 3 ПРИКЛАДНІ АСПЕКТИ ЗАСТОСУВАННЯ СИСТЕМИ АВТОМАТИЧНОГО ВИЯВЛЕННЯ ФІШИНГОВИХ САЙТІВ НА ОСНОВІ БАЗ ВІДКРИТИХ ЗАГРОЗ ТА ПОВЕДІНКОВИХ ОЗНАК	89
3.1. Демонстрація практичних можливостей розробленої системи.....	89

	12
3.1.1. Розгортання та підготовка середовища	89
3.1.2 Демонстрація сценарію перевірки.....	94
3.1.3 Демонстрація поведінкового аналізу і ланцюга переходів	102
3.1.4 Демонстрація кешування та повторної перевірки	106
3.2. Практичне застосування та можливості інтеграції системи.....	108
3.3. Програмна документація на розроблене програмне забезпечення	109
ВИСНОВОК ДО РОЗДІЛУ 3	121
РОЗДІЛ 4 РОЗРОБКА СТАРТАП-ПРОЄКТУ	122
4.1. Опис ідеї проєкту	122
4.2. Технологічний аудит ідеї проєкту.....	125
4.3. Аналіз ринкових можливостей запуску стартап-проєкту	128
4.4. Розроблення ринкової стратегії проєкту	138
4.5. Розроблення маркетингової програми стартап-проєкту.....	142
ВИСНОВОК ДО РОЗДІЛУ 4	148
ВИСНОВКИ.....	149
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	151
ДОДАТКИ	155
ДОДАТОК А	156
ДОДАТОК Б	164
ДОДАТОК В.....	170
ДОДАТОК Г	183
ДОДАТОК Д.....	207
ДОДАТОК Е.....	219

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API	-	Application Programming Interface
B2B	-	Business-to-Business
BEC	-	Business Email Compromise
CSIRT	-	Computer Security Incident Response Team
CSS	-	Cascading Style Sheets
DNS	-	Domain Name System
DOM	-	Document Object Model
EDR	-	Endpoint Detection and Response
GDPR	-	General Data Protection Regulation
HTML	-	HyperText Markup Language
HTTP	-	HyperText Transfer Protocol
HTTPS	-	HyperText Transfer Protocol Secure
IP	-	Internet Protocol
JSON	-	JavaScript Object Notation
JSONB	-	JavaScript Object Notation Binary
LLM	-	Large Language Model
MSSP	-	Managed Security Service Provider
MVP	-	Minimum Viable Product
QR	-	Quick Response (code)
REST	-	Representational State Transfer
ROI	-	Return on Investment
SaaS	-	Software as a Service
SCM	-	Source Control Management
SEO	-	Search Engine Optimization
SIEM	-	Security Information and Event Management
SLA	-	Service Level Agreement
SMB	-	Small and Medium-sized Business
SMS	-	Short Message Service

SOC	-	Security Operations Center
SQL	-	Structured Query Language
SSE	-	Server-Sent Events
SSL	-	Secure Sockets Layer
TCO	-	Total Cost of Ownership
UI	-	User Interface
URL	-	Uniform Resource Locator
UX	-	User Experience
XDR	-	Extended Detection and Response
БД	-	База даних
ІБ	-	Інформаційна безпека
ІТ	-	Інформаційні технології
ПЗ	-	Програмне забезпечення

ВСТУП

Фішингові атаки це одна з найпоширеніших і найнебезпечніших загроз у сфері кібербезпеки. Їх суть полягає у використанні методів соціальної інженерії, коли зловмисники, видаючи себе за надійні джерела або осіб, отримують доступ до конфіденційної інформації користувачів. Із розвитком штучного інтелекту та автоматизованих інструментів створення контенту, рівень складності фішингових атак суттєво зріс, що значно ускладнило їхнє виявлення.

У першій половині 2025 року в Україні зафіксовано понад 3000 кіберінцидентів – на 17 % більше, ніж за попередні шість місяців. Найбільшу частку серед них (27%) займають саме фішингові атаки. Все частіше такі атаки координуються з іншими кіберопераціями або навіть ракетними ударами, що свідчить про використання фішингу як одного з інструментів гібридної війни [1].

У світовому контексті, фішинг залишається головним каналом компрометації даних – 80% фішингових кампаній націлені на викрадення облікових даних користувачів; понад 80% фішингових сайтів використовують HTTPS, що ускладнює їхнє візуальне виявлення як шахрайських [2]. Крім того, світова практика показує, що фішингові атаки еволюціонують: зловмисники дедалі частіше використовують методи щодо масового створення підроблених сайтів, штучний інтелект (ШІ) для створення переконливих фішингових повідомлень.

Актуальність роботи обумовлена тим, що під час військового стану в Україні кіберпростір набуває особливого значення: інформаційна інфраструктура – це стратегічний актив, а фішингові атаки можуть бути не лише фінансово-шахрайськими, але й спрямованими на компрометацію персональних даних, військових комунікацій чи державних служб. Крім того, через зростання кількості фішингових атак і з огляду на те, що традиційні методи захисту (лише на рівні антивірусів, сигнатурних баз) стають недостатніми, виникає потреба в новому інженерному рішенні – системі, що об'єднує різнорівневу перевірку (бази даних, поведінковий аналіз, симуляція дій). Це створює підстави для розроблення комплексної автоматизованої системи, що поєднує кілька рівнів захисту.

Метою роботи є розробка та реалізація системи автоматичного виявлення фішингових сайтів, яка використовує бази відкритих загроз, поведінковий аналіз веб-контенту та симуляцію дій користувача, реалізована як набір мікросервісів у контейнерах.

Завданням дослідження є:

- аналіз існуючих підходів і технологічних рішень для автоматичного виявлення фішингових сайтів;
- розроблення архітектури системи з використанням мікросервісного підходу та контейнеризації Docker;
- створення окремих модулів перевірки посилань у відкритих базах загроз, аналізу доменів та контенту сторінки;
- розроблення механізму поведінкової симуляції користувача;
- створення бази даних для зберігання результатів перевірок та реалізація веб-інтерфейсу взаємодії;
- проведення тестування роботи системи.

Об'єктом дослідження є процес автоматичного виявлення фішингових сайтів у мережі Інтернет.

Предметом дослідження є методи комбінованого аналізу веб-ресурсів, що поєднують перевірку у базах відкритих загроз, аналіз контенту сторінки та поведінкові ознаки користувача.

Методами дослідження є аналіз предметної області, узагальнення відомих підходів до автоматичного виявлення фішингових сайтів, моделювання процесів обробки та аналізу даних, а також проєктування і реалізація програмної системи з використанням сучасних засобів розробки та методів інженерії програмного забезпечення.

Наукова новизна полягає у створенні комбінованої системи аналізу фішингових сайтів, яка об'єднує результати з відкритих баз загроз, аналіз контенту

сторінки та поведінкові ознаки користувача в єдиному мікросервісному середовищі.

Практичне значення одержаних результатів полягає у можливості інтеграції розробленої системи у корпоративні або державні платформи моніторингу безпеки для автоматичного виявлення фішингових сайтів у реальному часі, що підвищує рівень кіберзахисту в умовах сучасних загроз.

РОЗДІЛ 1

АНАЛІЗ ВІДОМИХ ПІДХОДІВ АВТОМАТИЧНОГО ВИЯВЛЕННЯ ФІШИНГОВИХ САЙТІВ НА ОСНОВІ БАЗ ВІДКРИТИХ ЗАГРОЗ ТА ПОВЕДІНКОВИХ ОЗНАК

У сучасному світі фішингові веб-сайти є основним джерелом кіберзлочинності, оскільки вони призводять до того, що люди втрачають облікові дані, платіжну інформацію або доступ до корпоративних ресурсів. Завдяки зростанню автоматизації фішингових кампаній і постійному появі нових доменів проблема їх виявлення стає ще більш актуальною. Таким чином, розробка ефективних методів і систем для автоматичного аналізу веб-сайтів є основним напрямком підвищення кіберстійкості підприємств і державних установ.

У цьому розділі розглянуто теоретичні та практичні аспекти виявлення фішингових вебсайтів. Проведено огляд природи фішингу, його основних типів, життєвого циклу та характерних технічних індикаторів, які лежать в основі сучасних методів детекції. Далі проаналізовано існуючі технічні рішення, бази відкритих загроз і методологічні підходи, що використовуються у виявленні фішингових сайтів, зокрема на рівні статичного, поведінкового та гібридного аналізу. Це дозволяє визначити недоліки наявних систем і сформулювати науково-технічне підґрунтя для подальшої розробки власної комплексної системи автоматичного аналізу вебресурсів.

1.1. Огляд загальних відомостей про фішинг

1.1.1. Класифікація типів фішингу

Фішинг являє собою одну з найпоширеніших форм кіберзлочинів, що використовує соціальну інженерію для отримання конфіденційних даних користувачів – логінів, паролів, платіжних реквізитів, персональних даних тощо. Основна ідея фішингу полягає в тому, щоб змусити користувача довіритися фальшивому ресурсу, який імітує легітимний вебсайт або службу. У сучасному середовищі більшість таких атак орієнтована саме на веб-сайти, що

використовують підроблені домени, копії відомих брендів, та шкідливі посилання [3].

Основні типи фішингу:

- Email phishing – класичний і найпоширеніший тип, у якому користувачу надсилають електронні листи, що виглядають як повідомлення від відомих компаній чи банків. Мета – змусити перейти за посиланням на фальшивий сайт, де користувач самостійно вводить свої дані.
- Spear phishing – цілеспрямована атака, яка використовує персональні дані жертви (посаду, місце роботи, коло контактів) для створення правдоподібного повідомлення. Такий підхід робить атаку значно ефективнішою.
- Whaling (цільовий фішинг на керівництво) – різновид spear phishing, спрямований на топменеджерів чи фінансових керівників. Мета – отримання доступу до корпоративних ресурсів або здійснення шахрайських транзакцій.
- Business Email Compromise (BEC) – атака, у якій зловмисник підміняє або компрометує корпоративну адресу керівника, надсилаючи фальшиві листи від його імені з вимогою здійснити оплату чи передати дані.
- HTTPS phishing – створення фальшивих вебсайтів із дійсним SSL-сертифікатом (https://), що вводять користувача в оману. Цей тип особливо небезпечний для веб-платформ, адже підроблений сайт може виглядати цілком безпечним.
- Website spoofing – повна імітація відомого вебсайту (логотип, стиль, домен зі схожою назвою). Такі сайти використовуються для збору логінів, паролів або платіжних даних користувачів.
- DNS spoofing (pharming) – технічна атака, за якої користувач потрапляє на підроблений вебсайт, навіть якщо вводить правильну адресу. Це досягається шляхом маніпуляцій із DNS-записами [4].
- Clone phishing – копіювання легітимного листа або вебсторінки з незначними змінами (підміна посилання або файлу). Користувач вважає, що взаємодіє з перевіреним ресурсом, тоді як насправді – із підробкою.

- Search engine phishing (SEO poisoning) – створення підроблених сайтів, які просуваються у результатах пошуку за популярними запитами (наприклад, «вхід до банку», «служба підтримки»). Користувач переходить на них, не підозрюючи шахрайства.
- Pop-up phishing – використання шкідливих спливаючих вікон на вебсайтах. Вони можуть виглядати як системні повідомлення («Ваш комп'ютер заражено» або «Оновіть антивірус») та спонукають до натискання на фальшиве посилання.
- Quishing (QR phishing) – сучасний різновид, у якому користувачеві пропонується відсканувати QR-код, що веде на підроблений вебсайт або активує завантаження шкідливого ПЗ [4].
- Evil Twin – створення фальшивої точки Wi-Fi-доступу, яка імітує легітимну мережу. Після підключення користувача зловмисник отримує доступ до його трафіку, включно з веб-запитами [3].
- Malware-based phishing – поєднання вебфішингу із завантаженням шкідливого ПЗ через вебсторінку або вкладення. Після зараження комп'ютера користувача зловмисник може отримати повний контроль над його системою [4].

Більшість сучасних фішингових атак базується саме на імітації вебресурсів та маніпуляціях із URL-адресами, що робить веб-фішинг найбільш загрозливим напрямом. Поштові, SMS- і телефонні фішинги залишаються важливими, однак вони здебільшого є допоміжними механізмами для залучення користувачів на підроблені вебсайти. Таким чином, основним об'єктом захисту є веб-середовище, у якому здійснюється взаємодія користувача з фальшивими сторінками, що імітують законні онлайн-сервіси.

1.1.2. Життєвий цикл фішингової атаки

Життєвий цикл фішингової атаки складається з послідовних етапів, що охоплюють підготовку, реалізацію та подальше використання отриманих даних.

Незалежно від каналу доставки (електронна пошта, соціальні мережі чи SMS), ключовою метою кожного етапу залишається спонукання користувача до взаємодії з підробленим вебресурсом, де відбувається крадіжка його облікових даних або фінансової інформації [5].

Перший етап полягає у збиранні інформації про потенційну жертву та створенні максимально достовірного фішингового середовища. Зловмисники аналізують соціальні профілі, публічні бази даних, історію доменів та навіть витoki з незначних сервісів, щоб сформува ти уявлення про поведінку користувача або компанії. На цьому етапі створюються імітації легітимних вебсайтів – фальшиві сторінки входу, копії банківських або корпоративних порталів, форми авторизації. Зовні такі ресурси виглядають цілком безпечними, оскільки використовують HTTPS, коректні логотипи й адаптивний дизайн. Таким чином, перший етап формує основу атаки – правдоподібність вебсайту, що має викликати довіру та змусити користувача взаємодіяти з ним.

Другий етап це активна взаємодія з жертвою, коли атакувальний ресурс або повідомлення спонукає користувача перейти на підроблений вебсайт. Зловмисники створюють емоційний або інформаційний тиск – повідомлення про «підозрілу активність», «заблокований акаунт» або «термінову необхідність підтвердження даних». Користувач, бажаючи якнайшвидше вирішити проблему, переходить за посиланням на підроблену сторінку та вводить свої облікові дані. Для веб-фішингу саме цей етап є ключовим, адже користувач здійснює перехід і вводить інформацію у форму, візуально схожу на офіційну. У деяких випадках сторінка автоматично перенаправляє користувача на справжній сайт після відправлення даних, що ускладнює виявлення шахрайства [6].

На завершальному етапі фішингової атаки зловмисники отримують і використовують викрадену інформацію. Щодо фішингових сайтів, отримані облікові дані можуть бути використані для входу до банківських систем, внутрішніх корпоративних мереж або поштових сервісів. Далі ці акаунти часто застосовуються для ланцюгових атак, коли з заражених облікових записів розсилаються нові фішингові листи. Крім того, отримані дані можуть використовуватися для

довгострокового доступу – наприклад, для викрадення фінансової інформації або подальшої компрометації корпоративних систем.

Таким чином, життєвий цикл фішингової атаки можна описати як замкнену послідовність дій: збирання інформації; створення фальшивого вебресурсу; залучення користувача; викрадення даних; подальша експлуатація або повторне використання отриманої інформації. У контексті веб-фішингу всі етапи зосереджені навколо взаємодії користувача з підробленою сторінкою, яка є основним інструментом атаки. Розуміння цього циклу дозволяє ефективніше будувати системи виявлення фішингових сайтів та створювати превентивні механізми захисту.

1.1.3. Характерні технічні індикатори

Однією з найважливіших задач у виявленні фішингових атак є ідентифікація технічних індикаторів компрометації, тобто сукупності ознак, які дозволяють розпізнати шкідливий або підроблений вебресурс. Для веб-фішингу характерна комбінація зовнішніх (видимих користувачу) та внутрішніх (структурних, мережевих) показників, які відрізняють фальшиві сайти від легітимних.

Одним із перших елементів, що видає фішинговий сайт, є аномалії в доменній адресі. Типові ознаки включають: незвичні або зайві піддомени, наприклад login-security-bank.com замість bank.com; гомографічні атаки – заміна символів на схожі візуально (rn замість m, vv замість w); використання новостворених або короткочасних доменів – фішингові сайти зазвичай існують кілька днів або тижнів; використання URL-скорочувачів (bit.ly, tinyurl.com), що приховують справжню адресу [2]; невідповідність доменного імені та змісту сторінки – наприклад, сайт із доменом, схожим на банківський, але без жодних ознак фінансової установи.

Такі URL-маскування дозволяють обійти базові фільтри безпеки та створити ілюзію легітимності. Навіть HTTPS-протокол не гарантує безпеки – фішингові сторінки часто використовують безкоштовні SSL-сертифікати, тому наявність замка у браузері не є показником достовірності ресурсу [6].

Додатковими індикаторами є характеристики домену, які можна перевірити через WHOIS або інші сервіси: молодий вік домену (до 3 місяців) часто вказує на його одноразове використання у фішинговій кампанії; прихована інформація про реєстратора або власника; реєстрація у нестандартних доменних зонах (.xyz, .top, .club тощо), які часто використовуються зловмисниками для економії коштів і уникнення блокування.

Фішингові вебсайти відрізняються від легітимних і за структурою HTML-документа: велика кількість зовнішніх скриптів або підключених елементів із неперевіраних джерел; некоректна верстка, невідповідність стилів, різні шрифти, неякісні зображення або некоректні логотипи; наявність елементів, що імітують форми входу (логін, пароль, дані картки) без функціоналу двофакторної перевірки або безпосередньої взаємодії із сервером автентифікації.

Фішингові сторінки також можуть містити елементи соціальної інженерії у вигляді попереджень («ваш акаунт буде заблоковано»), закликів («підтвердіть зараз»), або анімацій, що створюють відчуття терміновості.

На технічному рівні для фішингових сайтів характерні такі особливості: відсутність або аномальна конфігурація DNS-записів; IP-адреси, що належать до підозрілих або нещодавно зареєстрованих хостинг-провайдерів; відсутність реального SSL-сертифіката від авторитетного центру сертифікації; аномальна поведінка сторінки – автоматичні редиректи, приховане завантаження скриптів або підключення до сторонніх API без очевидної причини [2].

У більш розвинених випадках спостерігається використання JavaScript-обфускації або динамічного завантаження елементів, які змінюють структуру DOM після відкриття сторінки. Це ускладнює виявлення фішингових шаблонів як для антивірусів, так і для аналітичних систем [6].

Окрім технічних параметрів, важливими є поведінкові індикатори. Якщо користувач вводить дані на сайті, система може зафіксувати: відправку конфіденційних даних у незашифрованому вигляді; аномальні запити POST/GET до сторонніх серверів; повторні редиректи між різними доменами перед отриманням

відповіді [6]. Такі ознаки свідчать про спробу викрадення введеної інформації або збору токенів авторизації через проміжні сервери.

Характерні технічні індикатори фішингових вебсайтів охоплюють сукупність ознак у доменній, структурній та поведінковій площинах. Основним критерієм залишається невідповідність між зовнішнім виглядом сайту, його доменом та фактичною логікою взаємодії.

1.1.4. Бізнес-наслідки та мотивація для детекції

Фішинг на вебсайтах має не лише технічну, а й глибоку бізнесову складову, адже його наслідки безпосередньо впливають на фінансову стабільність, репутацію та функціонування організацій. Хоча кінцева мета зловмисників не завжди полягає у прямому викраденні коштів, більшість атак спрямована на отримання конфіденційних даних, які надалі використовуються для фінансового або конкурентного шантажу. Це робить завдання автоматичного виявлення фішингових вебресурсів не лише технічною, а й стратегічною бізнес-потребою.

Основний наслідок успішної фішингової атаки – прямі фінансові втрати. Зловмисники часто імітують корпоративні платформи, вебсайти банків або постачальників, змушуючи користувачів здійснювати платежі на підроблені рахунки або вводити банківські реквізити. Однак не менш серйозним є викрадення корпоративних даних через фішингові сайти, які імітують внутрішні вебпортали або системи входу. Отримавши такі облікові дані, зловмисники отримують можливість доступу до внутрішніх баз, що може призвести до масових витоків персональної інформації клієнтів або комерційної таємниці [7].

Фішингові атаки на вебресурси здатні паралізувати роботу компанії. Потрапивши у внутрішню мережу, зловмисники можуть встановити шкідливе ПЗ, заблокувавши доступ до корпоративних систем. Навіть тимчасове порушення вебінфраструктури може коштувати компанії втрати клієнтів, зниження продажів або неможливості виконання контрактних зобов'язань.

Найглибші втрати від фішингових атак часто нематеріальні. Після компрометації корпоративного вебсайту або витоків даних довіра клієнтів різко

падає. Зловмисники можуть використовувати викрадені доменні адреси для розсилки спаму або шкідливих листів від імені компанії, що призводить до асоціації бренду з шахрайством. Поширення інформації про інцидент у медіа призводить до публічного іміджевого удару, який може зберігатися роками.

Фішингові інциденти часто впливають на капіталізацію та інвестиційний рейтинг компаній. Інвестори сприймають кіберкомпрометацію як показник слабкої системи безпеки, що знижує довіру до менеджменту та довгострокових прогнозів.

Успішна фішингова атака, що призвела до витоку персональних або фінансових даних, майже завжди супроводжується штрафами. Відповідно до законодавства (зокрема, GDPR у ЄС), штрафи можуть досягати до 4 % від річного обороту компанії. Реальні приклади – штрафи British Airways (£20 млн) та Marriott Hotels (£18,4 млн) за недбале зберігання даних [7].

Ураховуючи масштаби збитків, виявлення фішингових вебсайтів стає стратегічним напрямом кіберзахисту. Автоматизовані системи виявлення фішингових вебсайтів, побудовані на аналізі поведінкових ознак, метаданих доменів та контенту сторінок, дозволяють знизити ймовірність інцидентів ще до того, як користувач взаємодіє з шахрайським ресурсом.

1.2. Аналіз існуючих технічних рішень і баз загроз для виявлення фішингу

1.2.1. VirusTotal

VirusTotal – це одна з найвідоміших платформ для аналізу потенційно шкідливих URL-адрес, файлів і доменів, яка об'єднує результати понад 70 антивірусних механізмів і вебсканерів [8]. Її принцип роботи ґрунтується на багаторівневому аналізі даних та колективному внеску користувачів, що робить її важливим елементом у системах виявлення фішингових вебсайтів.

VirusTotal отримує об'єкт для перевірки (URL, домен або файл) через: вебінтерфейс; браузерні розширення; або програмний API, що дозволяє інтегрувати сервіс у зовнішні системи моніторингу. Після надсилання посилання або файлу

система виконує паралельний аналіз у десятках антивірусних рушіїв і фішинг-баз, таких як Google Safe Browsing, Kaspersky, ESET, Avira тощо. Кожен рушій формує власний детекційний ярлик, наприклад phishing.site або malware.url, а результати агрегуються в єдиному звіті (рис. 1.1).

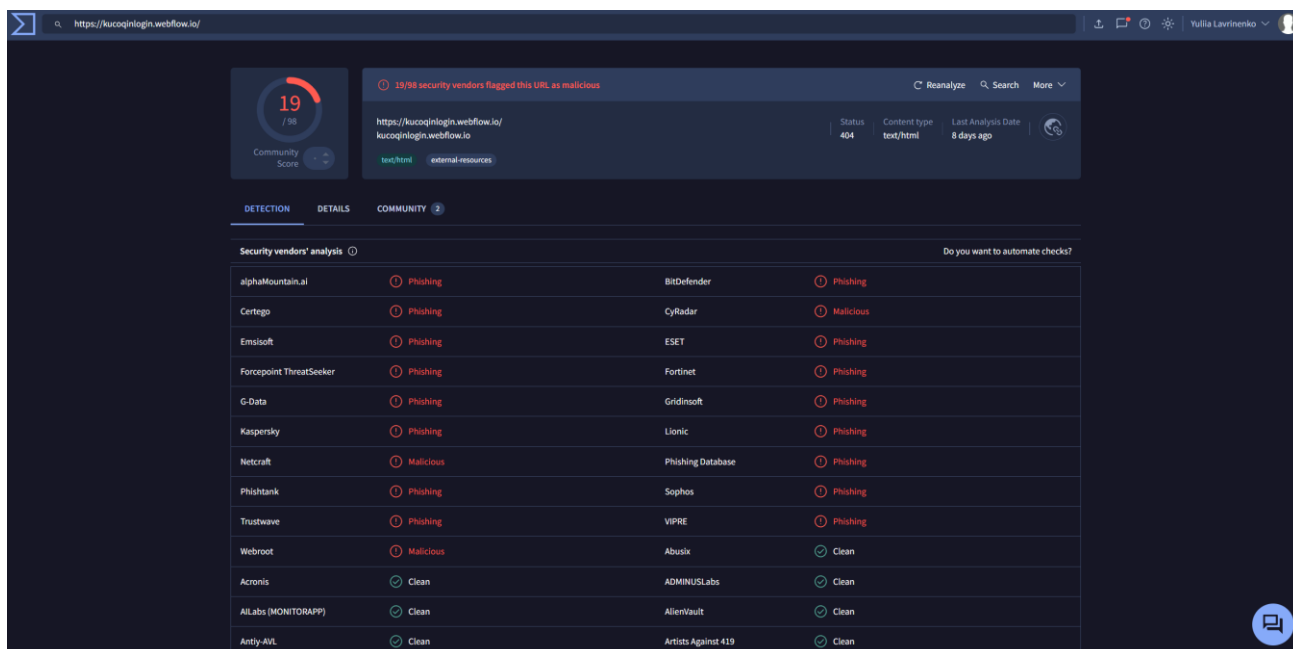


Рис. 1.1. Результат перевірки URL у VirusTotal

Для виявлення фішингу VirusTotal є цінним тим, що: надає чітку класифікацію типу загрози, розділяючи «phishing», «malware», «suspicious»; деякі рушії додають додаткові описи, зокрема який бренд імітує сайт, або до якого ботнету належить сторінка; оновлює сигнатури в режимі реального часу, тому результати відображають актуальні статуси блокувань [8]. VirusTotal може миттєво показати, чи внесено сайт до чорних списків антивірусів, що робить його незамінним елементом у багаторівневих системах фішинг-детекції.

VirusTotal підтримує спільноту користувачів (VirusTotal Community), де аналітики можуть залишати коментарі до доменів або сайтів, позначати підозрілі об'єкти, а також ділитися висновками про хибні спрацьовування. Це створює ефект колективного навчання, що підвищує точність класифікації [8].

Для розробників передбачено VirusTotal API, який дозволяє автоматично перевіряти домени чи URL-адреси без використання вебінтерфейсу (рис. 1.2).

Через API можна: виконувати масові запити перевірки; отримувати структурований JSON-відповідь із результатами; інтегрувати систему з власними модулями.

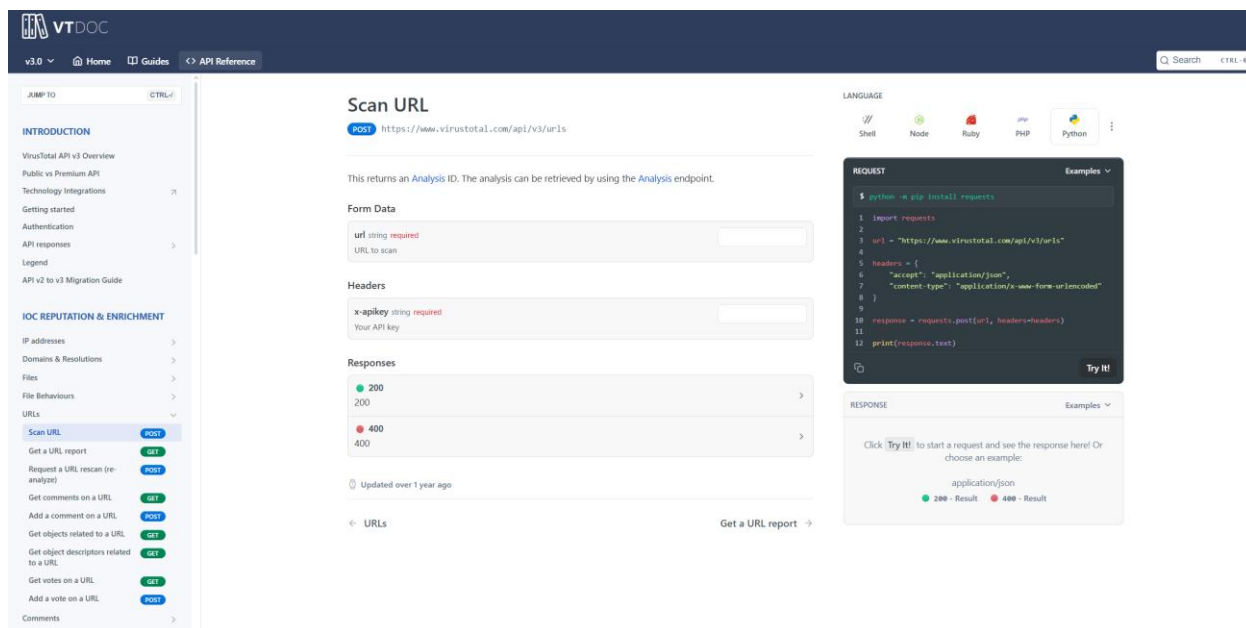


Рис. 1.2. Сторінка документації VirusTotal API, перевірка URL

Таким чином, VirusTotal виконує роль агрегатора інформації про загрози, поєднуючи результати десятків незалежних антивірусних рушіїв і баз даних фішингових сайтів. Його використання в системах автоматичного аналізу дозволяє: підтверджувати наявність фішингової активності на вебсайті; ідентифікувати джерело загрози через міжплатформні сигнатури; зменшити кількість хибнопозитивних результатів завдяки спільноті та узгодженим даним. VirusTotal – це не просто база даних шкідливих URL, а глобальна система колективного аналізу, що підвищує загальний рівень кіберзахисту і забезпечує надійну перевірку вебресурсів у рамках систем детекції фішингу.

VirusTotal – корисний агрегатор, який перевіряє URL і домени десятками антивірусних рушіїв і дає швидкий «зріз» репутації, але він по суті пасивний і залежить від того, чи внесли сайт у чужі бази. Якщо фішинговий сайт новий або працює через динамічні тригери (редиректи після кліку, форми, що з'являються тільки після взаємодії), VirusTotal це може не помітити; він не симулює поведінку користувача і не аналізує, як сторінка реагує на дії. Також сервіс дає агрегований статус від інших рушіїв, тож затримка між появою загрози й її фіксацією можлива.

1.2.2. Google Web Risk та Safe Browsing

Web Risk та Safe Browsing – сервіси, що належать Google і призначені для перевірки URL-адрес чи вебсайтів на наявність загроз: фішингу, шкідливого ПЗ, небажаного програмного забезпечення. Safe Browsing захищає понад 5 мільярдів пристроїв щодня, показуючи попередження, коли користувач переходить на небезпечний сайт [9]. Web Risk описується як «потужна Cloud служба», яка використовує величезну мережу інтелектуальної інформації Google: сканує понад 10 мільярдів URL і файлів щодня, виявляючи фішинг-сайти, завантаження шкідливого ПЗ та інше [10]. Таким чином, обидва рішення мають спільну мету – виявляти та блокувати фішингові вебресурси.

Хоча цілі у них схожі, між Web Risk і Safe Browsing є ключові відмінності. Safe Browsing спочатку створювався для блокування при навігації користувача (наприклад, у браузері) через списки небезпечних сайтів. Web Risk позиціонується як підприємницьке рішення, яке дає API для перевірки URL і інтеграції у платформу, з акцентом на брендовий захист, масове сканування й API-інтеграцію. Вигляд деталей підключеного API-сервісу наведено на рис. 1.3.

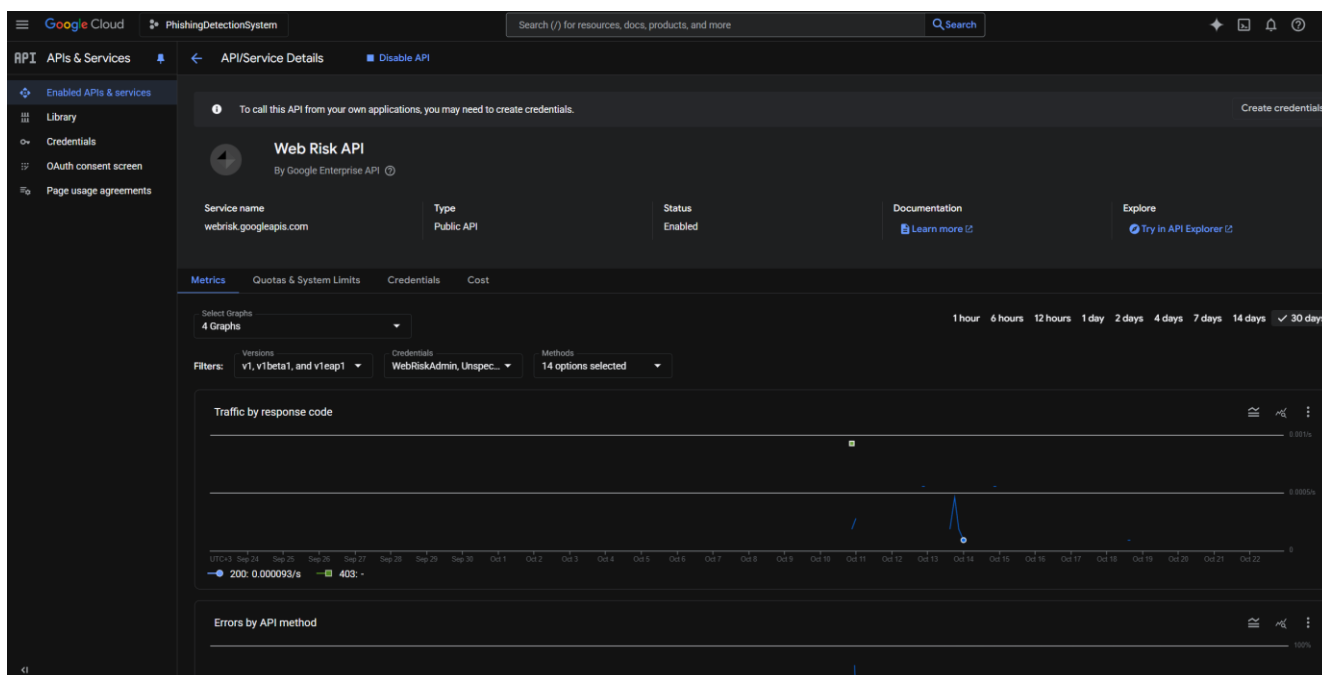


Рис. 1.3. Вигляд деталей Web Risk API у кабінеті користувача

Safe Browsing API має обмеження для комерційного використання – призначений переважно для некомерційного застосування. Web Risk не має такого обмеження і призначений саме для інтеграцій у бізнес-середовищі. Web Risk акцентує увагу на масштабному скануванні «10+ млрд URL/файлів щодня». Safe Browsing більше фокусується на браузерній навігації, захисті користувачів та інтеграції у браузери, пошукові системи, Gmail.

Попри потужність обох систем, вони мають обмеження, які особливо важливі при фішингу вебсайтів. Обидва рішення в значній мірі залежні від наявності сайту в існуючій базі загроз. Якщо фішинговий сайт новий або створився буквально за лічені години, він може не бути внесеним і залишитися непоміченим. Вони не емулюють користувацьку взаємодію зі сторінкою (натискання кнопок, введення даних, редиректи після кліку) і тому можуть пропустити фішингові сценарії, які активуються лише після взаємодії. Час між появою нової фішингової сторінки і її класифікацією може бути суттєвим, що дає зловмиснику можливість діяти до того, як сайт буде заблоковано.

Обидві служби орієнтовані на блокування або сигналізацію, але не дають глибоку аналітику про структуру сайту, поведінку клієнта, динамічні домени, форми, скрипти – а це критично при детекції вебфішингу.

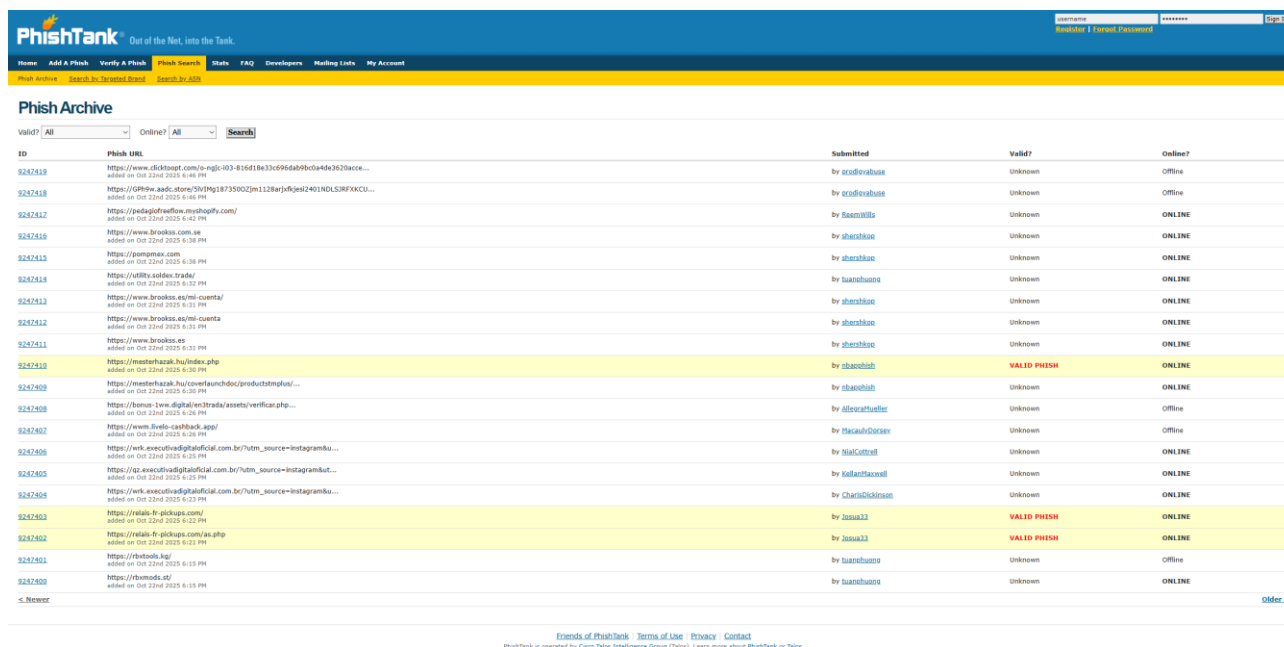
Отже, хоча Web Risk і Safe Browsing – потужні інструменти, для комплексного захисту від фішингу вебсайтів бажано доповнювати їх рішеннями, які аналізують поведінку сайту в реальному часі, динамічні елементи і контент-аналіз.

1.2.3. Спільні відкриті репозиторії та сервіси

У відкритому просторі є кілька сервісів і репозиторіїв, які збирають і публікують списки фішингових URL – саме ті дані, які часто використовують для первинної фільтрації та підживлення чорних списків у системах безпеки. Надалі розглянуто три найбільш відомі з них: PhishTank, OpenPhish і URLhaus – кожен має свою специфіку й випадки застосування.

PhishTank – це платформа, куди користувачі надсилають підозрілі URL, а спільнота їх валідує; дані доступні через API і використовуються багатьма

інструментами для швидкої перевірки посилань. Для досліджень корисно, що PhishTank містить історію подачі та статус валідації (рис. 1.4), але іноді дані можуть запізнюватись або підлягати спаму через людський фактор, через це доступ до реєстрації нових користувачів на даний момент заблоковано [11].



The screenshot shows the PhishTank Phish Archive interface. At the top, there's a navigation bar with links like Home, Add A Phish, Verify A Phish, Phish Search, Status, FAQ, Downloaders, Mailing Lists, and My Account. Below this is a search bar with filters for Valid? (All), Online? (All), and a Search button. The main content is a table listing various phishing URLs with columns for ID, Phish URL, Submitted, Valid?, and Online?.

ID	Phish URL	Submitted	Valid?	Online?
9247419	https://www.dickbapt.com/0-rgj-03-816d1ba33c996ab9b0e4de3629acc...	by xaddevabuse	Unknown	Offline
9247418	https://GPhiv.aadl-store/5V1q18735002jm1128arj0je2401NOL5RFXCC...	by xaddevabuse	Unknown	Offline
9247417	https://pedagofreeflow.myshopify.com/	by Beemills	Unknown	ONLINE
9247416	https://www.brooks.com.ua	by aharabkue	Unknown	ONLINE
9247415	https://pompmex.com	by aharabkue	Unknown	ONLINE
9247414	https://villitv.soldex.trade/	by aharabkue	Unknown	ONLINE
9247413	https://www.brooks.com.es/mi-cuenta/	by aharabkue	Unknown	ONLINE
9247412	https://www.brooks.es/mi-cuenta	by aharabkue	Unknown	ONLINE
9247411	https://www.brooks.es	by aharabkue	Unknown	ONLINE
9247410	https://mestorhazak.hu/index.php	by aharabkue	VALID PHISH	ONLINE
9247409	https://mestorhazak.hu/coverlaunchdoc/productstmp/...	by aharabkue	Unknown	ONLINE
9247408	https://benin-1wv.digitaltechnetradet/verificac.php...	by aharabkue	Unknown	Offline
9247407	https://www.bveto-cashback.app/	by aharabkue	Unknown	Offline
9247406	https://www.bveto-cashback.app/	by aharabkue	Unknown	ONLINE
9247405	https://www.bveto-cashback.app/	by aharabkue	Unknown	ONLINE
9247404	https://www.bveto-cashback.app/	by aharabkue	Unknown	ONLINE
9247403	https://www.bveto-cashback.app/	by aharabkue	VALID PHISH	ONLINE
9247402	https://www.bveto-cashback.app/	by aharabkue	VALID PHISH	ONLINE
9247401	https://www.bveto-cashback.app/	by aharabkue	Unknown	Offline
9247400	https://www.bveto-cashback.app/	by aharabkue	Unknown	ONLINE

Рис. 1.4. Сторінка з останніми запитами на перевірку сайтів та їх статусами

OpenPhish – сервіс, який автоматично збирає живі фішингові посилання з інтернету і надає їх у вигляді фідів. На відміну від краудсорсингових платформ, OpenPhish робить ставку на автоматизацію: він постійно сканує мережу, визначає активні фішингові сторінки й класифікує їх за базовими ознаками (наприклад, чи імітує сторінка певний банк або сервіс). Є безкоштовний «community» фід, який підходить для базової автоматичної перевірки посилань, і є комерційні фіди з частішими оновленнями та додатковими метаданими (перший час появи, ймовірна цільова організація, іноді мова чи регіон). Для задач виявлення веб-фішингу OpenPhish корисний тим, що дає «живі» рядки URL, які можна швидко підставити в блокувальник або перевірити додатковими інструментами [12].

URLhaus – відкритий репозиторій, який фіксує URL, помічені у розповсюдженні шкідливого ПЗ, але в базі також трапляються й фішингові записи. Головна мета URLhaus це зібрати технічні адреси, які реально використовуються

для злому або розповсюдження малварі, разом із простою технічною інформацією: коли помічено, який IP/хост їх обслуговує, та чи було вже завантаження шкідливого контенту. Для досліджень веб-фішингу URLhaus корисний, коли потрібно відстежити технічні деталі посилань, наприклад, повторні появи одного й того ж хосту з різними фішинговими сторінками. Це дає змогу помітити шаблонну інфраструктуру фішерів [13].

Варто зазначити, у багатьох системах агрегування загроз (зокрема в тих, що аналізувалися раніше, наприклад VirusTotal) результати з OpenPhish, PhishTank, URLhaus та інших відкритих фідів часто використовуються як частина загальної оцінки URL. Тобто ці репозиторії самі по собі – не всі інструменти, але вони є важливим джерелом «сирих» індикаторів, які потім доповнюють поведінкові та контент-аналізи при виявленні фішингових сайтів.

Усі ці джерела дають оперативні списки URL, які легко інтегрувати в блокувальники, поштові фільтри або власні конвейери детекції. Вони добре працюють як перший рівень фільтрації (blacklist/allowlist), як джерело «підозрілих екземплярів» для навчання моделей або як індикатори для автоматичних правил. Проте, мають спільні обмеження: немає повноти «поведінкових» сигналів: репозиторії дають URL і метадані, але не показують, що відбувається після кліку (редиректи, динамічні форми, JS-тригери); нові фішингові сторінки часто живуть хвилини-години – поки репозиторій збере й валідує запис, атака може пройти; мало інформації про візуальну схожість зі жертвою-брендом або семантику тексту на сторінці, що важливо для веб-фішингу.

1.2.4. Браузерні захисти та комерційні рішення

Браузери, такі як Google Chrome і Microsoft Edge, реалізують вбудовані механізми захисту від фішингових сайтів, засновані на перевірці репутації URL.

У Chrome цей механізм працює через раніше згаданий сервіс Google Safe Browsing, який щодня аналізує мільярди адрес і файлів, формуючи списки потенційно небезпечних ресурсів. Під час відкриття сторінки браузер локально порівнює URL із цими списками. Якщо збіг знайдено, користувачу показується

характерне червоне попередження із пропозицією повернутись на безпечну сторінку (приклад наведено на рис. 1.5).

У режимі Enhanced Protection Chrome додатково надсилає анонімізовані дані в хмару Google для машинного аналізу – це дозволяє виявляти нові фішингові сторінки, які ще не потрапили до репутаційних баз. Google підкреслює, що ця система використовує моделі штучного інтелекту для визначення фішингових шаблонів у реальному часі, завдяки чому ризикові сторінки блокуються ще до того, як користувач взаємодіє з формами введення [14].

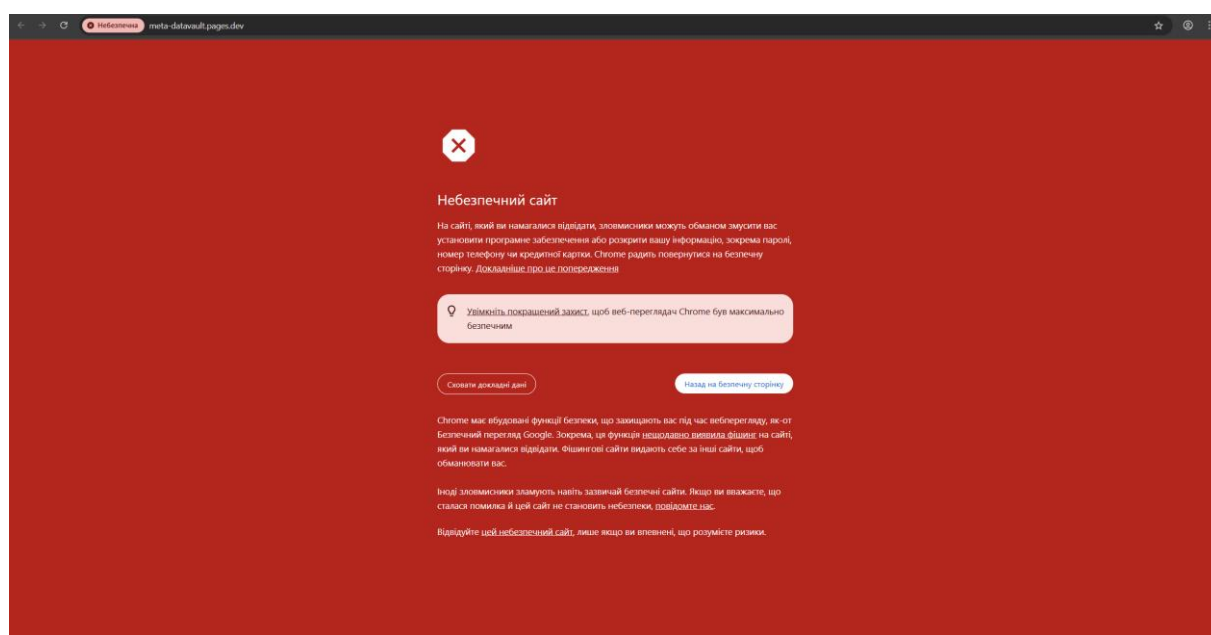


Рис. 1.5. Небезпечна сторінка у браузері Chrome

Microsoft Edge використовує технологію Microsoft Defender SmartScreen, що перевіряє як веб-адреси, так і завантажувані файли. Система порівнює URL із централізованим списком відомих фішингових сайтів і динамічними моделями поведінки. При виявленні підозрілої сторінки відображається попередження (рис. 1.6). У Windows 11 з'явилася функція Enhanced Phishing Protection, яка реагує, якщо користувач вводить корпоративний пароль на сумнівному сайті або повторно використовує його на кількох ресурсах [15].

Такі браузерні захисти є першим рубежем оборони від веб-фішингу, адже вони попереджають користувача ще до взаємодії зі сторінкою. Проте їх ефективність залежить від актуальності баз даних: нові фішингові сайти можуть

залишатися активними, поки не потраплять до списків, а користувач завжди має змогу ігнорувати попередження.

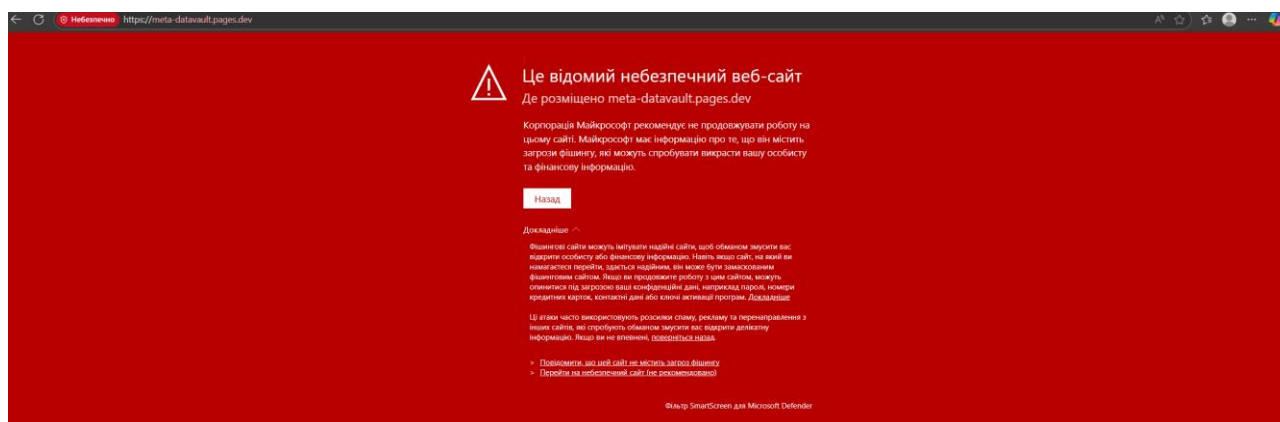


Рис. 1.6. Попередження про небезпечний сайт у Microsoft Edge

Антивірусні пакети та розширення, такі як Norton Safe Web (входить до складу Norton 360) і Bitdefender Anti-Phishing (частина Bitdefender Total Security), поєднують кілька технологій для захисту від веб-фішингу. Norton Safe Web аналізує сайти, на які збирається перейти користувач, на наявність шкідливого ПЗ, фішингових форм чи скриптів; він надає рейтинги безпеки сайтам ще до переходу на них (рис. 1.7), допомагаючи користувачу ухилитися від небезпечного ресурсу [16]. Bitdefender вказує, що його технологія анти-фішингу спирається на машинне навчання і поведінковий аналіз – вона сканує веб-трафік, HTML/JavaScript код, шаблони фішингових сторінок та блокує веб-посилання, які ведуть на шахрайські ресурси [17].

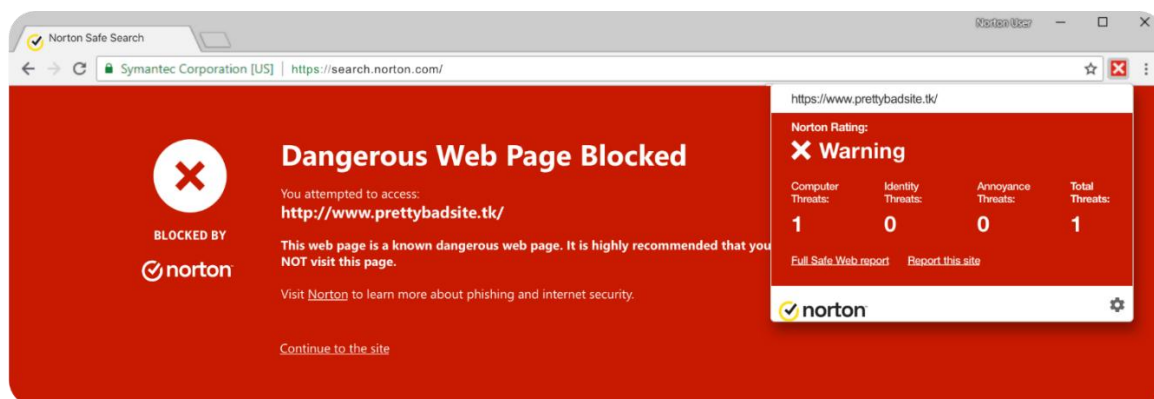


Рис. 1.7. Приклад інтерфейсу Norton Safe Web, на якому сайт позначено як небезпечний

Корпоративні шлюзи безпеки, як-от Cisco Umbrella, Zscaler Internet Access або Cloudflare Gateway, працюють інакше: вони фільтрують весь веб-трафік організації – спочатку на рівні DNS, потім аналізують HTTPS-запити через проксі, виконують SSL-інспекцію, перевіряють URL і контент сторінки та можуть запускати сумнівні ресурси в ізольованому браузері або режимі перегляду без можливості введення даних користувачем. Наприклад, Cloudflare Gateway дозволяє налаштувати політику «Browser Isolation» для категорій сайтів типу «typosquatting»: користувач бачить сторінку, але не може вводити свої облікові дані, таким чином мінімізуючи ризик фішингу [18]. Схема роботи зображена на рис. 1.8.

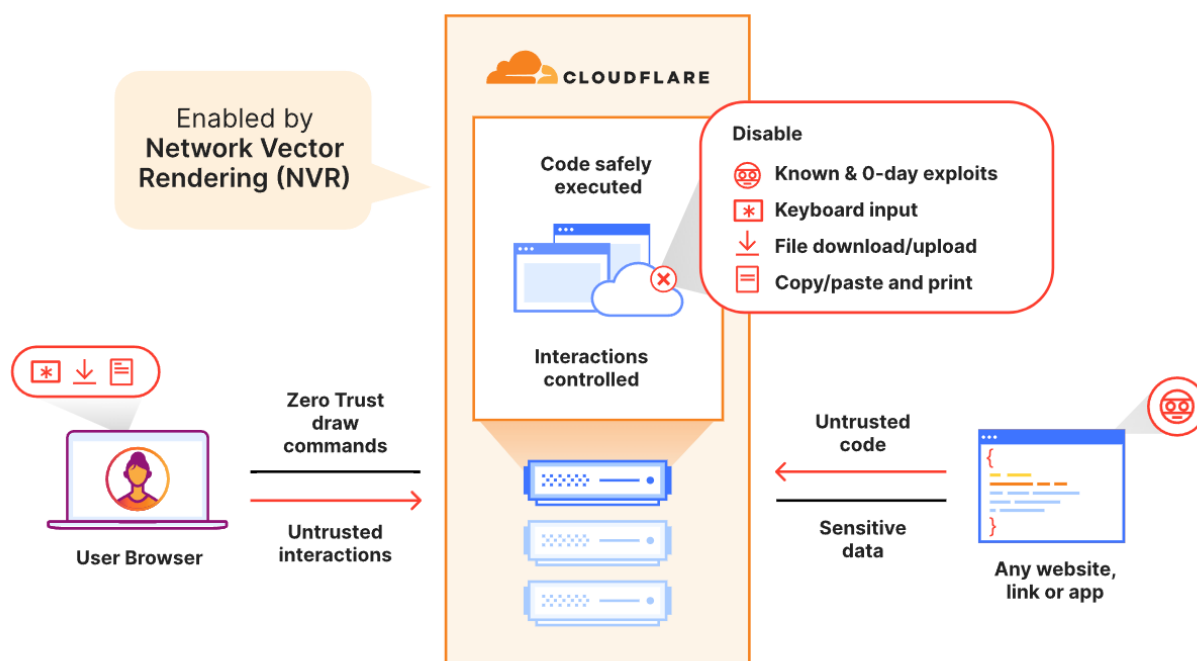


Рис. 1.8. Схема роботи політики «Browser Isolation» у Cloudflare Gateway

Таким чином, комерційні рішення й шлюзи забезпечують значно ширший захист порівняно з базовими браузерними фільтрами: вони можуть перевіряти ще до того, як користувач щось введе, використовувати машинне навчання, аналізувати контент сторінки, працювати корпоративно. Одночасно вони залежать від регулярного оновлення баз загроз і налаштувань (наприклад, політик ізоляції чи проксі) – без них захист може бути менш ефективним. Навіть коли використовується машинне навчання, аналіз відбувається на рівні мережеских

запитів або статичного контенту сторінки, без моделювання реальної взаємодії користувача.

1.2.5. Висновки до аналізу існуючих технічних рішень і баз загроз

Розглянуті рішення – VirusTotal, Google Web Risk, PhishTank, OpenPhish, URLhaus, а також браузерні та корпоративні системи захисту – формують основу сучасної інфраструктури репутаційного аналізу й блокування фішингових сайтів. Кожне з них охоплює певний етап життєвого циклу загрози: від первинного виявлення та додавання до бази до відображення попередження користувачу під час відкриття сторінки. Проте спільною рисою всіх підходів залишається пасивний характер аналізу – перевірка здійснюється лише щодо вже відомих індикаторів, без глибокої поведінкової оцінки. У табл. 1.1 наведений аналіз цих рішень.

Таблиця 1.1

Порівняльна характеристика технологічних рішень

Рішення / тип системи	Основна функція	Переваги	Обмеження	Рівень аналізу
VirusTotal	Агрегує результати десятків антивірусів та вебсканерів	Масштабна база, API-інтеграція	Пасивний, без поведінкової оцінки	Репутаційний
Google Web Risk / Safe Browsing	Перевірка URL на наявність у базах загроз	Висока актуальність, швидкість	Не аналізує контент, залежить від бази	Репутаційно-статичний
PhishTank / OpenPhish / URLhaus	Публічні фіди фішингових адрес	Відкриті дані, швидке оновлення	Відсутність контенту, людський фактор	Репутаційний

Продовження таблиці 1.1

Рішення / тип системи	Основна функція	Переваги	Обмеження	Рівень аналізу
Браузерні механізми	Попередження користувача під час навігації	Вбудованість, миттєва реакція	Може бути обійдений, залежить від списків	Клієнтський рівень
Корпоративні шлюзи (Cisco, Cloudflare, Zscaler)	Централізований захист трафіку	Повна інтеграція, політики ізоляції	Висока вартість, обмежений контент-аналіз	Мережевий, статичний

Попри різноманіття технологічних рішень, більшість із них працює реактивно, тобто лише після фіксації загрози в базах. Сучасний веб-фішинг натомість активно використовує динамічні сценарії: редиректи після кліку, елементи, що з'являються лише під час взаємодії, та контент, який адаптується під користувача.

Отже, існує суттєвий розрив між статичними системами перевірки репутації та сучасними динамічними фішинговими атаками. Це зумовлює потребу в комплексній системі, яка аналізує контент сторінки на рівні текстових і візуальних елементів; моделює поведінку користувача для виявлення прихованих форм і редиректів; інтегрує дані з відкритих баз загроз у єдину метрику ризику.

1.3. Аналіз методологічних підходів до детекції фішингу

1.3.1. Перевірка по базах даних

Перевірка по базах даних – це один із найпоширеніших методів виявлення фішингу на веб-сайтах; суть його полягає в швидкому зіставленні перевірюваної URL-адреси або домену з репутаційними списками відомих загроз. У практиці веб-

захисту такі репозиторії можуть містити як вручну верифіковані записи, так і результати масових автоматизованих сканувань: при зверненні до служби клієнтський додаток отримує вердикт для конкретної адреси або хеш-значення, що дає змогу показати користувачеві попередження або заблокувати доступ до ресурсу ще до взаємодії з формами введення паролів чи іншими елементами сторінки. Цей підхід широко застосовується як у браузерних рішеннях (через централізовані списки «небезпечних» URL), так і в сервісах-агрегаторах, які консолідують результати десятків антивірусних рушіїв і URL-сканерів для оперативного винесення вердикту.

Методологічно перевірка по базах даних дає явні переваги в продуктивності й простоті інтеграції: зіставлення рядка URL або хешу виконується дуже швидко, що дозволяє мінімізувати затримки при навігації, а централізовані оновлення забезпечують широку видимість відомих загроз. Однак цей підхід має і очевидні обмеження: він ефективний переважно щодо вже відомих або швидко подібних загроз і значно слабший проти короткоживучих доменів, динамічних тригерів (коли фішингова форма з'являється тільки після певної взаємодії) або технік, що маскують адресу. Окрім того, точність детекції залежить від частоти оновлення репозиторію та від джерела даних; агрегатори, які використовують внески спільноти або численні двигуни, можуть давати швидку оцінку, але іноді потребують додаткової валідації для зниження хибних спрацювань [8].

У контексті виявлення фішингу на веб-сайтах перевірка по базах даних найкраще працює в поєднанні з іншими підходами: вона оперативно відсіює вже відомі ризики та підживлює системи колективного інтелекту, але для виявлення «нових» фішингових сценаріїв (які активізуються лише після кліку або імітації користувацької сесії) її слід доповнювати методами поведінкового аналізу та емульованого виконання сторінки.

1.3.2. Статичний аналіз контенту сторінки

Статичний аналіз контенту сторінки полягає у витяганні ознак із самого HTML-документа, DOM-структури й текстових елементів без виконання скриптів

чи імітації користувацьких дій. Зі сторінки витягують адреси посилань, атрибути форм, заголовки, метадані й фрагменти тексту, після чого ці ознаки використовують для побудови моделей класифікації; такий підхід дозволяє швидко опрацьовувати великі масиви сторінок і служить першим фільтром у ланцюгу детекції.

У рамках цієї методики можна застосовувати й прості мовні моделі: вони аналізують текст сторінки на предмет соціальної інженерії – звернення до термінів типу «підтвердження», «терміново», «оновіть платіжні дані», наявність логотипів або імен брендів у контексті, який типово зустрічається у фішингових повідомленнях [19]. Мовні моделі добре працюють саме з текстовою складовою й допомагають відрізнити природний, інформаційний текст від підозрілих закликів до дії, тому їх легко інтегрувати як додатковий маркер у загальну систему статичної перевірки.

Перевага статичного підходу в його швидкості та детермінованості, але його межі очевидні: сторінки, які ховають шкідливу логіку за динамічними скриптами або показують фішингові форми тільки після взаємодії, залишаються поза полем зору статичного аналізу. Тому на практиці статичний аналіз із мовними моделями ефективно працює як попередній фільтр – він оперативно відсіює явні випадки й формує ознаки для подальшого глибшого дослідження з виконанням сторінки та поведінковим аналізом.

1.3.3. Поведінковий аналіз

Поведінковий аналіз у контексті виявлення фішингу на веб-сайтах фокусується на динамічній поведінці сторінки під час взаємодії, а не лише на її статичному вмісті. Сторінка завантажується в ізольованому середовищі, виконуються типові дії користувача – натискання кнопок, перехід по посиланнях, заповнення полів – і фіксуються зміни в документі, поява форм, редиректи та інші події, які можуть свідчити про намір вкрати облікові дані або інші дані. Такий підхід дозволяє виявляти ті сценарії, коли шкідлива логіка активується лише після певної послідовності дій і тому залишається непоміченою при простому перегляді [20].

Методологічно поведінковий аналіз видобуває часові ознаки: коли саме з'явилися нові елементи, які події їх викликали, чи з'являються форми після перевірки на ботів, куди відправляються дані форми і чи відбувається перенаправлення на інші домени. Ці ознаки дають значно більше інформації про «наміри» сторінки, ніж суто лексичні чи структурні маркери, і дозволяють моделям точніше розрізняти легітимні сайти від підготовлених пасток.

Поведінковий підхід має свої обмеження, насамперед пов'язані з ресурсною складністю: імітація взаємодії і збереження часових зрізів потребують більше обчислювальних потужностей та часу на аналіз, тому його часто застосовують вибірково – тільки до тих сторінок, які пройшли початковий статичний фільтр або мали підозрілі результати. Окрім цього, зловмисники можуть використовувати методи виявлення автоматичних агентів і показувати шкідливий контент лише для живих відвідувачів, що потребує від системи розумних стратегій емуляції та випадкових пауз при відтворенні дій. Попри це, у практиці поведінковий аналіз добре поєднується з іншими методами і часто слугує вирішальним етапом для виявлення «післяклікових» атак.

З методичної точки зору важливо, що поведінкові ознаки дозволяють не тільки класифікувати сторінку, а й інтерпретувати її наміри – наприклад, виділити ознаки збору облікових даних або фінансового шахрайства, що корисно для пріоритизації інцидентів і подальшого реагування. Тому сучасні наукові підходи прагнуть комбінувати візуальний аналіз, текстові моделі й динамічні спостереження, щоб отримати більш повну «картину» поведінки веб-сторінки і підвищити пояснюваність рішень. Ця комбінація дозволяє створювати інструменти, які не лише виявляють фішинг, але й пояснюють, які саме ознаки вказують на загрозу.

1.3.4. ML/LLM-підходи до аналізу контенту та візуальної подібності

Підхід, що базується на методах машинного навчання та великих мовних моделях, поєднує аналіз текстового вмісту сторінки з оцінкою візуальної подібності до відомих брендів чи шаблонів. Текстова частина використовує моделі для

розпізнавання ознак соціальної інженерії: неприродна лексика, заклики до термінових дій, підміна назв брендів у контексті. Великі мовні моделі здатні працювати з повноцінними текстовими описами сторінки, виділяти семантичні маркери підозрілої поведінки та надавати пояснення своєї оцінки у вигляді короткого резюме підстав підозри [21]. Це підвищує чутливість класифікації до складних текстових маніпуляцій, які важко виявити простими регулярними виразами або правилами.

Візуальна складова полягає у порівнянні образу сторінки з еталонними зразками: аналізують скріншот сторінки, порівнюють розміщення логотипів, кольорові палітри та розмітку елементів, а також застосовують методи пошуку подібних зображень і хешування [22]. Комп'ютерний зір дозволяє відокремити «дуже схожі» підроби від поверхневих наслідувань і тим самим фокусувати розслідування на випадках із високим ризиком. Такий підхід корисний саме для веб-фішингу, де значна частина атаки – візуальна підміна інтерфейсу відомих сервісів.

Методологічно продуктивним виявляється поєднання двох напрямів: мовної оцінки і візуальної подібності. Спочатку швидкий текстовий аналіз дає початковий відсоток підозрілості, після чого система автоматично будує візуальну репрезентацію сторінки і порівнює її з базою еталонів. Якщо обидва шляхи сигналізують про ризик, сторінка відправляється на глибший аналіз або ізоляцію. Поєднання зменшує відсоток хибних тривог і дозволяє виявляти ті приклади, де текст виглядає «чистим», а головна пастка – у макеті та графіці.

Практичні обмеження таких систем пов'язані з обсягом і якістю навчальних даних, швидкістю підробок та мультинаціональністю контенту. Великі мовні моделі потребують значних обчислювальних ресурсів і ретельного добору підготовчого корпусу, інакше вони можуть або недооцінювати хитрі маніпуляції, або давати занадто багато помилкових сигналів. Аналогічно, візуальна база еталонів має покривати різні варіанти брендингу й локалізації, інакше нові підроби пройдуть непоміченими. Крім того, зловмисники можуть навмисно

змінювати дизайн у дрібних деталях, щоб звести нанівець прості метрики візуальної схожості.

1.3.5. Гібридні стратегії

Гібридна стратегія поєднує кілька підходів до виявлення фішингу і прагне компенсувати слабкі місця кожного з них через їхню кооперацію. У контексті веб-фішингу це означає, що система одночасно використовує швидкий статичний аналіз вмісту сторінки, евристичні та модельні оцінки тексту, перевірки по базах репутацій і, за потреби, візуальне порівняння скріншоту з еталонами. Такий підхід дозволяє оперативно відсікати явні підробки на ранньому етапі й направляти складніші або двозначні випадки на глибшу перевірку, де застосовують більш важкі алгоритми або людський аналіз.

У робочому потоці гібридної системи статичний шар здобуває базові ознаки з HTML і тексту та швидко перевіряє URL і доменні репутації; якщо сумарний відсоток підозрілості перевищує поріг, система паралельно запускає модулі глибшого аналізу – машинне навчання для семантичної оцінки тексту і модулі візуальної подібності для скріншоту сторінки. Такий послідовно-паралельний режим дозволяє зберегти швидкість у звичайних випадках і одночасно підвищити точність для витончених підробок, де загальна підозра ховається в оформленні інтерфейсу або в нетиповому вживанні брендових назв.

Важливим елементом гібридної стратегії є інтеграція компоненту, що працює з користувачем: короткі підказки або чек-лісти (наприклад, перевірити відправника, посилання і зміст повідомлення) знижують ймовірність успішної атаки навіть у випадках, коли автоматичний фільтр пропускає підробку. Такий людо-технічний баланс дозволяє значно зменшити наслідки атак на рівні кінцевого користувача і одночасно знижує навантаження на автоматичні механізми, бо багато очевидних випадків блокуються ще до глибокої перевірки.

Серед обмежень гібридної стратегії – необхідність оркестрації різномірних модулів, яка ускладнює архітектуру системи та вимагає тонкого налаштування порогів і політик перенаправлення на глибоку перевірку. При недостатньому

налаштуванні система або буде давати забагато хибних тривог, або навпаки пропускати складні підробки. Окрім цього, гібридний підхід потребує постійного оновлення репутаційних баз та колекції візуальних еталонів, а також періодичного донавчання моделей на реальних прикладах веб-фішингу, локалізованих під регіон і мову.

1.3.6. Висновки до аналізу методологічних підходів

Проведений огляд методологічних підходів до виявлення фішингу засвідчив, що кожен із них має власну нішу ефективності та обмеження. Найпростіші методи, як-от перевірка по базах даних, забезпечують швидку реакцію, але не виявляють нові або динамічні загрози. Статичний аналіз дозволяє розпізнати структурні та текстові ознаки фішингу, проте не бачить поведінку сторінки. Поведінкові підходи забезпечують найглибший рівень спостереження, але потребують значних обчислювальних ресурсів і спеціального середовища. Методи машинного навчання та мовних моделей підвищують точність, дозволяючи виявляти приховані семантичні або візуальні закономірності, тоді як гібридні стратегії комбінують ці принципи в єдиний процес. Порівняльний аналіз наведений у табл. 1.2.

Таблиця 1.2

Порівняльна характеристика методологічних підходів

Підхід	Тип аналізу	Переваги	Обмеження	Рівень ефективності
Перевірка по базах даних	Репутаційний, сигнатурний	Висока швидкість, легкість інтеграції	Не виявляє нові загрози, залежить від бази	Середній
Статичний аналіз контенту	Структурний, текстовий	Швидкий, добре виявляє типові шаблони	Ігнорує динамічні елементи	Високий

Продовження таблиці 1.2

Підхід	Тип аналізу	Переваги	Обмеження	Рівень ефективності
ML/LLM-підходи	Семантичний, візуальний	Глибоке розуміння тексту й дизайну, адаптивність	Залежність від навчальних даних, ресурсоемність	Високий
Гібридні стратегії	Комбінований	Баланс точності й швидкодії	Складна оркестрація модулів	Найвищий

Узагальнюючи результати аналізу, можна зробити висновок, що найвищу ефективність у реальних умовах демонструє гібридна методологія, яка поєднує швидкодію репутаційних перевірок, аналітичну глибину статичного аналізу та точність поведінкових і модельних оцінок. Така багаторівнева структура забезпечує оптимальний баланс між швидкістю, повнотою та достовірністю результатів.

Поєднання методів дозволяє виявляти не лише відомі фішингові ресурси, але й нові, ще не внесені до баз, що робить систему придатною для реального корпоративного застосування. Гібридна архітектура забезпечує гнучкість і масштабованість: кожен модуль працює автономно, але їхні результати об'єднуються в єдину метрику ризику, що формує підґрунтя для подальшої розробки системи автоматичного виявлення фішингових вебсайтів.

1.4. Формалізація задачі та постановка часткових задач досліджень

У межах даної роботи ставиться завдання розробити систему автоматичного виявлення фішингових веб-сайтів, що поєднує аналітику відкритих баз загроз із динамічним та поведінковим аналізом контенту сторінки. Актуальність задачі визначається тим, що більшість існуючих рішень, як-от VirusTotal, Google Web Risk, PhishTank чи OpenPhish, ґрунтуються переважно на статичних базах даних, які

мають обмежену здатність реагувати на нові, ще не верифіковані фішингові кампанії. При цьому сучасні фішингові сайти використовують динамічні механізми: JavaScript-редиректи, приховані форми, адаптивні скрипти, які спрацьовують лише після взаємодії користувача. Тому метою дослідження є створення комплексного підходу, який поєднує переваги обох напрямів: швидкість репутаційних перевірок і глибину поведінкового аналізу.

Основна науково-прикладна задача полягає у формалізації процесу автоматизованого аналізу вебресурсів за комбінованими ознаками: ідентифікаційними; структурно-контентними; поведінковими. Кожен тип ознаки відображає певний рівень аналізу, а їх агрегація дозволяє отримати комплексний індикатор ризику, який підвищує ймовірність виявлення фішингового сайту навіть у разі відсутності його в базах даних.

Для реалізації поставленої мети доцільно розділити дослідження на кілька часткових задач:

- сформулювати узагальнену модель виявлення фішингових вебсайтів, що поєднує результати з баз відкритих загроз, аналіз контенту сторінки та поведінкові ознаки;
- розробити архітектуру мікросервісної системи;
- створити модулі перевірки доменів та URL-адрес у відкритих базах (VirusTotal, Google Web Risk);
- реалізувати механізм динамічного аналізу вебсторінок у ізольованому середовищі з емуляцією поведінки користувача для виявлення післяклікових загроз;
- розробити алгоритм агрегації та інтерпретації отриманих результатів з урахуванням їх надійності та контексту спрацьовування;
- провести тестування створеної системи на наборі реальних фішингових та легітимних сайтів.

Результатом розв'язання цих задач має стати прототип системи, здатний у реальному часі визначати рівень ризику вебресурсу та формувати узагальнений

висновок про його достовірність. Важливим є те, що така система не лише перевірятиме наявність сайту у базах загроз, а й аналізуватиме його фактичну поведінку та контент.

Отже, формалізована мета дослідження полягає в створенні системи, яка поєднує бази відкритих загроз із поведінковим аналізом для виявлення фішингових вебсайтів у динамічному режимі. Це дозволить зменшити затримку між появою нової загрози та її детекцією, збільшити точність класифікації і підвищити рівень автоматизації у веб-захисті.

ВИСНОВОК ДО РОЗДІЛУ 1

У результаті проведеного аналізу було досліджено природу фішингу, класифікацію його типів, життєвий цикл фішингових атак та характерні технічні індикатори, що дозволяють відрізнити шкідливі вебресурси від легітимних. Окрему увагу приділено існуючим технологічним рішенням, базам відкритих загроз і методологічним підходам до виявлення фішингових вебсайтів – від перевірки по базах даних і статичного аналізу контенту до поведінкових та гібридних стратегій.

Визначено, що існуючі сервіси, такі як VirusTotal, Google Web Risk, PhishTank і OpenPhish, ефективні у виявленні відомих загроз, але залишаються обмеженими щодо нових або динамічних фішингових сайтів, які активуються лише після взаємодії користувача. У той час як браузерні та корпоративні рішення (Microsoft Defender SmartScreen, Cisco Umbrella, Cloudflare Gateway) підвищують рівень захисту на мережевому рівні, вони також спираються на репутаційні бази й не забезпечують повного аналізу контенту та поведінки сторінки. Це зумовлює потребу у створенні систем нового покоління, здатних поєднувати різні рівні перевірки.

Науково-практичним результатом розділу є формування цілісного уявлення про сучасний стан проблеми виявлення фішингових вебресурсів і виявлення ключових недоліків існуючих рішень. Це створює підґрунтя для розроблення власної системи автоматичного аналізу фішингових сайтів, що поєднує репутаційні, структурні та поведінкові ознаки, забезпечуючи своєчасне виявлення нових загроз і підвищення рівня кіберстійкості веб-середовища.

РОЗДІЛ 2

РОЗРОБКА СИСТЕМИ АВТОМАТИЧНОГО ВИЯВЛЕННЯ ФІШИНГОВИХ САЙТІВ НА ОСНОВІ БАЗ ВІДКРИТИХ ЗАГРОЗ ТА ПОВЕДІНКОВИХ ОЗНАК

У цьому розділі розглянуто процес розробки системи автоматичного виявлення фішингових сайтів на основі поєднання перевірок у відкритих базах загроз і аналізу поведінкових ознак вебресурсів. Описано логіку роботи системи в межах мікросервісної архітектури, принципи організації інформаційних потоків, а також підхід до централізованої координації перевірок через сервіс агрегатора з урахуванням кешування результатів і контролю їх актуальності.

Наведено структурну схему та послідовність обміну даними між компонентами, обґрунтовано вибір технологій і контейнеризації, детально описано реалізацію кожного мікросервісу аналітичного рівня, сервісу взаємодії з базою даних, сховища та демонстраційного вебінтерфейсу. Розділ формує цілісне уявлення про програмну реалізацію системи, її модульність, масштабованість і практичну придатність для автоматизованої перевірки підозрілих посилань.

2.1. Розробка структурної схеми системи автоматичного виявлення фішингових сайтів

Під час проєктування системи автоматичного виявлення фішингових сайтів основним завданням є формування структури, що забезпечуватиме багаторівневу перевірку вебресурсів із використанням як відкритих баз даних загроз, так і поведінкових характеристик самих сайтів. Передбачається, що система буде побудована на основі мікросервісної архітектури, де кожен сервіс виконуватиме окрему функціональну роль, а взаємодія між ними здійснюватиметься через стандартизовані API-запити у форматі JSON. Такий підхід забезпечить гнучкість, масштабованість та ізоляцію окремих компонентів.

У загальному вигляді система має складатися з кількох логічно взаємопов'язаних рівнів: користувацького, аналітичного та рівня збереження даних.

Користувацький рівень включатиме вебінтерфейс (на початковому етапі розробки), за допомогою якого користувач зможе подати URL-адресу для перевірки. Інтерфейс виконуватиме функції ініціалізації запиту та відображення результатів. У перспективі передбачається можливість інтеграції системи у браузерне розширення або окремий клієнтський додаток.

Центральним елементом архітектури є агрегатор, який координує обмін даними між вебінтерфейсом і спеціалізованими мікросервісами аналізу. Після введення користувачем посилання агрегатор ініціює перевірку, звертаючись до сервісу взаємодії з базою даних для пошуку актуальних результатів попередніх перевірок. Якщо дані відсутні або втратили актуальність, агрегатор запускає процес аналізу та організовує паралельні запити до аналітичних компонентів, що дозволяє скоротити загальний час обробки.

Перевірка репутації посилання виконується сервісом аналізу посилань, який звертається до зовнішніх джерел відкритих загроз, зокрема Google WebRisk API та VirusTotal API, і повертає ознаки наявності ресурсу у репутаційних списках. Аналіз домену забезпечується окремим сервісом, що вилучає домен із вхідного посилання та отримує відповідні репутаційні показники через VirusTotal API. Контентний аналіз реалізовано як окремий мікросервіс, що завантажує вебсторінку у контрольованому браузерному середовищі, формує запит, після чого передає їх до Google AI API для виявлення фішингових патернів, ознак соціальної інженерії та підроблених елементів інтерфейсу.

Поведінковий аналіз виконує моделювання взаємодії користувача з ресурсом у браузерному середовищі. На основі рекомендацій мовної моделі формуються дії довірливого користувача, зокрема кліки та заповнення форм, а також відстежуються редиректи, відкриття нових вікон, появи сповіщень та зміни сторінки під час виконання сценарію. У разі суттєвих змін вмісту або переходу на нові сторінки поведінковий сервіс ініціює повторний контентний аналіз відповідних станів

сторінки, що дозволяє врахувати динамічну поведінку вебресурсу. Отримані результати для основного посилання та пов'язаних переходів агрегатор узагальнює і формує підсумковий висновок для користувача.

Збереження та актуалізація результатів покладені на сервіс взаємодії з базою даних, який виконує функції проміжного рівня між аналітичними компонентами та сховищем. База даних накопичує результати перевірок разом із часовими мітками та терміном актуальності, що дає змогу повторно використовувати непрострочені дані, зменшувати кількість звернень до зовнішніх API та підтримувати узгодженість історії перевірок. Після завершення аналізу агрегатор повертає результат через вебінтерфейс у вигляді стислого аналітичного звіту.

На рис. 2.1 наведено загальну структурну схему системи, що демонструє взаємозв'язки між основними компонентами та їхню взаємодію під час автоматичного аналізу фішингових посилань.

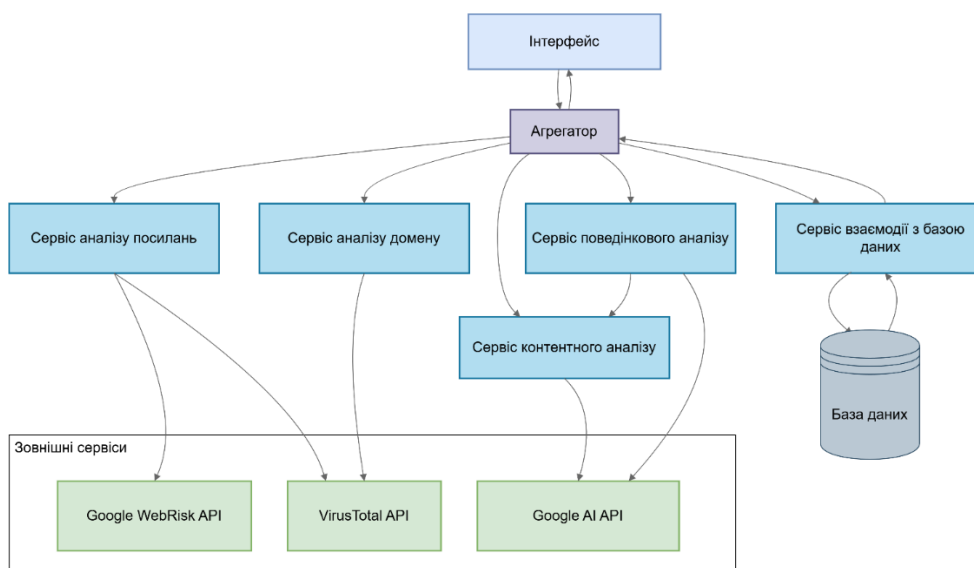


Рис. 2.1. Структурна схема системи автоматичного виявлення фішингових посилань

Запропонована структурна схема відображає логічну послідовність проходження інформаційних потоків під час аналізу вебресурсів. Кожен мікросервіс реалізує окремий тип перевірки, тоді як агрегатор синхронізує обчислення, об'єднує результати та забезпечує взаємодію з модулем збереження даних. Такий підхід підвищує масштабованість рішення, спрощує подальше

розширення функціональності й забезпечує ізоляцію компонентів під час експлуатації.

2.1.1. Загальна архітектура системи та взаємодія мікросервісів

Архітектура системи автоматичного виявлення фішингових сайтів побудована на принципі ізоляції мікросервісів, які взаємодіють через стандартизований обмін даними. Під час проєктування акцент зроблено на паралельному виконанні незалежних перевірок, низькій зв'язаності компонентів і централізованій координації процесу аналізу.

Мікросервіси розгортаються у контейнерах Docker із власним середовищем виконання та визначеним HTTP API. Вони об'єднані у спільну внутрішню мережу контейнерів, що обмежує прямий зовнішній доступ і забезпечує контрольований обмін даними. Передача результатів виконується у форматі JSON.

Координаційним центром системи є агрегатор, який приймає запит від користувацького інтерфейсу та ініціює повний цикл перевірки. На початку агрегатор звертається до сервісу взаємодії з базою даних для перевірки наявності попередніх результатів і їх актуальності. Якщо збережені дані знайдені та не прострочені, агрегатор організовує повернення результатів у вебінтерфейс у вигляді порційної відповіді без запуску повторних обчислень.

У разі відсутності всіх (або деяких) даних або їх прострочення агрегатор запускає паралельні запити до сервісів аналізу посилань, аналізу домену, контентного аналізу та поведінкового аналізу. Такий підхід скорочує загальний час обробки, оскільки незалежні перевірки виконуються одночасно. Поведінковий сервіс моделює дії користувача та відстежує зміни під час взаємодії зі сторінкою. За потреби, коли виявлено суттєві зміни контенту після редиректів або динамічних переходів, ініціюється повторний контентний аналіз для оновленого стану сторінки. Окремо в агрегаторі реалізовано опрацювання редиректів і нових вікон із рекурсивним переходом до заданої глибини, що дозволяє перевіряти не лише початковий ресурс, а й пов'язані з ним сторінки (з перевіркою наявності у базі даних).

Після отримання проміжних результатів агрегатор формує зведені дані, передає їх до сервісу взаємодії з базою даних для збереження та отримує підтвердження запису. Далі агрегатор повертає результати у вебінтерфейс порційно, а після цього інтерфейс надсилає окремий запит на розрахунок підсумкової оцінки. Агрегатор обчислює інтегральний бал ризику на основі заданих вагових коефіцієнтів та повертає фінальну оцінку для відображення користувачу.

Для відображення логіки комунікації між компонентами системи розроблено діаграму послідовності (рис. 2.2), яка ілюструє обробку одного запиту користувача з урахуванням кешування, паралельного запуску перевірок, повторного контентного аналізу після змін і опрацювання переходів. Прохід дочірніх посилань зображено спрощено, оскільки він дублює алгоритм перевірки основного посилання.

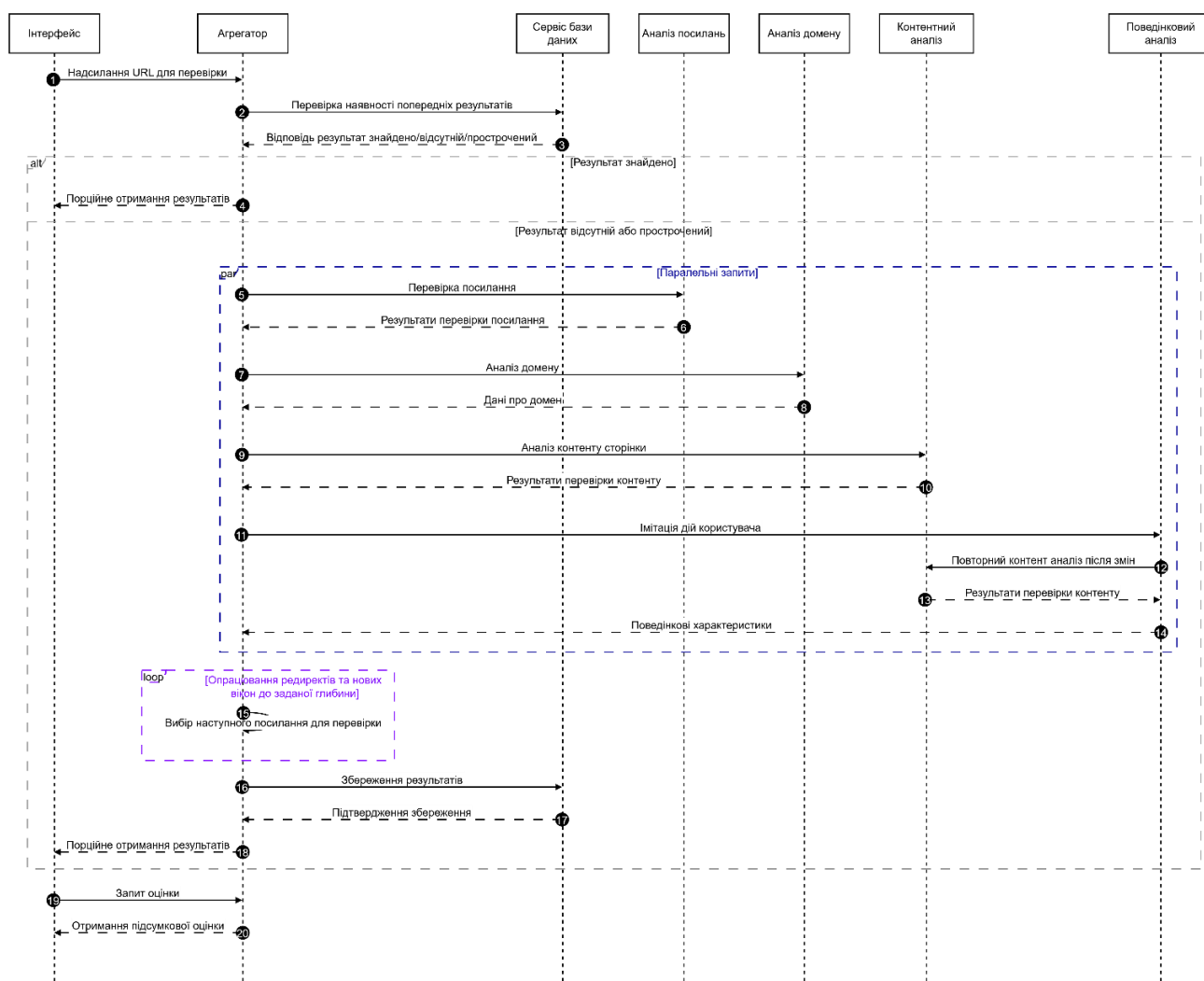


Рис. 2.2. Послідовність обміну даними між мікросервісами

На діаграмі показано, що користувач ініціює перевірку через інтерфейс, після чого агрегатор виконує перевірку наявності актуальних результатів у базі даних. Якщо результати наявні, вони повертаються одразу. Якщо ні, агрегатор паралельно запускає спеціалізовані сервіси аналізу, отримує від них проміжні дані, за потреби ініціює повторний контентний аналіз після змін і виконує опрацювання переходів до заданої глибини. Після цього результати зберігаються у базі та повертаються у вебінтерфейс разом із підсумковою оцінкою ризику. Така послідовність забезпечує узгодженість даних, зменшення часу обробки та можливість незалежного масштабування компонентів.

Таким чином, у рамках проєктованої архітектури взаємодія мікросервісів відбувається за принципом асинхронного обміну через центральний агрегатор, який виступає координаційним центром системи. Запропонована схема комунікації дозволяє забезпечити узгоджене функціонування незалежних компонентів, централізоване керування процесом перевірки та можливість подальшого розширення системи без зміни її загальної логіки.

2.1.2. Опис основних компонентів системи

Під час проєктування системи автоматичного виявлення фішингових сайтів важливим етапом є визначення функціонального призначення кожного компонента. Кожен мікросервіс системи виконує окрему задачу у межах єдиного процесу перевірки вебресурсу, а їх взаємодія забезпечується через централізований координаційний вузол – агрегатор.

Загалом система складається з таких основних компонентів: агрегатора, сервісів аналітичного рівня (аналіз посилань, домену, контенту та поведінкових ознак), сервісу взаємодії з базою даних, самої бази даних і користувацького інтерфейсу. Кожен із цих елементів виконує окрему функцію, але в сукупності вони утворюють єдиний логічний ланцюг перевірки та оцінки вебресурсів.

Агрегатор є центральним логічним компонентом системи. Його головним завданням є координація роботи всіх мікросервісів, керування потоками даних і формування підсумкового результату перевірки. Під час надходження запиту від

користувача агрегатор перевіряє наявність запису про цей ресурс у базі даних, визначає, чи не втратили попередні результати актуальність, та за необхідності ініціює новий цикл аналізу. У процесі роботи агрегатор надсилає паралельні запити до аналітичних сервісів, отримує від них результати у стандартизованому форматі JSON і виконує порційне повернення даних. Також він формує зведений звіт по перевіркам редіректів та нових вікон що з'явилися після поведінкового аналізу, та надсилає дані у поведінковий аналізатор – для отримання загального висновку.

Окрім координаційних функцій, агрегатор також здійснює:

- розрахунок інтегрального показника ризику фішингу на основі вагових коефіцієнтів, встановлених для кожного типу перевірки;
- передавання результатів до бази даних через окремий сервіс взаємодії;
- порційне повернення результатів сервісів;
- рекурсивне проходження по новим посиланням після поведінкового аналізу, з метою аналізу усіма сервісами та формування звіту для підсумкового поведінкового аналізу;
- повернення висновку до користувацького інтерфейсу.

Завдяки цьому агрегатор виконує роль логічного «центру керування» всією системою.

Сервіс аналізу посилань виконує перевірку поданого URL у відкритих базах кіберзагроз. Під час проєктування передбачається використання API VirusTotal та Google Web Risk, які надають відомості про виявлені загрози, категорії шкідливості та рівень репутації посилань. Отримані дані включатимуть статус (безпечний, шкідливий), ідентифікатори загроз, джерела виявлення та час останнього оновлення інформації.

Результати перевірки передаватимуться в агрегатор у стандартизованому форматі, що дозволить подальшу автоматизовану обробку без залежності від конкретного API.

Сервіс аналізу домену призначено для оцінки надійності домену, на якому розташовано досліджуваний сайт. Основним джерелом даних є VirusTotal API.

Результати дозволяють оцінити, чи може домен бути частиною фішингової інфраструктури.

Сервіс контентного аналізу, ставить собі на меті дослідження вмісту вебсторінки для виявлення характерних фішингових ознак. Для цього передбачається використання інструмента Selenium у режимі безпечного віртуального браузера, який завантажуватиме сторінку, відтворюючи її так, як це бачить реальний користувач.

Після повного рендерингу сторінки HTML-код, текстовий контент і метадані передаватимуться до мовної моделі LLM (Gemini) для семантичного аналізу.

Модель оцінюватиме:

- наявність закликів до введення конфіденційних даних;
- елементи візуальної підробки (логотипи, імітації брендів);
- підозрілі JavaScript-скрипти або приховані форми;
- некоректні посилання.

Отримані результати будуть представлені у вигляді опису знайдених ознак та оцінки рівня підозрілості контенту.

Поведінковий сервіс моделює дії користувача на сайті для виявлення прихованих сценаріїв фішингу, які активуються лише після взаємодії. Сервіс також використовує Selenium для імітації базових дій – заповнення форм, натискання кнопок, переходів за редіректами. Крім того, передбачається застосування мовної моделі для визначення логіки взаємодії: наприклад, які елементи на сторінці можуть спонукати користувача до введення даних або виконання ризикованих дій.

Під час аналізу збирається інформація про:

- активність скриптів під час взаємодії;
- поява сповіщень;
- нові сторінки, що відкриваються після кліків або редіректів;
- зміну контенту сторінки після виконання дій.

Якщо виявляється зміна контенту поточної сторінки, вона передається на повторний контентний аналіз, що створює послідовний ланцюг поведінкової оцінки

ресурсу. Редиректи та нові вікна формують загальний список, що надалі рекурсивно буде опрацьовуватись агрегатором.

Сервіс взаємодії з базою даних виконує проміжну функцію між аналітичними сервісами та базою даних. Його завданням є централізоване збереження результатів перевірок, забезпечення узгодженості даних і контроль термінів актуальності. Сервіс працює як API-шар, який приймає запити від агрегатора, проводить запис або оновлення даних у сховищі, а також надає агрегатору збережені результати у разі повторного звернення. Завдяки відокремленню цього рівня від основної логіки аналізу досягається підвищення безпеки – аналітичні сервіси не мають прямого доступу до бази даних.

Як основне сховище даних передбачається використання PostgreSQL, яка забезпечує стабільність, підтримку транзакцій і можливість масштабування.

База даних зберігатиме:

- відомості про результати перевірок у окремих таблицях;
- часові мітки проведення аналізу.

Для забезпечення актуальності передбачається механізм оновлення результатів після закінчення терміну їх дії, визначеного системними параметрами.

На етапі початкового проектування користувацький рівень реалізується у вигляді вебінтерфейсу. Він виконує роль вхідної точки для користувача, дозволяючи ввести посилання для перевірки, запустити процес аналізу та переглянути результати. Інтерфейс отримує дані через REST API агрегатора і відображає короткий висновок, а також детальний звіт із позначенням, які саме перевірки виявили потенційні ризики.

У подальшому передбачається можливість інтеграції цього модуля у браузерне розширення або мобільний додаток, що розширить практичне застосування системи.

Таким чином, кожен із описаних компонентів системи виконує чітко визначену функцію, а їх взаємодія у межах мікросервісної архітектури забезпечує ефективність, гнучкість і надійність майбутньої системи.

2.1.3. Використання контейнеризації Docker

Для побудови системи автоматичного виявлення фішингових сайтів передбачено застосування технології Docker, що забезпечує розгортання всіх компонентів у вигляді ізольованих контейнерів. Такий підхід дозволяє створити уніфіковане середовище, незалежне від конкретної операційної системи, та гарантує стабільність роботи кожного мікросервісу. Контейнеризація також спрощує тестування, оновлення й масштабування системи, роблячи її гнучкою та легко переносною між різними інфраструктурами [23].

Кожен сервіс системи розміщується у власному контейнері. Центральну роль виконує `risk_aggregator`, який координує запити, об'єднує результати аналізів і формує підсумкову оцінку ризику. Безпосередню взаємодію з користувачем здійснює контейнер `interface`, що реалізує вебінтерфейс для введення посилань і відображення результатів перевірки. Користувач може ввести URL, після чого вебінтерфейс надсилає запит до агрегатора й очікує відповідь із підсумковим висновком.

До аналітичної частини належать сервіси `threat_intel`, `domain_analyzer`, `content_analyzer` та `behavior_analyzer`. Вони функціонують незалежно та відповідають за різні аспекти аналізу: перевірку у відкритих базах загроз, оцінку домену, контентну перевірку та моделювання поведінки користувача.

Під час виконання контентного та поведінкового аналізу сторінки отримані HTML-дані передаються до зовнішньої мовної моделі через API, яка визначає наявність фішингових патернів, закликів до дій або підозрілих елементів інтерфейсу. Результати обробки повертаються у стандартизованому форматі JSON і передаються агрегатору для узагальнення.

Збереження даних здійснюється через `history_service`, який обробляє запити від агрегатора та записує результати у `postgres_db`. Це дозволяє системі зберігати історію перевірок і повторно використовувати результати для пришвидшення наступних запитів.

Усі контейнери з'єднані у спільну внутрішню мережу Docker, що забезпечує безпечний обмін даними без відкриття портів у зовнішнє середовище. Взаємодія між контейнерами відбувається через REST API з обміном даними у форматі JSON, що гарантує сумісність і масштабованість системи. На рис. 2.3 показано взаємодію контейнерів системи у середовищі Docker.

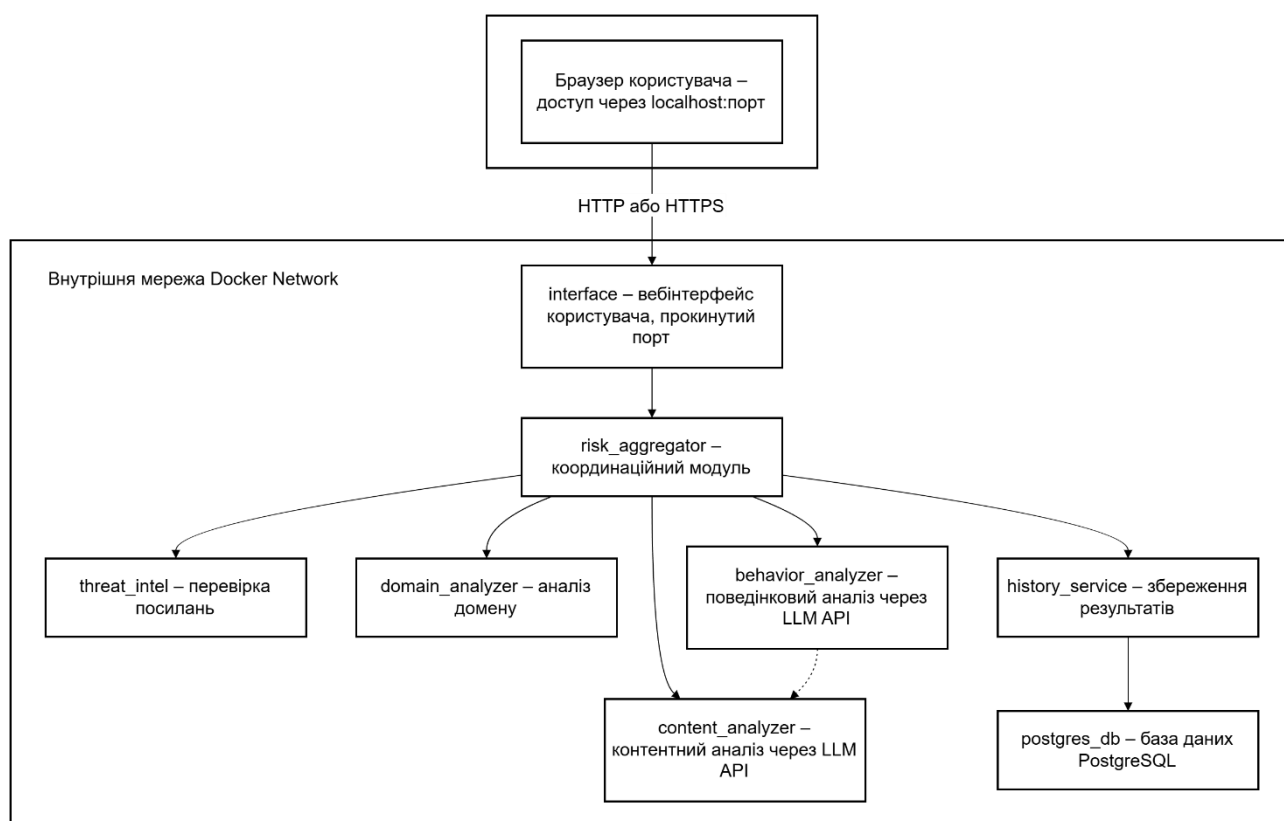


Рис. 2.3. Діаграма розгортання контейнерів системи

Користувач отримує доступ до системи через веббраузер, підключаючись до контейнера interface за допомогою прокинутого порту. Цей контейнер містить вебінтерфейс для введення посилань і відображення результатів перевірки. Запити від нього надходять до risk_aggregator, який координує роботу аналітичних сервісів: threat_intel, domain_analyzer, content_analyzer та behavior_analyzer. Після завершення аналізу результати передаються до history_service, що взаємодіє з базою postgres_db. Усі контейнери об'єднані спільною мережею Docker, що забезпечує внутрішню комунікацію без відкриття зовнішніх портів, окрім вебінтерфейсу. Така конфігурація підвищує безпеку та дозволяє масштабувати систему без зміни її структури.

2.2. Вибір технологій для реалізації мікросервісів системи

Для реалізації системи автоматичного виявлення фішингових сайтів було обрано стек технологій, який поєднує стабільність, масштабованість і зручність інтеграції зовнішніх API. Основною мовою програмування є Python, оскільки вона має широкий набір бібліотек для обробки вебданих, роботи з API, машинного аналізу та розгортання мікросервісів. Python є відкритим стандартом де-факто у сфері кібербезпеки, Data Science та вебавтоматизації завдяки своїй простоті, читабельності та наявності великої кількості фреймворків [24].

Мікросервіси системи розробляються мовою Python 3.11, що забезпечує сучасну підтримку типізації, асинхронного виконання та покращене управління пам'яттю.

Для логування використовується стандартний модуль logging, який дозволяє централізовано фіксувати події системи та зберігати їх у журналах для подальшого аналізу. Модуль os використовується для роботи з системними змінними середовища, а бібліотека dotenv – для завантаження конфіденційних налаштувань (ключів API, портів, параметрів бази даних) із файлу .env, що підвищує безпеку розробки.

Модулі datetime, timedelta та timezone застосовуються для ведення часових міток під час перевірки ресурсів і контролю терміну актуальності результатів. Стандартна бібліотека pathlib спрощує роботу з файловими шляхами в кросплатформному середовищі, а типізація через typing (List, Dict, Optional) покращує читабельність і стабільність коду.

Для взаємодії з відкритими базами загроз застосовується офіційний клієнт VirusTotal API через бібліотеку vt [8]. Вона дозволяє отримувати детальну інформацію про URL-адреси, IP, домени або файли, включаючи категорію шкідливості, кількість позитивних детекцій і дату останнього сканування.

Для доповнення перевірок використовується також Google Web Risk API, що надсилає запити до офіційного REST-ендпоінта <https://webrisk.googleapis.com/v1/uris:search>. Цей сервіс дозволяє перевіряти

посилання на наявність у базах соціальної інженерії, шкідливого ПЗ або небажаного контенту [10].

HTTP-взаємодія з цими сервісами здійснюється за допомогою бібліотеки `requests`, що є де-факто стандартом для Python-застосунків і підтримує роботу з сесіями, заголовками та обробкою помилок мережесих запитів.

Кожен компонент системи реалізовано як окремий мікросервіс із REST API на базі фреймворку `Flask` [25]. `Flask` відзначається мінімалістичною архітектурою, що дозволяє створювати легкі сервіси без надлишкової логіки.

Для завантаження вебсторінок і їхньої обробки використовується бібліотека `Selenium` у поєднанні з `Google Chrome WebDriver`. `Selenium` дозволяє автоматично відкривати сторінки, виконувати `JavaScript` і знімати повний HTML-код навіть у випадках, коли контент генерується динамічно [26].

Для аналізу структури сторінки використовується також `BeautifulSoup`, який забезпечує зручний синтаксичний парсинг HTML-документів [27].

Після отримання HTML-коду або текстового вмісту сторінка передається на обробку зовнішній мовній моделі через `Google AI Studio API` [28]. У системі передбачено підтримку моделей `Gemini 2.5 Flash Lite`, які спеціалізуються на обробці природної мови та розпізнаванні семантичних патернів. Вони дозволяють визначати фішингові елементи, такі як заклики до дій («`verify your account`»), візуальні підробки логотипів або запити на введення особистих даних. Зв'язок із моделлю здійснюється через HTTP-запит до ендпоінта.

Для обробки доменних імен використовується бібліотека `tlldextract`, яка розділяє URL на складові (домен, піддомен, суфікс) відповідно до офіційного списку публічних доменів. Це дозволяє формувати правильні запити до `VirusTotal API`.

Крім того, модуль `urllib.parse` використовується для аналізу структури посилань, нормалізації їхнього формату й перевірки коректності введених користувачем URL.

Як система керування базами даних використовується `PostgreSQL`, розгорнута у вигляді окремого контейнера. Вибір саме `PostgreSQL` зумовлений її

надійністю, підтримкою транзакцій і відповідністю ACID-принципам, що критично важливо для збереження результатів перевірок [29].

Усі сервіси розгортаються у середовищі Docker, а їхня взаємодія визначається у конфігураційному файлі `docker-compose.yml`. Кожен контейнер має власний образ, порт і змінні середовища, що завантажуються з `.env` файлу. Docker забезпечує ізоляцію процесів, стабільність виконання незалежно від операційної системи та просте масштабування. Для зберігання даних використовується том Docker Volume, який дозволяє зберігати історію перевірок навіть після перезапуску контейнерів [23].

Вебінтерфейс системи реалізовано у вигляді окремого мікросервісу на базі Streamlit [30], що забезпечує інтерактивну взаємодію користувача з системою. Через вебсторінку користувач вводить посилання для перевірки, спостерігає за процесом аналізу та отримує результати у вигляді динамічно оновлюваних блоків для кожного сервісу. Інтерфейс обмінюється даними з агрегатором через HTTP запити та отримує відповіді у форматі JSON, що забезпечує уніфіковану передачу результатів між компонентами. Після завершення збору даних інтерфейс надсилає зведені результати на ендпоінт розрахунку інтегральної оцінки, після чого відображає підсумковий бал ризику.

Сервіс поведінкового аналізу реалізовано на базі Selenium, що дозволяє емулювати взаємодію з вебсторінкою у браузерному середовищі, зокрема виконувати кліки та заповнення форм. На етапі підготовки сценарію дій використовується мовна модель Google AI Gemini, яка формує послідовність дій довірливого користувача. Під час виконання сценарію сервіс фіксує поведінкові індикатори, зокрема редиректи, відкриття нових вікон, появи алертів і суттєві зміни сторінки. У разі перевищення порогового значення зміни вмісту сервіс ініціює повторний контентний аналіз, передаючи поточні HTML та знімок екрана до сервісу контентного аналізу для уточнення висновку щодо безпечності сторінки.

Таким чином, реалізована система поєднує інтерактивний вебінтерфейс на Streamlit та багаторівневу мікросервісну архітектуру, де кожен компонент виконує окрему аналітичну функцію, а агрегатор координує паралельні перевірки та

взаємодію з базою даних. Завдяки використанню Python, Streamlit, Selenium, Docker і зовнішніх API VirusTotal, Google Web Risk та Google AI Gemini забезпечується розширювана й масштабована основа для автоматичного виявлення фішингових вебресурсів.

2.3. Розробка програмної компоненти системи автоматичного виявлення фішингових сайтів

2.3.1. Інженерія вимог до функціональних компонентів системи

На етапі інженерії вимог визначено цілі, функції та обмеження системи автоматичного виявлення фішингових сайтів. Головним завданням є формалізація поведінки кожного компонента в межах єдиної мікросервісної архітектури. Система повинна здійснювати комплексний аналіз вебресурсу за кількома напрямками – перевірка у базах відомих загроз, аналіз домену, оцінка контенту сторінки та дослідження поведінкових характеристик під час взаємодії користувача.

Розроблювана система має забезпечувати автоматичну перевірку URL-адреси, введеної користувачем через вебінтерфейс, із подальшим формуванням оцінки ризику фішингу. У процесі обробки запит проходить кілька етапів – агрегацію, аналітичну перевірку, формування звіту та збереження результатів у базі даних.

До ключових вимог належать:

- підтримка паралельного виконання запитів до аналітичних мікросервісів для скорочення часу відповіді;
- повторне використання збережених результатів за умови їх актуальності, а у разі прострочення – автоматичне оновлення;
- централізована координація обміну даними через сервіс агрегатора;
- формування структурованих результатів у форматі JSON для кожного модуля та підсумкової оцінки ризику.

Архітектура системи складається з незалежних мікросервісів, кожен з яких реалізує окрему логічну функцію. Центальною ланкою виступає агрегатор ризику,

який координує всі етапи обробки запиту. Він приймає URL від вебінтерфейсу, звертається до сервісу взаємодії з базою даних для перевірки історії і терміну актуальності, а за необхідності ініціює паралельну роботу аналітичних сервісів, збирає їхні відповіді та формує узагальнений результат. Окремо агрегатор забезпечує рекурсивне опрацювання посилань, що виникають внаслідок редиректів або відкриття нових вікон, із обмеженням максимальної глибини аналізу, що задається у змінних середовища.

Сервіс аналізу посилань здійснює запити до API VirusTotal та Google Web Risk. Його завдання – визначити, чи входить посилання до бази відомих загроз і який рівень довіри до нього наявний. Результатом роботи є набір показників безпеки з часовими мітками.

Сервіс аналізу домену працює із доменними характеристиками ресурсу. Він отримує інформацію про наявність даного домену у базах загроз VirusTotal API.

Сервіс контентного аналізу реалізує завантаження сторінки у безпечному середовищі за допомогою Selenium. Після повного рендерингу контент передається до мовної моделі через Google AI API (Gemini), де відбувається семантичний аналіз тексту та елементів інтерфейсу на наявність фішингових патернів, підроблених логотипів чи закликів до введення персональних даних.

Сервіс поведінкового аналізу виконує моделювання дій довірливого користувача під час відвідування сайту. На першому етапі він ініціалізує браузерне середовище, отримує поточний стан сторінки та за допомогою Google AI Gemini формує набір дій, зокрема кліки й заповнення форм. Далі дії виконуються у Selenium із фіксацією поведінкових індикаторів: редиректів, відкриття нових вікон, появи алертів і змін DOM. Якщо зміна сторінки перевищує заданий поріг, за замовчуванням 50 відсотків, сервіс ініціює повторний контентний аналіз, передаючи до сервісу контентного аналізу актуальну сторінку. Також сервіс формує окремий підсумковий висновок поведінки на основі зібраних результатів для основного URL та пов'язаних сторінок, що з'явилися після редиректів або відкриття нових вікон, із обмеженням глибини аналізу.

Сервіс взаємодії з базою даних відповідає за збереження, оновлення, перевірку актуальності та каскадне видалення результатів у PostgreSQL. Він контролює термін дії записів, забезпечує доступ до історії аналізів та зменшує дублювання запитів до зовнішніх сервісів за рахунок кешування. Ініціалізація структури бази даних виконується під час старту сервісу.

Вебінтерфейс є єдиною точкою зовнішньої взаємодії з системою та реалізований як окремий сервіс на Streamlit. Через нього користувач вводить URL, отримує результати перевірок у вигляді окремих блоків та підсумкову оцінку ризику. Інтерфейс здійснює обмін даними з агрегатором через HTTP запити з передачею JSON.

Окрім основного функціоналу, система повинна відповідати нефункціональним вимогам, що визначають її якісні характеристики:

- продуктивність – прийнятний час обробки запиту з урахуванням ресурсоємного поведінкового аналізу;
- масштабованість – можливість незалежного масштабування аналітичних сервісів у Docker середовищі;
- безпека – обмеження зовнішнього доступу лише до вебінтерфейсу, зберігання ключів доступу у змінних середовища;
- відмовостійкість – ізоляція компонентів, за якої збій окремого сервісу не призводить до некоректної роботи всієї системи;
- логування – фіксація викликів, помилок і результатів перевірок для подальшого аналізу;
- модульність – можливість додавання або заміни компонентів без зміни загальної логіки взаємодії.

Система розгортається у контейнерному середовищі Docker. Для коректної роботи необхідно забезпечити встановлення Docker Engine та Docker Compose, використання PostgreSQL як бази даних, наявність змінних середовища для API-ключів VirusTotal, Web Risk та Google AI, доступ до Інтернету для виконання

зовнішніх запитів, а також середовище виконання Python 3.11 і необхідні бібліотеки для реалізації HTTP сервісів та автоматизації браузера.

Таким чином, на етапі інженерії вимог визначено логічну структуру системи, функції кожного мікросервісу та їх взаємодію. Це забезпечує узгодженість компонентів, повторне використання результатів, можливість масштабування та подальший розвиток системи в умовах реального застосування.

2.3.2. Архітектурне проєктування програмного забезпечення

Архітектурне проєктування визначає логічну організацію програмних компонентів системи автоматичного виявлення фішингових сайтів та способи їх взаємодії. Метою цього етапу є побудова структури програмного забезпечення, яка забезпечить гнучкість, надійність і незалежність складових частин системи при збереженні їхньої взаємодії через стандартизовані інтерфейси.

Система реалізована за принципом мікросервісної архітектури, у межах якої кожен сервіс є самостійним застосунком із власним середовищем виконання, конфігурацією та мережевими портами. Всі сервіси розгортаються у контейнерах Docker, що забезпечує ізоляцію процесів і стабільність середовища.

Ключовою особливістю проєктного рішення є повна відсутність безпосередніх програмних залежностей між компонентами: сервіси не імпортують код один одного, не використовують спільні бібліотеки чи внутрішні виклики. Їхня взаємодія реалізована виключно через REST-запити з використанням протоколу HTTP та передачі даних у форматі JSON. Такий підхід гарантує слабку зв'язаність між частинами системи, що суттєво спрощує супровід і масштабування.

Під час проєктування архітектури програмного забезпечення враховано такі принципи:

- кожен мікросервіс має власний життєвий цикл і може запускатися, оновлюватися або зупинятися незалежно від інших.
- обмін даними здійснюється запитом HTTP типів GET і POST, структура відповіді стандартизована у JSON.

- агрегатор координує виконання перевірок одночасно у кількох сервісах, очікуючи їхні відповіді незалежно.
- усі сервіси підключені до внутрішньої мережі Docker, а зовнішній доступ надано лише контейнеру з вебінтерфейсом.
- конфігурація контейнерів і мережі описана у файлі `docker-compose.yml`.
- усі запити від користувача та відповіді від сервісів проходять через агрегатор ризику.

Усі сервіси мають вбудовану систему логування на основі стандартного модуля logging. Події фіксуються із зазначенням часу, рівня важливості та джерела. Для захисту системи передбачено:

- ізоляцію внутрішньої мережі Docker (жоден аналітичний сервіс не має відкритого порту назовні);
- авторизацію API-ключів для доступу до зовнішніх ресурсів;
- тайм-аути та повторні спроби запитів у разі мережових збоїв;
- перевірку коректності вхідних даних на рівні кожного сервісу.

Усі критичні помилки обробляються централізовано — у випадку недоступності одного сервісу агрегатор повертає часткові результати з відповідною позначкою у звіті.

Обраний підхід до архітектури дозволяє:

- забезпечити повну незалежність і замінність мікросервісів;
- масштабувати систему за рахунок дублювання окремих контейнерів;
- спрощувати супровід і тестування, оскільки кожен компонент розгортається автономно;
- підвищити безпеку, обмеживши взаємодію лише стандартизованими запитамі;
- легко інтегрувати нові аналітичні модулі, не змінюючи код існуючих сервісів.

Таким чином, архітектурне проєктування програмного забезпечення забезпечило створення стійкої, розподіленої та гнучкої інфраструктури, у якій

сервіси взаємодіють через мережеві API-запити, зберігаючи незалежність логіки, структури коду й середовища виконання. Такий підхід робить систему масштабованою, безпечною та придатною до подальшого розвитку.

2.3.3. Реалізація сервісу перевірки посилань

Сервіс перевірки посилань є базовим елементом аналітичного рівня системи автоматичного виявлення фішингових сайтів. Його призначення полягає у перевірці переданих URL-адрес на наявність загроз у відкритих базах даних. Основна логіка сервісу спрямована на інтеграцію з двома зовнішніми джерелами – Google Web Risk та VirusTotal, що забезпечує комбінований аналіз безпечності вебресурсу як на рівні репутації, так і за антивірусними сигнатурами.

Розробка виконана мовою Python із використанням фреймворку Flask, який реалізує вебсервер для прийому запитів від агрегатора. Окрім цього, застосовано бібліотеки requests для REST-комунікацій, vt-py для роботи з API VirusTotal, python-dotenv для зчитування змінних середовища та модуль logging для централізованого ведення журналу подій. Сервіс розгортається у середовищі Docker, що дозволяє ізолювати виконання, мінімізувати залежності та забезпечити відтворюваність.

Після запуску контейнера Flask-додаток очікує HTTP-запит від агрегатора, у якому передається URL-адреса для перевірки. Першим етапом відбувається ініціалізація клієнтів двох API – WebRiskClient і VirusTotalClient, яким передаються відповідні ключі доступу, завантажені з файлу .env. Після перевірки коректності ключів виконується послідовність двох основних операцій.

Спершу WebRiskClient звертається до API Google Web Risk, надсилаючи параметри uri, threatTypes і ключ автентифікації. У разі збігу виявляються типи загроз, наприклад SOCIAL_ENGINEERING або MALWARE, після чого формується структура результату з інформацією про рівень безпечності, часову мітку перевірки та термін дії результату.

Другим етапом є виклик клієнта VirusTotalClient, який використовує бібліотеку vt-py для звернення до REST API сервісу VirusTotal. URL перетворюється у внутрішній ідентифікатор за допомогою функції vt.url_id(), після

чого виконується запит до об'єкта `/urls/{id}`. Якщо дані для цієї адреси відсутні, сервіс ініціює автоматичне сканування. Отримані результати містять оцінки різних антивірусних рушіїв – таких як Google Safebrowsing, ESET, Kaspersky, OpenPhish тощо. Усі ці дані агрегуються за типами «malicious», «suspicious», «harmless» і «undetected». На основі зведеної статистики формується загальна оцінка безпеки, яка передається далі.

Після завершення обох запитів результати узагальнюються, і формується відповідь у форматі JSON. У ній зазначається, чи є посилання безпечним, додаються статистичні показники, часові межі дії результату та, за потреби, пряме посилання на детальний звіт VirusTotal. Якщо один із зовнішніх сервісів недоступний, система повертає частковий результат, що дозволяє зберегти працездатність.

Під час розробки особлива увага приділялась стабільності та безпеці виконання. Усі ключі для API передаються через змінні середовища, що виключає можливість витоку конфіденційних даних. Кожен запит до зовнішнього ресурсу має обмеження за часом (timeout), а винятки `requests.exceptions.RequestException` та `vt.error.APIError` обробляються централізовано, що запобігає аварійному завершенню процесу.

Сервіс не зберігає даних у власному контейнері – він функціонує лише як аналітичний модуль і повертає результат перевірки до агрегатора. Завдяки цьому він не має внутрішнього стану, що спрощує масштабування та розподілення навантаження.

Блок-схема алгоритму роботи наведена на рис. 2.4.

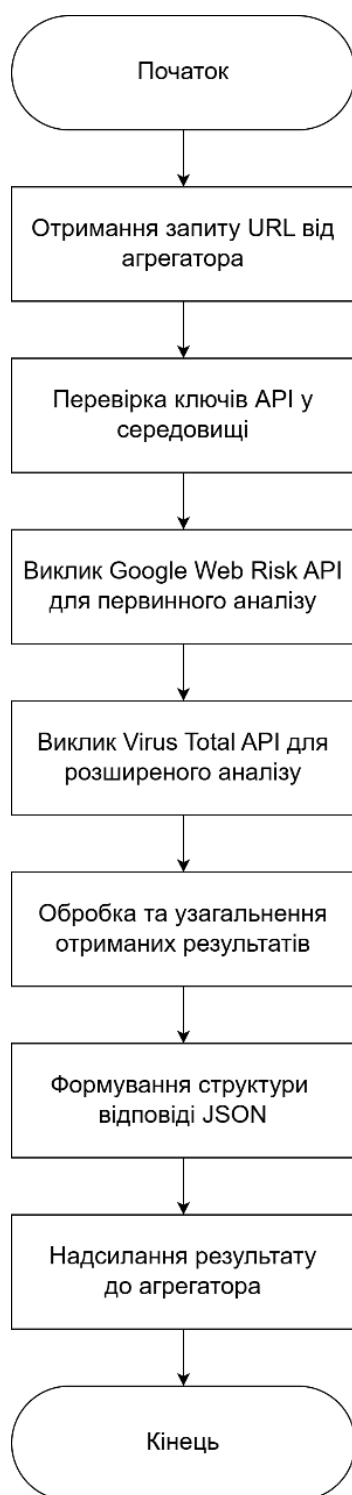


Рис. 2.4. Блок-схема алгоритму роботи сервісу аналізу посилань

Сервіс описано у файлі `Dockerfile`, який створює контейнер на базі образу `python:3.11-slim`. У контейнер копіюються вихідні файли, встановлюються залежності з `requirements.txt` і запускається Flask-сервер на визначеному порту. У `docker-compose.yml` цей сервіс має власну назву `threat_intel` і підключений до внутрішньої мережі системи без відкриття зовнішніх портів. Таким чином

забезпечується безпечна взаємодія лише через агрегатор, який виступає єдиною точкою.

Реалізований сервіс перевірки посилань успішно виконує одночасне звернення до двох незалежних API, що дозволяє підвищити достовірність результатів. Інтеграція з Google Web Risk забезпечує швидке виявлення сайтів, які вже відомі як шахрайські, а VirusTotal дозволяє виявляти нові, ще не класифіковані загрози завдяки використанню багатьох антивірусних рушіїв. Об'єднання цих підходів робить сервіс ефективним механізмом попереднього фільтрування у загальній архітектурі системи. Реалізацію процесу перевірки посилань винесено в Додаток А.

2.3.4. Реалізація сервісу перевірки доменів

Сервіс аналізу домену є другим аналітичним компонентом системи автоматичного виявлення фішингових сайтів. Його основне завдання – визначити репутацію домену на основі даних платформи VirusTotal, зокрема виявити наявність шкідливої активності, історію фішингових звітів або інших ознак компрометації. Результати перевірки використовуються агрегатором для підсумкової оцінки ризику.

Сервіс розроблено на Python 3.11 з використанням фреймворку Flask, який забезпечує обробку HTTP-запитів, і бібліотеки vt-py для інтеграції з API VirusTotal. Додатково використано tldextract для коректного виділення доменних частин із URL, dotenv для зчитування змінних середовища та logging для ведення системного журналу. Конфігурація середовища виконання описана у Dockerfile, а залежності зазначені у requirements.txt.

Після запуску контейнера Flask-додаток приймає запити від агрегатора через маршрут /check_domain. Переданий URL обробляється методом check_domain(), який послідовно виконує три ключові етапи: виділення домену з URL-адреси; отримання звіту з VirusTotal через офіційний API; формування узагальненого висновку про рівень безпечності; виділення домену виконується методом _extract_domain(), який за допомогою бібліотеки tldextract аналізує адресу та

повертає уніфікований формат виду `example.com`. Це дозволяє уникнути помилок під час обробки піддоменів і нестандартних URL.

Далі створюється екземпляр клієнта `vt.Client`, через який виконується запит: `/domains/{domain}`. У відповідь повертається об'єкт, що містить результати перевірок різними антивірусними рушіями. Однак сервіс не обробляє всі результати, для зменшення шуму використовується спеціальний набір релевантних рушіїв, серед яких PhishTank, Netcraft, OpenPhish, SafeToOpen та інші. Це дає змогу зосередитися на джерелах, що спеціалізуються саме на фішингових і доменних загрозах.

Після відбору здійснюється статистичне узагальнення даних. Метод `_summarize_relevant_stats()` класифікує результати за категоріями: `malicious`, `suspicious`, `harmless`, `undetected` і `timeout`. Якщо кількість перших двох категорій дорівнює нулю, домен вважається безпечним. На основі цих показників формується структура результату з полями `domain`, `safe`, `stats`, `checked_at`, `expire_time` і `details_url`. Час перевірки фіксується у форматі UTC, а термін актуальності визначається змінною `EXPIRED_DAYS` (за замовчуванням 7 днів).

Реалізація сервісу орієнтована на стійкість і незалежність. Усі змінні середовища, включно з ключем `VT_API_KEY`, завантажуються автоматично при запуску. При некоректному або порожньому URL система генерує інформативне повідомлення про помилку, а при недоступності API повертається структура з описом винятку, не перериваючи роботу контейнера.

Усі події журналюються через модуль `logging`, що дозволяє відстежувати кожен етап обробки: отримання запиту, виконання перевірки, результати аналізу або помилки підключення. Це забезпечує простоту налагодження й інтеграції сервісу з агрегатором.

Блок-схема алгоритму роботи сервісу наведена на рис. 2.5.



Рис. 2.5. Блок-схема алгоритму роботи сервісу аналізу домену

Схема демонструє послідовність дій від моменту надходження запиту до повернення результату. Усі етапи виконуються послідовно, а комунікація з VirusTotal реалізована у межах одного сеансу клієнта, який автоматично закривається після завершення обробки.

Сервіс описано у Dockerfile, який базується на образі python:3.11-slim. У процесі побудови контейнера копіюються файли додатка (app.py, domain_analyzer.py), встановлюються бібліотеки з requirements.txt і задаються змінні середовища для порту та ключів.

Контейнер підключений до внутрішньої мережі Docker, де взаємодіє лише з агрегатором. Завдяки цьому усі мережеві запити відбуваються у контрольованому середовищі без відкриття зовнішніх портів.

Реалізований сервіс успішно виконує аналіз доменів у реальному часі, повертаючи уніфікований JSON-відповідь, придатну для подальшої обробки іншими компонентами системи. Завдяки використанню релевантних антивірусних рушіїв зменшується кількість хибних спрацьовувань, а система отримує більш точну оцінку репутації ресурсу. Реалізацію ключових компонентів сервісу доменного аналізу наведено в Додатку Б.

Отримані результати дозволяють виявляти як давно скомпрометовані домени, так і нові загрози, що тільки починають поширювати фішингові кампанії. У поєднанні з аналізом контенту та поведінковими характеристиками цей сервіс формує один із ключових рівнів захисту у загальній архітектурі системи.

2.3.5. Реалізація сервісу аналізу контенту сторінки

Сервіс контентного аналізу є ключовим елементом системи, оскільки забезпечує безпосередній аналіз вмісту вебсторінки. Його завдання полягає у безпечному завантаженні сторінки у контрольованому середовищі, вилученні релевантних фрагментів, а також передачі цих даних для подальшої оцінки мовною моделлю. Таким чином, сервіс виконує роль «візуального сканера» вебресурсу, який імітує поведінку користувача, але без ризику зараження системи.

Реалізацію виконано мовою Python 3.11 із застосуванням Flask як HTTP-сервера, Selenium WebDriver для керування браузером, BeautifulSoup4 для синтаксичного розбору HTML, Pillow для обробки зображень і Google Generative AI API для аналізу контенту сторінки.

Після запуску контейнера Flask-сервер приймає запити від агрегатора на перевірку сторінки за переданим URL. Далі виконується основний цикл, реалізований у класі PageExtractor, який забезпечує безпечне відкриття сайту, симуляцію дій користувача, витягнення вмісту сторінки та генерацію скріншоту.

Для мінімізації ризиків під час відкриття потенційно шкідливих сайтів реалізовано віртуальне середовище перегляду: сторінка завантажується у headless-браузері Chrome із тимчасовим профілем користувача. Усі дії ізольовано всередині контейнера Docker.

Перед відкриттям вебресурсу створюється профіль із рандомним User-Agent, що імітує справжнього користувача. Сервіс застосовує набір параметрів, які зменшують ймовірність детектування автоматизації: вимикається атрибут navigator.webdriver, задаються типові мови та плагіни браузера. Також блокується завантаження рекламних і трекінгових скриптів (Google Analytics, Facebook), що знижує ризик виконання стороннього JavaScript-коду.

Після створення об'єкта PageExtractor і виклику методу fetch() послідовність дій виглядає так:

1. Ініціалізується браузер Chrome з параметрами безпеки.
2. Вебсторінка завантажується, після чого виконується симуляція руху миші, що знижує ймовірність блокування через захист від ботів.
3. Отримується HTML-код сторінки, фінальна адреса (з урахуванням редіректів) і cookies.
4. Знімається повноекранний скріншот, стискається зображення до безпечного розміру та конвертується у формат Base64.
5. HTML передається у функцію, де за допомогою BeautifulSoup виділяється текст, кількість форм, паролів, скриптів, зовнішніх доменів і інших структурних елементів.
6. Зібрані дані формують єдину структуру результату, що включає текст сторінки, метадані та зображення.

Особливою особливістю реалізації є функція, яка штучно генерує події руху курсора на сторінці. Це зменшує ризик блокування автоматизації, роблячи поведінку програми подібною до дій звичайного користувача. Також передбачено функцію, яка обчислює коефіцієнт стиснення скріншоту, щоб скоротити запит до мовної моделі. Також це дає змогу зберегти якість візуального представлення сторінки при мінімальному навантаженні на пам'ять.

Після вилучення структурованих даних контент сторінки може бути переданий у Google Gemini API, який аналізує його на наявність фішингових шаблонів: закликів до введення особистих даних, елементів підроблених форм авторизації, логотипів банків або інших ознак шахрайських сайтів. Використання зовнішньої мовної моделі дозволяє підвищити точність класифікації без потреби локального навчання власної ML-моделі.

Блок-схема алгоритму роботи сервісу наведена на рис. 2.6.

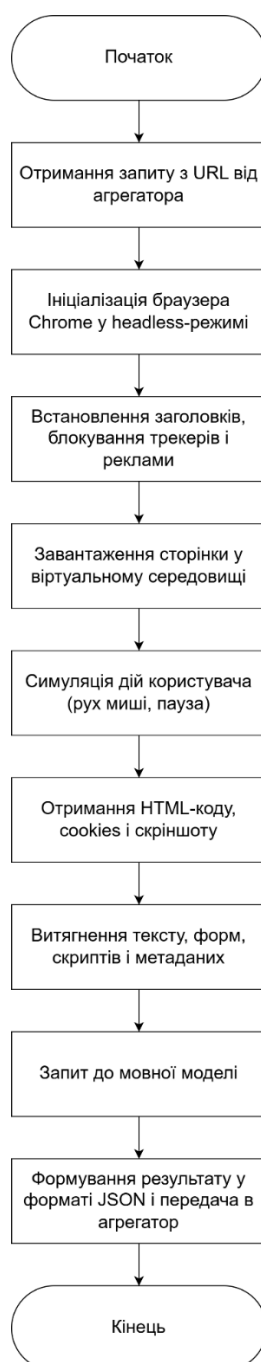


Рис. 2.6. Блок-схема алгоритму роботи сервісу аналізу контенту сторінки

Схема відображає лінійну логіку процесу, де кожен крок має завершення перед переходом до наступного. Усі помилки обробляються винятками `TimeoutException`, `WebDriverException` або загальним `Exception`, після чого браузер закривається у секції `finally`, що забезпечує безпечне завершення процесу навіть при збої під час виконання.

Сервіс описано у `Dockerfile`, який базується на образі `python:3.11-slim` з додатковими компонентами для роботи з `Chrome WebDriver`. У контейнер копіюються файли програми (`app.py`, `page_extractor.py`), встановлюються бібліотеки та створюється середовище виконання із заданим портом. Контейнер функціонує у внутрішній мережі `Docker` і не має відкритих зовнішніх портів.

Розроблений сервіс контентного аналізу забезпечує глибоке дослідження вебсторінок із використанням віртуального середовища та імітації людських дій. Отримані дані містять текстову, структурну та візуальну інформацію, що дозволяє мовній моделі робити висновки про ймовірність фішингу з високою точністю. Лістинг програмної реалізації сервісу контентного аналізу наведено в Додатку В.

Поєднання `Selenium`, `BeautifulSoup` та `Google AI API` створює надійну основу для поведінкового аналізу, уникаючи прямих взаємодій із потенційно шкідливим кодом. У межах загальної системи цей сервіс виступає як центральна ланка контентного рівня, що забезпечує найповніше уявлення про справжню природу перевірюваного вебресурсу.

2.3.6. Реалізація сервісу симуляції дій користувача

Сервіс симуляції дій користувача реалізує поведінковий аналіз вебресурсу шляхом моделювання взаємодії з сайтом у браузерному середовищі. Призначення сервісу полягає у відтворенні дій довірливого користувача, фіксації поведінкових індикаторів та формуванні зведеного висновку про фішингову спрямованість усього ланцюга взаємодії. Сервіс надає два основні ендпоінти, перший формує результати виконаних дій, другий формує підсумковий висновок за зведеним пакетом результатів від інших сервісів.

Реалізацію виконано мовою Python із використанням Flask для прийому HTTP запитів, Selenium для керування браузером, BeautifulSoup для обробки HTML, Pillow для роботи зі знімками екрана, бібліотеки google genai для виклику мовної моделі та requests для взаємодії з сервісом контентного аналізу. Для стабільної роботи застосовано параметри керування тайм-аутами та очікуваннями, а також централізоване логування подій.

Після надходження запиту у ендпоінт симуляції дій користувача з URL сервіс запускає аналіз у декілька спроб, кількість яких задається параметром конфігурації. У межах кожної спроби ініціалізується браузерний драйвер із параметрами безпеки та імітацією людської поведінки, завантажується сторінка, формується повний DOM і знімок екрана, після чого HTML очищується до релевантних елементів. Далі сервіс звертається до мовної моделі з вимогою повернути список дій у структурованому форматі, після чого виконує ці дії у браузері та фіксує індикатори редиректів, нових вікон, алертів і змін DOM. Якщо зміна DOM перевищує поріг, сервіс ініціює повторний контентний аналіз поточного стану сторінки, передаючи HTML та знімок екрана до content_analyzer. Після завершення спроб обирається найінформативніший результат, додаються часові мітки перевірки та термін актуальності, і відповідь повертається агрегатору у форматі JSON.

Блок схема алгоритму роботи отримання результатів симуляції наведена на рис. 2.7.

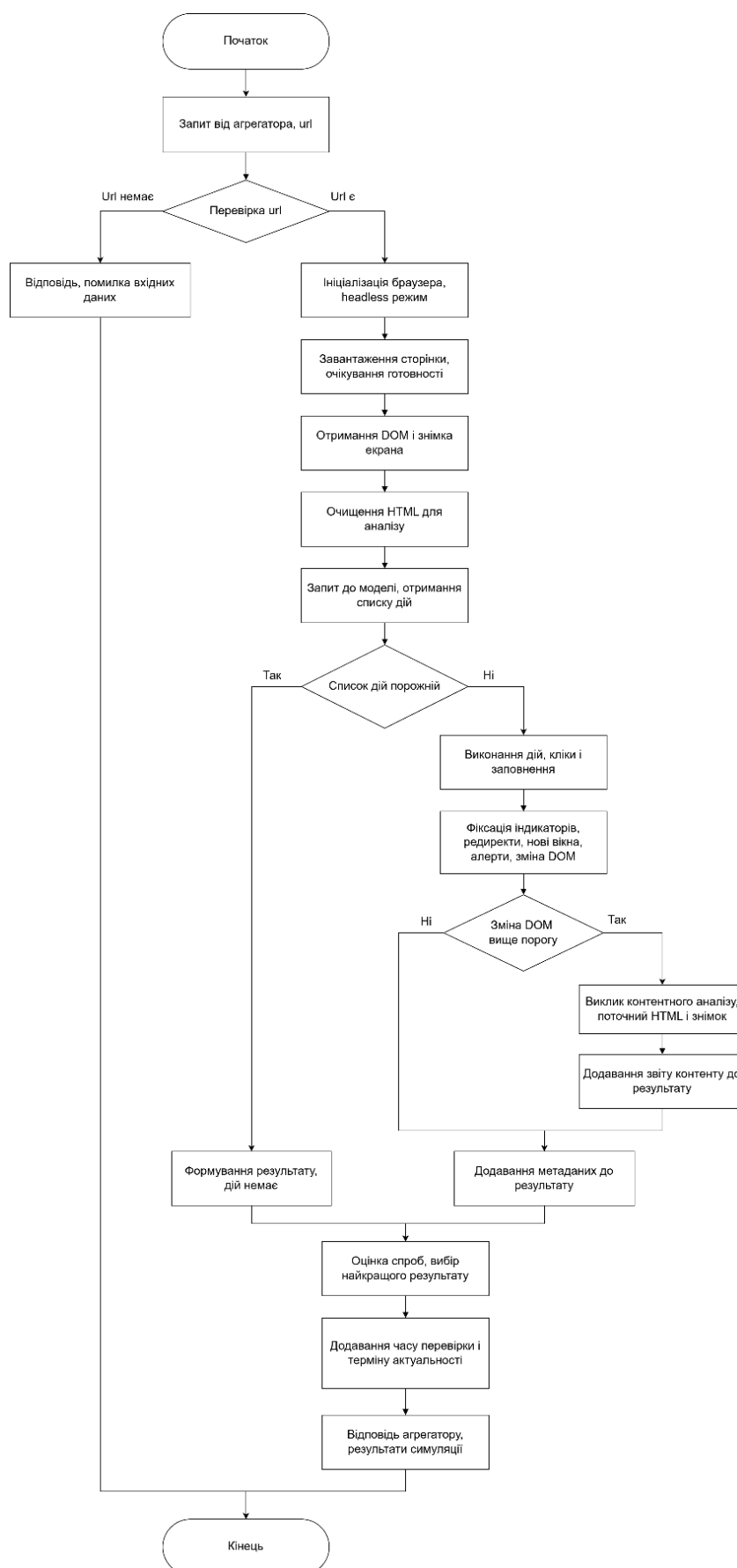


Рис. 2.7. Блок схема алгоритму роботи отримання результатів симуляції дій користувача

Другий ендпоінт призначений для узагальнення всього ланцюга взаємодії. На вхід подається `summary_payload`, сформований агрегатором на основі результатів для початкового URL та пов'язаних сторінок. На основі отриманих даних будується зведений промпт, що містить інформацію про ланцюг переходів і реконструйований таймлайн дій користувача із підкріпленням результатами інших сервісів. Далі виконується виклик мовної моделі з вимогою повернути JSON із полями вердикту та пояснення. Після парсингу відповіді формується структурований результат, додаються часові мітки та термін актуальності, і відповідь повертається агрегатору.

Блок схема алгоритму роботи підсумкового висновку наведена на рис. 2.8.

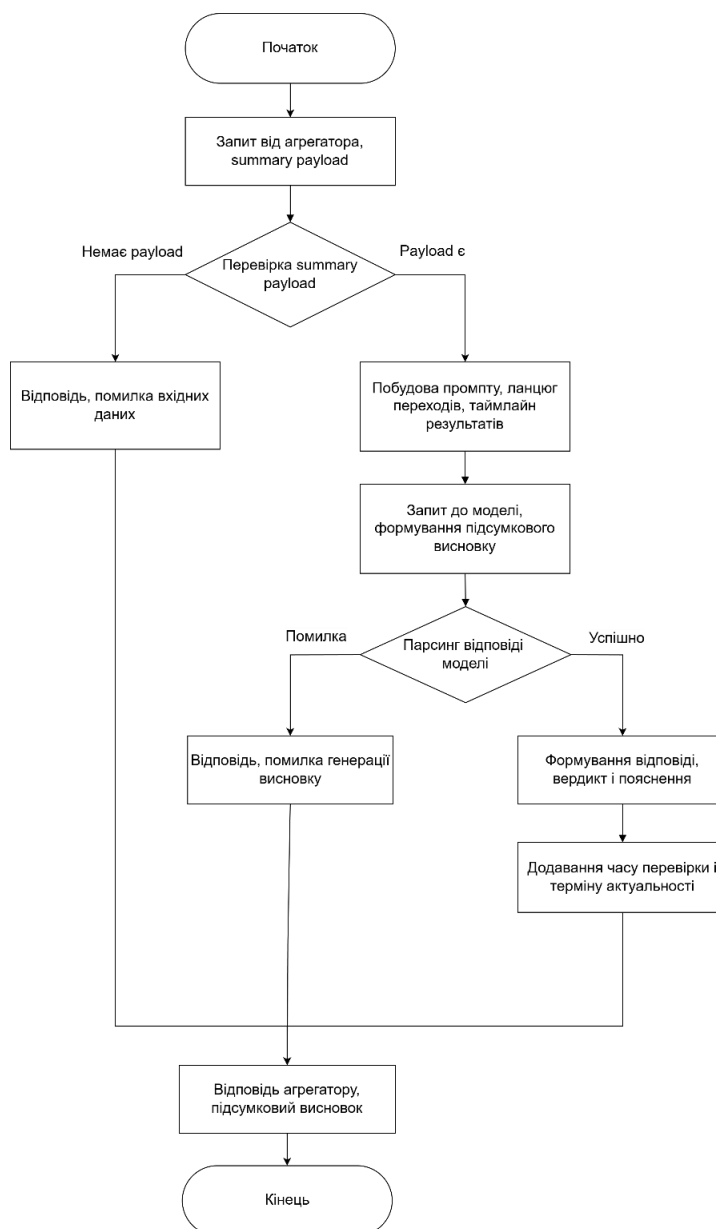


Рис. 2.8. Блок схема алгоритму роботи підсумкового висновку поведінки

У межах загальної архітектури сервіс є незалежним аналітичним модулем, не зберігає стан у власному контейнері та передає результати до агрегатора у форматі JSON. Комбінація симуляції дій у Selenium, фіксації поведінкових індикаторів та залучення мовної моделі для генерації дій і формування підсумкового висновку дозволяє оцінювати не лише статичні ознаки сторінки, а й реальну реакцію вебресурсу на взаємодію, що є критично важливим для виявлення фішингових сценаріїв. Лістинг програмної реалізації сервісу поведінкового аналізу наведено в Додатку Г.

2.3.7. Реалізація сервісу агрегатора результатів

Сервіс агрегатора результатів є центральним координатором системи автоматичного виявлення фішингових сайтів. Його призначення полягає в організації паралельних викликів аналітичних мікросервісів, контролі повторного використання попередніх результатів через сервіс історії перевірок, рекурсивному опрацюванні дочірніх посилань і формуванні підсумкових даних для вебінтерфейсу. Реалізацію виконано мовою Python на базі Flask із використанням ThreadPoolExecutor для паралельної обробки запитів. Для мережевої взаємодії використано бібліотеку requests, а конфігурація контейнера та ключові параметри завантажуються зі змінних середовища.

Агрегатор містить перелік доступних сервісів і їхніх маршрутів, зокрема модулі перевірки у базах загроз, аналізу домену, аналізу контенту, симуляції поведінки та підсумкового поведінкового висновку. Окремо задано адреси сервісу історії для перевірки кешу та оновлення записів, а також коефіцієнти ваг для розрахунку інтегральної оцінки ризику.

Ключова функція `call_service` реалізує три кроки взаємодії з кожним мікросервісом. Спочатку виконується перевірка наявності актуального результату через кеш. Якщо кеш знайдено, агрегатор повертає дані без повторного виклику сервісу. Якщо кешу немає, виконуються реальні запити до цільового сервісу з тайм-аутами, які збільшено для поведінкового аналізу, після чого результат зберігається у базі, якщо відповідь не містить помилки.

Ендпоінт перевірки URL реалізовано як потокову відповідь у форматі подій, що дозволяє вебінтерфейсу отримувати результати частинами у міру готовності. Перед запуском аналізу агрегатор нормалізує URL, перевіряє валідність доменної частини з урахуванням Unicode доменів і лише після цього генерує потік подій аналізу.

Основний алгоритм опрацювання URL побудовано на двох рівнях. На першому рівні для поточного посилання виконується паралельний запуск чотирьох аналітичних сервісів, а для кореневого запиту результати повертаються частинами в потік відповіді. Після отримання результатів сервісу поведінкового аналізу виконується вилучення дочірніх URL з редиректів і нових вікон.

На другому рівні агрегатор виконує обхід дочірніх посилань у ширину з обмеженням глибини, яке задається змінною NUM_REDIRECTS. Під час обходу формується ланцюг переходів і накопичуються проаналізовані результати для кореневої сторінки та дочірніх сторінок. Після завершення обходу агрегатор формує узагальнюючий `summary_payload` і передає його до ендпоінта підсумкового поведінкового аналізу, а отриманий висновок також повертається як фінальна подія потоку.

Блок схема алгоритму перевірки URL наведена на рис. 2.9.



Рис. 2.9. Блок схема алгоритму роботи перевірки URL у сервісі агрегатора результатів

Окремий ендпоінт /score виконує розрахунок інтегральної оцінки ризику на основі результатів, які надійшли з вебінтерфейсу. Інтегральний бал обчислюється за наступною формулою:

$$S = 100 \cdot \frac{\sum_{i \in A} w_i r_i}{\sum_{i \in A} w_i}, \quad r_i \in [0,1] \quad (1)$$

Тут S – інтегральний бал ризику (0–100), A – лише ті модулі, для яких у цьому запуску є дані, w_i – вагові коефіцієнти (з конфігурації), r_i – нормалізований ризик модуля: для репутаційних перевірок це частка підозрілі/всі (а для WebRisk: 0 або 1), для контенту та поведінки – 0/1 за вердиктом. Нормування на $\sum_{i \in A} w_i$ означає, що якщо частина даних відсутня, вага решти пропорційно переналаштовується і підсумок лишається коректним.

Агрегатор перетворює окремі результати у числові компоненти ризику, застосовує вагові коефіцієнти та, у разі відсутності даних від частини сервісів, виконує пропорційне перенормування ваг, щоб підсумкова оцінка була коректною. Після цього значення масштабується до діапазону 0–100 і формується пояснення, яке описує внесок кожного компонента у підсумок.

Таким чином, сервіс агрегатора результатів реалізує повний цикл координації перевірки URL у системі. Він забезпечує паралельність виконання, кешування результатів через сервіс історії, контроль глибини переходів для дочірніх посилань і зручне потокове повернення даних для вебінтерфейсу, а також виконує підсумковий розрахунок інтегральної оцінки ризику на основі вагових коефіцієнтів. Лістинг програмної реалізації сервісу агрегатора результатів наведено в Додатку Д.

2.3.8. Реалізація бази даних та сервісу-інтерфейсу

У системі автоматичного виявлення фішингових сайтів збереження результатів перевірок винесено в окремий рівень даних, який складається з контейнера PostgreSQL та мікросервісу history service. Такий підхід забезпечує повторне використання попередніх результатів, контроль їх актуальності за часовими мітками та ізоляцію доступу до сховища від аналітичних модулів. Усі сервіси взаємодіють із базою не напямую, а через history service, що уніфікує логіку кешування і спрощує супровід.

База даних розгортається як окремий контейнер Postgres у складі Docker Compose, а дані зберігаються у docker volume, змонтований у директорію data всередині контейнера, що дозволяє зберігати історію перевірок між перезапусками середовища.

Сервіс history service підключений до тієї ж внутрішньої мережі Docker, має окремий порт для внутрішніх викликів і запускається після готовності Postgres, що фіксується через залежність та healthcheck.

History service реалізовано на Flask з використанням SQLAlchemy, а конфігурація підключення до PostgreSQL задається змінною HISTORY DATABASE URL, яка за замовчуванням вказує на контейнер db.

Під час старту сервіс ініціалізує структуру таблиць, тому база автоматично готується до роботи без ручних SQL міграцій у межах поточної реалізації.

Структура даних організована навколо двох сутностей, URL та Domain, і набору таблиць результатів для кожного аналітичного сервісу. Для збереження первинних відповідей застосовано тип JSONB, що дозволяє фіксувати повний сирий результат сервісів без втрати деталей, паралельно з ключовими полями, потрібними для швидкого узагальнення та скорингу. Схема структур зображена на рис. 2.10.

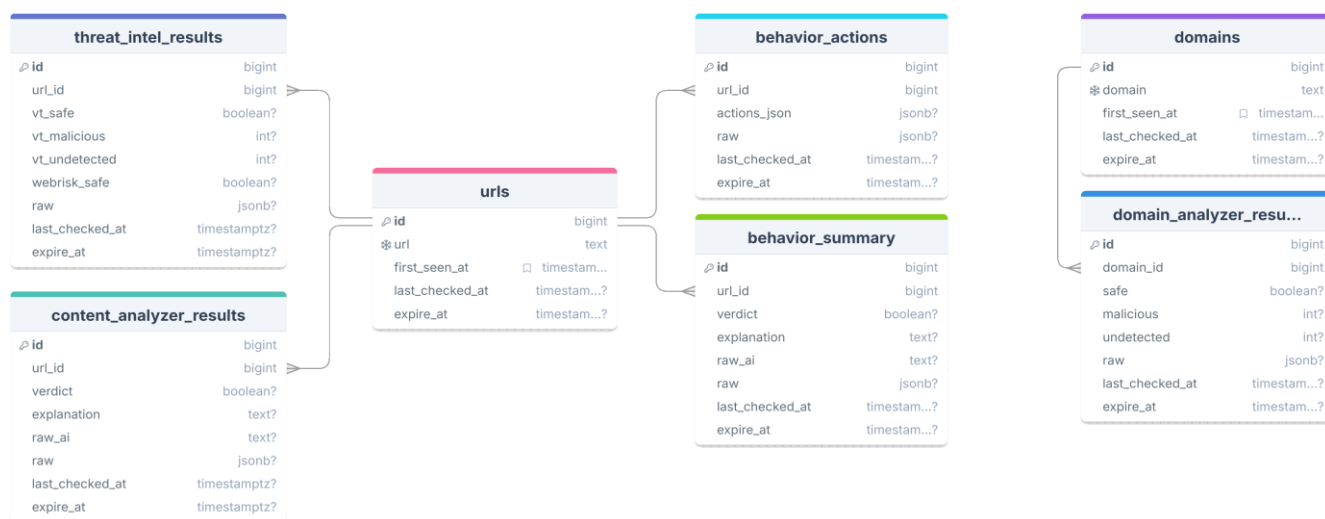


Рис. 2.10. Схема структури бази даних

Функціонально history service надає чотири ендпоінти. Ендпоінт health використовується для перевірки стану, що також задіяно в docker compose healthcheck.

Ендпоінт `check history` повертає кешований результат для пари сервіс і URL, якщо запис існує та не прострочений. При простроченні запис видаляється, а агрегатор отримує сигнал, що потрібен повторний аналіз.

Ендпоінт `update history` приймає результат перевірки, створює або оновлює записи в таблицях результатів, оновлює часові мітки та термін дії, після чого комітить транзакцію.

Ендпоінт `delete url` реалізує каскадне видалення результатів для конкретного URL і додатково очищує домен, якщо він більше не використовується іншими URL у таблиці URLs.

Окремою особливістю є розділення логіки кешування для URL рівня і доменного рівня. Для `domain analyzer`, `history service` виділяє домен із URL, шукає його у таблиці `Domains` та читає результат з `Domain analyzer results`, а для інших сервісів працює через таблицю `URLs` і відповідні таблиці результатів.

У взаємодії з іншими компонентами `history service` використовується агрегатором як проміжний шар кешування. Агрегатор спочатку викликає `check history`, і лише за відсутності актуального запису ініціює обчислення в аналітичних сервісах, після чого зберігає отримані результати через `update history`. Це дозволяє зменшити кількість звернень до зовнішніх API і повторних запусків Selenium, що є найбільш ресурсоємними операціями в системі.

Таким чином, реалізація PostgreSQL як окремого контейнера та винесення доступу до даних у `history service` забезпечують керовану історію перевірок, узгоджене кешування для всіх мікросервісів та підтримку сценаріїв повторної перевірки при завершенні терміну актуальності результатів. Лістинг програмної реалізації сервісу історії перевірок наведено в Додатку Е.

2.3.9. Реалізація демонстраційного веб-інтерфейсу

Демонстраційний вебінтерфейс призначений для взаємодії користувача із системою автоматичного виявлення фішингових сайтів та візуалізації результатів роботи мікросервісів у режимі одного запиту. Інтерфейс реалізовано на базі Streamlit, що дозволяє швидко організувати форму вводу URL, запуск перевірки та

динамічне оновлення блоків результатів без побудови окремого фронтенд застосунку.

Реалізація виконана мовою Python та використовує бібліотеки streamlit для побудови сторінки, requests для обміну даними з агрегатором, json для парсингу відповідей та deep translator для перекладу пояснень, які повертаються англійською мовою. Для захисту інтерфейсу від некоректного відображення помилок застосовано екранування HTML, а для покращення читабельності результатів використано власні HTML блоки з CSS стилями в кожній картці.

Після натискання кнопки перевірки інтерфейс формує запит до сервісу агрегатора. Адреса агрегатора визначається змінними середовища, що дозволяє запускати інтерфейс як у Docker мережі, так і локально. Отримання результатів реалізовано у потоковому режимі, інтерфейс читає події рядок за рядком, декодує JSON і оновлює потрібну картку на сторінці. Для візуалізації ходу виконання застосовано індикатор виконання, а до отримання реальних даних у кожній картці показується повідомлення про опрацювання.

Візуалізація організована у вигляді сітки з окремими блоками для сервісів. Для кожного типу результату передбачено функцію рендерингу, яка перетворює дані у HTML з однаковою логікою оформлення, зокрема кольорову рамку відповідно до рівня ризику, компактні метадані та повідомлення про помилки у разі відмови сервісу.

Після завершення отримання поточних подій інтерфейс формує мінімальний набір значень, необхідний для підрахунку інтегральної оцінки, і надсилає його в агрегатор на обчислення оцінки. Отриманий результат відображається у верхньому підсумковому блоці у вигляді числа від нуля до ста та текстового пояснення формули і внеску компонентів. Такий підхід розділяє завдання, інтерфейс лише відображає, а обчислення та вагові коефіцієнти зосереджені в агрегаторі.

Приклад відображення наведений на рис. 2.11.

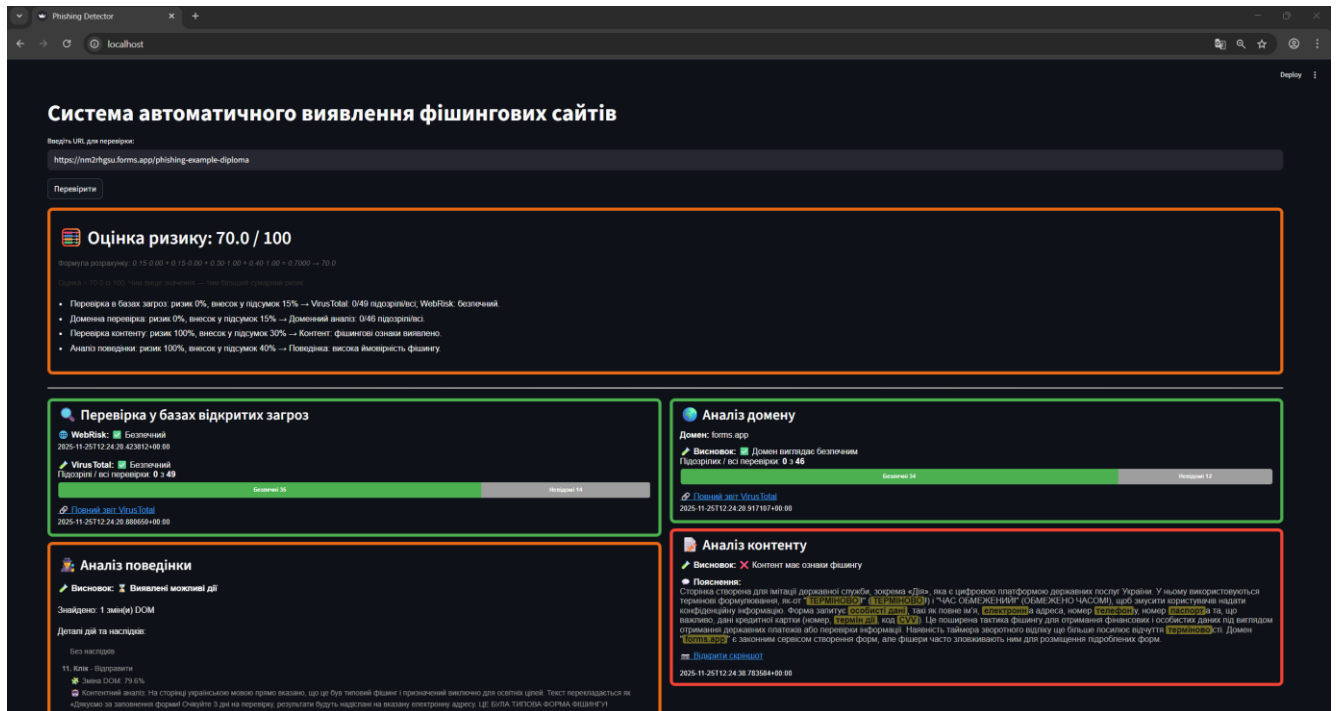


Рис. 2.11. Приклад відображення результатів запиту у вебінтерфейсі

Таким чином, демонстраційний веб інтерфейс забезпечує повний цикл взаємодії користувача із системою, запуск перевірки, потокове відображення результатів по сервісах та фінальне відображення інтегральної оцінки. Рішення на Streamlit є зручним для демонстрації роботи мікросервісної архітектури та тестування сценаріїв, оскільки дозволяє швидко змінювати структуру відображення і логіку викликів без ускладнення клієнтської частини.

ВИСНОВОК ДО РОЗДІЛУ 2

У даному розділі було спроектовано та реалізовано програмну компоненту системи автоматичного виявлення фішингових сайтів на основі баз відкритих загроз і поведінкових ознак, з формалізованою взаємодією між компонентами та контрольованими інформаційними потоками. Запропонована структурна схема і діаграма послідовності підтвердили доцільність централізованої координації через агрегатор і розділення функцій на спеціалізовані мікросервіси, що забезпечує низьку зв'язаність, ізоляцію виконання та можливість незалежного масштабування компонентів у контейнерному середовищі.

Практичним підсумком розділу є реалізований набір сервісів аналітичного рівня, які у сукупності забезпечують багаторівневу перевірку ресурсу. Перевірка у відкритих базах загроз та на рівні домену дає швидкий репутаційний сигнал і знижує ризик пропуску відомих шкідливих ресурсів. Контентний модуль забезпечує безпечне завантаження сторінки у браузерному середовищі та семантичну оцінку вмісту мовною моделлю, що підсилює виявлення фішингових патернів, які не фіксуються репутаційними базами. Поведінковий модуль доповнює систему аналізом реакції сайту на взаємодію користувача, дозволяючи фіксувати редиректи, нові вікна, сповіщення та суттєві зміни сторінки, а також ініціювати повторний контентний аналіз при перевищенні порогу змін, що є важливим для виявлення динамічних сценаріїв фішингу.

Окремим результатом розділу є реалізація механізму збереження та актуалізації даних у PostgreSQL через сервіс історії перевірок, що забезпечує повторне використання непрострочених результатів, зменшення кількості повторних звернень до зовнішніх API та зниження ресурсоємних повторних запусків Selenium. Реалізовано також агрегатор, який паралелізує виклики сервісів, виконує рекурсивне опрацювання пов'язаних посилань із заданим обмеженням глибини, повертає результати порційно та обчислює інтегральну оцінку ризику з пропорційним перерозподілом ваг за відсутності окремих складових. Наявність демонстраційного вебінтерфейсу підтверджує працездатність архітектури як

цілісного рішення та забезпечує зручний спосіб запуску перевірок і візуалізації проміжних та підсумкових результатів.

Таким чином, у межах розділу 2 сформовано завершену програмну основу системи, що відповідає вимогам до модульності, керованості, відтворюваності розгортання та практичної придатності до перевірки URL у напівавтоматичному режимі. Отримані результати створюють передумови для наступного розділу, де доцільно показати прикладні аспекти використання системи, сценарії демонстрації, інтерпретацію отриманих оцінок ризику та програмну документацію для відтворення й інтеграції рішення.

РОЗДІЛ 3

ПРИКЛАДНІ АСПЕКТИ ЗАСТОСУВАННЯ СИСТЕМИ АВТОМАТИЧНОГО ВИЯВЛЕННЯ ФІШИНГОВИХ САЙТІВ НА ОСНОВІ БАЗ ВІДКРИТИХ ЗАГРОЗ ТА ПОВЕДІНКОВИХ ОЗНАК

У цьому розділі подано прикладні аспекти застосування розробленої системи автоматичного виявлення фішингових вебсайтів на основі відкритих баз загроз і поведінкових ознак. Розділ логічно продовжує попередні частини роботи та демонструє практичну перевірку працездатності й відтворюваність отриманих результатів.

Наведено розгортання та типові сценарії перевірки, окремо показано поведінковий аналіз, ланцюги редиректів і кешування, а також описано можливості інтеграції. Завершує розділ програмна документація: API-контракти мікросервісів, формат SSE-повідомлень, правила кешування, експлуатаційні обмеження та параметри конфігурації.

3.1. Демонстрація практичних можливостей розробленої системи

3.1.1. Розгортання та підготовка середовища

У цьому пункті описано підготовку локального середовища для запуску системи, отримання ключів доступу до зовнішніх API та розгортання всіх мікросервісів у контейнерному середовищі Docker Compose. Після виконання наведених кроків середовище є готовим до подальшого тестування функціоналу, яке розглядається у наступному підпункті.

Для розгортання необхідні встановлені Docker Engine, Docker Compose, Git, а також стабільний доступ до Інтернету для викликів VirusTotal API, Google Web Risk API та Google Gemini API. Рекомендовано мати достатній обсяг оперативної пам'яті, оскільки контентний і поведінковий аналіз використовують браузерну автоматизацію. Інсталяційні інструкції для Docker наведено в офіційній документації [23]. Інсталяційні інструкції для Git наведено в офіційному довіднику Git SCM [31].

Система використовує три зовнішні сервіси, тому перед запуском потрібно отримати три ключі.

Google Web Risk API key – ключ формується у Google Cloud Console через механізм API keys, після чого використовується у запитах до ендпоінта `webrisk.googleapis.com`. Офіційний опис підключення Web Risk і механізму викликів API наведено у документації Web Risk [32]. Сторінка з увімкненим Web Risk API для проєкту в Google Cloud зображена на рис. 3.1. Ключ створюється та налаштовується у вкладці «Credentials», що можна бачити на рис. 3.2.

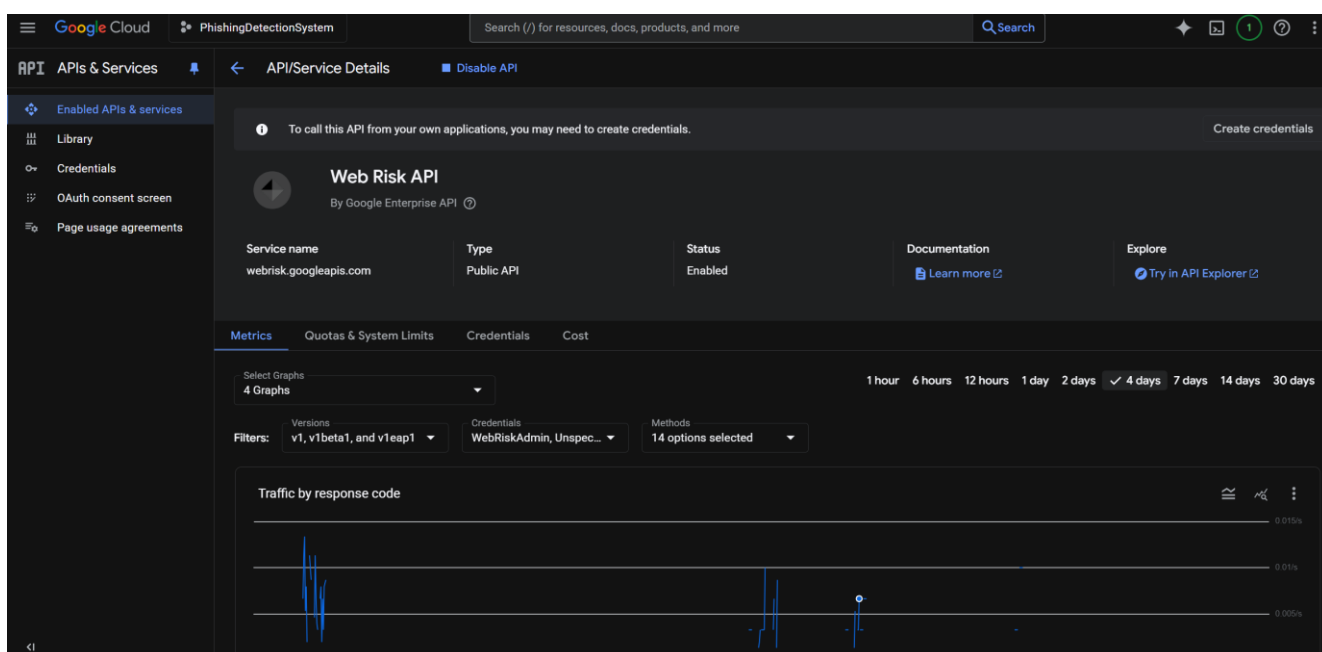


Рис. 3.1. Сторінка увімкненого WebRiskAPI у Google Cloud Console

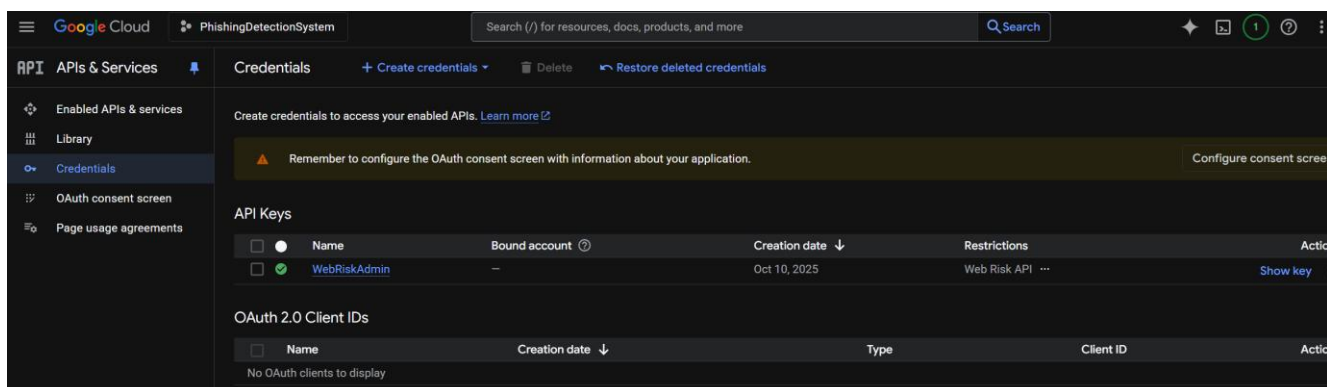


Рис. 3.2. Налаштування API ключів у «Credentials»

VirusTotal API key – ключ видається у кабінеті VirusTotal та використовується для авторизації запитів до API, стандартний варіант передавання ключа реалізується через заголовок. Спочатку треба зареєструватися на сторінці VirusTotal та отримати API-ключ у вкладці меню «API key» [33], сторінка цієї вкладки зображена на рис. 3.3. Можна бачити також обмеження по використанню безкоштовної версії.

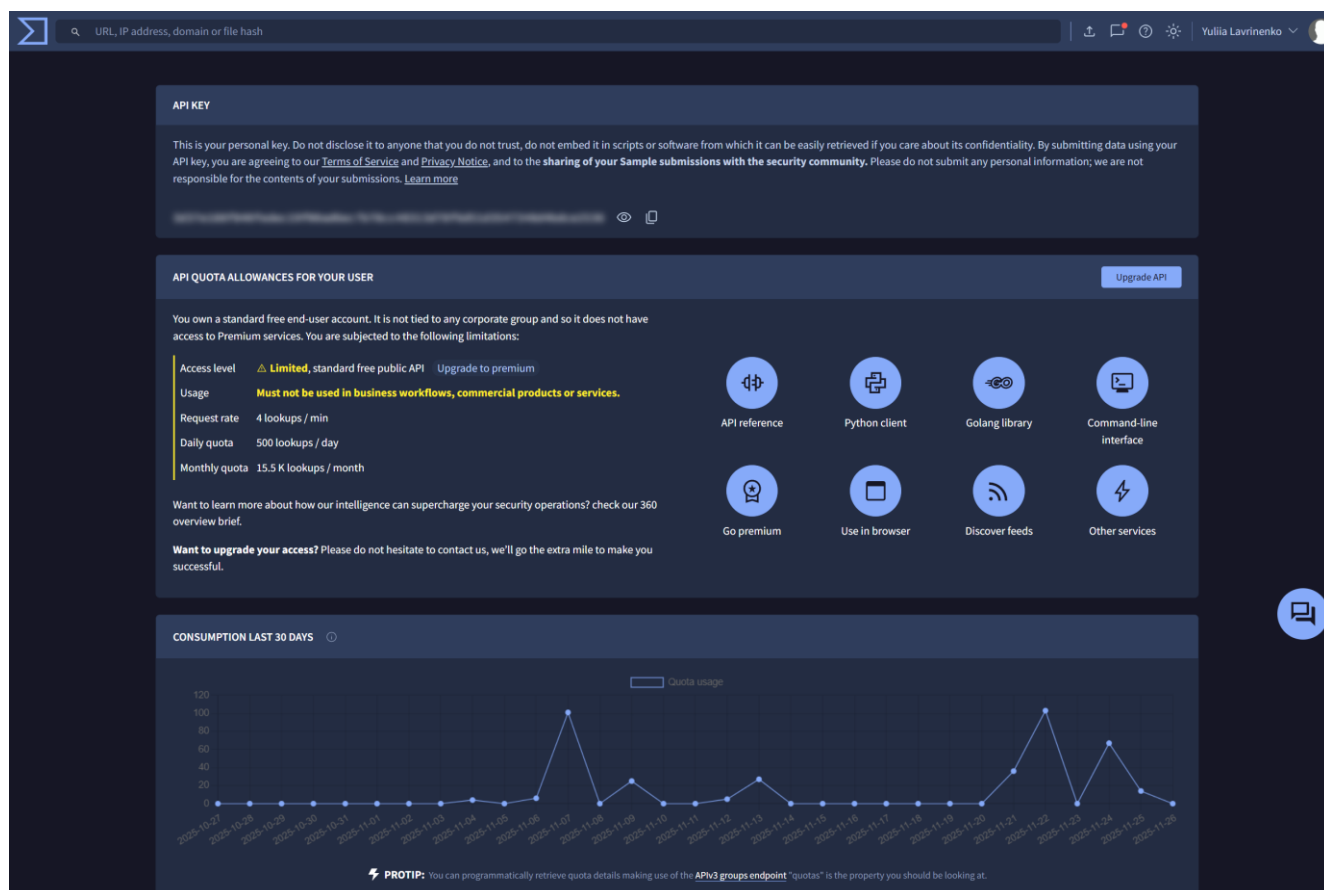


Рис. 3.3. Сторінка VirusTotal з згенерованим ключем

Google AI Studio API key для Gemini - ключ створюється в Google AI Studio та використовується для доступу до Gemini API, ключ передається як параметр у бібліотеці або через інші підтримувані механізми автентифікації. Посилання на сторінку отримання ключа і приклади автентифікації наведені у матеріалах Gemini API [28]. Створення та налаштування ключів відбувається у вкладці «API keys» (рис. 3.4); загальна панель проєкту, де можна слідкувати за обмеженнями та використаними моделями зображена на рис. 3.5.

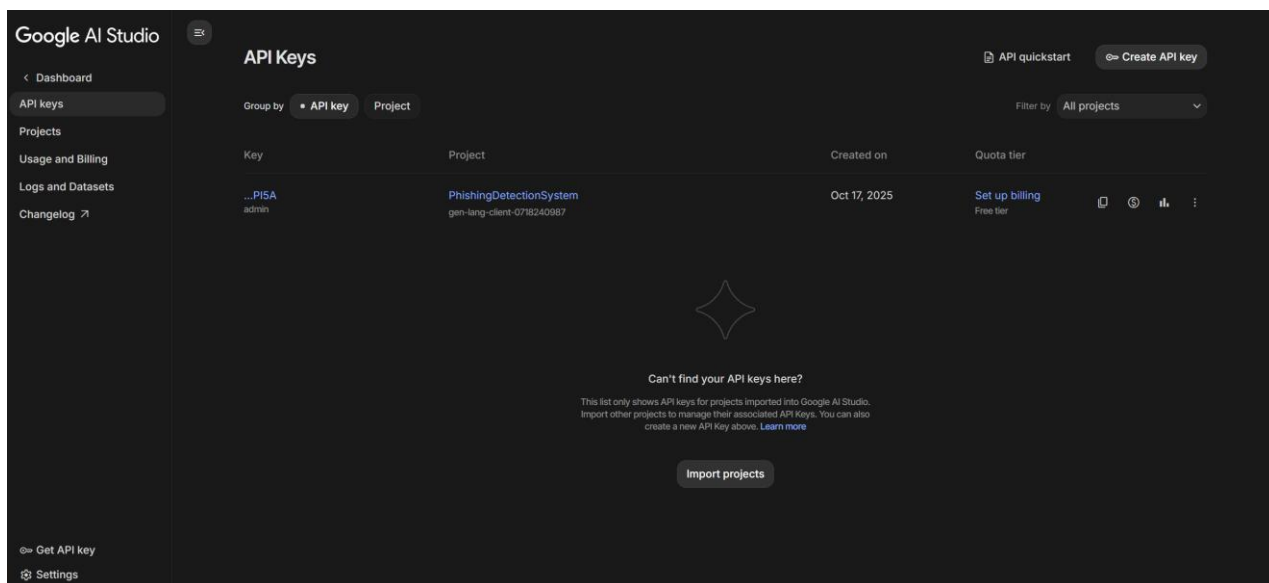


Рис. 3.4. Сторінка Google AI Studio, вкладка API keys

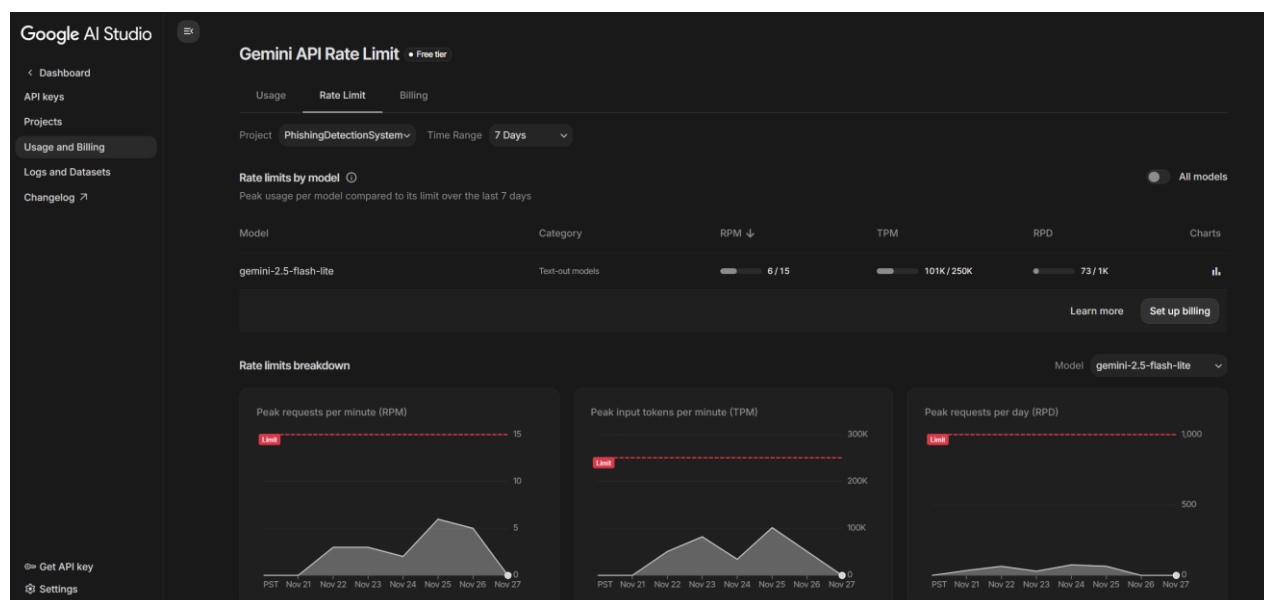


Рис. 3.5. Сторінка перегляду лімітів на використання моделей проекту

Далі виконується отримання вихідного коду, налаштування конфігурації та запуск контейнерів у Docker Compose.

Репозиторій проекту розміщено на GitHub за посиланням <https://github.com/lavrinenkoll/PhishingDetectionSystem>.

Для завантаження коду необхідно виконати клонування репозиторію та перейти в каталог проекту:

```
git clone https://github.com/lavrinenkoll/PhishingDetectionSystem.git
cd PhishingDetectionSystem
```

Після клонування потрібно підготувати файл змінних середовища. У репозиторії наявний шаблон `.env.example`, який копіюється у `.env`, після чого в ньому задаються ключі доступу до зовнішніх API та інші параметри конфігурації. У файлі `.env` необхідно заповнити як мінімум такі значення:

```
WEBRISK_API_KEY=YOUR_WEBRISK_API_KEY_HERE
```

```
VT_API_KEY=YOUR_VIRUSTOTAL_API_KEY_HERE
```

```
GOOGLE_AI_API=YOUR_GOOGLE_AI_API_KEY_HERE
```

Перед запуском контейнерів необхідно переконатися, що Docker Engine або Docker Desktop запущений. У разі коректної роботи Docker команди `docker --version` та `docker compose version` повертають встановлені версії:

```
docker --version
```

```
docker compose version
```

Запуск усіх мікросервісів виконується через Docker Compose. Команда нижче збирає образи проєкту та піднімає контейнери у фоновому режимі:

```
docker compose up -d --build
```

Після запуску рекомендується перевірити, що всі контейнери створені та знаходяться у стані `running`, а також що база даних пройшла ініціалізацію:

```
docker compose ps
```

За потреби можна переглянути логи старту сервісів, щоб переконатися у відсутності помилок конфігурації, відсутності ключів або проблем із підключенням до бази даних:

```
docker compose logs -f
```

Після успішного підняття контейнерів середовище вважається підготовленим до подальшого тестування. На рис. 3.6. наведений приклад успішного запуску усіх контейнерів у застосунку Docker Desktop. Вебінтерфейс буде доступний за посиланням `http://localhost/`. Додатково на `http://localhost:5050/` запущений PgAdmin для перегляду та взаємодії з базою даних.

☐	Name ↓	Container ID	Image	Port(s)	Actions
☐	☒ phishingdetectionsystem	-	-	-	☒ ⋮
☐	☒ ui-service	6d5f8a289b6c	phishingde	80:80 ↗	☒ ⋮
☐	☒ threat-intel	0daf4bbe723c	phishingde	8001:8001 ↗	☒ ⋮
☐	☒ risk-aggregator	db6665c1fe80	phishingde	8000:8000 ↗	☒ ⋮
☐	☒ postgres	64e0ef6e9d87	postgres:11	5432:5432 ↗	☒ ⋮
☐	☒ pgadmin	1743460a948c	dpage/pgadmin4	5050:80 ↗	☒ ⋮
☐	☒ history_service	03a9ce186560	phishingde	8005:8005 ↗	☒ ⋮
☐	☒ domain-analyzer	10fa4ac88453	phishingde	8002:8002 ↗	☒ ⋮
☐	☒ content-analyzer	cf54644fbf0a	phishingde	8003:8003 ↗	☒ ⋮
☐	☒ behavior-analyzer	fa9991d17c50	phishingde	8004:8004 ↗	☒ ⋮

Рис. 3.6. Успішний запуск проєкту, вигляд у застосунку Docker Desktop

3.1.2 Демонстрація сценарію перевірки

У цьому підпункті демонструється робота системи на чотирьох типових сценаріях, коли кінцевий вердикт формується різними групами ознак. Для кожної перевірки використовується один і той самий ланцюг взаємодії, користувач вводить URL у демонстраційному вебінтерфейсі, інтерфейс ініціює запит до агрегатора, агрегатор паралельно викликає аналітичні сервіси, повертає результати порційно, після отримання всіх блоків інтерфейс надсилає дані на ендпоінт скорингу та відображає інтегральну оцінку ризику.

Перед початком демонстрації доцільно зафіксувати стартові умови, усі контейнери запуснені, ключі доступу налаштовані в `.env`, база даних доступна, для першого прогону кожного URL записів у кеші ще немає, що примушує систему виконати повний цикл аналізу. На рис. 3.7 наведено стартовий стан інтерфейсу перед запуском перевірки, з полем введення та кнопкою старту.

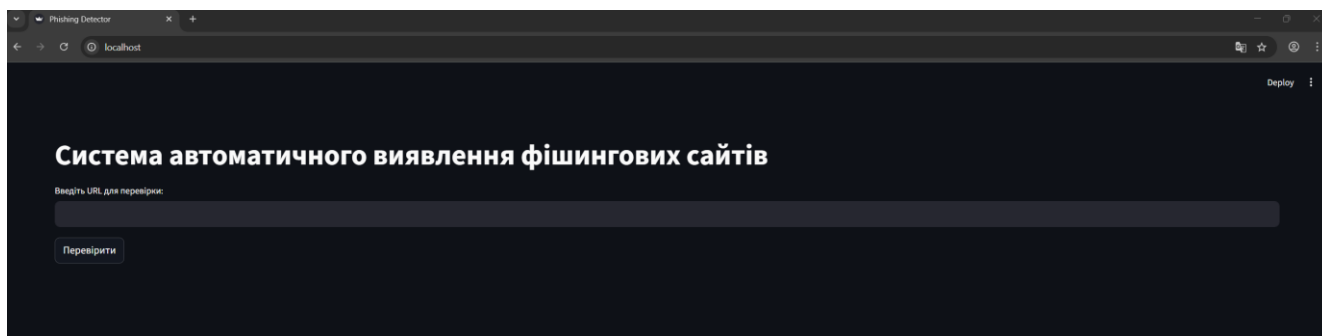


Рис. 3.7. Стартовий стан вебінтерфейсу

У першому сценарії використовується URL ресурсу, що не має ознак компрометації (сайт розкладу). Після натискання кнопки старту інтерфейс починає отримувати події потоку, картки сервісів оновлюються незалежно одна від одної. У блоці перевірки посилань очікується відсутність спрацювань у Google Web Risk та переважання безпечних категорій у зведенні VirusTotal. У блоці аналізу домену очікується домінування безпечних категорій серед обраних рушіїв. Контентний аналіз повертає нейтральний висновок, без ознак виманювання даних, підміни бренду або підозрілих форм. Поведінковий аналіз не фіксує відкриття нових вікон, нетипових редиректів або суттєвих змін DOM, підсумковий поведінковий висновок підтверджує безпечний характер взаємодії. Інтегральна оцінка ризику в цьому випадку низька та супроводжується поясненням, що всі компоненти дали безпечний результат. На рис. 3.8 можна бачити підсумковий вигляд інтерфейсу з усіма зеленими індикаторами та фінальним низьким балом.

Сценарій показав, що сайт класифікується як безпечний, ризик 0 з 100, бо WebRisk не знайшов загроз, VirusTotal дав 0 з 49 для URL і 0 з 46 для домену, контент не містить фішингових ознак, а поведінковий аналіз не зафіксував редиректів, нових вікон, сповіщень і суттєвих змін. В інтерфейсі можна натиснути «Повний звіт VirusTotal» для URL і домену, «Відкрити скріншот» у контентному аналізі, а також переглянути деталі виконаних дій та підсумковий блок поведінки.

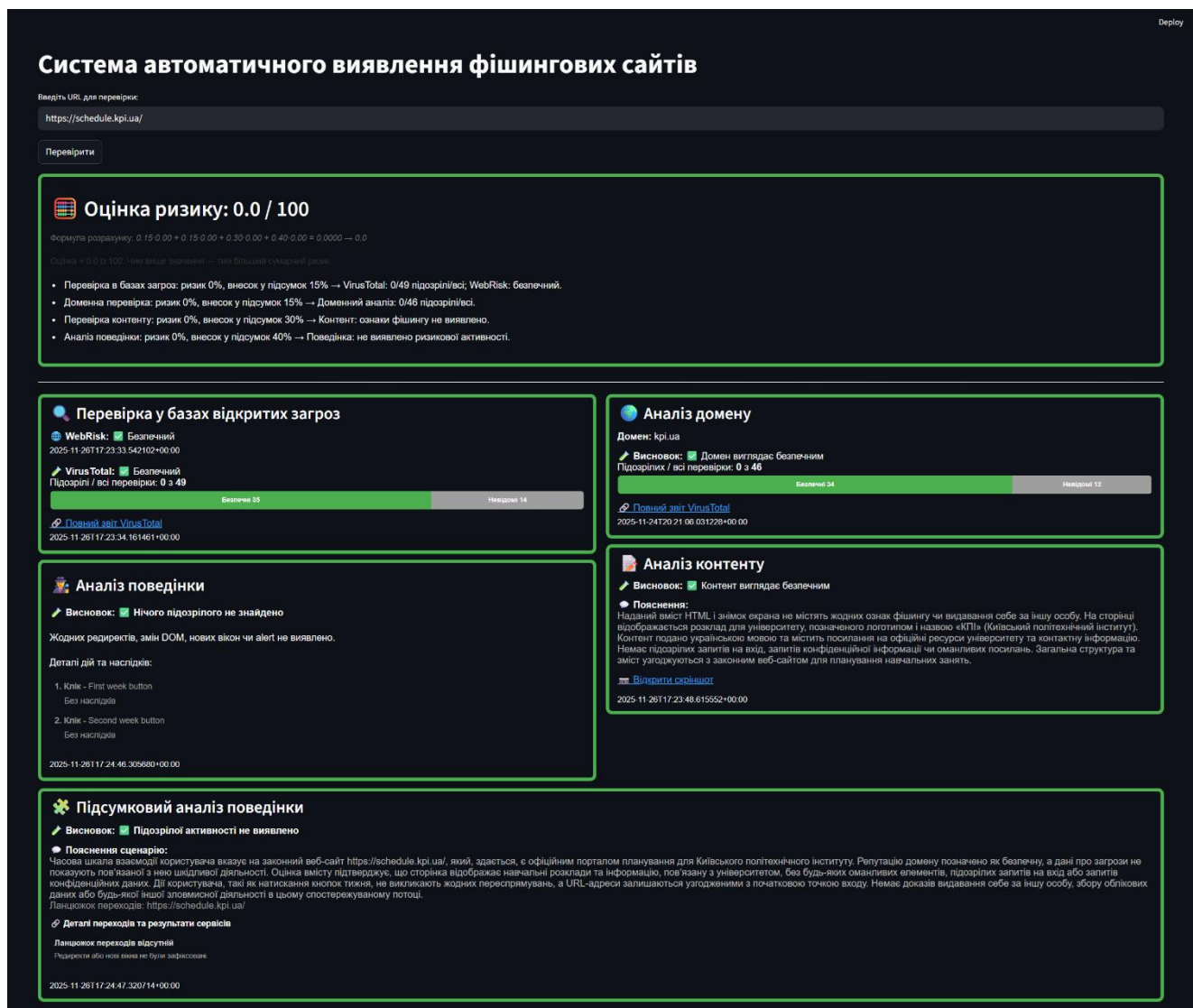


Рис. 3.8. Результат аналізу для безпечного сайту

Другий сценарій демонструє явний фішинговий ресурс, який було взято з бази PhishTank, і який дає негативні сигнали вже на репутаційному рівні. Для нього інтегральна оцінка ризику склала 83.2 з 100, оскільки Web Risk позначив посилання як небезпечне, а VirusTotal та доменний аналіз повернули підозрілі спрацювання, 3 з 49 для URL та 16 з 46 для домена, що підсилює висновок про компрометацію.

На контентному рівні виявлено ознаки підробленої сторінки входу під бренд банку, з полями для введення ідентифікатора та пароля, а поведінковий аналіз підтвердив ризиковий сценарій взаємодії, дії користувача призводять до редиректу на службову сторінку завантаження, після чого підсумковий модуль поведінки формує негативний вердикт про скоординовану фішингову активність.

На рис. 3.9 наведені результати з високим ризику та помаранчевими/червоними індикаторами в блоках перевірки у базах загроз і аналізу домену, разом із негативними блоками контенту та поведінки. У блоці дій можна бачити виконані дії та їх наслідки – остання спричинила редирект, тому у підсумковому блоці редирект перевірявся також усіма сервісами.

Система автоматичного виявлення фішингових сайтів

Введіть URL для перевірки:

<https://logowanipkko24pl.com/>

Перевірити

Оцінка ризику: 83.2 / 100

Формула розрахунку: $0.15 \cdot 0.53 + 0.15 \cdot 0.35 + 0.30 \cdot 1.00 + 0.40 \cdot 1.00 = 0.6318 \rightarrow 63.2$

Оцінка = 83.2 із 100. Чим вище значення, тим більший сукупний ризик.

- Перевірка в базах загроз: ризик 53%, внесок у підсумок 15% → VirusTotal: 3/49 підозріли/всі; WebRisk: небезпечний.
- Доменная перевірка: ризик 34%, внесок у підсумок 15% → Доменний аналіз: 16/46 підозріли/всі.
- Перевірка контенту: ризик 100%, внесок у підсумок 30% → Контент: фішингові ознаки виявлено.
- Аналіз поведінки: ризик 100%, внесок у підсумок 40% → Поведінка: висока ймовірність фішингу.

Перевірка у базах відкритих загроз

WebRisk: ❌ Небезпечний
2025-11-26T17:31:38.855663+00:00

VirusTotal: ✅ Безпечний
Підозріли / всі перевірки: 3 з 49

Повний звіт VirusTotal
2025-11-26T17:31:37.251628+00:00

Аналіз домену

Домен: logowanipkko24pl.com

Висновок: ⚠️ Домен може бути ризиковим
Підозрілих / всі перевірки: 16 з 46

Повний звіт VirusTotal
2025-11-24T13:26:18.632107+00:00

Аналіз поведінки

Висновок: ⚠️ Виявлені можливі дії

Знайдено: 1 редирект(ів)

Деталі дій та наслідків:

- Без наслідків
- Заповнення - Password
Без наслідків
- Клік - Remember my User ID
Без наслідків
- Клік - Login button
Редирект: <https://logowanipkko24pl.com/> → <https://logowanipkko24pl.com/loading.html>

2025-11-26T17:33:47.532903+00:00

Аналіз контенту

Висновок: ❌ Контент має ознаки фішингу

Пояснення:
Сторінка створена для імітації порталу входу DBS Bank. HTML містить поля введення «Ідентифікатор користувача» та «Пароль», типові для збору облікових даних. Заголовок сторінки: «DBS - Вхід», а вміст чітко вказує «Ласкаво просимо до DBS Bank». Однак домен «logowanipkko24pl.com» є дуже підозрілим і не схожий на законний домен банку. На наданому знімку екрана також показано логотип, схожий на «DBS PayLah!», який є послугою DBS Bank, що ще більше підсилює спробу видавання себе за іншу особу. Комбінація підробленого домену, форми входу та використання елементів брендування переконливо свідчить про спробу фішингу з метою вкрадення облікових даних користувача.

Відкрити скріншот
2025-11-26T17:31:53.686514+00:00

Підсумковий аналіз поведінки

Висновок: ❌ Виявлено скоординовану фішингову активність

Пояснення сценарію:
Володарка користувача чітко свідчить про спробу фішингу. Початкова URL-адреса, <https://logowanipkko24pl.com/>, містить сторінку, створену для імітації порталу входу в DBS Bank, про що свідчить заголовок сторінки «DBS - Вхід» і вміст, що говорить «Ласкаво просимо до DBS Bank». Важливо те, що на сторінці представлено поля введення для «Ідентифікатора користувача» та «Пароля», які безпосередньо запитують конфіденційні облікові дані. Домен «logowanipkko24pl.com» має погану репутацію, позначений як шкідливий і не схожий на законний банківський домен. Після того, як користувач ввів свої облікові дані та натиснув «кнопку входу», він був перенаправлений на сторінку завантаження за адресою <https://logowanipkko24pl.com/loading.html>. Хоча сама ця сторінка завантаження не містить додаткових елементів збору облікових даних, попередні дії на зловминому домені переконливо свідчать про те, що введені облікові дані користувача були перехоплені з зловминою метою. Більш потім організовано, щоб обманом змусити користувачів розкрити свою реєстраційну інформацію, імітуючи надійну службу на шахрайському домені.

Ланцюжок переходів: <https://logowanipkko24pl.com/> → <https://logowanipkko24pl.com/loading.html>

Деталі переходів та результати перевірок

Результат 1: <https://logowanipkko24pl.com/> → <https://logowanipkko24pl.com/loading.html>

Подія: клік «Login button»

- ⚠️ Домен: домен може бути ризиковим (logowanipkko24pl.com) (16/46 підозрілих/всі перевірки)
- ❌ Перевірка в базах: URL має ознаки небезпечності (14/49 підозрілих/всі перевірки)
- ⚠️ Контент: наданий контент не виявлено
- Пояснення: Наданий HTML код знімає екран зображень простої сторінки завантаження, яка нібито передає номер мобільного телефону для «PayLah!». Наявні видимі поля для введення конфіденційної інформації, наприклад, паролів чи номерів кредитних карток, а також жодних ознак безпеки. На сторінці відсутні будь-які чіткі ознаки фішингу або спроби видавання себе за іншу особу на основі наданої інформації.
- ❌ Скріншот контенту
- ⚠️ Поведінка: підозрілі дії не виявлено

2025-11-26T17:33:49.419027+00:00

Рис. 3.9. Приклад аналізу небезпечного сайту

Третій сценарій демонструє типову ситуацію, коли репутаційні бази ще «не знають» про загрозу, тому URL і домен виглядають безпечними, але ризик виявляється на рівні вмісту сторінки. Перевірки WebRisk і VirusTotal не дали

спрацювань, так само домен forms.app має позитивну репутацію, однак контентний аналіз визначив фішингові ознаки: імітацію державного опитування/виплат, використання термінових формулювань і таймера як тиску на користувача, а також наявність полів для збору чутливих даних (ПІБ, паспорт, телефон, і особливо реквізити банківської). Через це підсумкова оцінка ризику зросла до 70.0/100, і найбільший внесок у формулу дали саме контентний та поведінковий компоненти (обидва з ризиком 100%), тоді як репутаційні модулі залишилися «зеленими».

Важливо, що цей приклад є навчальним: форма створена на платформі forms.app як демонстраційний макет, який імітує фішинговий збір даних під прикриттям «державного» сценарію. Поведінковий модуль додатково підтвердив ризик саме як сценарій соціальної інженерії: він зміг автоматично заповнити поля та зафіксував суттєву зміну сторінки (79.6%) після натискання «Відправити», а підсумковий модуль поведінки сформував негативний вердикт для всього сценарію взаємодії.

На рис. 3.10 можна бачити, що репутаційні блоки залишаються безпечними, але контентний аналіз має негативний вердикт із поясненням (особливо акцент на полях картки та терміновості). Загальний бал 70.0/100, внесок іде саме від контенту та поведінки, попри нульові значення репутаційних перевірок.

Система автоматичного виявлення фішингових сайтів

Введіть URL для перевірки:

<https://nm2fhgsu.forms.app/phishing-example-diploma>

Перевірити

Оцінка ризику: 70.0 / 100

Формула розрахунку: $0.15 \cdot 0.00 + 0.15 \cdot 0.00 + 0.30 \cdot 1.00 + 0.40 \cdot 1.00 + 0.7000 \rightarrow 70.0$

Оцінка ≈ 70.0 із 100. Чим вище значення — тим більший сукупний ризик.

- Перевірка в базах загроз: ризик 0%, внесок у підсумок 15% → VirusTotal: 0/49 підозрілих/всі; WebRisk: безпечний.
- Доменна перевірка: ризик 0%, внесок у підсумок 15% → Доменний аналіз: 0/46 підозрілих/всі.
- Перевірка контенту: ризик 100%, внесок у підсумок 30% → Контент: фішингові ознаки виявлено.
- Аналіз поведінки: ризик 100%, внесок у підсумок 40% → Поведінка: висока ймовірність фішингу.

Перевірка у базах відкритих загроз

WebRisk: Безпечний
2025-11-26T18:02:29.373671+00:00

VirusTotal: Безпечний
Підозрілих / всі перевірки: 0 з 49

Безпечні: 38 | Небезпечні: 14

[Повний завіт VirusTotal](#)
2025-11-26T18:02:30.319991+00:00

Аналіз домену

Домен: forms.app

Висновок: Домен виглядає безпечним
Підозрілих / всі перевірки: 0 з 46

Безпечні: 34 | Небезпечні: 12

[Повний завіт VirusTotal](#)
2025-11-26T18:02:29.766585+00:00

Аналіз поведінки

Висновок: Виявлені можливі дії

Знайдено: 1 зміни(и) DOM

Деталі дій та наслідків:

Без наслідків

11. Клік - Відправити

Зміна DOM: 79.6%

Контентний аналіз: У змісті сторінки українською мовою чітко зазначено: «ЦЕ БУЛА ТИПОВА ФОРМА ФІШИНГУ! РЕЗУЛЬТАТИ НЕ ОБРОБЛЯЮТЬСЯ, НЕ БУДУТЬ НАДАВАТИСЯ ТРЕТІМ СТОРОНАМ. ЦЕ ЛИШЕ ДЛЯ НАВЧАЛЬНИХ ЦІЛЕЙ! Ця чітка відмова від відповідальності вказує на те, що сторінка не є справжньою спробою фішингу, а радше освітньою демонстрацією форми фішингу. Тому на практиці це не вважається спробою фішингу або видавання себе за іншу особу.

[Скажіть контенту](#)

2025-11-26T18:03:55.285396+00:00

Аналіз контенту

Висновок: Контент має ознаки фішингу

Пояснення:
Сторінка створена так, щоб видати державну службу з виплат громадянам, використовуючи термінові формулювання на кшталт «**ТЕРМІНОВО!** Виплати громадянам – перевірка державних виплатів та таймер зворотного відліку для тиску на користувачів. Він запитав конфіденційну особисту інформацію, включаючи повне ім'я, електронну адресу, номер телефону, номер паспорта а та, що важливо, дані кредитної карти (номер, термін дії, код CVV). Це класична тактика фішингу для збирання фінансових і особистих даних під виглядом законного державного процесу. Наявність полів введення для таких конфіденційних даних у поєднанні з оманливою терміновістю та заявою про «перевірку» для «державних платежів» переконливо свідчить про спробу фішингу.

[Відкрити сторінку](#)

2025-11-26T18:02:48.118029+00:00

Підсумковий аналіз поведінки

Висновок: Виявлено скоординовану фішингову активність

Пояснення сценарію:
Взаємодія користувача чітко свідчить про спробу фішингу. Початкова URL-адреса, розміщена на законній платформі форм (forms.app), веде на сторінку, яка імітує державну службу для платежів громадян. На сторінці використовуються термінові слова та таймер зворотного відліку, щоб змусити користувача надати конфіденційну інформацію, що є поширеною тактикою фішингу. Користувачеві пропонується заповнити особисті дані, такі як повне ім'я, контактна інформація та номер паспорта. Важливо те, що форма потім загрузить дані кредитної карти, включаючи номер карти, термін дії та код CVV. Це пряма спроба зібрати фінансову інформацію під виглядом законного процесу перевірки платежів. Відсутність будь-яких перенаправлень означає, що вся оманлива взаємодія відбувається на одній, хоча й шкідливій, сторінці. Дії користувача щодо заповнення всіх цих конфіденційних полів і наразі натискання «Відправити» (Надіслати) підтверджують його участь у фішинговій схемі.

Ланцюжок переходів: <https://nm2fhgsu.forms.app/phishing-example-diploma>

Деталі переходів та результати сервісів

Ланцюжок переходів відсутній
Рідерски або нова вкладка не були зафіксовані.

2025-11-26T18:03:56.916856+00:00

Рис. 3.10. Приклад аналізу сайту безпечного по репутаційним базам, але шкідливого за контентом та поведінкою

Четвертий сценарій показує випадок «динамічного» фішингу, коли стартова сторінка може не мати явних репутаційних спрацювань, але ризик проявляється лише після взаємодії та переходів. Для демонстрації використано URL із бази PhishTank.

На початку перевірки репутаційні модулі виглядають майже нормальними: WebRisk вважає ресурс безпечним, VirusTotal дає лише 1/49 підозрілих спрацювань, а домен у доменному аналізі не має детекцій (0/46). Водночас контентний аналіз уже на стартовій сторінці фіксує ознаки підміни бренду (використання логотипу ВТ на нетиповому домені) та підозрілий «клікабельний» сценарій з єдиною дією, що

веде на зовнішній ресурс, через що компонент контенту одразу формує негативний вердикт.

Ключова частина сценарію – поведінковий аналіз. Під час кліку по кнопці «Click here to listen» сервіс зафіксував редирект, після чого агрегатор автоматично включив дочірню сторінку в обхід і запустив для неї перевірки в межах заданої глибини. Саме на новій сторінці контентний аналіз виявив критичні ознаки фішингу: форму входу під BT з полями для email і пароля (з характерною помилкою «Passworcl»), тобто сторінку збору облікових даних. Далі за сценарієм відбувається ще один перехід на реальний bt.com, що типово використовується для маскування атаки після збору даних – у результаті поведінковий підсумок формує негативний вердикт по всьому ланцюгу взаємодії. Через те, що фішинговий етап з'являється «в середині» переходів, інтегральний бал зростає до 70.2/100 навіть за майже «зелених» репутаційних показників на вході.

На рис. 3.11 наведений аналіз стартової сторінки, можна бачити збільшений блок аналізу поведінки, оскільки редирект (до рівня глибини 2) перевірявся на типові заклики дій. На рис. 3.12 зображений початковий сайт, на рис. 3.13 – сайт, куди редиректить при натисканні на кнопку, система спрацювала вірно, та описала цю поведінку.

Система автоматичного виявлення фішингових сайтів

Введіть URL для перевірки:

<https://gudricfriendly.sellify.store/>

Перевірити

Оцінка ризику: 70.2 / 100

Формула розрахунку: $0.15 \cdot 0.01 + 0.15 \cdot 0.00 + 0.30 \cdot 1.00 + 0.40 \cdot 1.00 = 0.7015 \rightarrow 70.2$

Оцінка = 70.2 із 100. Чим вище значення — тим більший сукупний ризик.

- Перевірка в базах загроз: ризик 1%, внесок у підсумок 15% → VirusTotal: 1/49 підозріли/всі; WebRisk: безпечний.
- Доменная перевірка: ризик 0%, внесок у підсумок 15% → Доменный аналіз: 0/46 підозріли/всі.
- Перевірка контенту: ризик 100%, внесок у підсумок 30% → Контент: фішингові ознаки виявлено.
- Аналіз поведінки: ризик 100%, внесок у підсумок 40% → Поведінка: висока ймовірність фішингу.

Перевірка у базах відкритих загроз

WebRisk: Безпечний
2025-11-26T18:43:27.224966+00:00

VirusTotal: Безпечний
Підозрілі / всі перевірки: 1 з 49

Безпечні 34 | Небезпечні 14

[Повний звіт VirusTotal](#)
2025-11-26T18:43:27.591134+00:00

Аналіз домену

Домен: sellify.store

Висновок: Домен виглядає безпечним
Підозрілих / всі перевірки: 0 з 46

Безпечні 33 | Небезпечні 13

[Повний звіт VirusTotal](#)
2025-11-26T18:43:28.167067+00:00

Аналіз поведінки

Висновок: Виявлені можливі дії

Знайдено: 1 редирект(ів)

Деталі дій та наслідків:

1. Click - Click here to listen button
Редирект: <https://gudricfriendly.sellify.store/> → <https://adaptable-tour-727440.framer.app/>

2025-11-26T18:44:18.908204+00:00

Аналіз контенту

Висновок: Контент має ознаки фішингу

Пояснення:
На сторінці відображається логотип відомої телекомунікаційної компанії BT (British Telecom). Однак доменне ім'я «gudricfriendly.sellify.store» є дуже підозрілим і не відповідає законним доменам BT. Sellify — це платформа, за допомогою якої творці продають продукти, і її часто використовують для фішингу. На сторінці відсутній типовий вміст або послуги, пов'язані з BT, як-от керування обліковим записом або інформація про продукт. Замість цього він представляє одну кнопку «НАТИСНІТЬ ТУТ, ЩОБ СЛУХАТИ», яка посилається на зовнішній домен «adaptable-tour-727440.framer.app». Це переспрямування на непов'язаний і потенційно зловмисний домен у поєднанні з оманливим брендингом переконливо вказує на спробу видавання себе за іншу особу, яка, ймовірно, спрямована на перенаправлення користувачів на фішинговий сайт або розповсюдження зловмисного програмного забезпечення.

[Відкрити скріншот](#)
2025-11-26T18:43:44.813621+00:00

Підсумковий аналіз поведінки

Висновок: Виявлено скоординовану фішингову активність

Пояснення сценарію:
Спостережуваний ланцюжок взаємодії є яскравим прикладом фішингової атаки, призначеної для збору облікових даних і видавання бренду. Початкова точка входу, gudricfriendly.sellify.store, використовує оманливий логотип BT (British Telecom) на субдомейні Sellify, платформі, яку часто використовують для фішингу. Ця сторінка, незважаючи на оманливе брендівання, переспрямовує користувача на adaptable-tour-727440.framer.app. Цей другий домен містить сторінку, яка явно імітує портал входу BT, доповнений полями введення для «Адреси електронної пошти» та неправильно написаного «Паролю». Користувачеві пропонується ввести облікові дані в ці поля. Після надсилання користувач перенаправляється на законний домен bt.com. Остаточне переспрямування на автентичний сайт є значущою тактикою для створення помилкового відчуття законності після того, як конфіденційна інформація вже була захоплена на підробленій сторінці. Комбінація видавання відомого бренду, використання оманливої початкової цільової сторінки, надання фальшивої форми входу для збору облікових даних і подальшого переспрямування на законний сайт переконливо вказує на скоординовану спробу фішингу.

Ланцюжок переходів: <https://gudricfriendly.sellify.store/> → <https://adaptable-tour-727440.framer.app/> → <https://www.bt.com/>

Деталі переходів та результати перевірок

Результат 1: <https://gudricfriendly.sellify.store/> → <https://adaptable-tour-727440.framer.app/>

Подія: клік «Click here to listen button»

- Домен: домен виглядає безпечним (framer.app) (0/46 підозрілих/всі)
- Перевірка в базах: URL виглядає безпечним (0/49 підозрілих/всі)
- Контент: виявлено ознаки фішингу

Пояснення: Наданий HTML-код і знімок екрана вказують на спробу фішингу. На сторінці представлено форму входу з полями «Адреса електронної пошти» і «Пароль» (орфографічна помилка «Пароль»). Наявність форми входу, особливо з полями з помилками, є поширеною тактикою, яка використовується для збору облікових даних користувачів. Відображено логотип BT, але загальний дизайн і «Made in Framer» значок вказує на те, що це не офіційна сторінка входу BT. Для форми не визначено, але намір зібрати конфіденційну інформацію очевидний, що створює ризик фішингу.

[Відкрити скріншот](#)

Результат 2: <https://adaptable-tour-727440.framer.app/> → <https://www.bt.com/>

Подія: клік «Sign in button»

- Домен: домен виглядає безпечним (bt.com) (0/46 підозрілих/всі)
- Перевірка в базах: URL виглядає безпечним (0/49 підозрілих/всі)
- Контент: підозрілих ознак не виявлено

Пояснення: Наданий вміст HTML представляє офіційний веб-сайт BT. Усі поля вказують на законні домени BT (наприклад, bt.com, ee.co.uk, my.bt.com, youtube.bt.com). Немає підозрілих полів введення для збору облікових даних, немає оманливої мови, а загальна структура та вміст відповідають веб-сайту законного істотівця телекомунікацій. Наявність банера зображення щодо файлів cookie є стандартною для законних веб-сайтів.

[Відкрити скріншот](#)

Поведінка: підозрілих дій не виявлено

2025-11-26T18:46:35.782786+00:00

Рис. 3.11. Приклад аналізу сайту, що спровокував редирект

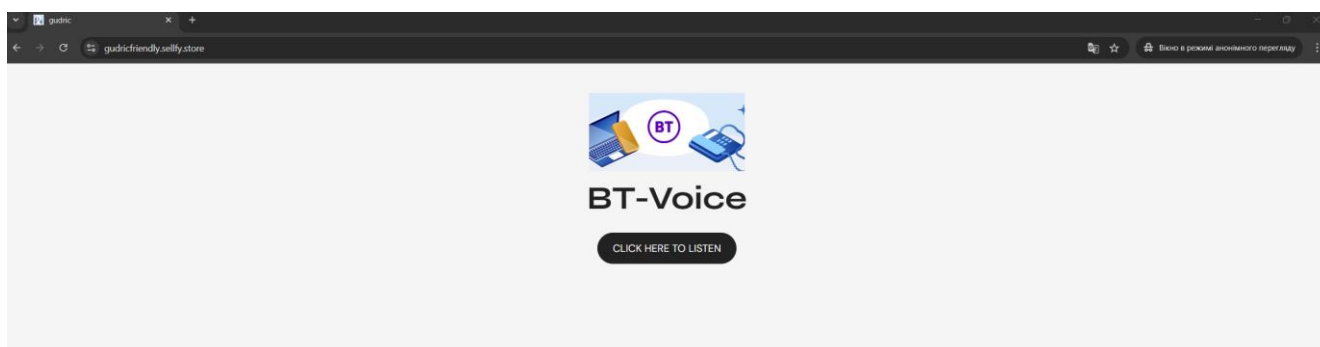


Рис. 3.12. Початкова сторінка фішингового посилання



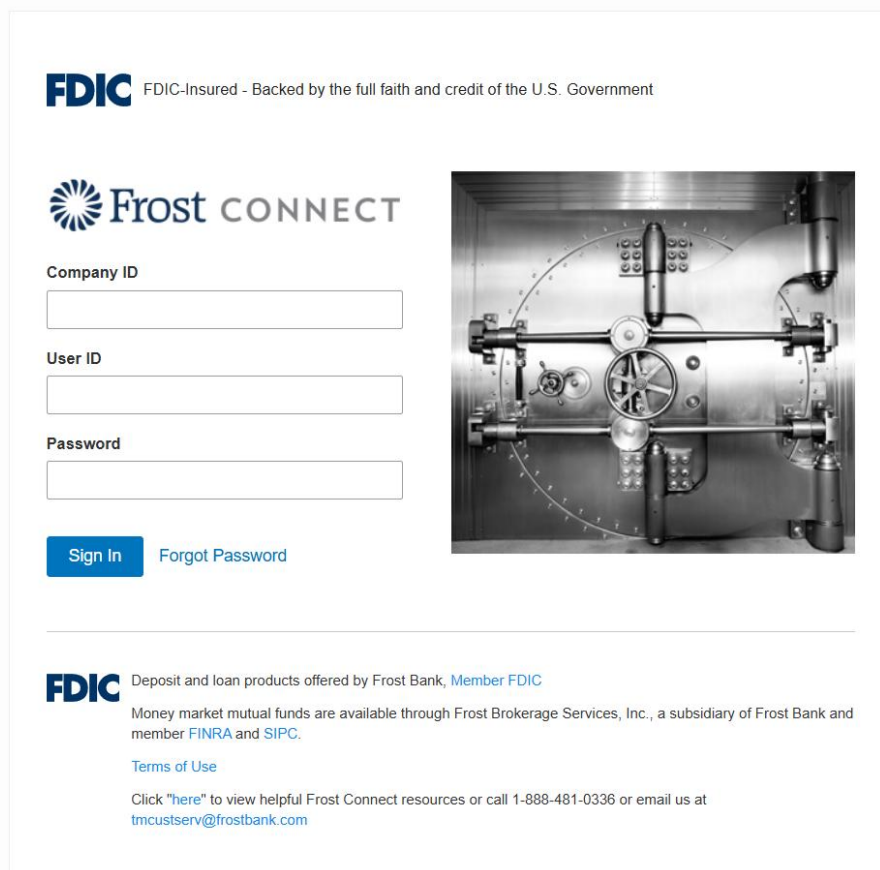
Рис. 3.13. Сторінка після редиректу з імітацією входу в інший сервіс

Підсумовуючи демонстрацію, можна зробити висновок, що система коректно формує кінцевий вердикт у різних умовах: коли загроза одразу підтверджується репутаційними джерелами, коли ризик проявляється на рівні вмісту сторінки за відсутності детекцій у базах, а також коли фішинговий сценарій «розкривається» лише під час взаємодії через редиректи/ланцюжки переходів і дочірні сторінки. Таким чином, комбінація репутаційних, контентних і поведінкових індикаторів забезпечує стійкість перевірки до типових обхідних тактик і дозволяє підвищити надійність висновку в порівнянні з підходом, що спирається лише на один рівень ознак.

3.1.3 Демонстрація поведінкового аналізу і ланцюга переходів

У цьому підпункті демонструється робота поведінкового модуля, який виконує послідовність дій «довірливого користувача» (натискання кнопок, заповнення полів) у браузерному середовищі та фіксує зміни стану сторінки.

Для демонстрації використано ресурс, який імітує фінансову установу та збирає облікові дані у кілька кроків. На рис. 3.14 наведено стартову сторінку сайту з формою входу (поля ідентифікаторів та пароль, брендovanі написи на кшталт «Frost CONNECT»).



FDIC FDIC-Insured - Backed by the full faith and credit of the U.S. Government

Frost CONNECT

Company ID

User ID

Password

[Sign In](#) [Forgot Password](#)

FDIC Deposit and loan products offered by Frost Bank, [Member FDIC](#)
 Money market mutual funds are available through Frost Brokerage Services, Inc., a subsidiary of Frost Bank and member [FINRA](#) and [SIPC](#).
[Terms of Use](#)
 Click ["here"](#) to view helpful Frost Connect resources or call 1-888-481-0336 or email us at tmcustserv@frostbank.com

Рис. 3.14. Початкова сторінка ресурсу із формою входу

Після натискання кнопки входу поведінковий сервіс фіксує зміну URL, тобто редирект на наступний крок сценарію. Це відображається у блоці «Деталі дій та наслідків», де дія («Sign In button») призводить до переходу. На рис. 3.15 наведений цей блок поведінкового аналізу.


Аналіз поведінки

 **Висновок:**  **Виявлені можливі дії**

Знайдено: 1 редирект(ів)

Деталі дій та наслідків:

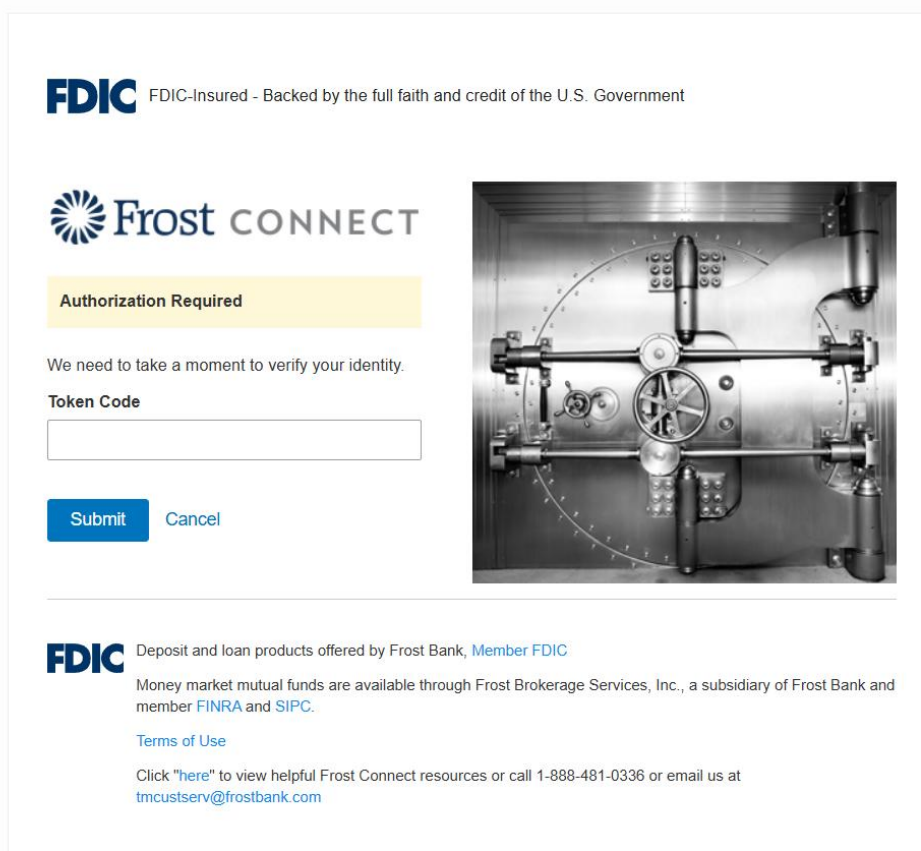
1. Клік - Sign In button

 Редирект: <https://frosttreasuryservice.vercel.app/> → <https://frosttreasuryservice.vercel.app/cde.html>

2025-11-26T19:18:45.994557+00:00

Рис. 3.15. Результат модуля: зафіксований редирект після натискання

На новій сторінці сценарій продовжується запитом додаткового секрету («код маркера»), тобто сайт намагається зібрати ще один тип конфіденційних даних під виглядом «потрібна авторизація/підтвердження особи». Після натискання кнопки відправки відбувається наступний перехід, що створює «завершальний» етап з повідомленням «успішно», характерний для маскуванню атаки після отримання даних. На рис. 3.16 наведений вигляд сторінки із полем токена/маркера та кнопкою підтвердження, як приклад другого кроку соціальної інженерії.



FDIC FDIC-Insured - Backed by the full faith and credit of the U.S. Government

Frost CONNECT

Authorization Required

We need to take a moment to verify your identity.

Token Code

Submit **Cancel**

FDIC Deposit and loan products offered by Frost Bank, [Member FDIC](#)

Money market mutual funds are available through Frost Brokerage Services, Inc., a subsidiary of Frost Bank and member [FINRA](#) and [SIPC](#).

[Terms of Use](#)

Click "[here](#)" to view helpful Frost Connect resources or call 1-888-481-0336 or email us at tmcustserv@frostbank.com

Рис. 3.16. Сторінка запиту «коду маркера» як продовження збору конфіденційних даних після редиректу

На рис. 3.17 наведений вигляд сторінки із повідомленням про «успішну перевірку», оскільки це демонструє завершення ланцюга переходів та типову тактику створення хибного відчуття легітимності після введення даних.

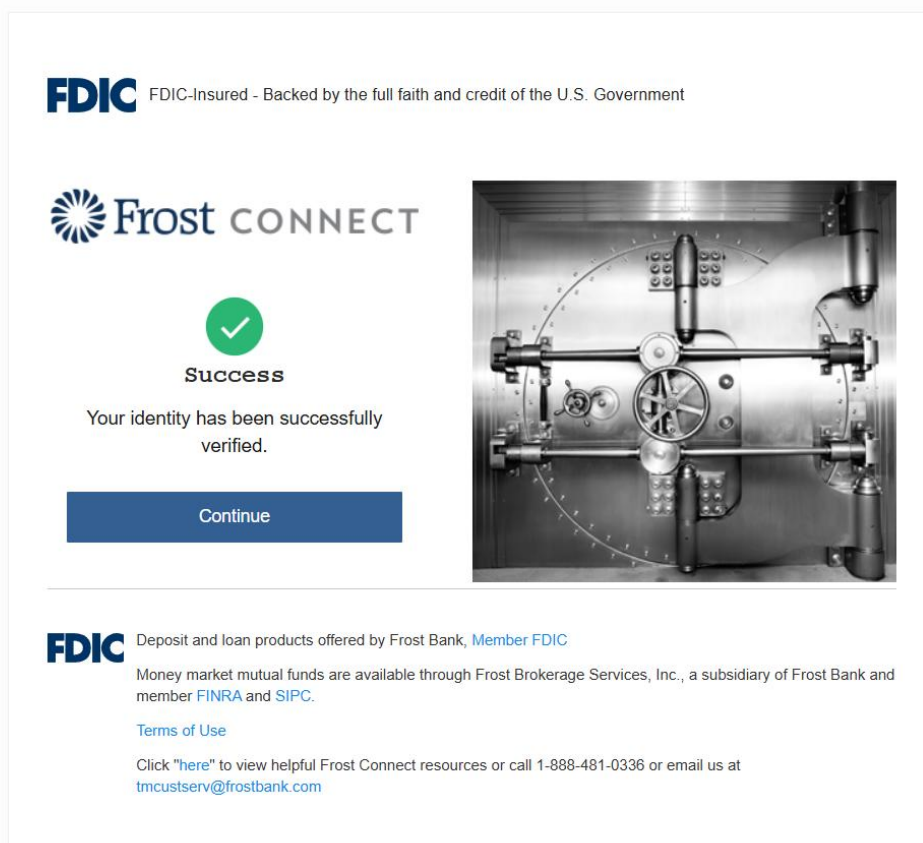


Рис. 3.17. Завершальна сторінка сценарію з повідомленням про «успішну перевірку»

Підсумковий модуль поведінки агрегує спостереження у вигляді ланцюжка переходів та формує негативний вердикт про скоординовану фішингову активність. Важливим є те, що висновок робиться не лише через наявність переходів, а через їхню логіку: послідовний збір різних типів облікових даних у межах одного сценарію та загальна невідповідність середовища легітимному фінансовому порталу (розміщення на домені платформи розгортання). На рис. 3.18 наведений блок із текстовим поясненням та відображеним ланцюжком переходів, оскільки саме він демонструє, як система з'єднує окремі події в цілісний доказовий сценарій. Також на кожен редирект або нове вікно представлений аналіз усіма сервісами, що враховується у підсумку.

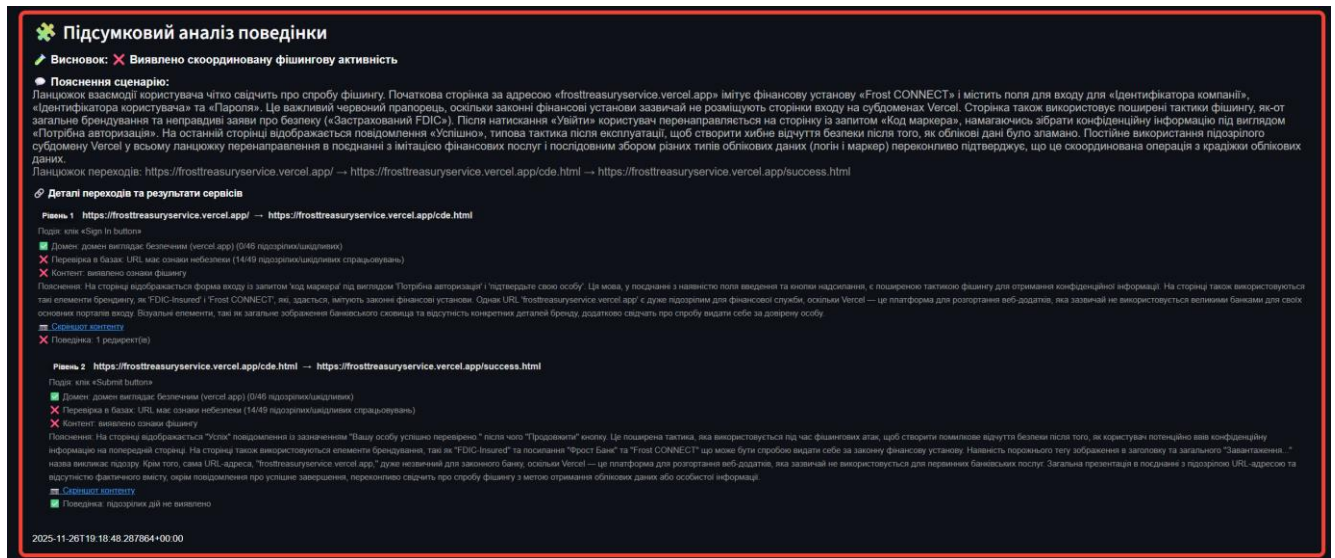


Рис. 3.18. Підсумковий аналіз поведінки: негативний вердикт та ланцюжок переходів з результатами сервісів

Таким чином, приклад підтверджує практичну роль поведінкового аналізу: навіть коли окремі сторінки можуть виглядати «акуратно» з точки зору верстки, саме динаміка взаємодії (переходи між кроками, контекст дій і поетапний збір секретів) дає змогу виділити фішинговий сценарій та відобразити його у вигляді прозорого ланцюга переходів із поясненням причин ризику.

3.1.4 Демонстрація кешування та повторної перевірки

У цьому підпункті показано роботу механізму кешування результатів, реалізованого через модуль історії перевірок і базу даних. Після першого запуску перевірки система зберігає нормалізовані результати для URL і домену разом із часовими мітками останньої перевірки та терміном актуальності запису. Під час повторного запуску (або перевірки іншого URL з тим самим доменом) агрегатор спочатку звертається до сервісу взаємодії з базою даних: якщо дані ще не прострочені, результати повертаються з бази без повторних викликів зовнішніх API, що суттєво зменшує час відповіді та навантаження на ліміти VirusTotal/WebRisk.

Демонстрація проводиться на двох URL, що належать одному домену. Перший запуск виконує повний цикл аналізу, а результати доменної перевірки

зберігаються у БД. На рис. 3.19 показано перший запуск перевірки: у відповідях доменного аналізатора відображається мітка перевірки, а також формується запис у сховищі історії.

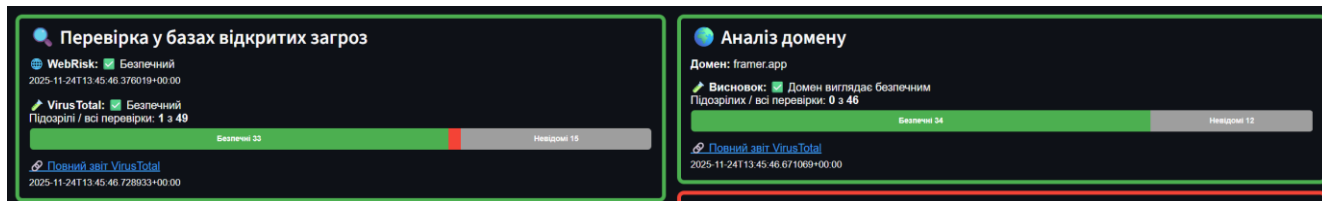


Рис. 3.19. Перший запуск перевірки за заданим доменом

Далі виконується перевірка другого URL з тим самим доменом. Оскільки домен уже присутній у БД і не вийшов за межі встановленого часу, доменний модуль повертає дані з кешу. Це видно за тим, що часова мітка перевірки домену збігається з попереднім запуском першого сайту, на відміну від перевірки посилання. На рис. 3.20 показано повторну перевірку: для домену повертається той самий результат, що й у першому запуску, без повторного запиту до VirusTotal Domain API.

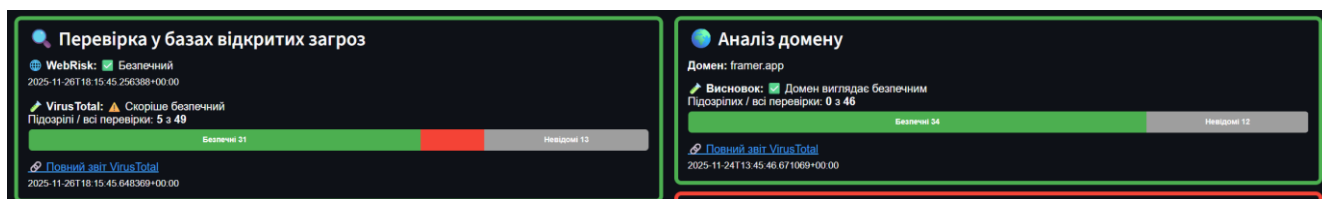


Рис. 3.20. Повторна перевірка іншого URL того самого домену

Ефект кешування відображається також на загальній швидкості роботи інтерфейсу: картка доменного аналізу заповнюється майже миттєво, а виведення підсумкового скорингу відбувається швидше через скорочення кількості зовнішніх викликів.

Таким чином, повторна перевірка ресурсів з однаковим доменом підтверджує коректність механізму кешування: система не дублює дорогі зовнішні запити, використовує збережені результати в межах терміну їх актуальності та забезпечує прискорений відгук при повторних аналізах.

3.2. Практичне застосування та можливості інтеграції системи

Розроблена система може використовуватися як прикладний інструмент для оперативної перевірки підозрілих вебресурсів у типових ситуаціях, коли користувач отримує посилання з пошти, месенджера або соціальних мереж і не має впевненості в його безпечності. Практична цінність підходу полягає у поєднанні кількох рівнів оцінювання: репутаційної перевірки посилання і домену, аналізу вмісту сторінки та поведінкового сценарію взаємодії. Це дозволяє виявляти як відомі фішингові ресурси, так і нові сторінки, що ще не потрапили до репутаційних баз, але містять ознаки соціальної інженерії або активуються після кліків і переходів.

У прикладному використанні систему доцільно застосовувати як сервіс перевірки перед переходом на сайт. У навчальному та організаційному контексті вона може підтримувати підвищення кіберобізнаності: демонструвати типові ознаки фішингу на реальних прикладах, пояснювати, які саме сигнали вплинули на ризикову оцінку, та формувати зрозумілий для користувача висновок. Додатково система придатна для первинного аналізу інцидентів: посилання з повідомлень або звернень користувачів можна проганяти через перевірку й отримувати структурований результат із ланцюгом переходів, знімками екрана та поясненнями, що спрощує подальше розслідування.

З точки зору інтеграції, архітектура мікросервісів і наявність окремого агрегатора дозволяють підключати систему до різних клієнтських інтерфейсів без зміни логіки аналізу. Найпростіший варіант – використання як внутрішнього вебсервісу в локальній або корпоративній мережі, коли користувачі надсилають посилання на перевірку через єдину сторінку або внутрішній портал. Інший сценарій – інтеграція у браузерне розширення: розширення може перехоплювати перехід на сайт, відправляти адресу в систему, а у відповідь показувати попередження або короткий підсумок ризику ще до завантаження сторінки. Також можливий окремий настільний застосунок, який працює як «перевірка перед відкриттям» – користувач вставляє адресу, отримує оцінку ризику та пояснення, після чого приймає рішення про відкриття ресурсу.

Оскільки система має чітко розділені модулі, її можна вбудовувати як компонент у більші рішення з кіберзахисту. Наприклад, модуль перевірки посилань може застосовуватися у поштовому шлюзі або внутрішньому сервісі фільтрації повідомлень, де підозрілі адреси автоматично перевіряються і маркуються за рівнем ризику. Окремо поведінковий аналіз і відстеження переходів можуть використовуватися у середовищі «пісочниці», коли організація хоче безпечно відкрити невідомий ресурс і зафіксувати його активність без участі реального користувача.

Таким чином, система придатна як для індивідуального використання під час перевірки посилань, так і для інтеграції у прикладні засоби захисту та навчальні сценарії. Гнучкість забезпечується тим, що логіка аналізу ізольована в сервісах, а підключення нових інтерфейсів або каналів виклику зводиться до надсилання адреси на перевірку та відображення отриманого структурованого результату.

3.3. Програмна документація на розроблене програмне забезпечення

У межах пункту наведено програмну документацію для розробленого ПЗ у вигляді стислих API-контрактів мікросервісів, а також узагальнено мережеву взаємодію компонентів у Docker-мережі та ключові параметри конфігурації через змінні середовища (файл `.env`). Окремо формалізовано структуру повідомлень агрегатора у потоці SSE, описано політику кешування результатів, обмеження та таймаути виконання, а також принципи часткової деградації та базові підходи до логування й діагностики.

API-контракти мікросервісів

Сервіс перевірки посилань у базах відкритих загроз (`threat_intel`) – сервіс первинної перевірки URL у зовнішніх джерелах загроз (Google Web Risk + VirusTotal) і повернення «сирих» результатів та коротких статистик для подальшої агрегації ризику. У табл. 3.1 наведено API-контракт сервісу, який забезпечує первинну перевірку заданого URL у зовнішніх джерелах загроз і повертає структуровані результати перевірки.

Таблиця 3.1

API-контракт сервісу перевірки посилань у базах

Endpoint	Метод	Параметри / тіло запиту	Формат відповіді (ключові поля)	Коди / типові помилки
/check	POST	JSON: {"url": "<string>"}	{"url": "...", "webrisk": {...}, "virustotal": {...}}, де webrisk містить safe, stats, raw, checked_at, expire_time (або error), а virustotal – safe, stats, details_url, raw, checked_at, expire_time (або error).	200 OK; 400 - url відсутній ({"error": "No URL provided"}); 500 - помилка виконання/виклику клієнтів ({"error": "..."}).
/health	GET	-	{"status": "ok"}	200 OK

Сервіс аналізу домену (domain_analyzer) – сервіс перевірки домену, виділеного з переданого URL, у VirusTotal (Domain API) з формуванням коротких статистик по релевантних рушіях та поверненням «сирих» результатів для подальшої агрегації ризику. У табл. 3.2 наведено API-контракт сервісу, який виконує перевірку домену та повертає структурований результат.

Таблиця 3.2

API-контракт сервісу аналізу домену

Endpoint	Метод	Параметри / тіло запиту	Формат відповіді (ключові поля)	Коди / типові помилки
/check_domain	POST	JSON: {"url": "<string>"}	{"url": "...", "virustotal_domain": {...}, "last_checked_at": "...", "expire_time": "...", "safe": <bool>}, де virustotal_domain зазвичай містить: domain, safe, stats, details_url, raw.	200 OK; 400 - url відсутній ({"error": "No URL provided"}); 500 - виникла помилка під час перевірки / виклику VT ({"error": "..."}).
/health	GET	–	{"status": "ok"}	200 OK

Сервіс аналізу контенту (content_analyzer) – сервіс контентної перевірки вебсторінки, який отримує HTML і скріншот (або самостійно завантажує сторінку за URL через Selenium), після чого виконує LLM-аналіз на ознаки фішингу/імітації (вердикт + пояснення) та повертає результат у структурованому вигляді для подальшої агрегації ризику. У табл. 3.3 наведено API-контракт сервісу.

Таблиця 3.3

API-контракт сервісу аналізу контенту

Endpoint	Метод	Параметри / тіло запиту	Формат відповіді (ключові поля)	Коди / типові помилки
/analyze_content	POST	JSON (мінімально одне з двох): {"url" : "<string>"} або {"html" : "<string>"}. Опційно: screenshot_base64, original_url, final_url.	{"verdict": <bool>, "explanation": "<string>","raw_ai": "<string>","screenshot_base64": "<string>","checked_at" : "<iso>","expire_time": "<iso>"}	200 OK; 400 якщо не вказано ні url, ні html ({"error": "Must provide either 'url' or 'html'"}) або якщо HTML порожній ({"error": "No HTML to analyze"}); 500 якщо не вдалося завантажити сторінку за URL ({"url": "...", "error": "..."})) або помилка ініціалізації/виклику AI ({"error": "..."}).
/health	GET	-	{"status": "ok"}	200 OK

Сервіс аналізу поведінки (behavior_analyzer) – сервіс поведінкової перевірки вебсторінки, який імітує дії «довірливого користувача» (кліки/заповнення полів) у браузері (Selenium), фіксує редиректи, появу нових вікон/алертів та зміни DOM, а за потреби ініціює додатковий контент-аналіз. Для зменшення випадкових збоїв передбачено кілька спроб аналізу (retries), після чого повертається найінформативніший результат. У табл. 3.4 наведено API-контракт сервісу.

Таблиця 3.4

API-контракт сервісу аналізу поведінки

Endpoint	Метод	Параметри / тіло запиту	Формат відповіді (ключові поля)	Коди / типові помилки
/get_behavior	POST	JSON: {"url": "<string>", "retries": <int, optional>}	{"status": "ok", "results": [...], "checked_at": "<iso>", "expire_time": "<iso>"}, де results[] містить виконану дію та спостереження: action, redirect, url_before/url_after, dom_changed, dom_change_percent, content_report (за потреби), new_window_opened, new_window_urls, alert_present, alert_text.	200 OK; 400 якщо відсутній url ({"error": "Missing 'url'"}); 500 якщо всі спроби завершилися невдачею ({"status": "error", "error": "All attempts failed."})
/get_summary	POST	JSON: {"summary_payload": {...}}	{"verdict":<bool>, "explanation": "<string>", "raw_ai": "<string>", "checked_at": "<iso>", "expire_time": "<iso>", "summary_payload": {...}}	200 OK; 400 якщо відсутній summary_payload ({"error": "missing summary_payload"}); 500 якщо збій виклику AI/обробки відповіді ({"error": "..."}))
/health	GET	-	{"status": "ok"}	200 OK

Сервіс взаємодії з базою даних (history_service) – сервіс централізованого збереження та отримання результатів перевірок для всіх мікросервісів системи. Він реалізує кешування за принципом перевірка строку дії (expire_at): дозволяє перевірити наявність актуального результату для конкретного URL/сервісу, оновити (записати) нові результати та виконати каскадне видалення історії для вказаного URL. У табл. 3.5 наведено API-контракт сервісу.

Таблиця 3.5

API-контракт сервісу взаємодії з БД

Endpoint	Метод	Параметри / тіло запити	Формат відповіді (ключові поля)	Коди / типові помилки
/check_history	POST	JSON: {"service": "<string>", "url": "<string>"}, service ∈ (threat_intel, domain_analyzer, content_analyzer, behavior_analyzer, behavior_summary)	якщо є актуальний кеш: {"cached": true, "result": {...}} (де result – збережений JSON результат сервісу); якщо кешу немає / прострочений: {"cached": false}	200 ОК – запит оброблено; 400 – якщо відсутній service або url ({"cached": false, "error": "missing service or url"}) або service невідомий ({"cached": false, "error": "unknown service"})
/update_history	POST	JSON: {"service": "<string>", "url": "<string>", "result": {...}} або {"service": "...", "url": "...", "data": {...}}.	{"status": "ok", "saved_for": "<service>", "url_id": <int>}	200 ОК; 400 якщо бракує service/url/result ({"status": "error", "error": "missing service, url or result"}) або невідомий service; 500 при помилках БД чи внутр. помилці.
/delete_url	DELETE	JSON: {"url": "<string>"}	{"status": "ok", "deleted_url": "...", "deleted_url_id": <int>}	200 ОК; 400 якщо url відсутній ({"status": "error", "error": "missing url"}); 404 якщо URL не знайдено ({"status": "error", "error": "url_not_found"}); 500 при помилках БД/внутрішніх.
/health	GET	-	{"status": "ok"}	200 ОК

Сервіс агрегатора ризику (risk_aggregator) – координуючий сервіс, який нормалізує та валідує вхідний URL, паралельно викликає мікросервіси аналізу, формує ланцюжок редиректів з обмеженням глибини, а також надає окремий ендпоінт для розрахунку інтегрального балу за вагами. У табл. 3.6 наведено API-контракт сервісу.

Таблиця 3.6

API-контракт сервісу агрегатора

Endpoint	Метод	Параметри / тіло запиту	Формат відповіді (ключові поля)	Коди / типові помилки
/check	POST	JSON: {"url": "<string>"}	Потік SSE з подіями data: {...}. Кожна подія містить: service, cached, url, depth, source, data (результат відповідного мікросервісу). Наприкінці надсилається подія для service = "behavior_summary" з підсумком по ланцюжку редиректів.	200 OK; 400 якщо відсутній url або невірний формат/ домен ({"error": "Expected JSON: ... "}/ {"error": "Invalid URL provided"})
/score	POST	JSON: {"values": { ... }} (об'єкт з частковими результатами сервісів; дозволено передавати не всі)	{"integro_score": <number or "N/A">, "explanation": "<string>", "formula": "<string>"}	200 OK (у т.ч. коли даних недостатньо: integro_score = "N/A"); 400 якщо JSON некоректний або відсутній/ порожній values ({"error": "Invalid JSON"}/ {"error": "Expected JSON with 'values' object"})
/health	GET	-	{"status": "ok"}	200 OK

Сервіс вебінтерфейсу (ui_service) наявний і реалізований як застосунок Streamlit, який надає користувачу вебсторінку для введення URL та відображення результатів перевірки. Власного REST API (яким би користувалися інші сервіси) він не публікує – натомість ініціює запити до risk_aggregator і рендерить отримані відповіді у вигляді карток.

Таблиця 3.7

Інтерфейс вебсервісу

Endpoint	Метод	Параметри / тіло запиту	Формат відповіді (ключові поля)	Коди / типові помилки
/ (головна сторінка Streamlit)	GET	- (взаємодія через форму: поле введення URL + кнопка «Перевірити»)	Web UI (Streamlit), REST API не надає; ініціює запити до агрегатора /check (SSE) та /score.	200 OK (сторінка); типові помилки відображаються як повідомлення в UI при неуспішній відповіді агрегатора або при внутрішніх винятках запиту/парсингу.

Формат обміну даними агрегатора (SSE)

Агрегатор ризику повертає результати роботи мікросервісів у вигляді потоку подій SSE із уніфікованою структурою повідомлення. Кожна подія містить службові метадані (який сервіс надіслав результат, чи він отриманий з кешу, для якого URL та на якій глибині ланцюга), а також корисне навантаження з даними сервісу. У разі помилки у повідомленні додатково передається поле error, що дозволяє UI та/або зовнішнім інтеграціям коректно відображати часткові результати та причини збоїв, структуру та призначення полів наведено у табл. 3.8.

Таблиця 3.8

Стандартна структура повідомлення агрегатора

Поле	Тип	Опис
service	string	Назва сервісу/модуля, результат якого передається (threat_intel, domain_analyzer, content_analyzer, behavior_analyzer, behavior_summary).

Продовження таблиці 3.8

Поле	Тип	Опис
cached	boolean	Ознака, що результат отримано з кешу (true) або сформовано під час поточного виконання (false).
url	string	URL, для якого сформовано результат (поточний елемент ланцюга перевірки).
depth	integer	Глибина в ланцюгу переходів/редиректів (0 – початковий URL).
source	string	Джерело результату/канал отримання.
data	object	Корисне навантаження: структурований результат відповідного сервісу.
error	string (опційно)	Опис помилки, якщо під час отримання/обробки результату виник збій.

Кешування та життєвий цикл результатів

У системі реалізовано кешування результатів перевірок на рівні history_service. Для кожного URL та конкретного сервісу зберігається останній результат і момент втрати актуальності (expire_at). Під час повторної перевірки агрегатор спочатку запитує історію; якщо запис існує та є валідним, він використовується як кеш. Якщо запис відсутній або прострочений, виконується повторний виклик відповідного сервісу, а отриманий результат записується в БД через history_service (оновлення кешу). Правила кешування наведені у табл. 3.9.

Таблиця 3.9

Правила кешування результатів

Параметр	Опис
Об'єкт кешування	Результат конкретного сервісу для заданого URL (окремо для threat_intel, domain_analyzer, content_analyzer, behavior_analyzer, behavior_summary).
Ознака валідності	Запис вважається актуальним, якщо expire_at > поточний час.
Джерело TTL	Тривалість зберігання визначається змінною середовища EXPIRED_DAYS (TTL).
Дія при простроченні	Повторний запит до відповідного сервісу та оновлення запису через ендпоінт history_service /update_history.

Експлуатаційні обмеження та таймаути

Для контрольованого часу виконання та стабільності роботи системи використовується набір експлуатаційних обмежень і таймаутів. Змінна NUM_REDIRECTS задає максимальну глибину обходу ланцюга редиректів/переходів під час перевірки URL. Для поведінкового аналізу передбачено повторні спроби: параметр MAX_ATTEMPTS (або retries, якщо передається в запиті) обмежує кількість перезапусків аналізу у разі збоїв чи нестабільного завантаження сторінки. Під час отримання контенту застосовується таймаут PAGE_FETCH_TIMEOUT, який запобігає тривалому очікуванню відповіді/завантаження.

Обробка помилок і часткова деградація

Система підтримує часткову деградацію: якщо один із мікросервісів аналізу повертає помилку або тимчасово недоступний, агрегатор не зупиняє виконання перевірки повністю, а повертає часткові результати для інших сервісів (у т.ч. у SSE-поточі). Під час розрахунку інтегрального балу ендпоінт /score враховує доступні складові, а за відсутності достатніх даних формує пояснення та може повернути значення інтегрального балу як «N/A», що унеможлиблює некоректні висновки при неповному наборі результатів.

Логування та діагностика

Для діагностики та контролю виконання у сервісах використовується стандартне логування з рівнями INFO/ERROR. Як правило, фіксуються ключові етапи обробки (початок перевірки URL, виклики зовнішніх API/LLM, підсумкові події) та помилки (некоректні параметри, збої звернення до зовнішніх сервісів, винятки Selenium/AI-клієнтів). Під час експлуатації логи контейнерів використовуються для швидкого виявлення причини збою.

Мережева конфігурація

Щодо мережевої конфігурації: кожен сервіс слухає на 0.0.0.0 у межах контейнера та має свій порт за замовчуванням; агрегатор формує внутрішні URL

викликів як `http://<CONTAINER_NAME>:<PORT>/<endpoint>` для кожного мікросервісу. Мережева конфігурація для сервісів наведена у табл. 3.10.

Таблиця 3.10

Мережева конфігурація

Компонент	Технологія	Внутрішній порт	Як викликається в Docker-мережі / примітка
threat_intel	Flask	8001	<code>http://THREAT_INTEL_CONTAINER_NAME:THREAT_INTEL_PORT/check</code>
domain_analyzer	Flask	8002	<code>http://DOMAIN_ANALYZER_CONTAINER_NAME:DOMAIN_ANALYZER_PORT/check_domain</code>
content_analyzer	Flask	8003	<code>http://CONTENT_ANALYZER_CONTAINER_NAME:CONTENT_ANALYZER_PORT/analyze_content</code>
behavior_analyzer (+ behavior_summary)	Flask	8004	<code>http://BEHAVIOR_ANALYZER_CONTAINER_NAME:BEHAVIOR_ANALYZER_PORT + /get_behavior</code> або <code>/get_summary</code>
history_service	Flask + SQL-Alchemy	8005	<code>http://HISTORY_SERVICE_CONTAINER_NAME:HISTORY_SERVICE_PORT + /check_history</code> або <code>/update_history</code> ; сервіс підключається до БД через <code>HISTORY_DATABASE_URL</code> .
risk_aggregator	Flask (SSE)	8000	<code>http://RISK_AGGREGATOR_CONTAINER_NAME:RISK_AGGREGATOR_PORT + /check (SSE)</code> або <code>/score</code> .
ui_service	Streamlit	80	UI не публікує REST API для інших сервісів; як клієнт викликає агрегатор. Доступний ззовні на <code>http://localhost/</code> .

Параметри конфігурації

У проєкті передбачено конфігурування через змінні середовища (.env), що завантажується при піднятті `docker compose`. У табл. 3.11 подано основні групи змінних, що є у файлі-прикладі конфігурації.

Таблиця 3.11

Основні групи змінних .env та обов'язковість конфігурації

Група	Змінні (приклади)	Обов'язково	Призначення / примітка
Ключі зовнішніх сервісів	WEBRISK_API_KEY, VT_API_KEY	Так (для відповідних перевірок)	Ключі доступу до Google Web Risk та VirusTotal; клієнти явно очікують їх у env, помилка якщо ключ не заданий.
Ключ/ модель LLM	GOOGLE_AI_API, GOOGLE_AI_MODEL	Так (для content_analyzer і behavior_summary)	Ключ для Google AI та назва моделі (має дефолт).
Хост/ порт сервісів (binding)	THREAT_INTEL_HOST, THREAT_INTEL_PORT; *_HOST, *_PORT (* - напр. DOMAIN_ANALYZER, BEHAVIOR_ANALYZER)	Ні (є дефолти), але бажано явно	Параметри запуску Flask-сервісів (на що «слухає» контейнер).
Імена контейнерів та порти для внутрішніх HTTP-викликів	*_CONTAINER_NAME, *_PORT	Так (для зв'язності мікро-сервісів)	Використовуються агрегатором для формування базових URL сервісів у Docker-мережі.
Підключення до БД	HISTORY_DATABASE_URL	Так (для history_service)	Рядок підключення до PostgreSQL, який використовує сервіс історії/кешу.
Параметри виконання аналізу	EXPIRED_DAYS, NUM_REDIRECTS, MAX_ATTEMPTS, HEADLESS	Ні (є дефолти), але бажано	TTL кешу/результатів та параметри поведінкового аналізу (кількість спроб, headless тощо).

Продовження таблиці 3.11

Група	Змінні (приклади)	Обов'язково	Призначення / примітка
Ваги інтегральної оцінки	*_WEIGHT (* - назва сервісу, напр. DOMAIN_ANALYZER)	Ні (є дефолти), але рекомендовано	Використовуються ендпоінтом /score для розрахунку інтегрального ризику.
Налаштування UI	RISK_AGGREGATOR_CONTAINER_NAME, RISK_AGGREGATOR_PORT	Так (щоб UI знав куди звертатись)	Streamlit-інтерфейс формує адреси агрегатора на /check і /score.

ВИСНОВОК ДО РОЗДІЛУ 3

У розділі продемонстровано практичну працездатність розробленої системи автоматичного виявлення фішингових вебресурсів та підтверджено відтворюваність її результатів. Описані етапи розгортання з налаштуванням ключів зовнішніх API і запуском мікросервісів у Docker Compose забезпечують повторюваний сценарій експлуатації, що є необхідним для коректного тестування та подальшого супроводу системи.

На прикладі типових сценаріїв перевірки показано, що інтегральний вердикт формується стійко за різних умов: як при наявності репутаційних детекцій у WebRisk/VirusTotal, так і у випадках, коли ризик проявляється на рівні контенту або тільки після взаємодії користувача через редиректи та ланцюги переходів. Це підтверджує доцільність поєднання репутаційних, контентних і поведінкових індикаторів та забезпечує підвищення надійності оцінювання порівняно з підходами, що спираються на один рівень ознак.

Окремо встановлено, що механізм кешування зменшує час повторних перевірок і навантаження на зовнішні сервіси, зберігаючи актуальність результатів у межах TTL та підтримуючи коректну повторну обробку URL одного домену. Розглянуті можливості інтеграції (внутрішній вебсервіс, розширення браузера, застосунок «перевірка перед відкриттям», використання як компонента в поштових або sandbox-процесах) вказують на практичну придатність системи до впровадження в навчальних і прикладних сценаріях кіберзахисту.

Наприкінці розділу систематизовано програмну документацію у вигляді API-контрактів мікросервісів, формату SSE-повідомлень агрегатора, правил кешування, експлуатаційних обмежень і параметрів конфігурації. Така уніфікація інтерфейсів і налаштувань забезпечує зрозумілу інтеграцію компонентів, спрощує тестування та підвищує керованість системи під час експлуатації.

РОЗДІЛ 4

РОЗРОБКА СТАРТАП-ПРОЄКТУ

У розділі розроблено стартап-проект «Система автоматичного виявлення фішингових сайтів на основі баз відкритих загроз та поведінкових ознак» і обґрунтовано доцільність його ринкового впровадження. Розділ містить опис ідеї та унікальних характеристик продукту, результати технологічного аудиту, а також аналіз ринкових можливостей і конкурентного середовища.

У межах розділу визначено потенційні сегменти споживачів і їхні вимоги до продукту, досліджено фактори загроз і можливостей маркетингового середовища, виконано аналіз конкуренції, сформовано фактори конкурентоспроможності та SWOT-матрицю. На основі отриманих результатів запропоновано альтернативи ринкової поведінки та обрано оптимальну альтернативу виходу на ринок, а також розроблено ринкову стратегію і маркетингову програму.

4.1. Опис ідеї проекту

Проект «Система автоматичного виявлення фішингових сайтів на основі баз відкритих загроз та поведінкових ознак» спрямований на зменшення ризиків кіберінцидентів, пов'язаних із переходами за шкідливими посиланнями, шляхом автоматизованої оцінки URL та формування підсумкового скорингу ризику з поясненнями.

Відмінними якостями проекту є:

- поєднання репутаційної перевірки (відкриті бази загроз) з аналізом домену, контенту та поведінкових ознак;
- модульна мікросервісна архітектура, що дозволяє масштабування й інтеграцію через API;
- наявність історії перевірок і кешування результатів для повторного використання та зменшення витрат;
- формування структурованого звіту для швидкого прийняття рішень і подальшого реагування.

Унікальність технології полягає у використанні гібридного підходу, де репутаційні дані доповнюються контентним та поведінковим аналізом (редиректи, відкриття нових вікон, наявність форм та зміни DOM), що підвищує ймовірність виявлення фішингових сторінок, які ще не потрапили до чорних списків.

Опис ідеї стартап-проєкту, напрями застосування та вигоди для користувачів наведено у табл. 4.1.

Таблиця 4.1

Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Платформа/сервіс для автоматичної перевірки URL на фішинг, що поєднує репутаційні джерела, аналіз домену, аналіз контенту сторінки та поведінкові ознаки (редиректи/нові вікна/зміни DOM/форми). Видає скоринг ризику + пояснення.	SOC/аналітики кібербезпеки (оперативні перевірки підозрілих посилань)	Швидше виявлення інцидентів, менше ручної рутини, прозорий підсумок, відтворюваність перевірок (історія/кеш).
	Банки/фінтех/e-commerce (захист клієнтів і бренду від фішингових лендінгів)	Менше шахрайства та репутаційних втрат, можливість автоматизувати реакцію, зниження навантаження на підтримку.
	Підприємства/SMB (перевірка URL у внутрішніх комунікаціях, інцидент-менеджмент)	Дешевша й простіша перевірка посилань без побудови «великого SOC», інтеграція через API, контроль політик (пороги ризику).
	Провайдери пошти/месенджери/ веб-шлюзи (URL-фільтрація)	Виявлення не лише «відомого» фішингу, а й підозрілих сторінок з поведінковими тригерами.
	Освітні/ державні установи (моніторинг фішингових сторінок під «держсервіси», навчальні полігони)	Підвищення кіберстійкості, демонстрація типових ознак фішингу на практиці, накопичення кейсів для навчання персоналу.

Основними аналогами запропонованого рішення є репутаційні URL-сервіси/блоклісти, агрегатори перевірок посилань та корпоративні антифішинг-платформи. Для визначення попередніх техніко-економічних переваг

запропонованої ідеї виконано порівняльний аналіз ключових характеристик із типовими класами конкурентних рішень. Результати аналізу сильних (S), слабких (W) і нейтральних (N) сторін наведено у табл. 4.2.

У ролі конкурентів/аналогів взято типові класи рішень:

- Конкурент 1: репутаційний сервіс/блокліст (URL reputation).
- Конкурент 2: агрегатор перевірок (multi-engine reputation/scan).
- Конкурент 3: корпоративна антифішинг-платформа (enterprise URL/email security).

Таблиця 4.2

Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

№ п/ п	Техніко- економічні характери- стики	(потенційні) товари/концепції конку- рентів				W	N	S
		Мій проєкт	Конкурент 1 (репу- тація)	Конкурент 2 (агрегатор)	Конкурент 3 (enterprise)			
1	Виявлення нових сторінок	репутація + контент + пове- дінка	здебільшо- го лише репутація	переважно репутація/ скани	часто комбі- новано		✓	
2	Аналіз редирект- ланцюгів і переходів	є, з фіксацією ланцюгів	обмежено/ частково	частково	часто є		✓	
3	Поведінко- вий аналіз	є (симу- ляція дій)	немає	рідко/ частково	інколи є			✓
4	Поясню- ваність результату	звіт по модулях	зазвичай обмежено	частково (деталі сканів)	залежить від платформи		✓	
5	Гнучкість інтеграції	висока	середня	середня	висока			✓
6	Вартість володіння для малого бізнесу	низька	низька/ середня	середня	висока			✓

Продовження таблиці 4.2

№ п/ п	Техніко- економічні характе- ристики	(потенційні) товари/концепції конку- рентів				W	N	S
		Мій проект	Конкурент 1 (репу- тація)	Конкурент 2 (агрегатор)	Конкурент 3 (enterprise)			
7	Продуктивні- сть/ затримка відповіді	частина перевірок «важка»	дуже швидко	середньо	середньо/ швидко	✓		
8	Масшта- бування та відтво- рюваність	висока	залежить від сервісу	залежить	висока			✓
9	Приватність/ контроль даних (on- prem)	можливий on-prem деплой	зазвичай SaaS	переважно SaaS	часто є on- prem/ гібрид		✓	
10	Стійкість до обхідних технік	сильніше за рахунок поведінки	слабше	середньо	середньо/ сильніше			✓

За результатами порівняння встановлено, що ключові конкурентні переваги пов'язані з наявністю поведінкового аналізу, гнучким налаштуванням логіки скорингу та можливістю інтеграції через API, тоді як потенційним обмеженням є більша тривалість «важких» перевірок, що потребує оптимізації за рахунок кешування, пріоритезації модулів та поетапного виконання аналізу.

4.2. Технологічний аудит ідеї проєкту

Для підтвердження технологічної здійсненності ідеї стартап-проєкту виконано аудит основних технологій, необхідних для реалізації системи автоматичного виявлення фішингових вебсайтів (збір і аналіз даних, модульні перевірки URL, зберігання історії, інтеграція через API та розгортання). Результати аналізу наявності та доступності технологій наведено у табл. 4.3.

Таблиця 4.3

Технологічна здійсненність ідеї проєкту

№ п/п	Ідея проєкту	Технології її реалізації	Наявність технологій (наявні / треба доробити)	Доступність технологій (доступні авторам / умови доступу)
1	Автоматична перевірка URL за відкритими базами загроз	Інтеграція з зовнішніми API (перевірка URL у базах загроз), HTTP-клієнт, обробка JSON	Наявні; потрібне налаштування запитів, лімітування та обробка помилок	Доступні: публічні/ комерційні API, доступ за ключем; реалізація можлива засобами Python
2	Аналіз домену та перевірка у базах	Виділення домену, перевірка у базах загроз	Наявні; можуть потребувати доробки правил вилучення	Доступні: відкриті бібліотеки Python, публічні/ комерційні API
3	Аналіз контенту вебсторінки (HTML, структура, підозрілі елементи)	Завантаження сторінки, парсинг HTML/ DOM, пошук форм/ скриптів/ ключових патернів	Наявні; доробка потрібна для різних типів сайтів і стабільності обробки	Доступні: бібліотеки парсингу + браузерна автоматизація; потрібні ресурси для запуску браузера в контейнері
4	Поведінковий аналіз (симуляція користувача, фіксація редиректів/ нових вікон, зміни DOM)	Автоматизація браузера для кліків/ вводу, зняття «знімків» стану (URL, вікна, DOM), метрики змін; сценарії взаємодії	Частково наявні; потрібна доробка сценаріїв, стабілізація та контроль безпечного виконання	Доступні: інструменти автоматизації; потрібні обмеження середовища (контейнер/ ізоляція) і налаштування ресурсів
5	Підсумковий скоринг ризику та агрегація результатів з модулів	Мікросервіс-агрегатор, паралельні виклики сервісів, підсумовування, нормалізація	Наявні; необхідне налаштування ваг, порогів	Доступні: реалізація власним кодом (Python), стандартні підходи до паралельності/ тайм-аутів

Продовження таблиці 4.3

№ п/п	Ідея проєкту	Технології її реалізації	Наявність технологій (наявні / треба доробити)	Доступність технологій (доступні авторам / умови доступу)
6	Зберігання історії перевірок	Реляційна БД, політика актуальності/ видалення, API історії	Наявні; необхідна схема БД під конкретні результати, індексація, правила прострочення	Доступні: PostgreSQL; локально в Docker або на сервері
7	Надання результатів користувачу та інтеграція	REST API для сервісів, UI-панель для запуску перевірки й відображення модулів, можливість інтеграції з іншими системами	Наявні; доробка – UX, формати відповідей, обробка помилок, стрімінг результатів	Доступні: вебфреймворки Python + UI-фреймворк; реалізація можлива силами авторів
8	Розгортання та експлуатація (SaaS/ on-prem)	Контейнеризація (Docker), конфігурація через змінні середовища, мережеві політики, логування/ моніторинг	Наявні; необхідна документація деплою, безпекові налаштування, масштабування	Доступні: Docker/ Compose; on-prem можливий на інфраструктурі замовника, SaaS – на хмарі/сервері
Обрана технологія реалізації ідеї проєкту: мікросервісна реалізація на Python із REST API, контейнеризацією Docker, базою даних PostgreSQL для історії/кешу та модульним підключенням компонентів аналізу (репутаційного, доменного, контентного, поведінкового) з подальшою агрегацією ризику.				

Технологічна реалізація проєкту є можливою, оскільки ключові технології вже існують на ринку та є доступними авторам, а необхідні доробки стосуються

переважно інженерної інтеграції, стабілізації та налаштування алгоритмів під практичні сценарії.

4.3. Аналіз ринкових можливостей запуску стартап-проєкту

Визначення сприятливих обставин, які можна використати для отримання переваг ринкових можливостей, є важливою частиною при запуску стартап-проєкту у сфері кібербезпеки. Це дозволяє спланувати напрями розвитку системи автоматичного виявлення фішингових вебсайтів, визначити потреби потенційних клієнтів та оцінити ризики. У табл. 4.4 відображено попередню характеристику потенційного ринку стартап-проєкту.

Таблиця 4.4

Попередня характеристика потенційного ринку стартап-проєкту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	20+
2	Загальний обсяг продаж, грн/ум.од	-
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу	Наявні (довіра до рішень, доступ до баз, інфраструктура для аналізу, інтеграції)
5	Специфічні вимоги до стандартизації та сертифікації	Наявні (вимоги до ІБ та захисту даних)
6	Середня норма рентабельності в галузі (або по ринку), %	20%

Можна зробити висновок, що ринок є привабливим для входження за попереднім оцінюванням, оскільки попит зростає, а потреба у захисті від фішингу є стабільною для більшості організацій.

У табл. 4.5 наведено визначені потенційні групи клієнтів та їх базові вимоги до продукту.

Таблиця 4.5

Характеристика потенційних клієнтів стартап-проєкту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Швидко визначати ризик URL і підтримувати визначення інцидентів	SOC/CSIRT, відділи ІБ, аналітики	Купують «для процесу реагування»; важливі швидкість, пояснюваність, повторюваність	Надійна робота, скоринг + пояснення, історія/кеш, API, журнали подій
2	Зменшити втрати від шахрайства та імперсонації бренду	Банки/фінтех/платіжні сервіси	Довші цикли погоджень, регуляторні вимоги, високі вимоги до конфіденційності	Контроль даних (можливість on-prem/гібрид), інтеграції, звітність, SLA
3	Автоматизувати перевірку посилань у бізнес-процесах	Середній/великий бізнес (B2B)	Часто стартують із пілоту; чутливі до вартості та простоти впровадження	Простий UI + API, ролі доступу, налаштування порогів/ваг, документація
4	Надавати послуги кіберзахисту клієнтам «під ключ»	MSSP/інтегратори	Потрібна масштабованість і можливість підключати багатьох клієнтів	Мульти-клієнтський режим, API/webhooks, експорт звітів, партнерські умови

Після визначення потенційних груп клієнтів доцільно проаналізувати ринкове середовище стартап-проєкту, виокремивши фактори, що можуть перешкоджати ринковому впровадженню, а також фактори, що сприяють йому. Для виявлення загроз можна спиратися на питання щодо посилення конкуренції, змін у регуляторних вимогах, доступності зовнішніх джерел даних, а також бар'єрів довіри до рішень з кібербезпеки. Фактори загроз наведено у табл. 4.6, а фактори можливостей – у табл. 4.7 (фактори подано в порядку зменшення значущості).

Таблиця 4.6

Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Посилення конкуренції з боку великих вендорів і нішевих рішень	Витіснення з ринку через сильні бренди, готові інтеграції та маркетингові бюджети конкурентів	Фокус на вузькій ніші (аналіз з поведінковими ознаками), пілоти з SOC/MSSP, унікальні функції (пояснюваність, модульність), партнерства
2	Обмеження доступу до зовнішніх джерел (ліміти, зміни API, платність)	Зниження повноти перевірок або зростання собівартості, залежність від сторонніх сервісів	Використання кількох джерел, кешування й оптимізація запитів, перехід на платні тарифи для B2B, локальні правила/евристики як резерв
3	Високі вимоги до довіри та конфіденційності	Відмова клієнтів через ризик витоку даних/невідповідність політикам, складні закупівлі	On-prem/гібридні варіанти, політики зберігання й анонімізації, аудит логів, підготовка документації безпеки
4	Висока ресурсоемність «важкого» аналізу (динаміка/ поведінка)	Повільна робота, високі витрати на інфраструктуру, складність масштабування	Багаторівнева перевірка, черги/пул воркерів, ліміти глибини, кеш, «on-demand» поведінковий аналіз для високого ризику
5	Ризик помилок детекції (false positive/ false negative) і репутаційні наслідки	Втрата довіри, зростання витрат на підтримку, відмова від використання продукту	Налаштовувані пороги/ваги, пояснюваність результатів, «review mode» для SOC, збір фідбеку, тестові набори та регресійне тестування
6	Юридичні/ етичні обмеження при аналізі контенту та симуляції дій (автокліки/ форми)	Ризики порушення правил сайтів, небажані взаємодії, претензії з боку третіх сторін	Безпечний режим виконання, заборона реальних транзакцій, маскування/контроль заповнення, дозволені сценарії, журналювання і обмеження дій

Продовження таблиці 4.6

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
7	Економічна нестабільність і скорочення бюджетів на ІБ у частини компаній	Відкладання закупівель, тиск на ціну, довші цикли продажів	Гнучкі тарифи (API/підписка), дешевий «base plan», швидкі пілоти з демонстрацією ROI, фокус на сегментах з високими втратами від фішингу

Таблиця 4.7

Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Зростання кількості фішингових атак і цифрового шахрайства	Стійкий попит на інструменти автоматичної перевірки URL та зниження ризику	Чітке позиціонування «URL risk scoring + пояснення», швидкий запуск MVP, кейси використання для SOC/бізнесу
2	Попит на пояснювані результати та автоматизацію виявлення	Команди ІБ перевантажені, потрібні інструменти, що скорочують ручну перевірку	Формування звіту «чому такий ризик», сценарії для SOC (експорт, інтеграції), історія перевірок і повторюваність
3	Розвиток партнерського каналу (MSSP/інтегратори)	MSSP можуть продавати рішення своїм клієнтам як компонент послуги	Партнерська програма, мультиклієнтський режим, API/webhooks, готові інтеграції та шаблони звітів
4	Можливість on-prem/ гібридного розгортання для чутливих клієнтів	Частина ринку не приймає чистий SaaS через політики безпеки/даних	Пакетування в Docker, документація деплою, налаштування ізоляції, підтримка інтеграції в приватні мережі
5	Диференціація через поведінковий аналіз	Поведінкові ознаки допомагають виявляти сторінки, яких ще немає у списках	Пріоритизація поведінкових тригерів, поведінку за умов ризику, демонстрація переваг на прикладах

Продовження таблиці 4.7

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
6	Додаткові джерела монетизації	Монетизація можлива не лише підпискою, а й за API-запитами/ корп-ліцензією	Тарифи: SaaS-підписка, API-пакети, ліцензія, платні опції (розширені звіти, інтеграції, SLA)
7	Використання продукту в навчанні та підвищенні кіберобізнаності	Організації зацікавлені у тренінгах і симуляціях фішингу	«Навчальний режим», бібліотека кейсів, звіти для навчання персоналу, співпраця з освітніми установами

Наведені загрози й можливості показують, що найбільш критичними для впровадження є питання довіри та залежності від зовнішніх джерел даних, тоді як ключовими можливостями виступають зростаючий попит і диференціація продукту за рахунок поведінкового аналізу та гнучкого розгортання.

У табл. 4.8 наведено загальні риси конкуренції на ринку рішень для виявлення/блокування фішингових загроз (URL reputation, anti-phishing, secure web/email security), що є релевантним для запуску стартап-проєкту з автоматичного виявлення фішингових вебсайтів.

Таблиця 4.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Тип конкуренції: монополістична	На ринку багато постачальників із схожою базовою функцією, але продукти відрізняються за якістю детекції, інтеграціями, пояснюваністю, політиками розгортання, підтримкою	Формувати чітке позиціонування: «гібридний аналіз + поведінкові ознаки + пояснюваний скоринг»; забезпечити надійність, кейси/демо, швидкий пілот для B2B

Продовження таблиці 4.8

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
2. За рівнем конкурентної боротьби: глобальний	Фішинг – глобальна загроза; клієнти можуть обирати інструменти незалежно від країни, особливо в B2B/SaaS	Орієнтація на міжнародні підходи: англomовний інтерфейс/документація, інтеграції (API), адаптація під різні політики безпеки
3. За галузевою ознакою: внутрішньогалузева	Конкуренція відбувається між IT/кібербезпековими продуктами та сервісами, що вирішують близьку задачу	Підвищувати технологічну цінність: якість детекції, оптимізація швидкодії; робити «продукт для процесу», а не лише «сканер»
4. Конкуренція за видами товарів: товарно-родова + між бажаннями	Клієнт може обрати різні типи рішень для зниження ризику фішингу: навчання персоналу, email security, secure web gateway, URL reputation, SOC-сервіси, а не лише окремий аналізатор	Показувати конкретний ефект у процесі: скорочення часу виявлення, зменшення інцидентів; пропонувати інтеграцію як «компонент» (API) і сценарії застосування (пошта, месенджери)
5. За характером конкурентних переваг: нецінова	Рішення обирають не тільки за ціною, а за точністю, швидкістю, пояснюваністю, інтеграціями, відповідністю вимогам безпеки/конфіденційності, підтримкою	Розвивати нецінові переваги: прозорі звіти «чому ризик», налаштування ваги/пороги, SLA, документація; додати on-prem/гібрид; зручний UI + API
6. За інтенсивністю: марочна	У сегменті кібербезпеки бренд/довіра суттєво впливають на вибір, особливо для enterprise та фінансового сектору; новим гравцям складніше продавати без репутації	Будувати довіру: публічні кейси/пілоти, результати тестування на наборах даних, прозора політика безпеки, партнерства з MSSP/інтеграторами, сервіси підтримки

Після ступеневого аналізу конкуренції доцільно деталізувати умови конкурентної боротьби в галузі за моделлю 5 сил М. Портера, щоб оцінити тиск з боку конкурентів, можливість входу нових гравців, вплив постачальників і клієнтів, а також ризику з боку товарів-замінників. Узагальнені результати такого аналізу для ринку рішень автоматичного виявлення фішингових вебсайтів наведено у табл. 4.9.

Таблиця 4.9

Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Поста-чальники	Клієнти	Товари-замінники
Складові аналізу	Рішення класів: URL reputation / blacklist; анти-фішингові платформи; secure gateway; платформи з URL-перевіркою.	Нові стартапи з кібер-безпеки; великі вендори, що можуть додати поведінкові/контентні модулі.	Поста-чальники баз/API; хмарні провайдери; постачальники LLM/API; компоненти браузерної автоматизації (оновлення/сумісність).	SOC та відділи ІБ; банки/фінтех; MSSP/інтегратори; e-commerce/онлайн-сервіси; держ/освітні установи.	Навчання персоналу; email security; secure web gateway/фільтрація URL; browser isolation; EDR/ XDR; аутсорсинг MSSP; ручний аналіз аналітиком.
Висновки:	На ринку багато конкурентів, конкуренція висока; потрібно диференціюватись.	Вхід можливий, але бар'єри суттєві; потенційні конкуренти є, вихід – орієнтовно 6–18 міс.	Поста-чальники частково диктують умови; потрібна диверсифікація джерел і оптимізація запитів.	Клієнти диктують вимоги; необхідні пілоти та доказ ефективності.	Замінники обмежують попит на продукт; варто позиціонувати рішення як компонент процесу та давати переваги.

Інтенсивність конкуренції з боку прямих конкурентів є високою, оскільки на ринку присутні численні програмні рішення. Можливості входу в ринок наявні, однак бар'єри для нових гравців суттєві. Постачальники частково диктують умови, а клієнти також диктують вимоги. Загроза замінників є середньо-високою, оскільки існують альтернативні підходи до зниження фішингових ризиків.

Отже, принципова можливість роботи на ринку є реальною, а щоб бути конкурентоспроможною, система повинна мати сильні сторони: пояснюваний скоринг, модульність та інтеграції (API), оптимізацію швидкодії.

На основі аналізу конкуренції, характеристик ідеї проєкту, вимог споживачів до товару та факторів маркетингового середовища визначено перелік ключових факторів конкурентоспроможності стартап-проєкту, наведено у табл. 4.10.

Таблиця 4.10

Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування
1	Точність виявлення	Визначає довіру та практичну цінність для SOC/бізнесу.
2	Пояснюваність результату	Потрібна для виявлення інцидентів і прийняття рішень.
3	Гібридний аналіз (репутація + контент + поведінка)	Підвищує виявлення «нового» фішингу та стійкість до обходу.
4	Швидкодія та масштабованість	Важливо через ресурсоемність динамічного/поведінкового аналізу.
5	Інтеграція через API та сумісність із процесами ІБ	Клієнти очікують вбудування в існуючі системи та процеси реагування.
6	Контроль даних і варіанти розгортання (on-prem/ гібрид)	Знімає бар'єри для фінтех/enterprise з вимогами конфіденційності.
7	Вартість володіння (ТСО)	Конкуренція йде не лише за ціною, а за сукупними витратами на використання.

Після визначення факторів конкурентоспроможності проведено оцінювання сильних і слабких сторін стартап-проєкту шляхом порівняння з типовими конкурентними рішеннями. Результати порівняльного аналізу наведено у табл. 4.11.

Таблиця 4.11

Порівняльний аналіз сильних та слабких сторін «Система автоматичного виявлення фішингових вебсайтів»

	Фактор конкурентоспроможності	Бали 1–20	Рейтинг товарів-конкурентів у порівнянні з системою автоматичного виявлення фішингових вебсайтів (–3 ... +3)						
			–3	–2	–1	0	+1	+2	+3
1	Точність виявлення	18				0			
2	Пояснюваність результату	16			–1				
3	Гібридний аналіз (репутація + контент + поведінка)	17		–2					
4	Швидкодія та масштабованість	12						+2	
5	Інтеграція через API та сумісність із процесами ІБ	14					+1		
6	Контроль даних і варіанти розгортання (on-prem/ гібрид)	13				0			
7	Вартість володіння (TCO)	15			–1				

У табл. 4.12 наведено SWOT-аналіз стартап-проєкту «Система автоматичного виявлення фішингових вебсайтів».

Таблиця 4.12

SWOT-аналіз стартап-проєкту

Сильні сторони:	Слабкі сторони:
- Гібридний підхід: репутаційний, контентний та поведінковий аналіз URL. - Пояснюваний скоринг (структурований звіт по модулях). - Мікросервісна архітектура, зручна інтеграція через API. - Можливість контейнеризованого/ on-prem розгортання.	- Ресурсоемність динамічного/поведінкового аналізу, можливі затримки відповіді. - Залежність від зовнішніх баз та LLM API. - Відсутність розкрученого бренду та референсів на початковому етапі. - Потреба у постійному оновленні правил/ евристик.

Продовження таблиці 4.12

Можливості:	Загрози:
- Зростання кількості фішингових атак і попиту на автоматичний аналіз URL. - Співпраця з MSSP та інтеграторами як партнерський канал збуту. - Попит на пояснювані рішення для SOC та аудиту безпеки. - Використання системи в навчанні та підвищенні кіберграмотності.	- Посилення конкуренції з боку великих вендорів кібербезпеки. - Зміни умов доступу до зовнішніх API (ліміти, тарифи, політики). - Економічні кризи та скорочення бюджетів на ІБ у частини клієнтів. - Замінники: email/ web-шлюзи, навчання персоналу замість впровадження нового продукту.

На основі SWOT-аналізу сформовано альтернативи ринкової поведінки для виведення стартап-проєкту «Система автоматичного виявлення фішингових вебсайтів на основі баз відкритих загроз та поведінкових ознак» на ринок. Узагальнений перелік альтернатив та орієнтовні строки їх реалізації наведено у табл. 4.13.

Таблиця 4.13

Альтернативи ринкового впровадження стартап-проєкту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Запуск MVP як API/веб-сервіс для SOC і технічних команд: реалізувати базовий потік перевірки URL, пояснюваний звіт, історію/кеш; підготувати демо і мінімальну документацію	Висока	3–4 місяці
2	Партнерський запуск через MSSP/інтеграторів: підготувати партнерський режим, API-ключі/тарифи, мультиклієнтський сценарій, шаблони звітів; узгодити 1–2 пілоти з партнерами для їхніх клієнтів	Середня	4–6 місяців
3	On-prem/enterprise-пакет для фінтех: контейнеризований реліз, інструкції деплою, політики зберігання даних, базові вимоги безпеки, можливість роботи з обмеженим виходом назовні; пілоти з 1–2 замовниками	Середня	6–8 місяців

Продовження таблиці 4.13

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
4	Освітній/навчальний напрям: кейси прикладів фішингу, пояснювані індикатори, матеріали для тренінгів; продаж доступу закладам/курсам	Висока	2–3 місяці
5	Посилення продукту на основі фідбеку: оптимізація швидкодії, зменшення помилок, розширення інтеграцій і звітності	Висока	6–9 місяців

Обрана альтернатива: Альтернатива 1, оскільки отримання ресурсів для неї є більш простим та ймовірним, а строки реалізації – більш стислі. Після підтвердження попиту доцільно переходити до альтернатив 2–3 як наступних етапів масштабування.

4.4. Розроблення ринкової стратегії проєкту

Перед розробленням ринкової стратегії стартап-проєкту доцільно визначити цільові групи споживачів, для яких продукт має найбільшу цінність. Для цього виконано порівняння потенційних сегментів сприйняттям продукту, попитом, інтенсивністю конкуренції та простотою входу. Результати наведено у табл. 4.14.

Таблиця 4.14

Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнто- вний попит в межах цільової групи (сегменту)	Інтенси- вність конку- ренції в сег- менті	Просто- та входу у сег- мент
1	SOC/CSIRT, відділи ІБ (B2B): команди реагування на інциденти, triage підозрілих URL	Висока	Високий	Висока	Середня

Продовження таблиці 4.14

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
2	MSSP/інтегратори: провайдери керованих послуг безпеки	Середня–висока	Середній–високий	Середня	Середня
3	Банки/фінтех/платіжні сервіси	Середня	Високий	Висока	Низька
4	Середній/великий бізнес: захист корпоративної пошти/процесів, helpdesk-перевірки URL	Середня	Середній	Середня	Середня
5	Е-commerce/онлайн-сервіси: швидка реакція на фішингові копії, бренд-імперсонація	Середня	Середній	Середня	Середня
6	Освітні/державні установи (навчальний/моніторинговий сценарій)	Висока	Низький–середній	Низька	Висока
Які цільові групи обрано: SOC/CSIRT та відділи інформаційної безпеки (як сегмент з найвищою готовністю до продукту та чітким попитом), а також MSSP/інтеграторів як канал масштабування за рахунок партнерської моделі.					

За результатами аналізу обрано стратегію диференційованого маркетингу, оскільки стартап-проект працює з кількома сегментами (SOC/CSIRT і MSSP), для яких доцільно використовувати різні підходи ринкового впливу (прямі пілоти та продажі для SOC; партнерська модель/white-label та інтеграційні пакети для MSSP).

Для роботи в обраних сегментах ринку необхідно сформулювати базову стратегію розвитку з урахуванням обраної альтернативи ринкового впровадження (альтернатива 1) та визначеної раніше стратегії охоплення ринку. Визначення базової стратегії розвитку наведено у табл. 4.15.

Таблиця 4.15

Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Альтернатива 1: запуск MVP як API/веб-сервіс для SOC і технічних команд	Диференційований маркетинг (SOC/ CSIRT як основний сегмент; MSSP - наступний етап)	Пояснюваний скоринг; гібридний аналіз; мікросервісна архітектура та API-інтеграції; можливість контейнеризованого/ on-prem розгортання	Стратегія диференціації

Наступним кроком розроблення ринкової стратегії є вибір стратегії конкурентної поведінки, яким чином стартап буде поводитися відносно конкурентів: чи є він «першопрохідцем», на кого буде орієнтуватися у залученні клієнтів та які характеристики конкурентних рішень доцільно перейняти як «кращі практики». Узагальнення вибраної конкурентної поведінки наведено у табл. 4.16.

Таблиця 4.16

Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Частково (новизна у поєднанні баз + контентного + поведінкового аналізу та пояснюваного скорингу)	Змішаний підхід: залучення нових споживачів і відбір частини клієнтів у конкурентів за рахунок кращої якості звіту	Так, частково: використання кращих практик конкурентів; без копіювання унікальних рішень - акцент на диференціації	Стратегія заняття ніші: фокус на сегменті SOC/CSIRT та технічних команд

Враховуючи вимоги споживачів обраних сегментів до продукту (табл. 4.5), а також обрану базову стратегію розвитку (табл. 4.15) і стратегію конкурентної поведінки (табл. 4.16), сформовано стратегію позиціонування стартап-проєкту. Вона полягає у визначенні комплексу ключових асоціацій, за якими цільова аудиторія має ідентифікувати продукт на ринку. Результати наведено у табл. 4.17.

Таблиця 4.17

Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформулювати комплексну позицію власного проєкту (три ключових)
1	Надійна робота, скоринг + пояснення, історія/кеш, API, журнали подій	Стратегія диференціації	Пояснюваний скоринг; гібридний аналіз; мікросервісна архітектура; API	Пояснюваність, довіра/аудитованість
2	Мульти-клієнтський режим, API, експорт звітів, партнерські умови	Стратегія диференціації	Модульність; масштабованість; API; зручне підключення як компонента послуги	Масштабованість, інтегрованість, партнерський продукт
3	Контроль даних (on-prem/гібрид), інтеграції, звітність, SLA (банки/фінтех)	Стратегія диференціації	Контейнеризоване розгортання; керовані політики; звітність; пояснюваний результат	Конфіденційність, керованість, відповідність вимогам
4	Простий UI + API, ролі доступу, налаштування порогів/ваг, документація	Стратегія диференціації	Налаштовуваний скоринг; API; зрозумілий звіт; можливість швидкого пілоту	Простота впровадження, гнучкість налаштувань, прозорий ризик

Результатом виконання підрозділу є узгоджена система рішень щодо ринкової поведінки стартап-компанії: позиціонування будується навколо пояснюваності, а для масштабування через партнерів – навколо інтегрованості та масштабованості.

4.5. Розроблення маркетингової програми стартап-проекту

Для формування маркетингової концепції продукту необхідно підсумувати результати попереднього аналізу конкурентоспроможності та сформулювати, яку потребу закриває продукт, яку вигоду отримує клієнт і чим саме рішення виграє на фоні конкурентів. Узагальнення ключових переваг концепції потенційного товару наведено у табл. 4.18.

Таблиця 4.18

Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Швидко визначати ризик URL і підтримувати тріаж інцидентів	Швидкий скоринг ризику + зрозуміле пояснення «чому так», з відтворюваними результатами	Пояснюваний звіт по модулях; гібридна перевірка (репутація + контент + поведінка); історія/ кеш для повторних перевірок
2	Автоматизувати перевірку посилань у бізнес-процесах	Вбудування перевірки URL у внутрішні системи без ручної роботи	API інтеграція; налаштовувані пороги/ ваги; стандартизовані результати й журнали подій
3	Надавати кіберзахист клієнтам «під ключ» (MSSP)	Масштабована перевірка URL для багатьох клієнтів із централізованими звітами	Мульти-клієнтський сценарій (потрібно створити/ посилити); експорт звітів/ webhooks; гнучка схематика під партнерські моделі
4	Контроль даних і відповідність вимогам (фінтех/enterprise)	Можливість використання рішення без виведення чутливих даних назовні	Контейнеризоване / on-prem / гібридне розгортання; керовані політики зберігання даних; звітність

Цим визначено базову маркетингову концепцію: продукт позиціонується як пояснюваний гібридний аналізатор URL, який легко інтегрується через API та може працювати у форматі SaaS або on-prem залежно від вимог сегмента.

У табл. 4.19 наведено трирівневу маркетингову модель товару, що уточнює сутність продукту, його складові у реальному виконанні, а також підхід до захисту від копіювання.

Таблиця 4.19

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Система автоматичного виявлення фішингових вебсайтів на основі баз відкритих загроз та поведінкових ознак. Базова потреба: швидко й обґрунтовано оцінити ризик посилання для підтримки тріажу інцидентів та перевірки URL у бізнес-процесах.		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	Пояснюваний скоринг	Нм	Тх/Ор
	Гібридний аналіз URL	Нм	Тх
	REST API, журнали подій	Нм	Тх/Ор
	Масштабованість	Нм	Тл/Тх
	Історія перевірок і кеш результатів	Нм	Тх
	Варіанти розгортання	Нм	Ор
	Якість: стабільна робота сервісів, відтворюваність результатів, логування помилок.		
Пакування: веб-інтерфейс + REST API + користувацька й технічна документація.			
Марка: PhishingDetectionSystem.			
III. Товар із підкріпленням	До продажу: демо-стенд/пілот, базова консультація з інтеграції, приклади запитів/скриптів		
	Після продажу: технічна підтримка, оновлення правил/евристик, оптимізація моделей, оновлення документації, можливий SLA.		
Захист від копіювання: авторське право на програмний код і документацію, ліцензійна модель використання; закрита логіка скорингу та агрегації ознак (ноу-хау), закриті датасети/правила, контроль доступу через ключі/API-токени.			

Для попереднього визначення цінових меж на сервіс PhishingDetectionSystem виконано експертне порівняння з товарами-замінниками (що можуть частково закривати потребу) та товарами-аналогами (подібні рішення з перевірки/аналізу URL), а також враховано платоспроможність цільових сегментів. Результати наведено у табл. 4.20.

Таблиця 4.20

Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	Низький–середній (навчання персоналу, базові фільтри/шлюзи)	Середній (URL reputation/ scan-сервіси, anti-phishing SaaS)	Середній–високий (бюджети ІБ у компаній із SOC/CSIRT)	нижня: 20 тис. грн/міс; верхня: 120 тис. грн/міс
2	Середній (комплексні шлюзи/ ізоляція браузера як альтернатива)	Середній–високий (інтегровані рішення для провайдерів послуг)	Високий (MSSP, що продають сервіс «під ключ»)	нижня: 50 тис. грн/міс; верхня: 200 тис. грн/міс (або залежно від обсягу клієнтів/перевірок)
3	Середній (корпоративні security-платформи як частковий замінник)	Високий (enterprise-рівень, розширені вимоги)	Високий (банки/ фінтех/ enterprise)	нижня: 100 тис. грн/міс; верхня: 400 тис. грн/міс (або ліцензія on-prem за договором)

У таблиці 4.21 наведено оптимальну систему збуту для стартап-проєкту PhishingDetectionSystem з урахуванням закупівельної поведінки цільових сегментів (SOC/CSIRT як основний сегмент; MSSP як етап масштабування; enterprise/on-prem як окремий формат).

Таблиця 4.21

Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	SOC/ CSIRT, відділи ІБ: купують «для процесу реагування», часто через пілот/ PoC, важливі технічні докази цінності та інтеграція	Презентація/ демо, PoC, технічна консультація, інтеграція через API, налаштування порогів/ ваг, підтримка/ оновлення	Прямий канал (0 рівнів)	Власний збут: прямі B2B продажі, технічний супровід
2	MSSP/ інтегратори: потрібна масштабованість, партнерські умови, можливість «вбудувати» продукт у послугу та продавати під клієнта	Партнерський пакет (API, документація, звіти), навчання партнера, техпідтримка, SLA-умови для сервісу	1 рівень (1 посередник)	Залучена система збуту: партнерський канал через MSSP/ інтеграторів (white-label/ реселл)
3	Фінтех/ enterprise (on-prem): довгі погодження, вимоги до контролю даних, безпеки та юридичних умов	Пресейл аудит вимог, узгодження договору, інсталяція/ реліз, інтеграція, підтримка, оновлення	Прямий канал (0 рівнів) або 1 рівень	Комбінована модель: прямі enterprise-продажі + (за потреби) інтегратор як технічний партнер впровадження

Обрана система збуту: комбінована – прямі продажі для SOC/enterprise, а для масштабування – партнерський канал через MSSP/інтеграторів.

У таблиці 4.22 наведено концепцію маркетингових комунікацій для стартап-проекту PhishingDetectionSystem, сформовану з урахуванням специфіки поведінки цільових сегментів, обраного позиціонування (табл. 4.17) та моделі збуту (табл. 4.21).

Таблиця 4.22

Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	SOC/CSIRT, відділи ІБ: вибір через пілот, потрібна якість, важлива пояснюваність	Професійні спільноти (LinkedIn), профільні медіа, вебінари, конференції, прямі листи/контакти	Пояснюваність; довіра/аудитованість	Показати, що продукт скорочує час та дає звіт; підштовхнути до демо	«Пояснюваний ризик URL: швидко, відтворювано, з доказами по модулях»
2	MSSP/інтегратори: оцінюють вигоду як модуль у послугі, важлива масштабованість і партнерство	Партнерські контакти, B2B-зустрічі, конференції, LinkedIn, технічні презентації	Масштабованість; інтегрованість; партнерський продукт	Показати, що рішення легко «вбудовується» в сервіс MSSP та підвищує цінність їх послуги; мотивувати до партнерства	«Готовий модуль перевірки URL для вашого SOC-as-a-Service: API, звіти, мультиклієнтський режим»
3	Банки/фінтех: довгі погодження, високі вимоги до контролю даних; важлива надійність	Офіційні B2B-канали, тендери, профільні конференції, рекомендації інтеграторів, прямі переговори	Конфіденційність; керованість; відповідність вимогам	Підкреслити контроль даних (on-prem/гібрид), звітність і керовані політики; спонукати до пілоту	«Фішинг-аналіз без виведення даних назовні: on-prem/гібрид, політики, звітність, SLA»

Продовження таблиці 4.22

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими кори- стуються цільові клієнти	Ключові позиції, обрані для пози- ціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
4	Середній/ великий бізнес: часто стартують з простого сценарію, чутливі до ціни та простоти впро- вадження	Веб-сайт продукту, SEO/ контент, кейси, реко- ментації партнерів, LinkedIn, прямі звернення	Простота впро- вадження; гнучкість налаштувань; прозорий ризик	Показати простий старт і швидку користь (перевірка URL у процесах), підвести до тестового вико- ристання	«Підключіть перевірку посилань у ваші процеси за 1 день: UI + API, налаштування порогів, зрозумілий звіт»

ВИСНОВОК ДО РОЗДІЛУ 4

У розділі сформовано стартап-проект «Система автоматичного виявлення фішингових сайтів на основі баз відкритих загроз та поведінкових ознак» та обґрунтовано можливість його комерціалізації. Визначено цінність продукту: гібридна перевірка URL (репутація + домен + контент + поведінка) з пояснюваним скорингом і можливістю інтеграції через API та розгортання в Docker/on-prem. Технологічний аудит підтвердив здійсненність реалізації на базі доступних технологій, а ключові технічні ризики пов'язані з ресурсоемністю поведінкового аналізу та залежністю від зовнішніх API, що потребує кешування, поетапного запуску модулів і резервних підходів.

Ринковий аналіз показав наявність попиту та зростання ринку антифішингових рішень. Основними цільовими групами визначено SOC/CSIRT (як сегмент із найвищою готовністю) та MSSP/інтеграторів (як канал масштабування); також релевантні банки/фінтех і B2B-компанії з вимогами до контролю даних. Конкуренція на ринку висока й глобальна, бар'єри входу пов'язані з довірою, інтеграціями та інфраструктурою, а також існують замітники (шлюзи, ізоляція браузера, навчання персоналу). Конкурентоспроможність проекту забезпечується факторами: точність виявлення, пояснюваність, гібридний аналіз, API-інтеграції, масштабованість/швидкодія, можливість on-prem та прийнятний TCO.

За результатами SWOT та аналізу альтернатив обрано Альтернативу 1 – запуск MVP як API/веб-сервісу для SOC і технічних команд, оскільки вона потребує найменше ресурсів і має найкоротші строки реалізації; після підтвердження попиту доцільно переходити до партнерського каналу (MSSP) та on-prem/enterprise-варіанта. Отже, подальша імплементація проекту є доцільною: попит на ринку наявний, перспективи впровадження реалістичні, а стратегія виходу відповідає конкурентній ситуації.

ВИСНОВКИ

У роботі реалізовано поставлену мету – розроблено й досліджено систему автоматичного виявлення фішингових сайтів, яка поєднує використання баз відкритих загроз, аналіз веб-контенту та поведінкові ознаки користувача в межах мікросервісної архітектури. Сформульовані у вступі завдання виконано в повному обсязі.

На основі аналізу предметної області та існуючих технічних рішень уточнено специфіку сучасних веб-фішингових атак, показано обмеження підходів, що спираються лише на сигнатурні бази або статичні ознаки сторінки, та обґрунтовано доцільність комбінованого підходу з використанням поведінкового аналізу. Розроблено структурну схему та архітектуру мікросервісної системи, до складу якої входять сервіси перевірки посилань у відкритих базах загроз, аналізу домену, контенту сторінки, симуляції дій користувача, агрегатор результатів, сервіс роботи з історією перевірок та демонстраційний веб-інтерфейс.

Реалізовано програмну систему, що забезпечує: інтеграцію з VirusTotal та Google Web Risk, аналіз HTML-структури та візуальних ознак сторінки, сценарії «довірливої» поведінки користувача з відстеженням редиректів і змін DOM, кешування результатів у базі даних із механізмом прострочення, а також інтерактивне відображення результатів у веб-інтерфейсі. Проведені експериментальні сценарії на реалістичних прикладах фішингових та легітимних сайтів підтвердили працездатність системи, її здатність виявляти підозрілі ознаки навіть у випадках, коли окремі джерела загроз не містять записів про URL, а також продемонстрували зменшення кількості повторних запитів за рахунок кешування.

Отримані результати відповідають сучасному рівню розвитку засобів кіберзахисту та мають як науково-технічну, так і практичну значущість. Наукова новизна полягає у поєднанні відкритих баз загроз, контентного та поведінкового аналізу в єдиному мікросервісному середовищі; практична цінність – у можливості інтеграції розробленого прототипу в корпоративні та державні системи

моніторингу безпеки або використання його як основи для стартап-рішення у форматі сервісу моніторингу фішингових веб-ресурсів.

Перспективними напрямками подальших досліджень є: розширення набору джерел загроз і поведінкових сценаріїв, побудова ML/LLM-моделей для автоматичного формування ризиків на основі накопичених даних, масштабування системи для обробки великих потоків URL у реальному часі та інтеграція з існуючими SOC/SIEM-платформами. Це дозволить підвищити точність, стійкість і придатність системи для промислової експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Фішинг, Zero Click-експлойти, AI-атаки: тенденції кіберзагроз в Україні у 2025 році [Електронний ресурс] // IT Ukraine Association. — Режим доступу: <https://itukraine.org.ua/fishing-zero-click-eksplotji-ai-ataki-tendentsiyi-kiberzagroz-v-ukrayini-u-2025-rotsi/> (дата звернення: 15.09.2025).
2. Phishing Trends Report [Електронний ресурс] // Hoxhunt. — Режим доступу: <https://hoxhunt.com/guide/phishing-trends-report> (дата звернення: 15.09.2025).
3. 19 Most Common Types of Phishing Attacks in 2025 [Електронний ресурс] // UpGuard Blog. — Режим доступу: <https://www.upguard.com/blog/types-of-phishing-attacks> (дата звернення: 16.09.2025).
4. The most common types of phishing attacks: Definitions and categories [Електронний ресурс] // NordVPN Blog. — Режим доступу: <https://nordvpn.com/ru/blog/types-of-phishing> (дата звернення: 16.09.2025).
5. The Three Stages Of a Phishing Attack - Bait, Hook And Catch [Електронний ресурс] — Режим доступу: <https://blog.usecure.io/three-steps-of-phishing> (дата звернення: 17.09.2025).
6. Що таке фішинг, як його розпізнати і як із ним боротися [Електронний ресурс] // Digvel. — Режим доступу: <https://digvel.com.ua/blog/shho-take-fishyng-i-yak-iz-nym-borotysya/> (дата звернення: 17.09.2025).
7. The ripple effect: How one phishing attack can cause disaster across your organization [Електронний ресурс] // CybSafe Blog. — Режим доступу: <https://www.cybsafe.com/blog/how-can-phishing-affect-a-business/> (дата звернення: 18.09.2025).
8. VirusTotal Documentation Hub [Електронний ресурс]. — Режим доступу: <https://docs.virustotal.com/> (дата звернення: 19.09.2025).
9. Google Safe Browsing [Електронний ресурс]. — Режим доступу: <https://safebrowsing.google.com/> (дата звернення: 19.09.2025).
10. Google Web Risk API [Електронний ресурс]. — Режим доступу: <https://cloud.google.com/security/products/web-risk> (дата звернення: 20.09.2025).

11. PhishTank [Электронный ресурс]. — Режим доступа: <https://www.phishtank.com/> (дата звернения: 21.09.2025).
12. OpenPhish [Электронный ресурс]. — Режим доступа: <https://openphish.com/index.html> (дата звернения: 21.09.2025).
13. URLhaus [Электронный ресурс]. — Режим доступа: <https://urlhaus.abuse.ch/> (дата звернения: 22.09.2025).
14. Defending 1 billion Chrome users with Enhanced Protection [Электронный ресурс] // Google Blog. — Режим доступа: <https://blog.google/products/chrome/google-chrome-safe-browsing-one-billion-users> (дата звернения: 22.09.2025).
15. Enhanced Phishing Protection in Microsoft Defender SmartScreen [Электронный ресурс] // Microsoft Learn. — Режим доступа: <https://learn.microsoft.com/en-us/windows/security/operating-system-security/virus-and-threat-protection/microsoft-defender-smartscreen/enhanced-phishing-protection> (дата звернения: 23.09.2025).
16. Norton Safe Web [Электронный ресурс]. — Режим доступа: <https://us.norton.com/feature/safe-web> (дата звернения: 23.09.2025).
17. What Is Phishing? [Электронный ресурс] // Bitdefender InfoZone. — Режим доступа: <https://www.bitdefender.com/en-us/business/infozone/what-is-phishing> (дата звернения: 24.09.2025).
18. Cloudflare Browser Isolation [Электронный ресурс]. — Режим доступа: <https://www.cloudflare.com/zero-trust/products/browser-isolation/> (дата звернения: 25.09.2025).
19. Multimodal LLMs for Phishing Detection [Электронный ресурс] // GASA. — Режим доступа: <https://www.gasa.org/post/multimodal-llms-for-phishing-detection> (дата звернения: 26.09.2025).
20. Omar A. R. From Phishing Behavior Analysis and Feature Selection to Enhance Prediction Rate in Phishing Detection [Электронный ресурс] / Asmaa Reda Omar, Shereen Taie, Masoud E. Shaheen // International Journal of Advanced Computer Science and Applications. — 2023. — Т. 14, № 5. — Режим

доступу: <https://doi.org/10.14569/ijacsa.2023.01405107> (дата
звернення: 27.09.2025).

21. Leveraging LLMs for Phishing email detection [Електронний ресурс] // LLM for Cyber. — Режим доступу: <https://www.llmfor cyber.com/blog/leveraging-llm-for-phishing-email-detection> (дата звернення: 28.09.2025).
22. Beyond Hashing: The Future of Anti-Phishing is Image Similarity Search [Електронний ресурс] // Bolster AI Blog. — Режим доступу: <https://bolster.ai/blog/computer-vision> (дата звернення: 30.09.2025).
23. What is Docker? [Електронний ресурс] // Docker Docs. — Режим доступу: <https://docs.docker.com/get-started/docker-overview/> (дата звернення: 05.10.2025).
24. Python 3.14.0 documentation [Електронний ресурс]. — Режим доступу: <https://docs.python.org/3/> (дата звернення: 06.10.2025).
25. Flask Documentation [Електронний ресурс]. — Режим доступу: <https://flask.palletsprojects.com/en/stable/> (дата звернення: 08.10.2025).
26. The Selenium Browser Automation Project [Електронний ресурс]. — Режим доступу: <https://www.selenium.dev/documentation/> (дата звернення: 09.10.2025).
27. BeautifulSoup Documentation [Електронний ресурс]. — Режим доступу: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/> (дата звернення: 10.10.2025).
28. Google AI Studio [Електронний ресурс]. — Режим доступу: <https://aistudio.google.com/> (дата звернення: 14.10.2025).
29. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс]. — Режим доступу: <https://www.postgresql.org/> (дата звернення: 17.10.2025).
30. Streamlit documentation [Електронний ресурс]. — Режим доступу: <https://docs.streamlit.io/> (дата звернення: 19.10.2025).
31. Git Documentation [Електронний ресурс]. — Режим доступу: <https://git-scm.com/> (дата звернення: 21.10.2025).

32. Detect malicious URLs with Web Risk [Электронный ресурс]. — Режим доступа: <https://docs.cloud.google.com/web-risk/docs/detect-malicious-urls> (дата звернення: 22.10.2025).
33. VirusTotal [Электронный ресурс]. — Режим доступа: <https://www.virustotal.com> (дата звернення: 22.10.2025).

ДОДАТКИ

ДОДАТОК А

Програмна реалізація сервісу перевірки посилань у базах відкритих загроз

```
# webrisk_client.py
import os
from pathlib import Path
from typing import List, Dict, Any, Optional
import requests
from dotenv import load_dotenv
from datetime import datetime, timedelta, UTC

EXPIRED_DAYS = int(os.getenv("EXPIRED_DAYS", 7))

class WebRiskClient:
    BASE_URL = "https://webrisk.googleapis.com/v1/uris:search"

    def __init__(self, api_key: Optional[str] = None, dotenv_path: Optional[Path] = None):
        if api_key:
            self.api_key = api_key
        else:
            self.api_key = os.getenv("WEBRISK_API_KEY")

            if not self.api_key:
                if dotenv_path is None:
                    dotenv_path = Path.cwd() / ".env"
                if dotenv_path.exists():
                    load_dotenv(dotenv_path=dotenv_path)
                    self.api_key = os.getenv("WEBRISK_API_KEY")

            if not self.api_key:
                raise ValueError("API key not found. Provide WEBRISK_API_KEY via env or pass api_key.")

        self.session = requests.Session()
        self.session.headers.update({"User-Agent": "webrisk-client/1.0"})

    def _call_api(self, uri: str, threat_types: List[str]) -> Dict[str, Any]:
        params = {
            "key": self.api_key,
            "threatTypes": ",".join(threat_types),
            "uri": uri
        }
```

```

resp = self.session.get(self.BASE_URL, params=params, timeout=10)
resp.raise_for_status()
return resp.json()

def check_url(self, url: str, threat_types: Optional[List[str]] = None) -> Dict[str, Any]:
    if threat_types is None:
        threat_types = ["SOCIAL_ENGINEERING"]
    try:
        resp_json = self._call_api(url, threat_types)
        threat = resp_json.get("threat")
        safe = not bool(threat and threat.get("threatTypes"))

        threats = []
        if threat:
            for t in threat.get("threatTypes", []):
                threats.append({"engine": "Google Web Risk", "type": t.lower(),
"category": "malicious"})

        stats = {
            "malicious": len(threats),
            "suspicious": 0,
            "harmless": 0 if threats else 1,
            "undetected": 0,
        }

        time_now = datetime.now(UTC)
        time_exp = time_now + timedelta(days=EXPIRED_DAYS)
        result = {
            "safe": safe,
            "stats": stats,
            "details_url": None,
            "raw": resp_json,
            "checked_at": time_now.isoformat(),
            "expire_time": time_exp.isoformat(),
        }

    except Exception as e:
        result = {"url": url, "error": str(e)}

    return result

def batch_check(self, urls: List[str], threat_types: Optional[List[str]] = None) ->
List[Dict[str, Any]]:

```

```

results = []
for u in urls:
    results.append(self.check_url(u, threat_types=threat_types))
return results

```

```

# virustotal_client.py
import logging
import os
from datetime import datetime, timedelta, UTC
from pathlib import Path
from typing import List, Dict, Any, Optional
import vt
from dotenv import load_dotenv

```

```

logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s]
%(message)s")
logger = logging.getLogger(__name__)

```

```

PHISHING_RELEVANT_ENGINES = {
    "Artists Against 419",
    "ChainPatrol",
    "Chong Lua Dao",
    "desenmascara.me",
    "Google Safebrowsing",
    "Yandex Safebrowsing",
    "Netcraft",
    "OpenPhish",
    "Phishing Database",
    "Phishtank",
    "PhishFort",
    "PhishLabs",
    "SafeToOpen",
    "Spam404",
    "URLQuery",
    "alphaMountain.ai",
    "Axur",
    "ZeroFox",

    "Abusix",
    "Acronis",
    "BitDefender",
    "Dr.Web",

```

```

"Emsisoft",
"ESET",
"ESTsecurity",
"Forcepoint ThreatSeeker",
"Fortinet",
"G-Data",
"Heimdal Security",
"Kaspersky",
"Quick Heal",
"Rising",
"Sangfor",
"Sophos",
"Trustwave",
"Webroot",
"Bkav",
"Mimecast",
"VIPRE",
"Xcitium Verdict Cloud",
"CyRadar",

"malwares.com URL checker",
"MalwarePatrol",
"MalwareURL",
"MalwareD",
"Quttera",
"Sucuri SiteCheck",
"Sansec eComscan",
"URLhaus",
}

```

```
EXPIRED_DAYS = int(os.getenv("EXPIRED_DAYS", 7))
```

```

class VirusTotalClient:
    def __init__(self, api_key: Optional[str] = None, dotenv_path: Optional[Path] =
None):
        if api_key:
            self.api_key = api_key
        else:
            self.api_key = os.getenv("VT_API_KEY")

        if not self.api_key:
            if dotenv_path is None:

```

```

        dotenv_path = Path.cwd() / ".env"
    if dotenv_path.exists():
        load_dotenv(dotenv_path=dotenv_path)
        self.api_key = os.getenv("VT_API_KEY")

    if not self.api_key:
        raise ValueError("API key not found. Provide VT_API_KEY via env or pass
api_key.")

    def _summarize_relevant_stats(self, relevant_results: List[Dict[str, Any]]) -> Dict[str,
int]:
        summary = {"malicious": 0, "suspicious": 0, "undetected": 0, "harmless": 0,
"timeout": 0}
        for r in relevant_results:
            category = (r.get("category") or "").lower()
            result = (r.get("result") or "").lower()
            if "malicious" in category or "phishing" in result:
                summary["malicious"] += 1
            elif "suspicious" in category:
                summary["suspicious"] += 1
            elif "harmless" in category or "clean" in result:
                summary["harmless"] += 1
            elif "undetected" in category or not result:
                summary["undetected"] += 1
            else:
                summary["timeout"] += 1
        return summary

    def check_url(self, url: str) -> Dict[str, Any]:
        with vt.Client(self.api_key) as client:
            try:
                url_id = vt.url_id(url)
                vt_url = client.get_object(f"/urls/{url_id}")
                all_results = vt_url.last_analysis_results

            except vt.error.APIError as e:
                if e.code == "NotFoundError":
                    logger.info(f"URL not found in VT, submitting for scan: {url}")
                    analysis = client.scan_url(url, wait_for_completion=True)
                    vt_analysis = client.get_object(f"/analyses/{analysis.id}")

                    if not hasattr(vt_analysis, "results") or not vt_analysis.results:

```

```

        return {"url": url, "error": "No analysis results found after scanning."}

    all_results = vt_analysis.results
    url_id = vt_analysis.id
    else:
        return {"url": url, "error": f"VT API error: {str(e)}"}

    relevant_results = []
    for engine_name, result in all_results.items():
        if engine_name in PHISHING_RELEVANT_ENGINES:
            relevant_results.append({
                "engine": engine_name,
                "category": result.get("category"),
                "result": result.get("result")
            })

    relevant_results.sort(
        key=lambda r: (
            0 if (r.get("category") == "malicious" or "phishing" in (r.get("result") or
"")) else 1
        )
    )

    stats = self._summarize_relevant_stats(relevant_results)
    safe = stats["malicious"] == 0 and stats["suspicious"] == 0

    time_now = datetime.now(UTC)
    time_exp = time_now + timedelta(days=EXPIRED_DAYS)
    return {
        "safe": safe,
        "stats": stats,
        "details_url": f"https://www.virustotal.com/gui/url/{url_id}",
        "raw": {"analysis_results": relevant_results},
        "checked_at": time_now.isoformat(),
        "expire_time": time_exp.isoformat()
    }

```

```

# app.py
import os
import logging
from pathlib import Path
from flask import Flask, request, jsonify
from dotenv import load_dotenv

```

```

from webrisk_client import WebRiskClient
from virustotal_client import VirusTotalClient

if not os.getenv("THREAT_INTEL_PORT") or not
os.getenv("THREAT_INTEL_HOST"):
    dotenv_path = Path(__file__).parent / ".env"
    if dotenv_path.exists():
        load_dotenv(dotenv_path=dotenv_path)

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s"
)
logger = logging.getLogger(__name__)

webrisk = WebRiskClient()
virustotal = VirusTotalClient()

app = Flask(__name__)

@app.route("/check", methods=["POST"])
def check_url():
    data = request.get_json()
    url = data.get("url")
    if not url:
        logger.warning("No URL provided in request")
        return jsonify({"error": "No URL provided"}), 400

    logger.info(f"Checking URL: {url}")
    try:
        wr_result = webrisk.check_url(url)
        vt_result = virustotal.check_url(url)

        safe = wr_result.get("safe", True) and vt_result.get("safe", True)
        logger.info(f"Check completed: {url} | Safe: {safe}")

    return jsonify({
        "url": url,
        "webrisk": wr_result,
        "virustotal": vt_result
    })

```

```

except Exception as e:
    logger.error(f'Error checking URL {url}: {e}')
    return jsonify({"error": str(e)}), 500

@app.route("/health", methods=["GET"])
def health():
    return jsonify({"status": "ok"}), 200

if __name__ == "__main__":
    port = int(os.getenv("THREAT_INTEL_PORT", 8001))
    host = os.getenv("THREAT_INTEL_HOST", "0.0.0.0")
    logger.info(f'Starting Threat Intelligence Service on {host}:{port}')
    app.run(host=host, port=port)

```

ДОДАТОК Б

Лістинг програмного коду сервісу аналізу домену

```
# domain_analyzer.py
import logging
import os
from datetime import datetime, timedelta, timezone
from pathlib import Path
from typing import Dict, Any, Optional, List
from urllib.parse import urlparse
import tldextract
import vt
from dotenv import load_dotenv

logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s]
%(message)s")
logger = logging.getLogger(__name__)

EXPIRED_DAYS = int(os.getenv("EXPIRED_DAYS", 7))

DOMAIN_RELEVANT_ENGINES = {
    "Abusix",
    "Acronis",
    "alphaMountain.ai",
    "Axur",
    "Bkav",
    "BitDefender",
    "ChainPatrol",
    "Chong Lua Dao",
    "CyRadar",
    "desenmascara.me",
    "Dr.Web",
    "Emsisoft",
    "ESET",
    "ESTsecurity",
    "Forcepoint ThreatSeeker",
    "Fortinet",
    "G-Data",
    "Google Safebrowsing",
    "Heimdal Security",
    "Kaspersky",
    "malwares.com URL checker",
    "Malwared",
```

```

"MalwarePatrol",
"MalwareURL",
"Mimecast",
"Netcraft",
"OpenPhish",
"PhishFort",
"PhishLabs",
"Phishing Database",
"Phishtank",
"Quttera",
"Quick Heal",
"SafeToOpen",
"Sansec eComscan",
"Sophos",
"Spam404",
"Sucuri SiteCheck",
"Trustwave",
"URLhaus",
"URLQuery",
"VIPRE",
"Webroot",
"Xcitium Verdict Cloud",
"Yandex Safebrowsing",
"ZeroFox",
}

```

```

class VirusTotalDomainClient:

```

```

    def __init__(self, api_key: Optional[str] = None, dotenv_path: Optional[Path] =
None):

```

```

        if api_key:

```

```

            self.api_key = api_key

```

```

        else:

```

```

            self.api_key = os.getenv("VT_API_KEY")

```

```

        if not self.api_key:

```

```

            if dotenv_path is None:

```

```

                dotenv_path = Path.cwd() / ".env"

```

```

            if dotenv_path.exists():

```

```

                load_dotenv(dotenv_path=dotenv_path)

```

```

                self.api_key = os.getenv("VT_API_KEY")

```

```

        if not self.api_key:

```

```

            raise ValueError("API key not found. Provide VT_API_KEY via env or pass

```

api_key.")

```

@staticmethod
def _extract_domain(url: str) -> str:
    parsed = urlparse(url)
    host = parsed.netloc or parsed.path
    ext = tldextract.extract(host)
    if not ext.domain:
        raise ValueError(f"Cannot extract domain from URL: {url}")
    return f"{ext.domain}.{ext.suffix}" if ext.suffix else ext.domain

@staticmethod
def _summarize_relevant_stats(relevant_results: List[Dict[str, Any]]) -> Dict[str, int]:
    summary = {"malicious": 0, "suspicious": 0, "undetected": 0, "harmless": 0,
"timeout": 0}
    for r in relevant_results:
        category = (r.get("category") or "").lower()
        result = (r.get("result") or "").lower()
        if "malicious" in category or "phishing" in result:
            summary["malicious"] += 1
        elif "suspicious" in category:
            summary["suspicious"] += 1
        elif "harmless" in category or "clean" in result:
            summary["harmless"] += 1
        elif "undetected" in category or not result:
            summary["undetected"] += 1
        else:
            summary["timeout"] += 1
    return summary

def check_domain(self, url: str) -> Dict[str, Any]:
    try:
        domain = self._extract_domain(url)
    except ValueError as e:
        logger.error(str(e))
        return {"url": url, "error": str(e)}

    with vt.Client(self.api_key) as client:
        try:
            vt_domain = client.get_object(f"/domains/{domain}")
            all_results = vt_domain.last_analysis_results
        except vt.error.APIError as e:
            return {"url": url, "domain": domain, "error": f"VT API error: {str(e)}"}

```

```

relevant_results = []
for engine_name, result in all_results.items():
    if engine_name in DOMAIN_RELEVANT_ENGINES:
        relevant_results.append({
            "engine": engine_name,
            "category": result.get("category"),
            "result": result.get("result")
        })

relevant_results.sort(
    key=lambda r: 0 if (r.get("category") == "malicious" or "phishing" in
(r.get("result") or "").lower()) else 1
)

stats = self._summarize_relevant_stats(relevant_results)
safe = stats["malicious"] == 0 and stats["suspicious"] == 0

time_now = datetime.now(timezone.utc)
time_exp = time_now + timedelta(days=EXPIRED_DAYS)
return {
    "domain": domain,
    "safe": safe,
    "stats": stats,
    "details_url": f"https://www.virustotal.com/gui/domain/{domain}",
    "raw": {"analysis_results": relevant_results},
    "last_checked_at": time_now.isoformat(),
    "expire_time": time_exp.isoformat()
}

# app.py
import os
import logging
from pathlib import Path
from flask import Flask, request, jsonify
from dotenv import load_dotenv

from domain_analyzer import VirusTotalDomainClient

if not os.getenv("DOMAIN_ANALYZER_PORT") or not
os.getenv("DOMAIN_ANALYZER_HOST"):
    dotenv_path = Path(__file__).parent / ".env"
    if dotenv_path.exists():

```

```

load_dotenv(dotenv_path=dotenv_path)

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s"
)
logger = logging.getLogger(__name__)

vt_domain_client = VirusTotalDomainClient()

app = Flask(__name__)

@app.route("/check_domain", methods=["POST"])
def check_domain():
    data = request.get_json()
    url = data.get("url")
    if not url:
        logger.warning("No URL provided in request")
        return jsonify({"error": "No URL provided"}), 400

    logger.info(f"Checking domain from URL: {url}")
    try:
        vt_result = vt_domain_client.check_domain(url)

        logger.info(f"Check completed: {url} | Safe: {vt_result.get('safe', True)}")
        last_checked_at = vt_result.get("last_checked_at", None)
        expire_time = vt_result.get("expire_time", None)
        return jsonify({
            "url": url,
            "virustotal_domain": vt_result,
            "last_checked_at": last_checked_at,
            "expire_time": expire_time,
            "safe": vt_result.get("safe", True)
        })

    except Exception as e:
        logger.error(f"Error checking domain {url}: {e}")
        return jsonify({"error": str(e)}), 500

@app.route("/health", methods=["GET"])
def health():
    return jsonify({"status": "ok"}), 200

```

```
if __name__ == "__main__":  
    port = int(os.getenv("DOMAIN_ANALYZER_PORT", 8002))  
    host = os.getenv("DOMAIN_ANALYZER_HOST", "0.0.0.0")  
    logger.info(f"Starting Domain Analyzer Service on {host}:{port}")  
    app.run(host=host, port=port)
```

ДОДАТОК В

Лістинг програмного коду сервісу контентного аналізу вебсторінок

```
# page_extractor.py
import json
import math
import random
import time
import logging
from typing import Dict, Any, List, Optional
from urllib.parse import urlparse
from io import BytesIO
import base64
import tempfile
from PIL import Image
import requests
from selenium import webdriver
from selenium.webdriver import ChromeOptions
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
from selenium.common.exceptions import WebDriverException, TimeoutException
from bs4 import BeautifulSoup, Comment

logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)

DEFAULT_USER_AGENTS = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
    Gecko) "
    "Chrome/120.0.0.0 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 14_0) AppleWebKit/605.1.15 (KHTML,
    like Gecko) "
    "Version/16.0 Safari/605.1.15",
]

DEFAULT_BLOCKLIST = [
    "*/www.google-analytics.com/*",
    "*/ssl.google-analytics.com/*",
    "*/www.googletagmanager.com/*",
    "*/googletagmanager.com/*",
    "*/www.facebook.com/*",
    "*/connect.facebook.net/*",
    "*/static.doubleclick.net/*",
    "*/ads*.google.com/*",
]
```

```

"*://pagead2.googlesyndication.com/*",
"*://www.googleadservices.com/*",
"*://analytics.google.com/*",
"*://cdn.segment.com/*",
"*://static.hotjar.com/*",
"*://script.hotjar.com/*",
"*://browser-update.org/*"
]

```

```
class PageExtractor:
```

```

    def __init__(
        self,
        headless: bool = False,
        user_agent: Optional[str] = None,
        accept_language: str = "en-US,en;q=0.9",
        blocklist: Optional[List[str]] = None,
        page_load_timeout: int = 20,
        script_timeout: int = 10,
        implicit_wait: int = 3,
    ):
        self.headless = headless
        self.user_agent = user_agent or random.choice(DEFAULT_USER_AGENTS)
        self.accept_language = accept_language
        self.blocklist = blocklist or DEFAULT_BLOCKLIST
        self.page_load_timeout = page_load_timeout
        self.script_timeout = script_timeout
        self.implicit_wait = implicit_wait

        self.driver: Optional[webdriver.Chrome] = None

    def _make_driver(self) -> webdriver.Chrome:
        options = ChromeOptions()
        tmp_profile = tempfile.mkdtemp()
        options.add_argument(f"--user-data-dir={tmp_profile}")

        options.add_argument("--window-size=1366,768")

        if self.headless:
            options.add_argument("--headless=new")

        options.add_argument("--disable-blink-features=AutomationControlled")

```

```

options.add_argument("--no-sandbox")
options.add_argument("--disable-dev-shm-usage")
options.add_experimental_option("excludeSwitches", ["enable-automation"])
options.add_experimental_option("useAutomationExtension", False)
options.add_argument("--disable-gpu")

caps = DesiredCapabilities.CHROME.copy()
caps["goog:loggingPrefs"] = {"performance": "ALL"}

driver = webdriver.Chrome(options=options)
driver.set_page_load_timeout(self.page_load_timeout)
driver.set_script_timeout(self.script_timeout)
driver.implicitly_wait(self.implicit_wait)

try:
    driver.execute_cdp_cmd(
        "Page.addScriptToEvaluateOnNewDocument",
        {
            "source": """
// overwrite the `navigator` properties to make automation less detectable
Object.defineProperty(navigator, 'webdriver', {get: () => undefined});
window.navigator.chrome = { runtime: {} };
Object.defineProperty(navigator, 'plugins', {get: () => [1,2,3,4,5]});
Object.defineProperty(navigator, 'languages', {get: () => ['en-US', 'en']}));
"""
        },
    )
except Exception as e:
    logger.debug("CDP script injection failed: %s", e)

try:
    driver.execute_cdp_cmd("Network.enable", {})
    driver.execute_cdp_cmd("Network.setBlockedURLs", {"urls": self.blocklist})
    headers = {"User-Agent": self.user_agent, "Accept-Language":
self.accept_language}
    driver.execute_cdp_cmd("Network.setExtraHTTPHeaders", {"headers":
headers})
except Exception as e:
    logger.debug("CDP network setup failed: %s", e)

return driver

def _human_interaction_sim(self, driver: webdriver.Chrome):

```

```

        try:
            js = """
(function(){
    function move() {
        var evt = new MouseEvent('mousemove', {
            view: window, bubbles: true, cancelable: true,
            clientX: Math.floor(Math.random()*800 + 100),
            clientY: Math.floor(Math.random()*400 + 100)
        });
        document.dispatchEvent(evt);
    }
    move();
})();
"""

            driver.execute_script(js)
            time.sleep(random.uniform(0.2, 0.8))
        except Exception:
            pass

def _extract_relevant(self, html: str, base_url: str) -> Dict[str, Any]:
    soup = BeautifulSoup(html, "html.parser")

    visible_text = soup.get_text(separator=" ", strip=True)
    visible_text = visible_text[:200000]

    forms = []
    for f in soup.find_all("form"):
        inputs = []
        for i in f.find_all("input"):
            inputs.append({"name": i.get("name"), "type": (i.get("type") or "").lower(),
"placeholder": i.get("placeholder")})
        forms.append({"action": f.get("action"), "method": (f.get("method") or
"get").lower(), "inputs": inputs})

    inputs_all = soup.find_all("input")
    num_password = sum(1 for i in inputs_all if (i.get("type") or "").lower() ==
"password")

    anchors = soup.find_all("a", href=True)
    hrefs = [a["href"] for a in anchors]
    parsed_base = urlparse(base_url)
    base_host = parsed_base.hostname or ""
    external_hosts = set()

```

```

for h in hrefs:
    try:
        p = urlparse(h)
        if p.hostname and p.hostname != base_host:
            external_hosts.add(p.hostname)
    except Exception:
        continue

scripts = soup.find_all("script")
script_info = {
    "num_scripts": len(scripts),
    "external_scripts": [s.get("src") for s in scripts if s.get("src")],
    "inline_script_length_max": max((len((s.get_text() or "")) for s in scripts),
default=0)
}

iframes = soup.find_all("iframe")
meta_robots = soup.find("meta", {"name": "robots"})
robots_noindex = False
if meta_robots and meta_robots.get("content") and "noindex" in
meta_robots.get("content").lower():
    robots_noindex = True

features = {
    "page_length_chars": len(html),
    "visible_text_length": len(visible_text),
    "num_forms": len(forms),
    "num_inputs": len(inputs_all),
    "num_password_inputs": num_password,
    "num_links": len(hrefs),
    "num_external_domains": len(external_hosts),
    "external_domains_sample": list(external_hosts)[:6],
    "meta_robots_noindex": robots_noindex,
    "script_info": script_info,
    "num_iframes": len(iframes),
    "forms_sample": forms[:3],
}

return {"text": visible_text, "features": features}

def _calc_compression_coef(self, width: int, height: int) -> float:
    total_pixels = width * height
    limit = 700_000

```

```

if total_pixels > limit:
    coef = math.sqrt(limit / total_pixels)
else:
    coef = 1.0

return max(coef, 0.05)

@staticmethod
def image_to_base64(source: str | bytes, max_width: int = 400, max_height: int =
400) -> str:
    try:
        if isinstance(source, str):
            resp = requests.get(source, timeout=5)
            resp.raise_for_status()
            img = Image.open(BytesIO(resp.content))
        else:
            img = Image.open(BytesIO(source))

        img.thumbnail((max_width, max_height))
        buffered = BytesIO()
        img.save(buffered, format="JPEG", quality=70)
        b64_str = base64.b64encode(buffered.getvalue()).decode("utf-8")
        logger.debug(f"Converted image to base64")
        return f"data:image/jpeg;base64,{b64_str}"
    except Exception as e:
        logger.debug(f"Failed to convert image to base64: {e}")
        return ""

@staticmethod
def clean_html_for_analysis(html: str) -> BeautifulSoup:
    soup = BeautifulSoup(html, "html.parser")

    for tag in soup([
        "script", "style", "link", "meta", "noscript", "canvas", "video", "svg"
    ]):
        tag.decompose()

    for tag in soup.find_all("script", {"type": "application/json"}):
        tag.decompose()

    for comment in soup.find_all(string=lambda x: isinstance(x, Comment)):

```

```

comment.extract()

important_attrs = {"type", "name", "placeholder", "value", "href", "id", "class",
"aria-label", "for", "role"}
for tag in soup.find_all():
    tag.attrs = {k: v for k, v in tag.attrs.items() if k in important_attrs}

return soup

def fetch(self, url: str, timeout_after_load: int = 2) -> Dict[str, Any]:
    result: Dict[str, Any] = {"url": url, "status": "error", "error": None}
    try:
        self.driver = self._make_driver()
    except WebDriverException as e:
        result["error"] = f'webdriver_init_failed: {e}'
        return result

    driver = self.driver
    try:
        time.sleep(random.uniform(0.3, 1.0))
        driver.get(url)
        self._human_interaction_sim(driver)
        time.sleep(timeout_after_load)

        final_url = driver.current_url
        raw_html = driver.page_source
        cookies = driver.get_cookies()

        total_height = driver.execute_script("return document.body.scrollHeight")
        total_width = driver.execute_script("return document.body.scrollWidth")
        driver.set_window_size(total_width, total_height)
        time.sleep(0.3)
        png = driver.get_screenshot_as_png()
        img = Image.open(BytesIO(png))

        coef = self._calc_compression_coef(img.width, img.height)
        screenshot_bytes = base64.b64decode(base64.b64encode(png))
        screenshot_base64 = self.image_to_base64(
            screenshot_bytes,
            max_width=int(img.width * coef),
            max_height=int(img.height * coef)
        )

```

```
# with open("base64_screenshot.txt", "w", encoding="utf-8") as f:
#     f.write(screenshot_base64)
```

```
raw_html = self.clean_html_for_analysis(raw_html).prettify()
soup = BeautifulSoup(raw_html, "html.parser")
visible_text = soup.get_text(separator=" ", strip=True)[:200000]
```

```
extracted = self._extract_relevant(raw_html, final_url)
```

```
result.update({
    "status": "ok",
    "final_url": final_url,
    "html": raw_html,
    "text": visible_text,
    "screenshot_base64": screenshot_base64,
    "features": extracted["features"],
    "cookies": cookies
})
return result
```

```
except TimeoutException as e:
    result["error"] = f'timeout: {e}'
    return result
except WebDriverException as e:
    result["error"] = f'webdriver_error: {e}'
    return result
except Exception as e:
    result["error"] = f'general_error: {e}'
    return result
finally:
    try:
        driver.quit()
    except Exception:
        pass
    self.driver = None
```

```
# app.py
import base64
import json
import os
import logging
import re
from pathlib import Path
from datetime import datetime, timedelta, timezone
```

```

from flask import Flask, request, jsonify
from dotenv import load_dotenv
from google.genai import types

from page_extractor import PageExtractor

try:
    from google import genai
except Exception:
    genai = None

if not os.getenv("CONTENT_ANALYZER_PORT") or not
os.getenv("CONTENT_ANALYZER_HOST"):
    dotenv_path = Path(__file__).parent / ".env"
    if dotenv_path.exists():
        load_dotenv(dotenv_path=dotenv_path)

# logging
logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s]
%(message)s")
logger = logging.getLogger(__name__)

EXPIRED_DAYS = int(os.getenv("EXPIRED_DAYS", 7))
GOOGLE_AI_API = os.getenv("GOOGLE_AI_API")
GOOGLE_AI_MODEL = os.getenv("GOOGLE_AI_MODEL", "gemini-2.5-flash-lite")
PAGE_FETCH_TIMEOUT = int(os.getenv("PAGE_FETCH_TIMEOUT", 2))

if not GOOGLE_AI_API:
    logger.warning("GOOGLE_AI_API is not set. AI calls will fail if attempted.")

_ai_client = None

def get_ai_client():
    global _ai_client
    if _ai_client is None:
        if genai is None:
            raise RuntimeError("google.genai client library is not installed or importable.")
        _ai_client = genai.Client(api_key=GOOGLE_AI_API)
    return _ai_client

CONTENT_SCHEMA = {
    "type": "object",

```

```

    "properties": {
        "is_phishing": {
            "type": "boolean",
            "description": "True if the page content represents phishing or impersonation,
otherwise False."
        },
        "explanation": {
            "type": "string",
            "description": "Explanation (4–6 sentences) describing why the content is or is
not phishing."
        }
    },
    "required": ["is_phishing", "explanation"],
    "additionalProperties": False,
}

```

```

app = Flask(__name__)
extractor = PageExtractor(headless=True)

```

```

def build_prompt(full_html: str, original_url: str, final_url: str) -> str:
    prompt = f"""
You are a security content analyzer.

```

Analyze the following FULL HTML page content. You may also use the provided PAGE SCREENSHOT IMAGE as supplemental information.

Your task:

Detect possible phishing or impersonation attempts based on both textual and visual indicators.

Consider:

- visible text or language suggesting urgency, fear, or credential harvesting ("verify", "confirm", "update", "enter password", "secure your account", etc.);
- presence of input fields for passwords, credit cards, or personal identifiers (<input>, <form> actions);
- deceptive links that point to external or mismatched domains;
- fake login prompts, cloned UI layouts, or reused branding;
- visual indicators of impersonation or suspicious branding (logos, trademarks, layouts), if clearly visible in the screenshot;
- suspicious or mismatched branding (e.g., the HTML claims to be from PayPal, but the logo or color palette doesn't match).

Your analysis should clearly state whether the page is likely involved in phishing or impersonation, and explain why, referencing concrete aspects of the HTML and, if relevant, the visual layout.

Original page URL: {original_url}

Final URL: {final_url}

HTML:

{full_html}

"""

return prompt.strip()

@app.route("/analyze_content", methods=["POST"])

def analyze_generic():

data = request.get_json(force=True)

html = data.get("html", "")

url = data.get("url", "")

screenshot_base64 = data.get("screenshot_base64", "")

original_url = url or data.get("original_url", "unknown")

final_url = data.get("final_url", original_url)

if not html and not url:

return jsonify({"error": "Must provide either 'url' or 'html'"}), 400

if url and not html:

logger.info("Fetching page from URL: %s", url)

fetch = extractor.fetch(url, timeout_after_load=PAGE_FETCH_TIMEOUT)

if fetch.get("status") != "ok":

logger.error("Page fetch failed: %s", fetch.get("error"))

return jsonify({"url": url, "error": fetch.get("error")}), 500

html = fetch.get("html", "")

final_url = fetch.get("final_url", url)

screenshot_base64 = fetch.get("screenshot_base64", "")

if not html:

return jsonify({"error": "No HTML to analyze"}), 400

logger.info("Analyzing page: %s", original_url)

prompt = build_prompt(html, original_url, final_url)

try:

client = get_ai_client()

```

except Exception as e:
    logger.exception("AI client init failed")
    return jsonify({"error": f"AI client init failed: {e}"}), 500

contents = [prompt]

if screenshot_base64.startswith("data:image"):
    match = re.match(r"data:(image/\w+);base64,(.*)", screenshot_base64)
    if match:
        mime_type, encoded = match.groups()
        image_bytes = base64.b64decode(encoded)
        image_part = types.Part.from_bytes(data=image_bytes, mime_type=mime_type)
        contents.append(image_part)

try:
    logger.info("Sending data to model: %s", GOOGLE_AI_MODEL)
    response_text = ""

    for chunk in client.models.generate_content_stream(
        model=GOOGLE_AI_MODEL,
        contents=contents,
        config={
            "temperature": 0.1,
            "top_p": 0.9,
            "max_output_tokens": 1500,
            "seed": 0,
            "response_mime_type": "application/json",
            "response_json_schema": CONTENT_SCHEMA,
        },
    ):
        if hasattr(chunk, "text") and chunk.text:
            response_text += chunk.text

    response_text = response_text.strip()
    data = json.loads(response_text)

    verdict_bool = bool(data.get("is_phishing", False))
    explanation = str(data.get("explanation", "")).strip()
    raw_ai = response_text
    now = datetime.now(timezone.utc)
    logger.debug("AI Response Parsed (JSON): verdict=%s, explanation=%s",
        verdict_bool, explanation)

    return jsonify({

```

```

        "verdict": verdict_bool,
        "explanation": explanation,
        "raw_ai": raw_ai,
        "screenshot_base64": screenshot_base64,
        "checked_at": now.isoformat(),
        "expire_time": (now + timedelta(days=EXPIRED_DAYS)).isoformat()
    })

except Exception as e:
    logger.exception("AI call failed")
    return jsonify({"error": f"AI call failed: {e}"}), 500

@app.route("/health", methods=["GET"])
def health():
    return jsonify({"status": "ok"}), 200

if __name__ == "__main__":
    port = int(os.getenv("CONTENT_ANALYZER_PORT", 8003))
    host = os.getenv("CONTENT_ANALYZER_HOST", "0.0.0.0")
    logger.info("Starting Page Analysis Service on %s:%s", host, port)
    app.run(host=host, port=port)

```

ДОДАТОК Г

Лістинг програмного коду сервісу поведінкового аналізу

```
# behavior_analyzer.py
import difflib
import os
import re
import json
import time
import math
import base64
import random
import logging
import tempfile
from io import BytesIO
from typing import List, Dict, Any, Optional

from dotenv import load_dotenv
from PIL import Image
from bs4 import BeautifulSoup, Comment
from selenium import webdriver
from selenium.webdriver import ChromeOptions
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

from google.genai import types

try:
    from google import genai
except Exception:
    genai = None

import requests

logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s]
%(message)s")
logger = logging.getLogger(__name__)

if not os.getenv("GOOGLE_AI_API") or not os.getenv("GOOGLE_AI_MODEL"):
```

```

dotenv_path = os.path.join(os.path.dirname(__file__), ".env")
if os.path.exists(dotenv_path):
    load_dotenv(dotenv_path=dotenv_path)

GOOGLE_AI_API = os.getenv("GOOGLE_AI_API")
GOOGLE_AI_MODEL = os.getenv("GOOGLE_AI_MODEL", "gemini-2.5-flash-lite")
PAGE_LOAD_TIMEOUT = int(os.getenv("PAGE_FETCH_TIMEOUT", "15"))
SCRIPT_TIMEOUT = int(os.getenv("SCRIPT_TIMEOUT", "10"))
IMPLICIT_WAIT = int(os.getenv("IMPLICIT_WAIT", "3"))
THRESHOLD_DOM_CHANGE = float(os.getenv("THRESHOLD_DOM_CHANGE",
"50.0"))

CONTENT_ANALYZER_CONTAINER_NAME =
os.getenv("CONTENT_ANALYZER_CONTAINER_NAME", "content-analyzer")
CONTENT_ANALYZER_PORT = int(os.getenv("CONTENT_ANALYZER_PORT",
8003))
CONTENT_ANALYZER_URL =
f"http://{CONTENT_ANALYZER_CONTAINER_NAME}:{CONTENT_ANALYZER
_PORT}/analyze_content"

DEFAULT_USER_AGENTS = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
    Gecko) Chrome/120.0.0.0 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 14_0) AppleWebKit/605.1.15 (KHTML,
    like Gecko) Version/16.0 Safari/605.1.15",
]

if not GOOGLE_AI_API:
    logger.warning("GOOGLE_AI_API is not set. AI calls will fail if attempted.")

ACTIONS_SCHEMA = {
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "action": {
                "type": "string",
                "description": "Type of user action: 'click' or 'fill'.",
                "enum": ["click", "fill"],
            },
        },
        "selector": {
            "type": "string",
            "description": (

```

```

        "Usable CSS selector for the target element. "
        "Should be directly usable with document.querySelector in the provided
HTML."
    ),
},

"selector_source": {
    "type": "string",
    "description": (
        "Source used to build the selector: 'id', 'name', 'aria_label', "
        "'label_for', 'placeholder', 'other_attribute', or 'text_match'."
    ),
    "enum": [
        "id",
        "name",
        "aria_label",
        "label_for",
        "placeholder",
        "other_attribute",
        "text_match",
    ],
},
"selector_attribute": {
    "type": "string",
    "description": (
        "If selector_source is attribute-based, this is the exact attribute name "
        "as it appears in HTML (e.g. 'id', 'name', 'aria-label', 'for')."
    ),
},
"selector_attribute_value": {
    "type": "string",
    "description": (
        "If selector_source is attribute-based, the exact attribute value as it "
        "appears in the HTML source, copied verbatim."
    ),
},
"field_label": {
    "type": "string",
    "description": (
        "Human-readable label/description of the element, used for UI. "
        "For fill: visible label/placeholder. "
        "For click: description of the button/link."
    ),
},

```

```

    },
    "selector_text": {
        "type": "string",
        "description": (
text)."
        ),
    },
    "value": {
        "type": "string",
        "description": "Realistic example value to fill into the field (for action='fill').",
    },
    },
    "required": ["action", "selector", "selector_source", "field_label"],
    "additionalProperties": False,
},
}

```

```

_ai_client = None
def get_ai_client():
    global _ai_client
    if _ai_client is None:
        if genai is None:
            raise RuntimeError("google.genai client library is not installed or importable.")
        _ai_client = genai.Client(api_key=GOOGLE_AI_API)
    return _ai_client

```

```

def _calc_compression_coef(width: int, height: int) -> float:
    total_pixels = width * height
    limit = 500_000
    return max(math.sqrt(limit / total_pixels) if total_pixels > limit else 1.0, 0.05)

```

```

def image_to_base64(source: str | bytes, max_width: int = 400, max_height: int = 400) -
> str:
    try:
        if isinstance(source, str):
            resp = requests.get(source, timeout=5)
            resp.raise_for_status()
            img = Image.open(BytesIO(resp.content))
        else:
            img = Image.open(BytesIO(source))

```

```

img.thumbnail((max_width, max_height))
buffered = BytesIO()
img.save(buffered, format="JPEG", quality=70)
b64_str = base64.b64encode(buffered.getvalue()).decode("utf-8")
logger.debug(f"Converted image to base64")
return f"data:image/jpeg;base64,{b64_str}"
except Exception as e:
    logger.debug(f"Failed to convert image to base64: {e}")
    return ""

def make_driver(headless=True, user_agent=None):
    options = ChromeOptions()

    profile_dir = tempfile.mkdtemp(prefix="chrome-profile-")
    options.add_argument(f"--user-data-dir={profile_dir}")

    options.add_argument("--window-size=1366,768")
    if headless:
        options.add_argument("--headless=new")

    options.add_argument("--disable-blink-features=AutomationControlled")
    options.add_argument("--no-sandbox")
    options.add_argument("--disable-dev-shm-usage")
    options.add_argument("--disable-infobars")
    options.add_argument("--disable-notifications")
    options.add_argument("--disable-gpu")
    options.add_argument("--remote-debugging-port=9222")
    options.add_experimental_option("excludeSwitches", ["enable-automation"])
    options.add_experimental_option("useAutomationExtension", False)

    if user_agent:
        options.add_argument(f"--user-agent={user_agent}")

    options.add_argument("--enable-logging")
    options.add_argument("--v=1")

    service = Service(log_path="/tmp/chromedriver.log")

    driver = webdriver.Chrome(service=service, options=options)

    driver.set_page_load_timeout(PAGE_LOAD_TIMEOUT)
    driver.set_script_timeout(SCRIPT_TIMEOUT)

```

```

driver.implicitly_wait(IMPLICIT_WAIT)
try:
    driver.execute_cdp_cmd(
        "Page.addScriptToEvaluateOnNewDocument",
        {"source": "Object.defineProperty(navigator, 'webdriver', {get: () =>
undefined});"},
    )
except Exception:
    pass

return driver

def human_like_movements(driver, iterations=3):
    try:
        for _ in range(iterations):
            x, y = random.randint(100, 900), random.randint(100, 700)
            driver.execute_script(f"""
                var ev = new MouseEvent('mousemove', {{bubbles:true, clientX:{x},
clientY:{y}}});
                document.dispatchEvent(ev);
            """)
            driver.execute_script(f"window.scrollBy(0, {random.randint(50, 300)});")
            time.sleep(random.uniform(0.3, 0.8))
    except Exception:
        pass

def get_full_dom(driver):
    import bs4

    def get_shadow_dom_script():
        return """
        function deepShadow(node) {
            let html = "";
            function traverse(n) {
                if (n.outerHTML) html += n.outerHTML;
                if (n.shadowRoot) {
                    html += "<shadow-root>";
                    Array.from(n.shadowRoot.children).forEach(traverse);
                    html += "</shadow-root>";
                }
                Array.from(n.children).forEach(traverse);
            }
        }
    """

```

```

        traverse(document.body);
        return html;
    }
    return deepShadow(document.body);
    """

def extract_iframes():
    iframe_htmls = []
    iframes = driver.find_elements(By.TAG_NAME, "iframe")
    for iframe in iframes:
        try:
            driver.switch_to.frame(iframe)
            inner = driver.execute_script("return
document.documentElement.outerHTML")
            iframe_htmls.append(inner)
            driver.switch_to.default_content()
        except Exception:
            continue
    return iframe_htmls

try:
    dom = driver.execute_script(get_shadow_dom_script())
    dom = bs4.BeautifulSoup(dom, "html.parser").prettify()
    iframes = extract_iframes()
    return dom + "\n" + "\n".join(iframes)
except Exception as e:
    print(f"Failed to get full DOM: {e}")
    return driver.page_source

def clean_html_for_analysis(html: str) -> BeautifulSoup:
    soup = BeautifulSoup(html, "html.parser")

    for tag in soup([
        "script", "style", "link", "meta", "noscript", "canvas", "video", "svg"
    ]):
        tag.decompose()

    for tag in soup.find_all("script", {"type": "application/json"}):
        tag.decompose()

    for comment in soup.find_all(string=lambda x: isinstance(x, Comment)):
        comment.extract()

```

```

important_attrs = {"type", "name", "placeholder", "value", "href", "id", "class", "aria-label", "for", "role"}
for tag in soup.find_all():
    tag.attrs = {k: v for k, v in tag.attrs.items() if k in important_attrs}

return soup

```

```
def ask_model_for_actions(html: str, screenshot_b64: str) -> Dict[str, Any]:
```

```
    try:
```

```
        client = get_ai_client()
```

```
        prompt = """
```

```
        You are simulating a naive human browsing a webpage.
```

```
        From the provided HTML and screenshot, extract ONLY the user actions that are
        clearly visible within the current viewport shown in the screenshot.
```

```
        Your task is to mimic what a real human would notice and interact with at first
        glance, with the goal of completing the MAIN user intent of the page (e.g. logging in,
        submitting a form, proceeding to the next step).
```

```
        Extract the following:
```

- Clickable elements: buttons, links, checkboxes, radio buttons, form submission elements that are fully visible in the screenshot and appear clearly interactable.
- Fillable fields: visible <input>, <textarea>, or other form elements with a visible label, placeholder, or descriptive context.

```
        You must return a list of actions, where each action describes either:
```

- a click on a clearly visible interactive element that is relevant to the main task (e.g. login, continue, submit, confirm), or
- filling a clearly visible form field with a realistic example value.

```
        STRICT SELECTOR RULES (MUST FOLLOW EXACTLY):
```

```
        1. Every selector MUST point to an element that actually exists in the provided
        HTML.
```

- Think as if you will later call document.querySelector(selector) on that HTML.
- Never invent attributes or values that are not present in the HTML.

```
        2. Selector building priority:
```

- First, try to use id → e.g. <input id="email-input"> → "input#email-input".
- If there is no id, use name → e.g. <input name="email"> → "input[name=\"email\"]".
- If there is no name, use aria-label → e.g. <button aria-label="Continue"> → "button[aria-label=\"Continue\"]".

- For label/field pairs, you may use the id referenced by label[for].
- Only if none of the above exist, you may use placeholder or another stable attribute.

3. For attribute-based selectors:

- Always use double quotes in CSS: [attr="value"].
- Copy attribute names and values EXACTLY as in HTML (case-sensitive, including non-ASCII characters).
- Do NOT change, translate, or normalize attribute values.

4. Avoid fragile selectors:

- Do NOT use :nth-child, :contains, long ancestor chains, or complex combinators.
- Prefer short, robust selectors like:
 - "button#login-btn"
 - "input[name=\"email\"]"
 - "button[aria-label=\"Continue\"]"

FOR EACH ACTION YOU MUST PROVIDE:

1. action: "click" or "fill".
2. selector: a usable CSS selector that matches an existing element in the HTML.
3. selector_source:
 - "id" → selector is primarily based on the id attribute.
 - "name" → selector is primarily based on the name attribute.
 - "aria_label" → selector is primarily based on the aria-label attribute.
 - "label_for" → selector is based on an id referenced by a label[for] attribute.
 - "placeholder" → selector is primarily based on the placeholder attribute.
 - "other_attribute" → selector is based on some other stable attribute.
 - "text_match" → selector is based mainly on visible text content.
4. If selector_source is one of: "id", "name", "aria_label", "label_for", "placeholder", "other_attribute":
 - You MUST set selector_attribute to the exact attribute name as in HTML (e.g. "id", "name", "aria-label", "for").
 - You MUST set selector_attribute_value to the exact attribute value, copied verbatim from HTML.
5. If selector_source is "text_match":
 - selector_attribute and selector_attribute_value may be empty or omitted.

LABELS AND VALUES:

1. field_label (for UI display) — REQUIRED for every action:
 - For action="fill": the user-facing label or placeholder of the field (e.g. "Email", "Номер картки", "Phone number").

- For action="click": a short human-readable description of the button or link (e.g. "Submit button", "Кнопка \"Увійти\"", "Continue link").
- 2. selector_text:
 - For action="click": the exact visible text shown on the button or link, if any.
 - For action="fill": may be empty or equal to the visible label if appropriate.
- 3. value:
 - Required ONLY for action="fill".
 - A realistic example value matching the language and context of field_label.
 - Examples:
 - Email: "example@example.com"
 - Phone: "380991234567"
 - Passport: "KB123456"
 - Card Number: "4000123456789010"
 - Expiry: "12/29"
 - CVV: "123"

VISIBILITY, RELEVANCE AND ORDER RULES:

1. Use ONLY elements that are clearly visible in the screenshot and located within the currently visible area.
2. Focus on actions that move the main task forward (e.g. logging in, sending a form, continuing a flow).
3. IGNORE purely auxiliary or cosmetic actions that do not change the main flow, such as:
 - "Show password" / "Hide password" toggles,
 - theme switches (dark/light mode),
 - language selectors (unless the ONLY visible primary action is to choose a language),
 - help/info icons that only open tooltips,
 - minor UI toggles that do not submit or change form data.
4. Include visible checkboxes and radio buttons only if they are clearly part of the main form logic (e.g. "I agree to terms", "Remember me").
5. Ignore hidden or collapsed content, modal windows, tabs, or sections not currently in the viewport.
6. Do NOT guess or include actions for elements that are not visible in the screenshot.
7. Do NOT repeat actions for the same element.

NATURAL SEQUENCE (ORDER MATTERS):

1. Your goal is to produce the minimal, realistic sequence of actions that a naive user would take to complete the main intent (e.g. successfully submit the login form once).
2. The order must be:

- First: all necessary fill actions for the main visible form, from top to bottom.
 - Then: any mandatory checkboxes or radio buttons directly related to that form (e.g. "I agree").
 - Finally: exactly one main submit/continue/login click that sends the form or moves to the next step.
3. Do NOT include clicks on:
- "Show password"/"Hide password" toggles,
 - secondary links like "Forgot password?", "Create account", unless they are the primary intent of the page,
 - extra navigation that is not part of the main flow.
4. If there are no visible user actions at all related to a main task, you must return an empty list [].

```

"""
contents = [prompt, html[:150000]]
if screenshot_b64.startswith("data:image"):
    match = re.match(r"data:(image/\w+);base64,(.*)", screenshot_b64)
    if match:
        mime_type, encoded = match.groups()
        img_bytes = base64.b64decode(encoded)
        img_part = types.Part.from_bytes(data=img_bytes, mime_type=mime_type)
        contents.append(img_part)

logger.info("Sending request to model %s for multi-action detection",
GOOGLE_AI_MODEL)
response_text = ""

for chunk in client.models.generate_content_stream(
    model=GOOGLE_AI_MODEL,
    contents=contents,
    config={
        "temperature": 0.05,
        "top_p": 0.9,
        "seed": 0,
        "response_mime_type": "application/json",
        "response_json_schema": ACTIONS_SCHEMA,
    },
):
    if hasattr(chunk, "text") and chunk.text:
        response_text += chunk.text

logger.info("Model response:\n %s\n-----", response_text)
response_text = response_text.strip()

```

```

try:
    actions = json.loads(response_text)
    if isinstance(actions, list):
        logger.info("Parsed actions: %d items", len(actions))
        if len(actions) == 0:
            return {"status": "no_actions", "actions": []}
        else:
            return {"status": "ok", "actions": actions}
    else:
        logger.warning("Model returned non-list JSON")
        return {"status": "error", "actions": [], "error": "Model output is not a list"}
except Exception as e:
    logger.warning("JSON parse failed: %s | text: %s", e, response_text[:300])
    return {"status": "error", "actions": [], "error": f"JSON parse failed: {e}"}

except Exception as e:
    logger.warning("ask_model_for_actions failed: %s", e)
    return {"status": "error", "actions": [], "error": str(e)}

def sanitize_filename(url: str) -> str:
    cleaned = re.sub(r"[^\w\-\_]", "_", url.replace("https://", "").replace("http://", ""))
    return cleaned[:150]

class BehavioralAnalyzer:
    def __init__(self, headless=True, user_agent=None):
        self.headless = headless
        self.user_agent = user_agent or random.choice(DEFAULT_USER_AGENTS)
        self.driver = None

    def _init_driver(self):
        self.driver = make_driver(self.headless, self.user_agent)

    def get_screenshot_base64(self) -> (str, bytes):
        driver = self.driver

        driver.execute_script("window.scrollTo(0, 0);")
        time.sleep(0.3)
        MAX_HEIGHT = 4000
        MAX_WIDTH = 2000
        total_height = driver.execute_script("return document.body.scrollHeight")

```

```
total_width = driver.execute_script("return document.body.scrollWidth")
driver.set_window_size(
    min(total_width, MAX_WIDTH),
    min(total_height, MAX_HEIGHT)
)
```

```
driver.execute_script("""
    document.body.style.overflow = 'hidden';
    document.body.style.position = 'relative';
""")
time.sleep(0.1)
```

```
png = driver.get_screenshot_as_png()
img = Image.open(BytesIO(png))
```

```
coef = _calc_compression_coef(img.width, img.height)
screenshot_bytes = base64.b64decode(base64.b64encode(png))
```

```
screenshot_b64 = image_to_base64(
    screenshot_bytes,
    max_width=int(img.width * coef),
    max_height=int(img.height * coef)
)
```

```
return screenshot_b64, png
```

```
@staticmethod
```

```
def calculate_dom_diff(before: List[str], after: List[str]) -> float:
```

```
    matcher = difflib.SequenceMatcher(None, before, after)
```

```
    similarity = matcher.ratio()
```

```
    delta_percent = round((1 - similarity) * 100, 2)
```

```
    return delta_percent
```

```
@staticmethod
```

```
def ask_content_analyzer(html: str, original_url: str, screenshot_b64: Optional[str] =
None) -> Dict[str, Any]:
```

```
    try:
```

```
        resp = requests.post(
            CONTENT_ANALYZER_URL,
            json={
                "html": html,
                "original_url": original_url,
                "screenshot_base64": screenshot_b64 or ""
            },
```

```

        timeout=50
    )
    resp.raise_for_status()
    return resp.json()
except Exception as e:
    logger.warning("Content analyzer request failed: %s", e)
    return {}

def perform_actions(self, actions: List[Dict[str, Any]]) -> List[Dict[str, Any]]:
    results = []
    for idx, action in enumerate(actions):
        try:
            selector = action.get("selector")
            if not selector:
                logger.warning("No selector provided for action: %s", action)
                continue

            if random.random() < 0.9:
                human_like_movements(self.driver, iterations=random.randint(1, 3))

            url_before = self.driver.current_url
            dom_before = get_full_dom(self.driver).splitlines()
            window_handles_before = self.driver.window_handles

            if action["action"] == "fill":
                value = action.get("value", "")
                logger.info("Filling [%s] with value [%s]", selector, value)
                element = self.driver.find_element(By.CSS_SELECTOR, selector)
                self.driver.execute_script("arguments[0].scrollIntoView({behavior:
'smooth', block: 'center'});",
                    element)
                element.clear()

                for char in value:
                    element.send_keys(char)
                    time.sleep(random.uniform(0.05, 0.15))

            elif action["action"] == "click":
                logger.info("Clicking on [%s]", selector)
                elements = self.driver.find_elements(By.CSS_SELECTOR, selector)
                if not elements:
                    raise ValueError(f"Element not found: {selector}")
                el = elements[0]

```

```

if not el.is_displayed():
    parent_label = self.driver.execute_script("""
        let input = arguments[0];
        while (input && input.tagName !== 'LABEL') {
            input = input.parentElement;
        }
        return input;
    """, el)
    if parent_label:
        logger.info("Clicking parent label for hidden input.")
        self.driver.execute_script("arguments[0].click();", parent_label)
    else:
        logger.warning("Element hidden and no label parent found.")
else:
    self.driver.execute_script(
        "arguments[0].scrollIntoView({behavior: 'smooth', block: 'center'});",
el)
    el.click()

time.sleep(random.uniform(0.8, 2.3))

alert_text = None
try:
    WebDriverWait(self.driver, 2).until(EC.alert_is_present())
    alert = self.driver.switch_to.alert
    alert_text = alert.text
    logger.info("Alert detected: %s", alert_text)
    alert.accept()
    logger.info("Alert accepted.")
except Exception:
    pass

url_after = self.driver.current_url
if url_before != url_after:
    time.sleep(15)
    url_after = self.driver.current_url
    logger.info("URL changed from %s to %s", url_before, url_after)

delta, content_report = 0.0, {}
if url_before == url_after:
    dom_after = get_full_dom(self.driver)
    delta = self.calculate_dom_diff(dom_before, dom_after.splitlines())
    if delta > THRESHOLD_DOM_CHANGE:

```

```

        logger.info("DOM changed by %.2f%% after action [%s]", delta,
action["action"])
        screen_b64, png_new = self.get_screenshot_base64()
        # with open(f"action_{idx+1}_screenshot.png", "wb") as f:
        #     f.write(png_new)
        dom_after = clean_html_for_analysis(dom_after).prettify()
        content_report = self.ask_content_analyzer(dom_after,
self.driver.current_url, screen_b64)
        logger.info("Content analyzer report: %s", content_report)

        window_handles_after = self.driver.window_handles
        new_window_opened = len(window_handles_after) >
len(window_handles_before)
        new_handles = list(set(window_handles_after) -
set(window_handles_before))
        new_window_urls = []
        for handle in new_handles:
            self.driver.switch_to.window(handle)
            new_window_urls.append(self.driver.current_url)
        self.driver.switch_to.window(window_handles_before[0])
        if new_window_opened:
            logger.info("New window(s) opened with URLs: %s", new_window_urls)

        results.append({"action": action,

                        "redirect": url_before != url_after,
                        "url_before": url_before,
                        "url_after": url_after,

                        "dom_changed": delta > THRESHOLD_DOM_CHANGE,
                        "dom_change_percent": delta,
                        "content_report": content_report,

                        "new_window_opened": bool(new_window_opened),
                        "new_window_urls": new_window_urls,

                        "alert_present": alert_text is not None,
                        "alert_text": alert_text
                        })

    if url_before != url_after:
        logger.info("Redirect detected, stopping further actions.")
        break

```

```

    except Exception as e:
        logger.warning("Failed to perform action %s: %s", action,
str(e).split("Stacktrace:")[0])

```

```

return results

```

```

def analyze(self, url: str, max_actions=20) -> Dict[str, Any]:
    try:
        self._init_driver()
        driver = self.driver
        driver.get(url)
        human_like_movements(driver)
    try:
        WebDriverWait(driver, 15).until(
            lambda d: d.execute_script("return document.readyState") == "complete"
        )
        time.sleep(5)
    except Exception:
        time.sleep(5)

    filename = sanitize_filename(url)
    screenshot_b64, png = self.get_screenshot_base64()
    # save png
    # with open(f'screenshot_{filename}.png', "wb") as f:
    #     f.write(png)

    html = get_full_dom(driver)
    soup = BeautifulSoup(html, "html.parser")
    html = soup.prettify()
    # with open(f'page_{filename}.html', "w", encoding="utf-8") as f:
    #     f.write(html)

    html = clean_html_for_analysis(html).prettify()
    res = ask_model_for_actions(html, screenshot_b64)
    if res.get("status") == "no_actions":
        logger.info("No user actions detected on the page.")
        return {"status": "no_actions", "actions": []}
    elif res.get("status") != "ok":
        logger.error("Action detection failed: %s", res.get("error"))
        return {"status": "error", "error": res.get("error")}

    actions = res.get("actions", [])[:max_actions]
    results = self.perform_actions(actions)

```

```

        return {"status": "ok", "results": results}

    except Exception as e:
        logger.error("Behavioral analysis failed: %s", e)
        return {"status": "error", "error": str(e)}

    finally:
        if self.driver:
            try:
                self.driver.quit()
                logger.info("ChromeDriver closed successfully.")
            except Exception as e:
                logger.warning("Failed to close ChromeDriver: %s", e)
            self.driver = None

# app.py
import json
import os
import logging
from flask import Flask, request, jsonify
from dotenv import load_dotenv
from behavior_analyzer import BehavioralAnalyzer
from datetime import datetime, timedelta, timezone

try:
    from google import genai
except Exception:
    genai = None

if not os.getenv("BEHAVIOR_ANALYZER_PORT") or not
os.getenv("BEHAVIOR_ANALYZER_HOST"):
    dotenv_path = os.path.join(os.path.dirname(__file__), ".env")
    if os.path.exists(dotenv_path):
        load_dotenv(dotenv_path=dotenv_path)

_ai_client = None
def get_ai_client():
    global _ai_client
    if _ai_client is None:
        if genai is None:
            raise RuntimeError("google.genai not installed")
        _ai_client = genai.Client(api_key=GOOGLE_AI_API)
    return _ai_client

```

```

app = Flask(__name__)
logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s]
%(message)s")
logger = logging.getLogger(__name__)

MAX_ATTEMPTS = int(os.getenv("MAX_ATTEMPTS", 3))
HEADLESS = os.getenv("HEADLESS", "true").lower() == "true"
EXPIRED_DAYS = int(os.getenv("EXPIRED_DAYS", 7))
GOOGLE_AI_API = os.getenv("GOOGLE_AI_API")
GOOGLE_AI_MODEL = os.getenv("GOOGLE_AI_MODEL", "gemini-2.5-flash-lite")

@app.route("/get_behavior", methods=["POST"])
def analyze():
    data = request.get_json(force=True)
    url = data.get("url")
    retries = data.get("retries", MAX_ATTEMPTS)

    if not url:
        return jsonify({"error": "Missing 'url'"}), 400

    logger.info("Starting behavioral analysis for: %s", url)
    results = []

    for attempt in range(retries):
        logger.info("Attempt %d of %d", attempt + 1, retries)
        analyzer = BehavioralAnalyzer(headless=HEADLESS)
        try:
            result = analyzer.analyze(url)
            results.append(result)
        except Exception as e:
            logger.warning("Attempt %d failed: %s", attempt + 1, e)

    def score(r):
        if r["status"] != "ok":
            return -1
        return sum([
            len(r.get("results", [])),
            sum(1 for a in r.get("results", []) if a.get("dom_changed")),
            sum(1 for a in r.get("results", []) if a.get("new_window_opened")),
        ])

    best = max(results, key=score, default=None)

```

```

if not best:
    return jsonify({"status": "error", "error": "All attempts failed."}), 500

# add checked_at and expire_time
now = datetime.now(timezone.utc)
best["checked_at"] = now.isoformat()
best["expire_time"] = (now + timedelta(days=EXPIRED_DAYS)).isoformat()

return jsonify(best)

```

```

def build_prompt(summary_payload: dict) -> str:
    origin = summary_payload.get("origin", "unknown origin")

    redirect_lines = []
    for step in summary_payload.get("redirect_chain", []):
        src = step.get("source")
        depth = step.get("depth")
        for child in step.get("children", []):
            redirect_lines.append(f"[depth={depth}] {src} → {child}")
    redirect_text = "\n".join(redirect_lines) if redirect_lines else "No redirects observed."

def integrate(entries, url_map, is_root=False):
    for entry in entries:
        url = entry.get("url")
        service = entry.get("service")
        data = entry.get("data", {})

        url_map.setdefault(url, {
            "type": "root" if is_root else "child",
            "domain": None,
            "content": None,
            "threat": None,
            "behavior": []
        })

    if service == "domain_analyzer":
        vtd = data.get("virustotal_domain", {})
        url_map[url]["domain"] = {
            "domain": vtd.get("domain"),
            "safe": vtd.get("safe"),
            "malicious": vtd.get("stats", {}).get("malicious", 0),
            "undetected": vtd.get("stats", {}).get("undetected", 0)
        }

```

```

    }

    elif service == "content_analyzer":
        url_map[url]["content"] = {
            "verdict": data.get("verdict"),
            "explanation": data.get("explanation", "").strip()
        }

    elif service == "threat_intel":
        vt = data.get("virustotal", {})
        url_map[url]["threat"] = {
            "safe": vt.get("safe"),
            "malicious": vt.get("stats", {}).get("malicious", 0),
            "harmless": vt.get("stats", {}).get("harmless", 0)
        }

    elif service == "behavior_analyzer":
        for act in data.get("results", []):
            url_map[url]["behavior"].append({
                "action": act["action"].get("action"),
                "field": act["action"].get("field_label") or act["action"].get("selector"),
                "value_filled": act["action"].get("value"),
                "redirect": act.get("redirect"),
                "new_window": act.get("new_window_opened"),
                "url_before": act.get("url_before"),
                "url_after": act.get("url_after"),
            })

url_map = {}
integrate(summary_payload.get("analyzed_root", []), url_map, is_root=True)
integrate(summary_payload.get("analyzed_children", []), url_map, is_root=False)

timeline = []
for url, info in url_map.items():
    role = "ROOT (initial page)" if info["type"] == "root" else "CHILD (redirect/new-
window)"
    section = f"URL: {url}\n Role: {role}"

    if info["domain"]:
        d = info["domain"]
        section += f"\n Domain reputation: {d['domain']} → safe={d['safe']}
(malicious={d['malicious']}, undetected={d['undetected']})"

    if info["threat"]:

```

```

    t = info["threat"]
    section += f"\n Threat intel: malicious={t['malicious']},
harmless={t['harmless']}, safe={t['safe']}"

    if info["content"]:
        c = info["content"]
        section += f"\n Content evaluation: {'PHISHING' if c['verdict'] else 'LEGIT'}
— {c['explanation']}"

    if info["behavior"]:
        section += "\n User interaction trace:"
        for b in info["behavior"]:
            section += f"\n   - Performed action: {b['action']} on '{b['field']}'" \
                f"{' (value=' + b['value_filled'] + ')' if b['value_filled'] else ''}" \
                f" → redirect={b['redirect']}, new_window={b['new_window']}" \
                f"\n     URL before: {b['url_before']}" \
                f"\n     URL after: {b['url_after']}"

        timeline.append(section)

    timeline_text = "\n\n".join(timeline)

    prompt = f"""
You are a professional cybersecurity analyst specializing in phishing detection,
credential harvesting analysis, and behavioral threat forensics.

Carefully analyze the entire chain of interaction, not only single-page indicators.
Pay specific attention to:
- Whether the user is guided toward entering login credentials
- How redirections change the domain trust profile over time
- Whether behavior resembles impersonation of a known service
- Whether redirects lead to legitimate authentication or mimicry / spoofing
- How content, domain reputation, and UI interactions reinforce each other

Start URL (initial user entry point):
{origin}

Observed redirection chain (chronological):
{redirect_text}

Cross-service evaluation and reconstructed user interaction timeline:
{timeline_text}

```

Based on this, decide whether the overall flow represents a coordinated phishing attempt aimed at credential theft, brand impersonation, or user deception. Your reasoning should be reflected in a clear, coherent explanation.

```

"""
.strip()

return prompt

@app.route("/get_summary", methods=["POST"])
def behavior_summary():
    data = request.get_json(force=True)
    summary_payload = data.get("summary_payload")

    if not summary_payload:
        return jsonify({"error": "missing summary_payload"}), 400

    prompt = build_prompt(summary_payload)

    try:
        client = get_ai_client()
        phishing_schema = {
            "type": "object",
            "properties": {
                "is_phishing": {
                    "type": "boolean",
                    "description": "True if this is phishing / credential theft flow, otherwise
False."
                },
                "explanation": {
                    "type": "string",
                    "description": "Detailed explanation (4–10 sentences) using concrete
evidence from the timeline."
                }
            },
            "required": ["is_phishing", "explanation"],
            "additionalProperties": False,
        }

        response = client.models.generate_content(
            model=GOOGLE_AI_MODEL,
            contents=prompt,
            config={
                "temperature": 0.1,
                "top_p": 0.9,

```

```

        "max_output_tokens": 1500,
        "seed": 0,
        "response_mime_type": "application/json",
        "response_json_schema": phishing_schema,
    },
)

data = json.loads(response.text)

verdict = bool(data.get("is_phishing", False))
explanation = str(data.get("explanation", "")).strip()
now = datetime.now(timezone.utc)

return jsonify({
    "verdict": verdict,
    "explanation": explanation,
    "raw_ai": response.text,
    "checked_at": now.isoformat(),
    "expire_time": (now + timedelta(days=EXPIRED_DAYS)).isoformat(),
    "summary_payload": summary_payload
})

except Exception as e:
    logger.exception("AI call failed")
    return jsonify({"error": str(e)}), 500

@app.route("/health", methods=["GET"])
def health():
    return jsonify({"status": "ok"}), 200

if __name__ == "__main__":
    port = int(os.getenv("BEHAVIOR_ANALYZER_PORT", 8004))
    host = os.getenv("BEHAVIOR_ANALYZER_HOST", "0.0.0.0")
    logger.info("Starting Behavior Analysis Service on %s:%s", host, port)
    app.run(host="0.0.0.0", port=8004)

```

ДОДАТОК Д

Лістинг програмного коду сервісу агрегатора результатів

```
# app.py
import os
import json
import logging
import requests
from flask import Flask, request, jsonify, Response
from dotenv import load_dotenv
from concurrent.futures import ThreadPoolExecutor, as_completed
from urllib.parse import urljoin, urlparse, urlunparse
import re
import idna

if not os.getenv("THREAT_INTEL_CONTAINER_NAME") or not
os.getenv("THREAT_INTEL_PORT"):
    dotenv_path = os.path.join(os.path.dirname(__file__), ".env")
    if os.path.exists(dotenv_path):
        load_dotenv(dotenv_path=dotenv_path)

NUM_REDIRECTS = int(os.getenv("NUM_REDIRECTS", 1))

app = Flask(__name__)
logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s]
%(message)s")
logger = logging.getLogger(__name__)

SERVICES = {
    "threat_intel": {
        "url":
f"http://{os.getenv('THREAT_INTEL_CONTAINER_NAME')}:{os.getenv('THREAT_
INTEL_PORT')}",
        "endpoint": "/check"
    },
    "domain_analyzer": {
        "url":
f"http://{os.getenv('DOMAIN_ANALYZER_CONTAINER_NAME')}:{os.getenv('DO
MAIN_ANALYZER_PORT')}",
        "endpoint": "/check_domain"
```

```

    },
    "content_analyzer": {
        "url":
f"http://{os.getenv('CONTENT_ANALYZER_CONTAINER_NAME')}:{os.getenv('C
CONTENT_ANALYZER_PORT')}",
        "endpoint": "/analyze_content"
    },
    "behavior_analyzer": {
        "url":
f"http://{os.getenv('BEHAVIOR_ANALYZER_CONTAINER_NAME')}:{os.getenv('B
EHAVIOR_ANALYZER_PORT')}",
        "endpoint": "/get_behavior"
    },
    "behavior_summary": {
        "url":
f"http://{os.getenv('BEHAVIOR_ANALYZER_CONTAINER_NAME')}:{os.getenv('B
EHAVIOR_ANALYZER_PORT')}",
        "endpoint": "/get_summary"
    }
}

```

```

HISTORY = {
    "check":
f"http://{os.getenv('HISTORY_CONTAINER_NAME')}:{os.getenv('HISTORY_SERV
ICE_PORT')}/check_history",
    "update":
f"http://{os.getenv('HISTORY_CONTAINER_NAME')}:{os.getenv('HISTORY_SERV
ICE_PORT')}/update_history"
}

```

```

WEIGHTS = {
    "threat_intel": float(os.getenv("THREAT_INTEL_WEIGHT", 0.15)),
    "domain_analyzer": float(os.getenv("DOMAIN_ANALYZER_WEIGHT", 0.15)),
    "content_analyzer": float(os.getenv("CONTENT_ANALYZER_WEIGHT", 0.3)),
    "behavior_analyzer": float(os.getenv("BEHAVIOR_ANALYZER_WEIGHT", 0.4)),
}

```

```

def resolve_final_url(url: str, max_steps: int = 5, timeout: int = 6) -> str:
    session = requests.Session()

    try:
        for _ in range(max_steps):
            resp = session.get(

```

```

        url,
        allow_redirects=False,
        timeout=timeout,
        headers={"User-Agent": "Mozilla/5.0"}
    )

    if "Location" not in resp.headers:
        return url

    url = urljoin(url, resp.headers["Location"])

    return url
except Exception:
    return url

def call_service(name: str, payload: dict) -> dict:
    service = SERVICES[name]
    url = service["url"] + service["endpoint"]

    # 1) history check
    try:
        history_resp = requests.post(
            HISTORY["check"],
            json={"service": name, "url": payload["url"]},
            timeout=5
        ).json()
    except Exception:
        history_resp = {"cached": False}
        logger.warning(f"[{name}] history check failed")

    if history_resp.get("cached"):
        return {"service": name, "cached": True, "data": history_resp["result"]}

    # 2) real call
    try:
        resp = requests.post(url, json=payload, timeout=240 if name ==
"behavior_analyzer" else 70).json()
        result = {"service": name, "cached": False, "data": resp}
    except Exception as e:
        result = {"service": name, "cached": False, "data": {"error": f"Service call failed:
{e}"}}

    # 3) history update

```

```

error = result.get("data", {}).get("error", None)
if not error:
    try:
        requests.post(
            HISTORY["update"],
            json={"service": name, "url": payload["url"], "result": result},
            timeout=5
        )
    except Exception:
        logger.warning(f"[{name}] history update failed")
else:
    logger.error(f"[{name}] service call failed: {error}")
    result = {"service": name, "cached": False, "data": {"error": error}}

return result

```

```

def extract_redirects(behavior_data: dict) -> list:
    redirects = set()
    if not isinstance(behavior_data, dict):
        return []
    for step in behavior_data.get("results", []):
        if step.get("redirect") and step.get("url_after"):
            redirects.add(step["url_after"])
        for w in step.get("new_window_urls", []):
            if w:
                redirects.add(w)
    return list(redirects)

```

```

def analyze_url_all_services(url: str, depth: int, source: str | None, stream_partial: bool):
    url = resolve_final_url(url)
    payload = {"url": url}
    full_results = {}
    enriched_list = []

```

```

    with ThreadPoolExecutor(max_workers=4) as executor:
        futures = {executor.submit(call_service, name, payload): name for name in
("threat_intel", "domain_analyzer", "content_analyzer", "behavior_analyzer")}
        for future in as_completed(futures):
            res = future.result()

            res_with_meta = {
                **res,

```

```

        "url": url,
        "depth": depth,
        "source": source
    }
    full_results[res["service"]] = res_with_meta
    enriched_list.append(res_with_meta)

    if stream_partial:
        yield ("partial", res_with_meta)

behavior = full_results.get("behavior_analyzer", {}).get("data", {})
behavior_redirects = extract_redirects(behavior)

yield ("complete", full_results, behavior_redirects, enriched_list)

def bfs_with_depth(origin_url: str):
    visited = set()
    redirects_chain = []
    accumulated_children = []
    root_behavior = []

    current_level = [(origin_url, 0, None)]
    while current_level:
        next_level = []
        for url, depth, source in current_level:
            if url in visited:
                continue
            visited.add(url)

            stream_partial = (depth == 0)

            last_complete = None
            for item in analyze_url_all_services(url, depth, source, stream_partial):
                if item[0] == "partial":
                    yield f"data: {json.dumps(item[1], ensure_ascii=False)}\n\n"
                else:
                    _, full_results, behavior_redirects, enriched_list = item
                    last_complete = (full_results, behavior_redirects, enriched_list)

            if not last_complete:
                continue

        full_results, behavior_redirects, enriched_list = last_complete

```

```

    if depth == 0:
        root_behavior = enriched_list
    else:
        accumulated_children.extend(enriched_list)

    if behavior_redirects:
        redirects_chain.append({
            "source": url,
            "depth": depth,
            "children": behavior_redirects
        })

    if depth < NUM_REDIRECTS:
        for child in behavior_redirects:
            if child not in visited:
                next_level.append((child, depth + 1, url))

    current_level = next_level

summary_payload = {
    "origin": origin_url,
    "redirect_chain": redirects_chain,
    "analyzed_root": root_behavior,
    "analyzed_children": accumulated_children,
    "depth_limit": NUM_REDIRECTS
}

summary_result = call_service(
    "behavior_summary",
    {"url": origin_url, "summary_payload": summary_payload}
)

summary = summary_result.get("data", summary_result)
cached = summary_result.get("cached", False)

final_event = {
    "service": "behavior_summary",
    "cached": cached,
    "url": origin_url,
    "depth": 0,
    "source": None,
    "data": summary
}

```

```
yield f'data: {json.dumps(final_event, ensure_ascii=False)}\n\n'
```

```
def normalize_url(url: str) -> str:
```

```
    if not url:
        return url
```

```
    url = url.strip()
```

```
    url = re.sub(r"\s+", "", url)
```

```
    if not re.match(r"^[a-zA-Z][a-zA-Z0-9+\-\.]*://", url):
```

```
        url = "https://" + url
```

```
    parsed = urlparse(url)
```

```
    scheme = "https" if parsed.scheme.lower() in ["http", "https"] else parsed.scheme
```

```
    netloc = parsed.netloc.lower()
```

```
    if netloc.endswith(":80"):
```

```
        netloc = netloc[:-3]
```

```
    elif netloc.endswith(":443"):
```

```
        netloc = netloc[:-4]
```

```
    path = parsed.path or ""
```

```
    if path != "/":
```

```
        path = path.rstrip("/")
```

```
    normalized = urlunparse((
```

```
        scheme,
```

```
        netloc,
```

```
        path if path else "/",
```

```
        parsed.params,
```

```
        parsed.query,
```

```
        parsed.fragment
```

```
    ))
```

```
    return normalized
```

```
def is_valid_unicode_domain(domain: str) -> bool:
```

```
    domain = domain.strip().rstrip(".")
```

```
    if not domain:
```

```
        return False
```

```
    if "." not in domain:
```

```

    return False

labels = domain.split(".")
if len(labels) < 2:
    return False

for label in labels:
    if not label:
        return False

    if len(label) > 63:
        return False

    try:
        idna.encode(label)
    except idna.IDNAError:
        return False

tld = labels[-1]

if len(tld) < 2:
    return False

if not any(c.isalpha() for c in tld):
    return False

return True


@app.route("/check", methods=["POST"])
def check():
    input_data = request.get_json(force=True)
    if not input_data or "url" not in input_data:
        return jsonify({"error": "Expected JSON: {"url": "https://example.com"}"}),
400

    origin_url = normalize_url(input_data["url"])

    parsed = urlparse(origin_url)
    host = parsed.netloc

    if not is_valid_unicode_domain(host):
        return jsonify({"error": "Invalid URL provided"}), 400

```

```
return Response(bfs_with_depth(origin_url), mimetype="text/event-stream")
```

```
@app.route("/score", methods=["POST"])
def score():
    try:
        payload = request.get_json(force=True)
    except Exception:
        return jsonify({"error": "Invalid JSON"}), 400

    values = payload.get("values") or {}
    if not isinstance(values, dict) or not values:
        return jsonify({"error": "Expected JSON with 'values' object"}), 400

    weight_by_service = {
        "threat_intel": WEIGHTS.get("threat_intel"),
        "domain_analyzer": WEIGHTS.get("domain_analyzer"),
        "content_analyzer": WEIGHTS.get("content_analyzer"),
        "behavior_summary": WEIGHTS.get("behavior_analyzer"),
    }

    component_scores = []

    if "threat_intel" in values:
        ti = values["threat_intel"] or {}
        vt_stats = ti.get("vt_stats") or {}
        webrisk_safe = ti.get("webrisk_safe", None)

        harmless = int(vt_stats.get("harmless", 0))
        suspicious = int(vt_stats.get("suspicious", 0))
        malicious = int(vt_stats.get("malicious", 0))
        undetected = int(vt_stats.get("undetected", 0))
        total = harmless + suspicious + malicious + undetected

        vt_risk = None
        if total > 0:
            vt_risk = (suspicious + malicious) / float(total)

        wr_risk = None
        if webrisk_safe is True:
            wr_risk = 0.0
        elif webrisk_safe is False:
            wr_risk = 1.0
```

```

ti_risk = None
if vt_risk is not None and wr_risk is not None:
    ti_risk = 0.5 * vt_risk + 0.5 * wr_risk
elif vt_risk is not None:
    ti_risk = vt_risk
elif wr_risk is not None:
    ti_risk = wr_risk

if ti_risk is not None:
    w = weight_by_service["threat_intel"]
    component_scores.append((
        "Перевірка в базах загроз",
        max(0.0, min(1.0, ti_risk)),
        w,
        f'VirusTotal: {suspicious + malicious}/{max(total, 1)} підозрілі/всі; "
        f'WebRisk: {'безпечний' if webrisk_safe else 'небезпечний' if webrisk_safe
is False else 'немає даних'}.'"
    ))

if "domain_analyzer" in values:
    da = values["domain_analyzer"] or {}
    stats = da.get("vt_domain_stats") or {}
    harmless = int(stats.get("harmless", 0))
    suspicious = int(stats.get("suspicious", 0))
    malicious = int(stats.get("malicious", 0))
    undetected = int(stats.get("undetected", 0))
    total = harmless + suspicious + malicious + undetected

    if total > 0:
        risk = (suspicious + malicious) / float(total)
        w = weight_by_service["domain_analyzer"]
        component_scores.append((
            "Доменна перевірка",
            risk,
            w,
            f'Доменний аналіз: {suspicious + malicious}/{total} підозрілі/всі.'"
        ))

if "content_analyzer" in values:
    ca = values["content_analyzer"] or {}
    verdict = ca.get("verdict", None)
    if isinstance(verdict, bool):
        risk = 1.0 if verdict else 0.0
        w = weight_by_service["content_analyzer"]

```

```

        component_scores.append((
            "Перевірка контенту",
            risk,
            w,
            f'Контент: {'фішингові ознаки виявлено' if verdict else 'ознаки фішингу
не виявлено'}.'"
        ))

```

```

if "behavior_summary" in values:
    ba = values["behavior_summary"] or {}
    verdict = ba.get("verdict", None)
    if isinstance(verdict, bool):
        risk = 1.0 if verdict else 0.0
        w = weight_by_service["behavior_summary"]
        component_scores.append((
            "Аналіз поведінки",
            risk,
            w,
            f'Поведінка: {'висока ймовірність фішингу' if verdict else 'не виявлено
ризикової активності'}.'"
        ))

```

```

valid_total_weight = sum(w for _, _, w, _ in component_scores)

```

```

if valid_total_weight == 0:
    return jsonify({
        "integro_score": "N/A",
        "explanation": "Недостатньо даних.",
        "formula": "N/A"
    }), 200

```

```

weighted_sum = 0.0
formula_parts_raw = []
normalized_parts = []

```

```

for svc_name, risk, weight, _ in component_scores:
    w_norm = weight / valid_total_weight
    weighted_sum += risk * w_norm

    formula_parts_raw.append(f'{weight:.2f} · {risk:.2f}')
    normalized_parts.append(f'{w_norm:.2f} · {risk:.2f}')

```

```

final_score = round(weighted_sum * 100, 1)

```

```

formula = (
    " + ".join(normalized_parts)
    + f" = {weighted_sum:.4f} → {final_score}"
)

parts = []
for svc_name, risk, weight, text in component_scores:
    perc = int(risk * 100)
    wperc = int(100 * (weight / valid_total_weight))
    parts.append(
        f"- {svc_name}: ризик {perc}%, внесок у підсумок {wperc}% → {text}"
    )

explanation = (
    f"Оцінка = {final_score} із 100.\n"
    "Чим вище значення — тим більший сумарний ризик.\n\n"
    + "\n".join(parts)
)

return jsonify({
    "integro_score": final_score,
    "explanation": explanation,
    "formula": formula
}), 200

@app.route("/health", methods=["GET"])
def health():
    return jsonify({"status": "ok"})

if __name__ == "__main__":
    RISK_AGGREGATOR_PORT = int(os.getenv("RISK_AGGREGATOR_PORT",
    8000))
    RISK_AGGREGATOR_HOST = os.getenv("RISK_AGGREGATOR_HOST",
    "0.0.0.0")
    app.run(host=RISK_AGGREGATOR_HOST, port=RISK_AGGREGATOR_PORT)

```

ДОДАТОК Е

Лістинг програмного коду сервісу взаємодії з базою даних

```
# app.py
import os
from datetime import datetime, timedelta
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.dialects.postgresql import JSONB
import logging
from urllib.parse import urlparse
from sqlalchemy.exc import SQLAlchemyError
import tldextract

logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s]
%(message)s")
logger = logging.getLogger(__name__)

app = Flask(__name__)

app.config["SQLALCHEMY_DATABASE_URI"] =
os.getenv("HISTORY_DATABASE_URL")
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False

db = SQLAlchemy(app)

# urls
class URL(db.Model):
    __tablename__ = "urls"
    id = db.Column(db.Integer, primary_key=True)
    url = db.Column(db.String, unique=True, nullable=False)
    first_seen_at = db.Column(db.DateTime, default=datetime.utcnow)
    last_checked_at = db.Column(db.DateTime)
    expire_at = db.Column(db.DateTime)

# domain
class Domain(db.Model):
    __tablename__ = "domains"
    id = db.Column(db.Integer, primary_key=True)
    domain = db.Column(db.String, unique=True, nullable=False)
    first_seen_at = db.Column(db.DateTime, default=datetime.utcnow)
```

```
last_checked_at = db.Column(db.DateTime)
expire_at = db.Column(db.DateTime)
```

```
# threat_intel result
```

```
class ThreatIntelResult(db.Model):
    __tablename__ = "threat_intel_results"
    id = db.Column(db.Integer, primary_key=True)
    url_id = db.Column(db.Integer, db.ForeignKey("urls.id"), nullable=False)
    vt_safe = db.Column(db.Boolean)
    vt_malicious = db.Column(db.Integer)
    vt_undetected = db.Column(db.Integer)
    webrisk_safe = db.Column(db.Boolean)
    raw = db.Column(JSONB)
    last_checked_at = db.Column(db.DateTime)
    expire_at = db.Column(db.DateTime)
```

```
# domain analyzer
```

```
class DomainAnalyzerResult(db.Model):
    __tablename__ = "domain_analyzer_results"
    id = db.Column(db.Integer, primary_key=True)
    domain_id = db.Column(db.Integer, db.ForeignKey("domains.id"), nullable=False)
    safe = db.Column(db.Boolean)
    malicious = db.Column(db.Integer)
    undetected = db.Column(db.Integer)
    raw = db.Column(JSONB)
    last_checked_at = db.Column(db.DateTime)
    expire_at = db.Column(db.DateTime)
```

```
# content analyzer
```

```
class ContentAnalyzerResult(db.Model):
    __tablename__ = "content_analyzer_results"
    id = db.Column(db.Integer, primary_key=True)
    url_id = db.Column(db.Integer, db.ForeignKey("urls.id"), nullable=False)
    verdict = db.Column(db.Boolean)
    explanation = db.Column(db.Text)
    raw_ai = db.Column(db.Text)
    raw = db.Column(JSONB)
    last_checked_at = db.Column(db.DateTime)
    expire_at = db.Column(db.DateTime)
```

```
# behavior actions
```

```
class BehaviorAction(db.Model):
    __tablename__ = "behavior_actions"
    id = db.Column(db.Integer, primary_key=True)
```

```

url_id = db.Column(db.Integer, db.ForeignKey("urls.id"), nullable=False)
actions_json = db.Column(JSONB)
raw = db.Column(JSONB)
last_checked_at = db.Column(db.DateTime)
expire_at = db.Column(db.DateTime)

# behavior summary
class BehaviorSummary(db.Model):
    __tablename__ = "behavior_summary"
    id = db.Column(db.Integer, primary_key=True)
    url_id = db.Column(db.Integer, db.ForeignKey("urls.id"), nullable=False)
    verdict = db.Column(db.Boolean)
    explanation = db.Column(db.Text)
    raw_ai = db.Column(db.Text)
    raw = db.Column(JSONB)
    last_checked_at = db.Column(db.DateTime)
    expire_at = db.Column(db.DateTime)

def get_or_create_url(url):
    record = URL.query.filter_by(url=url).first()
    if record:
        return record
    record = URL(url=url)
    db.session.add(record)
    db.session.commit()
    return record

def is_expired(record: URL):
    return record.expire_at and record.expire_at < datetime.utcnow()

def extract_domain(url: str) -> str:
    parsed = urlparse(url)
    host = parsed.netloc or parsed.path
    ext = tldextract.extract(host)
    if not ext.domain:
        raise ValueError(f"Cannot extract domain from URL: {url}")
    return f"{ext.domain}.{ext.suffix}" if ext.suffix else ext.domain

@app.route("/check_history", methods=["POST"])
def check_history():
    data = request.get_json(force=True)

```

```

service = data.get("service")
url = data.get("url")

if not service or not url:
    return jsonify({"cached": False, "error": "missing service or url"}), 400

if service in ("threat_intel", "content_analyzer", "behavior_analyzer",
"behavior_summary"):
    record = URL.query.filter_by(url=url).first()
    if not record or is_expired(record):
        return jsonify({"cached": False})

    if service == "threat_intel":
        r = ThreatIntelResult.query.filter_by(url_id=record.id).first()
    elif service == "content_analyzer":
        r = ContentAnalyzerResult.query.filter_by(url_id=record.id).first()
    elif service == "behavior_analyzer":
        r = BehaviorAction.query.filter_by(url_id=record.id).first()
    elif service == "behavior_summary":
        r = BehaviorSummary.query.filter_by(url_id=record.id).first()
    else:
        r = None

    if not r or not r.expire_at or r.expire_at < datetime.utcnow():
        return jsonify({"cached": False})

    return jsonify({"cached": True, "result": r.raw})

elif service == "domain_analyzer":
    domain = extract_domain(url)
    if not domain:
        return jsonify({"cached": False}), 400

    domain_record = Domain.query.filter_by(domain=domain).first()
    if not domain_record:
        return jsonify({"cached": False}), 400

    r = DomainAnalyzerResult.query.filter_by(domain_id=domain_record.id).first()
    if not r or not r.expire_at or r.expire_at < datetime.utcnow():
        return jsonify({"cached": False})

    return jsonify({"cached": True, "result": r.raw})

```

```
return jsonify({"cached": False, "error": "unknown service"}), 400
```

```
def parse_iso_datetime(s):
    if not s:
        return None
    try:
        dt = datetime.fromisoformat(s)
        if dt.tzinfo:
            dt = dt.astimezone(datetime.timezone.utc).replace(tzinfo=None)
        return dt
    except Exception:
        try:
            dt = datetime.strptime(s.split('.')[0], "%Y-%m-%dT%H:%M:%S")
            return dt
        except Exception:
            return None
```

```
@app.route("/update_history", methods=["POST"])
def update_history():
    payload = request.get_json(force=True)
    service = payload.get("service")
    url = payload.get("url")
    result_wrapper = payload.get("result") or payload.get("data") or payload

    if not service or not url or not result_wrapper:
        return jsonify({"status": "error", "error": "missing service, url or result"}), 400

    service_data = result_wrapper.get("data") if isinstance(result_wrapper, dict) and
    "data" in result_wrapper else result_wrapper
```

```
try:
    url_record = get_or_create_url(url)
    url_record.last_checked_at = datetime.utcnow()
except SQLAlchemyError as e:
    db.session.rollback()
    logger.exception("Failed to get/create url")
    return jsonify({"status": "error", "error": "db_url_error", "detail": str(e)}), 500
```

```
def bump_url_expire(new_dt):
    if not new_dt:
        return
    if new_dt.tzinfo:
```

```

    new_dt = new_dt.astimezone(datetime.timezone.utc).replace(tzinfo=None)
    if not url_record.expire_at or new_dt > url_record.expire_at:
        url_record.expire_at = new_dt

    try:
        if service == "domain_analyzer":
            virustotal_domain = service_data.get("virustotal_domain", {}) if
isinstance(service_data, dict) else {}
            domain_name = virustotal_domain.get("domain") or extract_domain(url)
            if not domain_name:
                return jsonify({"status": "error", "error": "cannot_extract_domain"}), 400

            domain_record = Domain.query.filter_by(domain=domain_name).first()
            if not domain_record:
                domain_record = Domain(domain=domain_name,
first_seen_at=datetime.utcnow())
                db.session.add(domain_record)
                db.session.flush()

            domain_record.last_checked_at = parse_iso_datetime(
                virustotal_domain.get("last_checked_at")) or datetime.utcnow()
            domain_expire = parse_iso_datetime(virustotal_domain.get("expire_time"))
            if domain_expire:
                domain_record.expire_at = domain_expire
                db.session.add(domain_record)

            dar =
DomainAnalyzerResult.query.filter_by(domain_id=domain_record.id).first()
            if not dar:
                dar = DomainAnalyzerResult(domain_id=domain_record.id)
                db.session.add(dar)

            dar.safe = service_data.get("safe")
            stats = virustotal_domain.get("stats", {})
            dar.malicious = stats.get("malicious", 0)
            dar.undetected = stats.get("undetected", 0)

            dar.raw = service_data

            dar.last_checked_at = domain_record.last_checked_at
            dar.expire_at = domain_expire

            bump_url_expire(domain_expire)

```

```

elif service == "threat_intel":
    # service_data expected: {"url":..., "virustotal": {...}, "webrisk": {...}}
    vt = service_data.get("virustotal", {}) if isinstance(service_data, dict) else {}
    webrisk = service_data.get("webrisk", {}) if isinstance(service_data, dict) else {}

    url_record.last_checked_at = parse_iso_datetime(vt.get("checked_at")) or
    parse_iso_datetime(webrisk.get("checked_at")) or url_record.last_checked_at

    tir = ThreatIntelResult.query.filter_by(url_id=url_record.id).first()
    if not tir:
        tir = ThreatIntelResult(url_id=url_record.id)
        db.session.add(tir)

    tir.vt_safe = vt.get("safe")
    tir.vt_malicious = vt.get("stats", {}).get("malicious") if isinstance(vt.get("stats"),
dict) else None
    tir.vt_undetected = vt.get("stats", {}).get("undetected") if
isinstance(vt.get("stats"), dict) else None
    tir.webrisk_safe = webrisk.get("safe") if webrisk else None

    tir.raw = service_data
    tir.last_checked_at = parse_iso_datetime(vt.get("checked_at")) or
    parse_iso_datetime(webrisk.get("checked_at")) or datetime.utcnow()
    tir.expire_at = parse_iso_datetime(vt.get("expire_time")) or
    parse_iso_datetime(webrisk.get("expire_time"))

    bump_url_expire(tir.expire_at)

elif service == "content_analyzer":
    # service_data expected: {"checked_at": "...", "expire_time": "...", "explanation":
"...", "raw_ai": "...", "verdict": true}
    checked = parse_iso_datetime(service_data.get("checked_at"))
    expire = parse_iso_datetime(service_data.get("expire_time"))

    car = ContentAnalyzerResult.query.filter_by(url_id=url_record.id).first()
    if not car:
        car = ContentAnalyzerResult(url_id=url_record.id)
        db.session.add(car)

    car.verdict = service_data.get("verdict")
    car.explanation = service_data.get("explanation")

    car.raw_ai = service_data.get("raw_ai") or service_data.get("rawAI") or None

```

```

car.raw = service_data
car.last_checked_at = checked or datetime.utcnow()
car.expire_at = expire

bump_url_expire(car.expire_at)

elif service == "behavior_analyzer":
    # service_data expected: {"results": [...], "status": "ok"}
    results = service_data.get("results") if isinstance(service_data, dict) else None
    checked = parse_iso_datetime(service_data.get("checked_at")) or
datetime.utcnow()
    expire = parse_iso_datetime(service_data.get("expire_time"))

    ba = BehaviorAction.query.filter_by(url_id=url_record.id).first()
    if not ba:
        ba = BehaviorAction(url_id=url_record.id)
        db.session.add(ba)

    ba.actions_json = results if results is not None else service_data
    ba.raw = service_data
    ba.last_checked_at = checked
    ba.expire_at = expire

    bump_url_expire(ba.expire_at)

elif service == "behavior_summary":
    checked = parse_iso_datetime(service_data.get("checked_at"))
    expire = parse_iso_datetime(service_data.get("expire_time"))

    bs = BehaviorSummary.query.filter_by(url_id=url_record.id).first()
    if not bs:
        bs = BehaviorSummary(url_id=url_record.id)
        db.session.add(bs)

    bs.verdict = service_data.get("verdict")
    bs.explanation = service_data.get("explanation")
    bs.raw_ai = service_data.get("raw_ai") or service_data.get("rawAI")
    bs.raw = service_data
    bs.last_checked_at = checked or datetime.utcnow()
    bs.expire_at = expire

    bump_url_expire(bs.expire_at)

else:

```

```

        return jsonify({"status": "error", "error": f"unknown service {service}"}, 400

    if not url_record.first_seen_at:
        url_record.first_seen_at = datetime.utcnow()
    url_record.last_checked_at = datetime.utcnow()
    db.session.add(url_record)
    db.session.commit()

except SQLAlchemyError as e:
    db.session.rollback()
    logger.exception("DB error during update_history")
    return jsonify({"status": "error", "error": "db_error", "detail": str(e)}), 500
except Exception as e:
    db.session.rollback()
    logger.exception("Unexpected error in update_history")
    return jsonify({"status": "error", "error": "internal_error", "detail": str(e)}), 500

return jsonify({"status": "ok", "saved_for": service, "url_id": url_record.id})

@app.route("/delete_url", methods=["DELETE"])
def delete_url():
    data = request.get_json(force=True)
    url = data.get("url")

    if not url:
        return jsonify({"status": "error", "error": "missing url"}), 400

    try:
        url_record = URL.query.filter_by(url=url).first()
        if not url_record:
            return jsonify({"status": "error", "error": "url_not_found"}), 404

        url_id = url_record.id

        ThreatIntelResult.query.filter_by(url_id=url_id).delete()
        ContentAnalyzerResult.query.filter_by(url_id=url_id).delete()
        BehaviorAction.query.filter_by(url_id=url_id).delete()
        BehaviorSummary.query.filter_by(url_id=url_id).delete()

        db.session.delete(url_record)

    try:
        domain_name = extract_domain(url)

```

```

domain_record = Domain.query.filter_by(domain=domain_name).first()

if domain_record:
    other_urls_exist = (
        db.session.query(URL)
        .filter(URL.url.like(f"%{domain_name}%"))
        .count()
    )

    if other_urls_exist == 0:

DomainAnalyzerResult.query.filter_by(domain_id=domain_record.id).delete()
    db.session.delete(domain_record)

except Exception:
    logger.warning(f"Domain cleanup failed for {url}")

db.session.commit()

return jsonify({
    "status": "ok",
    "deleted_url": url,
    "deleted_url_id": url_id
})

except SQLAlchemyError as e:
    db.session.rollback()
    logger.exception("DB error during delete_url")
    return jsonify({"status": "error", "error": "db_error", "detail": str(e)}), 500

except Exception as e:
    db.session.rollback()
    logger.exception("Unexpected error in delete_url")
    return jsonify({"status": "error", "error": "internal_error", "detail": str(e)}), 500

@app.route("/health", methods=["GET"])
def health_check():
    return jsonify({"status": "ok"})

if __name__ == "__main__":
    host = os.getenv("HISTORY_SERVICE_HOST", "0.0.0.0")

```

```
port = int(os.getenv("HISTORY_SERVICE_PORT", 8005))
logger.info(f"Starting History Service on {host}:{port}")
with app.app_context():
    db.create_all()
app.run(host=host, port=port)
```