КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)

Інформаційних технологій проєктування та прикладної математики
(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему: «Методи виявлення та нейтралізації загроз для безпеки даних в системах електронної комерції»

Хімченко Олексій Сергійович
(прізвище, ім’я та по батькові магістра повністю)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)

Інформаційних технологій проєктування та прикладної математики
(кафедра)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТППМ
д.т.н., професор Бородавка Є.В.
«____» _____ 2025 року

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему: «Методи виявлення та нейтралізації загроз для безпеки даних в системах електронної комерції»

Виконав: студент 2-го курсу, групи ІСТм-24
Спеціальності: 126 «Інформаційні системи і технології»
(шифр і назва напряму підготовки, спеціальності)

Магістрант: Хімченко О. С.
(прізвище та ініціали)

Керівник: д.т.н., проф. Бородавка Є. В.
(прізвище та ініціали)

Рецензент: д-р, філос. Рябчун Ю. В.
(прізвище та ініціали)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: Автоматизації та інформаційних технологій
Кафедра: Інформаційних технологій проєктування та ПМ
Освітній ступінь: «магістр з інформаційних систем і технологій»
Спеціальність: 126 «Інформаційні системи і технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТППМ
д.т.н., професор Бородавка Є. В.
«____» _____ 2025 року

ЗАВДАННЯ
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

Хімченко Олексій Сергійович

1. Тема роботи: «Методи виявлення та нейтралізації загроз для безпеки даних в системах електронної комерції»
затверджена наказом ректора КНУБА № 1619/23/25 від «29» вересня 2025 року
2. Керівник роботи: Бородавка Євгеній Володимирович, д.т.н., завідувач кафедри інформаційних технологій проєктування і прикладної математики
3. Строк подання студентом роботи до захисту: 18.12.2025
4. Зміст пояснювальної записки розділами:
Р.1. Аналіз предметної області та постановка задачі
Р.2. Основи загроз та методи їх виявлення
Р.3 Проєктування архітектури системи виявлення та нейтралізації загроз
Р.4. Програмна реалізація системи
Р.5. Ергономічний аналіз
5. Інформаційні слайди:
С.1. Титульна сторінка
С.2. Вступ
С.3. Аналіз предметної області
С.4. Аналіз предметної області

С.5. Аналіз предметної області
С.6. Види даних та вимоги до безпеки
С.7. Класифікація загроз та методи їх виявлення
С.8. Практична реалізація
С.9. Збереження даних користувача
С.10 Процес замовлення товару і додавання до БД
С.11 Перевірка безпеки
С.12 Реалізація безпеки від DDoS атаки
С.13 Панель безпеки адміна
С.14 Висновок

6. Календарний план виконання кваліфікаційної випускної роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	Вересень 2025
Розділ 2	Жовтень 2025
Розділ 3	Жовтень 2025
Розділ 4	Листопад 2025
Розділ 5	Листопад 2025
Остаточне оформлення роботи	Грудень 2025
Направлення роботи для перевірки на плагіат	Грудень 2025
Попередній захист роботи на випусковій кафедрі	Грудень 2025
Направлення роботи на рецензування	Грудень 2025

7. Консультанти розділів кваліфікаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис

8. Дата видачі завдання: 29.09.2025

Керівник _____ Бородавка Є. В.
(підпис) (прізвище та ініціали)

Магістрант _____ Хімченко О. С.
(підпис) (прізвище та ініціали)

РЕЗЮМЕ

Київський національний університет будівництва і архітектури

ХІМЧЕНКО ОЛЕКСІЙ СЕРГІЙОВИЧ

факультет автоматизації і інформаційних технологій,

група ІСТм-24

Тема кваліфікаційної випускної роботи: «Методи виявлення та нейтралізації загроз для безпеки даних в системах електронної комерції»

освітній рівень: магістр,

спеціальність: 126 «Інформаційні системи і технології»,

Науковий керівник: Бородавка Євгеній Володимирович,

доктор технічних наук, професор, завідувач кафедри інформаційних технологій проєктування та прикладної математики

Обсяг роботи. Кваліфікаційна випускна робота магістра складається з: 5 розділів, 140 сторінок, 42 рисунків, 4 таблиць, завдання, анотації, вступу, висновків, списку використаних джерел та 2 додатків.

Актуальність теми. Робота присвячена розробці захищеної веб-орієнтованої системи обробки запитів для сфери електронної комерції. Актуальність теми зумовлена стрімким зростанням кількості кібератак на онлайн-ресурси, зокрема DDoS-атак та спроб несанкціонованого доступу, що призводить до значних фінансових втрат бізнесу. Існуючі рішення часто є дороговартісними або складними в інтеграції для малого та середнього бізнесу. У відповідь на потребу в доступному та ефективному захисті, розробка спеціалізованої системи з інтегрованими механізмами моніторингу загроз та обмеження навантаження є доречною та своєчасною.

У вступі обґрунтовано актуальність теми, сформульовано мету та основні завдання роботи, визначено об’єкт і предмет дослідження, а також окреслено наукову новизну та практичну значущість отриманих результатів.

У першому розділі «Аналіз предметної області та постановка задачі» проведено комплексне дослідження сфери електронної комерції як об’єкта

захисту. Здійснено класифікацію інформаційних активів, що циркулюють у веб-системах, зокрема персональних, платіжних та комерційних даних, а також визначено рівні їх критичності. Сформульовано базові вимоги до безпеки системи на основі тріади CIA (конфіденційність, цілісність, доступність) та додаткових властивостей, таких як автентичність і невідмовність. Проаналізовано стан нормативного регулювання та стандартизації в галузі кібербезпеки. На основі огляду існуючих рішень та виявлених недоліків обґрунтовано актуальність розробки власної системи захисту та сформульовано чітку постановку задачі дослідження.

У другому розділі «Основи загроз та методи їх виявлення» викладено теоретичний базис побудови систем захисту. Розроблено детальну класифікацію загроз для e-commerce платформ, охоплюючи атаки на веб-додатки (SQL Injection, XSS), мережеву інфраструктуру (DDoS) та користувачів (соціальна інженерія). Проведено порівняльний аналіз методів детекції загроз, зокрема сигнатурного аналізу, методів виявлення аномалій та підходів на базі машинного навчання. Розглянуто архітектурні компоненти захисту (WAF, SIEM, IDS/IPS) та стратегії реагування на інциденти. Окрему увагу приділено математичному моделюванню процесів захисту: формалізовано поняття сукупного ризику та наведено модель оцінки впливу загроз на ключові параметри інформаційної безпеки.

У третьому розділі «Проєктування архітектури системи» розроблено загальну структуру клієнт-серверної системи на базі патерну MVC. Спроєктовано схему реляційної бази даних для зберігання інформації про користувачів, товари та замовлення, а також обґрунтовано вибір технологічного стека (Python, Flask, SQLAlchemy) для реалізації серверної логіки.

У четвертому розділі «Програмна реалізація та тестування системи» викладено етапи розробки програмних модулів, зокрема підсистеми моніторингу загроз ThreatMonitor та механізму Middleware для фільтрації запитів. Наведено результати функціонального та навантажувального тестування, що підтверджують стійкість системи до автоматизованих атак та коректність обробки бізнес-процесів.

У п’ятому розділі «Ергономічний аналіз» проведено оцінку користувацького інтерфейсу на відповідність міжнародним стандартам юзабіліті (ISO 9241-110). Проаналізовано психофізіологічний вплив обраної колірної гами та типографіки, обґрунтовано зручність панелі адміністрування для оперативного реагування на інциденти безпеки.

Ключові слова: електронна комерція, безпека даних, загроза, кібербезпека, автентифікація, веб-додаток.

Keywords: e-commerce, data security, threat, cybersecurity, authentication, web application.

Якість оформлення проєкту. Кваліфікаційна випускна робота магістра оформлена у відповідності до діючих нормативних документів та методичних вказівок до виконання дипломних робіт для студентів спеціальності 126 «Інформаційні системи і технології».

Загальний висновок стосовно роботи та присвоєння авторові освітньо-кваліфікаційного рівня «магістр». Робота виконана на високому рівні, студент продемонстрував високий рівень теоретичної підготовки та сформованих практичних навичок в області сучасних інформаційних технологій. Заслуговує оцінку «відмінно».

Науковий керівник: _____ Бородавка Є. В.
(підпис) (прізвище та ініціали)

Посада, місце роботи: _____

«_____» _____ 2025 р.

АНОТАЦІЯ

Хімченко О. С. «Методи виявлення та нейтралізації загроз для безпеки даних в системах електронної комерції».

Кваліфікаційна випускна робота магістра за спеціальністю: 126 «Інформаційні системи і технології». – Київський національний університет будівництва і архітектури. – Київ, 2025.

Робота присвячена дослідженню методів виявлення та нейтралізації загроз для безпеки в системах електронної комерції. Особливу увагу приділено дослідженню видів даних електронної комерції, можливих видів загроз для них і методів для їх пошуку та запобігання.

Ключові слова: електронна комерція, безпека даних, загроза, кібербезпека, автентифікація, веб-додаток.

SUMMARY

Khimchenko O. S. "Methods for detecting and neutralising threats to data security in e-commerce systems."

Master's degree final thesis in the specialty: 126 "Information Systems and Technologies." – Kyiv National University of Construction and Architecture. – Kyiv, 2025.

The work is devoted to the study of methods for detecting and neutralising threats to security in e-commerce systems. Particular attention is paid to the study of types of e-commerce data, possible types of threats to them, and methods for detecting and preventing them.

Keywords: e-commerce, data security, threat, cybersecurity, authentication, web application.

Зміст

Вступ

Розвиток інформаційних технологій та цифровізація економіки сприяли стрімкому зростанню електронної комерції, яка стала невід'ємною частиною повсякденного життя мільйонів користувачів. Сьогодні онлайн-покупки здійснюють люди різних вікових категорій і соціальних статусів, а обсяги цифрової торгівлі постійно зростають, охоплюючи як традиційний ритейл, так і нові моделі: маркетплейси, платформи обслуговування, мобільні додатки та інші канали.

Актуальність теми роботи зумовлена тим, що разом з розширенням електронної комерції, експоненціально зростають масштаби та складність загроз інформаційній безпеці. Системи е-комерції обробляють й акумулюють величезні обсяги критично важливих даних: персональну інформацію користувачів, платіжні реквізити, комерційні таємниці та історії транзакцій. Така концентрація чутливої інформації робить ці платформи привабливою мішенню для кіберзлочинців, хакерів та інших загроз.

На сьогодні продовжується постійне зростання кількості успішних кібератак на системи електронної комерції, що призводять до витоків даних, фінансових збитків для компаній та втрати довіри споживачів до цифрових каналів торгівлі. При цьому традиційні методи захисту даних, хоча й ефективні, часто виявляються неспроможними протидіяти новітнім, складно організованим атакам, які постійно вдосконалюються та адаптуються до існуючих механізмів протидії.

Мета роботи полягає у розробці та обґрунтуванні комплексу методів виявлення та нейтралізації загроз для безпеки даних у системах електронної комерції для автоматизації та спрощення оцінки стану безпеки, а також забезпечення швидкого та своєчасного реагування на виявлені загрози.

Об'єктом дослідження є дані у системах е-комерції, загрози безпеці та процеси їх виявлення та нейтралізації, що охоплюють автентифікацію користувачів, управління доступом, моніторинг операцій та реагування на інциденти.

Предметом дослідження є методи та алгоритми виявлення загроз, модельні підходи до класифікації атак та аномалій, а також механізми криптографічного захисту даних та управління аутентифікацією в e-commerce системах.

Методи дослідження. Для досягнення поставленої мети проведений аналіз існуючих підходів до кібербезпеки, розроблення алгоритмів виявлення загроз та проєктування архітектури інформаційної системи. Основні проєктні рішення передбачають розроблення гібридної системи, що поєднує правило-орієнтовані методи виявлення з аналізом аномалій у поведінці користувачів та операцій.

Отже, дане магістерське дослідження спрямоване на створення практичної інформаційної системи для виявлення та нейтралізації загроз безпеці даних у платформах електронної комерції, що забезпечить надійний захист персональної та фінансової інформації користувачів, сприятиме мінімізації ризиків для компаній та підвищить загальний рівень довіри до цифрових каналів торгівлі.

Розділ 1. Аналіз предметної області та постановка задачі

1.1 Опис предметної області

Дана освітньо-кваліфікаційна робота присвячена дослідженню загроз безпеці даних в системах електронній комерції і створенню інформаційної системи з використанням методів їх виявлення та нейтралізації. Основною метою проєкту є покращення та прискорення оцінки стану безпеки та реагування на виявлені загрози.

Предметною областю цієї роботи є електронна комерція (e-commerce) – галузь економіки, в якій реклама, просування продуктів, торговельні угоди та фінансові транзакції здійснюються безпосередньо в Інтернеті. Коли ви щось купуєте чи продаєте у мережі, це і є e-commerce.

З погляду виробників та постачальників електронна комерція – це просування та надання своїх товарів чи послуг через Інтернет. А з погляду покупців (клієнтів) це перегляд торгових пропозицій, вибір, замовлення та оплата прямо в мережі.

Розгляньмо найпоширеніші на сьогодні приклади електронної комерції:

- **роздрібна та гуртова торгівля**. Один з головних напрямків e-Commerce. Прикладів таких платформ існує безліч: Amazon, Aliexpress, Etsy та багато інших (рис. 1.1).

Рисунок 1.1 – Принцип роздрібної та гуртової торгівлі

- **Дропшипінг**. Продаж продукту від стороннього виробника чи постачальника. Дропшипінгом часто займаються невеликі онлайн-магазини та індивідуальні підприємці. Типовий приклад – продаж товарів з Китаю на Prom.ua чи OLX (рис. 1.2).

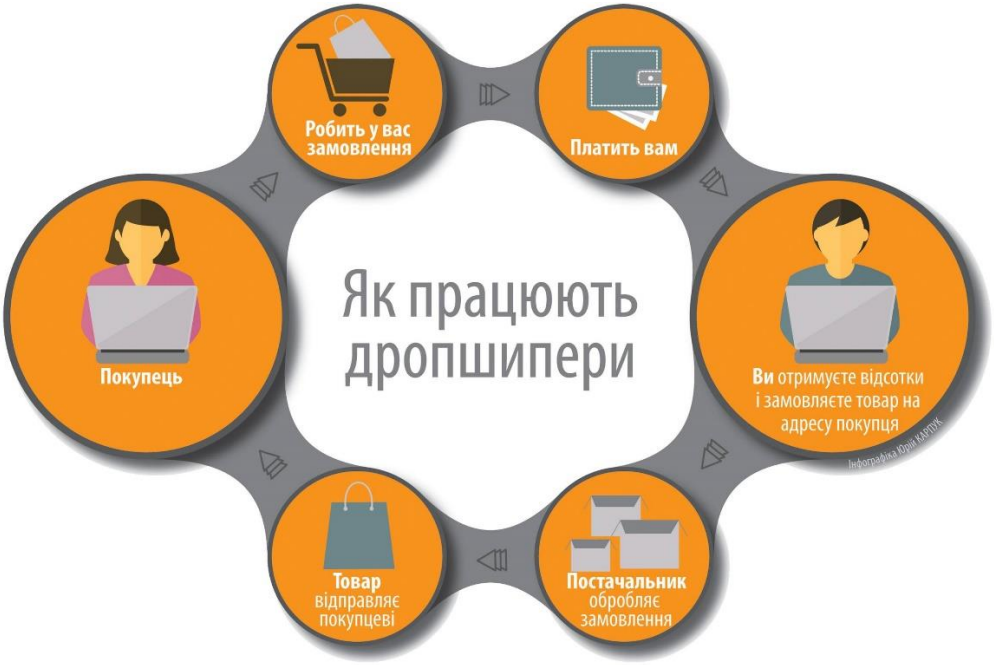


Рисунок 1.2 – Принцип роботи дропшипінгу

- **Краудфандинг**. Збір коштів зі споживачів з метою отримання стартового капіталу, щоб вивести продукцію на ринок (рис. 1.3). Як приклад, з краундфандигу почалась історія VR-гарнітури Oculus Rift, яка сьогодні належить холдингу Meta й відома як Meta Quest.

Рисунок 1.3 – Принцип краудфандингу

- **Цифрові продукти**. Наприклад, хмарні платформи для менеджменту, навчання, комунікації тощо (від Duolingo до Slack). Вони часто поширюються за передплатною моделлю, чи пропонують платний функціонал (рис. 1.4).

Рисунок 1.4 – Принцип роботи цифрових продуктів

- **Послуги онлайн**. Весь спектр сервісів, які можна запропонувати потенційним клієнтам в мережі. Від Uber до замовлення піци через додаток найближчого до вас ресторану (рис. 1.5).

Рисунок 1.5 – Принцип роботи сервісу онлайн послуг

E-commerce розділяють на шість основних видів залежно від взаємодії сторін: клієнтів, бізнесу та адміністрацій. Видами та типами електронної комерції є:

- **B2B** – електронна комерція для бізнесу. Це різновид електронної комерції, який працює за принципом "від бізнесу - бізнесу" (Business-to-Business). Тобто, за допомогою діджиталу одна компанія надає свої послуги або товари іншій компанії, а не роздрібному споживачеві.
- **B2C** – електронна комерція для споживача. Наразі модель Business-to-Consumer (B2C) – це найпоширеніший вид електронної комерції. В такій моделі компанія надає свої товари, послуги та сервіси безпосередньо роздрібному споживачеві. Більшість популярних онлайн-ритейлерів (від Amazon, до Rozetka), підпадають під категорію B2C, продаючи товари широкому загалу. Останніми роками в площину B2C йдуть навіть виробники. Наприклад, Nike продає свої кросівки прямо на офіційному сайті, оминаючи ритейлерів. Будь-який користувач може замовити собі пару безпосередньо у виробника.

- **C2C** – електронна комерція між споживачами. Це модель електронної комерції, в межах якої один споживач продає щось іншим споживачам, маючи з ними рівний статус (Consumer-to-Consumer). Така модель може стосуватися й надання певних послуг або сервісів. Приклад такої комерції можна побачити на порталах на кшталт eBay та OLX. Користувачі можуть виставляти на продаж практично будь-які товари, знаходити покупців і закривати угоди. Ясна річ, виробники й постачальники в цей процес не залучені.
- **C2B** – електронна комерція від споживача до бізнесу. Це фактично пряма протилежність B2C, оскільки в цьому випадку вже споживач надає певні товари та послуги бізнесу (Consumer-to-Business). Найпростіший приклад: майстер створює певну хенд-мейд продукцію (нехай це будуть шкіряні браслети) та пропонує їх на продаж ритейлерам чи онлайн-платформам для подальшого продажу. До цієї ж категорії можна віднести й надання послуг на фрілансі. Такі платформи як Upwork та Fiverr дозволяють приватним особам пропонувати свої послуги комерційним клієнтам: від копірайтингу та веб-дизайну, до відеомонтажу.
- **B2A** – бізнес-адміністрування**.** Модель B2A (Business-to-Administration) або B2G (Business-to-Government) подібна B2B. Але у її межах бізнес надає свої товари або послуги не іншим підприємствам, а державним установам. Електронна комерція B2A – це державні тендери на онлайн-майданчиках; контракти на постачання товарів/послуг для урядових установ та місцевих адміністрацій; послуги електронного уряду та пов`язані з ними онлайн-платежі тощо.
- **C2A** – електронна комерція між споживачами та адміністрацією. Поки що не дуже популярний тип електронної комерції, який включає транзакції між користувачами та органами державного управління (Consumer-to-Administration). Наприклад, запис на прийом до лікаря та оплата медичних послуг, формування податкових декларацій та відповідні платежі, дистанційне навчання тощо. Загалом C2A-модель виходить за

межі визначення електронної комерції, адже має потенціал революційно змінити взаємодію між громадянами та державою. В Україні з такими форматами сьогодні експериментує та ж “Дія”. Наприклад, у додатку Мінцифри вже сьогодні можна сплачувати штрафи [1].

Переваги електронної комерції:

1. Глобалізація. Географічні обмеження відсутні, а отже ви зможете продавати свої товари та послуги покупцям з майже будь-якої країни світу. Так, наприклад, працює Amazon, eBay, Aliexpress та інші.
2. Збільшення продажів. Як не крути, проте офлайн у ваш магазин завітають, припустимо, 30 клієнтів за день, а онлайн сторінку з товарами можуть переглянути й 100, й 200, й більше людей за добу. Що більше користувачів про вас знатимуть, то більше ви зможете продавати.
3. Економія коштів. Оренда приміщення, винайм персоналу, закупівля обладнання – усе це коштує чимало. Електронна комерція дозволяє зекономити, адже для запуску бізнесу онлайн потрібно лише, наприклад, створити сторінку в Instagram (безплатно), викласти фото і відео товару, а також за потреби запустити рекламу.
4. Пришвидшення процесів. В інтернеті всі процеси (оплата, відповідь клієнту чи зміна цінників) відбуваються швидше: поки людина дійде у ваш магазин офлайн для уточнення даних, інший користувач онлайн встигне й відповідь отримати, й замовлення оплатити. Що швидше процеси, то більше продажів.
5. Аналіз даних. Через CRM-системи чи вбудовану аналітику на сайтах і в соцмережах можна дізнатися, які люди цікавляться товаром (вік, стать, країна тощо), на яких етапах продаж зривається, які продукти купують частіше та багато іншого. Це допоможе провести роботу над помилками, покращити бізнес і, знову ж таки, продавати більше.
6. Покращення клієнтського досвіду. Нижчі ціни на товари/послуги завдяки скороченню витрат, доступність 24/7, оперативність відповідей та прийняття онлайн-оплати підвищують лояльність клієнтів, роблять шопінг комфортним і сприяють збільшенню продажів.

Недоліки електронної комерції:

E-commerce має й свої недоліки, які, до речі, не зупиняють охочих розпочати бізнес у мережі. Ділимось головними недоліками електронної комерції:

- висока конкуренція;
- низька довіра клієнтів до нових бізнесів через розповсюдження шахрайства в мережі;
- ризик відсутності продажів через поганий маркетинг (довгі відповіді, затримки відправлення товару тощо);
- залежність від стабільного інтернету та наявності електроенергії;
- ризик хакерських атак, зламання акаунтів чи крадіжки персональних даних;
- можливі збої та технічні неполадки в роботі застосунків, сайтів чи онлайн-банкінгу;
- необхідність «йти в ногу з часом» і постійно покращувати маркетинг компанії [2].

Отже, електронна комерція сьогодні є фундаментальною складовою глобальної економіки. Це складна екосистема, що охоплює роздрібну торгівлю, фінансові послуги, дропшипінг, краудфандинг та взаємодію з державним сектором. Ключовою особливістю цієї галузі є відсутність географічних бар'єрів та можливість масштабування бізнесу без значних капіталовкладень в оренду фізичних приміщень.

Різноманітність моделей взаємодії – від класичних B2C (бізнес-клієнт) та B2B (бізнес-бізнес) до новітніх C2A (клієнт-адміністрація) – свідчить про глибоку інтеграцію цифрових процесів у повсякденне життя. Переваги у вигляді автоматизації, детальної аналітики даних і швидкості обслуговування беззаперечно роблять e-commerce перспективним вектором розвитку для сучасного бізнесу.

1.2 Види даних електронної комерції

1.2.1 Персональні дані користувача

Персонально ідентифікована інформація (PII) становить основу клієнтської бази будь-якого онлайн-ритейлера. До цієї категорії належать дані, що дозволяють прямо або опосередковано ідентифікувати фізичну особу.

- Прямі ідентифікатори: Повне ім'я, фізична адреса доставки, адреса електронної пошти, номер телефону, паспортні дані (у випадку транскордонної торгівлі або кредитування).
- Цифрові ідентифікатори: IP-адреси, файли cookie, ідентифікатори мобільних пристроїв (Device ID), геолокаційні дані.
- Поведінкові дані: Історія переглядів, списки бажань, патерни купівельної активності. Згідно з GDPR, ці дані також підпадають під захист, оскільки дозволяють створювати детальні профілі особистості для таргетованого впливу [3].

Втрата PII несе прямі ризики крадіжки особистих даних (Identity Theft) та соціальної інженерії проти клієнтів, а для компанії – загрозу штрафів від регуляторів та втрату лояльності.

1.2.2 Платіжні та фінансові дані

Це найбільш чутлива категорія даних, обробка якої суворо регулюється стандартом PCI DSS. Вона поділяється на дві критичні підгрупи:

- Дані власника картки (Cardholder Data – CHD): Основний номер рахунку (PAN), ім'я власника картки, термін дії, код обслуговування. Ці дані можуть зберігатися в системі продавця, але виключно у захищеному вигляді (зашифрованому або токенізованому).
- Критичні автентифікаційні дані (Sensitive Authentication Data – SAD): Повні дані магнітної смуги або чипа, коди перевірки картки (CAV2/CVC2/CVV2/CID), PIN-коди. Стандарт PCI DSS категорично

забороняє зберігання цих даних після завершення авторизації транзакції, навіть у зашифрованому вигляді. Порушення цієї вимоги є однією з найпоширеніших причин компрометації платіжних систем [4].

Платіжні дані є найбільш бажаною мішенню для кіберзлочинців, оскільки можна безпосередньо використати для:

- несанкціонованих платежів та крадіжок коштів;
- продажу на чорному ринку (затьмарені ринки та форуми);
- видачі кредитів від імені жертви;
- використання у схемах відмивання грошей.

1.2.3 Комерційні дані та бізнес-логіка

Часто недооцінений аспект безпеки – захист власне комерційної інформації, яка становить інтелектуальну власність та конкурентну перевагу компанії.

- Цінові алгоритми та стратегії: Динамічні моделі ціноутворення, оптові прайс-листи, маржинальність товарів. Витік цієї інформації до конкурентів може призвести до втрати ринкової частки.
- Дані про інвентар та постачальників: Реальні залишки на складах, умови контрактів з постачальниками, логістичні ланцюги. Інформація про дефіцит товарів може бути використана для маніпуляцій ринком або атак на ланцюги постачання.
- Маркетингові плани: Чернетки кампаній, неопубліковані промокоди, плани запуску нових продуктів. Передчасне розголошення (наприклад, через неправильно налаштовані права доступу до "внутрішніх" документів) може зірвати маркетингову стратегію [5].

Витік комерційної інформації може призвести до:

- втрати конкурентної переваги;
- падіння вартості компанії на ринку;
- втрати клієнтів на користь конкурентів;
- матеріальних фінансових збитків.

1.2.4 Системні логи та журнали аудиту

Логи є "цифровим слідом" функціонування системи, критично важливим для виявлення інцидентів та проведення криміналістичного аналізу (forensics).

- Транзакційні логи: Записи про фінансові операції, статуси замовлень, відповіді платіжних шлюзів. Вони забезпечують цілісність фінансової звітності.
- Логи безпеки: Записи про невдалі спроби входу, блокування на фаєрволі, зміни привілеїв доступу, спрацювання систем виявлення вторгнень (IDS).
- Аудиторські сліди (Audit Trails): Хронологічні записи дій адміністраторів та користувачів (хто, коли і що змінив у налаштуваннях системи). Це основа для забезпечення підзвітності та невідмовності [6].

Журнали мають критичне значення для:

- виявлення та розслідування інцидентів безпеки;
- аудиту та відповідності нормативним вимогам;
- відновлення даних та операцій після інцидентів;
- кримінального розслідування у разі злочинів.

1.2.5 Моделі та рівні класифікації даних

Класифікація даних – це процес упорядкування даних за різними категоріями відповідно до їхньої чутливості. Вона є обов'язковою для декількох стандартів нормативного дотримання, таких як HIPAA (Закон про мобільність та відповідальність медичного страхування), SOX (закон Сарбейнса-Окслі) та GDPR (Загальний регламент про захист даних).

Чотири основні типи класифікації даних – це публічні, приватні, конфіденційні та обмежені (табл. 1.1). Однак організація може мати інші рівні класифікації залежно від своїх вимог.

Таблиця 1.1 Моделі та рівні класифікації даних

Рівень класифікації	Визначення та вплив розголошення	Приклади даних в E-commerce	Необхідні заходи безпеки
Рівень 1: Публічні дані (Public)	Інформація, призначена для вільного поширення. Розголошення не несе ризиків	Каталог товарів, маркетингові матеріали, прес-релізи, публічні умови оферти, контактна інформація служби підтримки	Захист цілісності (для запобігання дефейсу сайту). Доступність (CDN). Шифрування не вимагається для читання, але необхідне для адміністрування
Рівень 2: Внутрішні дані (Internal)	Інформація для службового користування. Розголошення може спричинити незначні операційні незручності або репутаційні втрати	Внутрішні політики, довідники співробітників, чернетки проєктів, внутрішня корпоративна переписка, інструкції для персоналу	Контроль доступу (ACL), автентифікація співробітників, резервне копіювання
Рівень 3: Конфіденційні дані (Confidential)	Чутлива інформація, розголошення якої призведе до фінансових втрат,	Персональні дані клієнтів (PII), історія замовлень, деталі контрактів з постачальниками,	Шифрування при передачі (TLS) та зберіганні. Суворий контроль доступу. Багатофакторна

	штрафів або втрати клієнтів	фінансові звіти до публікації, CRM-дані	автентифікація (MFA). Логування доступу
Рівень 4: Обмежені дані (Restricted)	Критично важлива інформація. Витік призводить до катастрофічних наслідків, кримінальної відповідальності, масових штрафів	Дані кредитних карток (PAN), криптографічні ключі, паролі адміністраторів, вихідний код ядра системи, медичні дані (якщо застосовано)	Використання HSM (апаратних модулів безпеки). Максимальна ізоляція мережі. Ефемерний доступ (Just-in-Time). Найвищі стандарти шифрування

Класифікація не є разовою дією. Сучасні системи використовують методи автоматизованої класифікації на основі вмісту (Content-based) та контексту (Context-based). Наприклад, DLP-системи можуть сканувати базу даних або файлове сховище, виявляючи патерни номерів кредитних карток або ключові слова ("Конфіденційно", "Договір") і автоматично присвоювати відповідні метадані (теги), які потім зчитуються системами захисту для застосування політик шифрування [7].

1.3 Вимоги до безпеки

Традиційна тріада інформаційної безпеки CIA (Confidentiality, Integrity, Availability) є необхідним, але недостатнім базисом для сучасних систем електронної комерції. Специфіка онлайн-торгівлі, де сторони часто не знайомі особисто і укладають фінансові угоди дистанційно, вимагає розширення моделі двома додатковими стовпами: Автентичністю (Authenticity) та Невідмовністю (Non-repudiation).

1.3.1 Конфіденційність

Конфіденційність гарантує, що інформація доступна лише авторизованим суб'єктам. В e-commerce це означає захист PII клієнтів та комерційних таємниць від несанкціонованого доступу.

Загрозами можуть бути: SQL-ін'єкції, перехоплення трафіку в незахищених мережах (Man-in-the-Middle), інсайдерські загрози (співробітники, що крадуть бази клієнтів), фізична крадіжка носіїв.

Механізми забезпечення конфіденційності:

- Шифрування при передачі (in transit):
  - використання протоколу HTTPS/TLS для кодування всіх даних, що передаються між клієнтом та сервером;
  - сертифікати SSL/TLS з мінімальною силою 256-біт;
  - обов'язкове шифрування для всіх сторінок, що містять чутливі дані.
- Шифрування при зберіганні (at rest):
  - застосування алгоритмів шифрування (AES-256) для платіжних даних та персональної інформації у базі даних;
  - зберігання ключів шифрування окремо від даних (у системах управління ключами);
  - зберігання паролів не у вільному текстовому форматі, а у вигляді криптографічних хешів (bcrypt, Argon2).
- Контроль доступу (Access Control):
  - розмежування прав доступу на основі ролей користувачів (RBAC – Role-Based Access Control;
  - реалізація принципу "найменших привілеїв" (principle of least privilege) – користувач отримує тільки той мінімум прав, який необхідний для виконання своїх функцій;
  - двофакторна автентифікація (2FA/MFA) для входу в адміністративні панелі;
  - логування та моніторинг всіх спроб доступу до критичних даних.

- Розділення даних:
  - архітектурне розділення даних користувачів, щоб мінімізувати вплив витоку однієї частини системи;
  - використання токенізації платіжних даних (платіжна система отримує лише токен замість справжнього номера карти) [8].

1.3.2 Цілісність

Цілісність забезпечує точність, повноту та достовірність даних протягом усього їх життєвого циклу, захищаючи від несанкціонованих змін.

Загрози:

- маніпуляція параметрами (Parameter Tampering): Атакуючий змінює приховане поле форми HTML або параметр URL, щоб змінити ціну товару зі 100$ на 1$ перед відправкою форми на сервер;
- атаки на ланцюг постачання (Supply Chain Attacks): Впровадження шкідливого коду в сторонні бібліотеки або скрипти (наприклад, Magecart), які змінюють платіжну форму "на льоту", перехоплюючи дані без зміни видимої роботи сайту.

Механізми забезпечення цілісності:

- Криптографічні хеші та цифрові підписи:
  - використання хеш-функцій (SHA-256, SHA-512) для генерування контрольної суми даних;
  - при зміні навіть одного символу у даних – хеш змінюється, що дозволяє виявити модифікацію;
  - цифрові підписи з використанням асиметричної криптографії для підтвердження походження та цілісності критичних операцій.
- Механізми транзакцій у базі даних:
  - ACID властивості (Atomicity, Consistency, Isolation, Durability) для забезпечення коректного виконання операцій;

  - відкат операцій (rollback) у разі помилки, щоб уникнути неповних змін даних;
  - блокування записів під час редагування, щоб запобігти конфліктам при одночасному доступі.
- Контроль версій та резервне копіювання:
  - ведення історії змін для критичних даних (хто змінив, коли, що було раніше);
  - щоденне резервне копіювання для можливості відновлення при зруйнуванні даних;
  - механізми перевірки цілісності резервних копій.
- Журналювання та аудит:
  - обов'язкове логування всіх модифікацій критичних даних;
  - неперевірені журнали аудиту, які неможливо видалити або змінити заднім числом;
  - формування звітів про суперечності у даних [15].

1.3.3 Доступність

Доступність гарантує, що авторизовані користувачі можуть безперешкодно отримувати доступ до потрібних даних та систем у той час, коли їм це потрібно. Система не повинна бути недоступною через збої, атаки або навмисне блокування.

Загрози:

- DDoS-атаки: Переповнення каналів зв'язку або ресурсів сервера паразитним трафіком;
- атаки на бізнес-логіку (Denial of Inventory): Використання ботів для додавання товарів у кошик без наміру купити, що призводить до їх "бронювання" і статусу "немає в наявності" для реальних покупців (Inventory Hoarding);
- Ransomware: Шифрування даних вірусами-вимагачами.

Механізми забезпечення доступності:

- Архітектура високої доступності (High Availability):
  - резервні сервери та реплікація даних на кілька фізичних місць;
  - балансування навантаження (Load Balancing) для розподілу запитів між кількома серверами;
  - автоматичне перемикання (failover) у разі відмови одного сервера;
  - стратегія «hot standby» – резервні сервери постійно готові до швидкого перемикання.
- Захист від DDoS (Distributed Denial of Service) атак:
  - використання спеціалізованих DDoS захисних сервісів;
  - обмеження частоти запитів (rate limiting) від однієї IP адреси;
  - розпізнавання та блокування підозрілих патернів трафіку;
  - масштабована інфраструктура, яка може швидко збільшити потужність при піку навантаження.
- Резервне копіювання та аварійне відновлення (Backup & Disaster Recovery):
  - регулярне резервне копіювання (щоденно або кілька разів на день);
  - зберігання резервних копій в географічно віддалених місцях;
  - тестування процедур відновлення щоквартально;
  - RPO (Recovery Point Objective) – максимальна кількість даних, які можуть бути втрачені (для e-commerce рекомендується < 1 години);
  - RTO (Recovery Time Objective) – максимальний час, необхідний для відновлення системи (для e-commerce < 4 годин).
- Моніторинг та управління ресурсами:
  - постійний моніторинг стану сервера (CPU, пам'ять, дисковий простір);
  - автоматичне оповіщення при виявленні потенційних проблем;
  - планування розширення інфраструктури на основі тенденцій зростання навантаження;
  - оптимізація продуктивності на основі аналізу логів та метрик [9].

1.3.4 Автентичність

Автентичність – це властивість, яка підтверджує, що суб'єкт (користувач, сервер, процес) є саме тим, за кого себе видає. Вона відрізняється від авторизації (яка визначає права доступу) і є передумовою довіри.

Механізми забезпечення автентичності [10]:

- Автентифікація користувачів:
  - Парольна автентифікація з використанням стійких паролів та їх безпечного зберігання (bcrypt, Argon2 з salt);
  - Мультифакторна автентифікація (MFA) – поєднання паролю з SMS-кодом, додатком TOTP або біометрією;
  - Перевірка email-адреси та номера телефону при реєстрації;
  - Двофакторна автентифікація для критичних операцій (зміна пароля, доступ до платіжної інформації).
- Механізми керування сеансами та токенами:
  - JWT токени (JSON Web Tokens) з криптографічним підписом для забезпечення того, що токен не був змінений;
  - обмеження часу життя токена (exp claim);
  - перевірка підпису токена на кожному запиті;
  - використання refresh-токенів для безпечного оновлення доступу без повторного введення паролю;
  - завершення сеансу при виході користувача.
- Автентифікація сервера:
  - SSL/TLS сертифікати з верифікацією доменного імені сервера;
  - перевірка сертифіката браузером перед встановленням шифрованого з'єднання;
  - Certificate Pinning (прив’язка сертифікатів) для мобільних додатків, щоб запобігти MITM-атакам;
  - використання HSTS (HTTP Strict Transport Security) щоб змусити браузер використовувати HTTPS.

- Автентичність даних:
  - цифрові підписи для критичних операцій (платежі, видалення облікового запису);
  - HMAC (Hash-based Message Authentication Code) для перевірки цілісності та джерела даних;
  - використання асиметричної криптографії (RSA, ECDSA) для невідмовності операцій.

1.3.5 Невідмовність

Невідмовність забезпечує неможливість заперечення суб'єктом факту виконання певної дії або авторства інформації. Це критично важливий юридичний та технічний аспект для вирішення суперечок.

Механізми забезпечення невідмовності:

- Журналювання операцій:
  - обов'язкове логування всіх значущих операцій (платежі, видалення облікового запису, зміна даних);
  - кожному запису логу повинна бути позначка часу (timestamp) та ідентифікатор користувача;
  - логи повинні зберігатися в незмінному вигляді (immutable logs).
  - ведення окремого журналу для особливо критичних операцій з посиленими гарантіями збереження.
- Цифрові підписи:
  - для платіжних операцій використовуються цифрові підписи, які не можуть бути заперечені;
  - підпис створюється приватним ключем користувача, який тільки він знає;
  - підпис може бути верифіковано публічним ключем, але неможливо підробити без приватного ключа;
  - архівування всіх підписаних операцій з метою забезпечення довгострокового зберігання доказів.

- Аудиторські сліди (Audit Trails):
  - детальний запис усіх дій користувача в системі;
  - невідмовні записи про те, коли користувач увійшов до системи, які операції здійснював та які дані переглядав;
  - збереження IP-адреси, з якої користувач здійснював операцію;
  - можливість відтворити всю послідовність операцій користувача у будь-який момент часу.
- Вимоги при платіжних операціях:
  - явне підтвердження операції користувачем (наприклад, введення OTP-коду);
  - Email/SMS підтвердження відправляється користувачу після кожної операції;
  - тимчасовий період, протягом якого користувач може ініціювати повернення платежу (chargeback protection);
  - координація з платіжним процесором для синхронізації та ведення власних журналів операцій [18].

1.4 Стандартизація та нормативне регулювання

1. ISO/IEC 27001:2022 – Управління інформаційною безпекою

Міжнародний стандарт для розроблення та впровадження систем управління інформаційною безпекою (ISMS) в організаціях будь-якого розміру [17].

Основні компоненти:

- Розроблення політики безпеки на рівні керівництва
- Управління ризиками (ідентифікація, оцінка, мітигація)
- Контроль доступу та авторизація
- Криптографічна політика
- Безпека комунікацій та операцій
- Безпечна розробка додатків
- Фізична та екологічна безпека

- Управління інцидентами
- Регулярні аудити та сертифікація

Для e-commerce: демонструє серйозний підхід до безпеки, зменшує ризики, зміцнює довіру клієнтів. Сертифікація дійсна 3 роки з щорічними перевірками.

2. PCI DSS (Payment Card Industry Data Security Standard)

Обов'язковий стандарт для організацій, що обробляють платіжні дані. Розроблений платіжними системами (Visa, MasterCard, American Express, Discover, JCB) [19].

PCI DSS розроблений радою безпеки платіжної індустрії (PCI Security Standards Council), до складу якої входять основні платіжні системи: Visa, MasterCard, American Express, Discover та JCB. Цей стандарт є обов'язковим для всіх організацій, які обробляють, зберігають або передають дані платіжних карток.

На відміну від ISO/IEC 27001, PCI DSS зосереджено виключно на захисті платіжних даних та встановлює конкретні, детальні вимоги щодо їх обробки.

12 основних вимог PCI DSS (табл. 1.2):

Таблиця 1.2 Структура PCI DSS

Розділ	Вимоги	Деталі
Мережева безпека	1-2	Firewall, видалення параметрів за замовчуванням
Захист платіжних даних	3-4	Шифрування даних (AES-256), HTTPS/TLS для передачі
Управління вразливістю	5-6	Антивірус, безпечна розробка, тестування на вразливості
Контроль доступу	7-8	RBAC, сильні паролі, MFA, логування входу
Моніторинг	9-10	Фізична безпека, логування всіх операцій (immutable logs)

Інциденти	11-12	Сканування вразливостей, тестування на проникнення, політика безпеки

Рівні відповідності:

- Рівень 1: > 6 млн операцій/рік – щорічний аудит від QSA
- Рівень 2: 1-6 млн операцій/рік – самооцінка або аудит
- Рівень 3: 20K-1 млн операцій/рік – самооцінка + сканування
- Рівень 4: < 20K операцій/рік – самооцінка + сканування

Штрафи: $5,000–$100,000 на місяць за невідповідність.

3. OWASP ASVS – Стандарт верифікації безпеки веб-додатків

Фреймворк OWASP для встановлення вимог безпеки веб-додатків на різних рівнях критичності. Якщо ISO 27001 фокусується на процесах, а PCI DSS на карткових даних, то OWASP ASVS (Application Security Verification Standard) надає технічний чек-лист для перевірки безпеки самого коду веб-застосунку.

Три рівні верифікації (табл. 1.3):

Таблиця 1.3 Рівні верифікації OWASP ASVS

Рівень	Для кого	Вимоги
Рівень 1	Прості веб-сайти	Базова безпека: паролі, HTTPS, захист від XSS/SQL ін’єкцій
Рівень 2	E-commerce, чутливі дані	MFA, управління сеансами, CSRF захист, XXE захист
Рівень 3	Критична інфраструктура	Двофакторна автентифікація, цифрові підписи, HSM для ключів, детектування атак

14 доменів безпеки: архітектура, автентифікація, управління сеансами, контроль доступу, валідація входу, криптографія, логування, захист даних, комунікації, бізнес-логіка, файли, API, конфігурація, мобільна безпека [20].

Для e-commerce: Детальний контрольний список для розробників. Безплатний та open-source. Більшість e-commerce потребує Рівня 2.

4. GDPR (General Data Protection Regulation)

Європейське законодавство, що встановлює суворі вимоги щодо обробки персональних даних. Обов'язковий закон для всіх компаній, що обробляють дані громадян ЄС.

6 ключових принципів:

1. законність, справедливість, прозорість – законна основа (згода, контракт, зобов'язання), прозора приватна політика;
2. обмеженість мети – дані можуть використовуватися тільки для цілей, для яких були зібрані;
3. мінімізація даних – зберігати тільки необхідні дані;
4. точність – дані повинні бути точні та актуальні;
5. обмеженість зберігання – встановити період видалення (наприклад, 30 днів для видалених облікових записів);
6. цілісність та конфіденційність – шифрування, контроль доступу, резервні копії.

6 прав користувачів:

1. право на доступ – запросити копію своїх даних (30 днів, безплатно);
2. право на виправлення – виправити неправильні дані (30 днів);
3. право на видалення – запросити видалення даних (30 днів);
4. право на обмеження обробки – припинити обробку без видалення;
5. право на портативність – експортувати дані в стандартному форматі;
6. право на заперечення – заперечити проти обробки для маркетингу.

Обов'язки організацій:

- мати законну основу для обробки кожного типу даних;
- написати зрозумілу приватну політику;
- управління згодою (явна, легко скасується);
- забезпечити безпеку даних (шифрування, контроль доступу, регулярні аудити);
- провести DPIA для ризикованих операцій;
- при витоку: повідомити регулятора (72 години), користувачів (без затримок) [21].

Штрафи: до €20 млн або 4% від глобального річного обороту за серйозні порушення. Приклади: Meta – €1.2 млрд, Google – €50 млн, Amazon – €746 млн.

Для e-commerce: Обов'язкова для всіх платформ з користувачами з ЄС. Демонстрація поваги до приватності – конкурентна перевага.

1.5 Актуальність теми дослідження

На сучасному етапі розвитку цифрової економіки електронна комерція (e-commerce) трансформувалася з допоміжного каналу продажів у критично важливу інфраструктуру глобального товарообігу. Актуальність наукового дослідження методів виявлення та нейтралізації загроз у цій сфері визначається конвергенцією трьох кризових факторів: експоненційним зростанням вартості активів, що обробляються онлайн, безпрецедентною ескалацією кіберзагроз із використанням штучного інтелекту, а також докорінною зміною нормативно-правового ландшафту України та ЄС. В умовах, коли глобальні збитки від кіберзлочинності, за прогнозами, сягнуть 10,5 трильйона доларів США щорічно до кінця 2025 року, розробка ефективних механізмів захисту даних стає питанням економічного виживання бізнесу та національної безпеки.

Стрімка цифровізація суспільства призвела до того, що значна частина світового капіталу перемістилася у кіберпростір. За прогнозами аналітиків, у 2025 році обсяг світового ринку e-commerce сягне 6,86 трильйона доларів США, а до 2027 року цей канал забезпечуватиме 22,6% усіх роздрібних продажів планети. Така концентрація фінансових потоків та персональних даних перетворює платформи електронної комерції на пріоритетну ціль для транснаціональних кіберугруповань.

Для України актуальність дослідження посилюється специфічним контекстом повномасштабної війни. Електронна комерція стала "економічним тилом", що дозволяє бізнесу функціонувати в умовах руйнування фізичної інфраструктури. У 2024 році український ринок e-commerce продемонстрував стійкість із обігом у 4,375 мільярда доларів, і очікується подальше зростання на 10-

15% у 2025 році. Проте вітчизняний бізнес стикається з подвійним тиском: глобальними трендами кіберзлочинності та цілеспрямованими деструктивними атаками на логістичну та платіжну інфраструктуру з боку ворожих держав (приклади атак на поштові сервіси та ритейлерів у 2023-2024 роках) [11].

Вартість інцидентів безпеки продовжує зростати. Середня ціна витоку даних у ритейлі у 2025 році склала 3,54 мільйона доларів, що на 18% більше порівняно з попереднім роком. Це свідчить про те, що існуючі методи захисту втрачають ефективність проти нових векторів атак, вимагаючи наукового переосмислення підходів до побудови систем безпеки.

Традиційні сигнатурні методи виявлення загроз стають неефективними в умовах появи нових, складних векторів атак, які використовують вразливості архітектури та людський фактор.

Ключові вектори загроз, що визначають актуальність теми:

1. Атаки на стороні клієнта (Client-Side Attacks / Magecart): зловмисники масово переходять від злому серверів до впровадження шкідливого JavaScript-коду у браузери користувачів через сторонні бібліотеки (рекламні трекери, чат-боти). Кількість таких атак зросла на 103% за пів року у 2025 році. Це створює "сліпу зону" для класичних засобів захисту (WAF), вимагаючи розробки нових методів моніторингу цілісності скриптів.
2. Вразливості API та бізнес-логіки: сучасна headless-архітектура e-commerce базується на API, які часто мають вразливості авторизації (BOLA). Крім того, зростає кількість атак на бізнес-логіку, таких як "Denial of Inventory", коли боти масово бронюють товари, роблячи їх недоступними для реальних покупців. Виявлення таких атак потребує поведінкового аналізу, а не просто блокування за IP.
3. Загрози, підсилені штучним інтелектом: доступність генеративного AI дозволила зловмисникам автоматизувати створення фішингу, що не відрізняється від легітимних листів, та генерувати "синтетичні особистості" (Synthetic Identity Fraud) для обходу систем антифроду. Протидія таким загрозам вимагає впровадження дзеркальних AI-методів захисту.

4. Ransomware 3.0: тактика зловмисників еволюціонувала від простого шифрування до "потрійного вимагання": шифрування систем, крадіжка даних клієнтів для шантажу та атаки на партнерів постраждалої компанії. Це вимагає перегляду стратегій резервного копіювання та реагування на інциденти.

1.6 Аналіз вже існуючих рішень

Історично склалося так, що захист фокусувався на мережевому рівні (L3/L4 за моделлю OSI). Проте, сучасні загрози для e-commerce масово змістилися на прикладний рівень (L7). Атаки, такі як SQL-ін'єкції (SQLi), міжсайтовий скриптинг (XSS) та підробка міжсайтових запитів (CSRF), залишаються актуальними, але до них додалися більш витончені загрози: зловживання API (API abuse), автоматизовані бот-мережі, що імітують поведінку людей, та атаки на ланцюжок поставок програмного забезпечення (supply chain attacks).

Традиційний Web Application Firewall (WAF) був розроблений для епохи статичного вебу. Функціонуючи як зворотний проксі-сервер, він інспектує вхідний HTTP/HTTPS трафік, порівнюючи його з базою сигнатур відомих атак. Проте, в динамічному середовищі e-commerce, де код оновлюється щодня через CI/CD пайплайни, а бізнес-логіка постійно змінюється, класичні WAF стикаються з критичними обмеженнями.

Відповіддю на ці виклики стала поява Web Application and API Protection (WAAP). Gartner та інші аналітичні агенції визначають WAAP як еволюцію WAF, що інтегрує чотири ключові компоненти в єдину платформу:

- WAF наступного покоління: використовує машинне навчання для профілювання легітимного трафіку та автоматичного створення правил, зменшуючи залежність від ручного налаштування;
- захист API: спеціалізовані модулі для розуміння структури API (REST, GraphQL, gRPC), валідації схем (OpenAPI/Swagger) та виявлення специфічних для API загроз;

- управління ботами (Bot Management): використання поведінкової біометрії, аналізу відбитків пристроїв (fingerprinting) та репутаційних баз для розрізнення людей, корисних ботів (пошуковиків) та шкідливих ботів;
- захист від DDoS: інтегрований захист від об'ємних атак на рівнях L3/L4/L7, що реалізується на межі мережі (edge), ближче до джерела атаки.

Ринок WAAP є висококонкурентним, з домінуванням гравців, що володіють глобальними мережами доставки контенту (CDN). Для українських e-commerce компаній вибір рішення часто залежить від балансу між вартістю, наявністю локальних точок присутності (PoP) та відповідністю регуляторним нормам.

Cloudflare: домінування за рахунок масштабу

Cloudflare займає левову частку ринку зворотних проксі (близько 81.9% веб-сайтів, що використовують такі послуги), що робить його де-факто стандартом для багатьох бізнесів.

- Технологічні переваги: глобальна мережа з пропускною здатністю понад 228 Тбіт/с дозволяє поглинати навіть найпотужніші DDoS-атаки без впливу на продуктивність. Інтеграція WAF з CDN забезпечує прискорення контенту, що є критичним для конверсії в e-commerce;
- управління ботами: функція "Super Bot Fight Mode" використовує машинне навчання для аналізу трафіку з мільйонів ресурсів, що дозволяє ефективно блокувати автоматизовані атаки. Проте, найбільш просунуті функції управління ботами доступні лише в Enterprise-планах;
- специфіка для України: Cloudflare має точку присутності в Києві, що забезпечує мінімальну затримку для локальних користувачів. Висока популярність сервісу в Україні (19.8% від усіх веб-сайтів) також зумовлена доступністю безкоштовних планів, які надають базовий захист;
- недоліки: Користувачі відзначають складність налаштування кастомних правил для складних бізнес-сценаріїв та певні обмеження в аналітиці на нижчих тарифних планах.

Akamai: еталон корпоративної безпеки

Akamai App & API Protector позиціонується як рішення преміум-класу для великих підприємств, фінансових установ та ритейлерів, для яких простій є неприпустимим [22].

- Глибина захисту: Akamai пропонує одні з найбільш зрілих рішень для захисту API та управління ботами. Їхня технологія "Page Integrity Manager" інтегрує захист на стороні клієнта безпосередньо в платформу WAAP, що є значною перевагою для боротьби з атаками Magecart;
- адаптивність: використання адаптивних моделей безпеки, що самонавчаються, дозволяє знизити рівень помилкових спрацьовувань та автоматизувати оновлення політик безпеки;
- недоліки: висока вартість та складність впровадження. Рішення Akamai часто вимагають залучення виділених фахівців або професійних сервісів для налаштування та підтримки, що може бути бар'єром для середнього бізнесу в Україні.

Imperva: гібридний підхід та захист даних

Imperva вирізняється фокусом на захисті даних та гібридних моделях розгортання, що включають як хмарні сервіси, так і рішення on-premise.

- RASP (Runtime Application Self-Protection): унікальною пропозицією Imperva є інтеграція технології RASP, яка працює всередині додатку і здатна блокувати атаки, що обійшли периметральний захист, аналізуючи виконання коду в реальному часі;
- точність: бенчмарки показують надзвичайно низький рівень помилкових спрацьовувань (близько 0.009%) для їх WAF, що є критичним для забезпечення безперебійної роботи e-commerce;
- захист баз даних: сильна інтеграція з продуктами для захисту баз даних робить Imperva привабливою для компаній, які зберігають великі обсяги чутливих даних клієнтів.

Wallarm: Спеціалізація на API

Wallarm представляє нове покоління рішень, орієнтованих на API-first компанії та захист від логічних атак.

- Активна верифікація загроз: Wallarm використовує унікальний підхід, перевіряючи потенційні атаки шляхом безпечного "перегравання" їх на копії трафіку, що дозволяє підтвердити вразливість перед блокуванням і мінімізувати false positives;
- виявлення BOLA: платформа спеціалізується на виявленні атак на бізнес-логіку, зокрема BOLA (Broken Object Level Authorization), які часто пропускаються традиційними WAAP;
- інтеграція з DevOps: Wallarm тісно інтегрується з інструментами CI/CD, забезпечуючи безпеку на етапі розробки та тестування.

Anti-Fraud та Bot Management: захист бізнес-логіки

Якщо WAAP захищає від технічних експлойтів, то системи Anti-Fraud та Bot Management призначені для боротьби зі зловживанням бізнес-логікою. Найбільш руйнівні атаки в e-commerce часто використовують абсолютно легітимні функції сайту: вхід в акаунт, додавання товару в кошик, оформлення замовлення.

Провідні рішення Anti-Fraud

- DataDome: спеціалізується на захисті від ботів у реальному часі на рівні Edge. Використовує AI для аналізу тисяч сигналів (рух миші, параметри браузера) для виявлення нелюдської поведінки. Вважається одним з кращих рішень для технічного блокування ботів;
- Sift: платформа "цифрової довіри", що фокусується на запобіганні платіжному шахрайству та зловживанням контентом. Використовує глобальну мережу даних для оцінки ризиків у реальному часі, дозволяючи приймати рішення (блокувати/дозволити) на основі скорингу, а не жорстких правил;
- Signifyd: пропонує модель "гарантованого захисту від шахрайства", беручи на себе фінансову відповідальність за чарджбеки. Це особливо привабливо для рітейлерів, які хочуть перекласти фінансові ризики на вендора;
- SEON: інструмент для збагачення даних (Data Enrichment), що будує "цифровий слід" користувача на основі аналізу email, телефону, IP та

соціальних мереж. Часто використовується як додатковий шар перевірки для зменшення false positives.

Client-Side Protection: Захист "сліпої зони" браузера

Традиційний захист на стороні сервера (WAF/WAAP) не бачить того, що відбувається в браузері користувача. Сучасні e-commerce сайти завантажують десятки сторонніх скриптів (аналітика, чат-боти, рекламні трекери). Компрометація будь-якого з цих скриптів (атака на ланцюжок поставок) дозволяє зловмисникам впроваджувати шкідливий код, що перехоплює дані платіжних карток безпосередньо з полів введення – техніка, відома як Magecart або Formjacking.

Огляд рішень Client-Side Protection

- Jscrambler: лідер у галузі захисту коду на стороні клієнта. Забезпечує поліморфну обфускацію (робить код нечитабельним для аналізу) та моніторинг цілісності веб-сторінок у реальному часі, блокуючи спроби ексфільтрації даних;
- Feroot: фокусується на автоматизації комплаєнсу PCI DSS. Інструменти "PageGuard" та "Inspector" автоматично виявляють всі скрипти, що працюють на сторінці, та місця, куди вони передають дані, забезпечуючи швидке реагування на інциденти;
- Akamai Page Integrity Manager: розширення платформи Akamai, що моніторить виконання скриптів у браузері, використовуючи глобальну базу репутації скриптів для виявлення аномалій;
- Content Security Policy (CSP): механізм браузера, що дозволяє створювати "білі списки" доменів, з яких дозволено завантаження скриптів. Хоча це безкоштовний інструмент, управління CSP є складним і часто призводить до порушення функціональності сайту, тому комерційні рішення часто використовуються для автоматизації управління політиками CSP.

1.7 Визначення цілей роботи та постановка задачі

Метою роботи є розробка та реалізація програмної системи, призначеної для автоматизованого та комплексного виявлення та нейтралізації загроз безпеки персональних та платіжних даних в системах електронної комерції. Система має забезпечити високий рівень ефективності та швидкості реагування на кіберзагрози, дозволяючи фахівцям та компаніям отримувати точну інформацію про ризики без необхідності значних витрат на залучення спеціалістів з інформаційної безпеки. Досягнення цієї мети сприятиме ранньому виявленню атак, своєчасному втручанню та моніторингу динаміки змін у ризиках безпеки.

Об’єктом дослідження є процеси захищеного обміну даними та моніторингу подій безпеки в інформаційних системах електронної комерції, а також технічні засоби та організаційні рішення, що забезпечують захист інформаційних активів.

Предметом дослідження є сукупність методів, засобів, алгоритмів та політик інформаційної безпеки, спрямованих на виявлення вразливостей, детекцію атак та нейтралізацію загроз конфіденційності, цілісності та доступності даних у системах електронної комерції, зокрема з використанням методів аналізу логів та аномалійної детекції.

Аналіз вимог до програмної системи.

Здійснити комплексний аналіз функціональних вимог до програмної системи виявлення та нейтралізації загроз, зокрема:

- забезпечення моніторингу логів e-commerce платформи в режимі, наближеному до реального часу;
- виявлення аномальної поведінки користувачів та операцій (підозрілі входи, нетипові транзакції, масові помилки запитів тощо);
- класифікація виявлених подій та загроз за рівнем серйозності (низький, середній, високий, критичний);
- підтримка механізмів автоматизованого реагування на критичні загрози (блокування доступу, призупинення операцій, активація додаткових засобів контролю);
- генерування алертів та сповіщень адміністраторам системи (email, системні повідомлення, інтеграція з існуючими системами моніторингу).

Визначити нефункціональні вимоги до програмної системи, зокрема:

- точність: досягнення мінімально необхідних показників якості детекції загроз, зокрема не нижче 85% для показника precision та не нижче 80% для показника recall в задачі виявлення атак;
- швидкодія: забезпечення часу виявлення загроз, який не перевищує однієї секунди від моменту появи відповідного запису в журналі подій, для сценаріїв, що потребують оперативної реакції;
- масштабованість: можливість обробки великого обсягу подій (логів) типових для e-commerce платформ з підтримкою горизонтального масштабування системи при збільшенні навантаження;
- надійність: забезпечення доступності системи на рівні не нижче 99,9% за рахунок використання резервування, відмовостійких механізмів та контрольованих процедур оновлення;
- безпека даних: застосування шифрування під час зберігання та передавання логів, реалізація ролей доступу до системи, ведення журналів доступу та дій користувачів;
- зручність інтерфейсу: розроблення інтуїтивно зрозумілого, наочного та ергономічного графічного інтерфейсу для адміністраторів та аналітиків, який забезпечує швидкий доступ до основних функцій моніторингу та аналізу.

Визначити критерії успішності проєкту, серед яких:

- досягнення максимального рівня точності виявлення атак відомих типів понад 90% за ключовими показниками якості;
- забезпечення середнього часу обробки одного окремого набору логів або пакета подій, що не перевищує 500 мс за типовим сценарієм використання;
- обмеження частки хибних спрацювань системи (помилкових тривог) до рівня менш ніж 10% від загальної кількості згенерованих алертів у заданому тестовому середовищі;
- забезпечення підтримки виявлення та класифікації значної кількості різних типів атак, релевантних для середовища електронної комерції;

– реалізація можливості виявлення раніше невідомих (нетипових) загроз та аномалій з прийнятними показниками якості.

Формування та підготовка набору даних для аналізу та тестування.

– Здійснити відбір джерел даних, необхідних для роботи системи, зокрема логів веб-серверів, додатків, систем автентифікації, платіжних шлюзів та інших компонентів e-commerce платформи.
– Реалізувати комбінований підхід до формування вибірки подій для аналізу, що включає:
  - симуляцію типових атак у тестовому середовищі (лабораторії) з подальшим збором відповідних логів;
  - запис реальних логів функціонування e-commerce платформ (за наявності дозволу та з дотриманням вимог безпеки та конфіденційності);
  - за необхідності – обмежене синтезування додаткових штучних даних для окремих сценаріїв з використанням сучасних методів генерації даних (наприклад, засобів на основі генеративних моделей), з подальшою ретельною валідацією цих даних;
  - провести очищення, нормалізацію та попередню обробку зібраних даних, включно з уніфікацією форматів логів, видаленням дублюючих або нерелевантних записів та анонімізацією персональних даних у разі необхідності.

Розробка загальної архітектури програмної системи та механізмів обробки загроз.

– Спроєктувати архітектуру системи, яка передбачає модулі збору, агрегації та зберігання логів, модуль аналізу та виявлення загроз, модуль прийняття рішень щодо реагування, а також модуль представлення результатів користувачам.
– Визначити взаємодію між компонентами системи, формати обміну даними, інтерфейси інтеграції з існуючими системами (наприклад, SIEM, WAF, системами логування тощо).

– Передбачити можливість подальшого розширення функціональності системи без суттєвих змін базової архітектури.

Створення програмної системи та користувацького інтерфейсу.

Імплементація основних функціональних модулів системи.

– Реалізувати серверну частину системи, що забезпечує:
  - приймання та обробку вхідних логів від e-commerce платформи;
  - застосування алгоритмів аналізу для виявлення аномалій та потенційних загроз;
  - збереження результатів аналізу та подій безпеки до бази даних;
  - формування алертів та рекомендацій щодо реагування.

Розробка графічного інтерфейсу користувача. Компоненти інтерфейсу мають включати:

– **Dashboard моніторингу (в режимі, наближеному до реального часу):**
  - графік активних загроз за часовою шкалою;
  - лічильник атак за типами та рівнями серйозності;
  - візуалізацію географічного розподілу виявлених загроз (у разі наявності відповідних даних);
  - індикатор загального поточного стану системи (наприклад, у вигляді кольорової статус-лінії).
– **Розділ завантаження та аналізу даних:**
  - форму для завантаження файлів логів або налаштувань для підключення до зовнішніх джерел;
  - засоби запуску процедури аналізу;
  - індикатор прогресу обробки логів.
– **Розділ результатів аналізу:**
  - таблицю виявлених загроз із зазначенням типу, часу, рівня серйозності та інших важливих параметрів;
  - можливість перегляду детальної інформації про кожну загрозу;
  - відображення рекомендацій щодо можливих дій у відповідь;

- функції експорту результатів аналізу у популярні формати (наприклад, CSV, PDF).

– **Розділ історії та статистики:**
  - графіки динаміки загроз за різні періоди часу;
  - перелік найбільш поширених (топ) типів атак;
  - перелік найбільш вразливих операцій, компонентів або сервісів системи;
  - аналітичні показники, що дозволяють оцінити ефективність заходів безпеки.
– **Розділ налаштувань та конфігурації системи:**
  - налаштування порогових значень для виявлення та класифікації загроз;
  - вибір або конфігурація алгоритмів (моделей) для аналізу подій;
  - керування параметрами сповіщень (канали, частота, фільтрація);
  - налаштування прав доступу для різних категорій користувачів системи.

Комплексне тестування та оцінка ефективності програмної системи.

– Провести тестування роботи системи із застосуванням незалежного набору даних (логів), який не використовувався в процесі розроблення та налаштування алгоритмів аналізу.
– Оцінити якість виявлення загроз за визначеними показниками (precision, recall, частка хибних спрацювань, час реакції тощо) та порівняти одержані результати з попередньо встановленими критеріями успішності.
– Проаналізувати типові випадки помилок системи, зокрема випадки невиявлених загроз та хибних тривог, з метою виявлення можливостей для покращення якості роботи.
– Сформулювати рекомендації щодо подальшого вдосконалення програмної системи, можливостей інтеграції з іншими засобами захисту та перспектив її використання в реальних умовах функціонування підприємств електронної комерції.

На рисунку 1.6 зображено дерево цілей для даної роботи.

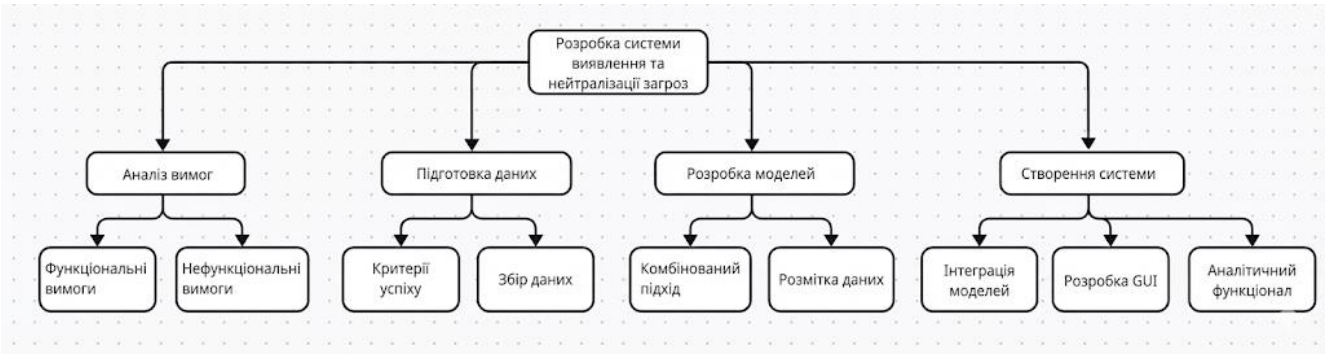


Рисунок 1.6 – Дерево цілей

Очікувані результати роботи

Технічні результати:

- Навчена та оптимізована модель (точність > 85%)
- Web-додаток з інтуїтивним інтерфейсом
- API для інтеграції з іншими системами
- Документація та гайди
- Unit та інтеграційні тести

Навчальні результати:

- Практичні навички в розробці моделей
- Знання про кіберзагрози та методи захисту
- Вміння розробляти web-системи на Python
- Навички в аналізі та інтерпретації результатів

Наукові результати:

- Публікація методів та результатів дослідження
- Порівняння із існуючими рішеннями
- Рекомендації щодо вдосконалення
- Можливості використання в реальній практиці

Розділ 2. Основи загроз та методи їх виявлення

2.1 Класифікація загроз для систем електронної комерції

Системи електронної комерції є комплексними розподіленими інформаційними системами, які обробляють персональні дані користувачів, платіжну інформацію, комерційні дані та системні журнали. Відповідно, вони піддаються широкому спектру кіберзагроз, які доцільно класифікувати за кількома ознаками: за рівнем реалізації атаки, за ціллю, за ступенем автоматизації та за джерелом походження.

У контексті даної роботи доцільно виділити такі основні групи загроз:

- атаки на веб-додаток;
- атаки на мережевій інфраструктурі;
- атаки на облікові записи користувачів;
- загрози, пов’язані з платіжними операціями;
- загрози на основі соціальної інженерії та внутрішніх зловживань.

2.1.1 Атаки на веб-додатки

До атак на веб-додаток належать загрози, що експлуатують помилки у логіці обробки HTTP-запитів, некоректну обробку вхідних даних, вразливості в механізмах автентифікації та авторизації, а також помилки конфігурації серверного програмного забезпечення.

Основні типи атак на веб-додатки [12]:

1. **SQL-ін’єкції (SQL Injection)**

Зловмисник передає у вхідних параметрах фрагменти SQL-коду, які некоректно вставляються у запити до бази даних. Наслідки:

- читання, модифікація або видалення даних;
- обхід механізмів автентифікації;
- отримання підвищених привілеїв на рівні СУБД.

Причини:

- відсутність параметризованих запитів;
- конкатенація рядків запиту з вхідними даними без валідації;
- надмірні привілеї користувача бази даних.

2. **Міжсайтове виконання скриптів (Cross-Site Scripting, XSS)** Вразливість, за якої користувачеві повертається сторінка із невірно екранованими даними, введеними іншим користувачем або зловмисником. Це дозволяє виконати довільний сценарій у браузері жертви. Наслідки:
   - викрадення сесійних токенів;
   - підміна вмісту сторінки;
   - перенаправлення на фішингові ресурси.
3. **Підробка міжсайтових запитів (Cross-Site Request Forgery, CSRF)** Атака, при якій зловмисник змушує браузер авторизованого користувача виконати небажані дії від його імені. Приклади:
   - зміна пароля;
   - оформлення замовлення;
   - зміна платіжних реквізитів.

   Причини:
   - відсутність CSRF-токенів;
   - відсутність перевірки походження запиту.
4. **Path Traversal** Спроби доступу до файлів або директорій за межами кореневого каталогу веб-додатка шляхом використання спеціально сформованих шляхів (наприклад, ../). Наслідки:
   - перегляд конфіденційних файлів;
   - розкриття конфігураційних даних;
   - можливе отримання паролів та ключів.
5. **Командні ін’єкції (Command Injection)** Включення даних користувача у системну команду, що виконується на сервері (через shell або системні виклики). Наслідки:
   - виконання довільних команд ОС;

- повний контроль над сервером при успішній експлуатації.

6. **Інші вразливості з OWASP Top 10**

Зокрема:

- небезпечна десеріалізація;
- неправильне керування сесіями;
- помилки в контролі доступу;
- витоки конфіденційної інформації через помилки в конфігурації.

Для систем електронної комерції атаки на веб-додаток є критичними, оскільки через веб-інтерфейс здійснюються реєстрація користувачів, авторизація, перегляд товарів та оформлення замовлень.

2.1.2. Атаки на мережевому рівні

Мережеві атаки спрямовані на порушення доступності сервісів, перехоплення або підміну трафіку, сканування та виявлення вразливостей на рівні протоколів.

Основні типи:

1. **DDoS-атаки (Distributed Denial of Service)**

Масовані атаки на перевантаження серверних або мережевих ресурсів шляхом надмірної кількості запитів [13]:

- HTTP flood;
- SYN flood;
- UDP flood тощо.

Наслідок – недоступність сайту для легітимних користувачів, що безпосередньо знижує доступність та доходи.

2. **Сканування портів та сервісів**

Попередній етап більшості атак, коли зловмисник визначає відкриті порти, запущені сервіси, їх версії та потенційні вразливості.

3. **Man-in-the-Middle (MITM)**

Перехоплення та, за можливості, модифікація трафіку між клієнтом і сервером. Особливо небезпечно при:

- відсутності HTTPS;
- використанні застарілих версій TLS;
- компрометації сертифікатів.

4. **DNS-атаки (DNS spoofing, DNS cache poisoning)**

Підміна DNS-відповідей, що призводить до перенаправлення користувачів на шкідливі або фішингові ресурси.

2.1.3. Атаки на облікові записи користувачів

Облікові записи є ключовою мішенню зловмисників, оскільки відкривають доступ до персональних даних, історії замовлень, платіжних інструментів.
Основні загрози [23]:

1. **Brute-force**

Перебір паролів (часто автоматизований) з метою підбору правильних облікових даних.

2. **Credential stuffing**

Використання пар логін/пароль що витекли з інших сервісів, розраховане на те, що користувачі повторно використовують одні й ті самі паролі.

3. **Password spraying**

Спроба використання невеликого набору популярних паролів для великої кількості користувачів.

4. **Session hijacking**

Перехоплення сесійних токенів (cookies, JWT) з подальшим використанням їх від імені жертви.

5. **Account Takeover (ATO)**

Комплекс методів, що в результаті дають зловмиснику повний контроль над обліковим записом користувача: зміну пароля, адреси, платіжних даних.

2.1.4. Загрози платіжній та фінансовій інформації

Окрему групу становлять загрози для платіжних даних та платіжної інфраструктури:

1. **Шахрайство з платіжними картками**
   Використання викрадених реквізитів карток для здійснення покупок.
2. **Впровадження сторонніх скриптів на сторінках оплати (Magecart-подібні атаки)**

Шкідливий JavaScript перехоплює дані платіжних форм безпосередньо у браузері користувача.

3. **Підміна платіжних реквізитів**

Неавторизована зміна реквізитів отримувача в особистому кабінеті чи в панелі адміністрування.

4. **Чарджбек-шахрайство**

Подання необґрунтованих запитів на повернення коштів після отримання товару або послуги.

2.1.5. Соціальна інженерія та внутрішні загрози

1. **Фішинг та spear-phishing**

Імітація легітимних листів, повідомлень чи інтерфейсів з метою отримання облікових даних, платіжної інформації або переконання користувача виконати небезпечні дії.

2. **Внутрішні загрози (insider threats)**

Несумлінні співробітники, підрядники або партнери, які мають легітимний доступ до системи і можуть:

- здійснювати несанкціоноване копіювання даних;
- змінювати налаштування безпеки;
- навмисно створювати вразливості.

3. **APT (Advanced Persistent Threats)**

Тривалі комплексні атаки на великі організації, побудовані як послідовність кроків: розвідка, первинне проникнення, закріплення, розширення привілеїв, ексфільтрація даних.

2.2. Методи аналізу та виявлення загроз

Методи виявлення загроз у системах електронної комерції можна поділити на такі групи:

- сигнатурні (правилові) методи;
- методи аномалійної детекції;
- методи машинного та глибинного навчання;
- поведінковий аналіз;
- контекстний та кореляційний аналіз подій.

2.2.1. Сигнатурне виявлення

Сигнатурний підхід ґрунтується на пошуку у подіях (логах, запитах) відомих шаблонів атак.

Приклади:

1. **Регулярні вирази (Regular Expressions)**
   Використовуються для пошуку підозрілих фрагментів у запитах:
   - патерни SQL-ін’єкцій (' OR 1=1, UNION SELECT, --, ; DROP TABLE тощо),
   - характерні елементи XSS (наприклад, <script>, onerror=),
   - підозрілі параметри у рядках запиту.
2. **YARA-правила**
   Формальний опис сигнатур, що включає:
   - константні рядки;
   - регулярні вирази;
   - логічні умови.

3. **Правила WAF та IDS/IPS**

Набори умов, які описують підозрілі HTTP-запити, аномальні заголовки, заборонені методи.

Переваги:

- висока точність для вже відомих типів атак;
- прозорість та пояснюваність роботи.

Недоліки:

- не виявляють нові (zero-day) атаки;
- потребують постійного оновлення бази сигнатур;
- чутливість до обфускації та модифікацій атак.

2.2.2. Методи аномалійної детекції

Мета – виявляти події, які істотно відрізняються від типових (нормальних), навіть якщо сигнатури цих атак невідомі.

1. **Статистичні методи**

a) Z-score:

$$z = \frac{x - \mu}{\sigma}, (2.1)$$

де x – поточне значення метрики, μ – середнє, σ – стандартне відхилення. Значення з великим $|z|$ (наприклад, > 3) розглядаються як аномальні.

b) IQR (Interquartile Range): Обчислюються перший і третій квартилі Q_1, Q_3 та інтерквартильний розмах:

$$IQR = Q_3 - Q_1. (2.2)$$

Аномальними вважаються значення поза межами:

$$[Q_1 - 1{,}5 \cdot IQR, Q_3 + 1{,}5 \cdot IQR]. (2.3)$$

c) EWMA (Exponential Weighted Moving Average) Використовується для згладжування часових рядів та виявлення різких змін.

2. **Алгоритми аномалійної детекції**

- **Isolation Forest** – ізоляція аномалій за допомогою випадкових розбиттів простору ознак.
- **Local Outlier Factor (LOF)** – оцінка локальної щільності даних; точки з низькою щільністю вважаються аномаліями.
- **One-Class SVM** – навчання на «нормальних» даних з подальшим виявленням відхилень.
- **Autoencoders** – нейронні мережі, що навчаються відтворювати нормальні дані; велика похибка реконструкції вказує на аномалію.

Переваги:

- здатність виявляти невідомі типи атак;
- можливість роботи без явної розмітки атак на етапі навчання.

Недоліки:

- більша кількість хибних тривог;
- потреба обережного налаштування порогів.

2.2.3. Методи машинного та глибинного навчання

Методи машинного навчання дозволяють будувати моделі, що автоматично вчаться розрізняти нормальну та шкідливу активність на основі даних.

1. **Класичні алгоритми (supervised learning)**

- логістична регресія;
- дерева рішень;
- Random Forest, Gradient Boosting, XGBoost;
- SVM.

Застосування:

- класифікація окремих запитів (нормальний / шкідливий);
- виявлення шахрайських транзакцій.

2. **Глибинні нейронні мережі**

- **LSTM/GRU** – для аналізу послідовностей логів, дій користувачів;
- **CNN** – для виявлення структурних патернів у представленні даних;

- **Autoencoders** – для аномалійної детекції.

3. **Гібридні та ансамблеві підходи**
   Комбінування сигнатурних, статистичних та ML-методів часто забезпечує кращі результати, ніж використання однієї техніки.

2.2.4. Поведінковий аналіз (UEBA)

Поведінковий аналіз фокусується на побудові профілю «нормальної» поведінки користувача або сутності (сервер, сервіс) та виявленні відхилень.

Ознаки:

- географія входів;
- типи пристроїв та браузерів;
- характерні часові шаблони активності;
- типова сума та частота замовлень;
- стандартні послідовності дій.

Відхилення від таких профілів можуть сигналізувати про компрометацію облікового запису або шахрайство.

2.2.5. Контекстний та кореляційний аналіз подій

Контекстний аналіз розглядає послідовності та взаємозв’язки між подіями з різних джерел (веб-сервер, БД, WAF, IDS, ОС, платіжні шлюзи).

Приклади:

- послідовність «сканування портів → підозрілі логіни → зміна платіжних реквізитів»;
- одночасні підозрілі дії з одного діапазону IP;
- синхронні аномалії у різних підсистемах.

Кореляція окремих сигналів дозволяє формувати цілісну картину інциденту.

2.3. Архітектурні підходи до виявлення та реагування на загрози

Сучасні рішення реалізують багаторівневий підхід:

- **SIEM (Security Information and Event Management)** – централізований збір і аналіз логів;
- **IDS/IPS** – виявлення та профілактика вторгнень;
- **WAF** – захист веб-додатків;
- **SOAR** – оркестрація та автоматизоване реагування.

2.3.1. SIEM-системи

Функції [24]:

- збір логів з різних джерел;
- нормалізація та уніфікація;
- кореляція подій;
- формування алертів;
- аналітика та звітність.

2.3.2. IDS/IPS

- Network IDS/IPS – аналіз трафіку, виявлення атак на рівні мережевих протоколів;
- Host-based IDS/IPS – контроль подій на конкретних хостах (процеси, файли, реєстр тощо).

2.3.3. WAF

Розташовується між клієнтом і веб-сервером, фільтрує HTTP/HTTPS-трафік, виявляє атакуючі запити й може блокувати їх ще до досягнення застосунка.

2.3.4. SOAR

Забезпечує:

- реалізацію playbook-сценаріїв реагування (блокування IP, блокування акаунта, повідомлення користувача);
- інтеграцію з SIEM, WAF, IDS;
- автоматизацію рутинних дій аналітиків безпеки.

2.4. Методи нейтралізації та протидії загрозам

Методи протидії поділяються на:

- превентивні (запобігання);
- детективні (виявлення);
- коректувальні (усунення наслідків);
- організаційні (політики, процеси, навчання).

2.4.1. Превентивні заходи

- криптографічний захист даних (AES, TLS, хешування паролів);
- безпечне програмування (validation, sanitization, parametrized queries);
- контроль доступу (RBAC, MFA);
- сегментація мережі, мінімізація поверхні атаки.

2.4.2. Детективні заходи

- логування та моніторинг ключових подій;
- системи IDS/IPS, SIEM, WAF;
- аналітика поведінки користувачів.

2.4.3. Коректувальні заходи

- блокування IP-адрес та облікових записів;
- скасування підозрілих транзакцій;
- відкат змін, відновлення з резервних копій.

2.4.4. Організаційні заходи

- політики паролів, доступу, резервного копіювання;
- регламенти реагування на інциденти;
- навчання персоналу та користувачів.

2.5 Математичні моделі та формалізація загроз та механізмів захисту

Для глибшого розуміння і можливості кількісного аналізу загроз та ефективності захисту доцільно використовувати математичні моделі. Такі моделі дозволяють:

- кількісно оцінювати ризики;
- обирати оптимальні стратегії захисту;
- прогнозувати ефективність систем виявлення;
- оптимізувати розподіл ресурсів на безпеку.

2.5.1. Основні поняття

Загрозою називається потенційна подія чи дія, яка може завдати шкоди інформаційним активам системи. Формально загрозу можна описати набором атрибутів:

$$Threat = (T, V, I, P), (2.4)$$

де T – тип загрози, V – множина вразливостей, I – вплив на CIA, P – ймовірність того, що загроза буде реалізована.

Ризик – це добуток ймовірності реалізації загрози на величину потенційного збитку:

$$R(t) = P(t) \times I(t), (2.5)$$

де $P(t)$ – ймовірність настання загрози t у часовому інтервалі $I(t)$ – величина впливу (збитків) у грошових одиницях або умовних одиницях серйозності.

Вразливістю називається слабкість у системі, яка може бути експлуатована для реалізації загрози:

$$Vulnerability = \left(V_{type}, V_{severity}, V_{eploitability}\right), (2.6)$$

де V_{type} – тип вразливості (наприклад, недостатня валідація вхідних даних), $V_{severity}$ – серйозність вразливості (за CVSS: 0-10), $V_{eploitability}$ – легкість експлуатації.

2.5.2. Модель сукупного ризику

$$R_{total} = \sum_{i=1}^{n} P_i \times I_i \times E_i\,, (2.7)$$

де n – кількість ідентифікованих загроз, P_i – ймовірність реалізації загрози i, I_i – вплив загрози на систему, E_i – коефіцієнт підсилення для критичних загроз.

2.5.3. Модель впливу на CIA-тріаду

Вектор впливу:

$$\vec{I_i} = \left(I_C^i, I_{Int}^i, I_A^i\right), (2.8)$$

де I_C^i – вплив на конфіденційність (0-1), I_{Int}^i – вплив на цілісність (0-1), I_A^i – вплив на доступність (0-1).

Для SQL-ін’єкцій: $\vec{I} = (0.9, 0.8, 0.3)$ – високий вплив на конфіденційність та цілісність, але низький на доступність.

Для DDoS: $\vec{I} = (0.0, 0.0, 1.0)$ – вплив лише на доступність.

з урахуванням ваг:

$$R_i = P_i \times \left(w_C I_C^i + w_{Int} I_{Int}^i + w_A I_A^i\right), (2.9)$$

$$w_C + w_{Int} + w_A = 1. (2.10)$$

2.6 Класифікація загроз електронної комерції

У таблиці 2.1 продемонстровано інформацію про кожен з видів атак, їх категорії, вплив та інше:

Таблиця 2.1 Класифікація загроз

№	Тип атаки	Категорія	Вектор	Вплив на CIA	Складність	Вірогідність
1	SQL Injection	Веб-додаток	Вхідні параметри	C, Int	Низька	Висока
2	XSS	Веб-додаток	DOM/HTML	C, Int	Низька	Висока
3	CSRF	Веб-додаток	Запит від браузера	Int	Низька	Середня
4	Path Traversal	Веб-додаток	URL параметри	C, Int	Низька	Середня
5	RCE	Веб-додаток	Вхідні дані	C, Int, A	Середня	Низька (з патчами)
6	DDoS (HTTP)	Мережа	Масовані запити	A	Низька	Висока
7	DDoS (SYN)	Мережа	TCP handshake	A	Низька	Висока
8	MITM	Мережа	Перехопленн я трафіку	C, Int	Середня	Середня
9	DNS Spoofing	Мережа	DNS запити	C	Середня	Низька (з DNSSEC)

10	Brute Force	Обліковий запис	Перебір паролю	Int	Дуже низька	Висока
11	Credential Stuffing	Обліковий запис	Утечені дані	Int	Дуже низька	Висока
12	Account Takeover	Обліковий запис	Комбінована	C, Int, A	Середня	Середня
13	Card Fraud	Платіж	Крадені дані	C, Int	Дуже низька	Висока
14	Phishing	Соціальна	Email/SMS	C, Int	Дуже низька	Висока
15	Insider Threat	Організаційна	Внутрішній доступ	C, Int, A	Середня	Низька

Розділ 3. Проєктування архітектури системи виявлення та нейтралізації загроз

3.1. Загальні принципи та цілі проєктування системи

Проєктування системи виявлення та нейтралізації загроз для безпеки даних у системах електронної комерції ґрунтується на комплексному підході, що враховує теоретичні основи, розглянуті у попередньому розділі, а також практичні обмеження сучасних корпоративних інфраструктур.

Основними архітектурними цілями є:

1. **Комплексність захисту:** Система повинна комбінувати детерміновані методи виявлення (сигнатурний аналіз) з імовірнісними (виявлення аномалій, машинне навчання) для максимального покриття як відомих (known threats), так і невідомих атак (zero-day exploits).
2. **Масштабованість (Scalability):** Архітектура повинна забезпечувати горизонтальне масштабування для обробки великих обсягів подій від розподіленої системи електронної комерції без деградації продуктивності.
3. **Реактивність (Real-time response):** Час від моменту виявлення загрози до ініціації автоматизованого реагування не повинен перевищувати 1 секунду, що є критичним для запобігання витоку даних.
4. **Відмовостійкість (Resilience):** Система повинна зберігати працездатність та цілісність даних навіть у випадку виходу з ладу окремих компонентів або вузлів.
5. **Інтероперабельність:** Архітектура повинна передбачати просту інтеграцію з існуючою інфраструктурою організації (SIEM, WAF, Identity Providers) через стандартизовані інтерфейси.
6. **Аудійованість:** Усі дії системи, від отримання логу до автоматичної реакції, повинні реєструватися для забезпечення можливості подальшого розслідування інцидентів та проведення аудиту.

3.1.1. Основні вимоги до системи

На основі аналізу загроз та вимог міжнародних стандартів (ISO/IEC 27001, PCI DSS, OWASP ASVS) сформульовано наступні вимоги:

Функціональні вимоги (Functional Requirements – FR):

- **FR1:** Система повинна здійснювати централізований збір журналів подій (логів) з веб-сервера, WAF, IDS/IPS, бази даних та операційної системи.
- **FR2:** Система повинна виконувати нормалізацію різнорідних форматів логів до єдиної канонічної моделі даних.
- **FR3:** Система повинна виявляти відомі атаки, використовуючи сигнатурний аналіз (наприклад, правила YARA).
- **FR4:** Система повинна виявляти аномалії у поведінці користувачів та мережевому трафіку методами машинного навчання.
- **FR5:** Система повинна класифікувати виявлені інциденти за ступенем ризику (критичний, високий, середній, низький).
- **FR6:** Система повинна генерувати сповіщення (алерти) та надсилати їх адміністраторам безпеки в режимі реального часу.
- **FR7:** Система повинна виконувати автоматизовані сценарії реагування (блокування IP-адреси, завершення сесії, блокування облікового запису).
- **FR8:** Система повинна забезпечувати довгострокове зберігання всіх подій та інцидентів для ретроспективного аналізу.
- **FR9:** Система повинна надавати веб-інтерфейс (Dashboard) для моніторингу стану безпеки та управління налаштуваннями.

Нефункціональні вимоги (Non-Functional Requirements – NFR):

- **NFR1:** Час обробки однієї події (Event Processing Latency): ≤ 100 мс.
- **NFR2:** Час реакції на критичну загрозу: ≤ 1 с.
- **NFR3:** Доступність системи (Availability): $\geq 99.5\%$.
- **NFR4:** Пропускна здатність: обробка $\geq 10{,}000$ подій на секунду (EPS).
- **NFR5:** Повнота виявлення атак (Recall): $\geq 85\%$.
- **NFR6:** Влучність виявлення атак (Precision): $\geq 85\%$.

- **NFR7:** Коефіцієнт хибнопозитивних спрацьовувань (False Positive Rate): ≤ 15%.

3.2 Архітектура системи на рівні компонентів

3.2.1 Загальна структурна схема

Архітектура системи базується на патерні конвеєрної обробки даних (Data Pipeline) і складається з 7 основних компонентів (рис. 3.1):

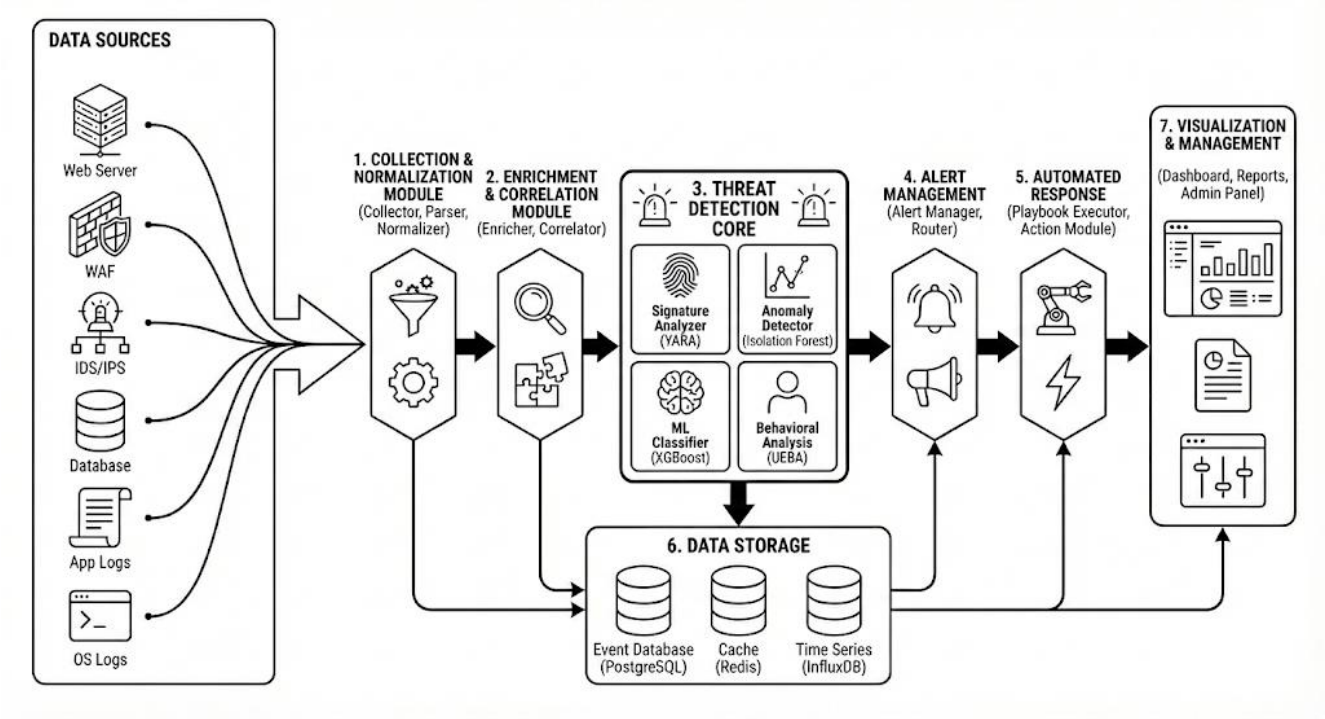


Рисунок 3.1 – Структурна схема системи

3.2.2 Детальний опис компонентів

Компонент 1: Збір та нормалізація журналів (Log Collection & Normalization)

Вхідні дані:

- HTTP запити/відповіді (Apache, Nginx);
- Логи веб-екрану (WAF – ModSecurity, Cloudflare);
- Логи систем виявлення вторгнень (Snort, Suricata, Zeek);

- Аудит бази даних (PostgreSQL audit logs);
- Логи прикладного рівня (application.log);
- Системні журнали (syslog, Windows Event Log).

Функціональність:

- Асинхронний прийом потоків даних через черги повідомлень (Kafka або RabbitMQ).
- Парсинг різнорідних форматів (JSON, CSV, Syslog, XML).
- Нормалізація даних до єдиної схеми: {timestamp, source_ip, destination_ip, event_type, severity, raw_data}.
- Фільтрація шумів та дедублікація записів.

Вихід: Нормалізовані події у форматі JSON, передані до внутрішньої черги подій.

Компонент 2: Збагачення та кореляція подій (Event Enrichment & Correlation)

Вхідні дані: Нормалізовані події, бази GeoIP, репутаційні списки (Threat Intelligence feeds), профілі користувачів.

Функціональність:

- **Збагачення:** Додавання географічних координат до IP-адрес, визначення провайдера (ISP).
- **Перевірка репутації:** Зіставлення IP-адрес та доменів з "чорними списками" (Blacklists).
- **Кореляція:** Об'єднання розрізнених подій у логічні ланцюжки (сесії) в межах часового вікна (наприклад, 5 хвилин) за ключовими атрибутами (IP, UserID).

Технічна реалізація: Використання Redis для швидкого доступу до контекстних даних; Python-скрипти для логіки кореляції.

Компонент 3: Механізм виявлення загроз (Threat Detection Engine)

Цей компонент є серцем системи і використовує гібридний підхід.

Сигнатурний детектор (YARA)

Використовує попередньо визначені правила для пошуку відомих патернів атак у тілі запитів [14].

Приклад правила для виявлення SQL-ін'єкції:

```
rule SQLInjection_OR {
    meta:
        description = "Detects SQL injection with OR operator"
        severity = "high"
    strings:
        $s1 = "' OR '"
        $s2 = "' OR 1=1"
    condition:
        any of them
}
```

Детектор аномалій (Isolation Forest)

Використовує алгоритм "Ізолюючого лісу" для виявлення подій, що статистично відхиляються від норми.

Ознаки: частота запитів, розмір пакетів, коди відповідей сервера, час між запитами.

Математична модель:

Для кожного вектора ознак $x = (x_1, x_2, \dots, x_k)$, оцінка аномальності обчислюється як:

$$s(x, n) = 2^{-\frac{E[h(x)]}{c(n)}}, (3.1)$$

де $E[h(x)]$ – середнє значення довжини шляху в деревах ізоляції, $c(n)$ – нормалізуючий коефіцієнт (середня довжина шляху для n елементів). Якщо $s(x, n) > 0.5$\$, подія вважається аномальною.

ML Класифікатор (XGBoost)

Ознаки для моделі (40+ ознак):

1. **Мережеві ознаки:**
    - Джерелова IP адреса (закодована)
    - Порт призначення
    - Розмір пакету

- Тип протоколу

2. **Ознаки HTTP запиту:**
    - Довжина URL
    - Кількість параметрів
    - Наявність спеціальних символів
    - HTTP метод
    - User-Agent fingerprint
3. **Часові ознаки:**
    - Час доби
    - День тижня
    - Відхилення від типового часу користувача
4. **Поведінкові ознаки:**
    - Кількість запитів користувача за останню хвилину
    - Кількість помилок авторизації
    - Кількість звернень до адміністративних сторінок
    - Чи першого разу користувач з цієї IP
5. **Вміст запиту:**
    - SQL-подібні патерни (наявність SELECT, UNION тощо)
    - JavaScript коди (наявність <script>, onerror тощо)
    - Довжина найбільшого повторюваного підрядка

Архітектура моделі:

- Дерева глибини 5–7
- 100–200 дерев
- Learning rate: 0.1
- Min child weight: 1
- Gamma: 0

Вихід: Ймовірність того, що подія є атакою (від 0 до 1)

Аналітика поведінки користувачів та сутностей (UEBA)

Формує динамічні профілі користувачів та виявляє відхилення від їхньої типової поведінки.

Профіль користувача P_u включає: типові IP-адреси, час активності, геолокацію, типовий набір дій.

Для виявлення відхилення нової дії x від профілю використовується відстань Махаланобіса:

$$D = \sqrt{(x-\mu)^T \sum^{-1} (x-\mu)}, (3.2)$$

де μ – центроїд профілю, $\sum$ – коваріаційна матриця. Якщо D перевищує порогове значення θ, подія маркується як підозріла.

Компонент 4: Генерація сповіщень та маршрутизація

Функціональність:

- Ансамблювання (Ensembling): Обчислення інтегральної оцінки ризику на основі вердиктів усіх детекторів:

$$RiskScore = 0.3 \cdot I_{YARA} + 0.25 \cdot S_{Anomaly} + 0.35 \cdot P_{ML} + 0.1 \cdot S_{UEBA}. (3.3)$$

- **Класифікація:**
  - Критичний: Score > 0.85
  - Високий: 0.70 < Score ≤ 0.85
  - Середній: 0.50 < Score ≤ 0.70
  - Низький: Score ≤ 0.50
- **Маршрутизація:** Критичні сповіщення миттєво передаються через PagerDuty/Slack, інші – зберігаються в системі тікетів.

Компонент 5: Автоматизоване реагування

Виконання заздалегідь підготовлених сценаріїв (Playbooks) залежно від типу загрози.

- **Playbook "SQL Injection":**

Умова: (YARA_SQLi_match OR XGBoost_prob > 0.9) AND Risk_Score > 0.7

Дії:

1. Заблокувати IP на 30 хвилин у WAF
2. Логувати повну послідовність запитів
3. Якщо користувач авторизований:

- Примусово завершити сесію
- Скинути пароль користувача
- Відправити повідомлення користувачу

4. Сповістити адміністратора
5. Запустити поглиблений аналіз запису у БД

- **Playbook "DDoS":**

Умова: Anomaly_score > 0.7 AND requests_per_min > 5000

Дії:

1. Активувати DDoS захист у WAF
2. Перенаправити трафік через Cloudflare
3. Запустити rate limiting (max 100 запитів/хв на IP)
4. Зібрати статистику атаки (IP, User-Agent, URL)
5. Сповістити адміністратора
6. Розповсюдити IP у список black-list

- **Playbook "Account Takeover":**

Умова: (UEBA_anomaly > 0.8) OR (Brute_force_detected)

Дії:

1. Заблокувати обліковий запис на 24 години
2. Примусово завершити всі сесії користувача
3. Зберегти всі логи входу
4. Відправити користувачеві повідомлення про найдавніший вхід
5. Вимагати двофакторної автентифікації для наступного входу
6. Сповістити команду fraud prevention

Компонент 6: Сховище даних

- **PostgreSQL:** Основне сховище для структурованих даних (користувачі, метадані подій, правила).

Схема таблиці (ER-діаграма) зображена на рисунку 3.2:

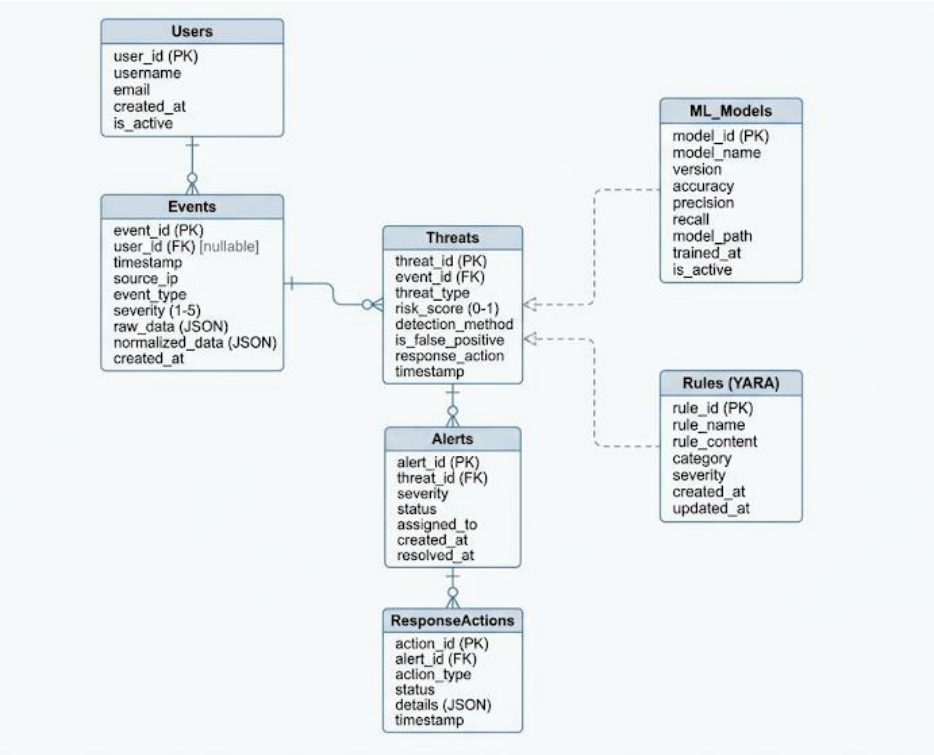


Рисунок 3.2 – ER-діаграма

- **InfluxDB:** База даних часових рядів для зберігання метрик продуктивності та статистики атак.

InfluxDB для часових рядів:

  – Метрики: events_per_second, threats_detected_per_hour, response_time_ms
  – Теги: severity, event_type, detection_method
  – Retention: 90 днів

- **Redis:** Оперативний кеш для профілів користувачів, репутаційних списків та стану сесій.

Redis кешування:

  – user_profile:{user_id} – профіль користувача (TTL: 24 години)
  – ip_reputation:{ip} – репутація IP (TTL: 12 годин)
  – rate_limit:{ip}:{endpoint} – лічильник запитів (TTL: 60 секунд)
  – active_alerts:{severity} – активні алерти (TTL: 1 година)

Компонент 7: Інтерфейс користувача (UI)

Веб-додаток (React) для візуалізації статистики в реальному часі, управління правилами YARA, перегляду журналу інцидентів та налаштування політик безпеки.

Dashboard (React):

- Головна сторінка: Real-time графіки атак
- Сторінка алертів: Таблиця з фільтрацією та сортуванням
- Сторінка користувачів: Пошук та перегляд профілів
- Сторінка статистики: Графіки тренів, хітмапи часу атак
- Сторінка налаштувань: Управління правилами YARA, порогами, playbook-ами
- Сторінка звітів: Завантаження CSV звітів

3.3 Діаграма потоків даних (DFD-діаграма)

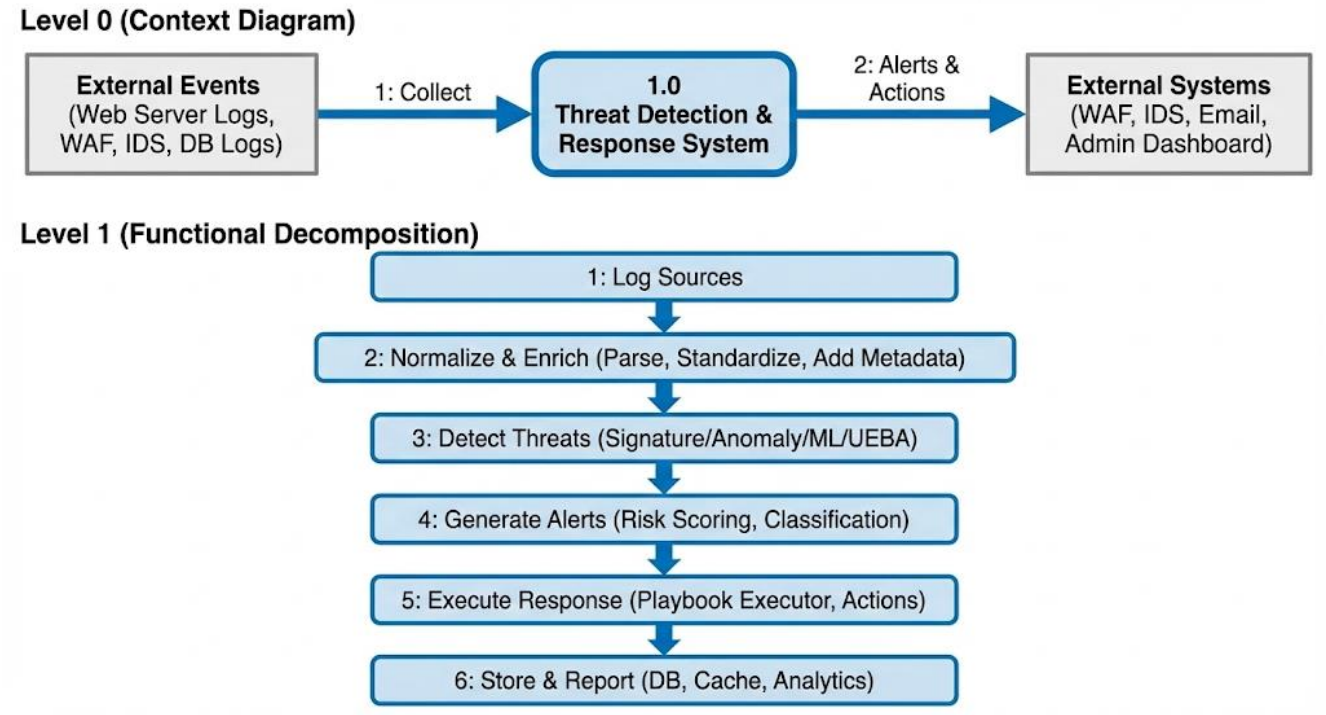


Рисунок 3.3 – Діаграма потоків даних

На рисунку 3.3 показано контекстну діаграму (DFD рівня 0) та діаграма потоку даних рівня 1 для системи виявлення загроз і реагування на них.

Діаграма рівня 0 визначає межі системи та її взаємодію високого рівня із зовнішніми об'єктами.

Основні вхідні дані надходять із джерел даних журналів, які генерують необроблені журнали та події безпеки з різних компонентів інфраструктури, таких

як веб-сервери, брандмауери веб-додатків (WAF), системи виявлення вторгнень (IDS) та бази даних. Центральний процес, 1.0 Система виявлення та реагування на загрози, приймає цей неоднорідний потік даних.

Система обробляє ці дані та генерує вихідні дані для приймачів реагування та моніторингу. Ці вихідні дані включають практичні сповіщення, що передаються зовнішнім інструментам безпеки (наприклад, WAF, IDS для блокування), повідомлення електронною поштою для адміністраторів безпеки та візуалізовані дані, що заповнюють панель адміністратора для моніторингу в режимі реального часу.

Діаграма потоку даних рівня 1 детально відображає внутрішні підпроцеси обробки в системі виявлення загроз і реагування на них. Вона демонструє лінійну трансформацію даних від їхнього надходження до зберігання та реагування.

Конвеєр складається з п'яти послідовних функціональних процесів:

- Нормалізація та збагачення: необроблені дані з джерел журналів приймаються, аналізуються у стандартному форматі (нормалізація) та доповнюються контекстними метаданими (збагачення).
- 2.0 Виявлення загроз: нормалізовані події аналізуються за допомогою різних методів, включаючи зіставлення сигнатур, моделі виявлення аномалій, класифікатори машинного навчання та аналіз поведінки користувачів і об'єктів (UEBA), щоб ідентифікувати потенційні індикатори загроз.
- 3.0 Генерація сповіщень: виявлені індикатори оцінюються, класифікуються за ступенем серйозності та отримують оцінки ризику для генерації пріоритетних сповіщень.
- 4.0 Виконання відповіді: сповіщення з високим пріоритетом запускають Playbook Executor, який ініціює автоматизовані захисні дії через зовнішні системи.
- 5.0 Зберігання та звітність: всі оброблені дані, включаючи згенеровані сповіщення та журнали виконаних дій, зберігаються в основній базі даних подій та аналітичному кеші. Цей сховище даних згодом подає інформацію до механізму звітності та аналітики для візуалізації на інформаційній панелі.

3.4 UML Use Case Діаграма

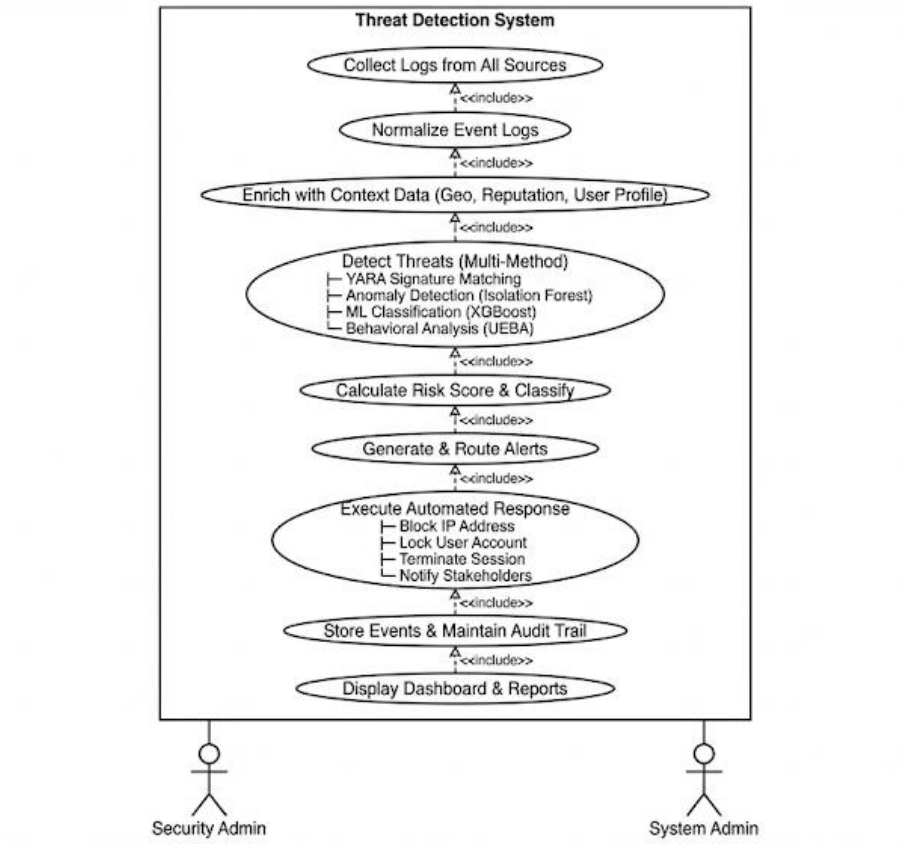


Рисунок 3.4 – Use Case Діаграма

На рисунку 3.4 зображена UML діаграма прецедентів (Use Case Diagram), яка візуалізує функціональність та архітектуру Системи виявлення загроз (Threat Detection System).

Вона демонструє основні процеси системи у вигляді послідовного конвеєра обробки даних та визначає користувачів, які взаємодіють із цією системою.

Опис компонентів діаграми:

1. Межі системи (System Boundary)

Великий прямокутник з назвою **"Threat Detection System"** визначає межі аналізованої системи. Усі функціональні процеси (прецеденти), розташовані всередині цього прямокутника, є внутрішніми функціями системи.

2. Актори (Actors)

В нижній частині діаграми, за межами системи, зображені дві дійові особи (актори), які взаємодіють із нею:

- **Security Admin (Адміністратор безпеки):** Фахівець, відповідальний за оперативний моніторинг безпеки, аналіз виявлених загроз та управління інцидентами.
- **System Admin (Системний адміністратор):** Фахівець, відповідальний за підтримку працездатності самої системи та її інфраструктури.

Обидва актори мають асоціативний зв'язок із системою в цілому, що вказує на те, що вони є користувачами результатів роботи всього конвеєра.

3. Прецеденти (Use Cases) та їхні зв'язки

У центрі діаграми розташовано вертикальний стек овалів, які представляють прецеденти (варіанти використання). Вони з'єднані пунктирними стрілками зі стереотипом <<include>> (включення).

Стрілки спрямовані знизу вгору. Це означає, що виконання нижнього прецеденту залежить від виконання верхнього, або включає його як обов'язкову частину. Така структура фактично моделює послідовний конвеєр обробки даних (data processing pipeline), де кожен наступний крок базується на результатах попереднього.

Розглянемо цей конвеєр (згори донизу, як рухаються дані):

1. **Collect Logs from All Sources (Збір логів з усіх джерел):** Базовий процес, з якого починається робота – агрегація "сирих" даних з різних систем.
2. **Normalize Event Logs (Нормалізація логів подій):** Приведення зібраних логів до єдиного стандартного формату. Цей крок включає попередній (збір).
3. **Enrich with Context Data (Збагачення контекстними даними):** Додавання додаткової інформації до нормалізованих подій (геолокація, репутація IP, профіль користувача).
4. **Detect Threats (Multi-Method) (Виявлення загроз мультиметодом):** Комплексний прецедент, який включає декілька методів аналізу:
   - YARA Signature Matching (Сигнатурний аналіз YARA)
   - Anomaly Detection (Isolation Forest) (Виявлення аномалій)
   - ML Classification (XGBoost) (Класифікація машинним навчанням)

   - Behavioral Analysis (UEBA) (Поведінковий аналіз користувачів та сутностей)
5. **Calculate Risk Score & Classify (Розрахунок оцінки ризику та класифікація):** На основі виявлених індикаторів система оцінює рівень небезпеки події.
6. **Generate & Route Alerts (Генерування та маршрутизація сповіщень):** Створення алертів для подій з високим ризиком та їх надсилання відповідним отримувачам.
7. **Execute Automated Response (Виконання автоматизованого реагування):** Прецедент, що описує автоматичні дії для блокування загрози:
   - Block IP Address (Блокування IP-адреси)
   - Lock User Account (Блокування облікового запису користувача)
   - Terminate Session (Завершення сесії)
   - Notify Stakeholders (Сповіщення зацікавлених сторін)
8. **Store Events & Maintain Audit Trail (Зберігання подій та ведення аудиторського сліду):** Усі оброблені дані та інформація про виконані дії зберігаються для історії та аудиту.
9. **Display Dashboard & Reports (Відображення дашборду та звітів):** Фінальний етап, який залежить від усіх попередніх. Це інтерфейс, через який актори (Адміни) отримують доступ до результатів роботи всієї системи.

3.5 UML Sequence Diagram (Послідовність подій при виявленні атаки)

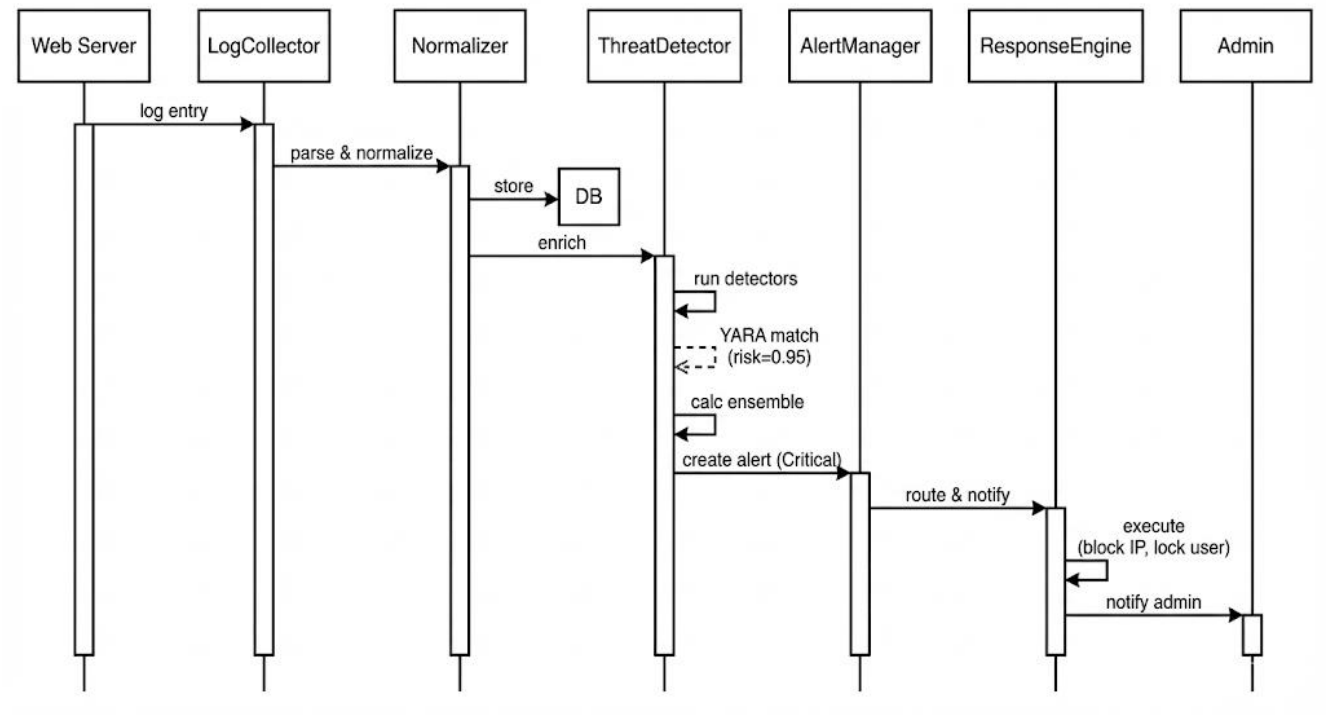


Рисунок 3.5 – Діаграма послідовності

На рисунку 3.5 зображено UML діаграму послідовності (Sequence Diagram). Вона деталізує динамічну взаємодію між компонентами системи в часі для конкретного сценарію: виявлення критичної загрози та автоматизованого реагування на неї.

Діаграма показує, як потік управління переходить від одного компонента до іншого, які повідомленнями вони обмінюються і в якій послідовності це відбувається.

Опис діаграми послідовності:

1. Учасники взаємодії

Вертикальні лінії представляють компоненти системи, залучені до цього сценарію:

- **Web Server (Веб-сервер):** Джерело "сирих" даних (подій).
- **LogCollector (Колектор логів):** Компонент, що приймає вхідний потік даних.
- **Normalizer (Нормалізатор):** Відповідає за парсинг та стандартизацію даних.

- **ThreatDetector (Детектор загроз):** Аналітичне ядро системи.
- **AlertManager (Менеджер сповіщень):** Керує життєвим циклом алертів.
- **ResponseEngine (Рушій реагування):** Виконує автоматичні дії.
- **Admin (Адміністратор):** Кінцевий користувач (людина), який отримує сповіщення.
- **DB (База даних):** Зовнішнє сховище (зображено як об'єкт, до якого звертаються).

2. Покроковий опис потоку (Scenario Walkthrough)

Процес читається згори донизу:

1. **Ініціація події:** Процес починається, коли Web Server надсилає повідомлення log entry (запис журналу) до LogCollector.
2. **Обробка та нормалізація:** LogCollector отримує дані та викликає Normalizer повідомленням parse & normalize для приведення логу до стандартного формату.
3. **Збереження та збагачення:**
   - Normalizer зберігає оброблені дані в базу даних, надсилаючи повідомлення store до об'єкта DB.
   - Після цього Normalizer передає дані далі для аналізу, надсилаючи повідомлення enrich (збагатити) до ThreatDetector.
4. **Аналіз загрози (Внутрішня обробка):**
   - ThreatDetector активується і виконує серію внутрішніх дій (self-calls). Спочатку він запускає детектори (run detectors).
   - Діаграма показує, що один із механізмів аналізу (наприклад, підсистема YARA) повернув позитивний результат: пунктирна стрілка YARA match (risk=0.95) вказує на виявлення загрози з високим рівнем ризику.
   - На основі цього результату ThreatDetector виконує внутрішній розрахунок загальної оцінки ризику (calc ensemble).
5. **Створення сповіщення:** Оскільки розрахований ризик високий, ThreatDetector надсилає повідомлення create alert (Critical) до AlertManager.

6. **Маршрутизація реагування:** AlertManager обробляє критичний алерт і ініціює реагування, надсилаючи повідомлення route & notify до ResponseEngine.
7. **Виконання реагування та сповіщення:**
   - ResponseEngine виконує автоматизовані дії для блокування загрози (внутрішній виклик execute (block IP, lock user)).
   - Паралельно ResponseEngine надсилає фінальне повідомлення notify admin безпосередньо Admin (людині), щоб поінформувати про інцидент та вжиті заходи.

3.6 Модель безпеки системи

Для захисту самої системи виявлення загроз передбачено наступні заходи:

1. **Автентифікація:** Використання JWT токенів з коротким терміном життя (1 година) та Refresh-токенів. Для адміністраторів обов'язкова двофакторна автентифікація (2FA).
2. **Контроль доступу (RBAC):**
   - Admin: Повний доступ до налаштувань та правил.
   - Analyst: Перегляд інцидентів, формування звітів.
   - System: Технічний обліковий запис для API.
3. **Шифрування:**
   - Дані в транзиті: TLS 1.3 для всіх з'єднань.
   - Дані у спокої: Шифрування AES-256 для конфіденційних полів у базі даних (PII, паролі).

Розділ 4. Програмна реалізація системи

4.1. Обґрунтування вибору інструментальних засобів розробки

Вибір засобів розробки є стратегічним рішенням, яке визначає життєвий цикл програмного продукту, його здатність до масштабування, легкість підтримки та швидкість виведення на ринок (Time-to-Market). У контексті даної дипломної роботи, де ключовими вимогами є гнучкість, швидкість розробки прототипу та висока читабельність коду, було проведено багатокритеріальний аналіз сучасних технологій.

4.1.1. Аналіз мови програмування Python

Для реалізації серверної логіки було обрано мову Python. Python є високорівневою інтерпретованою мовою загального призначення, яка за останні десятиліття стала стандартом де-факто у сферах веб-розробки, обробки даних (Data Science) та машинного навчання.

З технічної точки зору, Python забезпечує ефективну роботу з мережевими протоколами завдяки вбудованим бібліотекам та підтримці стандарту WSGI (Web Server Gateway Interface), що є критичним для веб-додатків. Хоча Python часто критикують за меншу швидкість виконання порівняно з компільованими мовами (C++, Go) через наявність Global Interpreter Lock (GIL), у контексті веб-серверів, де основним вузьким місцем є операції введення-виведення (I/O bound), а не обчислення процесора (CPU bound), продуктивність Python є більш ніж достатньою. Крім того, наявність розвиненої екосистеми пакетів (PyPI) дозволяє інтегрувати готові рішення для роботи з базами даних, кешуванням та серіалізацією даних, значно скорочуючи час розробки.

4.1.2. Порівняльний аналіз веб-фреймворків: Flask проти Django

Вибір фреймворку є наступним логічним кроком після вибору мови. У екосистемі Python існує "велика трійка" веб-фреймворків: Django, Flask та FastAPI. Кожен з них має свою філософію та сферу застосування.

Обґрунтування вибору Flask:

1. **Архітектурна чистота та контроль:** Django нав'язує розробнику жорстку структуру проєкту та використання власної ORM. У рамках дипломної роботи, де часто необхідно продемонструвати розуміння низькорівневих механізмів (наприклад, реалізація власного Rate Limiter замість використання готового middleware Django), Flask надає необхідну свободу. Розробник сам вирішує, як структурувати додаток, які бібліотеки підключати і як обробляти запити.
2. **Мікросервісна орієнтація:** Сучасна веб-розробка рухається в бік мікросервісної архітектури. Flask ідеально підходить для створення легковагових сервісів, які виконують одну конкретну функцію (у даному випадку – обробку запитів з обмеженням частоти). Він не тягне за собою зайвий вантаж невикористовуваного коду, що зменшує споживання пам'яті та прискорює "холодний старт" додатку.
3. **Навчальний потенціал:** Використання Flask вимагає від розробника глибшого розуміння роботи протоколу HTTP, управління сесіями, маршрутизації та контексту запиту, оскільки багато речей не приховані "під капотом", як у Django. Це робить його ідеальним інструментом для кваліфікаційної роботи, демонструючи технічну компетентність студента.
4. **Екосистема розширень:** Незважаючи на мінімалізм ("micro" у назві означає не малий функціонал, а мале ядро), Flask має потужну систему розширень (Extensions). Для будь-якої задачі (робота з БД, валідація форм, авторизація) існують перевірені бібліотеки (Flask-SQLAlchemy, Flask-WTF, Flask-Login), які інтегруються у проєкт "безшовно", але лише тоді, коли вони дійсно потрібні.

4.2. Архітектурне проєктування програмного додатка

Архітектура розробленого додатку базується на принципах REST (Representational State Transfer) та мікросервісного підходу. Додаток спроєктовано як автономний сервіс, що приймає HTTP-запити, обробляє їх згідно з бізнес-логікою та повертає відповіді у форматі JSON (JavaScript Object Notation).

Основні компоненти:

1. ThreatMonitor – система виявлення загроз
2. Security Middleware – перехоплення атак
3. Database Models – моделі користувачів, товарів, замовлень
4. Routes – 10 маршрутів для функціональності
5. HTML Template – єдиний шаблон для всіх сторінок

4.2.1. Структура проєкту та життєвий цикл запиту

Організація коду проєкту виконана з дотриманням принципів модульності та розділення відповідальності (Separation of Concerns).

- **Точка входу (**app.py **/** wsgi.py**):** Тут відбувається ініціалізація екземпляру класу Flask. Використовується патерн Application Factory, що дозволяє створювати додаток з різними конфігураціями (Development, Testing, Production) без зміни основного коду. Це критично важливо для забезпечення якості коду та проведення автоматизованого тестування.
- **Маршрутизація (Routing):** Flask використовує декоратори (наприклад, @app.route('/api/resource')) для зв'язування URL-адрес з функціями-контролерами (Views). Маршрутизація реалізована з використанням Blueprints – механізму Flask для групування пов'язаних маршрутів у окремі модулі. Це дозволяє масштабувати додаток, додаючи нові версії API або функціональні блоки без загрожування основному файлу додатку.

- **Механізм Middleware:** Проміжне програмне забезпечення відіграє ключову роль у реалізованій системі. Саме на цьому рівні інтегрується логіка Rate Limiting. Кожен вхідний запит перехоплюється спеціальним хуком before_request. На цьому етапі відбувається ідентифікація клієнта (за IP-адресою або API-ключем) та перевірка лімітів. Якщо ліміт перевищено, запит відхиляється ще до того, як він дійде до "важкої" бізнес-логіки або бази даних, що значно економить ресурси сервера.
- **Обробка відповідей:** Функції-контролери повертають об'єкти Python, які автоматично серіалізуються у JSON за допомогою jsonify. Важливим елементом є уніфікована обробка помилок (Error Handling). Спеціальні обробники (@app.errorhandler) перехоплюють винятки та повертають клієнту стандартизовані повідомлення з відповідними HTTP-кодами (400, 404, 429, 500).

4.2.2. Інтеграція з WSGI-сервером

Flask містить вбудований сервер для розробки, який не призначений для використання у продакшн-середовищі через однопотоковість та низьку продуктивність. Тому архітектура системи передбачає розгортання додатку під управлінням WSGI-сервера, такого як Gunicorn або uWSGI. WSGI (Web Server Gateway Interface) – це стандарт взаємодії між веб-сервером та веб-додатком Python. У такій конфігурації Gunicorn бере на себе керування пулом робочих процесів (workers), обробку вхідних з'єднань та балансування навантаження між процесами, тоді як Flask-додаток фокусується виключно на логіці обробки запитів. Це дозволяє системі ефективно використовувати багатоядерні процесори та обробляти сотні паралельних запитів.

4.3. Алгоритмічне забезпечення: Sliding Window Rate Limiting

Центральним елементом програмної реалізації, що забезпечує стабільність та захист системи від перевантажень, є алгоритм обмеження частоти запитів (Rate Limiting). У сучасному вебі контроль трафіку є необхідною умовою функціонування будь-якого публічного API для захисту від DDoS-атак, брутфорсу паролів та недобросовісного використання ресурсів (scrapers) [16].

4.3.1. Аналіз існуючих алгоритмів Rate Limiting

Перед вибором конкретного алгоритму було проаналізовано найбільш поширені підходи:

1. **Token Bucket (Маркерний кошик):** Алгоритм базується на аналогії з відром, в яке з певною швидкістю додаються токени. Кожен запит забирає токен. Дозволяє короткочасні сплески трафіку (bursts) до розміру ємності відра.
2. **Leaky Bucket (Діряве відро):** Запити надходять у чергу (відро) і обробляються з фіксованою швидкістю. Згладжує трафік, перетворюючи сплески на рівномірний потік, але може збільшувати латентність через чергу.
3. **Fixed Window (Фіксоване вікно):** Найпростіший підхід. Час ділиться на фіксовані вікна (наприклад, 1 хвилина). Лічильник скидається на початку кожного вікна. Головний недолік – проблема "кордону вікон": якщо клієнт зробить повну норму запитів в останню секунду однієї хвилини і ще одну норму в першу секунду наступної, сервер отримає подвійне навантаження за короткий проміжок часу.
4. **Sliding Window Log (Журнал ковзного вікна):** Зберігає часову мітку кожного запиту. Для перевірки підраховується кількість запитів у інтервалі [теперішній час - ширина вікна, теперішній час]. Це дуже точний метод, але надзвичайно вимогливий до пам'яті, оскільки потрібно зберігати дані про кожен запит.

4.3.2. Обґрунтування та математична модель Sliding Window Algorithm

Для реалізації було обрано алгоритм **Sliding Window Counter (Лічильник ковзного вікна)**. Цей алгоритм є гібридним рішенням, що поєднує низькі вимоги до пам'яті (як у Fixed Window) з точністю згладжування трафіку (наближеною до Sliding Window Log).

Суть алгоритму полягає у апроксимації кількості запитів у поточному "ковзному" вікні на основі даних поточного та попереднього фіксованих вікон. Це дозволяє уникнути проблеми подвійного навантаження на стику вікон без необхідності зберігати мітку кожного окремого запиту.

Математична модель розрахунку поточної оціночної частоти запитів ($Rate$) виглядає наступним чином:

$$Rate \approx R_{prev} \times \left(\frac{T_{window} - T_{elapsed}}{T_{window}}\right) + R_{curr}, (4.1)$$

де:

- R_{prev} – кількість запитів, зафіксованих у попередньому часовому вікні;
- R_{curr} – кількість запитів, зафіксованих у поточному часовому вікні;
- T_{window} – загальна тривалість часового вікна (наприклад, 60 секунд);
- $T_{elapsed}$ – час, що минув від початку поточного фіксованого вікна (наприклад, якщо зараз 12:00:15, а вікно – хвилина, то $T_{elapsed}$ = 15 с).

Коефіцієнт $\frac{T_{window}-T_{elapsed}}{T_{window}}$ представляє собою "вагу" попереднього вікна. Чим більше часу пройшло у поточному вікні, тим меншою стає вага попереднього вікна і тим більшим стає вклад поточного лічильника.

Приклад роботи:

Припустимо, ліміт – 100 запитів на хвилину.

- У попередню хвилину (наприклад, 12:00 – 12:01) було зроблено 80 запитів (R_{prev} = 80).
- Зараз 12:01:15 ($T_{elapsed}$ = 15 с, T_{window} = 60 с).
- У поточну хвилину вже зроблено 10 запитів R_{curr} = 10).

Розрахунок:

Ваговий коефіцієнт попереднього вікна: (60 - 15) / 60 = 0.75.

Оцінка Rate: 80 × 0.75 + 10 = 60 + 10 = 70.

Результат: 70 < 100, запит дозволено.

Такий підхід гарантує, що навіть при переході через межу хвилин сплеск трафіку буде коректно враховано, і система не пропустить зайві запити.

4.3.3. Програмна реалізація класу SlidingWindow

На основі проаналізованих підходів , програмна реалізація інкапсульована у класі SlidingWindow, який забезпечує зберігання стану та логіку перевірки. Клас оперує наступними атрибутами:

- capacity: Максимально допустима кількість запитів.
- time_unit: Одиниця часу вікна в секундах.
- cur_time: Часова мітка початку поточного вікна.
- cur_count: Лічильник запитів у поточному вікні.
- prev_count: Лічильник запитів у попередньому вікні.

Логіка методу handle(packet) (або check_limit):

1. Отримується поточний системний час (now = time()).
2. Перевіряється умова зміщення вікна: якщо now - cur_time > time_unit, відбувається ротація. prev_count отримує значення cur_count, cur_count скидається в 0, а cur_time оновлюється на поточний час. Це забезпечує ефект "ковзання" дискретними кроками.
3. Виконується розрахунок оціночної кількості запитів за наведеною вище формулою.
4. Якщо розраховане значення менше за capacity, лічильник cur_count інкрементується, і повертається позитивний результат (True).

5. Якщо значення перевищує ліміт, повертається відмова (False), і запит блокується без інкременту лічильника (або з інкрементом, залежно від стратегії покарання порушників).

Ця реалізація є ефективною з точки зору пам'яті, оскільки для кожного клієнта (ключа) зберігається лише кілька числових значень (O(1) пам'яті), на відміну від логів, де споживання пам'яті лінійно залежить від кількості запитів (O(N)). Часова складність перевірки також є константною O(1), що робить цей алгоритм ідеальним для високонавантажених систем.

4.4 Модуль безпеки

4.4.1 Threat monitor class

```
class ThreatMonitor:  1 usage
    def __init__(self):
        self.blacklist = set()        # Множина заблокованих IP
        self.request_history = {}     # Історія запитів по IP
        self.logs = []                # Журнал інцидентів

    def log_incident(self, ip, attack_type, payload):  2 usages
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        self.logs.append({
            'time': timestamp,
            'ip': ip,
            'type': attack_type,
            'payload': payload[:50]
        })

    def is_ip_blocked(self, ip):  1 usage
        return ip in self.blacklist

    def block_ip(self, ip):  1 usage
        self.blacklist.add(ip)
```

Рисунок 4.1 – Модуль безпеки

Функції, що зображені на рисунку 4.1:

log_incident(ip, attack_type, payload)

- Призначення: Логування атаки
- Параметри:
  - ip - IP адреса атакуючого
  - attack_type - тип атаки (SQL Injection, XSS, DDoS)
  - payload - корисна навантаження (перші 50 символів)
- Результат: Додає запис в список self.logs

is_ip_blocked(ip)

Призначення: Перевірка чи IP заблокована

Повертає: True якщо IP в чорному списку, інакше False

block_ip(ip)

- Призначення: Додавання IP в чорний список
- Результат: IP додається в self.blacklist

4.4.2 Attack signatures

```
ATTACK_SIGNATURES = {
    'SQL Injection':
                    [r"(?i)union\s+select",
                     r"(?i)or\s+1=1",
                     r"(?i)drop\s+table"],
    'XSS':
           [r"(?i)<script",
            r"javascript:",
            r"onerror="],
}
```

Рисунок 4.2 – Знаки атаки

Призначення: Регулярні вирази для виявлення атак

- (?i) - case-insensitive (без розрізнення великих/малих букв)
- \s+ - один або більше пробілів
- Якщо вхідні дані збігаються з будь-яким знаком → атака виявлена

4.4.3 Security Middleware

```
@app.before_request
def security_middleware():
```

Рисунок 4.3 – Функція для обробки запитів

Призначення: перехоплення кожного запиту до його обробки маршрутом (рис. 4.3)

Крок обробки 1 (рис. 4.4):

```
ip = request.remote_addr

if threat_engine.is_ip_blocked(ip):
    return "<h1>⛔ 403 Forbidden</h1><p>Ваш IP заблоковано системою безпеки.</p>", 403
```

Рисунок 4.4 – Перевірка чорного списку IP

- Отримує IP клієнта
- Якщо IP в чорному списку → блокує запит (HTTP 403)

Крок 2 (рис. 4.5):

```
now = time.time()
if ip not in threat_engine.request_history:
    threat_engine.request_history[ip] = []

threat_engine.request_history[ip] = [t for t in threat_engine.request_history[ip] if now - t < 10]
threat_engine.request_history[ip].append(now)

if len(threat_engine.request_history[ip]) > 30:
    threat_engine.block_ip(ip)
    threat_engine.log_incident(ip, attack_type: "DDoS Attack", payload: "Too many requests")
    return "429 Too Many Requests", 429
```

Рисунок 4.5 – Rate Limiting (захист від DDoS)

Алгоритм роботи:

1. Для кожної IP зберігаються часові мітки останніх запитів
2. Видаляються запити старіше 10 секунд
3. Додається поточний запит
4. Якщо більше 30 запитів → блокування IP

Крок 3 (рис. 4.6):

```
for value in payloads:
    if not isinstance(value, str):
        continue
    for attack_name, patterns in ATTACK_SIGNATURES.items():
        for pattern in patterns:
            if re.search(pattern, value):
                threat_engine.log_incident(ip, attack_name, value)
                return f"<h1>🚨 Security Alert</h1><p>Виявлено спробу атаки: {attack_name}</p>", 403
```

Рисунок 4.6 – WAF (аналіз знаків)

Що перевіряється:

- request.args - параметри в URL (?id=1&name=test)
- request.form - дані з форм (POST)
- request.json - JSON дані

4.4.4 Моделі бази даних

Модель 1 (рис. 4.7):

```
class User(UserMixin, db.Model):  8 usages
    id = db.Column(db.Integer, primary_key=True)                          # Унікальний ID
    username = db.Column(db.String(100), unique=True, nullable=False)     # Логін (унікальний)
    email = db.Column(db.String(100), unique=True, nullable=False)        # Email (унікальний)
    password_hash = db.Column(db.String(200), nullable=False)             # Хеш пароля (bcrypt)
    role = db.Column(db.String(20), default='user')                       # Роль (user/admin)
    created_at = db.Column(db.DateTime, default=datetime.datetime.now)    # Дата реєстрації
```

Рисунок 4.7 – User Model

Модель 2 (рис. 4.8):

```
class Product(db.Model):  13 usages
    id = db.Column(db.Integer, primary_key=True)              # Унікальний ID
    name = db.Column(db.String(100), nullable=False)          # Назва товару
    price = db.Column(db.Float, nullable=False)               # Ціна
    description = db.Column(db.String(500))                   # Опис
    image_emoji = db.Column(db.String(10), default="🛍️")
```

Рисунок 4.8 – Product Model

Модель 3 (рис. 4.9):

```
class Order(db.Model):  4 usages
    id = db.Column(db.Integer, primary_key=True)                                    # Унікальний ID
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)       # ID користувача
    total_amount = db.Column(db.Float, nullable=False)                              # Сума замовлення
    status = db.Column(db.String(50), default='Pending')                            # Статус замовлення
    items_json = db.Column(db.String(500))                                          # Список товарів
    created_at = db.Column(db.DateTime, default=datetime.datetime.now)              # Дата замовлення
```

Рисунок 4.9 – Order Model

4.4.5 Маршрути

Маршрут 1 (рис. 4.10):

```
@app.route('/')  6 usages
def index():
    products = Product.query.all()
    ... # Генерування HTML зі всіма товарами
    return render_template_string(HTML_TEMPLATE, content_html=content)
```

Рисунок 4.10 – GET / (Головна сторінка)

Функціональність:

- Отримує всі товари з БД
- Генерує HTML для кожного товару
- Якщо користувач не увійшов → показує "Увійдіть для покупок"
- Якщо увійшов → показує кнопку "+ Додати" для кожного товару

Доступ: Анонімний (без авторизації)

Маршрут 2 (рис. 4.11):

```
@app.route( rule: '/register', methods=['GET', 'POST'])
def register():
```

Рисунок 4.11 – GET /register (реєстрація)

GET запит (відображення форми):

Показує форму реєстрації з полями:

- Ім'я користувача (username)
- Email
- Пароль

```
username = request.form.get('username', '').strip()
email = request.form.get('email', '').strip()
password = request.form.get('password', '')

if not username or len(username) < 3:
    flash( message: 'Ім\'я повинно містити мінімум 3 символи', category: 'danger')
    return redirect(url_for('register'))

if User.query.filter_by(username=username).first():
    flash( message: 'Цей користувач вже існує', category: 'danger')
    return redirect(url_for('register'))

if User.query.filter_by(email=email).first():
    flash( message: 'Цей email вже зареєстрований', category: 'danger')
    return redirect(url_for('register'))

user = User(username=username, email=email, password_hash=generate_password_hash(password))
db.session.add(user)
db.session.commit()
```

Рисунок 4.12 – POST /register (обробка реєстрації)

Безпека:

- Пароль хешується через generate_password_hash() (bcrypt)
- Перевіряються унікальність username та email
- SQL injection неможна завдяки SQLAlchemy ORM

Результат: Користувач перенаправляється на сторінку входу

Маршрут 3 (рис. 4.13):

```
@app.route( rule: '/login', methods=['GET', 'POST'])  1 usage
def login():
    if request.method == 'POST':
        username = request.form.get('username', '').strip()
        password = request.form.get('password', '')
        user = User.query.filter_by(username=username).first()

        if user and check_password_hash(user.password_hash, password):
            login_user(user)
            flash( message: f'✅ Вітаємо, {username}!', category: 'success')
            return redirect(url_for('index'))
        else:
            flash( message: '❌ Неправильний логін або пароль', category: 'danger')
```

Рисунок 4.13 – POST/GET /login (вхід)

Безпека:

- Пароль не зберігається в чистому вигляді

- check_password_hash() порівнює хеші
- Помилка "Невірний логін" не розкриває чи існує користувач

Результат: Користувач автентифікується та перенаправляється на каталог

Маршрут 4 (рис. 4.14):

```
@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash( message: '✅ Ви вийшли з системи',  category: 'success')
    return redirect(url_for('index'))
```

Рисунок 4.14 – GET /logout (вихід)

Призначення: завершення сесії користувача

Маршрут 5 (рис. 4.15):

```
@app.route('/profile')  1 usage
@login_required
def profile():
    orders = Order.query.filter_by(user_id=current_user.id).order_by(Order.created_at.desc()).all()
```

Рисунок 4.15 – GET /profile (профіль користувача)

Показує:

- Інформацію про користувача:
  - Ім'я
  - Email
  - Дата реєстрації
- Всі замовлення користувача в таблиці:
  - ID замовлення
  - Сума
  - Статус
  - Дата

Маршрут 6 (рис. 4.16):

```python
@app.route( rule: '/add_to_cart/<int:product_id>', methods=['POST'])
@login_required
def add_to_cart(product_id):
    product = Product.query.get(product_id)
    if not product:
        flash( message: '❌ Товар не знайдено', category: 'danger')
        return redirect(url_for('index'))

    if 'cart' not in session:
        session['cart'] = {}

    session['cart'][str(product_id)] = session['cart'].get(str(product_id), 0) + 1
    session.modified = True
```

Рисунок 4.16 – POST /add_to_cart/<product_id> (додавання в кошик)

Як працює:

1. Перевіряє чи товар існує
2. Отримує кошик з сесії (якщо немає – створює новий)
3. Збільшує кількість товару на 1
4. Помічає сесію як змінену (для збереження в cookie)

Маршрут 7 (рис. 4.17):

```python
@app.route('/cart')  1 usage
@login_required
def cart():
    cart_ids = session.get('cart', {})
    if not cart_ids:
        flash( message: 'ℹ️ Ваш кошик порожній', category: 'info')
        return redirect(url_for('index'))

    products = Product.query.filter(Product.id.in_(map(int, cart_ids.keys()))).all()
```

Рисунок 4.17 – GET /cart (перегляд кошика)

Функціональність:

1. Отримує кошик з сесії
2. Отримує об'єкти товарів з БД
3. Обчислює суму
4. Показує таблицю з товарами та кнопкою оформлення

Маршрут 8 (рис. 4.18):

```python
@app.route( rule: '/checkout', methods=['POST'])
@login_required
def checkout():
    cart_ids = session.get('cart', {})
    if not cart_ids:
        flash( message: '⚠ Кошик порожній', category: 'danger')
        return redirect(url_for('cart'))

    products = Product.query.filter(Product.id.in_(map(int, cart_ids.keys()))).all()
    total = 0
    items_list = []

    for product in products:
        qty = cart_ids[str(product.id)]
        total += product.price * qty
        items_list.append(f"{product.name} x{qty}")

    order = Order(user_id=current_user.id, total_amount=total, items_json=", ".join(items_list),
                  status='Pending')
    db.session.add(order)
    db.session.commit()

    session.pop('cart', None)
```

Рисунок 4.18 – POST /checkout (оформлення замовлення)

Процес:

1. Отримує товари з кошика
2. Обчислює загальну суму (на сервері - для безпеки)
3. Створює запис Order в БД
4. Очищає кошик
5. Перенаправляє на профіль

Безпека: Сума обчислюється на сервері, тому клієнт не може змінити ціну.

Маршрут 10 (рис. 4.19):

```python
@app.route('/admin')
@login_required
def admin_dashboard():
    if current_user.role != 'admin':
        flash( message: '❌ Доступ заборонено', category: 'danger')
        return redirect(url_for('index'))
```

Рисунок 4.19 – GET /admin (панель адміністратора)

4.5 Тестування та верифікація програмної реалізації

У ході експериментальної перевірки працездатності розробленого програмного забезпечення було проведено наскрізне тестування ключових бізнес-процесів та механізмів захисту. Результати зафіксовані на серії скріншотів, що демонструють поведінку системи на різних етапах взаємодії.

4.5.1 Тестування функціоналу користувача (User Workflow)

Було перевірено повний цикл роботи користувача із системою: від реєстрації до створення замовлення.

- **Реєстрація та авторизація:**
  - На етапі реєстрації (рис. 4.20) було введено валідні дані користувача (ім'я "Олексій", email).

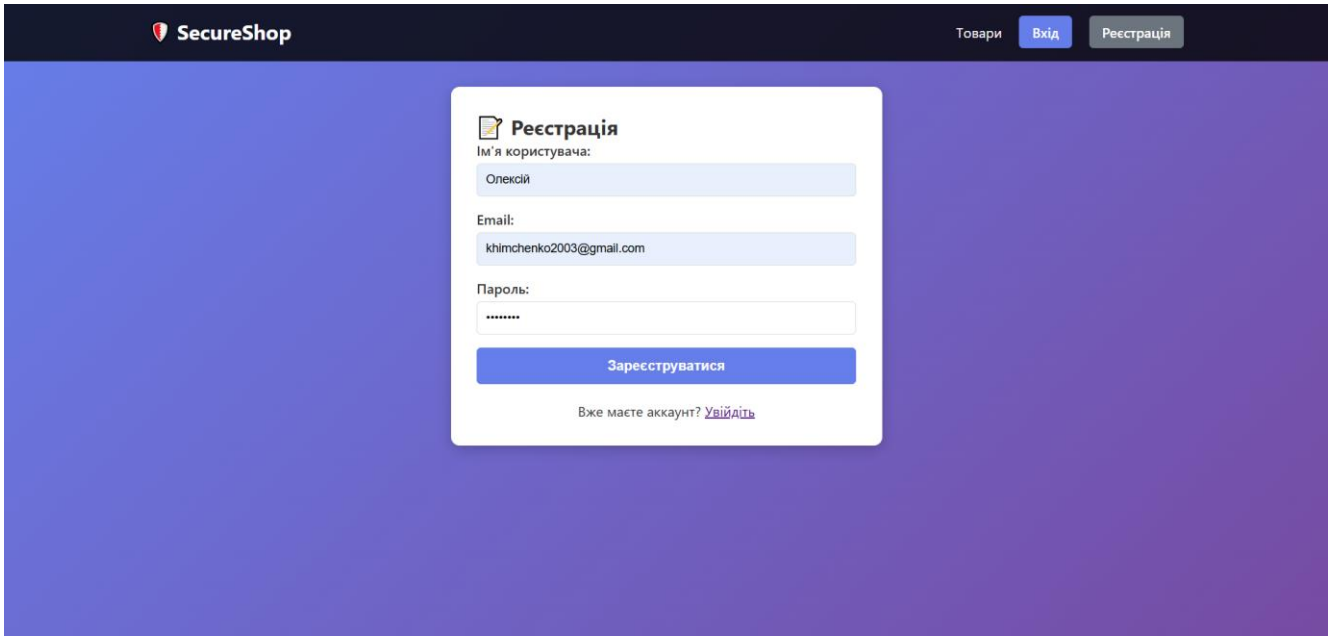


Рисунок 2.20 – Форма реєстрації

  - Система успішно обробила запит, створила новий запис у базі даних та відобразила повідомлення про успішну реєстрацію (рис. 4.21), після чого користувач отримав доступ до форми входу.

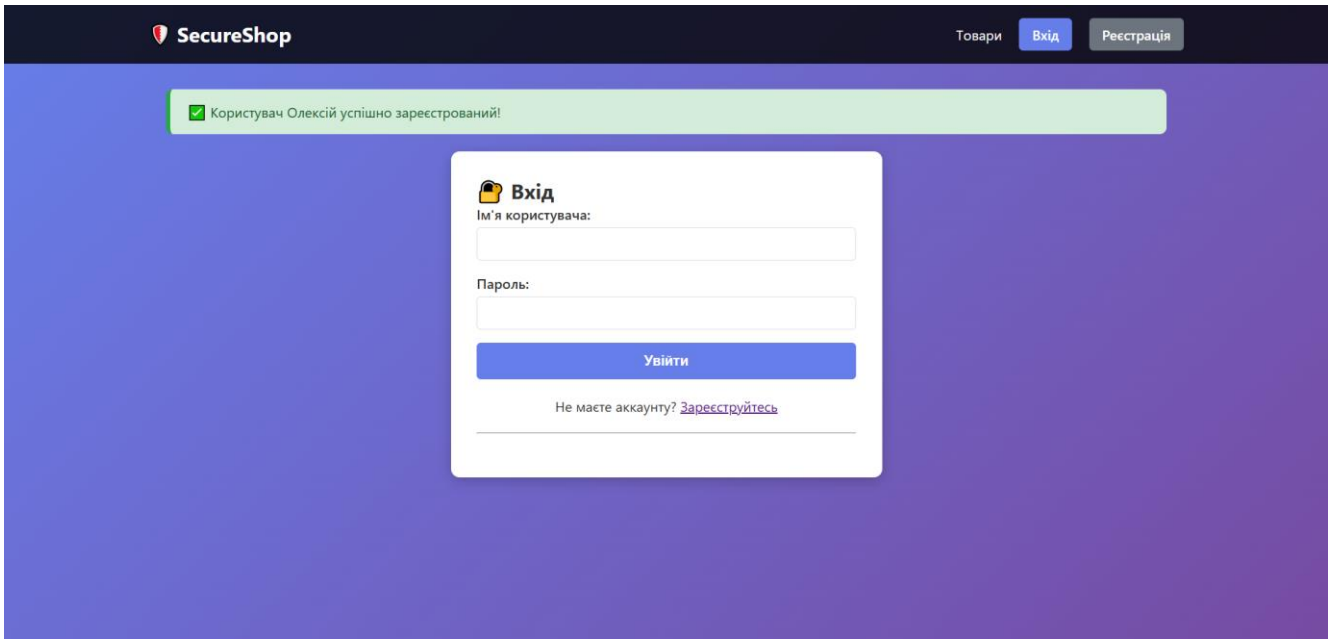


Рисунок 4.21 – Успішна реєстрація користувача

- **Взаємодія з каталогом та кошиком:**
  - Після авторизації користувач потрапив на головну сторінку з персоналізованим привітанням (рис. 4.22).

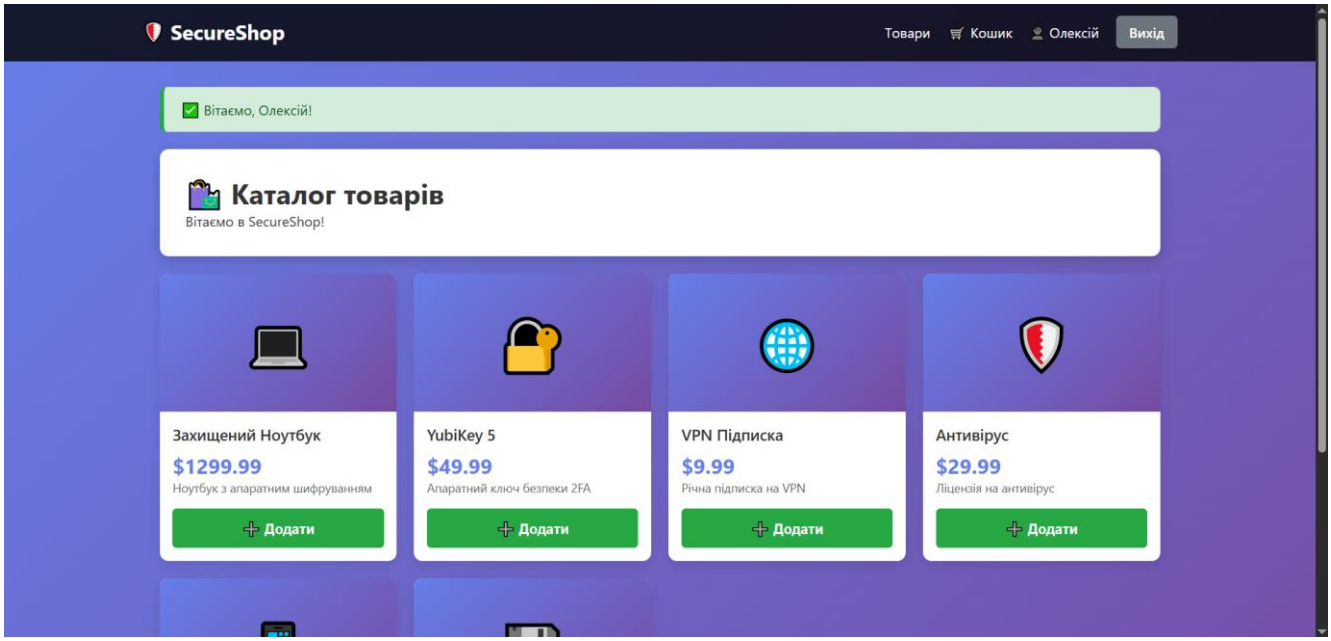


Рисунок 4.22 – Авторизована головна сторінка

  - Було протестовано додавання товарів у кошик. Користувач обрав три позиції: *Захищений Ноутбук*, *YubiKey 5* та *Антивірус*.
  - Сторінка кошика (рис. 4.23) коректно відобразила перелік товарів, їх кількість та розрахувала загальну суму замовлення ($1379.97), що підтверджує правильність роботи сесій Flask (session['cart']).

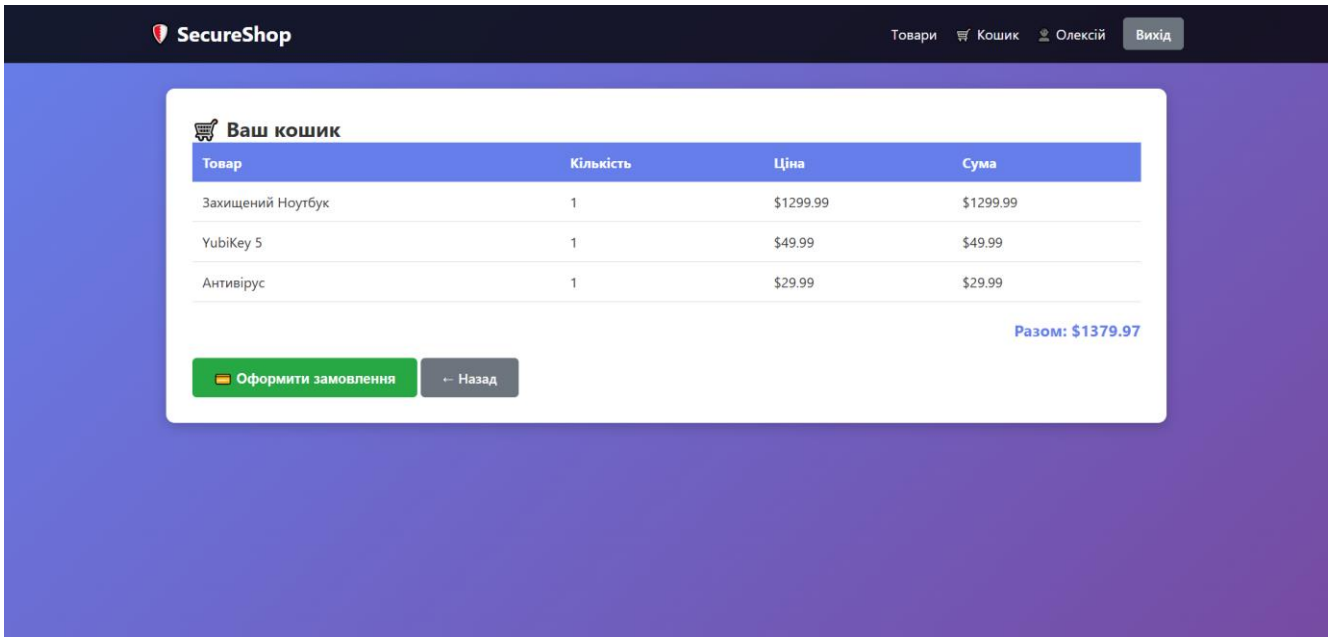


Рисунок 4.23 – Кошик користувача з доданими товарами

- **Оформлення замовлення:**
  - Після натискання "Оформити замовлення", система згенерувала Замовлення #3.
  - В особистому кабінеті (рис. 4.24) статус замовлення відображається як "Pending", що відповідає початковому стану кінцевого автомата замовлень.

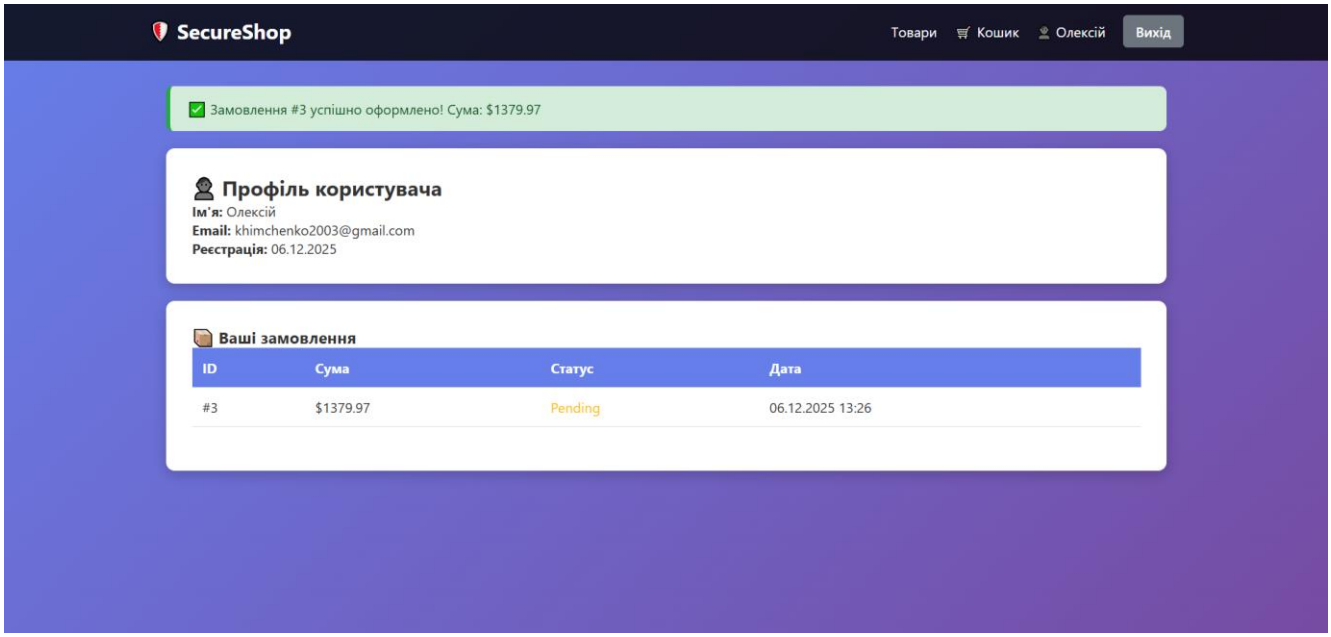


Рисунок 4.25 – Профіль користувача з підтвердженим замовленням

4.5.2. Верифікація цілісності даних (Data Integrity)

Для підтвердження коректності збереження даних було проведено перевірку на рівні СУБД (рис. 4.26).

- Використовуючи переглядач баз даних SQLite, було перевірено таблицю order.
- Запис з id=3 повністю відповідає даним з інтерфейсу: user_id=4, сума 1379.97, а поле items_json містить коректний перелік товарів. Це свідчить про надійну роботу ORM SQLAlchemy.

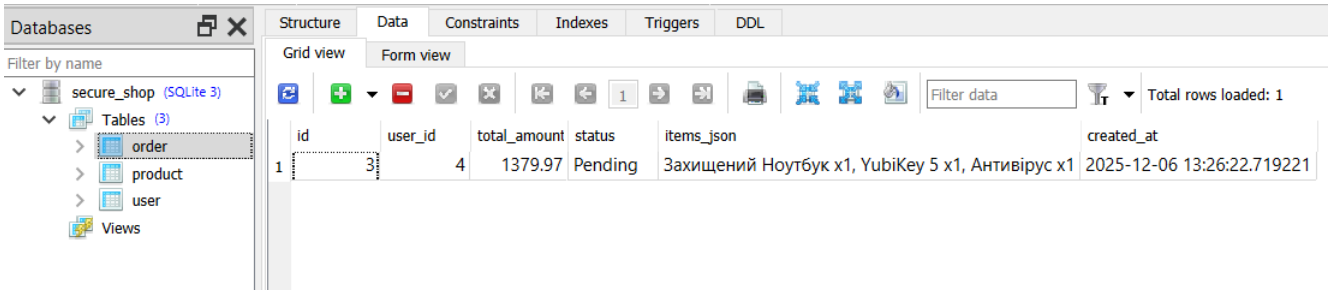


Рисунок 4.26 – Скриншот бази даних з доданим замовленням

4.5.3 Тестування підсистеми адміністрування та моніторингу (SIEM)

- Було перевірено рольову модель доступу. При вході під обліковим записом адміністратора (рис. 4.27) у навігаційній панелі з'явився додатковий пункт "Панель безпеки".

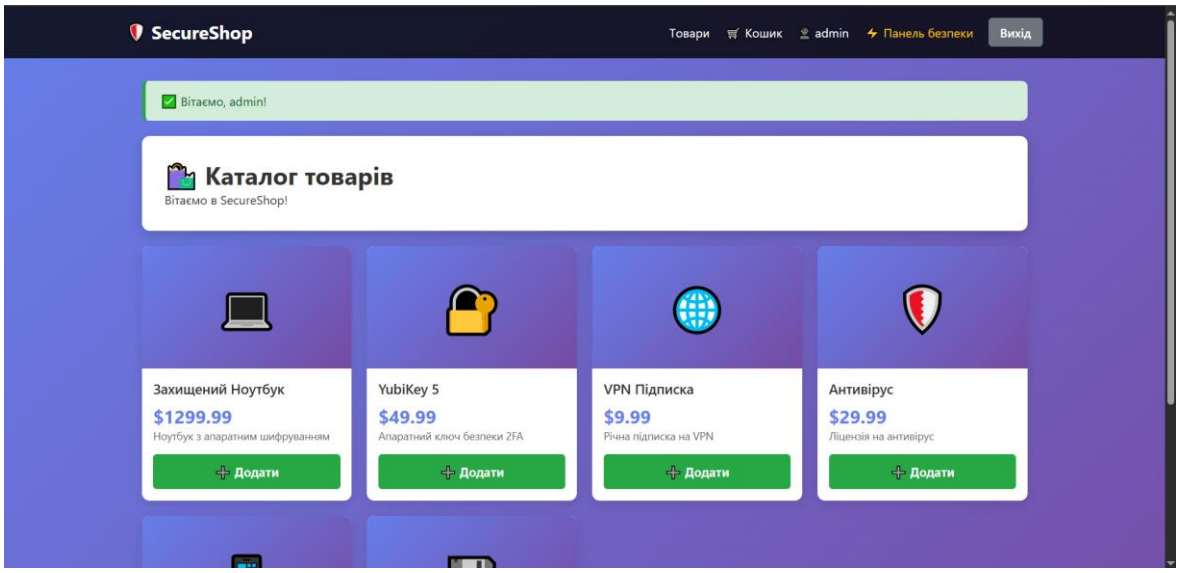


Рисунок 4.27 – Головна сторінка авторизованого адміна

- Панель безпеки (рис. 4.28) коректно агрегує статистику системи: відображає загальну кількість користувачів (4), замовлень (1) та статус системи моніторингу загроз (ThreatMonitor). На момент перевірки інцидентів не виявлено, система працює в штатному режимі.

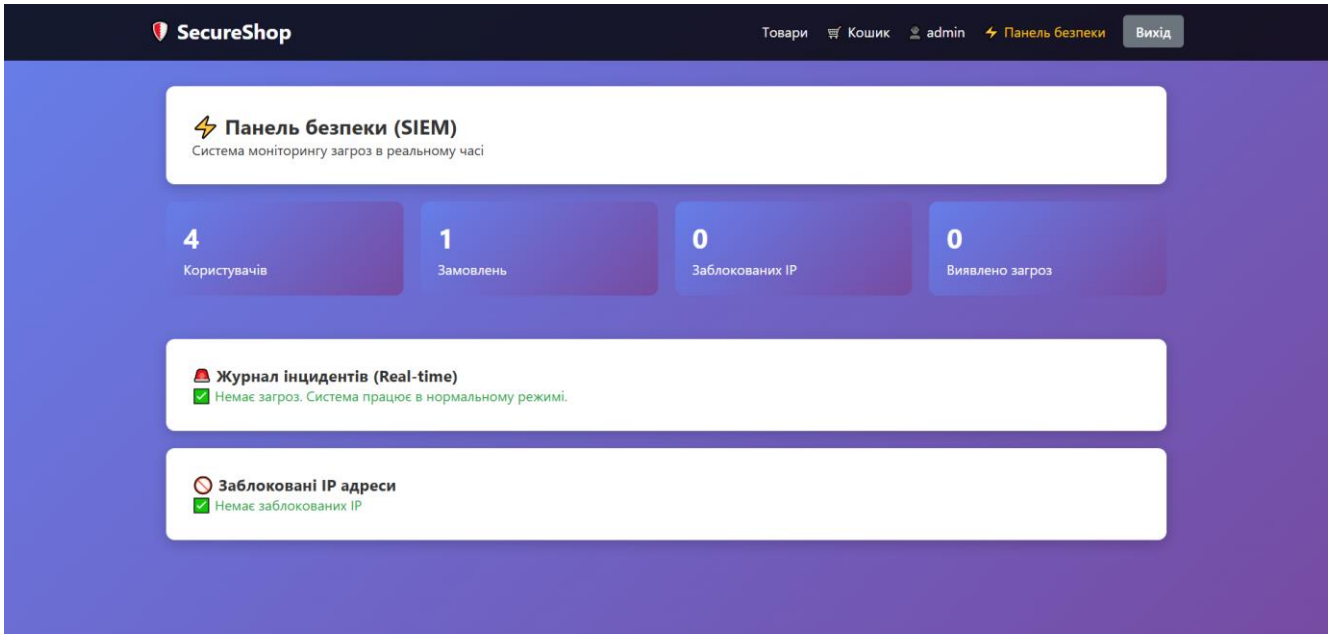


Рисунок 4.28 – Панель безпеки адміна

4.5.4 Стрес-тестування механізмів захисту (Rate Limiting)

- Для перевірки стійкості до DDoS-атак та брутфорсу було змодельовано ситуацію надсилання великої кількості запитів за короткий проміжок часу.
- Система захисту успішно ідентифікувала аномальну активність і заблокувала доступ, повернувши HTTP-код помилки **429 Too Many Requests** (рис. 4.29), після чого IP, з якого надсилались запити було заблоковано (рис. 4.30)

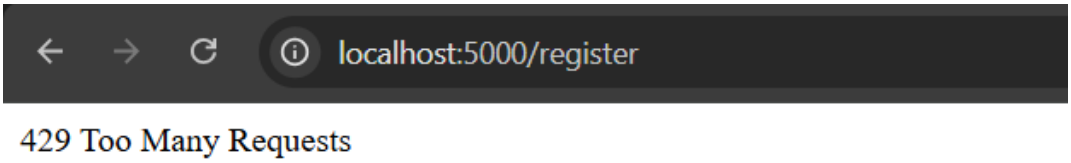


Рисунок 4.29 – Попередження про велику кількість запитів

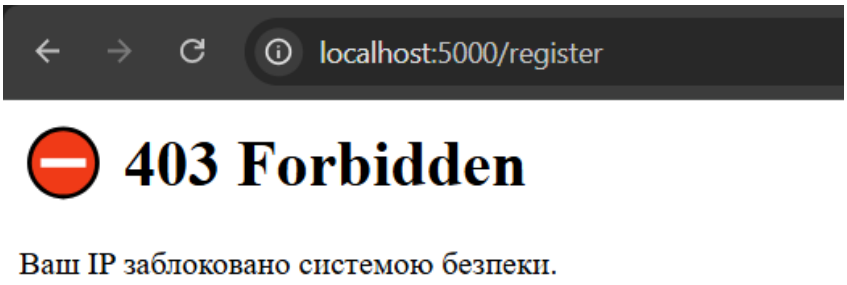


Рисунок 4.30 – Повідомлення про блокування IP адреси

- Це підтверджує коректність роботи алгоритму "Sliding Window", реалізованого у middleware додатку.

Розділ 5. Ергономічний аналіз

5.1. Теоретико-нормативна база ергономічної оцінки

Фундаментом для оцінки ергономіки розробленого рішення є комплекс міжнародних стандартів ISO 9241 "Ергономіка взаємодії людина-система". Цей стандарт еволюціонував від вимог до фізичного обладнання (відеотерміналів) до комплексних настанов щодо юзабіліті програмного забезпечення та людино-орієнтованого проєктування.

Ключовим документом є ДСТУ ISO 9241-11:2006 (ISO 9241-11:1998) "Настанови щодо прийнятності у використанні". Стандарт визначає юзабіліті (usability) не просто як "зручність", а як міру, з якою продукт може бути використаний певними користувачами для досягнення визначених цілей у певному контексті використання. Визначення базується на трьох метриках [25]:

1. **Ефективність (Effectiveness):** Точність і повнота, з якими користувачі досягають своїх цілей. У контексті розробленого API це означає коректність обробки даних та відсутність втрат інформації.
2. **Результативність (Efficiency):** Співвідношення між досягнутим результатом та витраченими ресурсами (часом, ментальними зусиллями). Впровадження Rate Limiting підвищує загальну результативність системи, запобігаючи деградації продуктивності при перевантаженнях, що гарантує стабільний час відгуку для легітимних користувачів.
3. **Задоволеність (Satisfaction):** Суб'єктивна реакція користувача, відсутність дискомфорту та позитивне ставлення до використання системи. Передбачуваність поведінки API (навіть у випадку помилок) є ключем до задоволеності розробників, які інтегрують даний сервіс.

5.2. Реалізація принципів діалогу згідно ДСТУ EN ISO 9241-110

Стандарт **ДСТУ EN ISO 9241-110:2022 "Принципи взаємодії"** (раніше відомий як "Принципи діалогу") встановлює сім фундаментальних евристик, які

описують якісну взаємодію між користувачем та системою. Нижче наведено детальний аналіз відповідності розробленої системи кожному з цих принципів [26].

5.2.1. Придатність для виконання завдання (Suitability for the task)

Діалог є придатним для виконання завдання, якщо він підтримує користувача у досягненні його цілей без зайвого навантаження.

- **Реалізація:** Розроблений Flask-додаток надає чіткий API, де кожен endpoint відповідає за конкретну атомарну операцію. Система не вимагає від користувача (або клієнтського ПЗ) виконання надлишкових кроків. Наприклад, авторизація та виконання запиту можуть відбуватися в рамках одного HTTP-виклику (через заголовки), що мінімізує кількість транзакцій.
- **Rate Limiting:** Механізм обмеження частоти запитів також працює на цей принцип, гарантуючи, що система залишається доступною (serviceable) для виконання завдань іншими користувачами, не "падаючи" від дій одного зловмисника.

5.2.2. Самоописовість (Self-descriptiveness)

Діалог є самоописовим, коли кожний крок взаємодії є зрозумілим сам по собі або має пояснення.

- **Реалізація:** Це критично важливо для API. Коли спрацьовує Rate Limiter, система повертає не просто помилку з'єднання, а конкретний HTTP статус 429 Too Many Requests. Тіло відповіді містить JSON з детальним описом ({"error": "Rate limit exceeded", "retry_after": 15}), що пояснює причину відмови та вказує, коли можна повторити спробу. Це робить систему "прозорою" для розробника-клієнта.

- **Навігація:** URL-адреси спроєктовані за принципами RESTful, де шлях до ресурсу (наприклад, /api/users/123) чітко описує сутність, до якої йде звернення.

5.2.3. Керованість (Controllability)

Користувач повинен мати можливість ініціювати та контролювати напрямок і темп взаємодії.

- **Реалізація:** У веб-інтерфейсах це реалізується через кнопки "Назад", "Скасувати". У контексті API це означає, що клієнт контролює момент надсилання запиту. Якщо система зайнята або ліміт вичерпано, інформація про час очікування (Retry-After) дає клієнту контроль над плануванням наступного запиту, замість того щоб бездумно "бомбити" сервер.

5.2.4. Відповідність очікуванням користувача (Conformity with user expectations)

Система повинна поводитися узгоджено та передбачувано, базуючись на попередньому досвіді користувача та загальноприйнятих конвенціях.

- **Реалізація:** Використання стандартних кодів HTTP (200 для успіху, 404 для відсутності, 500 для помилок сервера) повністю відповідає очікуванням будь-якого веб-розробника. Дизайн інтерфейсу (якщо розглядається адміністративна панель) використовує звичне розташування елементів: меню зліва або зверху, логотип як посилання на головну, стандартні іконки (лупа для пошуку, хрестик для закриття).

5.2.5. Толерантність до помилок (Error tolerance/Use error robustness)

Система повинна досягати очікуваного результату незважаючи на незначні помилки введення, або надавати інструменти для їх легкого виправлення [27].

- **Реалізація:** Валідація даних у Flask (наприклад, через Pydantic або Marshmallow) дозволяє перевірити тип даних до їх обробки. Якщо користувач надсилає рядок замість числа в ID, система поверне 400 Bad Request з вказівкою на конкретне поле, а не 500 Internal Server Error. Алгоритм Sliding Window також є толерантним: він не блокує користувача назавжди за перевищення ліміту, а лише тимчасово обмежує, автоматично відновлюючи доступ з часом.

5.2.6. Придатність для індивідуалізації (Suitability for individualization)

Можливість адаптації інтерфейсу під потреби користувача [28].

- **Реалізація:** У веб-додатку це може бути вибір мови інтерфейсу або темної/світлої теми. На рівні API індивідуалізація реалізується через параметри запиту (наприклад, фільтрація результатів, вибір полів для відображення), що дозволяє клієнту отримувати дані саме в тому вигляді, який йому потрібен.

5.2.7. Придатність для навчання (Suitability for learning/Learnability)

Система повинна підтримувати та направляти користувача в процесі освоєння її функцій.

- **Реалізація:** Наявність інтерактивної документації (наприклад, Swagger/OpenAPI), що генерується автоматично, дозволяє новим користувачам швидко зрозуміти можливості API, протестувати запити та вивчити структуру даних без необхідності читати багатосторінкові мануали.

5.3. Візуальна ергономіка та психофізіологічний вплив дизайну

Дизайн інтерфейсу – це не лише естетика, а й важливий ергономічний фактор, що впливає на втомлюваність та швидкість сприйняття інформації. Для розробленої системи (адміністративна панель або лендінг документації) було обрано специфічну колірну гаму.

5.3.1. Психологія та семантика кольору

Обрана палітра включає два домінуючі кольори (рис. 5.1): **Синій (#667eea)** та **Фіолетовий (#764ba2)**. Цей вибір обґрунтовується психологією сприйняття кольору у веб-дизайні.

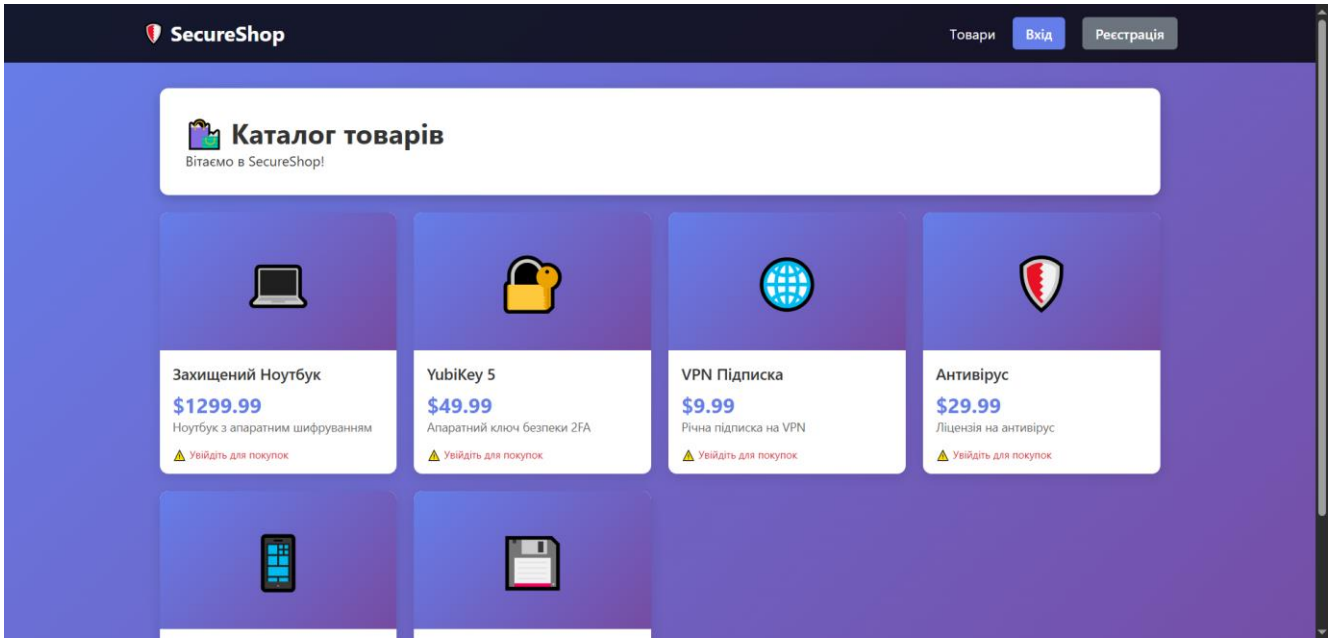


Рисунок 5.1 – Кольорова палітра головної сторінки

1. **Синій (#667eea):**
    - Характеристика: Цей відтінок (близький до "Royal Blue" або "Cornflower Blue") знаходиться в холодній частині спектру. За моделлю HSL (Hue: 229°, Saturation: 76%, Lightness: 66%) він є досить насиченим, але не агресивним.
    - Психологічний вплив: Синій колір традиційно асоціюється з довірою, спокоєм, технологічністю, логікою та безпекою. У фінансових та IT-

продуктах він є стандартом, оскільки знижує тривожність користувача та створює відчуття стабільності ("синій екран" Facebook, LinkedIn, Intel). Використання цього кольору для кнопок дії (CTA) або заголовків підсвідомо сигналізує користувачу: "тут безпечно натискати".

2. **Фіолетовий (#764ba2):**
    - Характеристика*:* Відтінок, що поєднує стабільність синього та енергію червоного. За HSL (Hue: 270°, Saturation: 37%, Lightness: 46%) він є темнішим та глибшим.
    - Психологічний вплив: Фіолетовий асоціюється з мудрістю, креативністю, уявою, а також розкішшю та якістю (преміальний сегмент). У дизайні інтерфейсів він часто використовується для акцентів, щоб підкреслити інноваційність продукту. Поєднання синього та фіолетового (часто у вигляді градієнту, популярного в сучасному стилі "Tech SaaS") створює баланс між надійністю (синій) та сучасністю/магією технологій (фіолетовий), уникаючи при цьому "канцелярської" нудьги чистого синього.

5.3.2. Доступність (Accessibility) та контрастність згідно WCAG

Ергономіка вимагає, щоб інтерфейс був доступним для людей з вадами зору. Стандарт **WCAG (Web Content Accessibility Guidelines) 2.1** встановлює суворі вимоги до контрастності тексту відносно фону.

Проведемо аналіз контрастності обраних кольорів на білому фоні (#FFFFFF), який є стандартним для веб-сторінок:

1. **Фіолетовий (#764ba2) на Білому:**
    - Коефіцієнт контрастності складає приблизно **8.42:1** (інші джерела вказують на високий контраст > 7:1 для подібних відтінків).
    - Вердикт: Цей показник значно перевищує мінімальний поріг **4.5:1** для рівня AA і навіть поріг **7:1** для рівня AAA (найвищий рівень доступності) для звичайного тексту. Це робить фіолетовий колір

ідеальним для основного тексту, заголовків та важливих повідомлень – він буде чітко читатися навіть людьми зі зниженим зором або на екранах з поганою передачею кольору.

2. **Синій (#667eea) на Білому:**
    - Коефіцієнт контрастності складає приблизно **4.19:1**.
    - Вердикт: Цей показник є меншим за необхідні 4.5:1 для звичайного тексту (рівень AA). Це означає, що використання цього кольору для дрібного тексту (менше 14pt bold або 18pt regular) є ергономічною помилкою, оскільки текст буде важко читати.
    - Рекомендація: Згідно з WCAG, цей колір проходить перевірку для великого тексту (Large Text, >18pt), де вимога становить 3:1. Тому в дизайні цей відтінок синього слід використовувати виключно для великих заголовків, графічних іконок або кнопок з білим текстом (за умови, що контраст тексту на кнопці буде достатнім), але не для основного контенту (body text).

5.4. Типографіка та сприйняття текстової інформації

Текст є основним носієм інформації в веб-системі. Вибір шрифту визначає швидкість читання та втомлюваність очей. Для системи було обрано шрифт **Segoe UI**.

Обґрунтування вибору Segoe UI:

1. **Спеціалізація для екранів:** Segoe UI (User Interface) – це шрифт сімейства гротесків (sans-serif), розроблений корпорацією Microsoft спеціально для відображення тексту на екранах моніторів. Його дизайн оптимізовано для технології субпіксельного рендерингу ClearType, що робить текст чітким та гладким навіть при низькій роздільній здатності екрану [29].
2. **Характеристики гліфів:** Шрифт має відкриті форми літер (open counters), що покращує розпізнавання символів. Відсутність засічок (serifs) зменшує

візуальний шум на екрані, полегшуючи швидке сканування тексту поглядом, що є типовим патерном поведінки веб-користувачів.

3. **Мультимовність:** Segoe UI підтримує величезний діапазон писемностей, включаючи розширену кирилицю. Це гарантує, що український текст (включаючи специфічні літери ґ, є, і, ї) буде відображатися коректно та гармонійно, без підміни символів з інших шрифтів.
4. **Варіативність накреслень:** Наявність широкого спектру накреслень (Light, Semilight, Regular, Semibold, Bold, Black) дозволяє дизайнеру будувати чітку візуальну ієрархію інформації (заголовки, підзаголовки, акценти) без необхідності підключати додаткові шрифтові гарнітури, що позитивно впливає на швидкість завантаження сторінки.

5.5. Охорона праці та вимоги до робочого середовища

Оскільки дипломна робота передбачає не лише створення продукту, а й аналіз умов його використання, необхідно розглянути аспекти охорони праці. Робота оператора розробленої системи відноситься до категорії робіт з відеотерміналами (ВДТ).

Згідно з ДСТУ ISO 9241-5 (Вимоги до розташування робочої станції) та ДСТУ ISO 9241-6 (Вимоги до середовища), необхідно забезпечити:

- **Освітлення:** Загальне та місцеве освітлення повинно бути організоване так, щоб виключити відблиски на екрані монітора, які знижують контрастність зображення (особливо критично при використанні світлих тем інтерфейсу) [30].
- **Режим праці:** Програмне забезпечення повинно сприяти ефективному виконанню завдань, щоб мінімізувати час перебування оператора за екраном. Висока юзабіліті та швидкодія системи (забезпечена технічними рішеннями Розділу 4) прямо сприяють зниженню зорової та нервово-емоційної напруги персоналу.

Висновок

У ході виконання дипломної роботи було вирішено актуальне науково-прикладне завдання розробки та ергономічного забезпечення веб-орієнтованої системи обробки запитів із захистом від автоматизованих атак. За результатами проведеного дослідження та практичної реалізації можна зробити наступні висновки:

Проведено аналіз предметної області та загроз безпеці. У перших розділах роботи було досліджено сучасний ландшафт веб-загроз. Встановлено, що для систем електронної комерції критичними вразливостями є SQL-ін'єкції, XSS-атаки та DDoS-атаки рівня додатку. Було обґрунтовано необхідність впровадження гібридної системи захисту, що поєднує сигнатурний аналіз вхідних даних та алгоритмічне обмеження частоти запитів (Rate Limiting).

Обґрунтовано вибір технологічного стека та методів захисту. Для реалізації серверної частини обрано мову програмування Python та мікрофреймворк Flask, що забезпечило гнучкість архітектури та високу швидкість розробки. В якості алгоритму обмеження навантаження обрано метод «Sliding Window» (Ковзне вікно), який, на відміну від методів «Fixed Window» чи «Leaky Bucket», забезпечує більш точне згладжування сплесків трафіку та справедливо розподіляє ресурси сервера без жорстких розривів на межах часових інтервалів.

Спроєктовано архітектуру системи SecureShop. Розроблено структуру бази даних (SQLite) з таблицями користувачів, товарів та замовлень, яка відповідає третій нормальній формі. Реалізовано архітектурний патерн MVC (Model-View-Controller) засобами Flask Blueprints та ORM SQLAlchemy. Створено підсистему моніторингу загроз ThreatMonitor, що діє як проміжне програмне забезпечення (middleware), перехоплюючи запити до їх обробки бізнес-логікою.

Виконано програмну реалізацію захищеного веб-додатку. Створено повнофункціональний прототип системи, що включає:

- Модуль авторизації з хешуванням паролів (werkzeug.security) та управлінням сесіями.

- Механізм WAF (Web Application Firewall) на основі регулярних виразів для блокування SQLi та XSS пайлоадів.
- Систему Rate Limiting, яка автоматично блокує IP-адреси при перевищенні ліміту запитів (повертаючи статус 429).
- Панель адміністратора для моніторингу інцидентів у реальному часі.

Проведено тестування та ергономічну оцінку.

- **Функціональне тестування** підтвердило коректність роботи бізнес-логіки (реєстрація, кошик, замовлення) та цілісність даних у БД.
- **Стрес-тестування** (зображене на скріншотах звіту) довело ефективність захисту: система успішно ідентифікувала та заблокувала аномальну активність, зберігши працездатність для легітимних користувачів.
- **Ергономічний аналіз** показав відповідність інтерфейсу стандартам ISO 9241-110. Використання колірної схеми (синій/фіолетовий) сприяє довірі користувачів, а шрифт Segoe UI забезпечує читабельність. Контрастність елементів відповідає вимогам WCAG, що робить систему доступною та зручною у використанні.

Список використаних джерел

1. E-Commerce - Що таке електронна комерція? | Wezom. *IT-компанія повного циклу розробки програмних продуктів WEZOM - Київ, Україна*. URL: https://wezom.com.ua/ua/blog/elektronna-komertsiya
2. E-commerce: що треба знати і які зараз тренди? | Школа бізнесу Нова Пошта. URL: https://online.novaposhta.education/blog/e-commerce-shho-treba-znati-i-yaki-zaraz-trendi
3. Data Classification (Data Management): A Complete Overview | Spirion. *Spirion*. URL: https://www.spirion.com/data-classification#phase-3
4. A Comprehensive Guide to PCI DSS SAQ Types CompliancePoint. *CompliancePoint*. URL: https://www.compliancepoint.com/assurance/a-comprehensive-guide-to-pci-dss-saq-types/
5. Top 4 Commercial Data Classification Levels and When to Use Them - Numerous.ai. *Numerous.ai*. URL: https://numerous.ai/blog/commercial-data-classification-levels
6. Log Types and Formats: A Comprehensive Guide. *Edge Delta*. URL: https://edgedelta.com/company/blog/log-types-and-formats
7. Data Classification: Explaining the What, Why, and How [+ Free Template]. *Secureframe*. URL: https://secureframe.com/blog/data-classification
8. What is the CIA Triad and Why is it important? | Fortinet. *Fortinet*. URL: https://www.fortinet.com/resources/cyberglossary/cia-triad
9. CIA triad: Confidentiality, integrity, and availability. *Unified identity security: The core of your modern enterprise*. URL: https://www.sailpoint.com/identity-library/cia-triad
10. The Five Pillars of Information Security: CIA Triad and More. *Destination Certification*. URL: https://destcert.com/resources/five-pillars-information-security/

11. Ukrainian eCommerce Trends in H1 2025 | Research & Insights by Promodo. *Promodo | Performance Marketing Agency*. URL: https://www.promodo.com/blog/ukrainian-ecommerce-market-in-h1-2025
12. OWASP Top 10:2021. *OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation*. URL: https://owasp.org/Top10/2021/
13. What is a DDoS attack? | Cloudflare. Learning. URL: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/
14. Cymulate. YARA Rules. *Cymulate*. URL: https://cymulate.com/cybersecurity-glossary/yara-rules/
15. ITE3. Інформаційна безпека підприємства – основні ризики та способи захисту. *ITE3*. URL: https://itez.com.ua/blog/information-security-best-practices-for-small-business-ukraine.html
16. Rate Limiting The Sliding Window Algorithm. *@m-elbably*. URL: https://medium.com/@m-elbably/rate-limiting-the-sliding-window-algorithm-daa1d91e6196
17. vkarpova7. Основні переваги сертифікації ISO/IEC 27001. *ISSP Training*. URL: https://www.issp.training/post/osnovni-perevahy-sertyfikatsiyi-iso-iec-27001
18. What is Non-repudiation in Cyber Security? | Bitsight. *Bitsight*. URL: https://www.bitsight.com/glossary/non-repudiation-cyber-security
19. Що таке сертифікація PCI DSS – Interkassa. *Прийом платежів на сайті з Interkassa*. URL: https://interkassa.com/blog/shcho-take-sertyfikatsiia-pci-dss
20. Gołąbek M. Implementing OWASP ASVS | SoftwareMill. *SoftwareMill*. URL: https://softwaremill.com/implementing-owasp-asvs/
21. GDPR. URL: https://gdpr-text.com/uk/
22. Client Side Protection Compliance. *akamai*. URL: https://www.akamai.com/resources/product-brief/client-side-protection-compliance.
23. 20 найпоширеніших видів атак на кібербезпеку. *Bittnet Training*. URL: https://www.bittnet.ro/uk/noutati/top-20-cele-mai-frecvente-tipuri-de-atacuri-

de-securitate-cibernetica/?srsltid=AfmBOoo4Mfxi1i2_F4xfDMRu-JLE4YraLmt_jVlbwITGKjoMqzRP7jbf

24. ІЩо таке SIEM? Компоненти, можливості та архітектура. *Stellar Cyber*. URL: https://stellarcyber.ai/uk/learn/what-is-siem/
25. Contributors to Wikimedia projects. ISO 9241 - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/ISO_9241
26. ISO’s dialogue principles (2020) – DialogDesign. *DialogDesign*. URL: https://www.dialogdesign.dk/isos-dialogue-principles-2019/
27. Bot Verification. *Remote User Testing Tool | Userpeek.com*. URL: https://userpeek.com/blog/iso-9241-110-ergonomics-of-human-system-interaction-part-110-interaction-principles/
28. Weibelzahl S. ISO 9241-110: Dialog Principles | UX & Usability Toolkit. *UX & Usability Plattform | UX & Usability Toolkit*. URL: https://www.softwareevaluation.de/en/foundations/iso-9241-110-dialog-principles/
29. Segoe UI font family - Typography. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/it-it/typography/font-list/segoe-ui
30. *Кафедра охорони праці, промислової та цивільної безпеки | Офіційний сайт*. URL: https://opcb.kpi.ua/wp-content/uploads/2014/09/Рекомендації-до-виконання-DP.pdf

Додаток А

Код програми

```
import os
import re
import datetime
import time
from flask import Flask, render_template_string, request, redirect, url_for,
flash, session
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user, logout_user,
login_required, current_user
from werkzeug.security import generate_password_hash, check_password_hash

app = Flask(__name__)
app.config['SECRET_KEY'] = os.urandom(24)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///secure_shop.db'
app.config['SESSION_COOKIE_HTTPONLY'] = True
app.config['SESSION_COOKIE_SECURE'] = False
app.config['SESSION_COOKIE_SAMESITE'] = 'Lax'

db = SQLAlchemy(app)
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'


# СИСТЕМА ВИЯВЛЕННЯ ЗАГРОЗ

class ThreatMonitor:
    def __init__(self):
        self.blacklist = set()       # Множина заблокованих IP
        self.request_history = {}    # Історія запитів по IP
        self.logs = []               # Журнал інцидентів

    def log_incident(self, ip, attack_type, payload):
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
        self.logs.append({
            'time': timestamp,
            'ip': ip,
            'type': attack_type,
            'payload': payload[:50]
        })

    def is_ip_blocked(self, ip):
        return ip in self.blacklist

    def block_ip(self, ip):
        self.blacklist.add(ip)


threat_engine = ThreatMonitor()

ATTACK_SIGNATURES = {
    'SQL Injection':
                    [r"(?i)union\s+select",
                     r"(?i)or\s+1=1",
                     r"(?i)drop\s+table"],
    'XSS':
           [r"(?i)<script",
            r"javascript:",
            r"onerror="],
}


@app.before_request
def security_middleware():
    ip = request.remote_addr

    if threat_engine.is_ip_blocked(ip):
        return "<h1>⛔ 403 Forbidden</h1><p>Ваш IP заблоковано системою
безпеки.</p>", 403

    now = time.time()
```

```
    if ip not in threat_engine.request_history:
        threat_engine.request_history[ip] = []

    threat_engine.request_history[ip] = [t for t in
threat_engine.request_history[ip] if now - t < 10]
    threat_engine.request_history[ip].append(now)

    if len(threat_engine.request_history[ip]) > 2:
        threat_engine.block_ip(ip)
        threat_engine.log_incident(ip, "DDoS Attack", "Too many requests")
        return "429 Too Many Requests", 429

    payloads = list(request.args.values()) + list(request.form.values())
    if request.is_json and request.json:
        payloads += [str(v) for v in request.json.values()]

    for value in payloads:
        if not isinstance(value, str):
            continue
        for attack_name, patterns in ATTACK_SIGNATURES.items():
            for pattern in patterns:
                if re.search(pattern, value):
                    threat_engine.log_incident(ip, attack_name, value)
                    return f"<h1>🚨 Security Alert</h1><p>Виявлено спробу атаки:
{attack_name}</p>", 403

# МОДЕЛІ БАЗИ ДАНИХ

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)                                #
Унікальний ID
    username = db.Column(db.String(100), unique=True, nullable=False)      # Логін
(унікальний)
    email = db.Column(db.String(100), unique=True, nullable=False)         # Email
(унікальний)
    password_hash = db.Column(db.String(200), nullable=False)              # Хеш
пароля (bcrypt)
```

```
    role = db.Column(db.String(20), default='user')                          # Роль
(user/admin)
    created_at = db.Column(db.DateTime, default=datetime.datetime.now)     # Дата
реєстрації


class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)            # Унікальний ID
    name = db.Column(db.String(100), nullable=False)        # Назва товару
    price = db.Column(db.Float, nullable=False)             # Ціна
    description = db.Column(db.String(500))                 # Опис
    image_emoji = db.Column(db.String(10), default="🛍️")


class Order(db.Model):
    id = db.Column(db.Integer, primary_key=True)                                #
Унікальний ID
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)   #
ID користувача
    total_amount = db.Column(db.Float, nullable=False)                          #
Сума замовлення
    status = db.Column(db.String(50), default='Pending')                        #
Статус замовлення
    items_json = db.Column(db.String(500))                                      #
Список товарів
    created_at = db.Column(db.DateTime, default=datetime.datetime.now)          #
Дата замовлення


@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))


# HTML ШАБЛОН

HTML_TEMPLATE = """
```

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SecureShop - Безпечний Магазин</title>
    <style>
        * { margin: 0; padding: 0; box-sizing: border-box; }
        body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
background: linear-gradient(135deg, #667eea 0%, #764ba2 100%); color: #333; min-
height: 100vh; }
        .navbar { background: rgba(0, 0, 0, 0.8); padding: 15px 20px; }
        .navbar-content { max-width: 1200px; margin: 0 auto; display: flex;
justify-content: space-between; align-items: center; }
        .navbar-brand { color: #fff; font-size: 24px; font-weight: bold; text-
decoration: none; }
        .nav-links { display: flex; gap: 20px; align-items: center; }
        .nav-links a { color: #fff; text-decoration: none; transition: 0.3s; }
        .nav-links a:hover { color: #ffc107; }
        .container { max-width: 1200px; margin: 30px auto; padding: 0 20px; }
        .card { background: white; border-radius: 10px; box-shadow: 0 4px 15px
rgba(0,0,0,0.2); padding: 30px; margin-bottom: 20px; }
        .alert { padding: 15px 20px; border-radius: 8px; margin-bottom: 20px;
animation: slideIn 0.3s; }
        .alert.success { background: #d4edda; color: #155724; border-left: 5px
solid #28a745; }
        .alert.danger { background: #f8d7da; color: #721c24; border-left: 5px
solid #dc3545; }
        .alert.info { background: #d1ecf1; color: #0c5460; border-left: 5px solid
#17a2b8; }
        .form-group { margin-bottom: 15px; }
        .form-group label { display: block; margin-bottom: 5px; font-weight: 600;
}
        .form-group input { width: 100%; padding: 10px; border: 1px solid #ddd;
border-radius: 5px; font-size: 14px; }
        .form-group input:focus { outline: none; border-color: #667eea; box-
shadow: 0 0 5px rgba(102, 126, 234, 0.5); }
```

```
        .btn { padding: 12px 25px; border: none; border-radius: 5px; cursor:
pointer; font-size: 16px; font-weight: 600; transition: 0.3s; text-decoration:
none; display: inline-block; }
        .btn-primary { background: #667eea; color: white; }
        .btn-primary:hover { background: #5568d3; }
        .btn-secondary { background: #6c757d; color: white; }
        .btn-secondary:hover { background: #5a6268; }
        .btn-success { background: #28a745; color: white; }
        .btn-success:hover { background: #218838; }
        .product-grid { display: grid; grid-template-columns: repeat(auto-fit,
minmax(250px, 1fr)); gap: 20px; margin-bottom: 30px; }
        .product-card { background: white; border-radius: 10px; overflow: hidden;
box-shadow: 0 4px 15px rgba(0,0,0,0.1); transition: transform 0.3s; }
        .product-card:hover { transform: translateY(-5px); }
        .product-image { font-size: 60px; text-align: center; padding: 40px 20px;
background: linear-gradient(135deg, #667eea 0%, #764ba2 100%); color: white; }
        .product-name { font-size: 18px; font-weight: 600; margin-bottom: 8px; }
        .product-price { font-size: 24px; color: #667eea; font-weight: bold; }
        .order-table { width: 100%; border-collapse: collapse; margin-bottom:
20px; }
        .order-table th { background: #667eea; color: white; padding: 12px; text-
align: left; }
        .order-table td { padding: 12px; border-bottom: 1px solid #ddd; }
        .stats-box { background: linear-gradient(135deg, #667eea 0%, #764ba2
100%); color: white; padding: 20px; border-radius: 10px; margin-bottom: 20px; }
        .stats-box h3 { font-size: 32px; margin-bottom: 5px; }
        .incident-log { background: #fff3cd; border: 1px solid #ffc107; border-
radius: 5px; padding: 15px; margin-bottom: 10px; }
        .form-container { max-width: 500px; margin: 0 auto; }
        .text-center { text-align: center; }
        @keyframes slideIn { from { opacity: 0; transform: translateY(-10px); } to
{ opacity: 1; transform: translateY(0); } }
    </style>
</head>
<body>
    <div class="navbar">
        <div class="navbar-content">
```

```
            <a href="/" class="navbar-brand">🛡️ SecureShop</a>
            <div class="nav-links">
                <a href="/">Товари</a>
                {% if current_user.is_authenticated %}
                    <a href="/cart">🛒 Кошик</a>
                    <a href="/profile">👤 {{ current_user.username }}</a>
                    {% if current_user.role == 'admin' %}
                        <a href="/admin" style="color: #ffc107;">⚡ Панель
безпеки</a>
                    {% endif %}
                    <a href="/logout" class="btn btn-secondary" style="margin: 0;
padding: 8px 15px;">Вихід</a>
                {% else %}
                    <a href="/login" class="btn btn-primary" style="margin: 0;
padding: 8px 15px;">Вхід</a>
                    <a href="/register" class="btn btn-secondary" style="margin:
0; padding: 8px 15px;">Реєстрація</a>
                {% endif %}
            </div>
        </div>
    </div>

    <div class="container">
        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                {% for category, message in messages %}
                    <div class="alert {{ 'success' if category == 'success' else
'danger' if category == 'danger' else 'info' }}">
                        {{ message }}
                    </div>
                {% endfor %}
            {% endif %}
        {% endwith %}

        {{ content_html|safe }}
    </div>
</body>
```

```
</html>
"""


# МАРШРУТИ

@app.route('/')
def index():
    products = Product.query.all()
    content = '<div class="card"><h1>🛍️ Каталог товарів</h1><p>Вітаємо в
SecureShop!</p></div><div class="product-grid">'
    for product in products:
        content += f'''
        <div class="product-card">
            <div class="product-image">{product.image_emoji}</div>
            <div class="product-info" style="padding: 15px;">
                <div class="product-name">{product.name}</div>
                <div class="product-price">${product.price:.2f}</div>
                <p style="font-size: 14px; color: #666; margin-bottom:
15px;">{product.description}</p>
        '''
        if current_user.is_authenticated:
            content += f'<form action="/add_to_cart/{product.id}"
method="POST"><button type="submit" class="btn btn-success" style="width:
100%;">➕ Додати</button></form>'
        else:
            content += '<p style="color: #dc3545; font-size: 12px;">⚠️ Увійдіть
для покупок</p>'
        content += '</div></div>'
    content += '</div>' # Генерування HTML зі всіма товарами
    return render_template_string(HTML_TEMPLATE, content_html=content)


@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form.get('username', '').strip()
```

```
        email = request.form.get('email', '').strip()
        password = request.form.get('password', '')

        if not username or len(username) < 3:
            flash('Ім\'я повинно містити мінімум 3 символи', 'danger')
            return redirect(url_for('register'))

        if User.query.filter_by(username=username).first():
            flash('Цей користувач вже існує', 'danger')
            return redirect(url_for('register'))

        if User.query.filter_by(email=email).first():
            flash('Цей email вже зареєстрований', 'danger')
            return redirect(url_for('register'))

        user = User(username=username, email=email,
password_hash=generate_password_hash(password))
        db.session.add(user)
        db.session.commit()
        flash(f'✅ Користувач {username} успішно зареєстрований!', 'success')
        return redirect(url_for('login'))

    content = '''
    <div class="card form-container">
        <h2>📝 Реєстрація</h2>
        <form method="POST">
            <div class="form-group">
                <label>Ім'я користувача:</label>
                <input type="text" name="username" required placeholder="Введіть
ім'я">
            </div>
            <div class="form-group">
                <label>Email:</label>
                <input type="email" name="email" required
placeholder="example@mail.com">
            </div>
            <div class="form-group">
```

```
                <label>Пароль:</label>
                <input type="password" name="password" required
placeholder="Мінімум 8 символів">
            </div>
            <button type="submit" class="btn btn-primary" style="width:
100%;">Зареєструватися</button>
        </form>
        <p class="text-center" style="margin-top: 20px;">Вже маєте аккаунт? <a
href="/login">Увійдіть</a></p>
    </div>
    '''
    return render_template_string(HTML_TEMPLATE, content_html=content)


@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username', '').strip()
        password = request.form.get('password', '')
        user = User.query.filter_by(username=username).first()

        if user and check_password_hash(user.password_hash, password):
            login_user(user)
            flash(f'✅ Вітаємо, {username}!', 'success')
            return redirect(url_for('index'))
        else:
            flash('❌ Неправильний логін або пароль', 'danger')

    content = '''
    <div class="card form-container">
        <h2>🤗 Вхід</h2>
        <form method="POST">
            <div class="form-group">
                <label>Ім'я користувача:</label>
                <input type="text" name="username" required>
            </div>
            <div class="form-group">
```

```
                <label>Пароль:</label>
                <input type="password" name="password" required>
            </div>
            <button type="submit" class="btn btn-primary" style="width:
100%;">Увійти</button>
        </form>
        <p class="text-center" style="margin-top: 20px;">Не маєте аккаунту? <a
href="/register">Зареєструйтесь</a></p>
        <hr style="margin: 20px 0;">
        <p style="font-size: 12px; color: #666;">
        </p>
    </div>
    '''
    return render_template_string(HTML_TEMPLATE, content_html=content)


@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('✅ Ви вийшли з системи', 'success')
    return redirect(url_for('index'))


@app.route('/profile')
@login_required
def profile():
    orders =
Order.query.filter_by(user_id=current_user.id).order_by(Order.created_at.desc()).a
ll()

    content = f'''
    <div class="card">
        <h2>👤 Профіль користувача</h2>
        <p><strong>Ім'я:</strong> {current_user.username}</p>
        <p><strong>Email:</strong> {current_user.email}</p>
```

```
        <p><strong>Реєстрація:</strong>
{current_user.created_at.strftime('%d.%m.%Y')}</p>
    </div>
    <div class="card">
        <h3>📦 Ваші замовлення</h3>
    '''

    if orders:
        content += '<table class="order-
table"><thead><tr><th>ID</th><th>Сума</th><th>Статус</th><th>Дата</th></tr></thead
><tbody>'
        for order in orders:
            status_color = '#28a745' if order.status == 'Completed' else '#ffc107'
if order.status == 'Pending' else '#dc3545'
            content +=
f'<tr><td>#{order.id}</td><td>${order.total_amount:.2f}</td><td style="color:
{status_color};">{order.status}</td><td>{order.created_at.strftime("%d.%m.%Y
%H:%M")}</td></tr>'
        content += '</tbody></table>'
    else:
        content += '<p style="color: #666;">У вас поки немає замовлень. <a
href="/">Почніть покупки</a></p>'
    content += '</div>'

    return render_template_string(HTML_TEMPLATE, content_html=content)


@app.route('/add_to_cart/<int:product_id>', methods=['POST'])
@login_required
def add_to_cart(product_id):
    product = Product.query.get(product_id)
    if not product:
        flash('❌ Товар не знайдено', 'danger')
        return redirect(url_for('index'))

    if 'cart' not in session:
        session['cart'] = {}
```

```
    session['cart'][str(product_id)] = session['cart'].get(str(product_id), 0) + 1
    session.modified = True
    flash(f'✅ {product.name} додано в кошик', 'success')
    return redirect(url_for('index'))


@app.route('/cart')
@login_required
def cart():
    cart_ids = session.get('cart', {})
    if not cart_ids:
        flash('ℹ️ Ваш кошик порожній', 'info')
        return redirect(url_for('index'))

    products = Product.query.filter(Product.id.in_(map(int,
cart_ids.keys()))).all()
    total = 0
    content = '<div class="card"><h2>🛒 Ваш кошик</h2><table class="order-
table"><thead><tr><th>Товар</th><th>Кількість</th><th>Ціна</th><th>Сума</th></tr><
/thead><tbody>'

    for product in products:
        qty = cart_ids[str(product.id)]
        cost = product.price * qty
        total += cost
        content +=
f'<tr><td>{product.name}</td><td>{qty}</td><td>${product.price:.2f}</td><td>${cost
:.2f}</td></tr>'

    content += f'</tbody></table><h3 style="text-align: right; margin-top: 20px;
color: #667eea;">Разом: ${total:.2f}</h3>'
    content += '<form action="/checkout" method="POST" style="margin-top:
20px;"><button type="submit" class="btn btn-success">💳 Оформити
замовлення</button> <a href="/" class="btn btn-secondary">←
Назад</a></form></div>'
```

```
    return render_template_string(HTML_TEMPLATE, content_html=content)


@app.route('/checkout', methods=['POST'])
@login_required
def checkout():
    cart_ids = session.get('cart', {})
    if not cart_ids:
        flash('⚠️ Кошик порожній', 'danger')
        return redirect(url_for('cart'))

    products = Product.query.filter(Product.id.in_(map(int,
cart_ids.keys()))).all()
    total = 0
    items_list = []

    for product in products:
        qty = cart_ids[str(product.id)]
        total += product.price * qty
        items_list.append(f"{product.name} x{qty}")

    order = Order(user_id=current_user.id, total_amount=total, items_json="
".join(items_list),
                  status='Pending')
    db.session.add(order)
    db.session.commit()

    session.pop('cart', None)
    session.modified = True

    flash(f'✅ Замовлення #{order.id} успішно оформлено! Сума: ${total:.2f}',
'success')
    return redirect(url_for('profile'))


@app.route('/admin')
@login_required
```

```
def admin_dashboard():
    if current_user.role != 'admin':
        flash('❌ Доступ заборонено', 'danger')
        return redirect(url_for('index'))

    logs = threat_engine.logs[-10:]
    blocked_ips = list(threat_engine.blacklist)
    total_users = User.query.count()
    total_orders = Order.query.count()
    total_threats = len(threat_engine.logs)

    content = f'''
    <div class="card">
        <h2>⚡ Панель безпеки (SIEM)</h2>
        <p>Система моніторингу загроз в реальному часі</p>
    </div>
    <div style="display: grid; grid-template-columns: repeat(auto-fit,
minmax(200px, 1fr)); gap: 20px; margin-bottom: 30px;">
        <div class="stats-box"><h3>{total_users}</h3><p>Користувачів</p></div>
        <div class="stats-box"><h3>{total_orders}</h3><p>Замовлень</p></div>
        <div class="stats-box"><h3>{len(blocked_ips)}</h3><p>Заблокованих
IP</p></div>
        <div class="stats-box"><h3>{total_threats}</h3><p>Виявлено
загроз</p></div>
    </div>
    <div class="card">
        <h3>🚨 Журнал інцидентів (Real-time)</h3>
    '''

    if logs:
        for log in reversed(logs):
            content += f'<div class="incident-log"><div class="time">⏰
{log["time"]}</div><div><span style="background: #f0f0f0; padding: 2px 6px;
border-radius: 3px; font-family: monospace;">{log["ip"]}</span> <strong
style="color: #d39e00;">{log["type"]}</strong> <small style="color:
#666;">"{log["payload"]}"</small></div></div>'
    else:
```

```
        content += '<p style="color: #28a745;">✅ Немає загроз. Система працює в нормальному режимі.</p>'

    content += '</div><div class="card"><h3>🚫 Заблоковані IP адреси</h3>'
    if blocked_ips:
        for ip in blocked_ips:
            content += f'<code style="background: #f0f0f0; padding: 3px 6px; margin-right: 5px; border-radius: 3px;">{ip}</code>'
    else:
        content += '<p style="color: #28a745;">✅ Немає заблокованих IP</p>'
    content += '</div>'

    return render_template_string(HTML_TEMPLATE, content_html=content)


# ІНІЦІАЛІЗАЦІЯ

def init_db():
    with app.app_context():
        db.create_all()

        if not User.query.filter_by(username='admin').first():
            admin = User(username='admin', email='admin@shop.com', password_hash=generate_password_hash('admin123'),
                         role='admin')
            db.session.add(admin)

        if not Product.query.first():
            products = [
                Product(name="Захищений Ноутбук", price=1299.99, description="Ноутбук з апаратним шифруванням",
                        image_emoji="💻"),
                Product(name="YubiKey 5", price=49.99, description="Апаратний ключ безпеки 2FA", image_emoji="🔐"),
                Product(name="VPN Підписка", price=9.99, description="Річна підписка на VPN", image_emoji="🌐"),
```

```
                Product(name="Антивірус", price=29.99, description="Ліцензія на
антивірус", image_emoji="🛡️"),
                Product(name="Чохол для телефону", price=19.99,
description="Захисний чохол", image_emoji="📱"),
                Product(name="USB Флешка 64GB", price=24.99,
description="Зашифрована флешка", image_emoji="💾"),
            ]
            db.session.add_all(products)

        db.session.commit()
        print("✅ База даних ініціалізована!")


if __name__ == '__main__':
    init_db()
    app.run(debug=True, port=5000, host='0.0.0.0')
```

Додаток Б

Презентація

Методи виявлення та нейтралізації загроз для безпеки даних у системах електронної комерції

Здобувач: Хімченко О. С. ІСТм-24
Керівник: д.т.н., професор Бородавка Є. В.

ВСТУП

Розвиток інформаційних технологій та цифровізація економіки сприяли стрімкому зростанню електронної комерції, яка стала невід'ємною частиною повсякденного життя мільйонів користувачів.

Разом з розширенням електронної комерції, експоненціально зростають масштаби та складність загроз інформаційній безпеці. Системи е-комерції обробляють й акумулюють величезні обсяги критично важливих даних: персональну інформацію користувачів, платіжні реквізити, комерційні таємниці та історії транзакцій. Така концентрація чутливої інформації робить ці платформи привабливою мішенню для кіберзлочинців, хакерів та інших загроз.

На сьогодні продовжується постійне зростання кількості успішних кібератак на системи електронної комерції, що призводять до витоків даних, фінансових збитків для компаній та втрати довіри споживачів до цифрових каналів торгівлі.

3

Аналіз предметної області та постановка задачі

Предметною областю цієї роботи є **електронна комерція** (e-commerce) – галузь економіки, в якій реклама, просування продуктів, торговельні угоди та фінансові транзакції здійснюються безпосередньо в Інтернеті. Коли ви щось купуєте чи продаєте у мережі, це і є e-commerce.

E-commerce розділяють на шість основних видів залежно від взаємодії сторін: клієнтів, бізнесу та адміністрацій. Видами та типами електронної комерції є:

- **B2B** – електронна комерція для бізнесу. Це різновид електронної комерції, який працює за принципом "від бізнесу - бізнесу" (Business-to-Business);

4

Аналіз предметної області та постановка задачі

- **B2C** – електронна комерція для споживача. Наразі модель Business-to-Consumer (B2C) – це найпоширеніший вид електронної комерції;
- **C2C** – електронна комерція між споживачами. Це модель електронної комерції, в межах якої один споживач продає щось іншим споживачам, маючи з ними рівний статус (Consumer-to-Consumer);
- **C2B** – електронна комерція від споживача до бізнесу. Це фактично пряма протилежність B2C, оскільки в цьому випадку вже споживач надає певні товари та послуги бізнесу (Consumer-to-Business);
- **B2A** – бізнес-адміністрування. Модель B2A (Business-to-Administration) або B2G (Business-to-Government) подібна B2B;
- **C2A** – електронна комерція між споживачами та адміністрацією (Consumer-to-Administration).

5

Аналіз предметної області та постановка задачі

Мета роботи полягає у розробці та обґрунтуванні комплексу методів виявлення та нейтралізації загроз для безпеки даних у системах електронної комерції для автоматизації та спрощення оцінки стану безпеки, а також забезпечення швидкого та своєчасного реагування на виявлені загрози.

Об'єктом дослідження є дані у системах е-комерції, загрози безпеці та процеси їх виявлення та нейтралізації, що охоплюють автентифікацію користувачів, управління доступом, моніторинг операцій та реагування на інциденти.

Предметом дослідження є методи та алгоритми виявлення загроз, модельні підходи до класифікації атак та аномалій, а також механізми криптографічного захисту даних та управління аутентифікацією в e-commerce системах.

6

Види даних та вимоги до безпеки

Які є **види даних** в системах електронної комерції:

- Персональні дані користувача
- Платіжні та фінансові дані
- Комерційні дані та бізнес-логіка
- Системні логи та журнали аудиту

Вимоги до інформаційної безпеки складає традиційна **тріада CIA** (Confidentiality, Integrity, Availability) але є недостатнім базисом для сучасних систем електронної комерції. Тому вона вимагає розширення ще двома додатковими елементами: Автентичністю (Authenticity) та Невідмовністю (Non-repudiation).

7

Класифікація загроз та методи їх виявлення

У контексті даної роботи було виділено такі основні **групи загроз**:

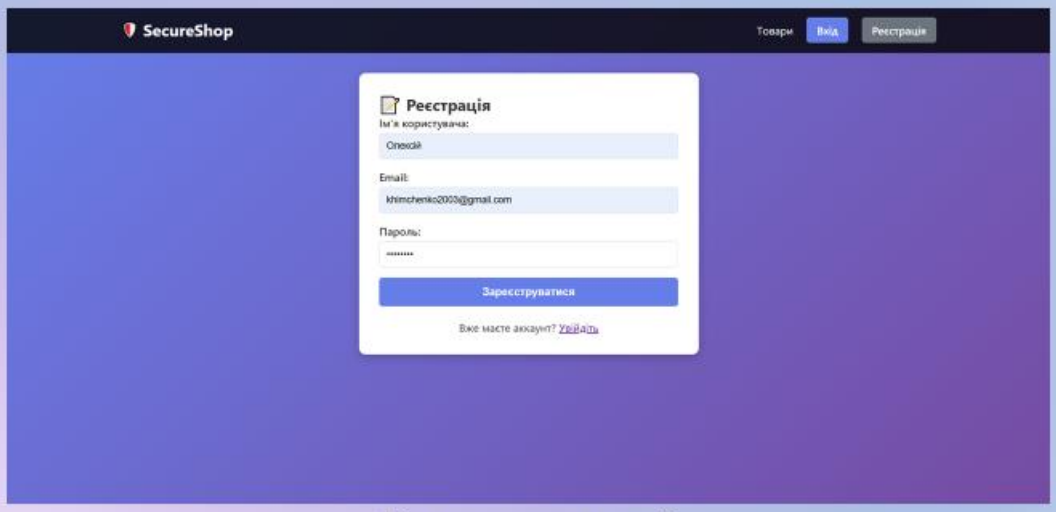
- атаки на веб-додаток;
- атаки на мережевій інфраструктурі;
- атаки на облікові записи користувачів;
- загрози, пов'язані з платіжними операціями;
- загрози на основі соціальної інженерії та внутрішніх зловживань.

Методи виявлення загроз у системах електронної комерції можна поділити на такі групи:

- сигнатурні (правилові) методи;
- методи аномалійної детекції;
- методи машинного та глибинного навчання;
- поведінковий аналіз;
- контекстний та кореляційний аналіз подій.

8

Практична реалізація



Процес реєстрації

Збереження даних користувача

9

Після успішної реєстрації, в базі даних з'являється запис про нового користувача. Як можна помітити, збережений пароль є зашифрованим

	id	username	email	password_hash	role	created_at
1	1	admin	admin@shop.com	e935887960d78ac38fd67cf1c4a6aa8c71376c81dd1c546e2793ef919fd2e2c48b5b2a241f0ef48f130b01fef7982065b295a1fb85b4ec6afa6	admin	2025-12-06 21:24:04.486003
2	2	Олексій	khimchenko2003@gmail.com	scrypt:...	user	2025-12-06 21:27:54.667503

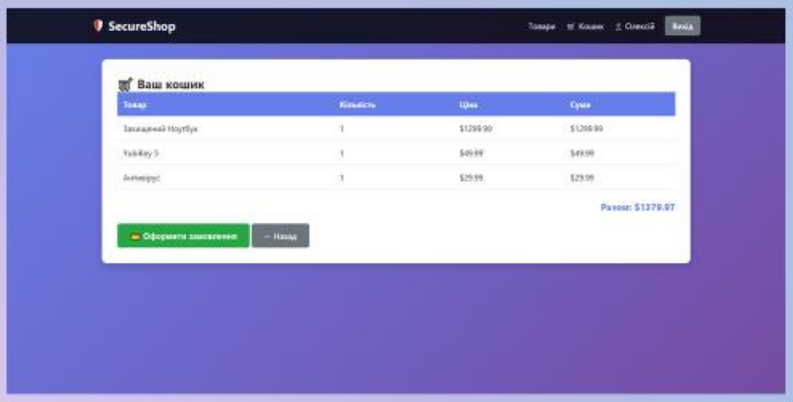
Для шифрування використовується бібліотека werkzeug мови програмування python

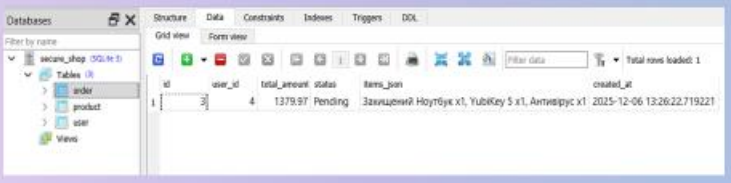
```
@app.route( rule: '/register', methods=['GET', 'POST']) 3 usages
def register():
    if request.method == 'POST':
        username = request.form.get('username', '').strip()
        email = request.form.get('email', '').strip()
        password = request.form.get('password', '')
```

```
user = User(username=username, email=email, password_hash=generate_password_hash(password))
db.session.add(user)
```

Процес замовлення товару і додавання до БД

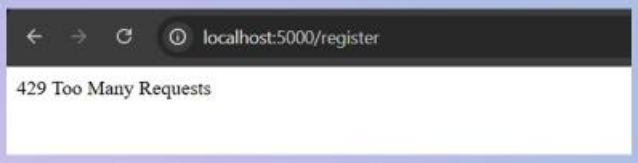
10





11

Перевірка безпеки



Після надсилання великої кількості запитів, система сприймає це як загрозу і блокує IP адресу, з якої була підозріла активність і надсилається попередження про блокування IP адреси.

12

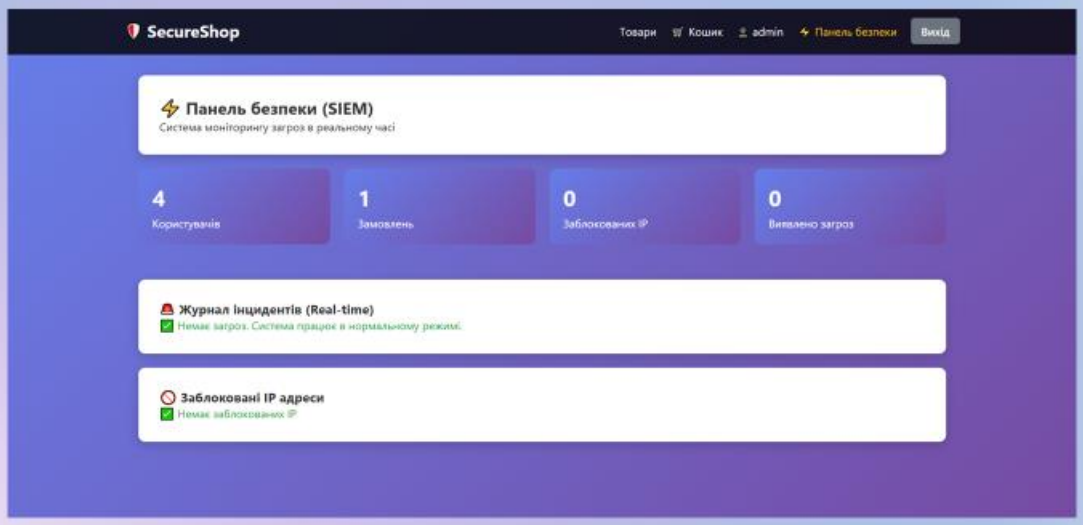
Реалізація безпеки від DDoS атаки

```python
now = time.time()
if ip not in threat_engine.request_history:
    threat_engine.request_history[ip] = []

threat_engine.request_history[ip] = [t for t in threat_engine.request_history[ip] if now - t < 10]
threat_engine.request_history[ip].append(now)

if len(threat_engine.request_history[ip]) > 30:
    threat_engine.block_ip(ip)
    threat_engine.log_incident(ip, attack_type: "DDoS Attack", payload: "Too many requests")
    return "429 Too Many Requests", 429
```

Захист від DDoS атак (надмірної кількості запитів) реалізується за допомогою перевірки кількості запитів, де запити старіше 10 секунд видаляються, а якщо кількість запитів перевищує 30, тоді IP потрапляє до чорного списку.

13

Панель безпеки адміна



14

Висновок

Під час виконання роботи було досліджено види даних в електронній безпеці та вимоги до їхньої безпеки. Було розглянуто класифікацію загроз і методи їх виявлення.

Після теоретичного аналізу було спроєктовано архітектуру системи і виконання програмної реалізації захищеної системи, яка враховує вимоги інформаційної безпеки.

Проведено тестування розробленої системи, під час якого відбулась перевірка хешування паролів користувачів і змодельовано DDoS атаку, яка була успішно виявлена та знешкоджена.