Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена Ольховська
(підпис)
«___» _____ 2024 р.

К В А Л І Ф І К А Ц І Й Н А Р О Б О Т А

НА ТЕМУ:

РОЗРОБКА ЗАСТОСУНКУ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ІДЕНТИФІКАЦІЇ ШКІДЛИВИХ URL-АДРЕС

Зі спеціальності 122 «Комп’ютерні науки»
Освітня програма «Комп’ютерні науки»
Ступеня магістра

Виконавець роботи Вовк Олексій Васильович

_____«___»_____2024р.
(підпис)

Науковий Керівник к.ф.-м.н., Олексійчук Юрій Федорович

_____«___»_____2024р.
(підпис)

Рецензент _____

ПОЛТАВА 2024

АНОТАЦІЯ

Записка: 49 стр., 15 рис., 1 додаток, 15 джерел.

ШТУЧНИЙ ІНТЕЛЕКТ, КІБЕРАТАКИ, ВЕБ-АДРЕСИ.

Об’єкт розробки – аналіз URL-адрес у реальному часі.

Предмет розробки – програма на основі технологій штучного інтелекту, для виявлення шкідливих URL-адрес.

Мета роботи – поліпшення безпеки користувачів, покращення рівня захисту від фішингових атак, завантажень зловмисного програмного забезпечення та витоку даних зі зловмисних веб-сайтів.

Методи дослідження – редактор коду Visual Studio Code, алгоритми машинного навчання, об’єктно-орієнтована мова програмування Python і її бібліотеки.

Результати дослідження. Зроблено огляд відомих сервісів для розпізнавання URL-адрес, розглянуто позитивні та негативні сторони. Розроблено алгоритм програми для виявлення шкідливих URL-адрес, побудовано її блок схему, реалізовано програмне забезпечення.

Здійснено програмну реалізацію застосунку на мові Python, використовуючи бібліотеки NumPy, Pandas, Scikit-learn, FastAPI для ефективного навчання та розгортання нейронної мережі, як у хмарних, так і на локальних серверах.

У результаті тестування виявлено, що розпізнавання в режимі реального часу було можливим із мінімальною затримкою, забезпечуючи своєчасний захист для користувачів, а точність програми склала 98%, що відповідає галузевим тестам.

Рекомендації щодо використання результатів дослідження. Результати роботи впроваджено як прикладний програмний інтерфейс, який може бути використаний сторонніми застосунками, надаючи попередження стосовно безпеки URL-адрес у реальному часі для тисяч клієнтів у всьому світі, а також у вигляді сторінки з лаконічним дизайном для користувачів, які можуть використати її задля перевірки власних посилань.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
AI	Штучний інтелект. Широке поле для створення комп'ютерів, що імітують людський інтелект. Це може включати такі речі, як розуміння мови, розпізнавання об'єктів або навчання на власному досвіді.
NLP	Обробка природної мови. Загальний напрям інформатики, штучного інтелекту та математичної лінгвістики. Він вивчає проблеми комп'ютерного аналізу та синтезу природної мови. Стосовно штучного інтелекту аналіз означає розуміння мови, а синтез — генерацію розумного тексту. Розв'язок цих проблем буде означати створення зручнішої форми взаємодії комп'ютера та людини.
Вразливість нульового дня	Вразливість програмного забезпечення, яка ще невідома користувачам чи розробникам програмного забезпечення та проти якої ще не розроблені механізми захисту
CLI	(Command line interface) Інтерфейс командного рядка. Різновид текстового інтерфейсу користувача й комп'ютера, в якому комп'ютеру можна дати інструкції тільки введенням текстових рядків.[20]

GUI	(Graphical User Interface) Графічний інтерфейс користувача – тип інтерфейсу, який дає змогу користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації.[21]
API	(Application programming interface) Інтерфейс програмування застосунків – спосіб завдяки якому дві або більше комп’ютерні програми можуть спілкуватися між собою. Набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.[19]

ВСТУП

Актуальність: У сучасну епоху поширення шкідливих URL-адрес становить значну загрозу як для приватних осіб, так і для підприємств та установ. Оскільки кібератаки стають все більш витонченими і поширеними, існує нагальна потреба в активних заходах для ефективного виявлення та блокування цих загроз.[18] Розробка програми, що використовує технології штучного інтелекту для виявлення шкідливих URL-адрес, задовольняє цю критичну потребу, пропонуючи передове рішення, яке може швидко аналізувати величезні обсяги даних і виявляти шкідливі посилання з високою точністю. Ця актуальність підкреслює важливість використання досягнень штучного інтелекту для посилення захисту онлайн-середовища, тим самим покращуючи безпеку і довіру інтернет-користувачів у всьому світі.

Метою роботи є поліпшення безпеки користувачів. Покращення рівня захисту від фішингових атак, завантажень зловмисного програмного забезпечення та витоку даних зі зловмисних веб-сайтів.

Об’єктом розробки в цій роботі є аналіз URL-адрес у реальному часі.

Предметом розробки є програма на основі технологій штучного інтелекту, для виявлення шкідливих URL-адрес.

Головне завдання – створити програмне забезпечення, яке покращить безпеку, та допоможе користувачам виявляти шкідливі URL-адреси.

Перелік **використаних методів** полягає у застосуванні технологій штучного інтелекту, а саме моделей керованого навчання, які можуть ефективно виявляти та класифікувати шкідливі URL-адреси.

У ході реалізації програми використано передові технології. Проведені ретельні процедури тестування для оцінки продуктивності та точності додатку. Це включало методи перехресної перевірки, де набір даних був розділений на навчальні та тестові набори, щоб оцінити продуктивність моделі в різних сценаріях. Завдяки ітеративному вдосконаленню та оптимізації програма досягла високих рівнів

точності та запам'ятовування, мінімізуючи помилкові спрацьовування та хибно негативні результати при виявленні шкідливих URL-адрес.

Програма готова до використання та пропонує користувачам надійне рішення для виявлення шкідливих URL-адрес і зменшення пов'язаних із цим ризиків. Завдяки комплексному набору алгоритмів і методологій на основі AI програма забезпечує безпрецедентну точність і ефективність у виявленні шкідливих посилань на різних інтерфейсах.

Робота складається з вступу, постановки завдання, інформаційного огляду, теоретичної та практичної частин і висновку. У першому розділі позначене головне завдання роботи, а також основні кроки для його виконання. У другому розділі розглянуті переваги та недоліки використання технологій AI в комп’ютерній безпеці, проведено огляд та аналіз існуючих сервісів для виявлення шкідливих URL-адрес, проведено огляд концепцій машинного та глибокого навчання, а також алгоритмів AI для класифікації. У третьому розділі розглянуто мову Python, як основний інструмент, завдяки якому буде побудовано програму, створено блок-схему додатку та описано алгоритм роботи програми. В четвертому розділі описано програмну реалізацію елементів алгоритму.

Обсяг пояснювальної записки: 49 стор., в т.ч. основна частина 47 стор., джерел - 15

РОЗДІЛ 1. ПОСТАНОВКА ЗАВДАННЯ

Першочерговим завданням кваліфікаційної роботи – є розробка програмного забезпечення для ідентифікації шкідливих URL-адрес використовуючи технології штучного інтелекту. Програма повинна приймати URL-адреси, перетворювати їх у матриці і при тренуванні використовувати отримані дані для навчання моделі. По закінченню, модель буде збережена для подальшого використання, що буде являти собою ті самі дії, як і при тренуванні, лише вкінці замість функції навчання буде використовуватися алгоритм передбачення, який видаватиме результат у вигляді булевих значень, а додаток перетворюватиме їх у зручні, лаконічні висновки для користувачів.

Завдання кваліфікаційної роботи:

- розглянути переваги та недоліки використання штучного інтелекту в інформаційній безпеці, концепції машинного та глибокого навчання а також алгоритми штучного інтелекту для класифікацій;
- провести огляд та аналіз існуючих сервісів для виявлення шкідливих URL-адрес;
- розглянути та дослідити мову програмування Python, як інструмент для створення додатків на основі технологій штучного інтелекту;
- створити блок-схему роботи програмного забезпечення;
- написати алгоритм роботи програми;
- написати програму для тренування і використання моделі штучного інтелекту;
- описати результати програмування елементів застосунку;
- тестування елементів застосунку.

Технічне завдання являє собою фундамент для подальшої роботи. В ньому прописуються всі деталі: структура програми, її дизайн, функціонал, а також можливості зміни програми.

Програмування − це процес написання програмного коду, використовуючи довільну мову програмування.

Макет дизайну програми – це візуальне уявлення структури та зовнішнього оформлення майбутнього проекту. Він може складатися з фото, навігаційних панелей, кнопок, іконок, додаткових вікон і таке інше.

Тестування – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому його мають використовувати.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Переваги та недоліки використання технологій штучного інтелекту в комп’ютерній безпеці

Комп’ютерні загрози, що змінюються і еволюціонують кожен день, вимагають не менш досконалих систем захисту. AI стає потужним інструментом в арсеналі інформаційної безпеки, пропонуючи значні переваги у виявленні, реагуванні та запобіганні атакам. Однак, як і будь-яка інша потужна технологія, AI має свій набір викликів, які потребують ретельних досліджень.

Переваги AI

Покращене виявлення загроз та реагування на них:

- аналіз великих даних: AI відмінно справляється з обробкою величезних обсягів даних мережевого трафіку і журналів активності користувачів. Це дозволяє виявляти тонкі аномалії, які можуть вислизнути від аналітиків і дає можливість виявляти зловмисну поведінку значно раніше;
- розпізнавання шаблонів: алгоритми AI можуть навчатися на історичних даних, щоб розпізнавати шаблони, пов'язані з відомими кібератаками. Це дозволяє їм виявляти зловмисні дії пов’язані з новими технологіями а також так звані “вразливості нульового дня”;
- швидше реагування: системи безпеки зі AI можуть реагувати на загрози в режимі реального часу, автоматично запускаючи контрзаходи для ізоляції та локалізації порушень до того, як вони завдадуть значної шкоди.

Покращене управління вразливостями:

- автоматизоване сканування вразливостей: AI може автоматизувати процес сканування систем на наявність вразливостей, надаючи можливість персоналу служби безпеки зосередитись на більш масштабних завданнях;
- прогнозоване обслуговування: алгоритми AI можуть аналізувати дані системи, щоб передбачити потенційні вразливості ще до того, як вони

будуть використані. Такий проактивний підхід дозволяє вживати превентивних заходів, мінімізуючи центри, що можуть бути уражені.

Підвищення ефективності та масштабованості:

- зменшення втоми від тривог: команди безпеки часто перевантажені постійним потоком сповіщень про загрози. AI може відфільтрувати хибні спрацьовування і визначити пріоритетність справжніх загроз, зменшуючи навантаження і дозволяючи аналітикам зосередитися на найбільш важливих питаннях;
- масштабованість для задоволення зростаючих потреб: рішення для безпеки на основі AI можна легко масштабувати відповідно до зростаючих мереж і обсягів даних, забезпечуючи надійний захист в процесі розширення організації.

Аналітика поведінки користувачів:

- виявлення внутрішніх загроз: AI може аналізувати моделі поведінки користувачів, щоб виявити підозрілі дії, які можуть свідчити про зловмисні наміри з боку персоналу;
- адаптивні заходи безпеки: AI може вчитися на поведінці користувачів і відповідно адаптувати протоколи безпеки, тим самим створюючи більш персоналізований і ефективний захист.

Безперервне навчання та адаптація:

- ландшафт загроз, що еволюціонує: Алгоритми AI можуть постійно навчатися на нових даних і адаптуватися до сучасних загроз. Це гарантує, що механізми безпеки випереджають зловмисників, які постійно розробляють та вдосконалюють технології атак;
- покращена розвідка загроз: AI може аналізувати дані з різних джерел для збору інформації про загрози, надаючи цінні відомості про тактику і мотивацію зловмисників.

Недоліки AI

Хибно позитивні та хибно негативні результати:

- надмірна залежність від історичних даних: системи AI навчені на історичних даних можуть не впоратися з новими методами атак, що призводить до відсутності реакції на загрозу і як результат її ігноруванню;
- втома від попереджень: хоча AI може зменшити кількість хибних спрацьовувань, погано налаштовані системи все одно можуть генерувати велику кількість сповіщень низького рівня, що заважатиме командам безпеки.

Розуміння і прозорість:

- проблема "чорної скриньки": складні процеси прийняття рішень у системах AI можуть бути непрозорими, що ускладнює розуміння того, як вони приходять до певних висновків. Ця відсутність ясності може перешкоджати довірі та підзвітності;
- розуміння логіки системи: ідентифікація причин, які лежать в основі прийняття рішень програм зі AI може бути складним завдання, що потребує значну кількість часу, а також залучення висококваліфікованого персоналу.

Вразливість до атак:

- зловмисні атаки: злочинці можуть використовувати слабкі місця в алгоритмах AI, щоб маніпулювати ними, генеруючи хибні спрацьовування або компрометуючи заходи безпеки;
- «отруєння даних»: наповнення систем AI даними, які були змінені злочинцями, може призвести до упереджених або неточних результатів, що потенційно послаблює загальну безпеку.

Етичні питання:

- питання конфіденційності: безпекові рішення на основі AI можуть викликати занепокоєння щодо конфіденційності користувачів, особливо коли йдеться про моніторинг їхньої поведінки та збір персональних даних;
- використання AI на озброєнні: потенціал кібератак з використанням AI може збільшити серйозність і частоту виникнення комп’ютерних загроз.

Витрати та проблеми впровадження:

- інвестиції в ресурси: розробка та розгортання систем безпеки на основі AI вимагає значних інвестицій в інфраструктуру, експертизу та постійне обслуговування;
- інтеграція з існуючими системами: інтеграція рішень зі AI з наявною інфраструктурою безпеки може бути складним і трудомістким процесом.

Підводячи підсумок можна сказати, що AI пропонує потужний набір інструментів для посилення інформаційної безпеки. Однак дуже важливо усвідомлювати обмеження та потенційні ризики, пов'язані з цією технологією. Ретельно зваживши як переваги, так і недоліки, організації можуть використовувати AI для створення багаторівневої та адаптивної системи безпеки, яка ефективно захищатиме їхні цифрові активи в умовах мінливого світу інформаційних загроз.

2.2 Огляд та аналіз існуючих сервісів для виявлення шкідливих веб-адрес

У сучасному світі, коли технології розвиваються і еволюціонують з неймовірною швидкість, а нові сервіси з’являються щодня, проблема безпеки активів компаній та їх користувачів є нагальною як ніколи, адже з розвитком програм з’являються нові вразливості, а атаки вдосконалюються і комбінуються. Одним з найпоширеніших методів сьогодення, який зловмисники використовують для вчинення злочинів в Інтернеті є використання шкідливих URL-адрес (Uniform Resource Locator), на які припадає близько 60% комп’ютерних атак. Шкідлива URL-адреса – веб-адреса, яка була створена або підроблена зі зловмисними намірами. Ці посилання зазвичай ведуть користувачів на веб-сайти або веб-сторінки, які містять зловмисне програмне забезпечення, фішингові афери, шахрайський контент. Шкідливі URL-адреси небезпечні тим, що вони можуть бути використані як вектори для багатьох атак, входячи в сценарії та складні схеми злочинів. Надалі будуть розглянуті сервіси, які допомагають ідентифікувати даний тип загроз.[1]

Norton Safe Web

Norton Safe Web – комплексна служба веб-безпеки від NortonLifeLock, призначена для захисту користувачів від таких загроз в Інтернеті, як шкідливе програмне забезпечення, фішингові шахрайства та зловмисні веб-сайти. Вона надає

користувачам рейтинги безпеки веб-сайтів, допомагаючи їм приймати обґрунтовані рішення щодо своєї діяльності в Інтернеті.

Позитивні сторони системи:

- комплексний захист від широкого спектру інтернет-загроз;
- зручний та інтуїтивно зрозумілий і простий у використанні інтерфейс;
- захист у режимі реального часі за допомогою розширень для браузерів;
- регулярні оновлення бази даних.

Негативні сторони:

- може вимагати значних ресурсів, особливо під час повного сканування;
- потрібна передплата для використання повного набору функцій;
- хибні спрацьовування.[22]

Підсумовуючи, Norton Safe Web - це надійний сервіс веб-безпеки, який надає користувачам комплексний захист від онлайн-загроз. Використання передових технологій, функції захисту в режимі реального часу та зручний інтерфейс роблять його цінним інструментом для захисту від злочинного програмного забезпечення, фішингових атак та шкідливих веб-сайтів. Хоча він може мати деякі недоліки, такі як значне споживання ресурсів системи і можливість хибних спрацьовувань, його загальна ефективність у захисті безпеки користувачів в Інтернеті робить його гідною інвестицією для тих, хто прагне спокою під час перегляду веб-сторінок.

Інструменти для встановлення репутації доменів WhoisXML API

WhoisXML API пропонує комплексний набір послуг, призначених для оцінки репутації та стану безпеки доменних імен в Інтернеті. Ці інструменти надають цінну інформацію про історичну поведінку, потенційні ризики та загальну репутацію доменів, дозволяючи користувачам приймати обґрунтовані рішення щодо діяльності в Інтернеті та заходів безпеки.

Позитивні сторони:

- комплексний аналіз доменних імен;
- моніторинг у режимі реального часу доменних імен за чорними списками;
- інтеграція з API;
- сповіщення, які можна налаштовувати.

Негативні сторони:

- доступ до розширених функцій і даних вимагає підписки або оплати;
- хибні спрацьовування.[23]

Підсумовуючи, Інструменти репутації доменів WhoisXML API – потужні ресурси для оцінки репутації та стану безпеки доменних імен в Інтернеті. Завдяки своїм комплексним можливостям аналізу, функціям моніторингу в режимі реального часу та технологіям інтеграції з API, ці інструменти надають цінну інформацію для фахівців з комп'ютерної безпеки, інтернет-дослідників та звичайних користувачів. Хоча вони можуть мати певні обмеження, такі як вартість і ймовірність помилкових спрацьовувань, їхня загальна ефективність у виявленні та пом'якшенні загроз, пов'язаних з доменами, робить їх цінним ресурсом для посилення безпеки в Інтернеті.

URLVoid

URLVoid – веб-сервіс, який пропонує аналіз репутації та безпеки веб-сайтів. Він надає користувачам інформацію про репутацію певної URL-адреси, скануючи її за кількома базами даних відомих загроз, включаючи шкідливе програмне забезпечення, фішинг і спам. URLVoid об'єднує дані з різних джерел для отримання оцінки кожної URL-адреси, допомагаючи користувачам приймати обґрунтовані рішення щодо відвідування або взаємодії з веб-сайтами.

Позитивні сторони:

- комплексний аналіз репутації та безпеки веб-сайтів;
- зручний інтерфейс;
- результати в режимі реального часу;
- безкоштовне використання.

Негативні сторони:

- обмежене покриття та неповне охоплення;
- може іноді генерувати помилкові спрацьовування;
- залежність від зовнішніх джерел інформації таких як чорні списки.[17]

Підсумовуючи, URLVoid – корисний інструмент для оцінки репутації та безпеки веб-сайтів, який пропонує користувачам комплексний аналіз і результати в

режимі реального часу. Зручний інтерфейс і безкоштовність роблять його доступним для широкого кола користувачів, від приватних осіб до організацій. Однак користувачі повинні знати про його обмеження, включаючи потенційні помилкові спрацьовування і залежність від зовнішніх джерел, і розглянути можливість використання URLVoid в поєднанні з іншими заходами безпеки для підвищення своєї безпеки в Інтернеті. В цілому, URLVoid надає цінну послугу для користувачів, які прагнуть захистити себе від потенційних загроз, пов'язаних зі шкідливими веб-сайтами.

VirusTotal

VirusTotal – це онлайн-сервіс, який забезпечує сканування та аналіз файлів і URL-адрес на наявність шкідливого програмного забезпечення. Він об'єднує дані з декількох антивірусних систем та інших інструментів безпеки для виявлення вірусів, комп’ютерних черв'яків, троянських програм та інших типів шкідливого програмного забезпечення. Користувачі можуть надсилати файли або URL-адреси до VirusTotal, який потім сканує їх і надає детальний звіт про потенційні загрози.

Позитивні сторони:

- комплексне виявлення шкідливих програм;
- аналіз в режимі реального часу;
- зручний інтерфейс;
- можливості інтеграції завдяки API.

Негативні сторони:

- хибні спрацьовування;
- обмежені можливості аналізу;
- занепокоєння щодо конфіденційності [8]

Підсумовуючи, VirusTotal – потужний і цінний інструмент для сканування та аналізу шкідливого програмного забезпечення, що пропонує користувачам комплексні можливості виявлення та аналізу потенційних загроз в режимі реального часу. Зручний інтерфейс, можливості інтеграції та широке охоплення антивірусних сервісів роблять його популярним вибором серед звичайних користувачів, фахівців з інформаційної безпеки та організацій, які прагнуть посилити свій рівень безпеки.

Однак при використанні VirusTotal слід пам'ятати про потенційні хибні спрацьовування, обмеження в можливостях аналізу та міркування конфіденційності. В цілому, VirusTotal є ефективним рішенням для швидкої оцінки безпеки файлів і URL-адрес та зменшення потенційних ризиків безпеки.

2.3 Концепції машинного та глибокого навчання

В епоху надлишку даних і обчислювальних потужностей машинне навчання стало трансформаційною технологією, що має широке застосування в різних галузях. Його алгоритми, в основі яких лежить ідея надання комп'ютерам можливості навчатися на основі даних без явного програмування, стали фундаментом сучасного технологічного прогресу. У сфері машинного навчання існує підрозділ, відомий як глибоке навчання, що характеризується здатністю автоматично вивчати репрезентації з даних за допомогою штучних нейронних мереж. Надалі будуть розглянуті ключові концепції машинного та глибинного навчання.

Основи машинного навчання

Машинне навчання – галузь штучного інтелекту, яка фокусується на розробці алгоритмів, здатних тренуватися на основі даних, щоб робити прогнози або приймати рішення. В основі машинного навчання лежить концепція навчання на основі досвіду, коли алгоритми з часом покращують свою роботу, отримуючи більше даних. Існує три основні типи алгоритмів машинного навчання: контрольоване навчання, неконтрольоване навчання та навчання з підкріпленням.

Контрольоване навчання представляє собою тренування моделі на маркованому наборі даних, де кожен елемент даних пов’язаний з відповідною ціллю або вихідною інформацією. Мета полягає в тому, щоб навчитися зіставляти вхідні елементи з вихідними мітками, дозволяючи моделі робити прогнози стосовно інформації, яку модель ще не бачила. Алгоритми керованого навчання включають лінійну регресію, дерева рішень і машини опорних векторів.

Неконтрольоване навчання має справу з немаркованими даними і має на меті виявити приховані шаблони або структури в них. Алгоритми кластеризації, такі як

к-середні та ієрархічна кластеризація, зазвичай використовуються в задачах неконтрольованого навчання для групування схожих точок даних на основі їхніх особливостей.

Навчання з підкріпленням фокусується на тренуванні моделей взаємодіяти з середовищем, щоб максимізувати сукупний сигнал винагороди. Шляхом спроб і помилок модель вчиться виконувати дії, які призводять до позитивних результатів, уникаючи небажаних. Навчання з підкріпленням знайшло застосування в різних сферах, включаючи робототехніку, ігри та автономне керування транспортними засобами.

Розуміння глибокого навчання

Глибоке навчання – підрозділ машинного навчання, який використовує штучні нейронні мережі з декількома шарами для автоматичного тренування на ієрархічно представлених даних. В основі цієї системи лежать штучні нейронні мережі, які є обчислювальними моделями, натхненними структурою і функціями біологічних нейронних мереж людського мозку.[3]

Штучна нейронна мережа складається з взаємопов'язаних вузлів, організованих у шари. Вхідний шар отримує необроблені вхідні дані, які потім обробляються через один або кілька прихованих шарів, що містять взаємопов'язані вузли, які називаються нейронами. Кожен нейрон застосовує перетворення до вхідних даних, використовуючи зважену суму своїх входів, після чого застосовується активаційна функція, яка вносить нелінійність у мережу.

Вихідний шар видає остаточні прогнози мережі на основі обробленої інформації з прихованих шарів. Під час процесу навчання мережа коригує ваги своїх зв'язків за допомогою процесу, відомого як зворотне поширення, де похибка між прогнозованими виходами та істинними мітками мінімізується за допомогою алгоритмів оптимізації, таких як градієнтний спуск.

2.4 Алгоритми штучного інтелекту для класифікації

Алгоритми AI для класифікації є фундаментальними інструментами машинного навчання, що дозволяють комп'ютерам автоматично розподіляти дані за

попередньо визначеними класами або категоріями. Класифікація відіграє ключову роль у різноманітних додатках, включаючи розпізнавання зображень, виявлення спаму, медичну діагностику та аналіз настроїв. Надалі будуть розглянуті популярні алгоритми класифікації AI, їхні основні принципи роботи, сильні і слабкі сторони, а також сфера застосування.

Наївний класифікатор Байєса (Naive Bayes Classifier)

Наївний класифікатор Байєса заснований на теоремі Байєса з припущенням про незалежність ознак. Він простий, швидкий та ефективний для задач класифікації тексту, таких як фільтрація спаму та аналіз настроїв. Однак він, як правило, погано працює, коли ознаки не є незалежними або коли має справу з безперервними даними.

Дерева рішень (Decision Trees)

Дерева рішень рекурсивно розбивають простір ознак на регіони, приймаючи рішення на основі значень ознак. Вони інтуїтивно зрозумілі, легко інтерпретуються і здатні обробляти як числові, так і категоріальні дані. Однак дерева рішень схильні до перенавчання, особливо у випадку складних наборів даних.[6]

к-найближчих сусідів (k-NN)

Алгоритм к-найближчих сусідів класифікує точки даних на основі більшості голосів їхніх найближчих сусідів у просторі ознак. Він є непараметричним і простим у реалізації, але страждає від прокляття розмірності і може потребувати більше обчислювальних ресурсів, особливо при роботі з великими наборами даних.[2]

Метод опорних векторів (SVM)

SVM будує гіперплощину, яка розділяє класи в просторі ознак з максимальним відривом. Він ефективний у просторах високої розмірності і стійкий до перенавчання завдяки принципу оптимізації поля. Однак SVM може бути чутливим до вибору функції ядра.[14]

Логістична регресія

Логістична регресія моделює ймовірність бінарних результатів за допомогою логістичної функції. Вона широко використовується для задач бінарної класифікації, таких як медична діагностика та аналіз кредитних ризиків. Логістична регресія дає

результати, які можна легко інтерпретувати, і є менш схильною до перенавчання порівняно з більш складними моделями.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Обґрунтування вибору мови програмування

При виборі відповідної мови програмування для розробки додатку, спрямованого на виявлення шкідливих URL-адрес з використанням технологій штучного інтелекту, слід враховувати такі фактори як легкість в створенні прототипів, рівень простоти розуміння коду, підтримка та доступність додаткових модулів, інтеграція з сторонніми сервісами, активність спільноти розробників та інше. Серед доступних варіантів, таких як Python, R та Julia, Python виявляється найбільш підходящим вибором з наступних причин.

Велика кількість бібліотек та фреймворків.

Python може похвалитися багатою екосистемою бібліотек і фреймворків, спеціально розроблених для задач AI та машинного навчання. Такі бібліотеки, як TensorFlow, PyTorch та scikit-learn пропонують комплексну підтримку для реалізації різних алгоритмів AI, включаючи моделі глибокого навчання для задач обробки природної мови. Ці бібліотеки надають готові модулі для таких завдань, як вилучення ознак, навчання моделей та оцінювання, що значно прискорює процес розробки. Крім того, менеджер пакетів Python, pip, спрощує встановлення та управління сторонніх модулів, забезпечуючи легкий доступ до найновіших інструментів та досягнень AI.[24]

Крім того, про широку популярність Python у сфері AI свідчить його використання в численних гучних проектах у різних галузях. Наприклад, GPT-моделі OpenAI, включаючи GPT-3 та GPT-4 побудовані за допомогою Python і TensorFlow, що демонструє можливості мови для підтримки передових досліджень і розробок у даній галузі. Широке використання даної мови у проектах такого масштабу підкреслює її універсальність та ефективність у вирішенні складних завдань AI.

Легкість у вивченні та використанні.

Мова Python відома своєю простотою та зрозумілістю, що робить її ідеальним вибором для розробників усіх рівнів кваліфікації, включно з початківцями. Її інтуїтивно зрозумілий синтаксис дозволяє швидко створювати прототипи та експериментувати, що є важливим для ітеративних процесів розробки, поширених у проектах зі створення продуктів AI. Крім того, обширна документація та онлайн-ресурси Python, включаючи підручники, посібники та форуми спільноти, полегшують самостійне навчання та вирішення проблем. Велика кількість навчальних матеріалів і курсів ще більше підвищує доступність, дозволяючи людям швидко набути навичок, необхідних для розробки AI.

Поміж іншого, зрозумілість і виразність мови Python сприяють її широкому поширенню серед аналітиків даних, дослідників і практиків у даній галузі. Простий синтаксис мови дозволяє розробникам зосередитися на вирішенні складних завдань AI замість того, щоб боротися з важкими структурами коду, стимулюючи творчість та інновації в розробці AI.

Отже, вибір Python як мови програмування для розробки програми для виявлення шкідливих URL-адрес за допомогою технологій AI є цілком виправданим. Великі бібліотеки, простота вивчення, потужна підтримка спільноти, сумісність з фреймворками AI та масштабованість роблять її ідеальною мовою для реалізації складних алгоритмів AI та створення надійних, ефективних рішень у сфері інформаційної безпеки. Використовуючи сильні сторони Python, розробники можуть створювати інноваційні та зручні підходи на основі AI для зменшення загроз та підвищення безпеки в Інтернеті для користувачів по всьому світу. Крім того, роль даної мови у розвитку досліджень у галузі AI та стимулюванні технологічних інновацій підкреслює його значення як інструменту для розробки AI в сучасну епоху.

3.2 Блок-схема програми

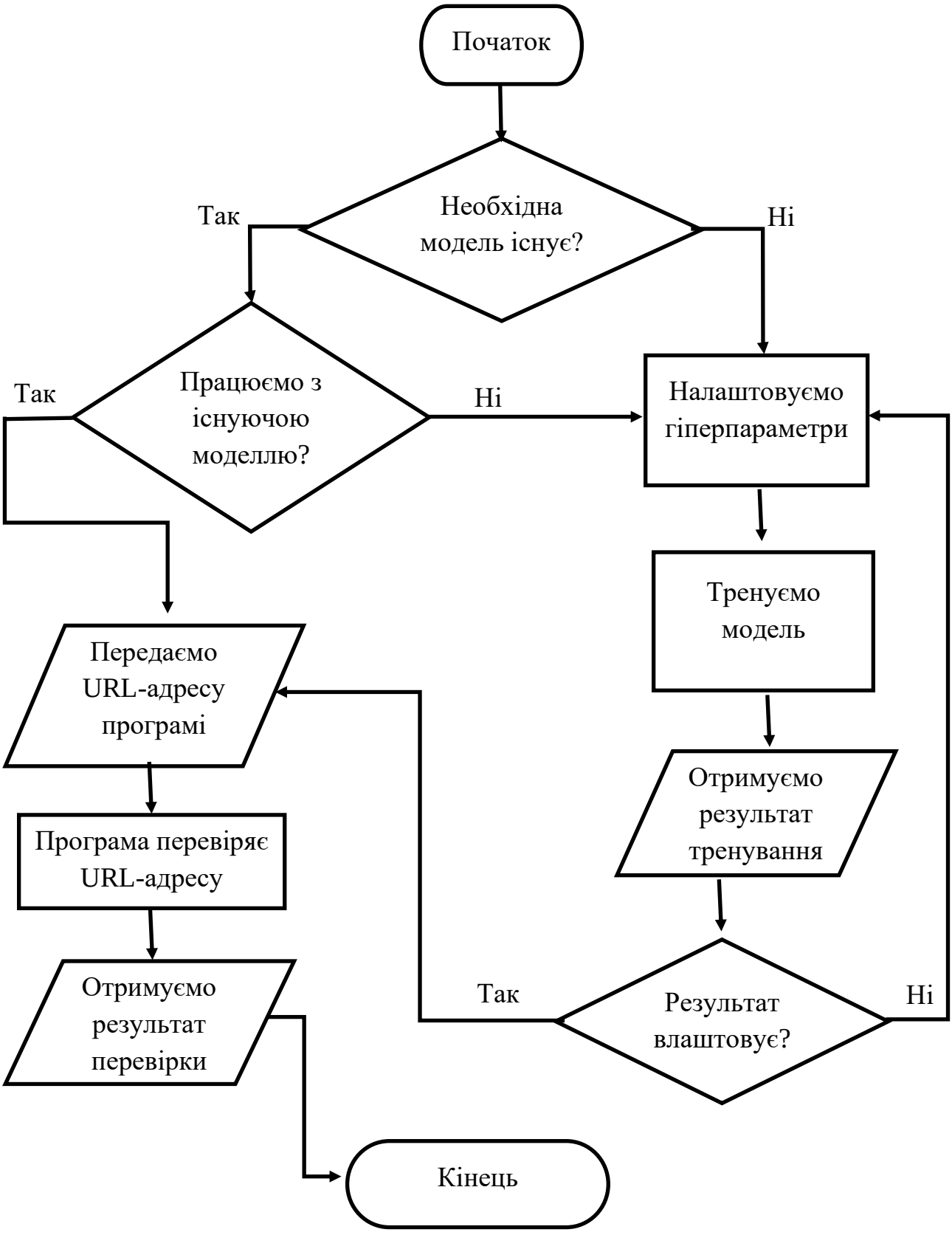

Рисунок 3.1 – Блок-схема алгоритму роботи програми

3.3 Алгоритм роботи програми по виявленню шкідливих URL-адрес

Даний застосунок створений на мові програмування Python, тому для його роботи потрібно, щоб інтерпретатор цієї мови був встановлений на пристрій, на якому програма буде запущена. Існує багато версій Python як і різних систем, на деяких він встановлений, інші його не мають, або ж доступна за замовчуванням версія занадто стара. Даний застосунок був написаний на версії 3.10 і лише на ній є гарантія, що він буде працювати, використання інших може потребувати додаткових налаштувань і змін в коді, тому рекомендується використовувати саме версію 3.10. Що стосується систем, кожна з них має свій спосіб встановлення і оновлення мови, тому найкращім варіантом буде звернутися до офіційного сайту Python за необхідними інструкціям.

Створене програмне забезпечення має два способи використання: тренування моделі та перевірку веб-адрес. Тренування ділиться на навчання нової й тренування моделі, що існує. Перевірка веб-адрес має три режими роботи: інтерфейс командного рядка – для людей, які бажають запустити програму у себе на комп'ютері, зручна форма на веб-сайті – для користувачів, яким комфортніше працювати в браузері, та прикладний програмний інтерфейс – для взаємодії з іншими додатками. Надалі кожен з варіантів буде розглянутий більше докладніше.

Згідно з найкращими практиками, деяку інформацію не слід зберігати в основних файлах програми, наприклад облікові дані користувачів та налаштування, які можуть часто змінюватися, тому для цієї інформації використано змінні оточення.[5] Щоб спростити і зробити взаємодію з додатком більш приємнішим, було використано спеціальні конфігураційні файли замість додавання необхідної інформації в змінні оточення системи.

Перш ніж користувач зможе почати використовувати програму, йому потрібно в папці з додатком створити файл з назвою *.env*, в якому будуть вказані всі необхідні налаштування. Якщо людина бажає лише сканувати URL-адреси, їй треба додати наступні записи в *.env* файл і зберегти його:

NGRAM_PATH="data/ngrams.json"

MODEL_PATH="data/sgd_model.pkl"

Наведені змінні вказують шлях до файлів з комбінаціями N-грама і натренованої моделі. Дані приклади є значеннями за замовчуванням, і можуть бути змінені за бажанням.

Далі потрібно створити віртуальне оточення, активувати його і встановити усі необхідні бібліотеки, які використовує додаток, для цього треба відкрити термінал і перейти в ньому до директорії, в якій знаходиться файл *requirements.txt* і в ній виконати наступні команди:

python3 -m venv env

. env/bin/activate

python -m venv env

. env\Scripts\activate

pip install -r requirements.txt

Перші дві команди треба використовувати на Unix системах, якщо додаток буде працювати на Windows, треба використовувати третю і четверту команди. Остання директива однакова для всіх.

Інтерфейс командного рядка

Після зроблених кроків, користувач може дізнатися про можливості додатка у інтерфейсі командного рядка за допомогою команди:

python cli.py -h

Ця команда виведе на екран допоміжну інформацію з усіма можливими варіантами взаємодії користувача з інтерфейсом командного рядка.

Щоб перевірити веб-адресу, потрібно виконати команду:

python cli.py -l <будь-яка URL-адреса>

Цей виклик проаналізує команду і виведе лаконічну відповідь стосовно статусу ресурсу.

Щоб дізнатися точність моделі, потрібно додати наступні записи в файл .env:

ROWS_NUMBER=500

DATASET_PATH = "data/malicious_url_dataset.csv"

Ці змінні вказують на кількість записів, які будуть оброблені в одній партії і шлях до файлу з набором даних для навчання і тестів. Якщо наявні достатні

обчислювальні потужності, значення кількості записів може бути збільшене, навантаження зросте, але зменшиться час перевірки і тренування. Після оновлення файлу *.env* потрібно запустити наступну команду:

python cli.py -c

У терміналі з’явиться інформація про кількість правильних і неправильних передбачень, а також точність моделі у відсотках.

Для тренування моделі потрібні всі чотири змінні в файлі .env які були зазначені вище. Для того, щоб розпочати процес навчання, необхідно виконати наступну команду:

python cli.py -t

Після закінчення тренування, з’явиться відповідне повідомлення, а також інформація про точність моделі та кількість правильних і не правильних передбачень, сама ж навчена модель буде збережене за замовчуванням у директорію *data* у файл *sgd_model.pkl*, за бажанням ці специфікації можуть бути змінені через редагування відповідної змінної у файлі *.env*.

Якщо в наявності велика кількість нових даних, існуючу модель можливо оновити, для цього необхідно створити файл у форматі *csv* з будь-якою назвою і в нього додати нову інформацію у наступному вигляді:

Веб-адреса,статус адреси

Статус адреси може мати лише значення 1 або 0, 1 – адреса шкідлива, 0 – безпечна. Приклад записів можна подивитись у файлі *malicious_url_dataset.csv*, який використовувався для навчання існуючої моделі. Після закінчення минулого кроку, потрібно в файлі *.env* оновити значення змінної *DATASET_PATH*, замінивши на шлях до нового файлу. Після цього необхідно виконати наступну команду:

Python cli.py -u

В залежності від кількості нових даних, оновлення моделі може зайняти деякий час. Після закінчення процесу, з’явиться відповідне повідомлення, а також інформація про точність моделі та кількість правильних і не правильних передбачень. Надалі при перевірці URL-адрес буде використовуватися вже оновлена модель.

Якщо в наявності нові дані, але їх кількість не велика, наприклад кілька сотень, існує два варіанти:

1. Дочекатись поки інформації про веб-адреси не назбирається хоча б тисяч п’ять або десять.
2. Додати нові дані в існуючий файл і провести навчання моделі спочатку.

Веб-додаток

Щоб використовувати програму через веб-інтерфейс, потрібно у *.env* файл до минулих записів додати наступні змінні:

USER_NAME=”any_name”

USER_PASS=”any_password”

SERVICE_HOST=”0.0.0.0”

SERVICE_PORT=8000

WEBSITE_ADDRESS=”http://localhost:8000”

Перші дві змінні задають ім’я користувача і пароль, які необхідно використовувати, щоб отримати доступ до функції перевірки веб-адрес. Третій і четвертий записи вказують адресу і порт через які можна підключитися до сервісу. Після того, як дані були додані, потрібно виконати наступну команду команду:

python web.py

Через невеликий проміжок часу сервер буде запущено, далі відкривши браузер, необхідно перейти по наступній адресі:

http://localhost:8000/

Якщо додаток розгорнутий на віддаленій машині, необхідності в попередніх кроках немає. Все, що потрібно зробити, це отримати облікові дані для автентифікації у застосунку.

На головній сторінці сайту присутня форма, яка приймає URL-адреси і після їх аналізу видає статус у вигляді короткого повідомлення. Набагато нижче форми додані діаграми, які мають на меті проінформувати користувача про те, яких схем, доменних імен та доменів вищого рівня більше всього зустрічалось в даних, на яких була навчена існуюча модель. Також присутня діаграма про кількість шкідливих і

безпечних URL-адрес у наведених даних. Усі графіки інтерактивні, нажимаючи мишкою на категорії в легенді, можна їх приховувати та відкривати.

Згори головної сторінки присутнє URL-посилання, воно відповідає за прикладний програмний інтерфейс. Там можна не тільки дізнатися, як підключити інші програми для взаємодії з даним додатком, а і перевірити кінцеві точки. Щоб провести тестування, потрібно обрати бажаний шлях, відкрити його за допомогою лівої клавіші миші, натиснути на кнопку “Try it out”, після чого за необхідністю заповнити параметри та тіло запиту, вкінці треба натиснути на кнопку “Execute”. При першій спробі з’явиться вікно автентифікації, де потрібно буде вписати значення, які були записані в змінні *USER_NAME* та *USER_PASS*, якщо облікові дані введено правильно, почнеться процес обробки запиту. По закінченню в розділі “Response” з’явиться відповідь сервера. При правильних даних та параметрах, код стану буде 200.

Підводячи підсумки, можна зазначити, що хоча налаштування і запуск програми потребує деяких додаткових кроків, всі вони досить прості і мають на меті розширити можливості взаємодії користувача з додатком, а також полегшити його налаштування.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1 Застосовані технології

Виявлення шкідливих URL-адрес є досить не тривіальною задачею, адже на відміну від вірусів або атак на вразливі додатки, веб-адреси не мають моделей поведінка, так як самі по собі не виконують зловмисних дій і не створюють записів в журналах, натомість вони покладаються на взаємодію з ними пересічних користувачів. Одним із поширених підходів до виявлення зловмисних веб-адрес є евристика, коли характеристики відомих шкідливих URL-адрес аналізуються для виявлення схожих шаблонів у нових посиланнях. Інший метод передбачає використання алгоритмів машинного навчання для класифікації URL-адрес на основі таких ознак, як репутація домену, довжина адреси та наявність підозрілих ключових слів. Однак ці підходи не є безпомилковими та можуть давати хибні спрацьовування або не виявляти складні атаки. Крім того, зловмисники постійно вдосконалюють свої методи, щоб уникнути виявлення, що ускладнює оновлення методів ідентифікації даних загроз. Деякі організації також використовують рішення для фільтрації URL-адрес, які блокують доступ до відомих шкідливих веб-сайтів на основі попередньо визначених чорних списків. Однак таке рішення може ненавмисно закрити доступ до безпечних веб-сайти або не вловити новостворені шкідливі веб-адреси. Тому провідними фахівцями рекомендується багаторівневий підхід, що поєднує різні методи виявлення.[4]

Для створення даної програми було використано наступні технології:

- Вбудовані модулі Python – набір бібліотек, попередньо встановлених разом із інсталяцією Python. Ці модулі забезпечують широкий спектр функцій, від операцій з файлами та даними системи до математичних обчислень і веб-сервісів.
- Scikit-learn (sklearn) – потужна бібліотека машинного навчання, яка надає прості та ефективні інструменти для інтелектуального аналізу даних.

- NumPy – фундаментальний пакет для наукових обчислень. Він забезпечує підтримку великих багатовимірних масивів і матриць, а також набір математичних функцій для ефективної роботи з ними.
- Pandas – потужна бібліотека для роботи з даними та їх аналізу. Вона надає конструкції та функції для роботи зі структурованою інформацією, такою як табличні дані та інформація з часовими мітками.
- NLTK – провідна платформа для створення програм на мові Python, які спеціалізуються на обробці даних мови. Вона надає прості у використанні інтерфейси до понад 50 корпусів і лексичних ресурсів, таких як WordNet, а також набір бібліотек для обробки тексту для класифікації, створення токенів, тегів, семантичного аналізу, обгортки для бібліотек NLP промислового рівня, а також активний форум для обговорень.
- FastAPI – сучасний, швидкий та високопродуктивний веб-фреймворк для створення API за допомогою Python. Він розроблений, щоб бути простим у використанні та забезпечувати автоматичну інтерактивну документацію API (використовуючи технології OpenAPI та JSON Schema).
- AIOHTTP – асинхронний клієнт-серверний фреймворк для Python. Він дозволяє створювати веб-сервери та клієнти, використовуючи методи асинхронного програмування, що надаються модулем asyncio Python. AIOHTTP відомий своєю простотою, гнучкістю та продуктивністю.

4.2 Програмна реалізація моделі AI

Одним з основним завдань було створити алгоритм для тренування моделі на основі ознак, вилучених з URL-адрес. Щоб програма могла працювати з необробленою інформацією, було використано N-грами для інтелектуального аналізу тексту та метод опорних векторів для його класифікації. Моделі N-грам є досить розповсюдженою технологією у попередній обробці тексту, в них використовується звичайний метод слово-в-вектор для перетворення заданого рядка у вектори. В нашому випадку було вирішено застосувати N-грам розміром в три символи, як найбільш оптимальний, тобто було використано три-грам.

Функція перетворення URL-адреси в N-грам:

```
@staticmethod
@logger.catch
def generate_ngrams_from_string(
    sentence: str,
    ngram_size: int = 3
) -> typing.List:
    """
    Summary:
        Generate ngram of specific size from string
    Args:
        sentence (str): sentence from which ngrams will be generated.
        ngram_size (int, optional): Size of the ngrams. Defaults to 3.
    Returns:
        List: Ngrams.
    """
    sentence = sentence.lower()
    sentence = ''.join(e for e in sentence if e.isalnum())
    processed_list = []
    for tup in list(nltk.ngrams(sentence, ngram_size)):
        processed_list.append((''.join(tup)))
    return processed_list
```

Метод опорних векторів показує чудові результати у задачах подібних до нашої завдяки роботі з гіперплощиною, яка ефективно розділяє текст на два класи з мінімальним перекриттям, це підвищує точність класифікації порівняно з альтернативами, такими як наївний баєсів класифікатор і модель “торба слів”. Бібліотека Scikit-learn має класифікатор стохастичного градієнтного спуску (SGDClassifier), який за замовчуванням відповідає машині лінійних опорних векторів, цим самим він повністю задовольняє наші потреби.

Функція тренування, в якій створюється об’єкт класифікатора:

```
@staticmethod
@logger.catch
def train_model(
    loss: str,
    penalty: str,
    max_iter: int,
    update: bool = False,
):
    """
    Summary:
        Full training process.
    Args:
        update (bool, optional): Flag show if model will be trained
        or updated. Defaults to False.
    """
    if update:
        classifier = ModelUtils.load_model()
    else:
        classifier = SGDClassifier(
            loss=loss,
            penalty=penalty,
            max_iter=max_iter
        )
    train_df, test_df = ModelUtils.split_dataframe()
    features_dict = NgramUtils().get_ngram_combinations()
    ModelUtils.insert_data(train_df, classifier, features_dict)
    ModelUtils.save_model(classifier)
    correct, incorrect = ModelUtils.test_model(
        test_df,
        classifier,
```

```
        features_dict
    )
    if update:
        print("Model have been updated!")
    else:
        print("Model have been trained!")
    print("Correct Predictions ", correct)
    print("Incorrect Predictions ", incorrect)
    accuracy = (correct / test_df.shape[0]) * 100
    print(f"Accuracy of the model is: {accuracy:.4g} %")
```

Завдяки вбудованим бібліотекам Python була полегшена генерація усіх можливих три-грам комбінацій, які є необхідними при побудові векторів, а для створення токенів, що були отримані з URL-адрес, задіяно інструменти з бібліотеки NLTK.

Функція створення N-грам комбінацій:

```
@logger.catch
def create_n_gram_combinations(self, ngram_size: int = 3) -> typing.Dict:
    """
    Summary:
        Create all possible ngram combinations from letters and numbers.
    Args:
        ngram_size (int, optional): Size of the ngrams.
        Defaults to 3 - trigram.
    Returns:
        dict: Ngrams. Key - ngram, value - number.
    """
    alphanum = [
        'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
        'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
        '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
```

```
    ]
    permutations = itertools.product(alphanum, repeat=ngram_size)
    featues_dict = {}
    counter = 0
    for perm in permutations:
        featues_dict[(''.join(perm))] = counter
        counter = counter + 1
    return featues_dict
```

Дані, які одержані з відкритих джерел, таких як URLhaus та Kaggle.com, були розділені на два набори, щоб мати можливість не тільки тренувати модель, а і перевірити її точність. Після цього, кожен з наборів був поділений на партії відповідно з розміром, який встановлений в налаштуваннях програми, а кожна з URL-адрес в партії пройшла процес перетворення в токени і потім в вектори, щоб на виході була отримана матриця, де місце кожного елемента відповідає спеціальному токену. Для цієї задачі задіяно інструменти з бібліотек Pandas, NumPy та Scikit-learn.

Функція розділення та оброблення даних:

```
@staticmethod
@logger.catch
def preprocess_dataset(
    dataframe: pd.DataFrame,
    features_dict: typing.Dict,
) -> typing.Generator[np.ndarray, np.ndarray, pd.DataFrame]:
    """
    Summary
        Split dataframe to batches.

    Args:
        dataframe (pd.DataFrame): dataframe.
        classifier (SGDClassifier): Linear classifiers
        features_dict (typing.Dict): Ngrams. Key - ngram, value - number.
```

```
    Returns:
        tuple: matrix, labels vector and batch.
    """
    rows_number = int(settings.ROWS_NUMBER)
    no_of_batches = int(dataframe.shape[0] / rows_number) + 1
    for i in range(0, no_of_batches):
        start = rows_number * i
        if start + rows_number > dataframe.shape[0]:
            batch = dataframe.iloc[start:, :]
        else:
            batch = dataframe.iloc[start:start + rows_number, :]
        batch = batch.reset_index()
        if batch.empty:
            break
        x, y = UrlUtils.preprocess_batch(
            features_dict,
            batch,
            np.zeros([batch.shape[0], len(features_dict)], dtype="int"),
            np.zeros(batch.shape[0], dtype="int")
        )
        yield x, y, batch
```

Отримані матриці передані в спеціальний метод partial_fit, який використовується для тренування моделі і має деякі особливості, зокрема:

- навчання моделі за допомогою невеликих партій даних, що дає змогу застосовувати технології глибокого навчання маючи великі об’єми інформації і відносно низькі обчислювальні потужності.
- можливість оновлення існуючої моделі новими даними без необхідності тренувати модель спочатку.

Функція тренування моделі на оброблених даних:

```
@staticmethod
@logger.catch
def insert_data(
    train_dataframe: pd.DataFrame,
    classifier: SGDClassifier,
    features_dict: typing.Dict
):
    """
    Summary:
        Insert processed data in the model. Train model.
    Args:
        train_dataframe (pd.DataFrame): test dataframe.
        classifier (SGDClassifier): Linear classifiers
        features_dict (typing.Dict): Ngrams. Key - ngram, value - number.
    """
    for x, y, _ in ModelUtils.preprocess_dataset(
        train_dataframe,
        features_dict
    ):
        classifier.partial_fit(
            x,
            y,
            classes=np.unique(y)
        )
```

Після закінчення процесу навчання, натренована модель буде збережена в файл для подальшого використання у ідентифікації шкідливих URL-адрес.

Алгоритм визначення типу веб-адреси майже повністю повторює дії при тренуванні моделі. Передана користувачем URL-адреса проходить процес розбиття на токени, а далі на вектори, використовуючи ті самі бібліотеки, як були задіяні при

навчанні. Далі, отриману матрицю приймає модель і за допомогою методу predict робить передбачення стосовно статусу ресурсу.

Функція тестування URL-адреси:

```
@staticmethod
@logger.catch
def test_url(
    url: str,
    model_path: str = settings.MODEL_PATH,
    console: bool = False
) -> bool:
    """
    Summary:
        Test url.
    Args:
        url (str): url
        model_path (str, optional): path to trained model.
        console (bool, optional): print result to console.
        Defaults to False.
    Returns:
        bool: True if url is malicious, False if url is safe.
    """
    statuses = {
        True: "URL is Malicious", False: "URL is Safe"
    }
    classifier = ModelUtils.load_model(model_path)
    features_dict = NgramUtils().get_ngram_combinations()
    url_matrix = UrlUtils.process_url(url, features_dict)
    pred = classifier.predict(url_matrix)
    url_is_malicious = bool(pred[0])
    if console:
```

```
        print(statuses.get(url_is_malicious))
    return url_is_malicious
```

4.3 Додаткові сервіси

Задля більш точної інформації додано функції, які роблять запити до спеціальних ресурсів, щоб отримати такі дані як час реєстрації домену, та перевірити, чи не знаходиться веб-сайт у чорному списку. Бібліотека AIOHTTP надає інструменти щоб пришвидшити цей процес через асинхронні запити, тому не важливо, скільки URL-адрес перевіряє програма, час очікування залежить лише від найповільнішого.

Наразі було впроваджено три сервіси: WHOIS, VirusTotal та Blacklist Checker.

WHOIS – це протокол запитів до бази даних, яка зберігає інформацію про зареєстровані інтернет ресурси. Завдяки цьому сервісу ми можемо дізнатися час реєстрації, час оновлення даних, коли спливає термін дії, а також доменне ім’я, яке нам не доведеться діставати з адреси власноруч.

Клас моделі даних сервісу WHOIS:

```
class WhoIsModel(BaseModel):
    """
    Summary:
        Model for data from whois service.
    """
    valid_response: Optional[bool] = None
    domain_name: Optional[tuple] = None
    creation_date: Optional[tuple] = None
    expiration_date: Optional[tuple] = None
    updated_date: Optional[tuple] = None
```

VirusTotal – онлайн сервіс, який надає можливість перевіряти IP-адреси, файли, ім’я доменів та URL-адреси. Завдяки даному ресурсу, ми можемо перевірити, чи перевірялась адреса взагалі, якщо так, то коли це було, скільки сторонніх сервісів було використано та їх вердикт.

Клас моделі даних сервісу VirusTotal:

```
class VirusTotalModel(BaseModel):
    valid_response: Optional[bool] = False
    message: Optional[str] = None
    permalink: Optional[str] = None
    positives: Optional[int] = None
    response_code: Optional[int] = None
    scan_date: Optional[str] = None
    scan_id: Optional[str] = None
    scans: Optional[dict] = None
    total: Optional[int] = None
    url: Optional[str] = None
```

Blacklist Checker – сервіс, який дозволяє перевіряти IP-адреси, ім’я доменів а також назви електронних скриньок біль ніж у сотні чорних списках. Завдяки цьому ми можемо дізнатися, чи був хоча б один з параметрів нашої URL-адреси пов’язаний колись із злочинними діями.

Клас моделі даних сервісу Blacklist Checker:

```
class BlackListChecker(BaseModel):
    valid_response: Optional[bool] = False
    message: Optional[str] = None
    status: Optional[str] = None
    input_raw: Optional[str] = None
    input_type: Optional[str] = None
    input_domain: Optional[str] = None
    ip_address: Optional[str] = None
    detections: Optional[int] = None
    blacklists: Optional[list] = None
    checks_remaining: Optional[int] = None
```

Окрім вищезазначених сервісів, також було додано декілька окремих перевірок, які надають можливість дізнатися чи є інтернет ресурс, який ми

аналізуємо наразі активним, а також статус його SSL сертифікату. Завдяки цій інформації користувач має змогу побачити, чи можливо, а також безпечно користуватися ресурсом в даний момент.

Функція перевірки URL-адреси в сторонніх сервісах:

```
def perform_services_checks(url_models: URLsSettings) -> URLsSettings:
    """
    Summary:
        Peform checks with additional services.
    Args:
        url_models (URLsSettings): set of the URL settings.
    Returns:
        URLsSettings: set of the URL settings.
    """
    async_requester = AsyncRequester(url_models)
    async_requester.check_url_statuses()
    get_website_main_infos(url_models)
    async_requester.check_url_certs()
    get_whois_info(url_models)
    async_requester.virustotal_check()
    async_requester.blacklist_checker_check()
    return url_models
```

Отриманий результат буде виведено у вигляді лаконічного повідомлення на інтерфейс, який було обрано користувачем для взаємодії з програмою.

4.4 Інтерфейси програми

Програма має три типи інтерфейсу, кожен зі своїми особливостями, тому вони підходять для різних ситуацій. Перший тип – CLI, реалізований за допомогою вбудованих модулів мови Python, він чудово підходить для людей знайомих з програмування, а також тих, хто бажає не тільки користуватися додаток задля

ідентифікації шкідливих URL-адрес, а і використовувати власні дані щоб оновлювати існуючу модель, або навчати власні.

```
usage: cli.py [-h] [-t | --train | --no-train]
              [--loss {hinge,log_loss,log,modified_huber,squared_hinge,perceptron,squared_error,huber,epsilon_insensitive,squared_e
psilon_insensitive,hinge}]
              [--penalty {l2,l1,elasticnet,l2}] [--max_iter MAX_ITER] [-u | --update | --no-update] [-l LINK]
              [-a | --additional_checks | --no-additional_checks] [-c | --check | --no-check]

Next options currently available for use:

options:
  -h, --help            show this help message and exit
  -t, --train, --no-train
                        Train model with provided data. Example: python cli.py -t (default: False)
  --loss {hinge,log_loss,log,modified_huber,squared_hinge,perceptron,squared_error,huber,epsilon_insensitive,squared_epsilon_insens
itive,hinge}
                        Model loss function
  --penalty {l2,l1,elasticnet,l2}
                        Model penalty function
  --max_iter MAX_ITER   Model max number of iterations
  -u, --update, --no-update
                        Update model data. Example: python cli.py -u (default: False)
  -l LINK, --link LINK  Test link. Example: -l 'https://example.com' Example: -l
                        'https://example.com/products.php?linkcomplet=iphone-6-plus-apple-64gb-cinza-espacial-
                        tele-5-5-retin-4g-camera-8mp-frontal-ios-10-proc.-m8/p/2116558/te/ipho/&amp;id=10'
  -a, --additional_checks, --no-additional_checks
                        Test link with additional services. Can be used only with -l option. Example: -l -a 'https://example.com'
                        Example: -l -a 'https://example.com/products.php?linkcomplet=iphone-6-plus-apple-64gb-cinza-espacial-
                        tele-5-5-retin-4g-camera-8mp-frontal-ios-10-proc.-m8/p/2116558/te/ipho/&amp;id=10' (default: False)
  -c, --check, --no-check
                        Check model accuracy. Example: python cli.py -c (default: False)
```

Рисунок 4.1 – Інтерфейс командного рядка

Другим типом є GUI, він реалізований завдяки веб-фреймворку FastAPI, який окрім того, що є сучасним, швидким та надійним, ще і має вбудовану систему для документування та тестування API.

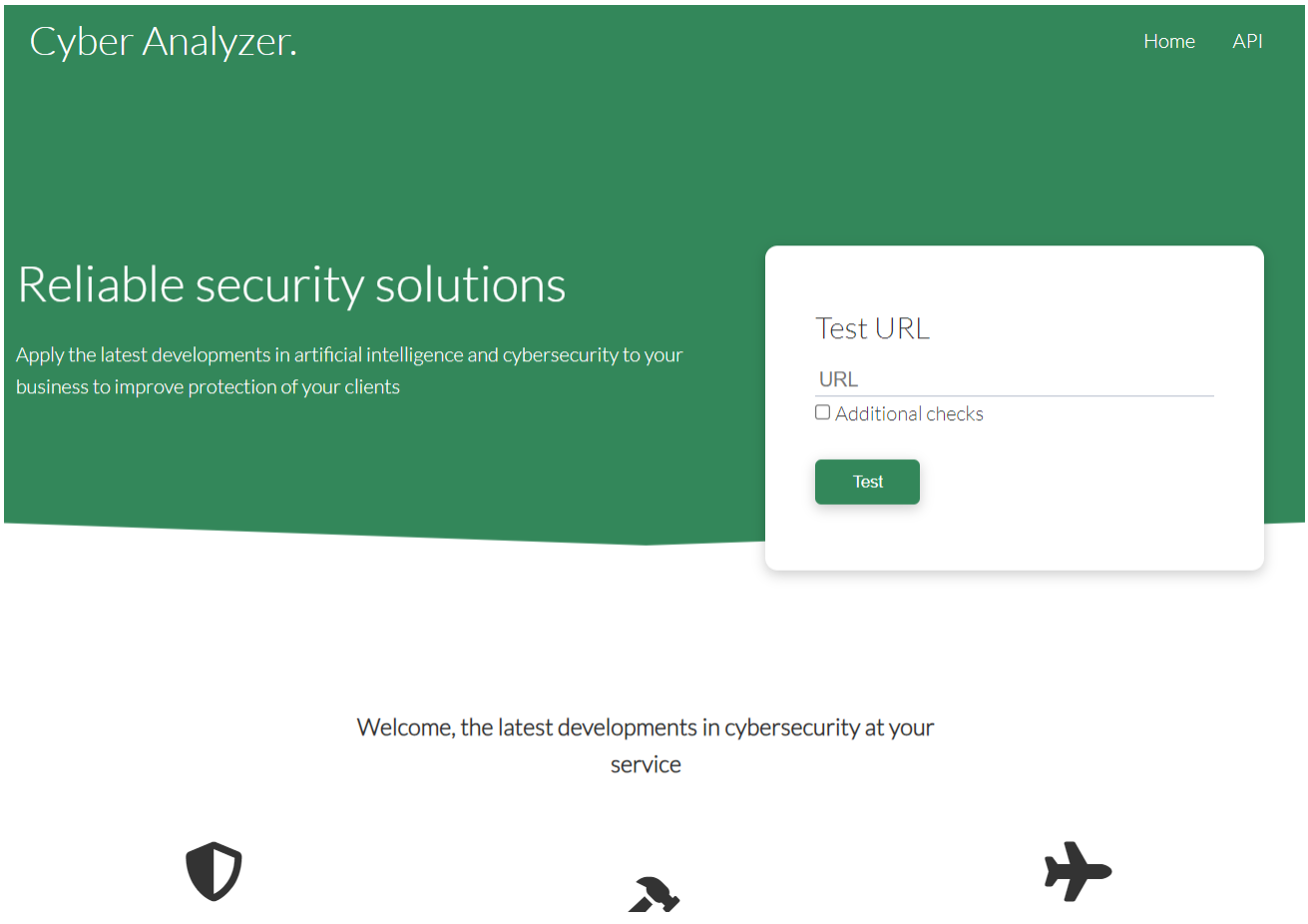

Рисунок 4.2 – Графічний інтерфейс користувача

Третій тип – API, він надає можливість стороннім додаткам взаємодіяти з програмою зручним та простим способом.

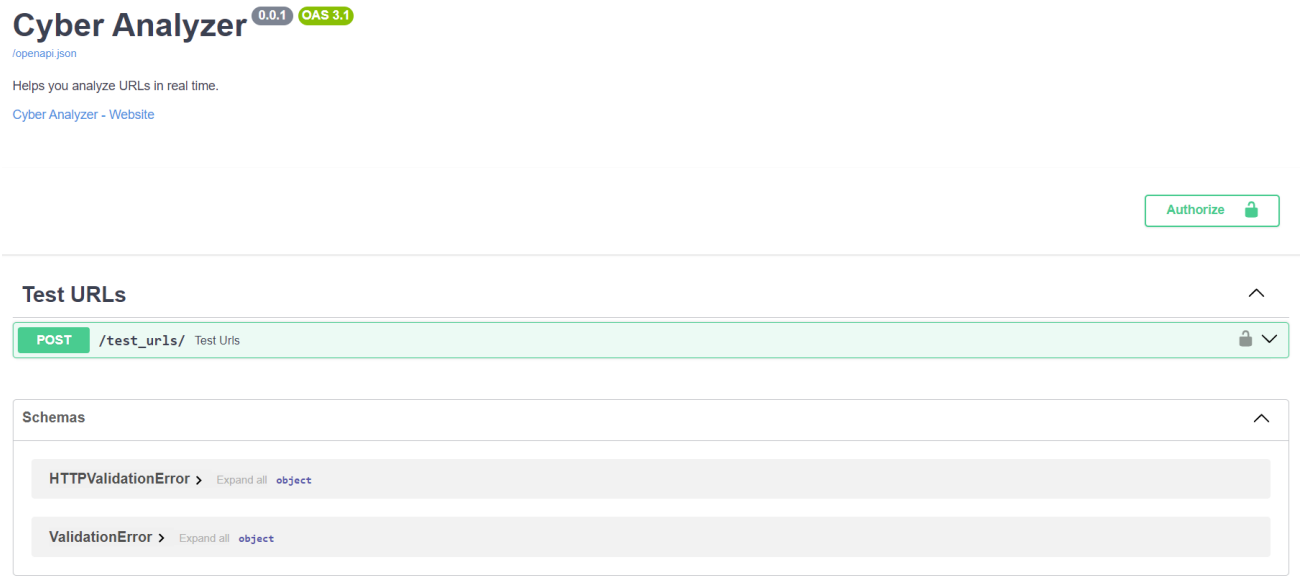

Рисунок 4.3 – Прикладний програмний інтерфейс

CLI як спосіб взаємодії з програмою має як свої переваги, так і негативні сторони. До сильних сторін можна віднести низьке споживання ресурсів та простий спосіб впровадження взаємодії з іншими програмами, що ж до слабких сторін, то тут це насамперед важкість у взаємодії для недосвідчених користувачів та схильність до помилок.

Як було зазначено вище, CLI потребує менше ресурсів, тому всі важкі та тривалі за часом задачі були відведені для даного типу інтерфейсу.

Частина класу, який реалізує CLI:

```
class MainCLI:
    """
    Summary:
        Command-line interface.
    """
    def __init__(self):
        self.args = None
        self.parser = argparse.ArgumentParser(description='CLI.')
```

```
def __call__(self, *args, **kwds):
    self.set_arguments()
    self.check_main_action()

def set_arguments(self):
    """
    Summary:
        Set available options.
    """
    self.parser.description = (
        "Next options currently available for use:"
    )
    self.parser.add_argument(
        "-t",
        "--train",
        default=False,
        action=argparse.BooleanOptionalAction,
        help="Train model with provided data. "
             "Example: python cli.py -t"
    )
    self.parser.add_argument(
        '--loss',
        choices=[
            "hinge", "log_loss", "log", "modified_huber",
            "squared_hinge", "perceptron", "squared_error", "huber",
            "epsilon_insensitive", "squared_epsilon_insensitive", "hinge"
        ],
        default="hinge",
        help='Model loss function'
```

```
        )
        self.parser.add_argument(
            '--penalty',
            choices=["l2", "l1", "elasticnet", "l2"],
            default="l2",
            help='Model penalty function'
        )
        self.parser.add_argument(
            '--max_iter',
            type=int,
            default=1000,
            help='Model max number of iterations'
        )
```

В свою чергу GUI, завдяки з’єднанню з веб-сервером, буде значно швидше приймати URL-адреси та повертати результати перевірки, при цьому використання ресурсів сервера в кожен конкретний проміжок часу буде зменшено, що дозволить більшій кількості користувачів скористатися додатком, як через форму на веб-сайті так і через сторонні програми.

Функція, яка відображає головну сторінку GUI:

```
@logger.catch
@app.get("/", include_in_schema=False, response_class=HTMLResponse)
async def get_index(request: Request):
    """
    Summary:
    Args:
        request (Request): request instance.
    Returns:
        TemplateResponse
    """
    response_data = get_visualizations_data()
```

```
    response_data.update({"request": request})
    response = templates.TemplateResponse(
        "index.html",
        response_data
    )
    return response
```

Також треба зазначити, що простота та легкість у використанні графічного інтерфейсу зробить взаємодію з додатком приємною справою навіть для користувачів з низьким рівнем володіння комп'ютером.

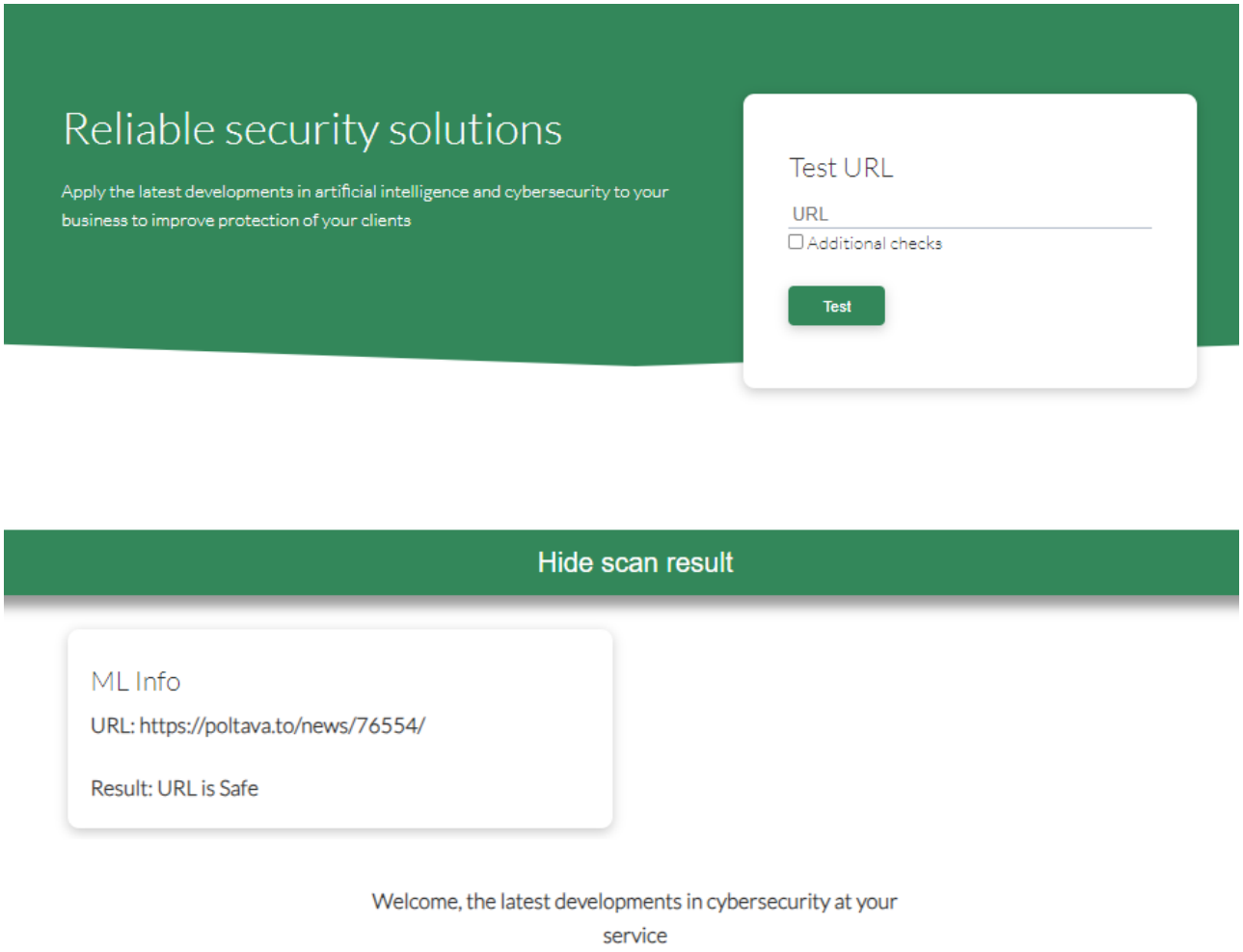

Рисунок 4.4 – Приклад головної сторінки з перевіреною URL-адресою

Окрім можливості перевірити URL-адресу за допомогою натренованої моделі, користувач GUI також має змогу зробити запити до додаткових сервісів, щоб отримати більше інформації про веб-сайт.

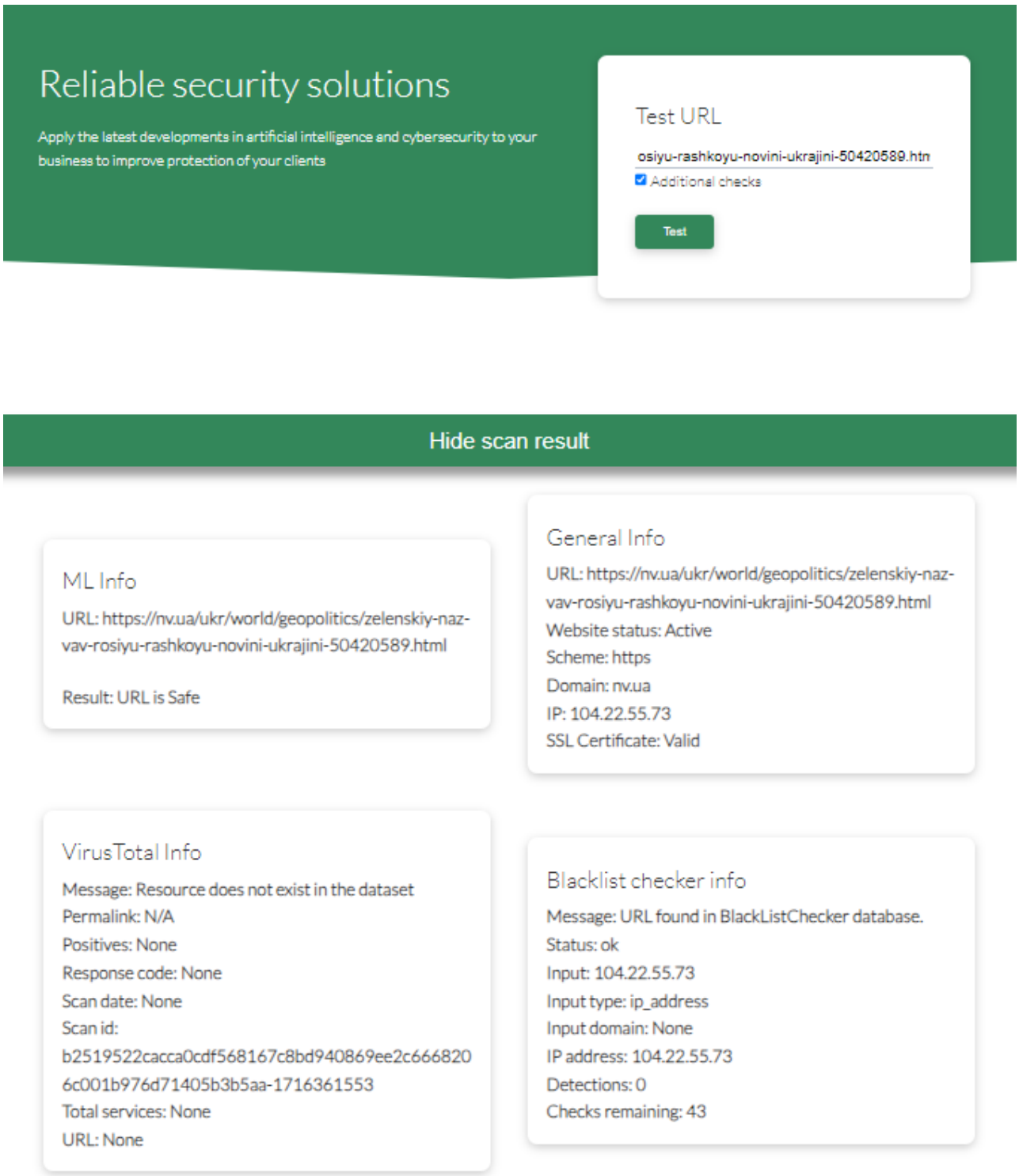

Рисунок 4.5 – Приклад отриманої інформації з додаткових сервісів

Фреймворк FastAPI полегшує реалізацію, тестування, документування та розширення API, який в свою чергу дозволяє різним програмним системам спілкуватися та взаємодіяти одна з одною, незалежно від базових платформ або технологій. Це сприяє комунікації, та робить набагато простішим процес інтеграції між різними системами.

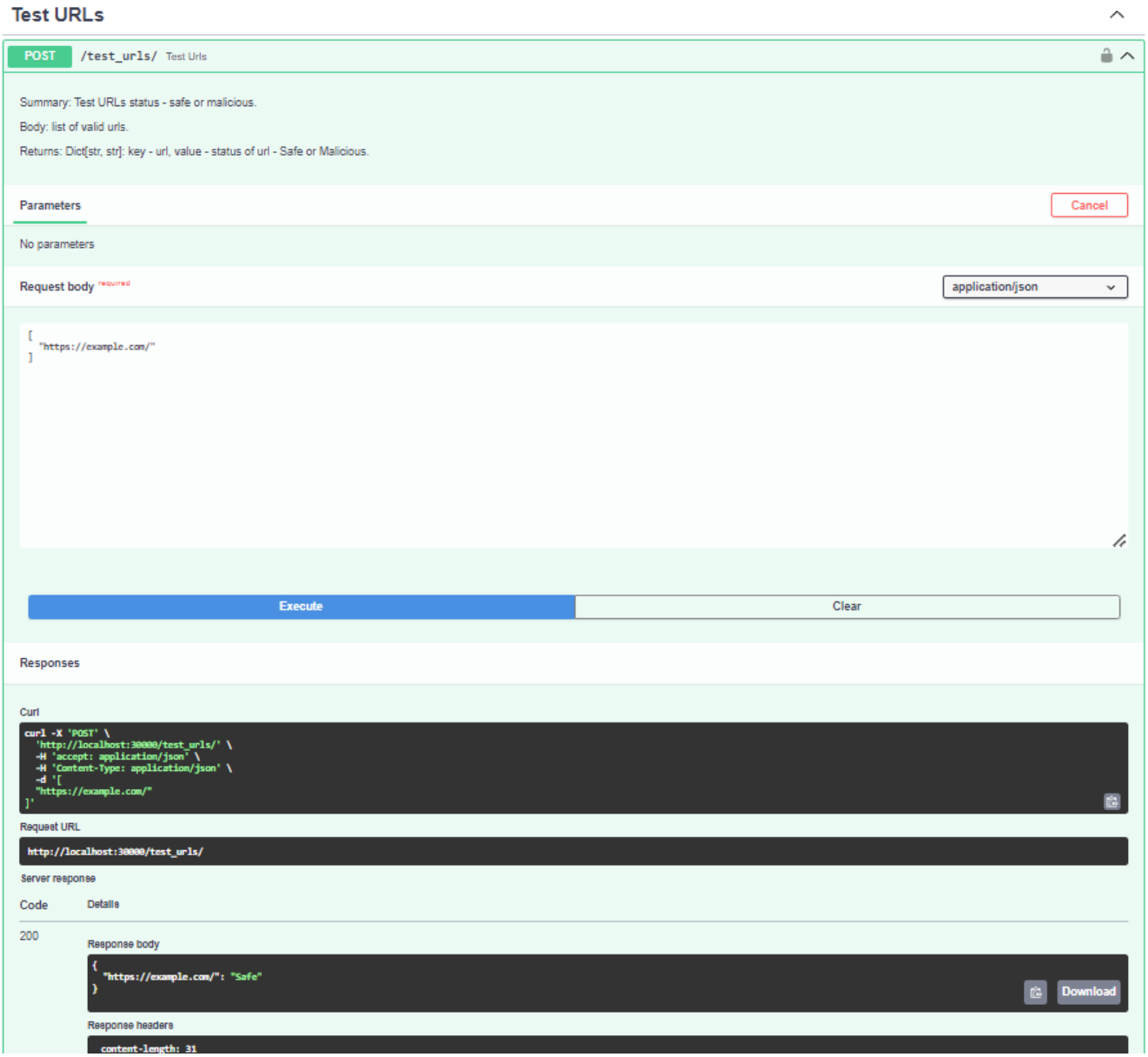

Рисунок 4.6 – Вбудована система документування та тестування API

Блочний дизайн API, дозволяє компонентам програмного забезпечення взаємодіяти через чітко визначені інтерфейси. Цей модульний підхід покращує підтримку коду, масштабованість і повторне використання, оскільки окремі компоненти можна розробляти, тестувати та оновлювати незалежно. Також, однією з цінних рис даного інтерфейсу є те, що він заохочує розвиток екосистеми навколо платформи, залучаючи розробників, а також потенційних партнерів і клієнтів, які можуть використовувати можливості додатку та робити свій внесок у нього. Таке зростання екосистеми покращує ціннісну пропозицію базового програмного забезпечення та сприяє залученню спільноти.

Велика кількість позитивних сторін API робить його імплементацію необхідністю для даної програми, тому було створено окрему точку доступу, через яку будь-які додатки, та навіть звичайні користувачі зможуть передавати довільну кількість URL-адрес задля отримання системного аналізу цих ресурсів. За необхідністю нові точки доступу можуть бути додані, впроваджуючи додатковий функціонал, при цьому існуюча кодова база залишиться незмінною.

Підсумовуючи, можна сказати, що хоч програмна реалізація виявилась досить великою та розгалуженою, кожна її частина має під собою вагомі причини для існування. CLI створений для досвідчених користувачів, він надає можливість не тільки використовувати існуючу модель, а і створювати свої, експериментувати з налаштуванням та навіть використовувати програму у складних сценаріях. В свою чергу GUI, в нашому випадку реалізований через веб-інтерфейс, має просту і лаконічну структуру, він побудований для звичайного, пересічного користувача, який не знайомий з технологіями програмування та AI, і має бажання лише перевіряти веб-сторінки на безпечність. Останнім є API, він створений для сторонніх сервісів. Якщо розробник бажає додати дану програму до свого додатку, він може зайти на спеціальну сторінку з назвою “API”, на якій у стандартній формі описано, що потрібно передавати у запиті, на яку адресу, а також яка повинна бути відповідь при коректній та помилковій введеній інформації. Окрім цього, він може перевірити працездатність програми на тій самій сторінці, щоб переконатись, що при інтеграції з його додатком все буде працювати правильно.

ВИСНОВКИ

У наш час, коли технології розвиваються з неймовірною швидкістю, проблема безпеки користувачів та активів компаній встає гостро як ніколи. Кратне збільшення об’єму даних, які отримують команди з комп’ютерної безпеки лише погіршує ситуацію, адже комплексні програми по виявленню інформаційних загроз старого типу не можуть впоратися з цією проблемою. Результат стає перевантажений відділ безпеки, який замість того, щоб стежити за глобальною ситуацією у мережі, розбирає безмежну кількість спрацювань, які надсилають системи по виявленню загроз. Вагомий вклад в цю проблему вносять хакери, та інші злочинці, які роблять свої атаки більш витонченими та професійними. Однією з таких загроз є шкідливі URL-адреси. Шкідлива URL-адреса – посилання, створене з метою обману та шахрайства. Натиснувши на нього, користувач може завантажити безліч шкідливих програм, які скомпрометують його машину чи навіть цілу мережу. Ці адреси є добре відомими загрозами, які діють як ефективний інструмент для розповсюдження вірусів, комп’ютерних черв’яків та інших типів зловмисного програмного забезпечення в Інтернеті. Їх можуть надсилати за допомогою електронної пошти, текстових повідомлень, спливаючої реклами в браузері тощо.

Для подолання цих проблем фахівці у галузі комп’ютерних технологій рекомендують використовувати новітні досягнення в царині AI. Передові розробки в сфері машинного навчання, що є підрозділом AI, легко справляються з великою кількістю даних, вивільняючи людський ресурс, а також роблять передбачення, які дозволяють виявляти нові типи атак.

Написана програма надає можливість користувачам перевіряти URL-адреси на безпечність. Окрім цього, за бажанням, можливо продовжувати навчання існуючої моделі для покращення її результатів, або тренувати нові. Користувачі в змозі працювати з програмою через різні інтерфейси, це робить її більш привабливою для широкої аудиторії. Кожен інтерфейс має свої вимоги та призначення. Як було зазначено у роботі, технології AI не ідеальні і мають свої проблеми, тому було

впроваджено додаткові перевірки – робота зі сторонніми сервісами, ціль яких зменшити кількість хибних спрацювань.

Для розробки елементів застосунку була використана мова Python, та бібліотеки Scikit-learn, NumPy, Pandas, NLTK, FastAPI, AIOHTTP.

У кваліфікаційній роботі виконано поставлені завдання:

- розглянуто переваги та недоліки використання AI в інформаційній безпеці, концепції машинного та глибокого навчання а також алгоритми AI для класифікацій;
- проведено огляд та аналіз існуючих сервісів для виявлення шкідливих URL-адрес;
- розглянуто та досліджено мову програмування Python, як інструмент для створення додатків на основі технологій AI;
- створено блок-схему роботи програмного забезпечення;
- написано алгоритм роботи програми;
- написано програму для тренування і використання моделі AI; описано результати програмування елементів застосунку;
- програму перевірено.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. T. Emmanuel, Machine Learning for Cybersecurity Cookbook / T. Emmanuel – Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-78961-467-1, November 2019 – 338p.
2. P. Alessandro, Hands-On Artificial Intelligence for Cybersecurity / P. Alessandro – Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-78980-402-7, August 2019 – 524p.
3. D. Ravi, Practical AI for Cybersecurity. First edition published / D. Ravi – CRC Press 6000 Broken Sound Parkway NW, Suite 300 Boca Raton, FL 33487- 2742, 2021 – 293p.
4. G. Padmavathi. Handbook of Research on Machine and Deep Learning Applications for Cyber Security / G. Padmavathi, D. Shanmugapriya – Published in the United States of America by IGI Global Information Science Reference (an imprint of IGI Global) 701 E. Chocolate Avenue Hershey PA, USA 17033. 2020 – 507p.
5. S. Dafydd, The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws : 2nd edition / S. Dafydd, P. Marcus – Published by John Wiley & Sons, Inc. 10475 Crosspoint Boulevard Indianapolis. September 2011 – 912p.
6. Chiheb Chebbi, Mastering Machine Learning for Penetration Testing – Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-78899-740-9, June 2018 – 340p.
7. Davey Gibian, Hacking Artificial Intelligence – Published by Rowman & Littlefield An imprint of The Rowman & Littlefield Publishing Group, Inc. 4501 Forbes Boulevard, Suite 200, Lanham, Maryland 20706. 978-1-538-15508-0, May 2022 – 193p.
8. Donald A. Tevault, Mastering Linux Security and Hardening – Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-83898-177-8, February 2020 – 652p.

9. Hadelin de Ponteves, AI Crash Course – Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-83864-535-9, November 2019 – 361p.
10. Maxim Lapan, Deep Reinforcement Learning Hands-On – Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-83882-699-4, January 2020 – 827p.
11. Rowel Atienza, Advanced Deep Learning with TensorFlow 2 and Keras – Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-83882-165-4, February 2020 – 513p.
12. Sherwin John C. Tragura, Building Python Microservices with FastAPI – Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-80324-596-6, August 2022 – 418p.
13. Soledad Galli, Python Feature Engineering Cookbook – Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-78980-631-1, January 2020 – 558p.
14. An Introduction to Statistical Learning: textbook [J. Gareth, W. Daniela, H. Trevor, T. Robert] – 2nd edition, revised and supplemented – Springer Science+Business Media, LLC, part of Springer Nature 2021 – 616p.
15. Machine Learning and Security [Clarence Chio, David Freeman] – Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. ISBN: 978-1-491-97990-7, February 2018 – 385p.
16. Machine Learning with Python Cookbook [Kyle Gallatin, Chris Albon] – Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. ISBN: 978-1-098-13572-0, July 2023 – 416p.
17. Lawrence W. 9 Best URL Scanners to Check Link is Safe from Malware [Електронний ресурс] – Режим доступу: URL: https://www.guru99.com/url-scanners-checker.html – Назва з екрану. – Дата звернення: 23.03.24
18. Ron S. What Are Malicious URLs And Links? How To Identify And Fight Them [Електронний ресурс] – Режим доступу: URL:

https://www.clearnetwork.com/malicious-urls/ – Назва з екрану. – Дата звернення: 13.03.24

19. Прикладний програмний інтерфейс [Електронний ресурс] – Режим доступу: URL: https://uk.wikipedia.org/wiki/Прикладний_програмний_інтерфейс – Назва з екрану. – Дата звернення: 03.03.24
20. Інтерфейс командного рядка [Електронний ресурс] – Режим доступу: URL: https://uk.wikipedia.org/wiki/Інтерфейс_командного_рядка – Назва з екрану. – Дата звернення: 03.03.24
21. Графічний інтерфейс користувача [Електронний ресурс] – Режим доступу: URL: https://uk.wikipedia.org/wiki/Графічний_інтерфейс_користувача – Назва з екрану. – Дата звернення: 15.03.24
22. Norton Security Review [Електронний ресурс] – Режим доступу: URL: https://www.cloudwards.net/norton-security-review/ – Назва з екрану. – Дата звернення: 23.03.24
23. WhoisXML API Enterprise API and Data Feed Packages [Електронний ресурс] – Режим доступу: URL: https://www.trustradius.com/products/whoisxml-api-enterprise-api-and-data-feed-packages/reviews?qs=pros-and-cons#overview – Назва з екрану. – Дата звернення: 23.03.24
24. Why Is Python Best Adapted to AI and Machine Learning? [Електронний ресурс] – Режим доступу: URL: https://www.turing.com/kb/python-best-adapted-to-ai-and-machine-learning – Назва з екрану. – Дата звернення: 21.03.24
25. Олексійчук Ю. Ф., Вовк О.В. Використання штучного інтелекту для виявлення шкідливих веб-адрес. / ПУЕТ. Полтава, 2024. 2 с.
26. Вовк О.В. Організація освітнього процесу при підготовці кадрів в сучасних умовах: основні тенденції й особливості організації навчання з використанням сучасних цифрових трансформацій та інноваційних технологій. / ПУЕТ. Полтава, 2024. 10 с.

ДОДАТКИ

КОД ПРОЄКТА

Модуль web.py веб-додатку

```
"""Web application."""
import secrets
from typing import Dict

import uvicorn
from fastapi import Depends, FastAPI, Form, HTTPException, Request, status
from fastapi.responses import HTMLResponse
from fastapi.security import HTTPBasic, HTTPBasicCredentials
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from loguru import logger
from pydantic import ValidationError, AnyUrl, BaseModel

import settings
from utils.general_utils import ModelUtils
from utils.visualization_utils import get_visualizations_data
from utils.services_utils import perform_services_checks
from utils.custom_models import URLsSettings, URLSettings
from utils.custom_filters import underscore_to_whitespace, any_to_str



app = FastAPI(
    title="Cyber Analyzer",
    description="Helps you analyze URLs in real time.",
    version="0.0.1",
    contact={
        "name": "Cyber Analyzer",
        "url": settings.WEBSITE_ADDRESS
    }
)
app.mount("/static", StaticFiles(directory="static"), name="static")
security = HTTPBasic()
templates = Jinja2Templates(directory="templates")
templates.env.filters["underscore_to_whitespace"] = underscore_to_whitespace
templates.env.filters["any_to_str"] = any_to_str



class FormModel(BaseModel):
    url: str
    additional_checks: bool
```

```
def validate_credentials(
    credentials: HTTPBasicCredentials = Depends(security, use_cache=False)
) -> bool:
    """
    Summary:
        Check if provided credentials valid.

    Args:
        credentials (HTTPBasicCredentials, optional): username and password.

    Raises:
        HTTPException: Invalid username or password.

    Returns:
        bool: Show if credentials is valid.
    """
    input_user_name = credentials.username.encode("utf-8")
    input_password = credentials.password.encode("utf-8")

    is_username = secrets.compare_digest(
        input_user_name, settings.USER_NAME.encode("utf-8")
    )
    is_password = secrets.compare_digest(
        input_password, settings.USER_PASS.encode("utf-8")
    )
    if is_username and is_password:
        return True
    raise HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Invalid credentials",
        headers={"WWW-Authenticate": "Basic"}
    )


class URLs(FastAPI):
    """
    Summary:
        Provide list of endpoints.
    """
    @app.post("/test_urls/", tags=["Test URLs"], response_model=Dict[str, str])
    async def test_urls(
        urls: tuple[AnyUrl, ...],
```

```
    authenticated: str = Depends(validate_credentials)
) -> Dict[str, str]:
    """
    Summary:
        Test URLs status - safe or malicious.

    Body:
        list of valid urls.

    Returns:
        Dict[str, str]: key - url,
        value - status of url - Safe or Malicious.
    """
    statuses = {True: "Malicious", False: "Safe"}
    if authenticated:
        checked_urls = {
            str(url): statuses.get(ModelUtils.test_url(str(url)))
            for url in urls
        }
        return checked_urls
    return {"message": "User is not authenticated"}

@logger.catch
@app.get("/", include_in_schema=False, response_class=HTMLResponse)
async def get_index(request: Request):
    """
    Summary:

    Args:
        request (Request): request instance.

    Returns:
        TemplateResponse
    """
    response_data = get_visualizations_data()
    response_data.update({"request": request})
    response = templates.TemplateResponse(
        "index.html",
        response_data
    )
    return response

@logger.catch
@app.post("/", include_in_schema=False, response_class=HTMLResponse)
```

```
def post_index(
    request: Request,
    url: str = Form(...),
    additional_checks: bool = Form(False),
    authenticated: bool = Depends(validate_credentials, use_cache=False)
):
    """Summary:

    Args:
        request (Request): request instance.
        url (str, optional): provided url.
        authenticated (str, optional): auth checks.

    Returns:
        TemplateResponse
    """
    statuses = {True: "URL is Malicious", False: "URL is Safe"}
    if authenticated:
        error_message = None
        checked_url = None
        services_checks = None
        scan_result_exist = False
        try:
            AnyUrl(url=url)
        except ValidationError:
            error_message = "Invalid URL"
        else:
            checked_url = statuses.get(
                ModelUtils.test_url(url)
            )
            scan_result_exist = True
            if additional_checks:
                services_checks = perform_services_checks(
                    URLsSettings(urls=(URLSettings(url=url),))
                )
                services_checks = services_checks.model_dump()
        response_data = get_visualizations_data()
        response_data.update(
            {
                "request": request,
                "error_message": error_message,
                "checked_url": checked_url,
                "provided_url": url,
                "scan_result": services_checks,
```

```
            "scan_result_exist": scan_result_exist,
        }
    )
    response = templates.TemplateResponse(
        "index.html",
        response_data
    )
    return response


if __name__ == "__main__":
    uvicorn.run(
        app,
        host=settings.SERVICE_HOST,
        port=int(settings.SERVICE_PORT)
    )
```

Модуль settings.py

```
import os
from loguru import logger

from dotenv import load_dotenv

PROJECT_ROOT = os.path.abspath(os.path.dirname(__file__))

load_dotenv(verbose=True, dotenv_path=os.path.join(PROJECT_ROOT, '.env'))

# Number of rows processed with every batch.
ROWS_NUMBER = os.getenv("ROWS_NUMBER")

# path to file with ngram combinations.
NGRAM_PATH = os.getenv("NGRAM_PATH")

# path to dataset file.
DATASET_PATH = os.getenv("DATASET_PATH")

# path to model file.
MODEL_PATH = os.getenv("MODEL_PATH")

# auth creds.
USER_NAME = os.getenv("USER_NAME")
USER_PASS = os.getenv("USER_PASS")

# web settings
```

```
SERVICE_HOST = os.getenv("SERVICE_HOST")
SERVICE_PORT = os.getenv("SERVICE_PORT")

# URL address
WEBSITE_ADDRESS= os.getenv("WEBSITE_ADDRESS")

# Services:
# VirustTotal
VIRUS_TOTAL_REPORT_URL = os.getenv("VIRUS_TOTAL_REPORT_URL")
VIRUS_TOTAL_SCAN_URL = os.getenv("VIRUS_TOTAL_SCAN_URL")
VIRUS_TOTAL_API_KEY = os.getenv("VIRUS_TOTAL_API_KEY")

# api blacklist checker - (can check: email, domain or IP address)
BLACKLIST_CHECKER_API_KEY = os.getenv("BLACKLIST_CHECKER_API_KEY")
BLACKLIST_CHECKER_URL = os.getenv("BLACKLIST_CHECKER_URL")

logger.add(
    sink=os.path.join("debug_logs", "warning.log"),
    level="WARNING",
    rotation="1 day",
    compression="zip",
    encoding="utf-8",
    enqueue=True,
    backtrace=True,
    diagnose=False,
    serialize=True
)
```

Модуль cli.py

```
"""Project CLI."""
import sys
import argparse

from pydantic import ValidationError, AnyUrl
from utils.general_utils import ModelUtils
from utils.services_utils import perform_services_checks, render_services_checks
from utils.custom_models import URLsSettings, URLSettings

from settings import PROJECT_ROOT

sys.path.append(PROJECT_ROOT)



class MainCLI:
```

```
"""
Summary:
    Command-line interface.
"""
def __init__(self):
    self.args = None
    self.parser = argparse.ArgumentParser(description='CLI.')

def __call__(self, *args, **kwds):
    self.set_arguments()
    self.check_main_action()

def set_arguments(self):
    """
    Summary:
        Set available options.
    """
    self.parser.description = (
        "Next options currently available for use:"
    )
    self.parser.add_argument(
        "-t",
        "--train",
        default=False,
        action=argparse.BooleanOptionalAction,
        help="Train model with provided data. "
             "Example: python cli.py -t"
    )
    self.parser.add_argument(
        '--loss',
        choices=[
            "hinge", "log_loss", "log", "modified_huber",
            "squared_hinge", "perceptron", "squared_error", "huber",
            "epsilon_insensitive", "squared_epsilon_insensitive", "hinge"
        ],
        default="hinge",
        help='Model loss function'
    )
    self.parser.add_argument(
        '--penalty',
        choices=["l2", "l1", "elasticnet", "l2"],
        default="l2",
        help='Model penalty function'
    )
```

```
        self.parser.add_argument(
            '--max_iter',
            type=int,
            default=1000,
            help='Model max number of iterations'
        )
        self.parser.add_argument(
            "-u",
            "--update",
            default=False,
            action=argparse.BooleanOptionalAction,
            help="Update model data. "
                 "Example: python cli.py -u"
        )
        self.parser.add_argument(
            "-l",
            "--link",
            help="Test link. "
                 "Example: -l https://example.com "
                 "Example: -l https://example.com/products.php?linkcomplet=iphone-6-plus-apple-64gb-cinza-espacial-tele-5-5-retin-4g-camera-8mp-frontal-ios-10-proc.-m8/p/2116558/te/ipho/&amp;amp;id=10"
        )
        self.parser.add_argument(
            "-a",
            "--additional_checks",
            default=False,
            action=argparse.BooleanOptionalAction,
            help="Test link with additional services. "
                 "Can be used only with -l option. "
                 "Example: -l https://example.com -a"
                 "Example: -l https://example.com/products.php?linkcomplet=iphone-6-plus-apple-64gb-cinza-espacial-tele-5-5-retin-4g-camera-8mp-frontal-ios-10-proc.-m8/p/2116558/te/ipho/&amp;amp;id=10 -a"
        )
        self.parser.add_argument(
            "-c",
            "--check",
            default=False,
            action=argparse.BooleanOptionalAction,
            help="Check model accuracy. "
                 "Example: python cli.py -c"
        )
        self.args = self.parser.parse_args()
```

```
def check_main_action(self):
    """
    Summary:
        Check which action user selected:
        get additional information,
        create report,
        get general information.
    """
    if self.args.train:
        ModelUtils.train_model(
            loss=self.args.loss,
            penalty=self.args.penalty,
            max_iter=self.args.max_iter
        )
    elif self.args.update:
        ModelUtils.train_model(
            loss=self.args.loss,
            penalty=self.args.penalty,
            max_iter=self.args.max_iter,
            update=True,
        )
    elif self.args.link:
        try:
            AnyUrl(url=self.args.link)
        except ValidationError:
            print("Invalid URL")
        else:
            ModelUtils.test_url(self.args.link, console=True)
            if self.args.additional_checks:
                services_checks = perform_services_checks(
                    URLsSettings(urls=(URLSettings(url=self.args.link),))
                )
                render_services_checks(services_checks)
    elif self.args.check:
        ModelUtils.check_model_accuracy()
    else:
        print(
            "For Linux, iOS. Write: python3 cli.py --help \n"
            "For Windows. Write: python cli.py --help \n"
            "For virtual environments. Write: python cli.py --help \n"
            "to check available options. \n"
            "You can use only one flag at the same time!"
        )
```

```
if __name__ == '__main__':
    init_cli = MainCLI()
    init_cli()
```

Файл requirements.txt

```
aiohttp==3.9.5
aiosignal==1.3.1
annotated-types==0.6.0
anyio==3.7.1
async-timeout==4.0.3
attrs==23.2.0
certifi==2023.7.22
charset-normalizer==3.3.2
click==8.1.7
colorama==0.4.6
exceptiongroup==1.1.3
fastapi==0.104.1
fastapi-csrf-protect==0.3.3
frozenlist==1.4.1
gunicorn==21.2.0
h11==0.14.0
httptools==0.6.1
idna==3.4
itsdangerous==2.1.2
Jinja2==3.1.3
joblib==1.3.2
loguru==0.7.2
MarkupSafe==2.1.3
multidict==6.0.5
nltk==3.8.1
numpy==1.26.1
packaging==23.2
pandas==2.1.3
pydantic==2.4.2
pydantic_core==2.10.1
python-dateutil==2.8.2
python-dotenv==1.0.1
python-multipart==0.0.6
python-whois==0.9.4
pytz==2023.3.post1
PyYAML==6.0.1
regex==2023.10.3
requests==2.31.0
scikit-learn==1.3.1
```

scipy==1.11.3

six==1.16.0

sniffio==1.3.0

starlette==0.27.0

threadpoolctl==3.2.0

tqdm==4.66.1

typing_extensions==4.8.0

tzdata==2023.3

urllib3==2.0.7

uvicorn==0.27.0.post1

watchfiles==0.21.0

websockets==12.0

win32-setctime==1.1.0

yarl==1.9.4

Модуль services_utils.py

```
import whois

from .custom_models import URLsSettings, WhoIsModel

from .request_utils import AsyncRequester

from .url_utils import get_url_scheme, get_url_domain, get_domain_ip

from pydantic import ValidationError



def get_website_status_codes(urls: URLsSettings):
    """
    Summary:
        Make async requests to notifications api.

    Args:
        url (str): Url.

    Returns:
        int: Website status code.
    """
    async_requester = AsyncRequester(urls)
    async_requester.check_url_statuses()
```

```
def get_whois_info(url_models: URLsSettings):
    for url in url_models.urls:
        whois_data = whois.whois(str(url.url))
        try:
            whois_obj = WhoIsModel(
                valid_response=True,
                domain_name=tuple(
                    whois_data.domain_name
                    if isinstance(whois_data.domain_name, list)
                    else [whois_data.domain_name]
                ),
                creation_date=tuple(
                    whois_data.creation_date
                    if isinstance(whois_data.creation_date, list)
                    else [whois_data.creation_date]
                ),
                expiration_date=tuple(
                    whois_data.expiration_date
                    if isinstance(whois_data.expiration_date, list)
                    else [whois_data.expiration_date]
                ),
                updated_date=tuple(
                    whois_data.updated_date
                    if isinstance(whois_data.updated_date, list)
                    else [whois_data.updated_date]
                )
            )
        except ValidationError as _:
            whois_obj = WhoIsModel(
                valid_response=False
            )
        url.whois_data = whois_obj
```

```
def get_website_main_infos(url_models: URLsSettings):
    for url in url_models.urls:
        url.scheme = get_url_scheme(url.url)
        url.domain = get_url_domain(url.url)
        if url.status_code == 200:
            url.ip = get_domain_ip(url.domain)


def perform_services_checks(url_models: URLsSettings) -> URLsSettings:
    """
    Summary:
        Peform checks with additional services.

    Args:
        url_models (URLsSettings): set of the URL settings.

    Returns:
        URLsSettings: set of the URL settings.
    """
    async_requester = AsyncRequester(url_models)
    async_requester.check_url_statuses()
    get_website_main_infos(url_models)
    async_requester.check_url_certs()
    get_whois_info(url_models)
    async_requester.virustotal_check()
    async_requester.blacklist_checker_check()
    return url_models

def render_services_checks(url_models: URLsSettings):
    if url_models.urls[0].status_code != 404:
        print("Website status: Active")
    else:
        print("Website status: Inactive")
    print(f'URL Certificate: {url_models.urls[0].ssl_certificate}')
```

```
    if url_models.urls[0].whois_data.valid_response:
        if isinstance(url_models.urls[0].whois_data.creation_date, tuple):
            print(f'Creation date: {url_models.urls[0].whois_data.creation_date[0]}')
        else:
            print(f'Creation date: {url_models.urls[0].whois_data.creation_date}')
        if isinstance(url_models.urls[0].whois_data.expiration_date, tuple):
            print(f'Expiration date: {url_models.urls[0].whois_data.expiration_date[0]}')
        else:
            print(f'Expiration date: {url_models.urls[0].whois_data.expiration_date}')
        if isinstance(url_models.urls[0].whois_data.updated_date, tuple):
            print(f'Updated date: {url_models.urls[0].whois_data.updated_date[0]}')
        else:
            print(f'Updated date: {url_models.urls[0].whois_data.updated_date}')
    if url_models.urls[0].virustotal.valid_response:
        print(f'VirusTotal message: {url_models.urls[0].virustotal.message}')
        print(f'VirusTotal positives: {url_models.urls[0].virustotal.positives}')
    if url_models.urls[0].blacklist_checker.valid_response:
        print(
            'Blacklist checker message '
            f'{url_models.urls[0].blacklist_checker.message}'
        )
        print(
            'Blacklist checker status '
            f'{url_models.urls[0].blacklist_checker.status}'
        )
        print(
            'Blacklist checker detections '
            f'{url_models.urls[0].blacklist_checker.detections}'
        )
```

Модуль request_utils.py

```
import asyncio
import typing
```

```
import aiohttp

import settings
from utils.custom_models import (
    URLsSettings, URLSettings, VirusTotalModel, BlackListChecker
)
from pydantic import ValidationError



class AsyncRequester:
    """
    Summary:
        Make async requests to notifications api.
    """
    def __init__(
        self,
        url_models: URLsSettings,
    ) -> None:
        """
        Summary:
            Perform main requests.


        Args:
            url_models (URLsSettings): set of the URL settings.
        """
        self.url_models = url_models


    async def perform_code_request(
        self,
        session: aiohttp.ClientSession,
        url: URLSettings,
    ) -> None:
        """
```

```
    Summary:
        Get notifications from endpoint.

    Args:
        session (aiohttp.ClientSession): Object of interface
        for http requests.
        url (URLSettings): Settings of the URL.
    """
    async with session.request(
        'GET',
        url=str(url.url),
        headers={
            "Content-Type": "application/json",
            "accept": "application/json"
        },
        timeout=5
    ) as response:
        url.status_code = response.status


async def pefrom_cert_request(
    self,
    session: aiohttp.ClientSession,
    url: URLSettings
):
    """
    Summary:
        Perfom request to check certificate.

    Args:
        session (aiohttp.ClientSession): Object of interface
        for http requests
        url (URLSettings): Settings of the URL.
    """
```

```
    if url.scheme and url.scheme.name == "HTTP":
        return
    try:
        async with session.request(
            'GET',
            url=str(url.url),
            headers={
                "Content-Type": "application/json",
                "accept": "application/json"
            },
            timeout=5
        ) as response:
            if response.status == 200:
                url.ssl_certificate = "Valid"
            else:
                url.ssl_certificate = "Invalid"
    except aiohttp.ClientSSLError as e:
        url.ssl_certificate = "Invalid or cannot be verified."
        print("SSL certificate is invalid or cannot be verified.", f"My error is {e}")
    except aiohttp.ClientError as e:
        print("Error:", e)


async def get_virustotal_scan_id(
    self,
    session: aiohttp.ClientSession,
    url: URLSettings
):
    """
    Summary:
        Get resource scan id.

    Args:
        session (aiohttp.ClientSession): Object of interface
```

```
    url (URLSettings): Settings of the URL.
"""
data = {
    'apikey': settings.VIRUS_TOTAL_API_KEY, 'url': str(url.url)
}
try:
    async with session.request(
        'POST',
        url=settings.VIRUS_TOTAL_SCAN_URL,
        data=data,
        timeout=5,
    ) as response:
        response_data = await response.json()
        if response_data:
            if response_data.get('response_code') == 1:
                try:
                    model_data = VirusTotalModel(
                        valid_response=True,
                        message=response_data.get(
                            "verbose_msg",
                            "URL found in VirusTotal's database."
                        ),
                        scan_id=response_data.get('scan_id')
                    )
                except ValidationError as _:
                    model_data = VirusTotalModel(
                        valid_response=False,
                        message=response_data.get(
                            "verbose_msg",
                            ("Data types have changed, "
                             "model need to be updated.")
                        )
                    )
```

```
                url.virustotal = model_data
            else:
                model_data = VirusTotalModel(
                    valid_response=True,
                    message=response_data.get(
                        "verbose_msg",
                        "URL not found in VirusTotal's database."
                    )
                )
                url.virustotal = model_data
        else:
            print("Failed to make request to VirusTotal API.")
    except aiohttp.ClientError as e:
        print("Error:", e)

async def get_virustotal_report(
    self,
    session: aiohttp.ClientSession,
    url: URLSettings
):
    """
    Summary:
        Get report by scan id.

    Args:
        session (aiohttp.ClientSession): Object of interface
        url (URLSettings): Settings of the URL.
    """
    if not url.virustotal.scan_id:
        return
    params = {
        'apikey': settings.VIRUS_TOTAL_API_KEY,
        'resource': str(url.virustotal.scan_id)
```

```
}
try:
    async with session.request(
        'GET',
        url=settings.VIRUS_TOTAL_REPORT_URL,
        params=params,
        timeout=5,
    ) as response:
        response_data = await response.json()
        if response_data:
            if response_data.get('response_code') == 1:
                try:
                    url.virustotal.valid_response = True
                    url.virustotal.message = response_data.get(
                        "verbose_msg",
                        "URL found in VirusTotal's database."
                    )
                    url.virustotal.permalink = response_data.get(
                        'permalink', 'N/A'
                    )
                    url.virustotal.positives = response_data.get(
                        'positives', 0
                    )
                    url.virustotal.response_code = response_data.get(
                        'response_code', 0
                    )
                    url.virustotal.scan_date = response_data.get(
                        'scan_date', 'N/A'
                    )
                    url.virustotal.scans = response_data.get(
                        'scans', {}
                    )
                    url.virustotal.total = response_data.get(
```

```
                        'total', 0
                    )
                    url.virustotal.url = response_data.get(
                        'url', 'N/A'
                    )
                except ValidationError as _:
                    url.virustotal.valid_response = False
                    url.virustotal.message = response_data.get(
                        "verbose_msg",
                        ("Data types have changed, "
                            "model need to be updated.")
                    )
            else:
                url.virustotal.valid_response = True
                url.virustotal.message = response_data.get(
                    "verbose_msg",
                    "URL not found in VirusTotal's database."
                )
        else:
            print("Failed to make request to VirusTotal API.")
    except aiohttp.ClientError as e:
        print("Error:", e)


async def perform_blacklist_checker_request(
    self,
    session: aiohttp.ClientSession,
    url: URLSettings
):
    """
    Summary:
        Perform request to BlackListChecker API.

    Args:
```

```
    session (aiohttp.ClientSession): Object of interface
    url (URLSettings): Settings of the URL.
"""
if not url.ip and not url.domain:
    return
ip_or_domain = url.ip if url.ip else url.domain
full_url = f"{settings.BLACKLIST_CHECKER_URL}{ip_or_domain}"
try:
    async with session.request(
        'GET',
        url=full_url,
        auth=aiohttp.BasicAuth(settings.BLACKLIST_CHECKER_API_KEY, ''),
        timeout=5,
    ) as response:
        response_data = await response.json()
        if response_data:
            try:
                model_data = BlackListChecker(
                    valid_response=True,
                    message=(
                        "URL found in BlackListChecker database."
                    ),
                    status=response_data.get("status", "N/A"),
                    input_raw=response_data.get("input_raw", "N/A"),
                    input_type=response_data.get("input_type", "N/A"),
                    input_domain=response_data.get(
                        "input_domain",
                        "N/A"
                    ),
                    ip_address=response_data.get("ip_address", "N/A"),
                    detections=response_data.get("detections", 0),
                    blacklists=response_data.get("blacklists", []),
                    checks_remaining=response_data.get(
```

```
                    "checks_remaining",
                    0
                )
            )
            except ValidationError as _:
                model_data = BlackListChecker(
                    valid_response=False,
                    message=("Data types have changed, model "
                        "need to be updated.")
                )
            url.blacklist_checker = model_data
        else:
            print("Failed to make request to BlackListChecker API.")
    except aiohttp.ClientError as e:
        print("Error:", e)


async def gather_data(self, func: typing.Callable, ssl=False):
    """
    Summary:
        Gather async tasks.
    """
    async with aiohttp.ClientSession(
        connector=aiohttp.TCPConnector(ssl=ssl)
    ) as session:
        tasks = []
        for url in self.url_models.urls:
            task = asyncio.create_task(func(
                session,
                url
            ))
            tasks.append(task)
        await asyncio.gather(*tasks)
```

```
    def check_url_statuses(self):
        """
        Summary:
            Check status codes of urls.
        """
        asyncio.run(self.gather_data(self.perform_code_request))

    def check_url_certs(self):
        """
        Summary:
            Check certs of urls.
        """
        asyncio.run(self.gather_data(self.pefrom_cert_request, ssl=True))

    def virustotal_check(self):
        """
        Summary:
            Check URL with VirusTotal service.
        """
        asyncio.run(self.gather_data(self.get_virustotal_scan_id))
        asyncio.run(self.gather_data(self.get_virustotal_report))

    def blacklist_checker_check(self):
        """
        Summary:
            Check IP address with BlackListChecker service.
        """
        asyncio.run(
            self.gather_data(self.perform_blacklist_checker_request)
        )
```

Модуль general_utils.py

```
"""Module providing a functions to train and use trained model."""
import itertools
```

```
import json
import os
import re
import sys
import typing

import joblib
import nltk
import numpy as np
import pandas as pd
from loguru import logger
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split

import settings


class NgramUtils:
    """
    Summary:
        Create, save and load ngrams.
    """

    def __init__(self, ngram_path: str = settings.NGRAM_PATH):
        self.ngram_path = ngram_path

    @logger.catch
    def create_n_gram_combinations(self, ngram_size: int = 3) -> typing.Dict:
        """
        Summary:
            Create all possible ngram combinations from letters and numbers.

        Args:
            ngram_size (int, optional): Size of the ngrams.
            Defaults to 3 - trigram.

        Returns:
            dict: Ngrams. Key - ngram, value - number.
        """
        alphanum = [
            'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
            'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
            '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
        ]
```

```
    permutations = itertools.product(alphanum, repeat=ngram_size)
    featues_dict = {}
    counter = 0
    for perm in permutations:
        featues_dict[(''.join(perm))] = counter
        counter = counter + 1
    return featues_dict

@logger.catch
def get_ngram_combinations(self) -> typing.Dict:
    """
    Summary:
        Get ngrams from file or create it, if file doesn't exist.

    Returns:
        dict: Ngrams. Key - ngram, value - number.
    """
    if os.path.isfile(self.ngram_path):
        with open(self.ngram_path, encoding='utf8') as ng:
            try:
                ngrams = json.load(ng)
            except json.decoder.JSONDecodeError:
                ngrams = self.create_n_gram_combinations()
                self.save_ngram_combinations_from_file(ngrams)
    else:
        ngrams = self.create_n_gram_combinations()
        self.save_ngram_combinations_from_file(ngrams)
    return ngrams

def save_ngram_combinations_from_file(self, n_grams: typing.Dict):
    """
    Summary:
        Save ngrams to file.

    Args:
        ngrams (dict): Ngrams. Key - ngram, value - number.
    """
    with open(self.ngram_path, "w", encoding='utf8') as file_obj:
        json.dump(n_grams, file_obj, ensure_ascii=False, indent=4)

@staticmethod
@logger.catch
def generate_ngrams_from_string(
    sentence: str,
```

```
        ngram_size: int = 3
    ) -> typing.List:
        """
        Summary:
            Generate ngram of specific size from string

        Args:
            sentence (str): sentence from which ngrams will be generated.
            ngram_size (int, optional): Size of the ngrams. Defaults to 3.

        Returns:
            List: Ngrams.
        """
        sentence = sentence.lower()
        sentence = ''.join(e for e in sentence if e.isalnum())
        processed_list = []
        for tup in list(nltk.ngrams(sentence, ngram_size)):
            processed_list.append((''.join(tup)))
        return processed_list


class ModelUtils:
    """
    Summary:
        Train, test, save and load model.
    """
    @staticmethod
    @logger.catch
    def train_model(
        loss: str,
        penalty: str,
        max_iter: int,
        update: bool = False,
    ):
        """
        Summary:
            Full training process.

        Args:
            update (bool, optional): Flag show if model will be trained
            or updated. Defaults to False.
        """
        if update:
            classifier = ModelUtils.load_model()
```

```
    else:
        classifier = SGDClassifier(
            loss=loss,
            penalty=penalty,
            max_iter=max_iter
        )
    train_df, test_df = ModelUtils.split_dataframe()
    features_dict = NgramUtils().get_ngram_combinations()
    ModelUtils.insert_data(train_df, classifier, features_dict)
    ModelUtils.save_model(classifier)
    correct, incorrect = ModelUtils.test_model(
        test_df,
        classifier,
        features_dict
    )
    if update:
        print("Model have been updated!")
    else:
        print("Model have been trained!")
    print("Correct Predictions ", correct)
    print("Incorrect Predictions ", incorrect)
    accuracy = (correct / test_df.shape[0]) * 100
    print(f"Accuracy of the model is: {accuracy:.4g} %")

@staticmethod
@logger.catch
def update_model():
    """
    Summary:
        Update model.
    """
    classifier = ModelUtils.load_model()
    dataframe = ModelUtils.load_dataframe()
    features_dict = NgramUtils().get_ngram_combinations()
    ModelUtils.insert_data(dataframe, classifier, features_dict)
    ModelUtils.save_model(classifier)

@staticmethod
@logger.catch
def test_url(
    url: str,
    model_path: str = settings.MODEL_PATH,
    console: bool = False
) -> bool:
```

```
    """
    Summary:
        Test url.

    Args:
        url (str): url
        model_path (str, optional): path to trained model.
        console (bool, optional): print result to console.
        Defaults to False.

    Returns:
        bool: True if url is malicious, False if url is safe.
    """
    statuses = {
        True: "URL is Malicious", False: "URL is Safe"
    }
    classifier = ModelUtils.load_model(model_path)
    features_dict = NgramUtils().get_ngram_combinations()
    url_matrix = UrlUtils.process_url(url, features_dict)
    pred = classifier.predict(url_matrix)
    url_is_malicious = bool(pred[0])
    if console:
        print(statuses.get(url_is_malicious))
    return url_is_malicious

@staticmethod
@logger.catch
def check_model_accuracy():
    """
    Summary:
        Check accuracy of the trained model.
    """
    classifier = ModelUtils.load_model()
    _, test_df = ModelUtils.split_dataframe()
    features_dict = NgramUtils().get_ngram_combinations()
    correct, incorrect = ModelUtils.test_model(
        test_df,
        classifier,
        features_dict
    )
    print("Correct Predictions ", correct)
    print("Incorrect Predictions ", incorrect)
    accuracy = (correct / test_df.shape[0]) * 100
    print(f"Accuracy of the model is: {accuracy:.4g} %")
```

```
@staticmethod
@logger.catch
def save_dataframe(
    dataframe: pd.DataFrame,
    dataframe_path: str = settings.DATASET_PATH
):
    """
    Summary:
        Save dataframe to file.

    Args:
        dataframe (pd.DataFrame): data, that will be used to train model.
        dataframe_path (str, optional): path, where dataframe
        will be saved. Defaults to settings.DATASET_PATH.
    """
    pd.DataFrame.to_csv(dataframe, dataframe_path, index=False)

@staticmethod
@logger.catch
def split_dataframe(
    test_size: float = 0.2,
    dataset_path: str = settings.DATASET_PATH
) -> typing.Tuple:
    """
    Summary:
        Split provided data to train and test dataframes.

    Returns:
        tuple: trand and test dataframes.
    """
    try:
        url_dataframe = pd.read_csv(dataset_path)
    except FileNotFoundError:
        print("Dataframe file not found")
        sys.exit()
    train_df, test_df = train_test_split(
        url_dataframe,
        test_size=test_size
    )
    return train_df, test_df

@staticmethod
@logger.catch
```

```
def preprocess_dataset(
    dataframe: pd.DataFrame,
    features_dict: typing.Dict,
) -> typing.Generator[np.ndarray, np.ndarray, pd.DataFrame]:
    """
    Summary
        Split dataframe to batches.

    Args:
        dataframe (pd.DataFrame): dataframe.
        classifier (SGDClassifier): Linear classifiers
        features_dict (typing.Dict): Ngrams. Key - ngram, value - number.

    Returns:
        tuple: matrix, labels vector and batch.
    """
    rows_number = int(settings.ROWS_NUMBER)
    no_of_batches = int(dataframe.shape[0] / rows_number) + 1
    for i in range(0, no_of_batches):
        start = rows_number * i
        if start + rows_number > dataframe.shape[0]:
            batch = dataframe.iloc[start:, :]
        else:
            batch = dataframe.iloc[start:start + rows_number, :]
        batch = batch.reset_index()
        if batch.empty:
            break
        x, y = UrlUtils.preprocess_batch(
            features_dict,
            batch,
            np.zeros([batch.shape[0], len(features_dict)], dtype="int"),
            np.zeros(batch.shape[0], dtype="int")
        )
        yield x, y, batch

@staticmethod
@logger.catch
def insert_data(
    train_dataframe: pd.DataFrame,
    classifier: SGDClassifier,
    features_dict: typing.Dict
):
    """
    Summary:
```

```
        Insert processed data in the model. Train model.

    Args:
        train_dataframe (pd.DataFrame): test dataframe.
        classifier (SGDClassifier): Linear classifiers
        features_dict (typing.Dict): Ngrams. Key - ngram, value - number.
    """
    for x, y, _ in ModelUtils.preprocess_dataset(
        train_dataframe,
        features_dict
    ):
        classifier.partial_fit(
            x,
            y,
            classes=np.unique(y)
        )

@staticmethod
@logger.catch
def test_model(
    test_dataframe: pd.DataFrame,
    classifier: SGDClassifier,
    features_dict: typing.Dict
) -> tuple:
    """
    Summary
        Test trained model

    Args:
        test_dataframe (pd.DataFrame): test dataframe.
        classifier (SGDClassifier): Linear classifiers
        features_dict (typing.Dict): Ngrams. Key - ngram, value - number.

    Returns:
        tuple: Amount of correct and incorrect predictions.
    """
    correct = 0
    incorrect = 0
    for x, _, batch in ModelUtils.preprocess_dataset(
        test_dataframe,
        features_dict
    ):
        y_pred = classifier.predict(x)
        for index, row in batch.iterrows():
```

```
            if row['label'] == y_pred[index]:
                correct += 1
            else:
                incorrect += 1
    return correct, incorrect

@staticmethod
@logger.catch
def save_model(
    classifier: SGDClassifier,
    file_path: str = settings.MODEL_PATH
):
    """
    Summary:
        Save the trained model to a file

    Args:
        classifier (sklearn.linear_model.SGDClassifier): Trained model.
        file_path (str, optional): path to file, where trained model
        will be saved. Defaults to settings.MODEL_PATH.
    """
    joblib.dump(classifier, file_path)

@staticmethod
@logger.catch
def load_model(
    file_path: str = settings.MODEL_PATH
) -> SGDClassifier:
    """
    Summary:
        Load the saved model

    Args:
        filename (str, optional): Name of the file with trained model.
        Defaults to 'sgd_model.pkl'.

    Returns:
        sklearn.linear_model.SGDClassifier: Trained model.
    """
    try:
        model = joblib.load(file_path)
        return model
    except FileNotFoundError:
        print('Model file not found')
```

```
            sys.exit()

    @staticmethod
    @logger.catch
    def load_dataframe(file_path: str = settings.DATASET_PATH) -> pd.DataFrame:
        """
        Summary:
            Load dataframe from file.

        Args:
            file_path (str, optional): Path to dataframe file.
            Defaults to settings.DATASET_PATH.

        Returns:
            pd.DataFrame: loaded dataframe.
        """
        try:
            dataframe = pd.read_csv(file_path)
            return dataframe
        except FileNotFoundError:
            print('Dataframe file not found')
            sys.exit()


class UrlUtils:
    """
    Summary:
        Process urls.
    """
    @staticmethod
    @logger.catch
    def preprocess_batch(
        features_dict: typing.Dict,
        batch: pd.DataFrame,
        x: np.ndarray,
        y: np.ndarray
    ) -> typing.Tuple[np.ndarray, np.ndarray]:
        """
        Summary:
            Apply vectorization process to current batch.

        Args:
            features_dict (typing.Dict): Ngrams. Key - ngram, value - number.
            batch (pd.DataFrame): current batch of data.
```

```
        x (np.ndarray): matrix. Each row is a vector.
        y (np.ndarray): labels vector.

    Returns:
        typing.Tuple: matrix and labels vector.
    """
    for index, row in batch.iterrows():
        url = UrlUtils.clean_url(row['url'])
        for gram in NgramUtils.generate_ngrams_from_string(url):
            try:
                x[index][features_dict[gram]] = \
                    x[index][features_dict[gram]] + 1
            except KeyError:
                continue
        y[index] = int(row['label'])
    return x, y

@staticmethod
@logger.catch
def process_url(url: str, features_dict: typing.Dict) -> np.ndarray:
    """
    Summary:
        Apply vectorization process to url.

    Args:
        url (str): url provided by user.
        features_dict (typing.Dict): Ngrams. Key - ngram, value - number.

    Returns:
        np.ndarray: matrix.
    """
    x = np.zeros([1, len(features_dict)], dtype="int")
    url = UrlUtils.clean_url(url)
    for gram in NgramUtils.generate_ngrams_from_string(url):
        try:
            x[0][features_dict[gram]] = x[0][features_dict[gram]] + 1
        except KeyError:
            continue
    return x

@staticmethod
@logger.catch
def clean_url(url: str) -> str:
    """
```

```
    Summary:
        clean url of schema and top level domain

    Args:
        url (str): url.

    Returns:
        str: cleaned url
    """
    url = re.sub(r'https?:\/\/', '', url)
    url = re.sub(r'\.[A-Za-z0-9]+\/*', '', url)
    return url
```

Модуль url_utils.py

```
import socket
from urllib.parse import urlparse
from pydantic import AnyUrl
from .custom_models import URLScheme


def get_url_scheme(url: AnyUrl) -> URLScheme:
    """
    Summary:
        Get url scheme.

    Args:
        url (str): Url.

    Returns:
        str: Scheme.
    """
    if urlparse(str(url)).scheme == "http":
        return URLScheme.HTTP
    return URLScheme.HTTPS

def get_url_domain(url: AnyUrl) -> str:
    """
    Summary:
        Get url domain.

    Args:
        url (str): Url.

    Returns:
```

```
        str: Domain.
    """
    return urlparse(str(url)).netloc


def get_domain_ip(domain: str) -> str:
    """
    Summary:
        Get IP by domain name.

    Args:
        domain (str): domain name.

    Returns:
        str: IP.
    """
    return socket.gethostbyname(domain)
```

Модуль custom_models.py

```
from pydantic import AnyUrl, BaseModel
from typing import Optional
from enum import Enum



class URLScheme(Enum):
    HTTP = 'http'
    HTTPS = 'https'



class WhoIsModel(BaseModel):
    """
    Summary:
        Model for data from whois service.
    """
    valid_response: Optional[bool] = None
    domain_name: Optional[tuple] = None
    creation_date: Optional[tuple] = None
    expiration_date: Optional[tuple] = None
    updated_date: Optional[tuple] = None



class VirusTotalModel(BaseModel):
    valid_response: Optional[bool] = False
    message: Optional[str] = None
    permalink: Optional[str] = None
```

```
    positives: Optional[int] = None
    response_code: Optional[int] = None
    scan_date: Optional[str] = None
    scan_id: Optional[str] = None
    scans: Optional[dict] = None
    total: Optional[int] = None
    url: Optional[str] = None


class BlackListChecker(BaseModel):
    valid_response: Optional[bool] = False
    message: Optional[str] = None
    status: Optional[str] = None
    input_raw: Optional[str] = None
    input_type: Optional[str] = None
    input_domain: Optional[str] = None
    ip_address: Optional[str] = None
    detections: Optional[int] = None
    blacklists: Optional[list] = None
    checks_remaining: Optional[int] = None


class URLSettings(BaseModel):
    """
    Summary:
        Set base model of the URL configuration.
    """
    url: AnyUrl
    status_code: Optional[int] = None
    scheme: Optional[URLScheme] = None
    domain: Optional[str] = None
    ip: Optional[str] = None
    ssl_certificate: Optional[str] = "Unknown"
    whois_data: Optional[WhoIsModel] = None
    virustotal: Optional[VirusTotalModel] = None
    blacklist_checker: Optional[BlackListChecker] = None


class URLsSettings(BaseModel):
    """
    Summary:
        Set base model of the URL configuration.
    """
    urls: tuple[URLSettings, ...]
```

Модуль custom_filters.py

```
import typing
def underscore_to_whitespace(input_text: str) -> str:
    """
    Summary:
        Replace underscore to whitespace.
    Args:
        input (str): input text
    Returns:
        str: output text
    """
    return input_text.replace("_", " ")


def any_to_str(input_list: typing.Any) -> str:
    """
    Summary:
        Convert any to string.
    Args:
        input_list (typing.Any): input
    Returns:
        str: output
    """
    if (isinstance(input_list, list) or isinstance(input_list, tuple)) and len(
        input_list
    ) > 0:
        return str(input_list[0])
    return str(input_list)
```

Модуль visualization_utls.py

```
def get_visualizations_data():
    scheme_x = ['http', "https", "N/A"]
    scheme_y = [49500, 6303, 1133]
    domain_x = [
        '9a327404-a-62cb3a1a-s-sites.googlegroups.com',
        'installer.jdownloader.org',
        'liceulogoga.ro',
        'sites.google.com',
        'ak.imgfarm.com'
    ]
    domain_y = [283, 1013, 1064, 1243, 2760]
    top_level_domain_x = ['ru', 'br', 'org', 'net', 'com']
    top_level_domain_y = [1320, 1320, 2760, 3291, 30952]
    general_x = ['safe', 'malicious']
```

```
    general_y = [991638, 56936]
    return {
        "scheme_x": scheme_x,
        "scheme_y": scheme_y,
        "scheme_title": "Scheme Pie Chart",
        "domain_x": domain_x,
        "domain_y": domain_y,
        "domain_title": "Top Domains",
        "top_level_domain_x": top_level_domain_x,
        "top_level_domain_y": top_level_domain_y,
        "top_level_domain_title": "Top top-level domains",
        "general_x": general_x,
        "general_y": general_y,
        "general_title": "Safe vs Malicious URLs"
    }
```

Файл index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css"/>
    <link rel="stylesheet" href="static/css/utilities.css">
    <link rel="stylesheet" href="static/css/style.css">
    <title>Cyber Analyzer | Top Cybersecurity</title>
</head>
<body>
    <!-- Navbar -->
    <div class="navbar">
        <div class="container flex">
            <h1 class="logo"><a href="/">Cyber Analyzer.</a></h1>
            <nav>
                <ul>
                    <li><a href="/">Home</a></li>
                    <li><a href="docs/">API</a></li>
                </ul>
            </nav>
        </div>
    </div>

    <!-- Showcase -->
    <section class="showcase">
        <div class="container grid">
```

```
        <div class="showcase-text">
            <h1>Reliable security solutions</h1>
            <p>Apply the latest developments in artificial intelligence and cybersecurity to your business to improve protection of your clients</p>
        </div>

        <div class="showcase-form card">
            <h2>Test URL</h2>
             <form action="/" method="POST">
                <div class="form-control">
                    <input type="text" name="url" placeholder="URL" required>
                    <input type="checkbox" name="additional_checks" id="additional_checks">
                    <label for="additional_checks">Additional checks</label>
                </div>
                <input type="submit" value="Test" class="btn btn-primary">
            </form>
            {% if error_message %}
                <p id="url-error">{{ error_message }}</p>
            {% endif %}
        </div>
    </div>
</section>

<!-- Scan info -->
<section class="scan-info">
    {% if checked_url %}
        <div class="scan-info-separator"></div>
        <button class="scan-info-button" onclick="toggleVisibility('scan-list')">Show scan result</button>
        <div class="container grid" id="scan-list">
            <div class="card">
                <h2>ML Info</h2>
                <h3>URL: {{ provided_url }} </h3><br>
                <h3>Result: {{ checked_url }} </h3>
            </div>
            {% if scan_result and scan_result.urls[0] %}

                <!-- General part -->
                <div class="card">
                    <h2>General Info </h2>
                    <h3>URL: {{ scan_result.urls[0].url | default('N/A') }}</h3>
                    {% if scan_result.urls[0].status_code != 404 %}
                        <h3>Website status: Active</h3>
                    {% else %}
                        <h3>Website status: Inactive</h3>
```

```
            {% endif %}
            <h3>Scheme: {{ scan_result.urls[0].scheme.value | default('N/A') }}</h3>
            <h3>Domain: {{ scan_result.urls[0].domain | default('N/A') }}</h3>
            <h3>IP: {{ scan_result.urls[0].ip | default('N/A') }} </h3>
            <h3>SSL Certificate: {{ scan_result.urls[0].ssl_certificate | default('N/A') }}</h3>
        </div>

        <!-- VirusTotal part -->
        <div class="card">
            <h2>VirusTotal Info </h2>
            <h3>Message: {{ scan_result.urls[0].virustotal.message | default('N/A') }}</h3>
            <h3>
                Permalink:
                {% if scan_result.urls[0].virustotal.permalink %}
                    <a href="{{ scan_result.urls[0].virustotal.permalink }}">{{ scan_result.urls[0].virustotal.permalink
}}</a>
                {% else %}
                    N/A
                {% endif %}
            </h3>
            <h3>Positives: {{ scan_result.urls[0].virustotal.positives | default('N/A') }}</h3>
            <h3>Response code: {{ scan_result.urls[0].virustotal.response_code | default('N/A') }}</h3>
            <h3>Scan date: {{ scan_result.urls[0].virustotal.scan_date | default('N/A') }}</h3>
            <h3>Scan id: {{ scan_result.urls[0].virustotal.scan_id | default('N/A') }}</h3>
            <h3>Total services: {{ scan_result.urls[0].virustotal.total | default('N/A') }}</h3>
            <h3>URL: {{ scan_result.urls[0].virustotal.url | default('N/A') }}</h3>
        </div>

        <!-- ApiBlackListChecker part -->
        <div class="card">
            <h2>Blacklist checker info </h2>
            <h3>Message: {{ scan_result.urls[0].blacklist_checker.message | default('N/A') }}</h3>
            <h3>Status: {{ scan_result.urls[0].blacklist_checker.status | default('N/A') }}</h3>
            <h3>Input: {{ scan_result.urls[0].blacklist_checker.input_raw | default('N/A') }}</h3>
            <h3>Input type: {{ scan_result.urls[0].blacklist_checker.input_type | default('N/A') }}</h3>
            <h3>Input domain: {{ scan_result.urls[0].blacklist_checker.input_domain | default('N/A') }}</h3>
            <h3>IP address: {{ scan_result.urls[0].blacklist_checker.ip_address | default('N/A') }}</h3>
            <h3>Detections: {{ scan_result.urls[0].blacklist_checker.detections | default('N/A') }}</h3>
            <h3>Checks remaining: {{ scan_result.urls[0].blacklist_checker.checks_remaining | default('N/A')
}}</h3>
        </div>

        <!-- Whois part -->
        <div class="card">
```

```
                <h2>WHOIS Info</h2>
                <h3>Domain name: {{ scan_result.urls[0].whois_data.domain_name | default('N/A') | any_to_str
}}</h3>
                <h3>Creation date: {{ scan_result.urls[0].whois_data.creation_date | default('N/A') | any_to_str }}</h3>
                <h3>Expiration date: {{ scan_result.urls[0].whois_data.expiration_date | default('N/A') | any_to_str
}}</h3>
                <h3>Updated date: {{ scan_result.urls[0].whois_data.updated_date | default('N/A') | any_to_str }}</h3>
              </div>
            {% endif %}
          </div>
      {% endif %}
    </section>

    <!-- Intro -->
    <section class="intro">
      <div class="container">
        <h3 class="intro-heading text-center my-1">
          Welcome, the latest developments in cybersecurity at your service
        </h3>

        <div class="grid grid-3 text-center my-4">
          <div>
            <i class="fas fa-shield-alt fa-3x"></i>
            <h3>More than 1M analyzed entries</h3>
            <p class="text-secondary">Verified data</p>
          </div>
          <div>
            <i class="fas fa-hammer fa-3x"></i>
            <h3>Anything/Anytime</h3>
            <p class="text-secondary">Custom solutions</p>
          </div>
          <div>
            <i class="fas fa-plane fa-3x"></i>
            <h3>57 ms per single request</h3>
            <p class="text-secondary">Fast</p>
          </div>
          <div>
            <i class="fas fa-money-check-alt fa-3x"></i>
            <h3>--/--</h3>
            <p class="text-secondary">Affordable</p>
          </div>
          <div>
            <i class="fas fa-mountain fa-3x"></i>
            <h3>24/7</h3>
```

```
            <p class="text-secondary">Reliable</p>
          </div>
          <div>
            <i class="fas fa-mask fa-3x"></i>
            <h3>No personal info.</h3>
            <p class="text-secondary">Anonymous</p>
          </div>
        </div>
    </div>
</section>

<!-- Undermain 1 -->
<section class="undermain bg-primary my-2 py-2">
    <div class="container grid">
        <div class="text-center">
            <h2 class="lg">Simple and convenient security solutions</h2>
            <p class="lead my-1">Analyze any data from any source. Fast, efficient and anonymous</p>
        </div>
        <div class="big-icon">
            <i class="fas fa-check-square fa-8x"></i>
        </div>
    </div>
</section>

<!-- cards -->
<section class="cards">
    <div class="container grid">
        <div class="card">
            <h3>Easy to use, simple forms</h3>
        </div>
        <div class="card">
            <h3>Practical API</h3>
        </div>
        <div class="card">
            <h3>Free statistics and reports</h3>
        </div>
        <div class="card">
            <h3>Free and paid solutions</h3>
        </div>
        <div class="card">
            <h3>Any format</h3>
        </div>
        <div class="card">
            <h3>No data limitation</h3>
```

```
        </div>
      </div>
    </section>

    <!-- Undermain 2 -->
    <section class="undermain bg-primary my-2 py-2">
      <div class="container grid">
        <div class="text-center">
          <h2 class="lg">Statistics and reports </h2>
          <p class="lead my-1">Detailed analysis and visualization of processed data and reports with advices for system improvements.</p>
        </div>
        <div class="big-icon">
          <i class="fas fa-chart-area fa-8x"></i>
        </div>
      </div>
    </section>

    <!-- Visualizations -->
    <section class="cards">
      <div class="container pies">
        <div class="visualiztion-item">
          <canvas id="schemeVis" style="width:100%;max-width:700px"></canvas>
        </div>
        <div class="visualiztion-item">
          <canvas id="generalVis" style="width:100%;max-width:700px"></canvas>
        </div>
        <div class="visualiztion-item">
          <canvas id="domainVis" style="width:100%;max-width:700px"></canvas>
        </div>
        <div class="visualiztion-item">
          <canvas id="topLevelDomainVis" style="width:100%;max-width:700px"></canvas>
        </div>
      </div>
    </section>

    <!-- Undermain 3 -->
    <section class="undermain bg-primary my-2 py-2">
      <div class="container grid">
        <div class="text-center">
          <h2 class="lg">Vast amount of supported sources.</h2>
          <p class="lead my-1">All official sources have implemented solutions, for unofficial - custom solution could be created.</p>
        </div>
```

```
        <div class="big-icon">
            <i class="fas fa-cogs fa-8x"></i>
        </div>
    </div>
</section>

<!-- Sources -->
<section class="sources">
    <h2 class="md text-center my-2">
        Supported Sources
    </h2>
    <div class="container flex">
        <div class="card">
            <h4>URLs</h4>
            <i class="fas fa-search-minus fa-3x"></i>
        </div>
        <div class="card">
            <h4>Firewalls</h4>
            <i class="fas fa-fire fa-3x"></i>
        </div>
        <div class="card">
            <h4>Unix</h4>
            <i class="fas fa-sitemap fa-3x"></i>
        </div>
        <div class="card">
            <h4>Windows</h4>
            <i class="fas fa-border-all fa-3x"></i>
        </div>
        <div class="card">
            <h4>CIEM solutions</h4>
            <i class="fas fa-cloud-meatball fa-3x"></i>
        </div>
        <div class="card">
            <h4>WLC</h4>
            <i class="fas fa-wifi fa-3x"></i>
        </div>
        <div class="card">
            <h4>Others</h4>
            <i class="fas fa-spinner fa-3x"></i>
        </div>
    </div>
</section>

<!-- Footer -->
```

```
<footer class="footer bg-dark py-5">
    <div class="container grid grid-3">
        <div>
            <h1>Cyber Analyzer
            </h1>
            <p>Copyright &copy; 2024</p>
        </div>
        <nav>
            <ul>
                <li><a href="/">Home</a></li>
                <li><a href="docs/">API</a></li>
            </ul>
        </nav>
    </div>
</footer>

<div class="backend-values" style="display: none;">
    <p id="scheme_x">{{ scheme_x }}</p>
    <p id="scheme_y">{{ scheme_y }}</p>
    <p id="scheme_title">{{ scheme_title }}</p>
    <p id="domain_x">{{ domain_x }}</p>
    <p id="domain_y">{{ domain_y }}</p>
    <p id="domain_title">{{ domain_title }}</p>
    <p id="top_level_domain_x">{{ top_level_domain_x }}</p>
    <p id="top_level_domain_y">{{ top_level_domain_y }}</p>
    <p id="top_level_domain_title">{{ top_level_domain_title }}</p>
    <p id="general_x">{{ general_x }}</p>
    <p id="general_y">{{ general_y }}</p>
    <p id="general_title">{{ general_title }}</p>
    <p id="scan_result_exist">{{ scan_result_exist }}</p>
</div>

<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></script>
<script>
    function setURLStatus() {
        var urlStatusElem = document.getElementById('url-error')
        if (urlStatusElem !== null) {
            urlStatusElem.className += "url-invalid"
        }
    }

    function strToArray(string) {
        string = string.textContent.replace(/'/g, '"')
        string = JSON.parse(string)
```

```
    return string
}

function createVisualization(name, title, labels, data, type) {
    var barColors = [
        "#b91d47",
        "#00aba9",
        "#2b5797",
        "#e8c3b9",
        "#1e7145"
    ];

    new Chart(name, {
        type: type,
        data: {
            labels: labels,
            datasets: [{
                backgroundColor: barColors,
                data: data
            }]
        },
        options: {
            title: {
                display: true,
                text: title
            }
        }
    });
}

function toggleVisibility(id) {
    var element = document.getElementById(id);
    element.classList.toggle('hidden');
    if (element.classList.contains("hidden")) {
        document.querySelector('.scan-info-button').textContent = 'Show scan result';
    } else {
        document.querySelector('.scan-info-button').textContent = 'Hide scan result';
    }
}

function changeIntroPadding() {
    var scanResultExist = document.getElementById('scan_result_exist')
    if (scanResultExist !== null && scanResultExist.textContent === 'True') {
        document.querySelector('.intro').style.paddingTop = '40px';
```

```
        }
    }

    var schemeX = document.getElementById('scheme_x')
    var schemeY = document.getElementById('scheme_y')
    var domainX = document.getElementById('domain_x')
    var domainY = document.getElementById('domain_y')
    var topLevelDomainX = document.getElementById('top_level_domain_x')
    var topLevelDomainY = document.getElementById('top_level_domain_y')
    var generalX = document.getElementById('general_x')
    var generalY = document.getElementById('general_y')
    var schemeTitle = document.getElementById('scheme_title')
    var domainTitle = document.getElementById('domain_title')
    var topLevelDomainTitle = document.getElementById('top_level_domain_title')
    var generalTitle = document.getElementById('general_title')
    schemeTitle = schemeTitle.textContent
    domainTitle = domainTitle.textContent
    topLevelDomainTitle = topLevelDomainTitle.textContent
    generalTitle = generalTitle.textContent

    setURLStatus()
    changeIntroPadding()

    schemeX = strToArray(schemeX)
    schemeY = strToArray(schemeY)
    domainX = strToArray(domainX)
    domainY = strToArray(domainY)
    topLevelDomainX = strToArray(topLevelDomainX)
    topLevelDomainY = strToArray(topLevelDomainY)
    generalX = strToArray(generalX)
    generalY = strToArray(generalY)

    createVisualization("schemeVis", schemeTitle, schemeX, schemeY, "pie")
    createVisualization("domainVis", domainTitle, domainX, domainY, "polarArea")
    createVisualization("topLevelDomainVis", topLevelDomainTitle, topLevelDomainX, topLevelDomainY,
"doughnut")
    createVisualization("generalVis", generalTitle, generalX, generalY, "pie")
  </script>
</body>
</html>
```

Файл style.css

```
@import url('https://fonts.googleapis.com/css2?family=Lato:wght@300&display=swap');
```

```
:root {
  --primary-color: #33875a;
  --secondary-color: #1c3fa8;
  --dark-color: #60ac59;
  --light-color: #f4f4f4;
  --success-color: #5cb85c;
  --error-color: #d9534f;
}

* {
  box-sizing: border-box;
  padding: 0;
  margin: 0;
}

body {
  font-family: 'Lato', sans-serif;
  color: #333;
  line-height: 1.6;
}

ul {
  list-style-type: none;
}

a {
  text-decoration: none;
  color: #000000;
}

.card a:hover {
  color: var(--primary-color);
  text-decoration: underline;
}

h1,
h2 {
  font-weight: 300;
  line-height: 1.2;
  margin: 10px 0;
}

p {
  margin: 10px 0;
```

```
}

img {
  width: 100%;
}

/* Navbar */
.navbar {
  background-color: var(--primary-color);
  color: #fff;
  height: 70px;
}

.navbar ul {
  display: flex;
}

.navbar a {
  color: #fff;
  padding: 10px;
  margin: 0 5px;
}

.navbar ul a:hover {
  border-bottom: 2px #fff solid;
}

.navbar .flex {
  justify-content: space-between;
}

/* Showcase */
.showcase {
  height: 400px;
  background-color: var(--primary-color);
  color: #fff;
  position: relative;
}

.showcase h1 {
  font-size: 40px;
}

.showcase p {
```

```
  margin: 20px 0;
}

.showcase .grid {
  overflow: visible;
  grid-template-columns: 55% auto;
  gap: 30px;
}

.showcase-text {
  animation: slideInFromLeft 1s ease-in;
}

.showcase-form {
  position: relative;
  top: 60px;
  height: 260px;
  width: 400px;
  padding: 40px;
  z-index: 100;
  justify-self: flex-end;
  animation: slideInFromRight 1s ease-in;
}

.showcase-form .form-control {
  margin: 15px 0;
}

.showcase-form input[type='text'],
.showcase-form input[type='email'] {
  border: 0;
  border-bottom: 1px solid #b4becb;
  width: 100%;
  padding: 3px;
  font-size: 16px;
}

.showcase-form input:focus {
  outline: none;
}

.showcase::before {
  content: '';
  position: absolute;
```

```
  height: 100px;
  bottom: -70px;
  right: 0;
  left: 0;
  background: #fff;
  transform: skewY(2deg);
  -webkit-transform: skewY(2deg);
  -moz-transform: skewY(2deg);
  -ms-transform: skewY(2deg);
}
.showcase::after {
  content: '';
  position: absolute;
  height: 100px;
  bottom: -70px;
  right: 0;
  left: 0;
  background: #fff;
  transform: skewY(358  deg);
  -webkit-transform: skewY(358deg);
  -moz-transform: skewY(358deg);
  -ms-transform: skewY(358deg);
}

#url-error {
  margin: 0;
  font-weight: bold;
  width: fit-content;
}

.url-invalid {
  color: #FF0000;
}

.url-safe {
  color: #33875a;
}

/* Scan info */
.scan-info-separator {
  padding-top: 115px;
}

.scan-info-footer {
```

```
  height: 50px;
  width: 100%;
  background-color: var(--primary-color);
}

.hidden {
  max-height: 0;
  padding: 0;
  margin-bottom: 0px;
}

#scan-list {
  transition: max-height 0.3s ease-in-out, padding 0.3s ease-in-out;
}

.scan-info-button {
  width: 100%;
  margin-bottom: 20px;
  display: inline-block;
  padding: 15px 25px;
  font-size: 24px;
  cursor: pointer;
  text-align: center;
  text-decoration: none;
  outline: none;
  color: #fff;
  background-color: var(--primary-color);
  border: none;
  border-radius: 0px;
  box-shadow: 0 10px 10px #999;;
}

.scan-info-button:hover {background-color: #3eb073}

.scan-info-button:active {
  background-color: #33875a;
  box-shadow: 0 5px #666;
  transform: translateY(4px);
}

/* Intro */
.intro {
  padding-top: 100px;
  animation: slideInFromBottom 1s ease-in;
```

```
}

.intro-heading {
  max-width: 500px;
  margin: auto;
}

.intro .grid h3 {
  font-size: 35px;
}

.intro .grid p {
  font-size: 20px;
  font-weight: bold;
}

/* cards */
.cards .grid {
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(2, 1fr);
}

.cards .grid > *:first-child {
  grid-column: 1 / span 1;
  grid-row: 1 / span 1;
}

/* Undermain */
.undermain .grid {
  grid-template-columns: 4fr 3fr;
}

/* sources */

.sources .flex {
  flex-wrap: wrap;
}

.sources .card {
  text-align: center;
  margin: 18px 10px 40px;
  transition: transform 0.2s ease-in;
}
```

```
.sources .card h4 {
  font-size: 20px;
  margin-bottom: 10px;
}

.sources .card:hover {
  transform: translateY(-15px);
}

/* Footer */

.big-icon {
  padding: 0px 10em;
}

.visualiztion-item {
  width: 100%;
}

/* Animations */
@keyframes slideInFromLeft {
  0% {
    transform: translateX(-100%);
  }

  100% {
    transform: translateX(0);
  }
}

@keyframes slideInFromRight {
  0% {
    transform: translateX(100%);
  }

  100% {
    transform: translateX(0);
  }
}

@keyframes slideInFromTop {
  0% {
    transform: translateY(-100%);
  }
```

```
  100% {
    transform: translateX(0);
  }
}

@keyframes slideInFromBottom {
  0% {
    transform: translateY(100%);
  }

  100% {
    transform: translateX(0);
  }
}
@media (max-width: 1000px){

  .showcase-form{
    width: 340px;
  }

  .pies {
    grid-template-columns: repeat(1, 1fr);
  }
}

/* Tablets and under */
@media (max-width: 768px) {
  .grid,
  .showcase .grid,
  .intro .grid,
  .undermain .grid {
    grid-template-columns: 1fr;
    grid-template-rows: 1fr;
  }

  .showcase {
    height: auto;
  }

  .showcase-text {
    text-align: center;
    margin-top: 40px;
    animation: slideInFromTop 1s ease-in;
```

```
  }

  .showcase-form {
    justify-self: center;
    margin: auto;
    animation: slideInFromBottom 1s ease-in;
  }

  .big-icon {
    padding: 0 5em;
  }
}


/* Mobile */
@media (max-width: 500px) {
  .cards .grid {
    grid-template-columns: 1fr;
    grid-template-rows: 1fr;
  }

  .navbar {
    height: 110px;
  }

  .navbar .flex {
    flex-direction: column;
  }

  .navbar ul {
    padding: 10px;
    background-color: rgba(0, 0, 0, 0.1);
  }

  .showcase-form {
    width: 300px;
  }

  .container {
    padding: 0px;
  }

}
```

Файл utilities.css

```
.container {
  max-width: 1100px;
  margin: 0 auto;
  overflow: auto;
  padding: 0 40px;
}

.card {
  background-color: #fff;
  color: #333;
  border-radius: 10px;
  box-shadow: 0 3px 10px rgba(0, 0, 0, 0.2);
  padding: 20px;
  margin: 10px;
  overflow-wrap: break-word;
  word-wrap: break-word;
  word-break: break-word;
  -ms-hyphens: auto;
  -moz-hyphens: auto;
  -webkit-hyphens: auto;
  -ms-word-break: break-all;
  hyphens: auto;
}

.btn {
  margin: 10px 0;
  display: inline-block;
  padding: 10px 30px;
  cursor: pointer;
  background: var(--primary-color);
  color: #fff;
  border: none;
  border-radius: 5px;
  box-shadow: 0 3px 10px rgba(0, 0, 0, 0.2);
}

.btn-outline {
  background-color: transparent;
  border: 1px #fff solid;
}

.btn:hover {
```

```
  transform: scale(0.98);
}

/* Backgrounds & colored buttons */
.bg-primary,
.btn-primary {
  background-color: var(--primary-color);
  color: #fff;
}

.bg-dark {
  background-color: var(--dark-color);
  color: #fff;
}

.bg-primary a,
.btn-primary a,
.bg-dark a {
  color: #fff;
}

.text-secondary {
  color: var(--secondary-color);
}

/* Text sizes */
.lead {
  font-size: 20px;
}

.sm {
  font-size: 1rem;
}

.md {
  font-size: 2rem;
}

.lg {
  font-size: 3rem;
}

.xl {
  font-size: 4rem;
```

```
}

.text-center {
  text-align: center;
}

.flex {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100%;
}

.grid {
  display: grid;
  grid-template-columns: repeat(2, 1fr);
  gap: 20px;
  justify-content: center;
  align-items: center;
  height: 100%;
}

.pies {
  display: grid;
  grid-template-columns: repeat(2, 1fr);
  gap: 20px;
  justify-content: center;
  align-items: center;
  height: 100%;
}

.grid-3 {
  grid-template-columns: repeat(3, 1fr);
}

/* Margin */
.my-1 {
  margin: 1rem 0;
}

.my-2 {
  margin: 1.5rem 0;
}
```

```
.my-3 {
  margin: 2rem 0;
}

.my-4 {
  margin: 3rem 0;
}

.my-5 {
  margin: 4rem 0;
}

.m-1 {
  margin: 1rem;
}

.m-2 {
  margin: 1.5rem;
}

.m-3 {
  margin: 2rem;
}

.m-4 {
  margin: 3rem;
}

.m-5 {
  margin: 4rem;
}

/* Padding */
.py-1 {
  padding: 1rem 0;
}

.py-2 {
  padding: 1.5rem 0;
}

.py-3 {
  padding: 2rem 0;
}
```

```
.py-4 {
  padding: 3rem 0;
}

.py-5 {
  padding: 4rem 0;
}

.p-1 {
  padding: 1rem;
}

.p-2 {
  padding: 1.5rem;
}

.p-3 {
  padding: 2rem;
}

.p-4 {
  padding: 3rem;
}

.p-5 {
  padding: 4rem;
}
```

Файл .gitignore

```
__pycache__/
.env
env/
```