Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка програмного засобу телеметричного моніторингу аномальних активностей програмних компонентів для захисту інформаційних систем з використанням технології ШІ та ELK stack»

Виконав: студент 2(4) курсу, гр.1КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Присяжна Д. О._____
(прізвище та ініціали)

Керівник: к.т.н., доц., кер. каф. МБІС
Карпінець В. В._____
(прізвище та ініціали)
«____» _____ 2025 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
Колесник І. С._____
(прізвище та ініціали)
«____» _____ 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
“____” _____2025р.

Вінниця ВНТУ – 2025

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

д.т.н., проф. Яремчук Ю.Є.
" 20 " березня 2025 р.

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Присяжна Дар'я Олександрівна
(прізвище, ім`я, по-батькові)

1. Тема роботи Розробка програмного засобу телеметричного моніторингу аномальних активностей програмних компонентів для захисту інформаційних систем з використанням технології ШІ та ELK stack
Керівник роботи к.т.н, доц., доцент каф. МБІС Карпінець Василь Васильович
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 20 " березня 2025року № 97
2. Термін подання студентом роботи
3. Вихідні дані до роботи: Вимоги до систем телеметричного моніторингу для інформаційної безпеки, архітектура стека ELK (Elasticsearch, Logstash, Kibana) для зберігання логів; алгоритми машинного навчання для виявлення аномальної активності; потік обробки телеметрії; практичні вимоги до інтеграції штучного інтелекту в систему.
4. Зміст текстової частини: У роботі описано етапи проєктування, розробки та впровадження програмного засобу телеметричного моніторингу аномальної активності для захисту інформаційних систем. Надано опис архітектури системи, логіки взаємодії її компонентів, алгоритмів виявлення аномалій із використанням методів машинного навчання, а також принципів інтеграції з технологіями ELK Stack. Представлено результати тестування працездатності розробленого засобу
5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
– Рисунок 1.1 – Приклад діаграми трафіку мережі з двома аномаліями;
– Рисунок 1.2 – Приклад графіку з точковими аномаліями;
– Рисунок 1.3 – Приклад графіку з контекстуальними аномаліями
– Рисунок 1.4 – Приклад графіку з колективною аномалією
– Рисунок 1.5 – Схематичне зображення збору телеметричних даних

– Рисунок 1.6 – Схематичне зображення роботи ELK Stack
– Таблиця 1.1 –Порівняння програмних засобів телеметричного моніторингу;
– Рисунок 2.1 – Схема руху даних програмним засобом;
– Рисунок 2.5 – Блок схема обробки телеметричних даних
– Рисунок 2.6 – Блок схема обробки
– Таблиця 3.1 – Порівняння VPS та хмарного хостингу;
– Рисунок 3.1 – Створення умови в Kibana, з порогом відхилення 75
– Рисунок 3.2 –Інфраструктура середовища розгортання.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Карпінець В.В. к.т.н., доцент каф. МБІС		
Розділ II	Карпінець В.В. к.т.н., доцент каф. МБІС		
Розділ III	Карпінець В.В. к.т.н., доцент каф. МБІС		

7.Дата видачі завдання 20 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025	Виконано
2	Аналіз предметної області обраної теми	24.03.2025	13.04.2025	Виконано
3	Розробка алгоритму роботи	14.04.2025	27.04.2025	Виконано
4	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025	Виконано
5	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025	Виконано
6	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	08.06.2025	Виконано
7	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025	Виконано

Студент (підпис) Присяжна Д.О. (прізвище та ініціали)

Керівник роботи (підпис) Карпінець В. В. (прізвище та ініціали)

АНОТАЦІЯ

Дана дипломна робота присвячена розробці програмного засобу для телеметричного моніторингу аномальних активностей програмних компонентів, спрямованого на захист інформаційних систем із використанням технологій штучного інтелекту та стеку ELK.

У першому розділі проаналізовано сучасні методи та інструменти для забезпечення безпеки інформаційних систем, досліджено концепції аномалій у контексті кіберзагроз, а також проведено огляд можливостей застосування штучного інтелекту та стеку ELK для моніторингу.

У другому розділі детально розглянуто процес проектування програмного засобу, включаючи визначення вимог, розробку архітектури системи, проектування бази даних для зберігання телеметричних даних та інтеграцію компонентів штучного інтелекту.

Третій розділ присвячено програмній реалізації розробленого рішення, впровадженню алгоритмів виявлення аномалій, налаштуванню та оптимізації стеку ELK для обробки телеметрії, а також тестуванню програмного засобу з метою аналізу його ефективності.

Розроблений програмний засіб дозволяє своєчасно виявляти потенційні кіберзагрози, забезпечуючи підвищений рівень захисту інформаційних систем.

Ключові слова: кібербезпека, телеметрія, аномалії, штучний інтелект, ELK Stack, моніторинг.

ABSTRACT

This thesis is devoted to developing a software tool for telemetric monitoring of anomalous activities of software components aimed at protecting information systems using artificial intelligence technologies and the ELK stack.

The first section analyzes modern methods and tools for ensuring information system security, investigates anomalies in the context of cyber threats, and reviews the possibilities of using artificial intelligence and the ELK stack for monitoring.

The second section describes in detail the software design process, including the definition of requirements, system architecture development, design of a database for storing telemetry data, and integration of artificial intelligence components.

The third section is devoted to the software implementation of the developed solution, the implementation of anomaly detection algorithms, the configuration and optimization of the ELK stack for telemetry processing, and the testing of the software tool to analyze its effectiveness.

The developed software tool allows timely detection of potential cyber threats, providing increased protection for information systems.

Keywords: cybersecurity, telemetry, anomalies, artificial intelligence, ELK Stack, monitoring.

ЗМІСТ

ВСТУП

У даній дипломній роботі описано процес розробки програмного засобу для телеметричного моніторингу аномальних активностей програмних компонентів, що спрямований на підвищення рівня захисту інформаційних систем.

Актуальність задачі. Аномальні активності можуть сигналізувати про зловмисні дії, тому їх моніторинг є важливою складовою захисту інформаційних систем. Одним із ефективних підходів є використання телеметричних даних у поєднанні з методами штучного інтелекту, що дозволяє автоматизувати процес виявлення загроз. Сучасний стан технологій і постійне зростання кількості кібератак ставлять перед фахівцями з кібербезпеки складне завдання — своєчасне виявлення та запобігання загрозам.

Актуальність дослідження полягає в необхідності розробки рішень, здатних забезпечити своєчасне виявлення загроз в умовах постійного ускладнення атак. Проаналізувавши сучасні методи моніторингу, архітектурні підходи та засоби обробки даних, у роботі реалізовано прототип системи, що відповідає вимогам до захисту інформаційних систем

Метою даної бакалаврської кваліфікаційної роботи є підвищення рівня захищеності інформаційних систем у реальному часі шляхом створення програмного засобу телеметричного моніторингу, що забезпечує точне та своєчасне виявлення аномальних активностей програмних компонентів на основі алгоритмів штучного інтелекту та стеку ELK. Реалізація запропонованого підходу дозволяє досягти вищої швидкості обробки даних, зменшення кількості хибнопозитивних спрацювань, покращення достовірності виявлення загроз та стійкості системи до змін у поведінці користувачів і середовища, порівняно з наявними аналогами.

Для досягнення цієї мети необхідно вирішити такі **задачі**:

1. Розробити архітектуру програмного засобу, що забезпечить ефективний збір та обробку телеметричних даних інформаційної системи.

2. Визначити ключові параметри моніторингу для виявлення різних типів аномальних активностей.

3. Реалізувати інтеграцію технологій штучного інтелекту та машинного навчання для автоматичного аналізу телеметричних даних та виявлення аномалій.

4. Візуалізація результатів моніторингу, який забезпечить наочне представлення виявлених аномалій та спростить аналіз інцидентів безпеки.

5. Забезпечити рішення для обробки телеметричних даних з використанням ELK stack.

6. Реалізувати механізм оповіщення та реагування на виявлені аномальні активності, який дозволить своєчасно вживати заходів для запобігання та нейтралізації загроз.

7. Тестування програмного засобу для оцінки його ефективності в реальних умовах експлуатації.

Предметом дослідження є захист інформаційних систем в реальному часі, на основі аналізу телеметричних даних алгоритмами ШІ.

Об’єктом дослідження є програмні засоби збору телеметричних даних, засоби обмеження доступу в інформаційних системах, методи ШІ для виявлення аномальних активностей.

Практична цінність отриманих результатів полягає в збільшенні стійкості інформаційних систем за рахунок зменшення часу на реагування. Використання прогресивних методів ШІ, дозволяють миттєво блокувати аномальну активність, та повідомити адміністраторів про інцидент. Дане рішення в змозі зменшити витрати на захист, завдяки тому що здатне до самонавчання та не потребує обслуговування.

Сформовані підходи можуть бути адаптовані для різних типів інформаційних систем і застосовані в умовах реальної експлуатації. Отримані результати створюють основу для подальшого вдосконалення інструментів безпеки з урахуванням сучасних технологічних викликів.

1 ТЕОРЕТИЧНІ ОСНОВИ СУЧАСНИХ МЕТОДІВ МОНІТОРИНГУ АНОМАЛЬНИХ АКТИВНОСТЕЙ ІНФОРМАЦІЙНИХ СИСТЕМ ТА МЕРЕЖ

У даному розділі проведено комплексний аналіз теоретичних засад та підходів до моніторингу аномальних активностей в інформаційних системах та мережах. Розділ розкриває фундаментальні концепції аномалій у контексті кіберзагроз та досліджує їхні типи та методи виявлення. Також проведено огляд SIEM-рішень, які є важливим інструментом централізованого моніторингу безпеки й виявлення загроз. Окрему увагу приділено телеметричному моніторингу, як перспективному напрямку розвитку технологій збору та аналізу даних про активність у системах та мережах. Проаналізовано архітектуру та принципи роботи ELK stack у контексті обробки великих обсягів телеметричних даних. Значну увагу приділено інтеграції технологій штучного інтелекту та машинного навчання в системи виявлення аномалій.

На основі проведеного аналізу обґрунтовано доцільність розробки програмного засобу телеметричного моніторингу, що поєднує переваги технологій штучного інтелекту та масштабованої аналітичної платформи ELK stack для ефективного виявлення аномальних активностей у сучасних інформаційних системах.

1.1 Аналіз концепцій аномалій у контексті кіберзагроз

На теперішній день неможливо недооцінити важливість ролі, яку відіграють інформаційні системи та мережі у функціонуванні бізнесу, державного управління та суспільства в цілому. Як наслідок, високий рівень критичності даних провокує високу ймовірності виникнення атак та кіберзагроз.

Серед найпоширеніших форм порушення кібербезпеки є анамольна активність, що частіше за все стає передумовою можливої атаки, зловмисних дій чи несправності інформаційної системи чи мережі. Аналіз концепцій аномалій дає

змогу зрозуміти природу таких явищ, класифікувати їх та визначити підходи до виявлення і запобігання.

Аномалія (грец. ἀνωμαλία – нерівність, відхилення) – неправильність, відхилення від загальної закономірності, ненормальність. [1]

Якщо використовувати даний термін у контексті кібербезпеки інформаційних систем та мереж, то будь-яке відхилення від стандартної поведінки системи вважається аномалією й може свідчити про потенційну загрозу.

На рис. 1.1 наведено наочний приклад аномальних активностей, що позначені червоними точками на діаграмі трафіку мережі.

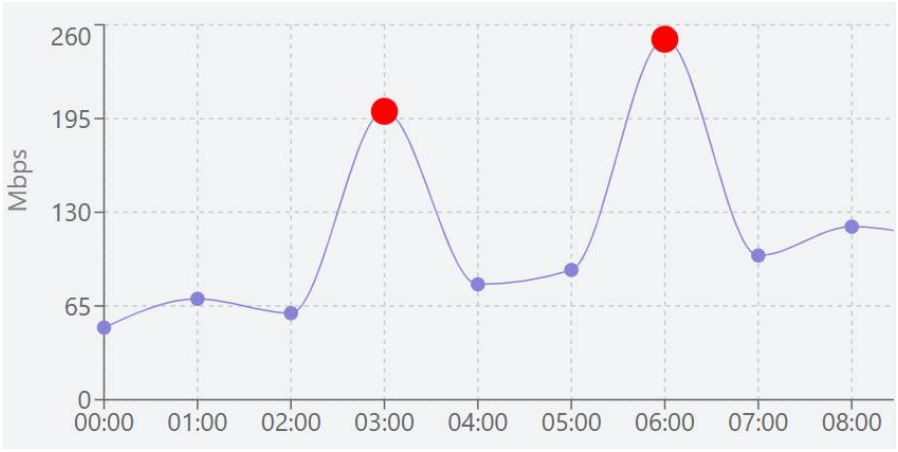


Рисунок 1.1 – Приклад діаграми трафіку мережі з двома аномаліями

Загалом виділяють три основних типи аномалій: точкові, контекстні та колективні. Кожен тип аномалій підсвічує різні вразливості інформаційних систем й відіграє важливу роль у виявленні загроз для кібербезпеки.

Найординарнішим видом аномалії в контексті кібербезпеки вважається точкова аномалія. Вона характеризується появою одного об’єкта даних, що суттєво відрізняється від загальної маси даних і відхиляється від норми. Наприклад, точковий сплеск (звідси й назва аномалії) мережевої активності, як от сплеск спроб входу на веб-сайт, свідчать про появу точкової аномалії, яка в свою чергу може супроводжуватися порушенням безпеки або несанкціонованим доступом. [2]

На рис. 1.2 яскраві червоні точки, що знаходяться за межою звичної області даних є точковими аномаліями, оскільки вони розташовані далеко на графіку відносно типових точок даних.

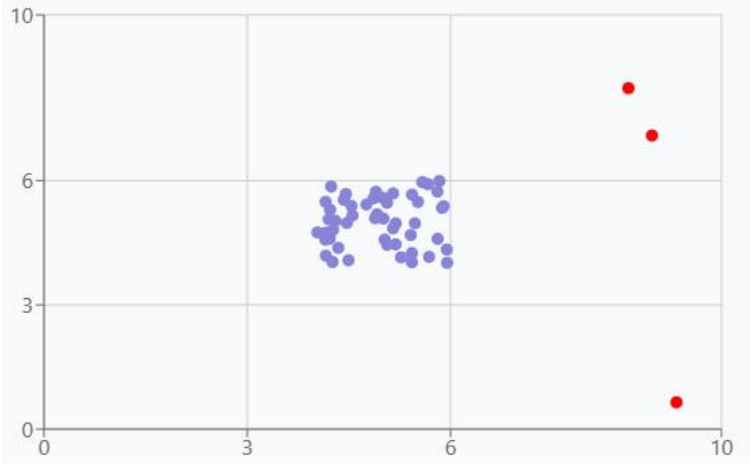


Рисунок 1.2 – Приклад графіку з точковими аномаліями

Контекстуальними аномаліями або також відомими як умовними аномаліями прийнято вважати об’єкти даних, що лише за певних обставин, контексту є нетиповим. Контекст у даній ситуації визначається набором даних і є частиною формулювання проблеми.

Щоб зрозуміти чи об’єкт даних знаходиться в межах норми відносно контексту, то до нього застосовують два набори зміних: контекстні та поведінкові. Контекстні змінні використовуються для визначення контексту та визначення референтних груп (об'єкти, які мають подібну контекстну інформацію), тоді як поведінкові змінні допомагають виміряти, наскільки об'єкт суттєво відхиляється від своєї референтної групи. [3]

Як приклад контекстної аномалії уявімо ситуацію, що ви передаєте великий файл в робочий час на комп’ютер керівника вашого підприємства, що зазвичай є цілком прийнятним, але підозрілою дана ситуація стане тоді, коли це відбуватиметься пізно вночі. Контекстні аномалії часто сигналізують про інсайдерські загрози або компрометації акаунтів.

На рис. 1.3 яскраві червоні точки є контекстуальними аномаліями, оскільки попри те, що вони знаходяться у межах звичної області даних, при наявності відхилень в змінних контексту та поведінки вони набувають аномальних рис.

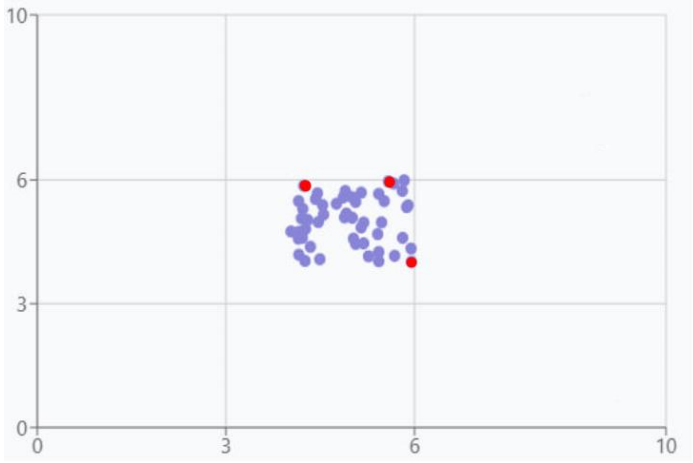


Рисунок 1.3 – Приклад графіку з контекстуальними аномаліями

Якщо певна сукупність об’єктів даних відхиляються разом по відношенню до всього набору даних, то це вважається колективною аномалією. Тобто поява певних даних водночас у групі є основною ознакою даної аномалії, оскільки самі по собі окремі об’єкти даних, що мають схоже значення можуть не бути аномаліями. Наприклад, системи, які одночасно взаємодіють зі зловмисним сервером або скоординовані атаки, такі як DDoS є наслідками колективних аномалій. Такі аномалії зустрічаються рідше, але, як правило, вказують на більш складні, організовані загрози. [4]

На рис. 1.4 наведено приклад колективної аномалії, що позначена червоними точками на графіку.

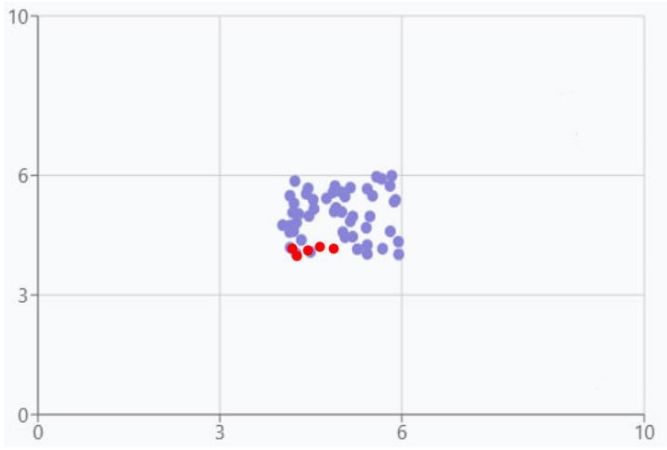


Рисунок 1.4 – Приклад графіку з колективною аномалією

Слід зазначити, що в той час як точкові аномалії можуть виникати в будь-якому наборі даних, колективні аномалії можуть виникати тільки в наборах даних,

в яких екземпляри даних пов'язані між собою. На відміну від них, виникнення контекстних аномалій залежить від наявності атрибутів контексту в даних. Точкова аномалія або колективна аномалія також може бути контекстною аномалією, якщо аналізувати її з урахуванням контексту. Таким чином, проблема виявлення точкових або колективних аномалій може бути перетворена на проблему виявлення контекстних аномалій шляхом включення інформації про контекст.[2]

Виявлення аномалій - це процес, спрямований на виявлення об'єктів, які відхиляються від очікуваної поведінки. Існує кілька підходів до виявлення аномалій в кібербезпеці [5]:

а) Традиційні методи – данні методи були зосереджені на виявленні статистичних аномалій у даних, таких як викиди або відхилення від норми. Однак ці методи часто не могли виявити більш складні кіберзагрози, оскільки зловмисники навчилися краще уникати класичних методів виявлення аномалій.

б) Виявлення аномалій на основі графів набувають все більшого значення як альтернатива традиційним методам. Ці алгоритми використовують взаємозв'язки між точками даних, представленими у вигляді вузлів і ребер графа. Цей підхід особливо добре підходить для складних наборів даних, таких як бази даних. Серед ключових методів виявлення аномалій на основі графів є графові нейронні мережі (GNN), виявлення поведінкових аномалій на основі графів (GBBAD), граф поведінкової ідентифікації (BIG) та виявлення ботнетів на основі графів (GBBD).

в) Нелінійний аналіз часових рядів – це підхід, що використовує нелінійні моделі для аналізу даних часових рядів і виявлення аномалій в додатках кібербезпеки, особливо в контексті хмарних систем IoT.

г) Варіаційні автокодери використовують для вивчення нормальних шаблонів у даних і виявлення аномалій на основі помилки реконструкції.

ґ) Ансамблеве навчання для виявлення аномалій – це метод з використанням мета-алгоритмів, які використовують кілька різних алгоритмів або один алгоритм з різними пороговими значеннями, що визначають, що є аномалією, на одному наборі даних. Цей підхід може використовувати сильні сторони різних алгоритмів виявлення аномалій для досягнення більшої точності та надійності у виявленні

незвичних або зловмисних дій. Одним із прикладів підходу на основі ансамблевого навчання є використання байєсівського аналізу чутливості гіперпараметрів.

Аналіз аномалій у контексті кіберзагроз є першим основним етапом для розробки програмного засобу моніторингу цих самих аномалій для захисту інформаційних систем. Розуміння концепції аномалії, їхніх типів та методів їхнього виявлення, дозволяє обґрунтувати вибір технологій для розробки програмних засобів. Хоча аномалії є важливими сигналами загроз, їх виявлення у інформаційних системах залишається складним завданням, що потребує спеціалізованих методів. А отже саме правильно обрана система превентивних заходів відіграє ключову роль у захисті даних, оскільки дозволяє не лише виявляти аномалії, а й запобігати потенційним атакам.

1.2 Огляд SIEM рішення для моніторингу безпеки інформаційних систем та мереж

Зв’язок між аномаліями та їхнім моніторингом є безумовним, оскільки ефективне виявлення аномалій неможливе без сучасних методів моніторингу інформаційних систем і мереж. На даний момент існує безліч рішень, що можуть забезпечити моніторинг даних, проте застосування усіх рішень водночас є хаотичним та беззмістовним.

Звичайний збір даних без належного аналізу не дасть результату, тоді як SIEM система забезпечує комплексний підхід—збір, кореляцію та оцінку подій безпеки. Отже, у даному підрозділі буде розглянуто SIEM рішення, яке дозволяє ефективно відстежувати аномалії, що становлять загрозу.

Система візуалізації та управління подіями (security information and event management або SIEM) – об'єднання двох термінів, що позначають область застосування ПЗ: SIM (Security information management) – управління інформацією про безпеку, і SEM (Security event management) – управління подіями безпеки. Розв'язувані завдання [6]:

– збір, обробка та аналіз подій безпеки, що надходять до системи з багатьох джерел;

– виявлення в режимі реального часу атак та порушень критеріїв та політик безпеки;

– оперативна оцінка захищеності інформаційних, телекомунікаційних та інших критично важливих ресурсів;

– аналіз та управління ризиками безпеки;

– проведення розслідувань інцидентів;

– прийняття ефективних рішень щодо захисту інформації;

– формування звітних документів.

Технологія SIEM забезпечує аналіз журналів подій безпеки та інших телеметричних даних з різних джерел в IT-інфраструктурі організації в режимі реального часу. Основною метою SIEM є моніторинг та управління подіями, пов'язаними з безпекою, виявлення загроз та оперативне реагування на інциденти безпеки. [7]

SIEM є важливим компонентом кібербезпеки та ефективного моніторингу аномалій. Таке рішення дозволяє збирати та аналізувати великі обсяги даних з додатків, пристроїв та серверів у режимі реального часу. Об'єднуючи цей величезний масив даних у єдину уніфіковану платформу, рішення SIEM надає комплексне уявлення про стан безпеки організації, дозволяючи центрам управління безпекою (security operation centers або SOC) швидко і ефективно виявляти, розслідувати і реагувати на інциденти безпеки. [8]

Рішення SIEM починаються зі збору та зберігання величезних обсягів даних, переважно у вигляді логів та мережевих даних.

Логи (log-файли, журнали) - це текстові файли, які допомагають адміністраторам визначити час і характер конкретних дій або подій, що відбулися в операційній системі, додатку, хмарному або іншому середовищі. Вони можуть деталізувати будь-що: запити на вхід/вихід, звіти про помилки, запити на файли або їх передачу, а також багато інших типів подій. [9]

Мережеві дані містять джерело, пункт призначення, номери портів, мітки часу та інші метадані, згенеровані мережевим трафіком. Мережеві дані можуть надходити у вигляді даних мережевого потоку, логів брандмауера, логів

балансувальника навантаження, хмарних логів, систем виявлення вторгнень, перехоплень пакетів тощо. [9]

У той час як системні журнали й логи додатків забезпечують повний запис конкретних подій або дій, мережеві дані надають ширшу картину моделей поведінки. Разом вони створюють надійну базу даних, яку інструменти SIEM потім використовують.

Отже, на SIEM рішення встановлюють певні умови або правила, які спрямовують автоматизовану систему на виявлення сигнатур, шаблонів або паттернів, пов'язаних з відомими векторами атак або узгоджених з вимогами безпеки підприємства. Щоразу, коли система виявляє збіг, SIEM генерує сповіщення, яке може мати форму відображення на інформаційній панелі, електронного листа або іншого повідомлення.

SIEM рішення є ключовим інструментом для комплексного моніторингу безпеки, оскільки воно поєднує збір, аналіз та кореляцію даних. Як було зазначено раніше, SIEM аналізує загрози, базуючись на телеметричних даних. Саме ці дані дозволяють виявляти аномалії в режимі реального часу.

1.3 Поняття телеметрії та телеметричного моніторингу

Телеметрія – це сукупність технічних засобів автоматичного збору, моніторингу та дистанційної передачі даних про стан інформаційної системи з віддалених джерел або важкодоступних місць. [7]

У сфері кібербезпеки телеметрія передбачає безперервне отримання та опрацювання даних про мережевий трафік, функціонування системних компонентів та інциденти безпеки з метою ідентифікації можливих вразливостей та атак.

На відміну від традиційних статичних підходів, телеметричний моніторинг являє собою динамічну, адаптивну методологію неперервного спостереження за даними, що дозволяє миттєво реагувати на найменші зміни в системному середовищі.

Ключова роль телеметричного моніторингу полягає в забезпеченні організацій засобів оцінки стану їхньої мережевої архітектури та захищеності

інформаційної системи. Організації отримують можливість виявляти характерні закономірності, аномалії та ймовірні інциденти порушення безпеки завдяки комплексному збору та аналізу даних з різних джерел. Потенційні загрози, від яких захищає телеметричний моніторинг:

– DDoS-атаки;

– мережева розвідка;

– активність ботнетів;

– несанкціонований доступ;

– витік даних;

– атаки нульового дня.

Телеметрія в сфері інформаційних технологій охоплює багато компонентів програмного забезпечення як наприклад сервери, мережу, додатки, хмарні ресурси та користувачів. Моніторинг серверів дозволяє оцінювати використання процесора, перевантаження, продуктивність фізичної пам’яті та аналізувати активність користувачів. Надмірне навантаження на ЦП може сигналізувати про неефективну роботу програми, тоді як низьке використання вказує на проблеми із запитами чи функціональністю. Мережевий моніторинг забезпечує контроль пропускної здатності, використання додатків, безпеки мережевих портів та сховищ. Аналіз цих параметрів допомагає виявляти затримки в передачі даних, потенційні загрози та збої в системах резервного копіювання. У телеметрії додатків важливо контролювати доступ до баз даних, швидкість обробки запитів, помилки, аномалії та ключові показники ефективності, що впливають на користувацький досвід. Хмарна телеметрія включає моніторинг доступності, затримок, інтернет-маршрутизації та споживання ресурсів. Користувацька телеметрія аналізує продуктивність додатку з точки зору кінцевого користувача, дозволяючи виявити проблеми ще до їхнього впливу на роботу системи. [10]

На рис. 1.5 зображено процес збору телеметричних даних з різних компонентів для інформування IT фахівця.

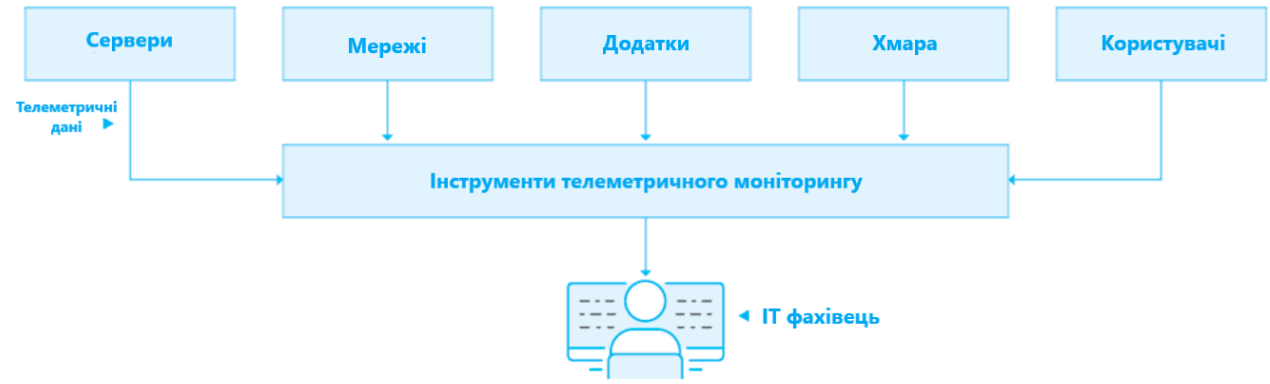


Рисунок 1.5 – Схематичне зображення збору телеметричних даних

Телеметричний моніторинг дозволяє збирати та аналізувати великі обсяги даних із різних джерел, забезпечуючи своєчасне виявлення загроз. Однак ефективність таких рішень значно підвищується завдяки штучному інтелекту.

1.4 Використання технологій штучного інтелекту для виявлення аномальних активностей

Ускладнення та ріст об’ємів даних в інформаційних системах з постійною еволюцією кіберзагроз створюють виклики для цих систем. Традиційні підходи до виявлення аномалій часто не відповідають сучасним реаліям через їхню обмежену здатність обробляти великі обсяги інформації та адаптуватися до нових видів загроз. У цьому контексті технології штучного інтелекту (ШІ) стали важливим інструментом у виявленні аномальних активностей.

Застарілі методи виявлення аномалій, такі як візуалізація та статистичні тести, довгий час використовувалися для аналізу відхилень у даних. Візуалізація дозволяє аналітикам оцінювати дані за допомогою графіків та діаграм, що може бути ефективним для невеликих наборів даних, проте цей метод стає малопридатним для великих обсягів інформації, оскільки потребує значних людських ресурсів і суб’єктивної оцінки. Статистичні тести, зокрема тест Граббса або Колмогорова-Смірнова, дають змогу порівнювати дані з очікуваними розподілами, однак вони ефективні лише у випадках, коли дані підпорядковуються певним закономірностям. [11]

У складних сценаріях, де аномалії не мають чітко визначених характеристик, ці методи втрачають точність. На відміну від них, алгоритми машинного навчання, засновані на штучному інтелекті, здатні автоматично аналізувати великі обсяги даних, визначати приховані закономірності та адаптуватися до змін у структурі даних.

Серед найпоширеніших алгоритмів виявлення аномалій за допомогою машинного навчання є [11]:

а) Ізоляційний ліс (Isolation Forest) – метод ансамблевого навчання, який ізолює аномалії, випадково обираючи ознаки та значення для розбиття, що дозволяє ефективно виявляти відхилення.

б) Однокласова машина опорних векторів (One-Class SVM) – алгоритм, який навчається лише на «нормальних» даних, створюючи межу, за якою всі інші точки вважаються аномальними.

в) k-Nearest Neighbors (k-NN) – алгоритм, що класифікує точку даних за її найближчими сусідами, де точки з малою кількістю схожих сусідів можуть вважатися аномаліями.

г) Наївний байєсівський метод – оцінює ймовірність події на основі факторів, що на неї впливають, допомагаючи виявляти залежності та взаємозв’язки між даними.

ґ) Автокодери (Autoencoders) – нейронні мережі, які аналізують історичні дані з позначкою часу для прогнозування та виявлення відхилень від звичних шаблонів.

д) Локальний фактор відхилення (LOF) – алгоритм, що вимірює щільність точок даних у порівнянні з їхніми сусідами, де точки з меншою щільністю вважаються аномальними.

е) Кластеризація методом k-середніх (k-Means Clustering) – аналізує середню відстань між точками даних та групує їх у кластери, де віддалені точки можуть сигналізувати про аномалії.

Поєднання телеметричних даних і штучного інтелекту створили новий виток розвитку сфери відстеження аномалій та загроз. Телеметрія збирає та аналізує дані

з мережевих пристроїв, серверів і кінцевих точок, створюючи повну картину активностей, в той час як алгоритми машинного навчання дозволяють автоматизувати процес виявлення та реагування на аномалії.

1.5 Огляд ELK stack та його використання для обробки телеметричних даних

Для здійснення телеметричного моніторингу та обробки відстежуваних даних необхідно запровадити відповідний інструмент. За допомогою правильно обраного програмного забезпечення для моніторингу інформаційної системи можна ефективно обробляти телеметричні дані, виявляти аномалії та уникати загроз. На ринку існує велика кількість різних подібних інструментів. ELK Stack є одним з найефективніших інструментів з-поміж усіх і в даному підрозділі буде це обґрунтовано.

Elastic Stack - це група продуктів з відкритим вихідним кодом від Elastic, призначених для того, щоб допомогти користувачам отримувати дані з будь-якого типу джерел і в будь-якому форматі. Цей набір інструментів дає можливість зібрати дані всіх логів системи і додатків, проаналізувати, візуалізувати їх, та здійснювати моніторинг інфраструктури для швидшого вирішення проблем безпеки інформаційної системи.

Раніше група продуктів була відома як ELK Stack для основних продуктів - Elasticsearch, Logstash і Kibana - але була перейменована в Elastic Stack. Згодом до стеку було додано четвертий продукт - Beats. Elastic Stack може бути розгорнутий локально або доступний як програмне забезпечення як послуга (SaaS). Elasticsearch підтримує Amazon Web Services (AWS), Google Cloud Platform і Microsoft Azure.[14]

Elasticsearch, який є ядром ELK stack – це RESTful пошуковий та аналітичний двигун, розроблений на базі бібліотеки Apache Lucene, яка здатна працювати з великими об’ємами структурованих та неструктурованих даних. Elasticsearch базується на Java та характеризується [12]:

– потужними можливостями повнотекстового пошуку;

– розвиненими аналітичними функціями (агрегація, фільтрація, групування);

– розподіленою архітектурою з горизонтальним масштабуванням;

– обробкою даних у реальному часі з мінімальними затримками;

– відкритим програмним забезпеченням із можливістю інтегрувати плагіни;

– вбудованими інструментами обробки текстових даних (токенізація, стемінг).

Logstash є компонентом для збору, перетворення та обробки даних, що інтегрується з Elasticsearch і Kibana. Він отримує дані з різних джерел (лог-файли, бази даних, потоки подій) та надає змогу гнучко конфігурувати збір та обробку лише потрібної інформації. . Його основні характеристики [12]:

– підтримка різноманітних джерел даних через систему вхідних плагінів;

– комплексна обробка даних (фільтрація, структурування, нормалізація та збагачення);

– гнучка система вихідних плагінів для інтеграції з різними сховищами (Elasticsearch, Apache Kafka, Amazon S3, тощо);

– розширені можливості обробки: парсинг регулярних виразів, геопросторовий аналіз та робота з даними JSON і CSV.

Kibana функціонує як інструмент візуалізації та аналітики даних, що зберігаються в Elasticsearch. Основні функції Kibana [12]:

– створення різноманітних типів візуалізацій (лінійні графіки, діаграми, звіти, дашборди та інші);

– конструювання комплексних інтерактивних панелей для моніторингу даних;

– фільтрування даних за різними параметрами для цільового аналізу

– доступні аналітичні інструменти для часових рядів, агрегації, фільтрації та сортування;

– геопросторова візуалізація для аналізу інформації, що містить геодані;

– рольова модель доступу для контролю авторизації.

Beats – це постачальники даних, які встановлюються на сервери як агенти, що використовуються для надсилання різних типів оперативних даних до Elasticsearch

безпосередньо або через Logstash, де ці дані можуть бути покращені або заархівовані. [14]

ELK Stack має зрозумілий та чіткий принцип роботи. Logstash виконує роль механізму для збору даних з різних джерел, який пересилає їх до Elasticsearch (або до іншого потрібного місця призначення); він працює з конфігураційним файлом, який визначає вхідні дані (наприклад, UDP з певним портом) і формат даних. Elasticsearch використовується як база даних, що індексує, аналізує та зберігає отримані дані в розподіленому форматі на основі JSON. Kibana в свою чергу підключається до Elasticsearch та займається візуалізацією отриманої інформації з бази даних.

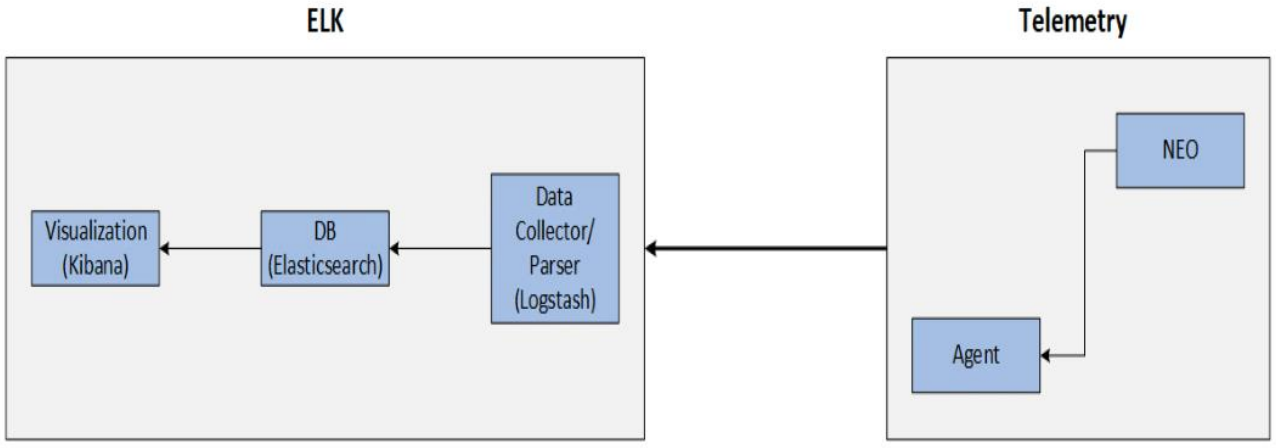


Рисунок 1.6 – Схематичне зображення роботи ELK Stack

Стек ELK використовується для вирішення широкого спектру завдань, включаючи аналітику журналів, пошук документів, впровадження SIEM рішення й моніторинг. Він забезпечує механізм пошуку та аналітики, отримання даних і їхню візуалізацію. [13]

Інтеграція ELK Stack з ШІ дозволяє збільшити ефективність використання телеметричних даних. Наприклад, дані, які збираються за допомогою ELK Stack, можуть слугувати навчальним набором для алгоритмів машинного навчання. Ці алгоритми, у свою чергу, можуть знаходити складні закономірності та взаємозв’язки, таким чином помічаючи аномальну поведінку системи.

Також можна автоматизувати процес виявлення аномалій: алгоритми ШІ, інтегровані з ELK, аналізують дані з Elasticsearch і генерують попередження в Kibana у разі виявлення підозрілих активностей.

За допомогою модулів машинного навчання Kibana може автоматично виявляти аномалії в логах та інших телеметричних даних, допомагаючи виявляти сплески трафіку, підозрілі IP-з'єднання або різкі зміни в продуктивності додатків.

ELK Stack є універсальним інструментом для обробки телеметричних даних, особливо в контексті кібербезпеки. Його здатність збирати, аналізувати та візуалізувати дані робить його незамінним. У поєднанні з технологіями штучного інтелекту ELK Stack відкриває нові можливості для створення сучасних систем моніторингу, здатних ефективно протистояти кіберзагрозам.

1.6 Аналіз існуючих засобів захисту на основі моніторингу активності користувачів за допомогою технологій ШІ

Моніторинг аномальної активності з використанням ШІ став ключовим елементом сучасного кіберзахисту. Для оцінки переваг запропонованого рішення доцільно порівняти його з такими системами, як Darktrace, Vectra AI та CrowdStrike Falcon.

Darktrace – це програма, що спеціалізується на передових рішеннях машинного навчання та штучного інтелекту, які покликані доповнити існуючий комплексний периметр кібербезпеки. Серед переваг Darktrace виділяють комплексне рішення для мережевого виявлення та реагування на основі ШІ, ефективне запобігання, стримування та карантин шкідливого трафіку в корпоративній мережі та можливість блокування шкідливих вкладень і фішингових листів. Серед недоліків даного програмного засобу виділяють високу ціну, заплутаний користувальницький інтерфейс, надмірне блокування, ідентифікація пристрою, адже просте сканування за допомогою nmap часом працює краще, відсутність комплексного відображення мережевого трафіку та неточне виявлення загроз та неповних можливостей звітування у порівнянні з інструментами з відкритим кодом. [36]

Vectra AI – це платформа для виявлення загроз та реагування на них на основі штучного інтелекту для гібридних та мультихмарних підприємств. Платформа Vectra AI забезпечує інтегрований сигнал для публічних хмар, SaaS, мереж ідентифікації та центрів обробки даних в єдиній платформі. Серед переваг Vectra AI є простий та інтуїтивно зрозумілий інтерфейс, відсутність хибних спрацьовувань та інтеграція з такими платформами, як Office 365. Серед недоліків Vectra AI не зменшив навантаження на аналітиків безпеки, оскільки він не може замінити необхідну SIEM для забезпечення відповідності, що призводить до незмінних сповіщень та реєстрації. Гнучкість архітектури потребує вдосконалення, щоб забезпечити більший контроль над датчиками Vectra, оскільки зміна режимів часто є складною.[37]

CrowdStrike Falcon – це провідна американська технологічна компанія в галузі кібербезпеки, що пропонує різні послуги, включаючи хмарні робочі навантаження, захист кінцевих точок, розвідку загроз, реагування на кіберзагрози та послуги SOC. Серед його переваг є хмарне розгортання, розширене виявлення загроз за допомогою натренованого ШІ що може виявляти тонкі закономірності та автоматизоване усунення. Серед недоліків CrowdStrike є відносно дорога ціна, хибні спрацьовування та складне налаштування. [38]

Отже, було проведено порівняння програмних засобів моніторингу з використанням ШІ за різними критеріями та оформлено у таблицю 1.1:

Таблиця 1.1 – Порівняння програмних засобів телеметричного моніторингу з використанням ШІ

Критерій	Darktrace	Vectra AI	CrowdStrike Falcon	Розроблений програмний засіб
Хмарне розгортання	+/-	+	+	+
Платна ліцензія/підписка	+	+	+	-
Часті хибнопозитивні спрацювання	+	-	+	-

Зручний інтерфейс або легкість налаштування	-	+/-	+	+
Інтеграція з сторонніми сервісами	+	+	+	+

Ґрунтуючись на порівняльному аналізі характеристик, можна прийти до висновку, що запропонований у даній роботі розроблений програмний засіб телеметричного моніторингу з використанням технологій ШІ та ELK Stack є актуальною розробкою, що покриватиме недоліки існуючих засобів.

1.7 Висновок до розділу 1 та постановка задач

У даному розділі було з'ясовано, що традиційні підходи до виявлення аномальних активностей в інформаційних системах не здатні ефективно захистити від сучасних кіберзагроз.

Аналіз концепцій аномалій показав, що вони поділяються на точкові, контекстні та колективні, кожна з яких має свої особливості та вимагає специфічних методів виявлення.

Дослідження SIEM рішень продемонструвало, що вони є ефективним інструментом централізованого моніторингу безпеки та виявлення загроз та потребує інтеграції ШІ для більш гнучких методів аналізу даних.

Проведений аналіз телеметричного моніторингу встановив, що він забезпечує динамічне та безперервне спостереження за компонентами інформаційних систем, дозволяючи отримувати оперативні дані про їхній стан та активності.

Огляд технологій штучного інтелекту показав, що їх інтеграція в системи виявлення аномалій значно підвищує ефективність та точність ідентифікації загроз, особливо в умовах великих обсягів даних та складних сценаріїв атак. Алгоритми машинного навчання здатні автоматично аналізувати телеметричні дані, визначати приховані закономірності та адаптуватися до змін у структурі даних.

Дослідження ELK stack продемонструвало, що ця технологія є потужною платформою для обробки телеметричних даних, яка поєднує механізми збору, аналізу та візуалізації інформації, що робить її придатною для розробки систем моніторингу аномальних активностей в інформаційних системах.

На основі проведеного аналізу встановлено, що для ефективного захисту інформаційних систем від сучасних кіберзагроз необхідно розробити програмний засіб телеметричного моніторингу, який інтегрує технології штучного інтелекту та ELK stack.

Для реалізації поставленої мети необхідно вирішити наступні задачі:

1. Розробити архітектуру програмного засобу, що забезпечить ефективний збір та обробку телеметричних даних інформаційної системи.

2. Визначити ключові параметри моніторингу для виявлення різних типів аномальних активностей.

3. Реалізувати інтеграцію технологій штучного інтелекту та машинного навчання для автоматичного аналізу телеметричних даних та виявлення аномалій.

4. Візуалізації результатів моніторингу, який забезпечить наочне представлення виявлених аномалій та спростить аналіз інцидентів безпеки.

5. Забезпечити рішення для обробки телеметричних даних з використанням ELK stack.

6. Реалізувати механізм оповіщення та реагування на виявлені аномальні активності, який дозволить своєчасно вживати заходів для запобігання та нейтралізації загроз.

7. Тестування програмного засобу для оцінки його ефективності в реальних умовах експлуатації.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ ТЕЛЕМЕТРИЧНОГО МОНІТОРИНГУ ТА ЗАХИСТУ ІНФОРМАЦІЙНОЇ СИСТЕМИ

У даному розділі описано процес проектування програмного засобу телеметричного моніторингу аномальних активностей, що поєднує можливості технологій штучного інтелекту та стеку ELK. Основною метою є створення логічно цілісної та функціонально орієнтованої структури, яка забезпечує ефективний оброблювальний цикл: від збору телеметричних даних до виявлення загроз і реагування на виявлені аномалії.

2.1 Розробка архітектури програмного засобу та її візуалізація за допомогою UML діаграм

Процес розробки програмного засобу починається зі створення структури цього засобу, відомої як архітектура програми, яка має містити усі основні методи та процеси для забезпечення ефективного функціонування системи. Важливість даного етапу не може бути недооцінена, оскільки він впливає на всі подальші фази реалізації програми.

Архітектура програмного засобу телеметричного моніторингу аномальних активностей є розробленою структурою програми, яка визначає взаємозв’язки компонентів та розподіл функцій між ними. Побудова архітектури забезпечує логічну організацію системи, враховуючи інтеграцію блок стеку сайту з базою даних. Проектування архітектури здійснювалося з урахуванням сучасних технологій як штучний інтелекту та ELK Stack та вимог до високопродуктивних систем телеметричного моніторингу аномальних активностей.

Розробка архітектури було почато з визначення центрального функціонального елемента системи. Телеметричний моніторинг – основний елемент навколо якого вибудовуються всі інші методи та процеси. Також архітектура має включати в себе такі процеси як збір, обробка, зберігання, аналіз, реагування та візуалізація даних. Для кожного з них виділено окремі рівні, які надалі будуть взаємодіяти через інтерфейси.

Архітектура програмного засобу передбачає побудову багаторівневої системи з чіткою ієрархією. Кожен метод знаходиться на певному рівні, виконує специфічну функцію та має власні механізми відмовостійкості. Отже, архітектура програмного засобу включатиме такі рівні:

- Рівень збору даних, що відповідає за отримання телеметричних даних із сайту з базою даних. Він взаємодіє із зовнішніми системами за допомогою API, які передають запити до бази даних та мережеву активність сайту до системи обробки та зберігання даних.
- Рівень обробки даних бере отримані на попередньому рівні дані, трансформує, фільтрує та передає їх у систему зберігання. Цей рівень включає Logstash, який відповідає за підготовку даних для подальшого аналізу.
- Рівень зберігання підтримує оброблені телеметричні дані в Elasticsearch, де вони індексуються для швидкого пошуку та доступу до них.
- Аналітичний рівень, використовуючи алгоритми машинного навчання, здійснює аналіз даних для виявлення аномалій. Цей рівень інтегрує ШІ модулі, які обробляють дані в реальному часі або виконують ретроспективний аналіз.
- Рівень реагування та прийняття рішень автоматично або з участю оператора здійснює реакцію на помічені аномалії. Це може здійснюватися як за рахунок генерації сповіщень і запуску сценаріїв реагування, так і блокуванням підозрілих дій і обмеженням доступу. Рішення приймаються на основі попередньо заданих політик безпеки або за допомогою інтелектуальних моделей оцінки ризику.
- Рівень візуалізації використовує Kibana для представлення даних у вигляді графіків та діаграм для легкого відстеження стану системи та оперативного реагування на загрози.

Архітектура передбачає багаторівневий захист інформації за рахунок рольової моделі доступу й керування, а також багатофакторної автентифікації.

Також для подальшого планування та полегшення проектування програмного засобу, зобразимо схему руху даних (рис.2.1), що включає послідовність таких етапів:

1. Збір первинних метрик з сховища даних
2. Попередня фільтрація даних
3. Потокова обробка даних в розподілених чергах
4. Аналіз даних на аномалії технологіями ШІ
5. Формування попереджень та реакцій
6. Перенаправлення даних відповідно до налаштованого блекліста

Рисунок 2.1 – Схема руху даних програмним засобом

Під час створення архітектури програмного засобу було використано мікросервісний підхід, що дозволяє розбити ПЗ на невеликі незалежні сервіси, які були представлені як рівні для забезпечення гнучкості системи. Кожному методу на рівні надано певний функціонал та рівні взаємодії з іншими методами через легковагові протоколи [15]

Для опису архітектури ПЗ буде використано UML (Unified Modeling Language) діаграми, що активно використовуються для проектування інформаційних систем. UML це методика схематичного проектування за допомогою діаграм або позначень для специфікації, візуалізації та документації моделі об'єктно-орієнтованих програмних систем. UML управляється Object Management Group (OMG) і є промисловим стандартом, що описує моделі ПЗ. [16]

Існує 14 видів UML діаграм. Розглянемо три найпопулярніші діаграми: діаграму прецедентів (Use-case diagram), діаграму послідовності (Sequence diagram); та діаграму класів (Class diagram).

Діаграма прецедентів – це графічне зображення взаємодії акторів (користувачів або інших систем) із програмною системою через прецеденти (сценарії використання).Вона показує хто взаємодіє із системою (актори), що саме робить система для актора (прецеденти) та як вони пов’язані між собою. Діаграма допомагає зрозуміти, яку поведінку очікують від системи користувачі та інші зовнішні сутності.[16]

На рисунку 2.2 зображено діаграму прецедентів для системи моніторингу:

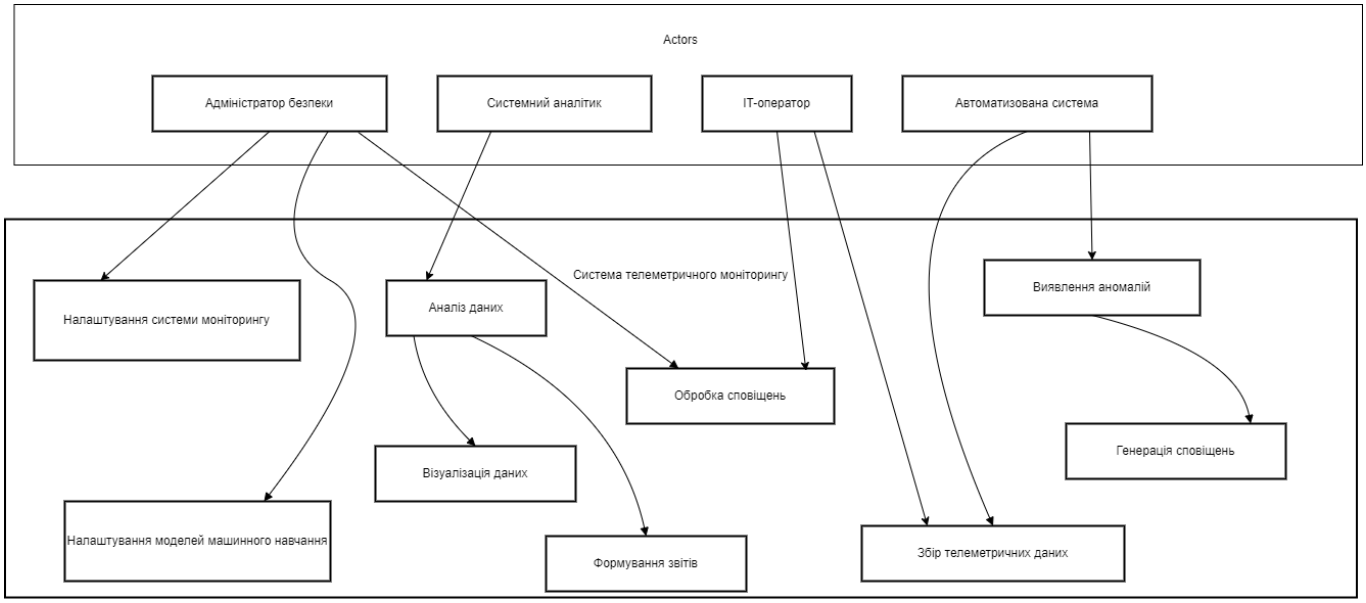


Рисунок 2.2 – Діаграма прецедентів

Діаграма прецедентів як видно з рисунку 2.2 описує взаємодію користувача з системою з погляду зовнішньої поведінки, але не показує, як реалізована система зсередини. Вона не дає уявлення про структуру модулів, компоненти ШІ чи потоки телеметричних даних. Також планується автоматизувати більшість процесів і не залучати акторів.

Діаграма послідовності – це графічне зображення взаємодії об’єктів у часі, яка показує, в якій послідовності вони обмінюються повідомленнями для виконання певного сценарію або завдання. Вона відображає об’єкти, які беруть участь у взаємодії, повідомлення, які передаються між ними та порядок і час цих

взаємодій. На діаграмі горизонтально розміщені лінії життя об’єктів, а вертикально – вісь часу, зверху вниз. Ключова увага на динаміці, а не на зв’язках між об’єктами. [17]

На рисунку 2.3 зображено діаграму послідовності для системи моніторингу:

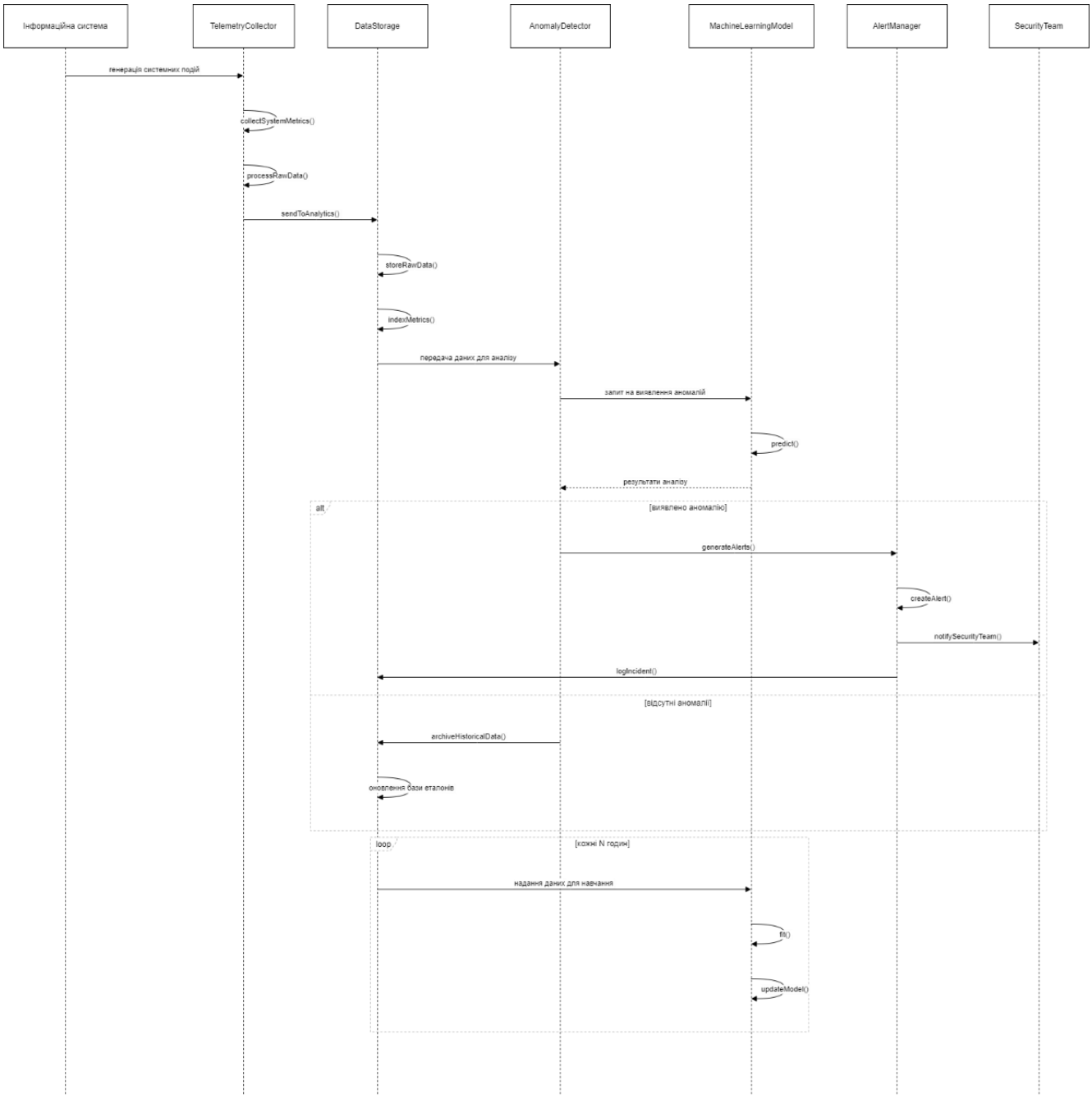


Рисунок 2.3 – Діаграма послідовності

Проаналізувавши діаграма послідовності з рисунка 2.3 стає зрозуміло, що ця діаграма корисна для опису сценаріїв взаємодії об'єктів у часі, але вона не відображає, які саме об’єкти існують у системі. Вона не підходить для візуалізації логічної структури системи і для планування її програмної реалізації

Діаграма класів – це статичне графічне представлення структури програми, яке показує класи, їх поля, методи та зв’язки між класами. Вона допомагає формалізувати логічну модель системи та зрозуміти, з яких елементів складається програма. У діаграмі класів описують стан (через поля) і поведінку (через методи) кожного класу. На діаграмі клас зображується прямокутником, поділеним на три частини: назва, поля, методи. Також вона використовує принципами ООП згідно з якими класи інкапсулюють дані та сервіси. [18]

На рисунку 2.4 зображено діаграму класів для системи моніторингу:

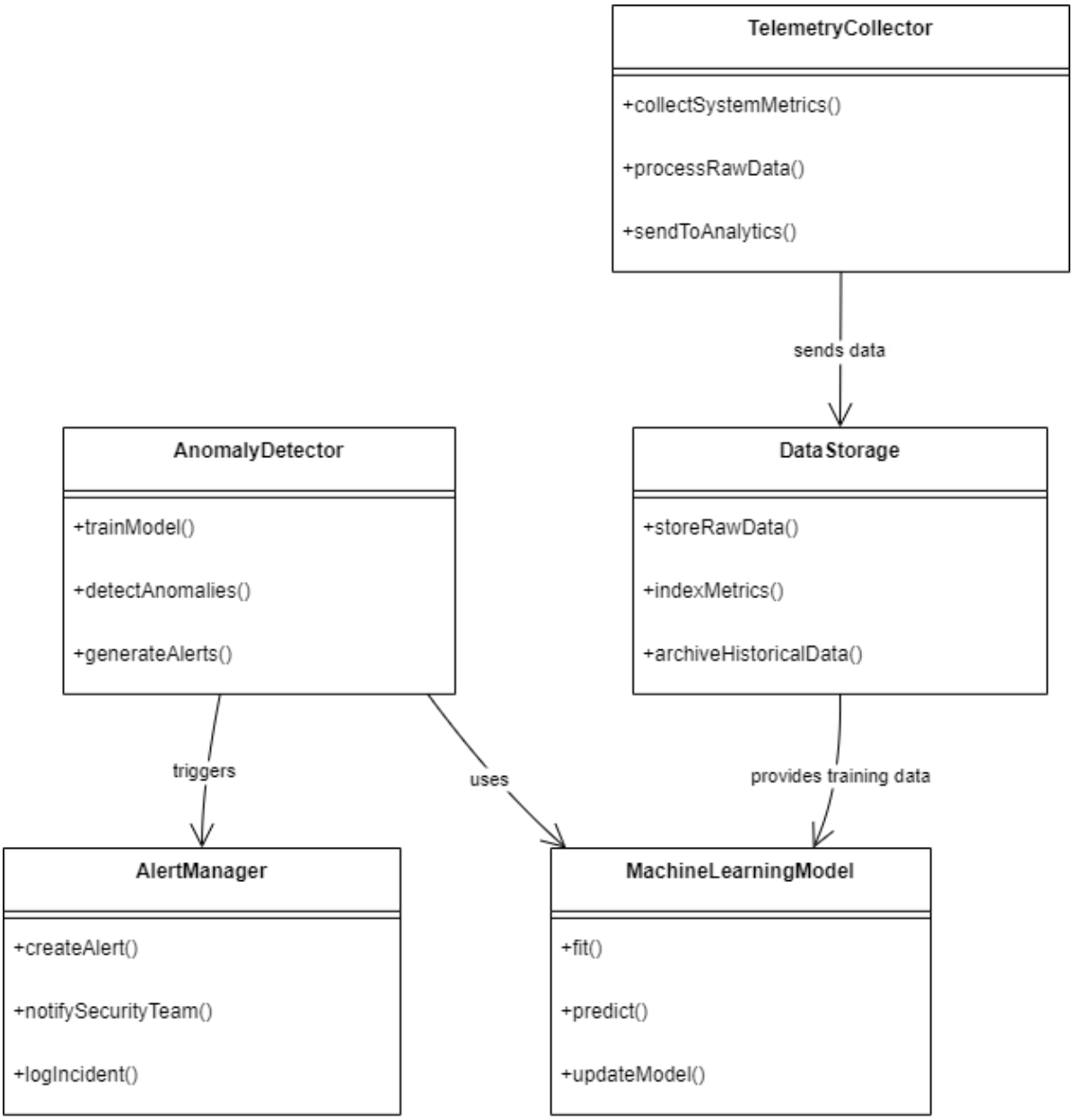


Рисунок 2.4 – Діаграма класів

Представлена UML діаграма класів демонструє основні компоненти системи та їх взаємозв'язки. На діаграмі показано п'ять ключових класів: TelemetryCollector, AnomalyDetector, DataStorage, AlertManager та MachineLearningModel. Ці класи забезпечують повний цикл телеметричного моніторингу.

Кожен клас має власний набір методів, що відповідають за специфічні функції: збір даних, виявлення аномалій, збереження даних, створення попереджень та машинне навчання.

Стрілки на діаграмі демонструють взаємодію між компонентами: як вони обмінюються даними, використовують спільні ресурси та впливають один на одного в процесі роботи системи.

Розроблена архітектура є гнучкою та масштабованою. Вона забезпечує ефективне збирання, обробку та аналіз телеметричних даних. Використання UML діаграми класів допомогло зобразити кожен аспект системи, створивши чітке уявлення про її компоненти та взаємозв’язки. Ця модель слугуватиме основою для реалізації програмного забезпечення.

2.2 Розробка блок-схем системи захисту на основі телеметричного моніторингу

Для опису архітектури програмного забезпечення у даній роботі буде використано блок-схеми, які є наочним інструментом для представлення логіки роботи системи, взаємозв’язків між компонентами та послідовності обробки даних. Блок-схеми дозволяють зрозуміло і структуровано зобразити основні етапи функціонування системи, включаючи процеси збору телеметрії, обробки за допомогою ШІ, виявлення аномалій та реагування. Завдяки своїй простоті та універсальності, блок-схема є зручним засобом візуального проєктування, що особливо ефективно використовується в технічній документації для складних інформаційних систем.

Представлена на рисунку 2.5 блок-схема ілюструє послідовність етапів обробки телеметричних даних у системі моніторингу. Вона охоплює підключення

до Elasticsearch, побудову та виконання запиту, класифікацію результатів за допомогою ШІ та завершення сеансу шляхом закриття клієнта.

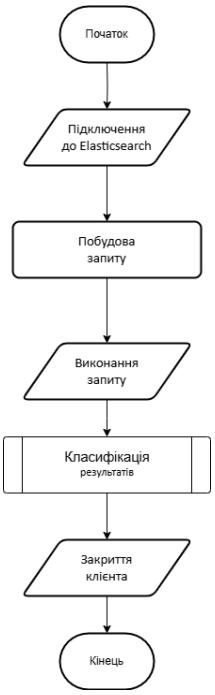


Рисунок 2.5 – Блок-схема системи захисту

Блок-схема на рисунку 2.6 демонструє логіку фільтрації IP-адрес. Процес починається з ініціалізації фільтру, після чого з запиту отримується IP-адреса. Якщо вона входить до списку заблокованих (BLOCKED_IPS), відбувається її блокування. У протилежному випадку запит проходить далі, і фільтр знищується.

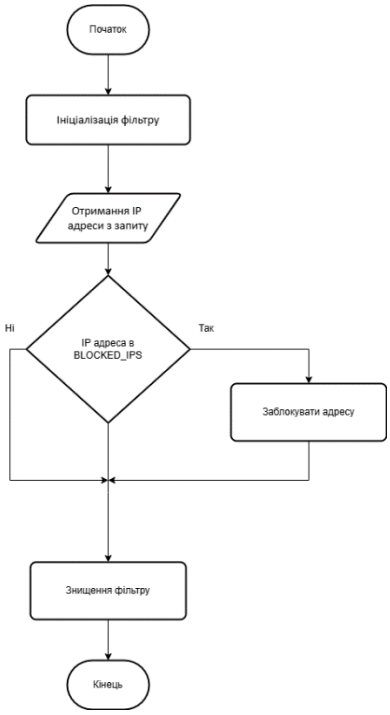


Рисунок 2.6 – Блок-схема модуля фільтрації IP-адрес

2.3 Розробка методу збору та обробки даних

На початковому етапі проектування необхідно було розробити метод збору телеметричних даних, що передбачав інтеграцію з базою даних певного сайту, зокрема з модулями збору мережевої статистики та колекторами системних журналів. Для забезпечення уніфікованого підходу до збору було розроблено абстрактний клас TelemetryCollector з відповідними методами, зокрема collectSystemMetrics().

Наступним кроком було впровадження механізмів попередньої обробки телеметричної інформації. Було реалізовано валідацію структури вхідних даних, їх нормалізацію до уніфікованого формату та очищення від некоректних записів. До даних було автоматично додано мітки часу. Дані класифікувалися за категоріями для подальшого аналізу. Усі операції обробки реалізовано з використанням шаблону проектування «Ланцюжок відповідальності», що дозволило модульно реалізувати метод processRawData() та забезпечити його прозорість через логування проміжних етапів обробки.

Як сховище телеметричних даних використовувалась система Elasticsearch з реалізованою індексацією. Для збереження даних були створені окремі методи: storeRawData() для збереження необробленої інформації, indexMetrics() для індексації вже оброблених даних, а також archiveHistoricalData() — для архівації даних.

На завершальному етапі було реалізовано інтерфейс передачі даних до аналітичних модулів. Компонент TelemetryCollector забезпечує надсилання інформації до модуля AnomalyDetector. Повідомлення передаються у відповідні черги з налаштованим часом життя (TTL) та підтвердженням доставки, що гарантує надійність передачі даних.

2.4 Розробка методу моніторингу, аналізу поведінки та виявлення аномалій

Розробки методу моніторингу, аналізу поведінки та виявлення аномалій базувалася на побудові логічної взаємодії між основними модулями системи на основі створеної UML-діаграми класів.

Модуль TelemetryCollector відповідає за передачу телеметричних даних до компонента DataStorage. Цей компонент забезпечує підготовку цих даних для навчання моделі машинного навчання. Для цього проаналізовано телеметричні дані, отримані з компонента DataStorage, для виділення основних параметрів, що характеризують стабільну роботу інформаційної системи. Для кожної метрики були встановлені допустимі діапазони значень на основі середнього значення.

Далі аналітичний модуль AnomalyDetector взаємодіє з навченою моделлю MachineLearningModel, використовуючи її для виявлення аномалій, створення відповідних сповіщень та ініціювання реагування через компонент AlertManager.

Наступним кроком є вибір та тренування алгоритмів машинного навчання. Для цього реалізовано клас MachineLearningModel, у якому метод fit() відповідає за навчання на телеметричних даних, отриманих із модуля зберігання, а також метод predict() для виявлення потенційних аномалій. При розробці системи передбачалося врахування типу моделі (такої як Isolation Forest, Autoencoder або LSTM) та категорій даних, що аналізуються, зокрема використання CPU, пам’яті, мережевого трафіку тощо. Важливою умовою стало регулярне оновлення моделі через функціонал updateModel() на основі нових, історично накопичених даних, що дозволяє оновлювати параметрів моделі у разі зміни патернів поведінки.

Останнім етапом стала розробка контекстної підсистеми аналізу, яка розширила метод detectAnomalies() для факторів навколишнього середовища. Було впроваджено механізм врахування часу доби, навантаження на систему, типів активності користувачів та запланованих дій у системі. Контекстуалізація вхідних подій дозволила суттєво знизити кількість хибнопозитивних спрацювань. Також запроваджено типізацію сценаріїв аномальної поведінки, що дозволило порівнювати поточні ситуації з агрегованими шаблонами поведінки, збереженими в модулі DataStorage.

2.5 Розробка методу реагування та прийняття рішень

Першочергово була виконана категоризація типів аномалій за рівнем критичності. Така класифікація охоплює відхилення, що не потребують негайного втручання, підозрілі активності середнього ризику, критичні інциденти з високим пріоритетом реагування, а також надзвичайні ситуації, які вимагають миттєвих дій. Для кожного типу аномалій визначено відповідні реакції. Це дозволило сформувати узгоджену матрицю «аномалія–реакція», яка слугує основою для прийняття рішень.

Центральним елементом архітектури реагування є компонент AlertManager, функціональність якого охоплює створення структурованих сповіщень про інциденти, їх фіксацію в системному журналі та передачу відповідальним особам. Повідомлення формуються з урахуванням рівня критичності, типу аномалії, рівня впевненості моделі в коректності класифікації та часу виявлення. Усі події додатково документуються з метою подальшого аналізу та навчання моделей.

Компонент взаємодіє з системою машинного навчання через обмін інформацією щодо точності виявлення та характеру загроз. У процесі реагування використовується функція predict() для уточнення ймовірності помилкового спрацювання, а також адаптивно коригуються пороги чутливості виявлення. Зворотний зв’язок від фахівців із безпеки інтегрується в модель через відповідний механізм оновлення, що дозволяє системі самонавчатися на основі підтверджених або відхилених сповіщень.

Для забезпечення довгострокової ефективності розробленої системи було створено документацію, яка охоплює алгоритми реагування, інструкції для персоналу та регламенти технічного обслуговування..

У результаті реалізований метод забезпечує повний цикл реагування на аномалії: від виявлення інциденту модулем AnomalyDetector, формування структурованого сповіщення засобом AlertManager, до виконання автоматичних або ручних дій реагування з подальшим самонавчанням системи на основі накопиченого досвіду.

2.6 Опис алгоритмів машинного навчання для збору та обробки телеметричних даних та виявлення аномалій

Використання алгоритмів машинного навчання (ML) у виявленні аномалій є ключовим аспектом сучасних систем кібербезпеки. Завдяки здатності ML-алгоритмів вивчати великі обсяги даних, виявляти приховані залежності та адаптуватися до нових сценаріїв, вони стали невід'ємним елементом телеметричного моніторингу. У цьому розділі розглянуто основні підходи до машинного навчання, які застосовуються для збору та обробки телеметричних даних, а також алгоритми, що забезпечують виявлення аномалій із високою точністю.

Залежно від специфіки даних і задач, машинне навчання використовує три основні підходи:

1. Навчання з учителем (Supervised Learning). Цей підхід передбачає наявність навчального набору даних, який включає як нормальні випадки, так і аномальні. Моделі, такі як логістична регресія, дерева рішень чи нейронні мережі, навчаються класифікувати нові дані, ґрунтуючись на маркованих прикладах.
2. Навчання без учителя (Unsupervised Learning). У цьому підході алгоритми працюють із немаркованими даними, шукаючи відхилення від нормальних патернів. Методи, як-от кластеризація (k-means, DBSCAN) або метод головних компонент (PCA), часто використовуються для ідентифікації потенційно аномальних активностей.
3. Напівконтрольоване навчання (Semi-Supervised Learning). Цей підхід поєднує елементи навчання з учителем та без учителя, працюючи з обмеженою кількістю маркованих даних і великою кількістю немаркованих. Наприклад, алгоритми Deep Semi-Supervised Learning використовуються для розширення знань моделі на основі невеликих маркованих наборів.

Кожен із цих підходів має свої переваги й обмеження залежно від доступних даних і характеру телеметричних аномалій.

Метод опорних векторів широко використовується для класифікації даних і виявлення аномалій завдяки своїй здатності знаходити оптимальні гіперплощини для розділення класів. У контексті аномалій SVM може використовуватися в однокласовому режимі (One-Class SVM), де модель створює межу, яка відокремлює нормальні випадки від потенційно аномальних.

One-Class SVM є ефективним у роботі з високовимірними даними, що робить його корисним для аналізу складних телеметричних даних. Однак його продуктивність залежить від правильного вибору параметрів, таких як ядро функції (kernel) і значення регуляризації.

Кластеризація є основним методом навчання без учителя, який використовується для групування подібних даних. У контексті телеметрії цей підхід дозволяє ідентифікувати групи нормальної поведінки та виділяти точки, які не вписуються в ці групи, як потенційні аномалії.

До популярних алгоритмів кластеризації належать:

- K-means: дозволяє розділити дані на k кластерів, порівнюючи відстані між точками та центроїдами кластерів. Цей метод ефективний для великих наборів даних, але чутливий до вибору значення k.
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise): працює на основі щільності даних, автоматично виділяючи аномальні точки, які не входять до жодного кластеру. DBSCAN є менш чутливим до вибору початкових параметрів, але складніше працює з високовимірними даними.

Нейронні мережі є потужним інструментом для виявлення аномалій завдяки їхній здатності обробляти великі обсяги даних і виявляти складні залежності між параметрами. У телеметрії часто застосовуються:

- Автокодери (Autoencoders): це архітектура нейронної мережі, яка навчається стискати дані у внутрішнє представлення з подальшим їхнім відновленням. Аномальні точки, які важко відновити, вважаються відхиленнями.
- Рекурентні нейронні мережі (RNN): корисні для аналізу послідовних даних, наприклад, телеметричних даних, які змінюються з часом. RNN дозволяють

прогнозувати наступний крок у послідовності, а значні відхилення від передбаченого значення розглядаються як аномалії.

Метод ізоляційного лісу (Isolation Forest) є інноваційним підходом до виявлення аномалій. Isolation Forest базується на ідеї, що аномальні точки в даних легше ізолювати, ніж нормальні, через їхню віддаленість або унікальність. Алгоритм створює кілька дерев, у кожному з яких випадкові точки розділяються, доки не будуть ізольовані. Чим менше глибина дерева, на якій ізолюється точка, тим вищою є ймовірність, що ця точка є аномальною.

PCA дозволяє зменшити кількість вимірів у даних, зберігаючи основні характеристики. У контексті виявлення аномалій метод аналізує відхилення у проекціях даних на нові виміри. Значні відхилення можуть свідчити про наявність аномалій. PCA часто використовується як попередній етап обробки даних перед застосуванням інших методів.

Збір телеметрії є основою для роботи алгоритмів машинного навчання. Дані мають бути повними, якісними та відповідати структурі, необхідній для аналізу. Процес обробки включає:

1. Попередню обробку даних: очищення, нормалізація та перетворення вхідних даних для усунення шуму та розбіжностей.
2. Агрегацію та трансформацію: об’єднання даних із різних джерел (серверів, мережевих пристроїв, системних журналів) і приведення їх до єдиного формату.
3. Формування ознак: виділення релевантних характеристик із даних, які мають найбільший вплив на точність алгоритмів.

Алгоритми машинного навчання, інтегровані з процесами збору та обробки телеметричних даних, забезпечують високу ефективність у виявленні аномалій. Вибір конкретного алгоритму залежить від характеру даних, доступності маркованих прикладів та вимог до точності і швидкості аналізу. Завдяки цим алгоритмам телеметрія перетворюється на потужний інструмент для проактивного захисту інформаційних систем.

2.7 Інтеграція технологій штучного інтелекту з компонентами ELK Stack

ELK Stack сам по собі забезпечує ефективний збір, обробку та візуалізацію великих обсягів даних, але для реалізації аналітичного функціоналу такого як виявлення прихованих закономірностей, необхідна інтеграція з технологіями машинного навчання.

Поєднання компонентів ELK Stack з алгоритмами штучного інтелекту формує основу для побудови сучасної системи моніторингу, орієнтованої на виявлення аномалій та автоматизоване реагування на загрози в телеметричних даних.

Штучний інтелект обробляє дані, збережені в Elasticsearch, через API. Моделі (наприклад, автокодери, LSTM, Isolation Forest) навчаються на історичних логах для виявлення нетипових патернів. Після навчання вони можуть працювати з потоком нових даних у реальному часі, генеруючи сигнали про аномалії. Результати аналізу можуть записуватись у нові індекси Elasticsearch, наприклад, окремий індекс для «аномалій», що виводиться на дашбордах Kibana.

Якість телеметрії напряму впливає на точність моделей. Logstash видаляє дублі, шумові дані, перетворює формати, класифікує потоки. Це знижує похибку алгоритмів і підвищує ефективність виявлення відхилень.

Elasticsearch накопичує історичні дані, які використовуються для формування навчальних вибірок. Інтеграція з фреймворками типу TensorFlow, scikit-learn або PyTorch відбувається через API або плагіни. Навчання включає:

- нормалізацію;
- вибір архітектури моделі;
- тренування;
- валідацію;
- оновлення моделей на основі зворотного зв’язку.

Системи підтримують адаптивні моделі з механізмами самонавчання.

Модулі ШІ можуть працювати як мікросервіси, що отримують дані через API або потоки Logstash. Це забезпечує незалежну масштабованість. Взаємодія здійснюється через:

- REST API Elasticsearch;
- логічні конвеєри Logstash;
- плагіни Kibana для імпорту результатів ML;
- сервіси обробки у Python/Java.

Наприклад, після кластеризації модель передає ідентифікатори підозрілих подій назад до Elasticsearch, а Kibana автоматично формує попередження.

Інтегровані моделі можуть оцінювати рівень загрози й ініціювати тригери від створення алерта до запуску скриптів реагування. Це дозволяє автоматизувати рутинні дії операторів SOC та зменшує час реагування.

Машинне навчання, особливо для потокових даних, потребує обчислювальних ресурсів. Рішення: використання розподілених обчислень. Elasticsearch вже має кластерну архітектуру, яку можна поєднати з розгортанням ML у хмарі або на edge-серверах.

Кожен компонент ELK Stack може бути інтегрований окремо з ML-модулями, що дозволяє адаптувати систему до конкретних сценаріїв: кіберзахисту, моніторингу IT-інфраструктури, IoT або DevOps.

Інтеграція штучного інтелекту в ELK Stack створює основу для побудови інтелектуальних систем моніторингу з підтримкою виявлення, прогнозування та реагування на інциденти. Вона дозволяє об’єднати гнучкість обробки логів з аналітичними можливостями ML, зменшуючи людський фактор та підвищуючи рівень кібербезпеки.

У розробці програмного засобу буде інтегровано вбудовані функції машинного навчання в ELK Stack для виявлення аномалій, оскільки вони мають значну перевагу, бо вбудовані в екосистему Elastic. Це усуває потребу в зовнішніх інструментах або складних конвеєрах даних. Можливості машинного навчання Elastic базуються на перевірених методах, включаючи кластеризацію, різні типи декомпозиції часових рядів, моделювання байєсівського розподілу та кореляційний

аналіз. Ці методи дозволяють системі ефективно моделювати нормальну поведінку та виявляти ледь помітні аномалії. Використання цієї вбудованої інтеграції спрощує розгортання, обслуговування та забезпечує сумісність між середовищами Elasticsearch та Kibana.[19]

2.8 Висновки до розділу 2

У результаті виконаного проектування було сформовано багаторівневу архітектуру програмного засобу телеметричного моніторингу, що враховує вимоги до обробки великих обсягів даних. Кожен рівень системи реалізує окремі функції — від збору, обробки й зберігання до аналізу, реагування та візуалізації. Обґрунтовано доцільність використання UML діаграми класів як найкращого способу представлення логічної структури компонентів. Запропонована архітектура є важливою основою для подальшої розробки системи виявлення та реагування на аномальні активності.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

У даному розділі описано програмну реалізацію розробленого засобу телеметричного моніторингу аномальних активностей з інетеграцією технологій ШІ та стеку ELK. Також у цьому розділі обґрунтовано вибір середовища розгортання, продемонстровано скрипти програмного засобу та графічний інтерфейс системи, та результати тестування системи на її можливості реагування на виявлені аномалії.

3.1 Вибір середовища розгортання та застосування студентських можливостей з Microsoft Azure та AKS

У цьому підрозділі розглянуто можливі варіанти розміщення програмного засобу, зокрема на VPS/VDS та у хмарному середовищі. Буде проаналізовано ці три популярні способи хостингу та з'ясовано, який з них є кращим у різних сценаріях. Розгляд охоплює технічні та практичні аспекти впровадження, враховуючи доступні ресурси, зокрема студентські програми підтримки для хмарних сервісів.

Віртуальний приватний сервер (VPS) і віртуальний виділений сервер (VDS) – передбачають надання клієнту віртуальних серверів із заданими параметрами. Іноді вони відрізняються за технологією віртуалізації. VDS зазвичай базується на апаратній віртуалізації, тоді як VPS означає віртуалізацію на рівні операційної системи.

Віртуальний виділений сервер (VDS) - це тип хостингу, який надає виділене середовище віртуального сервера. Це ізольоване серверне середовище з виділеними ресурсами, такими як процесор, оперативна пам'ять і сховище. Завдяки VDS є повний root-доступ до свого сервера. Існує можливість встановлювати програмне забезпечення або налаштовувати серверне середовище за власним бажанням.[20]

Віртуальний приватний сервер (VPS), як і VDS, надає віртуальне серверне середовище. Однак є кілька ключових відмінностей. З VPS кілька веб-сайтів також

використовують один фізичний сервер, і хоча ресурси все ще є спільними, кожен сайт отримує свої власні ресурси. Це забезпечує кращу продуктивність. Наприклад, якщо один сайт зазнає раптового сплеску трафіку, це не вплине на інші сайти на тому ж сервері. Крім того, VPS забезпечує підвищену безпеку завдяки ізоляції. Це означає, що якщо один сайт буде зламано, це не вплине на безпеку інших сайтів на тому ж сервері.[20]

Користувач VPS/VDS отримує віртуальний сервер, виділений під приватне використання. VPS створюється за допомогою спеціального програмного забезпечення, гіпервізора, який встановлюється на фізичний сервер та поділяє ресурси. Користувачу виділяється частина ресурсів одного фізичного сервера. Коли йому потрібно більше/менше ресурсів, – замовляється новий VPS, за іншим тарифом. [21]

Хмарний сервер — це віртуальний сервер, який є віддаленим і використовує обчислювальні потужності та місце фізичних серверів, зазвичай, розташованих у центрах обробки даних/ЦОД, тобто доступ до нього здійснюється через Інтернет. Ви орендуєте “частину” фізичного сервера, а не володієте ним. [22]

Головною перевагою хмарного сервера є масштабованість - ви можете збільшити кількість ресурсів за потреби в кілька кліків, не турбуючись про інвестиції в додаткове обладнання. Це дозволяє легко адаптуватися до мінливих вимог бізнесу. Крім того, хмарні сервери розміщуються в декількох географічно розподілених дата-центрах, а це означає, що в разі виходу з ладу одного сервера ресурси можуть бути швидко і безперешкодно перенесені до іншого дата-центру. Клієнти отримують доступ до ресурсів з будь-якого місця, що дозволяє їм працювати віддалено або співпрацювати зі співробітниками з різних місць.[20]

Хмарне сховище – це фактично віртуальний комп’ютер. Користувачі мають доступ до всієї інформації, з якою вони працюють через хмарний сервер. Головне – наявність підключення до Інтернету. Завдяки такій можливості хмара – прекрасний вибір для людей, які часто використовуватимуть її з дому або в дорозі, без потреби застосовувати мережу VPN. Хмарні сервери також чудово підходять для виконання програм, які завжди мають бути оновлені та доступні. Крім цього, у

хмарі можна легко обмінюватися файлами, співпрацювати в реальному часі та краще керувати версіями. І насамкінець, деякі постачальники хмарного сховища пропонують диференційну синхронізацію, яка значно скорочує час передавання та зменшує пропускну здатність, що використовується під час внесення змін до великих файлів. [23]

Отже, було проведено порівняння різних середовищ розгортання програмного засобу за різними критеріями та оформлено у таблиці 3.1:

Таблиця 3.1 – Порівняння VPS та хмарного хостингу

Особливості	VPS хостинг	Хмарний хостинг
Модель веб-хостингу	Приватний сервер	Виділений хмарний сервер
Оплата за використання	Так	Так
Низька вартість	Так	Так
Безпека	Нижча	Вища
Налаштування	Вище	Нижче
Гнучкість	Нижча	Вища
Масштабованість	Нижча	Вища
Кілька центрів обробки даних	Залежить від постачальника	Так

Порівняння VPS і хмарного хостингу демонструє перевагу хмарних рішень у контексті безпеки, масштабованості та гнучкості. Хмарні сервіси забезпечують вищий рівень захисту даних, можливість автоматичного розширення ресурсів без простоїв та доступ до інфраструктури з кількома дата-центрами. Попри дещо обмежені можливості налаштування порівняно з VPS, хмарний хостинг пропонує простіше управління та ширші можливості для розвитку. З огляду на ці переваги, хмарне середовище є більш ефективним варіантом для розгортання програмного засобу в рамках дипломного проєкту.

Завдяки співпраці з Центром інформаційних технологій і захисту інформації (ЦІТЗІ), студенти мають доступ до хмарного пакету Office 365. З 2012 року ЦІТЗІ надає можливість безкоштовного використання Office 365 for Education — платформи, створеної спеціально для освітніх потреб, яка включає сучасні хмарні інструменти та підтримується на всіх популярних пристроях. Цей факт суттєво

зменшує фінансові бар’єри для студентів, які хочуть використовувати професійні хмарні сервіси. Таким чином, недолік хмарного розгортання у вигляді високої вартості втрачає актуальність. У зв’язку з цим було обравно середовище Microsoft Azure, про що докладніше йтиметься далі. [24]

Microsoft Azure - це платформа хмарних обчислень, розроблена корпорацією Майкрософт, що пропонує широкий спектр послуг, включаючи обчислення, аналітику, зберігання та мережу. Вона дозволяє користувачам створювати, керувати та розгортати додатки в глобальній мережі центрів обробки даних. Azure підтримує різні мови програмування та фреймворки, забезпечуючи гнучкість і масштабованість для підприємств і розробників по всьому світу. Завдяки вбудованим функціям безпеки та сертифікатам відповідності, Azure гарантує безпечні та надійні хмарні рішення. [25]

На хмарній платформі Microsoft Azure можна розміщувати наявні програми та спрощувати створення нових. Вона містить сервіси, які потрібні, щоб розробляти, тестувати та розгортати додатки. [26]

Microsoft Azure надає широкий спектр сервісів — понад 200 рішень для підтримки штучного інтелекту, обробки даних, роботи з контейнерами, IoT тощо. Залежно від потреб, користувач може обрати як класичні віртуальні машини (IaaS), так і більш керовані середовища на базі моделей PaaS або FaaS.

Основні сервіси в Azure: [27]

– Virtual Machines (Віртуальні машини) - це служба IaaS, що дозволяє розгортати віртуальні машини і управляти ними всередині віртуальної мережі (VNet).

– App Service (Служба додатків) - це PaaS для розміщення веб-додатків, серверних частин мобільних додатків, RESTful API або автоматизованих бізнес-процесів.

– Service Fabric - це платформа розподілених систем, яка може працювати в багатьох середовищах, включаючи Azure або локально.

– Azure Kubernetes - управляє розміщеної службою Kubernetes для запуску контейнерних додатків.

– Azure Container Instances пропонує найшвидший і простий спосіб запустити контейнер в Azure без необхідності розгортати будь-які віртуальні машини і використовувати службу вищого рівня.

– Azure Functions - це керована служба FaaS (функція як послуга).

– Cloud serverless - це модель динамічного розподілу ресурсів сервера між споживачами з динамічним ціноутворенням їх використання, на відміну від традиційної моделі оплати певної фіксованої ємності.

– Azure Batch - це керована служба для запуску великомасштабних додатків паралельних і високопродуктивних обчислень (HPC).

– Cloud Services - це керована служба для запуску хмарних додатків. Використовує модель хостингу PaaS.

Для проєктів, пов’язаних із телеметрією, обробкою великих обсягів даних і компонентною архітектурою, особливо актуальними є сервіси для роботи з контейнерами. У цьому контексті доцільно перейти до розгляду служби Azure Kubernetes Service, яка забезпечує ефективне розгортання, масштабування й управління контейнеризованими компонентами системи.

Azure Kubernetes Service (AKS) - це керована служба Kubernetes, яку можна використовувати для розгортання та керування контейнерними програмами. AKS зменшує складність і операційні накладні витрати на керування Kubernetes, перекладаючи більшу частину відповідальності на Azure. AKS - ідеальна платформа для розгортання й керування контейнерними програмами, які потребують високої доступності, масштабованості та портативності. [28]

AKS полегшує розгортання керованих кластерів Kubernetes в Azure. AKS розвантажує розробників та адміністраторів від критично важливих завдань, таких як моніторинг стану та обслуговування. Для створення кластерів AKS можна використовувати Azure CLI або Azure Portal. Інший варіант - використовувати рішення «інфраструктура як код» (IaC), такі як TerraForm. [29]

Цей фрагмент конфігурації Kubernetes демонструє запуск базового модуля системи телеметричного моніторингу в кластері Azure Kubernetes Service:

```
apiVersion: v1
```

```
kind: Pod
metadata:
  name: telemetry-monitor
  labels:
    app: telemetry
spec:
  containers:
  - name: monitor
    image: telemetry-monitor:latest
    ports:
    - containerPort: 8080
```

Така структура дозволяє легко масштабувати систему та забезпечити її стабільну роботу в хмарному середовищі

3.2 Програмна реалізація системи захисту на основі телеметричного моніторингу

Для реалізації захисту, потрібно реалізувати систему моніторингу, яка буде включати в себе отримання даних телеметрії (користувацької активності), класифікація активності, блокування абонента при виявленні аномалії.

Для отримання телеметрії потрібно виконати запит до Elasticsearch та передати дані на класифікацію.

```
import org.elasticsearch.action.search.SearchRequest;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.index.query.BoolQueryBuilder;
import org.elasticsearch.index.query.RangeQueryBuilder;
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.search.builder.SearchSourceBuilder;
import org.elasticsearch.search.SearchHit;
import org.elasticsearch.common.unit.TimeValue;

import java.io.IOException;
import java.time.Instant;
import java.util.concurrent.TimeUnit;

public class WebServerLogFetcher {

   public static void main(String[] args) throws IOException {
      // 1. Підключення до Elasticsearch
      RestHighLevelClient client = new RestHighLevelClient(
         RestClient.builder(new org.apache.http.HttpHost("localhost", 9200, "http")));
```

```
        // 2. Побудова запиту
        SearchRequest searchRequest = new SearchRequest("weblogs-*");
        SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();

        // 3. Фільтр по часу: останні 5 хвилин
        RangeQueryBuilder timeRange = QueryBuilders
           .rangeQuery("@timestamp") // поле часу
           .gte("now-5m")
           .lte("now");

        BoolQueryBuilder boolQuery = QueryBuilders.boolQuery().filter(timeRange);
        sourceBuilder.query(boolQuery);
        sourceBuilder.size(1000000); // кількість логів

        // Час очікування: timeout
        sourceBuilder.timeout(new TimeValue(30, TimeUnit.SECONDS));
        searchRequest.source(sourceBuilder);

        // 4. Виконання запиту
        SearchResponse response = client.search(searchRequest, RequestOptions.DEFAULT);

        // 5. Класифікація результатів
        OutlierDetection.Classify(response);

        // 6. Закриття клієнта
        client.close();
    }
}
```

Класифікація користувацької активності відбувається за допомогою засобів ШІ та будуть детально розглянути в наступному розділі, так як має нероздільний зв’язок з машинним навчанням, є складовою більш загального процесу, та не відображає картину в цілому. Зрозуміло те, що у випадку якщо активність є аномальною, абонента потрібно блокувати. Розглянемо більш прикладну задачу – блокування абонента. Для блокування доступу до інформаційної системи можна використати будь-який мережевий екран, який має API для керування. На сьогодні, для більшості компаній, інформаційні ресурси представляють веб ресурси або веб додатки, відомі як SaaS. В зв’язку з цим, буде доцільним використання WEB Application Firewall. Дане рішення має API для керування, є частиною Azure, та може легко бути вбудованим в K8S кластер за допомогою Terraform. Нижче представлено клас IpBlockFilter який містить метод doFilter, за допомогою якого ми проведемо блокування.

```
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

public class IpBlockFilter implements Filter {

   // Базовий список заблокованих IP-адрес
   private static final Set<String> BLOCKED_IPS = new HashSet<>();

   @Override
   public void init(FilterConfig filterConfig) {
      System.out.println("IP Block Filter ініціалізовано");
   }

   @Override
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
         throws IOException, ServletException {

      HttpServletRequest req = (HttpServletRequest) request;
      String ipAddress = request.getRemoteAddr();

      if (BLOCKED_IPS.contains(ipAddress)) {
         HttpServletResponse res = (HttpServletResponse) response;
         res.setStatus(HttpServletResponse.SC_FORBIDDEN); // 403
         res.getWriter().write("Access denied for IP: " + ipAddress);
         System.out.println("Блоковано запит від: " + ipAddress);
         return;
      }

      chain.doFilter(request, response); // IP не заблокований — передати далі
   }

   @Override
   public void destroy() {
      System.out.println("IP Block Filter знищено");
   }
}
```

Також, WEB Application Firewall може бути скофігуровано за допомогою CLI команди:

```
az network application-gateway waf-policy custom-rule update \
  --name block-bad-ip \
  --policy-name siteWafPolicy \
  --resource-group myRg \
  --set matchConditions[0].matchValues='[ "198.51.100.123" ]'
```

3.3 Впровадження алгоритмів класифікації активності за допомогою ШІ

При розробці програмного інструменту для телеметричного моніторингу аномальної активності в програмних компонентах, інтеграція можливостей машинного навчання (ML) Elastic Stack є ключовим моментом. Функції машинного навчання Elastic полегшують аналіз часових рядів даних для виявлення відхилень від встановлених шаблонів, що дозволяє виявити потенційні загрози безпеці або збої в роботі системи.

Аналіз користувацької активності складається з класифікації, регресії та виявлення аномалій що є складовими процесу Data Frame Analytics.

Класифікація — це задача машинного навчання, в якій модель навчається визначати, до якого з кількох категоричних класів належить об’єкт на основі вхідних даних. Клас Classification.java реалізує налаштування для задач класифікації в рамках Data Frame Analytics. Цей клас дозволяє налаштувати параметри задачі класифікації, такі як вибір цільової змінної, розподіл даних на тренувальні та тестові, а також визначення, скільки інформації про важливість ознак та ймовірності класів слід зберігати. Метод getResultMappings повертає карту класифікації.

```
@SuppressWarnings("unchecked")
  @Override
  public Map<String, Object> getResultMappings(String resultsFieldName, FieldCapabilitiesResponse
fieldCapabilitiesResponse) {
    Map<String, Object> additionalProperties = new HashMap<>();
    additionalProperties.put(resultsFieldName + ".is_training", Collections.singletonMap("type",
BooleanFieldMapper.CONTENT_TYPE));
    additionalProperties.put(
      resultsFieldName + ".prediction_probability",
      Collections.singletonMap("type", NumberFieldMapper.NumberType.DOUBLE.typeName())
    );
    additionalProperties.put(
      resultsFieldName + ".prediction_score",
      Collections.singletonMap("type", NumberFieldMapper.NumberType.DOUBLE.typeName())
    );
    additionalProperties.put(resultsFieldName + ".feature_importance", FEATURE_IMPORTANCE_MAPPING);

    Map<String, FieldCapabilities> dependentVariableFieldCaps =
fieldCapabilitiesResponse.getField(dependentVariable);
    if (dependentVariableFieldCaps == null || dependentVariableFieldCaps.isEmpty()) {
      throw ExceptionsHelper.badRequestException("no mappings could be found for required field [{}]",
DEPENDENT_VARIABLE);
    }
    Object dependentVariableMappingType =
dependentVariableFieldCaps.values().iterator().next().getType();
```

```
        additionalProperties.put(
            resultsFieldName + "." + predictionFieldName,
            Collections.singletonMap("type", dependentVariableMappingType)
        );

        Map<String, Object> topClassesProperties = new HashMap<>();
        topClassesProperties.put("class_name", Collections.singletonMap("type",
dependentVariableMappingType));
        topClassesProperties.put("class_probability", Collections.singletonMap("type",
NumberFieldMapper.NumberType.DOUBLE.typeName()));
        topClassesProperties.put("class_score", Collections.singletonMap("type",
NumberFieldMapper.NumberType.DOUBLE.typeName()));

        Map<String, Object> topClassesMapping = new HashMap<>();
        topClassesMapping.put("type", NestedObjectMapper.CONTENT_TYPE);
        topClassesMapping.put("properties", topClassesProperties);

        additionalProperties.put(resultsFieldName + ".top_classes", topClassesMapping);
        return additionalProperties;
    }
```

Регресія — це задача, де модель передбачає неперервне числове значення на основі вхідних ознак. Клас Regression.java реалізує налаштування для задач регресії в рамках Data Frame Analytics. Цей клас дозволяє налаштувати параметри задачі регресії, такі як вибір цільової змінної, розподіл даних на тренувальні та тестові, а також визначення, скільки інформації про важливість ознак слід зберігати. Метод getResultMappings повертає карту регресії.

```
@Override
    public Map<String, Object> getResultMappings(String resultsFieldName, FieldCapabilitiesResponse
fieldCapabilitiesResponse) {
        Map<String, Object> additionalProperties = new HashMap<>();
        additionalProperties.put(resultsFieldName + ".is_training", Collections.singletonMap("type",
BooleanFieldMapper.CONTENT_TYPE));
        additionalProperties.put(resultsFieldName + ".feature_importance", FEATURE_IMPORTANCE_MAPPING);
        // Prediction field should be always mapped as "double" rather than "float" in order to increase precision
in case of
        // high (over 10M) values of dependent variable.
        additionalProperties.put(
            resultsFieldName + "." + predictionFieldName,
            Collections.singletonMap("type", NumberFieldMapper.NumberType.DOUBLE.typeName())
        );
        return additionalProperties;
    }
```

Виявлення аномалій — це задача, мета якої — знайти нетипові або підозрілі спостереження, які суттєво відрізняються від решти даних. Клас OutlierDetection.java реалізує налаштування для задач виявлення викидів (аномалій) в рамках Data Frame Analytics. Цей клас дозволяє налаштувати

параметри задачі виявлення викидів, такі як вибір методу, обчислення впливу ознак, очікувана частка викидів та стандартизація даних.

```
@Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    OutlierDetection that = (OutlierDetection) o;
    return Objects.equals(nNeighbors, that.nNeighbors)
      && Objects.equals(method, that.method)
      && Objects.equals(featureInfluenceThreshold, that.featureInfluenceThreshold)
      && computeFeatureInfluence == that.computeFeatureInfluence
      && outlierFraction == that.outlierFraction
      && standardizationEnabled == that.standardizationEnabled;
  }
```

У процесі реалізації алгоритмів виявлення аномалій було використано Kibana Sample Data Logs, що включають попередньо налаштовані приклади машинного навчання а також дані, зібрані в рамках дослідження з реальних систем. Через розділ Machine Learning у Kibana було відкрито Data Visualizer, де обрано відповідне представлення даних. Після цього досліджено розподіл значень і наявність ключових полів, що стало основою для налаштування майбутніх задач виявлення аномалій. За допомогою функції Use full data виконано аналіз повного обсягу даних у вибраному діапазоні часу. Дослідження полів дозволило виявити, які з них найбільш репрезентативні для побудови моделей.[31]

Після ознайомлення з даними, було створено три задачі виявлення аномалій із використанням рекомендованих конфігурацій. Перша задача — low_request_rate — застосовує функцію low_count для виявлення аномально низьких частот запитів за IP-адресами. Друга — response_code_rates — використовує count з розбиттям за response.keyword для виявлення незвичних частот відповіді HTTP-кодів. Третя — url_scanning — заснована на функції high_distinct_count і виконує популяційний аналіз поля clientip, щоб виявити клієнтів, які звертаються до незвично великої кількості унікальних URL-адрес. [30]

Після створення завдання його відкривають і запускають відповідний потік даних, щоб розпочати аналіз. Аномалії виявляються, коли спостережувані значення значно відхиляються від прогнозів моделі, а результати доступні через Kibana's Anomaly Explorer і Single Metric Viewer. Ці інструменти надають візуалізацію та

детальну інформацію про виявлені аномалії, включаючи оцінку серйозності та потенційні фактори впливу.

Щоб підвищити швидкість реагування системи моніторингу, було впроваджено механізм сповіщень для завдань виявлення аномалій, використовуючи вбудований фреймворк сповіщень Elastic. Після налаштування та запуску завдань ML було створено сповіщення на основі правил у Kibana > Stack Management > Rules. Було інтегровано тип правила «Виявлення аномалій», яке відстежує вихідні дані завдань машинного навчання на предмет аномальної поведінки. .[32]

Під час налаштування було обрано завдання low_request_rate, і встановлено умову для спрацьовування сповіщень, коли оцінка аномалії перевищує 75 - поріг, що вказує на потенційно критичні відхилення. Заплановано запуск правила кожні 5 хвилин для виявлення аномалій у режимі, близькому до реального часу.

Ось як було визначено правило за допомогою API Kibana:

```
POST /api/alerting/rule
{
  "params": {
    "jobIds": ["low_request_rate"],
    "anomalyThreshold": 75,
    "resultType": "record"
  },
  "consumer": "ml",
  "schedule": { "interval": "5m" },
  "actions": [
    {
      "group": "anomaly_score_match",
      "params": {
        "message": "⚠ Anomaly detected in job {{context.jobIds}} with score {{context.score}} at {{context.timestamp}}"
      },
      "connector_type_id": ".webhook",
      "id": "your-webhook-connector-id"
    }
  ],
  "rule_type_id": "xpack.ml.anomaly_detection_alert"
}
```

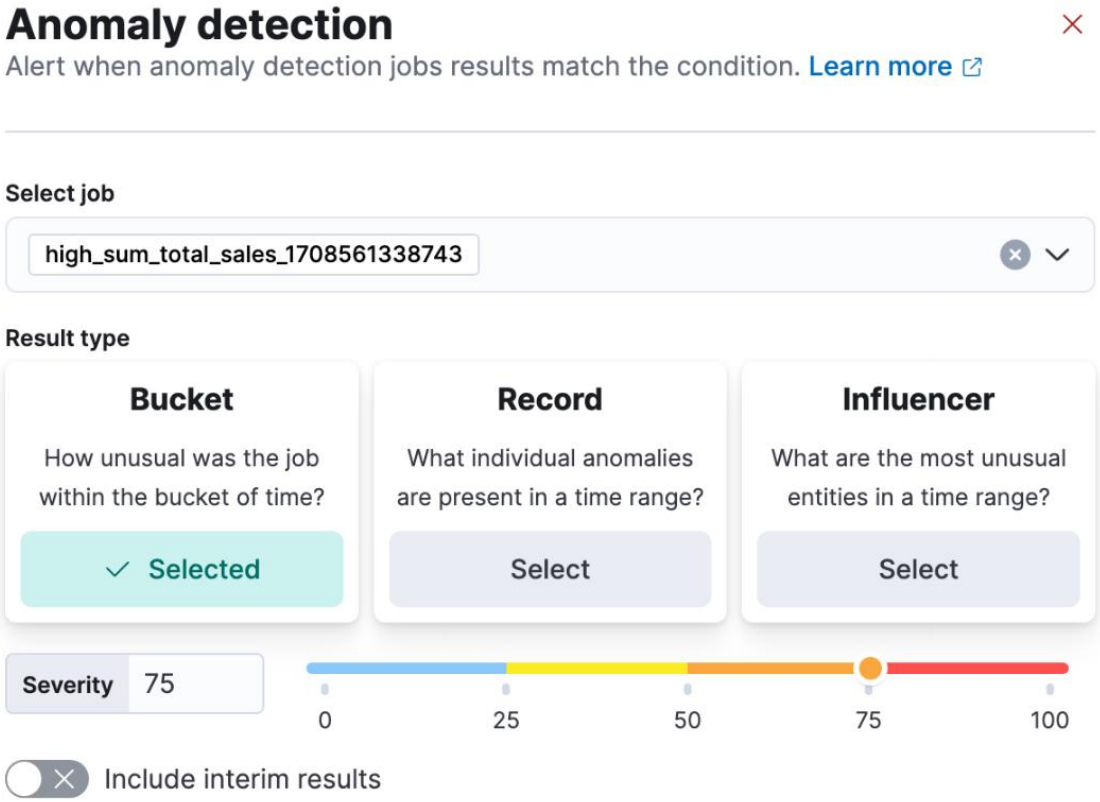


Рисунок 3.1 – Створення умови в Kibana, з порогом відхилення 75

Інтеграція можливостей машинного навчання Elastic в інструмент телеметричного моніторингу покращує проактивну ідентифікацію аномальної поведінки, тим самим посилюючи безпеку та надійність інформаційних систем.

3.4 Налаштування ELK Stack для обробки телеметричних даних

У цьому розділі розглянуто процес розгортання ELK Stack для обробки телеметричних даних у хмарному середовищі. Для автоматизації налаштування інфраструктури обрано інструмент Terraform. Далі буде наведено обґрунтування цього вибору та описано етапи інтеграції ELK Stack із кластером Kubernetes.

Terraform - це інструмент інфраструктури як коду (IaC) з відкритим вихідним кодом, створений компанією HashiCorp. Terraform дозволяє розробникам створювати, оновлювати та знищувати локальні та хмарні компоненти інфраструктури, такі як віртуальні машини та кластери Kubernetes, шляхом написання конфігураційних файлів, придатних для читання людиною. [33]

Terraform використовує власну мову конфігурацій — HCL (HashiCorp Configuration Language). Завдяки HCL ти можеш зберігати конфігурації в репозиторії, редагувати їх, перевіряти перед запуском та повторно застосовувати для створення інфраструктури в різних середовищах. Компоненти інфраструктури, якими керує Terraform, називаються ресурсами. Це можуть бути віртуальні машини, таблиці баз даних, бакети в S3, VPC (віртуальні приватні мережі) та інше. Кожен ресурс описується окремим блоком у HCL, де вказуються його параметри та налаштування. [34]

Нижче наведено простий приклад опису ресурсу для створення VPC:

```
resource “azure_vpc” “default_vpc” {
  cidr_block = “172.31.0.0/16”
  tags = {
    Name = “example_vpc”
} }]
```

Terraform пропонує надійний підхід до управління інфраструктурою Azure, що забезпечує кілька ключових переваг: [35]

- Інструмент Common Infrastructure as Code (IaC) – провайдери Terraform Azure забезпечують централізоване управління всіма ресурсами Azure за допомогою уніфікованого декларативного синтаксису. Це включає в себе налаштування основних компонентів платформи, проектів і конвеєрів Azure DevOps для автоматизації розгортання інфраструктури та додатків та надання ресурсів Azure, необхідних для додатків.
- Автоматизоване управління інфраструктурою – синтаксис конфігурації Terraform на основі шаблонів дозволяє визначати і розгортати інфраструктуру в повторюваній і передбачуваній манері, що дозволяє повторно використовувати шаблони для створення ідентичних середовищ розробки, тестування та виробництва.
- Попередній перегляд змін в інфраструктурі – оскільки системи стають все більш складними, розуміння і контроль змін стає критично важливим. Terraform CLI надає інструменти для перевірки та попереднього перегляду запланованих модифікацій інфраструктури, які: полегшують командну

роботу та допомагають виявити і запобігти ненавмисним модифікаціям на ранній стадії процесу розробки.

Разом ці функції роблять Terraform потужним інструментом для підтримки безпечної, стабільної та ефективної інфраструктури в екосистемі Azure.

Отже, щоб реалізувати масштабоване середовище для Elastic Stack, використовуємо Terraform для надання інфраструктури на AKS. Цей підхід забезпечив автоматизацію, відтворюваність та відповідність принципам Infrastructure as Code (IaC).

На рисунку 3.1 зображено інфраструктуру середовища розгортання.

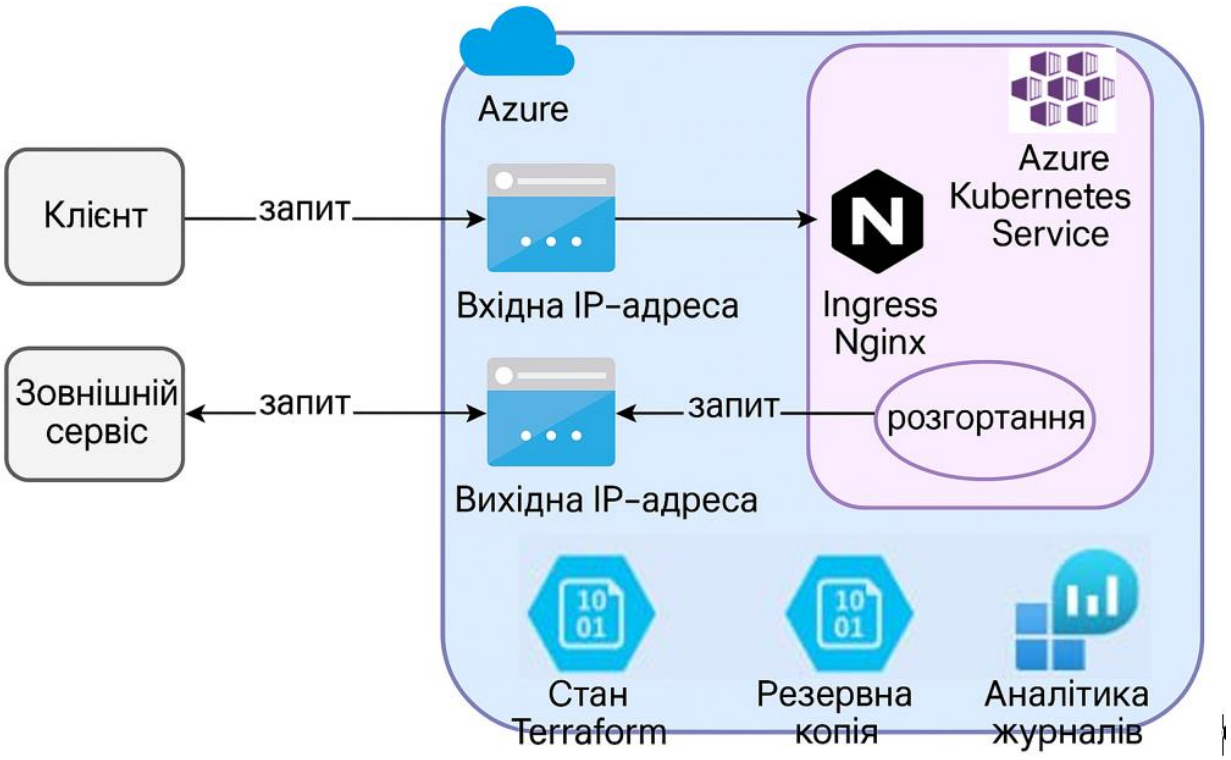


Рисунок 3.2 – Інфраструктура середовища розгортання

Процес розгортання розпочався з клонування попередньо налаштованого проекту Terraform:

```
git clone https://github.com/mchudinov/K8sAzureTerraform.git
cd K8sAzureTerraform
```

Використовуючи наданий скрипт deploy.sh, було створено кластер AKS, визначивши ключові параметри, такі як ім'я кластера, кількість вузлів, регіон та облікові дані головної служби:

```
./deploy.sh -c mytestk8s -n 3 -r westeurope -p <service_principal_id> -s terraformstate
```

де:

-c задає ім'я кластера,

-n визначає кількість робочих вузлів,

-r задає регіон Azure,

-p надає основний ідентифікатор служби для автентифікації,

-s визначає обліковий запис для зберігання стану Terraform.

Це автоматизувало створення:

- кластера Kubernetes в Azure,
- статичні публічні IP-адреси для вхідного та вихідного трафіку,
- NGINX-контролера входу для маршрутизації,
- драйвера Azure CSI для безпечного управління секретами,
- робочої області Azure Log Analytics для моніторингу.

Після успішного виконання Terraform повернула результати, які були використані для доступу та управління кластером:

```
fqdn = "k8s-mytestk8s-<unique_id>.hcp.westeurope.azmk8s.io"
host = "https://k8s-mytestk8s-<unique_id>.hcp.westeurope.azmk8s.io:443"
public_ip_inbound_address = "<inbound_ip_address>"
```

Щоб керувати кластером за допомогою kubectl, було додано його облікові дані до локального kubeconfig:

```
az aks get-credentials --name mytestk8s --resource-group rg-mytestk8s
```

Було перевірено, що драйвер Azure Key Vault CSI розгорнуто правильно:

```
kubectl get csidrivers
kubectl describe csidriver secrets-store.csi.k8s.io
kubectl get pods -l app=secrets-store-csi-driver
```

Використовуючи Terraform, було ефективно визначено та надано інфраструктуру Azure, забезпечуючи узгодженість між середовищами розробки та тестування. Ця основа відіграла вирішальну роль в успішному розгортанні компонентів Elastic Stack на AKS.

3.5 Тестування програмного засобу

У даному буде проведено тестування програмного засобу захисту. Для тестування запропоновано сценарій:

1. Користувач перевіряє з’єднання та доступність веб ресурсу.
2. Користувач відправляє запит з аномальною активністю, наприклад з вмістом SQL ін’єкції.
3. Система виявляє аномальну активність та блокує користувача по IP.
4. Користувач спостерігає проблеми з’єднання, веб ресурс не доступний.

Отже, маємо веб ресурс «https://dasha-pr-fkggcdcybgafh2g9.centralus-01.azurewebsites.net», та перевіряємо з’єднання з ним.

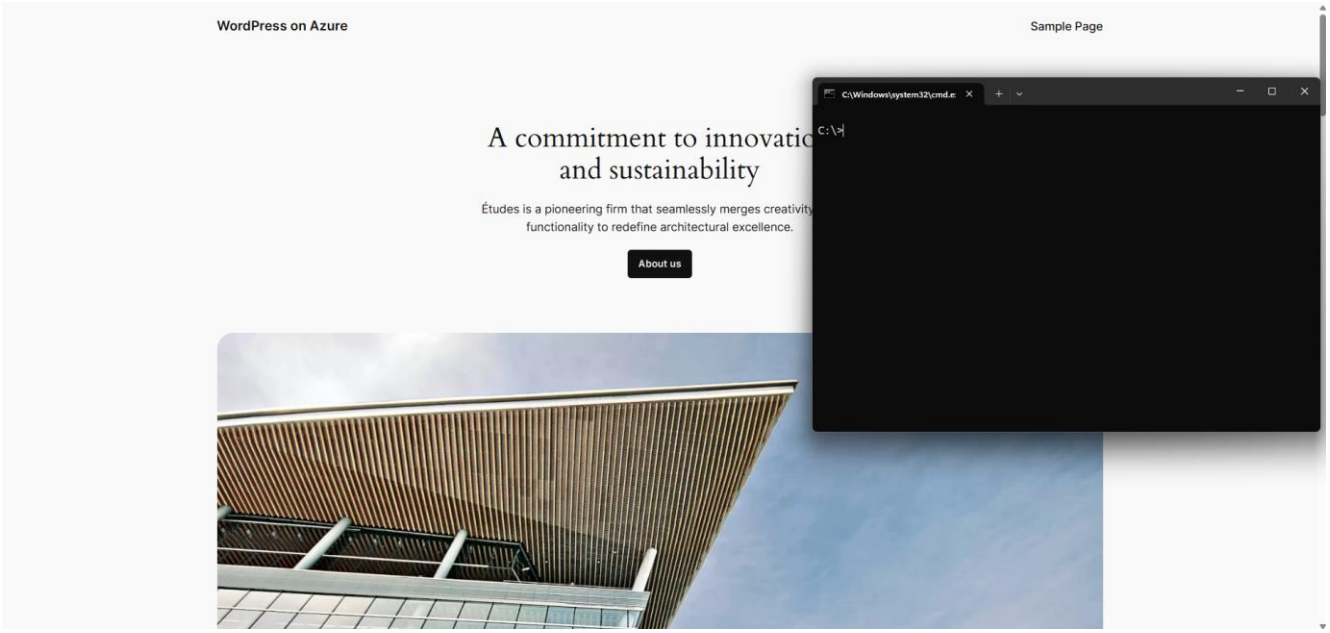


Рисунок 3.3 – Перевірка доступності та з’єднання з ресурсом.

Як показано на рисунку 3.3, користувач має доступ до веб ресурсу, з’єднання стабільне. Використовуючи утиліту curl, відправимо аномальний запит який містить SQL ін’єкцію (рисунок 3.4).

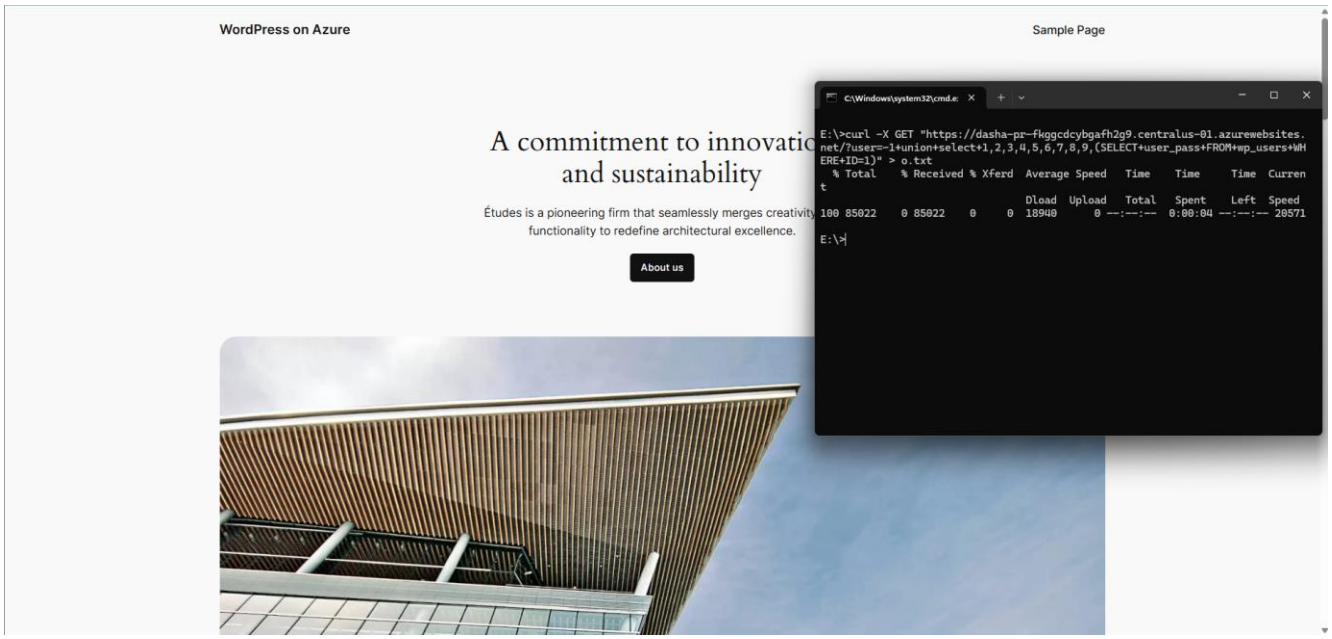


Рисунок 3.4 – Відправка аномальної активності.

Після відправки аномального запиту, можна спостерігати що доступ до ресурсу заблоковано.

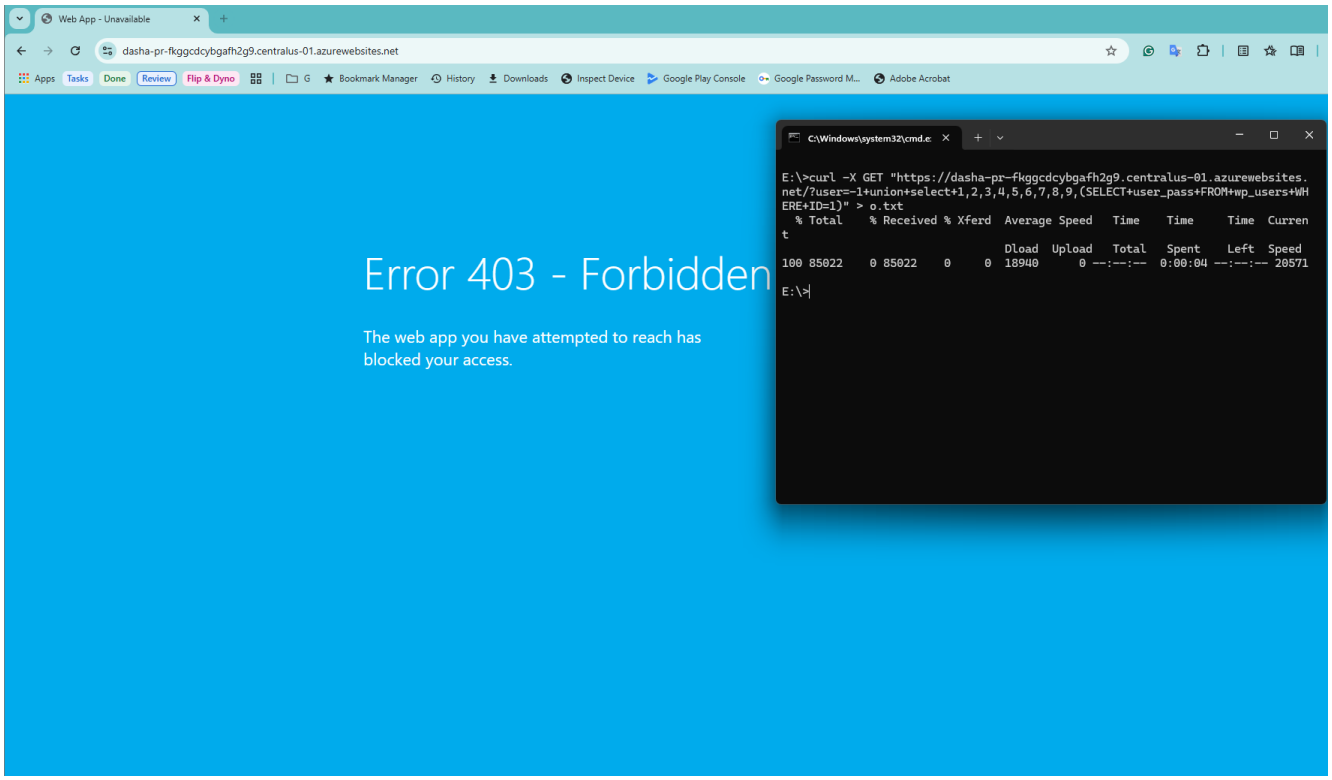


Рисунок 3.5 – Результат блокування користувача.

Отже, в результаті тестування ми переконались в коректності роботи програмного засобу. Доступ до ресурсу було заблоковано, час спрацювання менше 150мс що є чудовим показником. Додатково варто зауважити що система здатна до самовдосконалення, за рахунок навчання. Тобто, можна очікувати на меншу кількість хибних спрацювань, та обробка виключень будь-якого типу нових, невідомих (Zero-Day) атак.

3.6 Висновки до розділу 3

У цьому розділі було реалізовано програмну частину засобу телеметричного моніторингу аномальних активностей з інтеграцією технологій штучного інтелекту та стеку ELK. Проведено обґрунтований вибір середовища розгортання, де найбільш доцільним виявилося використання хмарної платформи Microsoft Azure з Azure Kubernetes Service (AKS), що забезпечує масштабованість, гнучкість та високу надійність інфраструктури.

Створено базову архітектуру контейнеризованої системи, налагоджено збір та обробку телеметрії з подальшою класифікацією даних за допомогою моделей машинного навчання Elastic ML. Здійснено реалізацію сценаріїв виявлення аномальної поведінки користувачів із автоматичним блокуванням доступу до інформаційного ресурсу. Налаштовано механізм сповіщень на основі порогових значень, що дозволяє реагувати на інциденти у режимі, наближеному до реального часу.

Візуалізація результатів була забезпечена через Kibana, а інфраструктура – розгорнута і автоматизована засобами Terraform. Проведене тестування показало високу ефективність системи: швидкість реагування на аномалії становить менше 150 мс, що є відмінним показником для систем моніторингу безпеки.

.

ВИСНОВКИ

У ході виконання бакалаврської роботи було успішно розв’язано поставлену науково-прикладну проблему: розробку та реалізацію програмного засобу для телеметричного моніторингу аномальних активностей інформаційних систем з використанням технологій штучного інтелекту та компонентів ELK Stack.

У рамках виконання задачі було спроєктовано архітектуру, що охоплює всі ключові етапи життєвого циклу телеметричних даних від збору до реагування. Створена система забезпечує високу продуктивність обробки в режимі реального часу, демонструючи високу пропускну здатність і відмовостійкість.

Для автоматичного аналізу телеметрії було впроваджено модуль машинного навчання, який використовує кластеризацію, різні типи декомпозиції часових рядів, моделювання байєсівського розподілу та кореляційний аналіз.

Розроблений модуль візуалізації на базі Kibana забезпечує інтерактивну побудову дашбордів та діаграм, що відображають динаміку ключових параметрів, дозволяючи візуально ідентифікувати аномалії та оцінювати масштаб інцидентів.

Усі дані телеметрії обробляються через конвеєри Logstash та зберігаються в Elasticsearch, що дозволило досягти середнього часу відповіді на пошукові запити менше ніж 150 мс, навіть при високому навантаженні. Таким чином, було забезпечено виконання п’ятої задачі.

Механізм реагування включає систему автоматичних сповіщень, що працює на основі оцінки критичності інцидентів. Рівень затримки між виявленням аномалії та її передачею відповідальній особі не перевищує 3 секунд, що відповідає вимогам реального часу.

Отримані результати можуть бути використані для побудови систем моніторингу в корпоративних мережах, середовищах IoT та кіберзахисту критичної інфраструктури. Подальші дослідження доцільно спрямувати на вдосконалення системи обробки інцидентів за допомогою систем обробки черг на зразок Apache Kafka та Rabbit MQ.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аномалія / В. І. Бондар, В. В. Засенко // Енциклопедія Сучасної України [Електронний ресурс] / Редкол. : І. М. Дзюба, А. І. Жуковський, М. Г. Железняк [та ін.] ; НАН України, НТШ. – К. : Інститут енциклопедичних досліджень НАН України, 2001. – Режим доступу: https://esu.com.ua/article-44373 (дата звернення 19.03.2025).

2. cucis.ece.northwestern.edu. – Режим доступу: http://cucis.ece.northwestern.edu/projects/DMS/publications/AnomalyDetection.pdf (дата звернення 19.03.2025).

3. link.springer.com. – Режим доступу: https://link.springer.com/article/10.1007/s41060-024-00586-x (дата звернення 21.03.2025).

4. What is anomaly detection?. Darktrace. Anomaly Detection. – Режим доступу: https://darktrace.com/cyber-ai-glossary/anomaly-detection (дата звернення 21.03.2025).

5. rflow.ai. – Режим доступу: https://rflow.ai/researches/how-to-study-anomaly-detection-in-cybersecurity (дата звернення 25.03.2025).

6. e-tk.lntu.edu.ua. – Режим доступу: https://e-tk.lntu.edu.ua/pluginfile.php/25373/mod_resource/content/1/%D0%A2%D0%95%D0%9C%D0%90%2014.pdf (дата звернення 25.03.2025).

7. vpnunlimited.com. – Режим доступу: https://www.vpnunlimited.com/ua/help/cybersecurity/telemetry (дата звернення 28.03.2025).

8. microsoft.com. – Режим доступу: https://www.microsoft.com/en-us/security/business/security-101/what-is-siem (дата звернення 31.03.2025).

9. corelight.com. – Режим доступу: https://corelight.com/resources/glossary/siem-security-information-event-management (дата звернення 31.03.2025).

10. techtarget.com. – Режим доступу: https://www.techtarget.com/whatis/definition/telemetry (дата звернення 31.03.2025).

11. ibm.com. – Режим доступу: https://www.ibm.com/think/topics/anomaly-detection (дата звернення 04.04.2025).

12. freehost.com.ua. – Режим доступу: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-elk-stack/#:~:text=... (дата звернення 07.04.2025).

13. aws.amazon.com. – Режим доступу: https://aws.amazon.com/what-is/elk-stack/#:~:text=... (дата звернення 09.04.2025).

14. techtarget.com. – Режим доступу: https://www.techtarget.com/searchitoperations/definition/Elastic-Stack (дата звернення 09.04.2025).

15. foxminded.ua. – Режим доступу: https://foxminded.ua/mikroservisna-arkhitektura/ (дата звернення 09.04.2025).

16. it.nmu.org.ua. – Режим доступу: https://it.nmu.org.ua/ua/scientific_method_materials/lecture_notes/Конспект_лекцій_Проєктування_IC_2020.pdf (дата звернення 11.04.2025).

17. mmsa.kpi.ua. – Режим доступу: http://mmsa.kpi.ua/sites/default/files/disciplines/Розробка%20і%20тестування%20програм/didkovska_m_v_testing_lecture_6.pdf (дата звернення 11.04.2025).

18. duikt.edu.ua. – Режим доступу: https://duikt.edu.ua/ua/news-1-0-8002-zastosuvannya-uml-chastina-3-diagrama-klasiv----class-diagram (дата звернення 15.04.2025).

19. elastic.co. – Режим доступу: https://www.elastic.co/docs/explore-analyze/machine-learning/anomaly-detection/ml-ad-algorithms (дата звернення 15.04.2025).

20. cloud4u.com. – Режим доступу: https://www.cloud4u.com/blog/cloud-vs-vds-vps/ (дата звернення 18.04.2025).

21. sim-networks.com. – Режим доступу: https://www.sim-networks.com/ukr/blog/vps-vs-cloud-vm (дата звернення 18.04.2025).

22. onbiz.biz. – Режим доступу: https://onbiz.biz/cloud-vs-dedicated-server-where-to-store-data/ (дата звернення 18.04.2025).

23. microsoft.com. – Режим доступу: https://www.microsoft.com/uk-ua/microsoft-365/business-insights-ideas/resources/cloud-storage-vs-on-premises-servers (дата звернення 21.04.2025).

24. citzi.vntu.net. – Режим доступу: https://citzi.vntu.net/office-365/ (дата звернення 21.04.2025).

25. azure.microsoft.com. – Режим доступу: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure (дата звернення 21.04.2025).

26. hub.kyivstar.ua. – Режим доступу: https://hub.kyivstar.ua/articles/shho-take-microsoft-azure-najvazhlyvishe-pro-globalnu-hmarnu-platformu (дата звернення 23.04.2025).

27. simpla.com.ua. – Режим доступу: https://simpla.com.ua/blog/shcho-take-microsoft-azure (дата звернення 23.04.2025).

28. learn.microsoft.com. – Режим доступу: https://learn.microsoft.com/en-us/azure/aks/what-is-aks (дата звернення 25.04.2025).

29. netapp.com. – Режим доступу: https://www.netapp.com/learn/azure-anf-blg-kubernetes-in-azure-architecture-and-service-options/ (дата звернення 25.04.2025).

30. elastic.co. – Режим доступу: https://sites.google.com/elastic.co/ml-workshop/lab-1-sample-anomaly-detection-jobs (дата звернення 30.04.2025).

31. elastic.co. – Режим доступу: https://www.elastic.co/docs/explore-analyze/machine-learning/anomaly-detection/ml-getting-started (дата звернення 30.04.2025).

32. elastic.co. – Режим доступу: https://www.elastic.co/docs/explore-analyze/machine-learning/anomaly-detection/ml-configuring-alerts (дата звернення 01.05.2025).

33. ibm.com. – Режим доступу: https://www.ibm.com/think/topics/terraform (дата звернення 02.05.2025).

34. itedu.center. – Режим доступу: https://itedu.center/ua/blog/review/shho_take_terraform/?srsltid=AfmBOor1ktTS5dJN6xRJMTN8t4ZUTb37yd7U3XbXarIoa2SzGyPt2D5W (дата звернення 05.05.2025).

35. learn.microsoft.com. – Режим доступу: https://learn.microsoft.com/en-us/azure/developer/terraform/overview (дата звернення 05.05.2025).

36. trustradius.com. – Режим доступу: https://www.trustradius.com/products/darktrace/reviews/all (дата звернення 12.05.2025).

37. gartner.com. – Режим доступу: https://www.gartner.com/reviews/market/network-detection-and-response/vendor/vectra-ai/product/vectra-ai-platform/likes-dislikes (дата звернення 12.05.2025).

38. teramind.co. – Режим доступу: https://www.teramind.co/blog/crowdstrike-pros-and-cons/ (дата звернення 12.05.2025).

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
" 20 " березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської кваліфікаційної роботи на тему:
Розробка програмного засобу телеметричного моніторингу аномальних активностей програмних компонентів для захисту інформаційних систем з використанням технології ШІ та ELK stack

08-72.БКР.022.00.114.ТЗ

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доц., зав. каф. МБІС
_____ Карпінець В. В.
" _____ " _____

Вінниця – 2025 р.

1. Найменування та область застосування

Програмний засіб телеметричного моніторингу аномальних активностей програмних компонентів для захисту інформаційних систем з використанням технології ШІ та ELK Stack.

Область застосування: забезпечення виявлення та реагування на аномальну активність у програмних компонентах у реальному часі. Програмний засіб призначено для використання в інформаційних системах з підвищеними вимогами до безпеки, зокрема в корпоративних, державних та інфраструктурних середовищах.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 97 від 20.03.2025 р.

3. Мета та призначення розробки

3.1 Мета розробки: підвищення рівня захищеності інформаційних систем шляхом створення інтелектуального механізму моніторингу телеметричних даних і виявлення аномалій із використанням алгоритмів машинного навчання та стека технологій ELK.

3.2 Призначення: розроблений програмний засіб здійснює збір, обробку та аналіз телеметрії з подальшим автоматичним виявленням і реагуванням на підозрілу активність, що дозволяє зменшити час реагування на загрози та покращити інформованість адміністраторів щодо інцидентів безпеки.

4. Джерела розробки

4.1. elastic.co. – Режим доступу: https://www.elastic.co/docs/explore-analyze/machine-learning/anomaly-detection/ml-getting-started

4.2. microsoft.com. – Режим доступу: https://www.microsoft.com/en-us/security/business/security-101/what-is-siem

4.3. ami.lnu.edu.ua. – Режим доступу: https://ami.lnu.edu.ua/wp-content/uploads/2024/02/Povtoratskyy.pdf (дата звернення 06.06.2025).

4.4. elastic.co. – Режим доступу: https://www.elastic.co/docs/explore-analyze/machine-learning/anomaly-detection/ml-ad-algorithms

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача, що забезпечує доступ до телеметричних дашбордів та інтерпретації аномалій.

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків — використано open-source компоненти, зокрема Elasticsearch, Logstash, Kibana та фреймворки машинного навчання.

5.1.3 Програмний засіб повинен виконувати процес автентифікації користувачів у системі, використовуючи рольову модель доступу та багатофакторну автентифікацію.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, а у випадку критичних ситуацій виводити відповідні повідомлення та логи подій у Kibana.

5.2.2 Бази даних повинні бути налаштовані на автоматичне створення резервних копій індексів у Elasticsearch.

5.2.3 Програмний засіб повинен виконувати свої функції збору, аналізу, візуалізації та реагування у режимі реального часу з урахуванням вимог до стійкості системи.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb (рекомендовано 4 Gb для оптимальної роботи з ML-модулями);

– середовище функціонування – операційна система сімейства Windows або Linux з підтримкою Docker-контейнерів;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати чинним стандартам з охорони праці при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4, включає порядок розгортання ELK Stack та запуск модулів машинного навчання.

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання шляхом реалізації механізмів аутентифікації та логування доступів.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів гарантується за допомогою контрольованого доступу та системи ролей у Kibana.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проєкту повинна перевищувати витрати на його реалізацію завдяки використанню безкоштовних рішень і автоматизації реагування.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу – зокрема, в установах із підвищеними вимогами до безпеки.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025
2.	Аналіз предметної області обраної теми	24.03.2025	13.04.2025
3.	Розробка алгоритму роботи	14.04.2025	27.04.2025
4.	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025
5.	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	08.06.2025
7.	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

– ПЗ до бакалаврської кваліфікаційної роботи;

- демонстрація результату бакалаврської кваліфікаційної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензент

Технічне завдання до виконання прийняв _____ Присяжна Д. О.

Додаток Б. Лістинг програми

Створення задачі класифікації

```
import org.elasticsearch.client.ml.dataframe.DataFrameAnalyticsConfig;
import org.elasticsearch.client.ml.dataframe.DataFrameAnalyticsSource;
import org.elasticsearch.client.ml.dataframe.DataFrameAnalyticsDest;
import org.elasticsearch.client.ml.dataframe.analyses.Classification;
import org.elasticsearch.client.ml.dataframe.analyses.DataFrameAnalysis;
import org.elasticsearch.client.ml.PutDataFrameAnalyticsRequest;
import org.elasticsearch.client.ml.PutDataFrameAnalyticsResponse;
import org.elasticsearch.client.ml.StartDataFrameAnalyticsRequest;
import org.elasticsearch.client.ml.StartDataFrameAnalyticsResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestHighLevelClient;

...

// Ініціалізація клієнта Elasticsearch
RestHighLevelClient client = new RestHighLevelClient(RestClient.builder(new HttpHost("localhost", 9200, "http")));

// Налаштування джерела даних
DataFrameAnalyticsSource source = new DataFrameAnalyticsSource.Builder()
   .setIndex("source_index")
   .build();

// Налаштування призначення результатів
DataFrameAnalyticsDest dest = new DataFrameAnalyticsDest.Builder()
   .setIndex("dest_index")
   .build();

// Налаштування аналізу класифікації
Classification classification = new Classification("target_field");

// Створення конфігурації задачі
DataFrameAnalyticsConfig config = new DataFrameAnalyticsConfig.Builder()
   .setId("classification_job")
   .setSource(source)
   .setDest(dest)
   .setAnalysis(classification)
   .build();

// Відправка запиту на створення задачі
PutDataFrameAnalyticsRequest putRequest = new PutDataFrameAnalyticsRequest(config);
PutDataFrameAnalyticsResponse putResponse = client.machineLearning().putDataFrameAnalytics(putRequest, RequestOptions.DEFAULT);

// Запуск задачі
```

```
StartDataFrameAnalyticsRequest startRequest = new StartDataFrameAnalyticsRequest("classification_job");
StartDataFrameAnalyticsResponse startResponse = client.machineLearning().startDataFrameAnalytics(startRequest, RequestOptions.DEFAULT);
```

Створення задачі регресії

```
import org.elasticsearch.client.ml.dataframe.analyses.Regression;

...

// Налаштування аналізу регресії
Regression regression = new Regression("target_field");

// Створення конфігурації задачі
DataFrameAnalyticsConfig config = new DataFrameAnalyticsConfig.Builder()
   .setId("regression_job")
   .setSource(source)
   .setDest(dest)
   .setAnalysis(regression)
   .build();

// Відправка запиту на створення задачі
PutDataFrameAnalyticsRequest putRequest = new PutDataFrameAnalyticsRequest(config);
PutDataFrameAnalyticsResponse putResponse = client.machineLearning().putDataFrameAnalytics(putRequest, RequestOptions.DEFAULT);

// Запуск задачі
StartDataFrameAnalyticsRequest startRequest = new StartDataFrameAnalyticsRequest("regression_job");
StartDataFrameAnalyticsResponse startResponse = client.machineLearning().startDataFrameAnalytics(startRequest, RequestOptions.DEFAULT);
```

Створення задачі виявлення аномалій

```
import org.elasticsearch.client.ml.dataframe.analyses.OutlierDetection;

...

// Налаштування аналізу виявлення викидів
OutlierDetection outlierDetection = new OutlierDetection();

// Створення конфігурації задачі
DataFrameAnalyticsConfig config = new DataFrameAnalyticsConfig.Builder()
   .setId("outlier_detection_job")
   .setSource(source)
   .setDest(dest)
   .setAnalysis(outlierDetection)
   .build();

// Відправка запиту на створення задачі
```

```
PutDataFrameAnalyticsRequest putRequest = new PutDataFrameAnalyticsRequest(config);
PutDataFrameAnalyticsResponse putResponse = client.machineLearning().putDataFrameAnalytics(putRequest, RequestOptions.DEFAULT);

// Запуск задачі
StartDataFrameAnalyticsRequest startRequest = new StartDataFrameAnalyticsRequest("outlier_detection_job");
StartDataFrameAnalyticsResponse startResponse = client.machineLearning().startDataFrameAnalytics(startRequest, RequestOptions.DEFAULT);

package org.elasticsearch.xpack.core.ml.dataframe.analyses;

import org.elasticsearch.action.fieldcaps.FieldCapabilities;
import org.elasticsearch.action.fieldcaps.FieldCapabilitiesResponse;
import org.elasticsearch.common.Randomness;
import org.elasticsearch.common.io.stream.StreamInput;
import org.elasticsearch.common.io.stream.StreamOutput;
import org.elasticsearch.core.Nullable;
import org.elasticsearch.index.mapper.BooleanFieldMapper;
import org.elasticsearch.index.mapper.KeywordFieldMapper;
import org.elasticsearch.index.mapper.NestedObjectMapper;
import org.elasticsearch.index.mapper.NumberFieldMapper;
import org.elasticsearch.xcontent.ConstructingObjectParser;
import org.elasticsearch.xcontent.ParseField;
import org.elasticsearch.xcontent.XContentBuilder;
import org.elasticsearch.xcontent.XContentParser;
import org.elasticsearch.xpack.core.ml.MlConfigVersion;
import org.elasticsearch.xpack.core.ml.inference.preprocessing.LenientlyParsedPreProcessor;
import org.elasticsearch.xpack.core.ml.inference.preprocessing.PreProcessor;
import org.elasticsearch.xpack.core.ml.inference.preprocessing.StrictlyParsedPreProcessor;
import org.elasticsearch.xpack.core.ml.inference.trainedmodel.ClassificationConfig;
import org.elasticsearch.xpack.core.ml.inference.trainedmodel.InferenceConfig;
import org.elasticsearch.xpack.core.ml.inference.trainedmodel.PredictionFieldType;
import org.elasticsearch.xpack.core.ml.utils.ExceptionsHelper;
import org.elasticsearch.xpack.core.ml.utils.NamedXContentObjectHelper;

import java.io.IOException;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Objects;
import java.util.Set;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import static org.elasticsearch.xcontent.ConstructingObjectParser.constructorArg;
```

```
import static org.elasticsearch.xcontent.ConstructingObjectParser.optionalConstructorArg;

public class Classification implements DataFrameAnalysis {

   public static final ParseField NAME = new ParseField("classification");

   public static final ParseField DEPENDENT_VARIABLE = new ParseField("dependent_variable");
   public static final ParseField PREDICTION_FIELD_NAME = new ParseField("prediction_field_name");
   public static final ParseField CLASS_ASSIGNMENT_OBJECTIVE = new ParseField("class_assignment_objective");
   public static final ParseField NUM_TOP_CLASSES = new ParseField("num_top_classes");
   public static final ParseField TRAINING_PERCENT = new ParseField("training_percent");
   public static final ParseField RANDOMIZE_SEED = new ParseField("randomize_seed");
   public static final ParseField FEATURE_PROCESSORS = new ParseField("feature_processors");
   public static final ParseField EARLY_STOPPING_ENABLED = new ParseField("early_stopping_enabled");

   private static final String STATE_DOC_ID_INFIX = "_classification_state#";

   private static final String NUM_CLASSES = "num_classes";

   private static final ConstructingObjectParser<Classification, Void> LENIENT_PARSER = createParser(true);
   private static final ConstructingObjectParser<Classification, Void> STRICT_PARSER = createParser(false);

   /**
    * The max number of classes classification supports
    */
   public static final int MAX_DEPENDENT_VARIABLE_CARDINALITY = 100;

   @SuppressWarnings("unchecked")
   private static ConstructingObjectParser<Classification, Void> createParser(boolean lenient) {
      ConstructingObjectParser<Classification, Void> parser = new ConstructingObjectParser<>(
         NAME.getPreferredName(),
         lenient,
         a -> new Classification(
            (String) a[0],
            new BoostedTreeParams(
               (Double) a[1],
               (Double) a[2],
               (Double) a[3],
               (Integer) a[4],
               (Double) a[5],
               (Integer) a[6],
               (Double) a[7],
               (Double) a[8],
```

```
                    (Double) a[9],
                    (Double) a[10],
                    (Double) a[11],
                    (Integer) a[12]
                ),
                (String) a[13],
                (ClassAssignmentObjective) a[14],
                (Integer) a[15],
                (Double) a[16],
                (Long) a[17],
                (List<PreProcessor>) a[18],
                (Boolean) a[19]
            )
        );
        parser.declareString(constructorArg(), DEPENDENT_VARIABLE);
        BoostedTreeParams.declareFields(parser);
        parser.declareString(optionalConstructorArg(), PREDICTION_FIELD_NAME);
        parser.declareString(optionalConstructorArg(), ClassAssignmentObjective::fromString,
CLASS_ASSIGNMENT_OBJECTIVE);
        parser.declareInt(optionalConstructorArg(), NUM_TOP_CLASSES);
        parser.declareDouble(optionalConstructorArg(), TRAINING_PERCENT);
        parser.declareLong(optionalConstructorArg(), RANDOMIZE_SEED);
        parser.declareNamedObjects(
          optionalConstructorArg(),
          (p, c, n) -> lenient
            ? p.namedObject(LenientlyParsedPreProcessor.class, n, new
PreProcessor.PreProcessorParseContext(true))
            : p.namedObject(StrictlyParsedPreProcessor.class, n, new
PreProcessor.PreProcessorParseContext(true)),
          (classification) -> {/*TODO should we throw if this is not set?*/},
          FEATURE_PROCESSORS
        );
        parser.declareBoolean(optionalConstructorArg(), EARLY_STOPPING_ENABLED);
        return parser;
      }

      public static Classification fromXContent(XContentParser parser, boolean
ignoreUnknownFields) {
        return ignoreUnknownFields ? LENIENT_PARSER.apply(parser, null) :
STRICT_PARSER.apply(parser, null);
      }

      private static final Set<String> ALLOWED_DEPENDENT_VARIABLE_TYPES = Stream.of(
        Types.categorical(),
        Types.discreteNumerical(),
        Types.bool()
      ).flatMap(Set::stream).collect(Collectors.toUnmodifiableSet());
      /**
       * Name of the parameter passed down to C++.
```

```
     * This parameter is used to decide which JSON data type from {string, int, bool} to use when writing the prediction.
     */
    private static final String PREDICTION_FIELD_TYPE = "prediction_field_type";

    /**
     * As long as we only support binary classification it makes sense to always report both classes with their probabilities.
     * This way the user can see if the prediction was made with confidence they need.
     */
    private static final int DEFAULT_NUM_TOP_CLASSES = 2;

    private static final List<String> PROGRESS_PHASES = Collections.unmodifiableList(
        Arrays.asList("feature_selection", "coarse_parameter_search", "fine_tuning_parameters", "final_training")
    );

    static final Map<String, Object> FEATURE_IMPORTANCE_MAPPING;
    static {
        Map<String, Object> classesProperties = new HashMap<>();
        classesProperties.put("class_name", Collections.singletonMap("type", KeywordFieldMapper.CONTENT_TYPE));
        classesProperties.put("importance", Collections.singletonMap("type", NumberFieldMapper.NumberType.DOUBLE.typeName()));

        Map<String, Object> classesMapping = new HashMap<>();
        classesMapping.put("dynamic", false);
        classesMapping.put("type", NestedObjectMapper.CONTENT_TYPE);
        classesMapping.put("properties", classesProperties);

        Map<String, Object> properties = new HashMap<>();
        properties.put("feature_name", Collections.singletonMap("type", KeywordFieldMapper.CONTENT_TYPE));
        properties.put("classes", classesMapping);

        Map<String, Object> mapping = new HashMap<>();
        mapping.put("dynamic", false);
        mapping.put("type", NestedObjectMapper.CONTENT_TYPE);
        mapping.put("properties", properties);

        FEATURE_IMPORTANCE_MAPPING = Collections.unmodifiableMap(mapping);
    }

    private final String dependentVariable;
    private final BoostedTreeParams boostedTreeParams;
    private final String predictionFieldName;
    private final ClassAssignmentObjective classAssignmentObjective;
    private final int numTopClasses;
    private final double trainingPercent;
```

```
        private final long randomizeSeed;
        private final List<PreProcessor> featureProcessors;
        private final boolean earlyStoppingEnabled;

        public Classification(
            String dependentVariable,
            BoostedTreeParams boostedTreeParams,
            @Nullable String predictionFieldName,
            @Nullable ClassAssignmentObjective classAssignmentObjective,
            @Nullable Integer numTopClasses,
            @Nullable Double trainingPercent,
            @Nullable Long randomizeSeed,
            @Nullable List<PreProcessor> featureProcessors,
            @Nullable Boolean earlyStoppingEnabled
        ) {
            if (numTopClasses != null && (numTopClasses < -1 || numTopClasses > 1000)) {
                throw ExceptionsHelper.badRequestException(
                    "[{}] must be an integer in [0, 1000] or a special value -1",
                    NUM_TOP_CLASSES.getPreferredName()
                );
            }
            if (trainingPercent != null && (trainingPercent <= 0.0 || trainingPercent > 100.0)) {
                throw ExceptionsHelper.badRequestException("[{}] must be a positive double in (0, 100]", TRAINING_PERCENT.getPreferredName());
            }
            this.dependentVariable = ExceptionsHelper.requireNonNull(dependentVariable, DEPENDENT_VARIABLE);
            this.boostedTreeParams = ExceptionsHelper.requireNonNull(boostedTreeParams, BoostedTreeParams.NAME);
            this.predictionFieldName = predictionFieldName == null ? dependentVariable + "_prediction" : predictionFieldName;
            this.classAssignmentObjective = classAssignmentObjective == null
                ? ClassAssignmentObjective.MAXIMIZE_MINIMUM_RECALL
                : classAssignmentObjective;
            this.numTopClasses = numTopClasses == null ? DEFAULT_NUM_TOP_CLASSES : numTopClasses;
            this.trainingPercent = trainingPercent == null ? 100.0 : trainingPercent;
            this.randomizeSeed = randomizeSeed == null ? Randomness.get().nextLong() : randomizeSeed;
            this.featureProcessors = featureProcessors == null ? Collections.emptyList() : Collections.unmodifiableList(featureProcessors);
            // Early stopping is true by default
            this.earlyStoppingEnabled = earlyStoppingEnabled == null ? true : earlyStoppingEnabled;
        }

        public Classification(String dependentVariable) {
            this(dependentVariable, BoostedTreeParams.builder().build(), null, null, null, null, null, null, null);
        }
```

```
        public Classification(StreamInput in) throws IOException {
            dependentVariable = in.readString();
            boostedTreeParams = new BoostedTreeParams(in);
            predictionFieldName = in.readOptionalString();
            classAssignmentObjective = in.readEnum(ClassAssignmentObjective.class);
            numTopClasses = in.readOptionalVInt();
            trainingPercent = in.readDouble();
            randomizeSeed = in.readOptionalLong();
            featureProcessors =
Collections.unmodifiableList(in.readNamedWriteableCollectionAsList(PreProcessor.class));
            earlyStoppingEnabled = in.readBoolean();
        }

        public String getDependentVariable() {
            return dependentVariable;
        }

        public BoostedTreeParams getBoostedTreeParams() {
            return boostedTreeParams;
        }

        public String getPredictionFieldName() {
            return predictionFieldName;
        }

        public ClassAssignmentObjective getClassAssignmentObjective() {
            return classAssignmentObjective;
        }

        public int getNumTopClasses() {
            return numTopClasses;
        }

        @Override
        public double getTrainingPercent() {
            return trainingPercent;
        }

        public long getRandomizeSeed() {
            return randomizeSeed;
        }

        public List<PreProcessor> getFeatureProcessors() {
            return featureProcessors;
        }

        public Boolean getEarlyStoppingEnabled() {
            return earlyStoppingEnabled;
```

```
    }

    @Override
    public String getWriteableName() {
      return NAME.getPreferredName();
    }

    @Override
    public void writeTo(StreamOutput out) throws IOException {
      out.writeString(dependentVariable);
      boostedTreeParams.writeTo(out);
      out.writeOptionalString(predictionFieldName);
      out.writeEnum(classAssignmentObjective);
      out.writeOptionalVInt(numTopClasses);
      out.writeDouble(trainingPercent);
      out.writeOptionalLong(randomizeSeed);
      out.writeNamedWriteableCollection(featureProcessors);
      out.writeBoolean(earlyStoppingEnabled);
    }

    @Override
    public XContentBuilder toXContent(XContentBuilder builder, Params params) throws
IOException {
      MlConfigVersion version = MlConfigVersion.fromString(params.param("version",
MlConfigVersion.CURRENT.toString()));

      builder.startObject();
      builder.field(DEPENDENT_VARIABLE.getPreferredName(), dependentVariable);
      boostedTreeParams.toXContent(builder, params);
      builder.field(CLASS_ASSIGNMENT_OBJECTIVE.getPreferredName(),
classAssignmentObjective);
      builder.field(NUM_TOP_CLASSES.getPreferredName(), numTopClasses);
      if (predictionFieldName != null) {
        builder.field(PREDICTION_FIELD_NAME.getPreferredName(), predictionFieldName);
      }
      builder.field(TRAINING_PERCENT.getPreferredName(), trainingPercent);
      if (version.onOrAfter(MlConfigVersion.V_7_6_0)) {
        builder.field(RANDOMIZE_SEED.getPreferredName(), randomizeSeed);
      }
      if (featureProcessors.isEmpty() == false) {
        NamedXContentObjectHelper.writeNamedObjects(builder, params, true,
FEATURE_PROCESSORS.getPreferredName(), featureProcessors);
      }
      builder.field(EARLY_STOPPING_ENABLED.getPreferredName(), earlyStoppingEnabled);
      builder.endObject();
      return builder;
    }

    @Override
```

```
    public Map<String, Object> getParams(FieldInfo fieldInfo) {
      Map<String, Object> params = new HashMap<>();
      params.put(DEPENDENT_VARIABLE.getPreferredName(), dependentVariable);
      params.putAll(boostedTreeParams.getParams());
      params.put(CLASS_ASSIGNMENT_OBJECTIVE.getPreferredName(),
classAssignmentObjective);
      params.put(NUM_TOP_CLASSES.getPreferredName(), numTopClasses);
      if (predictionFieldName != null) {
        params.put(PREDICTION_FIELD_NAME.getPreferredName(), predictionFieldName);
      }
      String                                predictionFieldType                                =
getPredictionFieldTypeParamString(getPredictionFieldType(fieldInfo.getTypes(dependentVariable)));
      if (predictionFieldType != null) {
        params.put(PREDICTION_FIELD_TYPE, predictionFieldType);
      }
      params.put(NUM_CLASSES, fieldInfo.getCardinality(dependentVariable));
      params.put(TRAINING_PERCENT.getPreferredName(), trainingPercent);
      if (featureProcessors.isEmpty() == false) {
        params.put(
          FEATURE_PROCESSORS.getPreferredName(),
          featureProcessors.stream().map(p      ->      Collections.singletonMap(p.getName(),
p)).collect(Collectors.toList())
        );
      }
      params.put(EARLY_STOPPING_ENABLED.getPreferredName(), earlyStoppingEnabled);
      params.put(RANDOMIZE_SEED.getPreferredName(), randomizeSeed);
      return params;
    }

    private      static      String      getPredictionFieldTypeParamString(PredictionFieldType
predictionFieldType) {
      if (predictionFieldType == null) {
        return null;
      }
      return switch (predictionFieldType) {
        case NUMBER ->
          // C++ process uses int64_t type, so it is safe for the dependent variable to use long
numbers.
          "int";
        case STRING -> "string";
        case BOOLEAN -> "bool";
      };
    }

    public      static      PredictionFieldType      getPredictionFieldType(Set<String>
dependentVariableTypes) {
      if (dependentVariableTypes == null) {
        return null;
      }
```

```
            if (Types.categorical().containsAll(dependentVariableTypes)) {
               return PredictionFieldType.STRING;
            }
            if (Types.bool().containsAll(dependentVariableTypes)) {
               return PredictionFieldType.BOOLEAN;
            }
            if (Types.discreteNumerical().containsAll(dependentVariableTypes)) {
               return PredictionFieldType.NUMBER;
            }
            return null;
         }

         @Override
         public boolean supportsCategoricalFields() {
            return true;
         }

         @Override
         public Set<String> getAllowedCategoricalTypes(String fieldName) {
            if (dependentVariable.equals(fieldName)) {
               return ALLOWED_DEPENDENT_VARIABLE_TYPES;
            }
            return Types.categorical();
         }

         @Override
         public List<RequiredField> getRequiredFields() {
            return Collections.singletonList(new RequiredField(dependentVariable,
ALLOWED_DEPENDENT_VARIABLE_TYPES));
         }

         @Override
         public List<FieldCardinalityConstraint> getFieldCardinalityConstraints() {
            // This restriction is due to the fact that currently the C++ backend only supports binomial
classification.
            return Collections.singletonList(FieldCardinalityConstraint.between(dependentVariable, 2,
MAX_DEPENDENT_VARIABLE_CARDINALITY));
         }

         @SuppressWarnings("unchecked")
         @Override
         public Map<String, Object> getResultMappings(String resultsFieldName,
FieldCapabilitiesResponse fieldCapabilitiesResponse) {
            Map<String, Object> additionalProperties = new HashMap<>();
            additionalProperties.put(resultsFieldName + ".is_training",
Collections.singletonMap("type", BooleanFieldMapper.CONTENT_TYPE));
            additionalProperties.put(
               resultsFieldName + ".prediction_probability",
```

```
            Collections.singletonMap("type",
NumberFieldMapper.NumberType.DOUBLE.typeName())
          );
          additionalProperties.put(
            resultsFieldName + ".prediction_score",
            Collections.singletonMap("type",
NumberFieldMapper.NumberType.DOUBLE.typeName())
          );
          additionalProperties.put(resultsFieldName + ".feature_importance",
FEATURE_IMPORTANCE_MAPPING);

          Map<String, FieldCapabilities> dependentVariableFieldCaps =
fieldCapabilitiesResponse.getField(dependentVariable);
          if (dependentVariableFieldCaps == null || dependentVariableFieldCaps.isEmpty()) {
            throw ExceptionsHelper.badRequestException("no mappings could be found for
required field [{}]", DEPENDENT_VARIABLE);
          }
          Object dependentVariableMappingType =
dependentVariableFieldCaps.values().iterator().next().getType();
          additionalProperties.put(
            resultsFieldName + "." + predictionFieldName,
            Collections.singletonMap("type", dependentVariableMappingType)
          );

          Map<String, Object> topClassesProperties = new HashMap<>();
          topClassesProperties.put("class_name", Collections.singletonMap("type",
dependentVariableMappingType));
          topClassesProperties.put("class_probability", Collections.singletonMap("type",
NumberFieldMapper.NumberType.DOUBLE.typeName()));
          topClassesProperties.put("class_score", Collections.singletonMap("type",
NumberFieldMapper.NumberType.DOUBLE.typeName()));

          Map<String, Object> topClassesMapping = new HashMap<>();
          topClassesMapping.put("type", NestedObjectMapper.CONTENT_TYPE);
          topClassesMapping.put("properties", topClassesProperties);

          additionalProperties.put(resultsFieldName + ".top_classes", topClassesMapping);
          return additionalProperties;
        }

        @Override
        public boolean supportsMissingValues() {
          return true;
        }

        @Override
        public boolean persistsState() {
          return true;
        }
```

```
        @Override
        public String getStateDocIdPrefix(String jobId) {
            return jobId + STATE_DOC_ID_INFIX;
        }

        @Override
        public List<String> getProgressPhases() {
            return PROGRESS_PHASES;
        }

        @Override
        public InferenceConfig inferenceConfig(FieldInfo fieldInfo) {
            PredictionFieldType predictionFieldType = getPredictionFieldType(fieldInfo.getTypes(dependentVariable));
            return ClassificationConfig.builder()
                .setResultsField(predictionFieldName)
                .setNumTopClasses(numTopClasses)
                .setNumTopFeatureImportanceValues(getBoostedTreeParams().getNumTopFeatureImportanceValues())
                .setPredictionFieldType(predictionFieldType)
                .build();
        }

        @Override
        public boolean supportsInference() {
            return true;
        }

        public static String extractJobIdFromStateDoc(String stateDocId) {
            int suffixIndex = stateDocId.lastIndexOf(STATE_DOC_ID_INFIX);
            return suffixIndex <= 0 ? null : stateDocId.substring(0, suffixIndex);
        }

        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() != o.getClass()) return false;
            Classification that = (Classification) o;
            return Objects.equals(dependentVariable, that.dependentVariable)
                && Objects.equals(boostedTreeParams, that.boostedTreeParams)
                && Objects.equals(predictionFieldName, that.predictionFieldName)
                && Objects.equals(classAssignmentObjective, that.classAssignmentObjective)
                && Objects.equals(numTopClasses, that.numTopClasses)
                && Objects.equals(featureProcessors, that.featureProcessors)
                && Objects.equals(earlyStoppingEnabled, that.earlyStoppingEnabled)
                && trainingPercent == that.trainingPercent
                && randomizeSeed == that.randomizeSeed;
```

```
    }

    @Override
    public int hashCode() {
        return Objects.hash(
            dependentVariable,
            boostedTreeParams,
            predictionFieldName,
            classAssignmentObjective,
            numTopClasses,
            trainingPercent,
            randomizeSeed,
            featureProcessors,
            earlyStoppingEnabled
        );
    }

    public enum ClassAssignmentObjective {
        MAXIMIZE_ACCURACY,
        MAXIMIZE_MINIMUM_RECALL;

        public static ClassAssignmentObjective fromString(String value) {
            return ClassAssignmentObjective.valueOf(value.toUpperCase(Locale.ROOT));
        }

        @Override
        public String toString() {
            return name().toLowerCase(Locale.ROOT);
        }
    }
}

package org.elasticsearch.xpack.core.ml.dataframe.analyses;

import org.elasticsearch.action.fieldcaps.FieldCapabilitiesResponse;
import org.elasticsearch.common.Randomness;
import org.elasticsearch.common.io.stream.StreamInput;
import org.elasticsearch.common.io.stream.StreamOutput;
import org.elasticsearch.core.Nullable;
import org.elasticsearch.index.mapper.BooleanFieldMapper;
import org.elasticsearch.index.mapper.KeywordFieldMapper;
import org.elasticsearch.index.mapper.NestedObjectMapper;
import org.elasticsearch.index.mapper.NumberFieldMapper;
import org.elasticsearch.xcontent.ConstructingObjectParser;
import org.elasticsearch.xcontent.ParseField;
import org.elasticsearch.xcontent.XContentBuilder;
import org.elasticsearch.xcontent.XContentParser;
import org.elasticsearch.xpack.core.ml.MlConfigVersion;
import org.elasticsearch.xpack.core.ml.inference.preprocessing.LenientlyParsedPreProcessor;
```

```
import org.elasticsearch.xpack.core.ml.inference.preprocessing.PreProcessor;
import org.elasticsearch.xpack.core.ml.inference.preprocessing.StrictlyParsedPreProcessor;
import org.elasticsearch.xpack.core.ml.inference.trainedmodel.InferenceConfig;
import org.elasticsearch.xpack.core.ml.inference.trainedmodel.RegressionConfig;
import org.elasticsearch.xpack.core.ml.utils.ExceptionsHelper;
import org.elasticsearch.xpack.core.ml.utils.NamedXContentObjectHelper;

import java.io.IOException;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Objects;
import java.util.Set;
import java.util.stream.Collectors;

import static org.elasticsearch.xcontent.ConstructingObjectParser.constructorArg;
import static org.elasticsearch.xcontent.ConstructingObjectParser.optionalConstructorArg;

public class Regression implements DataFrameAnalysis {

    public static final ParseField NAME = new ParseField("regression");

    public static final ParseField DEPENDENT_VARIABLE = new
ParseField("dependent_variable");
    public static final ParseField PREDICTION_FIELD_NAME = new
ParseField("prediction_field_name");
    public static final ParseField TRAINING_PERCENT = new ParseField("training_percent");
    public static final ParseField RANDOMIZE_SEED = new ParseField("randomize_seed");
    public static final ParseField LOSS_FUNCTION = new ParseField("loss_function");
    public static final ParseField LOSS_FUNCTION_PARAMETER = new
ParseField("loss_function_parameter");
    public static final ParseField FEATURE_PROCESSORS = new ParseField("feature_processors");
    public static final ParseField EARLY_STOPPING_ENABLED = new
ParseField("early_stopping_enabled");

    private static final String STATE_DOC_ID_INFIX = "_regression_state#";

    private static final ConstructingObjectParser<Regression, Void> LENIENT_PARSER =
createParser(true);
    private static final ConstructingObjectParser<Regression, Void> STRICT_PARSER =
createParser(false);

    @SuppressWarnings("unchecked")
    private static ConstructingObjectParser<Regression, Void> createParser(boolean lenient) {
      ConstructingObjectParser<Regression, Void> parser = new ConstructingObjectParser<>(
        NAME.getPreferredName(),
```

```
            lenient,
            a -> new Regression(
              (String) a[0],
              new BoostedTreeParams(
                (Double) a[1],
                (Double) a[2],
                (Double) a[3],
                (Integer) a[4],
                (Double) a[5],
                (Integer) a[6],
                (Double) a[7],
                (Double) a[8],
                (Double) a[9],
                (Double) a[10],
                (Double) a[11],
                (Integer) a[12]
              ),
              (String) a[13],
              (Double) a[14],
              (Long) a[15],
              (LossFunction) a[16],
              (Double) a[17],
              (List<PreProcessor>) a[18],
              (Boolean) a[19]
            )
          );
          parser.declareString(constructorArg(), DEPENDENT_VARIABLE);
          BoostedTreeParams.declareFields(parser);
          parser.declareString(optionalConstructorArg(), PREDICTION_FIELD_NAME);
          parser.declareDouble(optionalConstructorArg(), TRAINING_PERCENT);
          parser.declareLong(optionalConstructorArg(), RANDOMIZE_SEED);
          parser.declareString(optionalConstructorArg(), LossFunction::fromString,
LOSS_FUNCTION);
          parser.declareDouble(optionalConstructorArg(), LOSS_FUNCTION_PARAMETER);
          parser.declareNamedObjects(
            optionalConstructorArg(),
            (p, c, n) -> lenient
              ? p.namedObject(LenientlyParsedPreProcessor.class, n, new
PreProcessor.PreProcessorParseContext(true))
              : p.namedObject(StrictlyParsedPreProcessor.class, n, new
PreProcessor.PreProcessorParseContext(true)),
            (regression) -> {/*TODO should we throw if this is not set?*/},
            FEATURE_PROCESSORS
          );
          parser.declareBoolean(optionalConstructorArg(), EARLY_STOPPING_ENABLED);
          return parser;
        }
```

```
        public static Regression fromXContent(XContentParser parser, boolean
ignoreUnknownFields) {
            return ignoreUnknownFields ? LENIENT_PARSER.apply(parser, null) :
STRICT_PARSER.apply(parser, null);
        }

        private static final List<String> PROGRESS_PHASES = Collections.unmodifiableList(
            Arrays.asList("feature_selection", "coarse_parameter_search", "fine_tuning_parameters",
"final_training")
        );

        static final Map<String, Object> FEATURE_IMPORTANCE_MAPPING;
        static {
            Map<String, Object> properties = new HashMap<>();
            properties.put("feature_name",                    Collections.singletonMap("type",
KeywordFieldMapper.CONTENT_TYPE));
            properties.put("importance",                      Collections.singletonMap("type",
NumberFieldMapper.NumberType.DOUBLE.typeName()));

            Map<String, Object> mapping = new HashMap<>();
            mapping.put("dynamic", false);
            mapping.put("type", NestedObjectMapper.CONTENT_TYPE);
            mapping.put("properties", properties);

            FEATURE_IMPORTANCE_MAPPING = Collections.unmodifiableMap(mapping);
        }

        private final String dependentVariable;
        private final BoostedTreeParams boostedTreeParams;
        private final String predictionFieldName;
        private final double trainingPercent;
        private final long randomizeSeed;
        private final LossFunction lossFunction;
        private final Double lossFunctionParameter;
        private final List<PreProcessor> featureProcessors;
        private final boolean earlyStoppingEnabled;

        public Regression(
            String dependentVariable,
            BoostedTreeParams boostedTreeParams,
            @Nullable String predictionFieldName,
            @Nullable Double trainingPercent,
            @Nullable Long randomizeSeed,
            @Nullable LossFunction lossFunction,
            @Nullable Double lossFunctionParameter,
            @Nullable List<PreProcessor> featureProcessors,
            @Nullable Boolean earlyStoppingEnabled
        ) {
            if (trainingPercent != null && (trainingPercent <= 0.0 || trainingPercent > 100.0)) {
```

```
            throw ExceptionsHelper.badRequestException("[{}] must be a positive double in (0, 100]", TRAINING_PERCENT.getPreferredName());
        }
        this.dependentVariable = ExceptionsHelper.requireNonNull(dependentVariable, DEPENDENT_VARIABLE);
        this.boostedTreeParams = ExceptionsHelper.requireNonNull(boostedTreeParams, BoostedTreeParams.NAME);
        this.predictionFieldName = predictionFieldName == null ? dependentVariable + "_prediction" : predictionFieldName;
        this.trainingPercent = trainingPercent == null ? 100.0 : trainingPercent;
        this.randomizeSeed = randomizeSeed == null ? Randomness.get().nextLong() : randomizeSeed;
        // Prior to introducing the loss function setting only MSE was implemented
        this.lossFunction = lossFunction == null ? LossFunction.MSE : lossFunction;
        if (lossFunctionParameter != null && lossFunctionParameter <= 0.0) {
            throw ExceptionsHelper.badRequestException("[{}] must be a positive double", LOSS_FUNCTION_PARAMETER.getPreferredName());
        }
        this.lossFunctionParameter = lossFunctionParameter;
        this.featureProcessors = featureProcessors == null ? Collections.emptyList() : Collections.unmodifiableList(featureProcessors);
        // Early stopping is true by default
        this.earlyStoppingEnabled = earlyStoppingEnabled == null ? true : earlyStoppingEnabled;
    }

    public Regression(String dependentVariable) {
        this(dependentVariable, BoostedTreeParams.builder().build(), null, null, null, null, null, null, null);
    }

    public Regression(StreamInput in) throws IOException {
        dependentVariable = in.readString();
        boostedTreeParams = new BoostedTreeParams(in);
        predictionFieldName = in.readOptionalString();
        trainingPercent = in.readDouble();
        randomizeSeed = in.readOptionalLong();
        lossFunction = in.readEnum(LossFunction.class);
        lossFunctionParameter = in.readOptionalDouble();
        featureProcessors = Collections.unmodifiableList(in.readNamedWriteableCollectionAsList(PreProcessor.class));
        earlyStoppingEnabled = in.readBoolean();
    }

    public String getDependentVariable() {
        return dependentVariable;
    }

    public BoostedTreeParams getBoostedTreeParams() {
        return boostedTreeParams;
```

```
}

public String getPredictionFieldName() {
   return predictionFieldName;
}

@Override
public double getTrainingPercent() {
   return trainingPercent;
}

public long getRandomizeSeed() {
   return randomizeSeed;
}

public LossFunction getLossFunction() {
   return lossFunction;
}

public Double getLossFunctionParameter() {
   return lossFunctionParameter;
}

public List<PreProcessor> getFeatureProcessors() {
   return featureProcessors;
}

public Boolean getEarlyStoppingEnabled() {
   return earlyStoppingEnabled;
}

@Override
public String getWriteableName() {
   return NAME.getPreferredName();
}

@Override
public void writeTo(StreamOutput out) throws IOException {
   out.writeString(dependentVariable);
   boostedTreeParams.writeTo(out);
   out.writeOptionalString(predictionFieldName);
   out.writeDouble(trainingPercent);
   out.writeOptionalLong(randomizeSeed);
   out.writeEnum(lossFunction);
   out.writeOptionalDouble(lossFunctionParameter);
   out.writeNamedWriteableCollection(featureProcessors);
   out.writeBoolean(earlyStoppingEnabled);
}
```

```
        @Override
        public XContentBuilder toXContent(XContentBuilder builder, Params params) throws
IOException {
            MlConfigVersion version = MlConfigVersion.fromString(params.param("version",
MlConfigVersion.CURRENT.toString()));

            builder.startObject();
            builder.field(DEPENDENT_VARIABLE.getPreferredName(), dependentVariable);
            boostedTreeParams.toXContent(builder, params);
            if (predictionFieldName != null) {
              builder.field(PREDICTION_FIELD_NAME.getPreferredName(), predictionFieldName);
            }
            builder.field(TRAINING_PERCENT.getPreferredName(), trainingPercent);
            if (version.onOrAfter(MlConfigVersion.V_7_6_0)) {
              builder.field(RANDOMIZE_SEED.getPreferredName(), randomizeSeed);
            }
            builder.field(LOSS_FUNCTION.getPreferredName(), lossFunction);
            if (lossFunctionParameter != null) {
              builder.field(LOSS_FUNCTION_PARAMETER.getPreferredName(),
lossFunctionParameter);
            }
            if (featureProcessors.isEmpty() == false) {
              NamedXContentObjectHelper.writeNamedObjects(builder, params, true,
FEATURE_PROCESSORS.getPreferredName(), featureProcessors);
            }
            builder.field(EARLY_STOPPING_ENABLED.getPreferredName(), earlyStoppingEnabled);
            builder.endObject();
            return builder;
          }

        @Override
        public Map<String, Object> getParams(FieldInfo fieldInfo) {
          Map<String, Object> params = new HashMap<>();
          params.put(DEPENDENT_VARIABLE.getPreferredName(), dependentVariable);
          params.putAll(boostedTreeParams.getParams());
          if (predictionFieldName != null) {
            params.put(PREDICTION_FIELD_NAME.getPreferredName(), predictionFieldName);
          }
          params.put(TRAINING_PERCENT.getPreferredName(), trainingPercent);
          params.put(LOSS_FUNCTION.getPreferredName(), lossFunction.toString());
          if (lossFunctionParameter != null) {
            params.put(LOSS_FUNCTION_PARAMETER.getPreferredName(),
lossFunctionParameter);
          }
          if (featureProcessors.isEmpty() == false) {
            params.put(
              FEATURE_PROCESSORS.getPreferredName(),
              featureProcessors.stream().map(p -> Collections.singletonMap(p.getName(),
p)).collect(Collectors.toList())
```

```
            );
        }
        params.put(EARLY_STOPPING_ENABLED.getPreferredName(), earlyStoppingEnabled);
        params.put(RANDOMIZE_SEED.getPreferredName(), randomizeSeed);
        return params;
    }

    @Override
    public boolean supportsCategoricalFields() {
        return true;
    }

    @Override
    public Set<String> getAllowedCategoricalTypes(String fieldName) {
        return Types.categorical();
    }

    @Override
    public List<RequiredField> getRequiredFields() {
        return Collections.singletonList(new RequiredField(dependentVariable,
Types.numerical()));
    }

    @Override
    public List<FieldCardinalityConstraint> getFieldCardinalityConstraints() {
        return Collections.emptyList();
    }

    @Override
    public Map<String, Object> getResultMappings(String resultsFieldName,
FieldCapabilitiesResponse fieldCapabilitiesResponse) {
        Map<String, Object> additionalProperties = new HashMap<>();
        additionalProperties.put(resultsFieldName + ".is_training",
Collections.singletonMap("type", BooleanFieldMapper.CONTENT_TYPE));
        additionalProperties.put(resultsFieldName + ".feature_importance",
FEATURE_IMPORTANCE_MAPPING);
        // Prediction field should be always mapped as "double" rather than "float" in order to
increase precision in case of
        // high (over 10M) values of dependent variable.
        additionalProperties.put(
            resultsFieldName + "." + predictionFieldName,
            Collections.singletonMap("type",
NumberFieldMapper.NumberType.DOUBLE.typeName())
        );
        return additionalProperties;
    }

    @Override
    public boolean supportsMissingValues() {
```

```
        return true;
    }

    @Override
    public boolean persistsState() {
        return true;
    }

    @Override
    public String getStateDocIdPrefix(String jobId) {
        return jobId + STATE_DOC_ID_INFIX;
    }

    @Override
    public List<String> getProgressPhases() {
        return PROGRESS_PHASES;
    }

    @Override
    public InferenceConfig inferenceConfig(FieldInfo fieldInfo) {
        return RegressionConfig.builder()
            .setResultsField(predictionFieldName)

.setNumTopFeatureImportanceValues(getBoostedTreeParams().getNumTopFeatureImportanceValue
s())
            .build();
    }

    @Override
    public boolean supportsInference() {
        return true;
    }

    public static String extractJobIdFromStateDoc(String stateDocId) {
        int suffixIndex = stateDocId.lastIndexOf(STATE_DOC_ID_INFIX);
        return suffixIndex <= 0 ? null : stateDocId.substring(0, suffixIndex);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Regression that = (Regression) o;
        return Objects.equals(dependentVariable, that.dependentVariable)
            && Objects.equals(boostedTreeParams, that.boostedTreeParams)
            && Objects.equals(predictionFieldName, that.predictionFieldName)
            && trainingPercent == that.trainingPercent
            && randomizeSeed == that.randomizeSeed
            && lossFunction == that.lossFunction
```

```
            && Objects.equals(featureProcessors, that.featureProcessors)
            && Objects.equals(lossFunctionParameter, that.lossFunctionParameter)
            && Objects.equals(earlyStoppingEnabled, that.earlyStoppingEnabled);
    }

    @Override
    public int hashCode() {
        return Objects.hash(
            dependentVariable,
            boostedTreeParams,
            predictionFieldName,
            trainingPercent,
            randomizeSeed,
            lossFunction,
            lossFunctionParameter,
            featureProcessors,
            earlyStoppingEnabled
        );
    }

    public enum LossFunction {
        MSE,
        MSLE,
        HUBER;

        private static LossFunction fromString(String value) {
            return LossFunction.valueOf(value.toUpperCase(Locale.ROOT));
        }

        @Override
        public String toString() {
            return name().toLowerCase(Locale.ROOT);
        }
    }
}

package org.elasticsearch.xpack.core.ml.dataframe.analyses;

import org.elasticsearch.action.fieldcaps.FieldCapabilitiesResponse;
import org.elasticsearch.common.io.stream.StreamInput;
import org.elasticsearch.common.io.stream.StreamOutput;
import org.elasticsearch.core.Nullable;
import org.elasticsearch.index.mapper.KeywordFieldMapper;
import org.elasticsearch.index.mapper.NestedObjectMapper;
import org.elasticsearch.index.mapper.NumberFieldMapper;
import org.elasticsearch.xcontent.ObjectParser;
import org.elasticsearch.xcontent.ParseField;
import org.elasticsearch.xcontent.XContentBuilder;
import org.elasticsearch.xcontent.XContentParser;
```

```
import org.elasticsearch.xpack.core.ml.inference.trainedmodel.InferenceConfig;
import org.elasticsearch.xpack.core.ml.utils.ExceptionsHelper;

import java.io.IOException;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Objects;
import java.util.Set;

public class OutlierDetection implements DataFrameAnalysis {

    public static final ParseField NAME = new ParseField("outlier_detection");

    public static final ParseField N_NEIGHBORS = new ParseField("n_neighbors");
    public static final ParseField METHOD = new ParseField("method");
    public static final ParseField FEATURE_INFLUENCE_THRESHOLD = new
ParseField("feature_influence_threshold");
    public static final ParseField COMPUTE_FEATURE_INFLUENCE = new
ParseField("compute_feature_influence");
    public static final ParseField OUTLIER_FRACTION = new ParseField("outlier_fraction");
    public static final ParseField STANDARDIZATION_ENABLED = new
ParseField("standardization_enabled");

    private static final ObjectParser<Builder, Void> LENIENT_PARSER = createParser(true);
    private static final ObjectParser<Builder, Void> STRICT_PARSER = createParser(false);

    private static ObjectParser<Builder, Void> createParser(boolean lenient) {
        ObjectParser<Builder, Void> parser = new ObjectParser<>(NAME.getPreferredName(),
lenient, Builder::new);
        parser.declareInt(Builder::setNNeighbors, N_NEIGHBORS);
        parser.declareString(Builder::setMethod, Method::fromString, METHOD);
        parser.declareDouble(Builder::setFeatureInfluenceThreshold,
FEATURE_INFLUENCE_THRESHOLD);
        parser.declareBoolean(Builder::setComputeFeatureInfluence,
COMPUTE_FEATURE_INFLUENCE);
        parser.declareDouble(Builder::setOutlierFraction, OUTLIER_FRACTION);
        parser.declareBoolean(Builder::setStandardizationEnabled,
STANDARDIZATION_ENABLED);
        return parser;
    }

    public static OutlierDetection fromXContent(XContentParser parser, boolean
ignoreUnknownFields) {
        return ignoreUnknownFields ? LENIENT_PARSER.apply(parser, null).build() :
STRICT_PARSER.apply(parser, null).build();
    }
```

```
        private static final List<String> PROGRESS_PHASES = Collections.singletonList("computing_outliers");

        static final Map<String, Object> FEATURE_INFLUENCE_MAPPING;
        static {
          Map<String, Object> properties = new HashMap<>();
          properties.put("feature_name", Collections.singletonMap("type", KeywordFieldMapper.CONTENT_TYPE));
          properties.put("influence", Collections.singletonMap("type", NumberFieldMapper.NumberType.DOUBLE.typeName()));

          Map<String, Object> mapping = new HashMap<>();
          mapping.put("dynamic", false);
          mapping.put("type", NestedObjectMapper.CONTENT_TYPE);
          mapping.put("properties", properties);

          FEATURE_INFLUENCE_MAPPING = Collections.unmodifiableMap(mapping);
        }

        /**
         * The number of neighbors. Leave unspecified for dynamic detection.
         */
        @Nullable
        private final Integer nNeighbors;

        /**
         * The method. Leave unspecified for a dynamic mixture of methods.
         */
        @Nullable
        private final Method method;

        /**
         * The min outlier score required to calculate feature influence. Defaults to 0.1.
         */
        @Nullable
        private final Double featureInfluenceThreshold;

        /**
         * Whether to compute feature influence or not. Defaults to true.
         */
        private final boolean computeFeatureInfluence;

        /**
         * The proportion of data assumed to be outlying prior to outlier detection. Defaults to 0.05.
         */
        private final double outlierFraction;

        /**
```

```
         * Whether to perform standardization.
         */
        private final boolean standardizationEnabled;

        private OutlierDetection(
            Integer nNeighbors,
            Method method,
            Double featureInfluenceThreshold,
            boolean computeFeatureInfluence,
            double outlierFraction,
            boolean standardizationEnabled
        ) {
            if (nNeighbors != null && nNeighbors <= 0) {
                throw ExceptionsHelper.badRequestException("[{}] must be a positive integer",
N_NEIGHBORS.getPreferredName());
            }

            if (featureInfluenceThreshold != null && (featureInfluenceThreshold < 0.0 ||
featureInfluenceThreshold > 1.0)) {
                throw ExceptionsHelper.badRequestException("[{}] must be in [0, 1]",
FEATURE_INFLUENCE_THRESHOLD.getPreferredName());
            }

            if (outlierFraction < 0.0 || outlierFraction > 1.0) {
                throw ExceptionsHelper.badRequestException("[{}] must be in [0, 1]",
OUTLIER_FRACTION.getPreferredName());
            }

            this.nNeighbors = nNeighbors;
            this.method = method;
            this.featureInfluenceThreshold = featureInfluenceThreshold;
            this.computeFeatureInfluence = computeFeatureInfluence;
            this.outlierFraction = outlierFraction;
            this.standardizationEnabled = standardizationEnabled;
        }

        public OutlierDetection(StreamInput in) throws IOException {
            nNeighbors = in.readOptionalVInt();
            method = in.readBoolean() ? in.readEnum(Method.class) : null;
            featureInfluenceThreshold = in.readOptionalDouble();
            computeFeatureInfluence = in.readBoolean();
            outlierFraction = in.readDouble();
            standardizationEnabled = in.readBoolean();
        }

        @Override
        public String getWriteableName() {
            return NAME.getPreferredName();
        }
```

```
        @Override
        public void writeTo(StreamOutput out) throws IOException {
            out.writeOptionalVInt(nNeighbors);

            if (method != null) {
                out.writeBoolean(true);
                out.writeEnum(method);
            } else {
                out.writeBoolean(false);
            }

            out.writeOptionalDouble(featureInfluenceThreshold);

            out.writeBoolean(computeFeatureInfluence);
            out.writeDouble(outlierFraction);
            out.writeBoolean(standardizationEnabled);
        }

        @Override
        public XContentBuilder toXContent(XContentBuilder builder, Params params) throws
IOException {
            builder.startObject();
            if (nNeighbors != null) {
                builder.field(N_NEIGHBORS.getPreferredName(), nNeighbors);
            }
            if (method != null) {
                builder.field(METHOD.getPreferredName(), method);
            }
            if (featureInfluenceThreshold != null) {
                builder.field(FEATURE_INFLUENCE_THRESHOLD.getPreferredName(),
featureInfluenceThreshold);
            }
            builder.field(COMPUTE_FEATURE_INFLUENCE.getPreferredName(),
computeFeatureInfluence);
            builder.field(OUTLIER_FRACTION.getPreferredName(), outlierFraction);
            builder.field(STANDARDIZATION_ENABLED.getPreferredName(), standardizationEnabled);
            builder.endObject();
            return builder;
        }

        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() != o.getClass()) return false;
            OutlierDetection that = (OutlierDetection) o;
            return Objects.equals(nNeighbors, that.nNeighbors)
                && Objects.equals(method, that.method)
                && Objects.equals(featureInfluenceThreshold, that.featureInfluenceThreshold)
```

```
            && computeFeatureInfluence == that.computeFeatureInfluence
            && outlierFraction == that.outlierFraction
            && standardizationEnabled == that.standardizationEnabled;
        }

        @Override
        public int hashCode() {
          return Objects.hash(
            nNeighbors,
            method,
            featureInfluenceThreshold,
            computeFeatureInfluence,
            outlierFraction,
            standardizationEnabled
          );
        }

        @Override
        public Map<String, Object> getParams(FieldInfo fieldInfo) {
          Map<String, Object> params = new HashMap<>();
          if (nNeighbors != null) {
            params.put(N_NEIGHBORS.getPreferredName(), nNeighbors);
          }
          if (method != null) {
            params.put(METHOD.getPreferredName(), method);
          }
          if (featureInfluenceThreshold != null) {
            params.put(FEATURE_INFLUENCE_THRESHOLD.getPreferredName(),
featureInfluenceThreshold);
          }
          params.put(COMPUTE_FEATURE_INFLUENCE.getPreferredName(),
computeFeatureInfluence);
          params.put(OUTLIER_FRACTION.getPreferredName(), outlierFraction);
          params.put(STANDARDIZATION_ENABLED.getPreferredName(), standardizationEnabled);
          return params;
        }

        @Override
        public boolean supportsCategoricalFields() {
          return false;
        }

        @Override
        public Set<String> getAllowedCategoricalTypes(String fieldName) {
          return Collections.emptySet();
        }

        @Override
        public List<RequiredField> getRequiredFields() {
```

```
            return Collections.emptyList();
        }

        @Override
        public List<FieldCardinalityConstraint> getFieldCardinalityConstraints() {
            return Collections.emptyList();
        }

        @Override
        public Map<String, Object> getResultMappings(String resultsFieldName,
FieldCapabilitiesResponse fieldCapabilitiesResponse) {
            Map<String, Object> additionalProperties = new HashMap<>();
            additionalProperties.put(
                resultsFieldName + ".outlier_score",
                Collections.singletonMap("type",
NumberFieldMapper.NumberType.DOUBLE.typeName())
            );
            additionalProperties.put(resultsFieldName + ".feature_influence",
FEATURE_INFLUENCE_MAPPING);
            return additionalProperties;
        }

        @Override
        public boolean supportsMissingValues() {
            return false;
        }

        @Override
        public boolean persistsState() {
            return false;
        }

        @Override
        public String getStateDocIdPrefix(String jobId) {
            throw new UnsupportedOperationException("Outlier detection does not support state");
        }

        @Override
        public List<String> getProgressPhases() {
            return PROGRESS_PHASES;
        }

        @Override
        public InferenceConfig inferenceConfig(FieldInfo fieldInfo) {
            return null;
        }

        @Override
        public boolean supportsInference() {
```

```
    return false;
  }

  public enum Method {
    LOF,
    LDOF,
    DISTANCE_KTH_NN,
    DISTANCE_KNN;

    public static Method fromString(String value) {
      return Method.valueOf(value.toUpperCase(Locale.ROOT));
    }

    @Override
    public String toString() {
      return name().toLowerCase(Locale.ROOT);
    }
  }

  public static class Builder {

    private Integer nNeighbors;
    private Method method;
    private Double featureInfluenceThreshold;
    private boolean computeFeatureInfluence = true;
    private double outlierFraction = 0.05;
    private boolean standardizationEnabled = true;

    public Builder() {}

    public Builder(OutlierDetection other) {
      this.nNeighbors = other.nNeighbors;
      this.method = other.method;
      this.featureInfluenceThreshold = other.featureInfluenceThreshold;
      this.computeFeatureInfluence = other.computeFeatureInfluence;
      this.outlierFraction = other.outlierFraction;
      this.standardizationEnabled = other.standardizationEnabled;
    }

    public Builder setNNeighbors(Integer nNeighbors) {
      this.nNeighbors = nNeighbors;
      return this;
    }

    public Builder setMethod(Method method) {
      this.method = method;
      return this;
    }
```

```
        public Builder setFeatureInfluenceThreshold(Double featureInfluenceThreshold) {
            this.featureInfluenceThreshold = featureInfluenceThreshold;
            return this;
        }

        public Builder setComputeFeatureInfluence(boolean computeFeatureInfluence) {
            this.computeFeatureInfluence = computeFeatureInfluence;
            return this;
        }

        public Builder setOutlierFraction(double outlierFraction) {
            this.outlierFraction = outlierFraction;
            return this;
        }

        public Builder setStandardizationEnabled(boolean standardizationEnabled) {
            this.standardizationEnabled = standardizationEnabled;
            return this;
        }

        public OutlierDetection build() {
            return new OutlierDetection(
                nNeighbors,
                method,
                featureInfluenceThreshold,
                computeFeatureInfluence,
                outlierFraction,
                standardizationEnabled
            );
        }
    }
}
```

Додаток В. Ілюстративний матеріал

РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ТЕЛЕМЕТРИЧНОГО МОНІТОРИНГУ АНОМАЛЬНИХ АКТИВНОСТЕЙ ПРОГРАМНИХ КОМПОНЕНТІВ ДЛЯ ЗАХИСТУ ІНФОРМАЦІЙНИХ СИСТЕМ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ ШІ ТА ELK STACK

Розробила студентка групи 1КІТС-21б Присяжна Д.О.

Науковий керівник к.т.н, доц., доцент каф. МБІС Карпінець В. В.

МЕТА РОБОТИ

Підвищення рівня захищеності інформаційних систем у реальному часі шляхом створення програмного засобу телеметричного моніторингу, що забезпечує точне та своєчасне виявлення аномальних активностей програмних компонентів на основі алгоритмів штучного інтелекту та стеку ELK. Реалізація запропонованого підходу дозволяє досягти вищої швидкості обробки даних, зменшення кількості хибнопозитивних спрацювань, покращення достовірності виявлення загроз та стійкості системи до змін у поведінці користувачів і середовища, порівняно з наявними аналогами.

АКТУАЛЬНІСТЬ РОБОТИ

Сучасний стан технологій і постійне зростання кількості кібератак ставлять перед фахівцями з кібербезпеки складне завдання — своєчасне виявлення та запобігання загрозам. Аномальні активності можуть сигналізувати про зловмисні дії, тому їх моніторинг є важливою складовою захисту інформаційних систем. Одним із ефективних підходів є використання телеметричних даних у поєднанні з методами штучного інтелекту, що дозволяє автоматизувати процес виявлення загроз.

Актуальність дослідження полягає в необхідності розробки рішень, здатних забезпечити своєчасне виявлення загроз в умовах постійного ускладнення атак. Проаналізувавши сучасні методи моніторингу, архітектурні підходи та засоби обробки даних, у роботі реалізовано прототип системи, що відповідає вимогам до захисту інформаційних систем

ПРЕДМЕТ ТА ОБ'ЄКТ ДОСЛІДЖЕННЯ

Предметом дослідження є захист інформаційних систем в реальному часі, на основі аналізу телеметричних даних алгоритмами ШІ.

Об'єктом дослідження є програмні засоби збору телеметричних даних, засоби обмеження доступу в інформаційних системах, методи ШІ для виявлення аномальних активностей.

ЗАДАЧІ

1. Розробити архітектуру програмного засобу, що забезпечить ефективний збір та обробку телеметричних даних інформаційної системи.

2. Визначити ключові параметри моніторингу для виявлення різних типів аномальних активностей.

3. Реалізувати інтеграцію технологій штучного інтелекту та машинного навчання для автоматичного аналізу телеметричних даних та виявлення аномалій.

4. Візуалізація результатів моніторингу, який забезпечить наочне представлення виявлених аномалій та спростить аналіз інцидентів безпеки.

5. Забезпечити рішення для обробки телеметричних даних з використанням ELK stack.

6. Реалізувати механізм оповіщення та реагування на виявлені аномальні активності, який дозволить своєчасно вживати заходів для запобігання та нейтралізації загроз.

7. Тестування програмного засобу для оцінки його ефективності в реальних умовах експлуатації.

ТЕЛЕМЕТРІЯ

Телеметрія — це автоматизований процес збору та передачі даних про стан інформаційної системи в реальному часі. У сфері кібербезпеки вона дозволяє виявляти аномалії, загрози (DDoS, ботнети, витоки даних) та забезпечує моніторинг серверів, мережі, додатків, хмари й користувачів. На відміну від статичних методів, телеметричний моніторинг забезпечує динамічну оцінку безпеки та дозволяє своєчасно реагувати на відхилення в системному середовищі.

SIEM РІШЕННЯ

SIEM-рішення є ключовим інструментом для моніторингу безпеки в режимі реального часу. Воно поєднують збір, аналіз та кореляцію подій, виявляючи аномалії на основі телеметричних даних. На відміну від хаотичного збору логів, SIEM забезпечує структурований підхід: централізовано збирає логи, мережеву активність, події безпеки, аналізує ризики та генерує сповіщення. Це дозволяє центрам безпеки швидко реагувати на загрози та розслідувати інциденти, формуючи повну картину стану інформаційної інфраструктури.

Ґрунтуючись на порівняльному аналізі характеристик, можна прийти до висновку, що запропонований у даній роботі розроблений програмний засіб телеметричного моніторингу з використанням технологій ШІ та ELK Stack є актуальною розробкою, що покриватиме недоліки існуючих засобів.

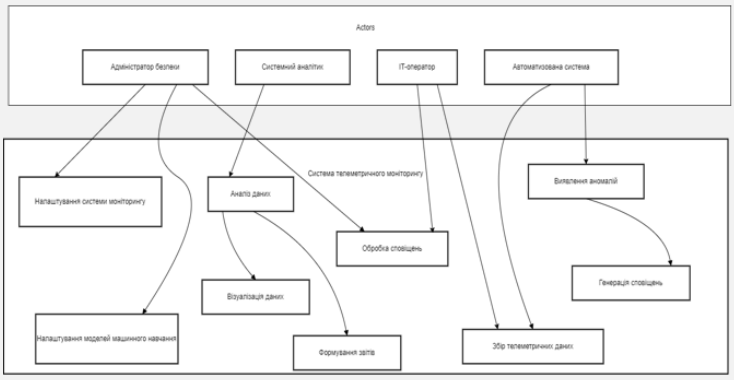
ПОРІВНЯННЯ ІСНУЮЧИХ ЗАСОБІВ ЗАХИСТУ НА ОСНОВІ МОНІТОРИНГУ АКТИВНОСТІ КОРИСТУВАЧІВ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ ШІ

Критерій	Darktrace	Vectra AI	CrowdStrike Falcon	Розроблений програмний засіб
Хмарне розгортання	+/-	+	+	+
Платна ліцензія/підписка	+	+	+	-
Часті хибнопозитивні спрацювання	+	-	+	-
Зручний інтерфейс або легкість налаштування	-	+/-	+	+
Інтеграція з сторонніми сервісами	+	+	+	+

СХЕМА РУХУ ДАНИХ

Для планування проектування програмного засобу, було створено схему руху даних, що включає послідовність таких етапів:

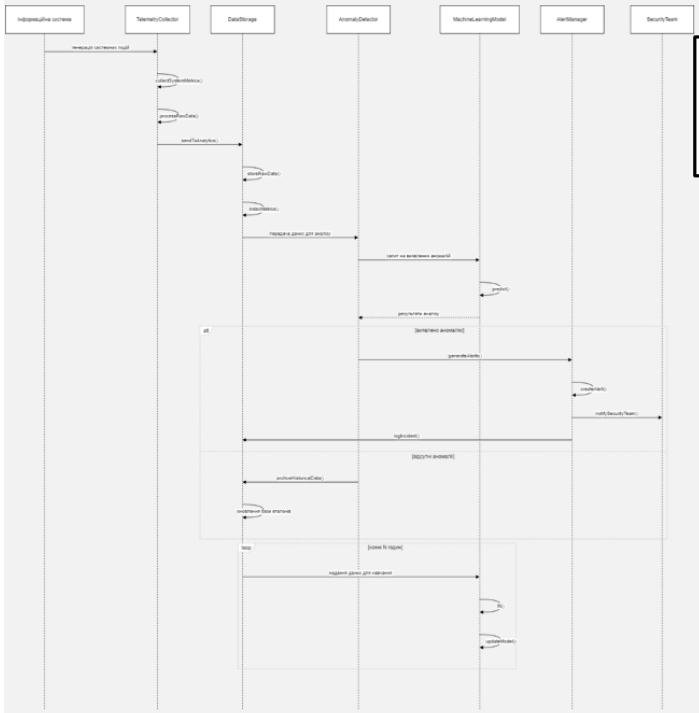
1. Збір первинних метрик з сховища даних
2. Попередня фільтрація даних
3. Потокова обробка даних в розподілених чергах
4. Аналіз даних на аномалії технологіями ШІ
5. Формування попереджень та реакцій
6. Перенаправлення даних відповідно до налаштованого блекліста



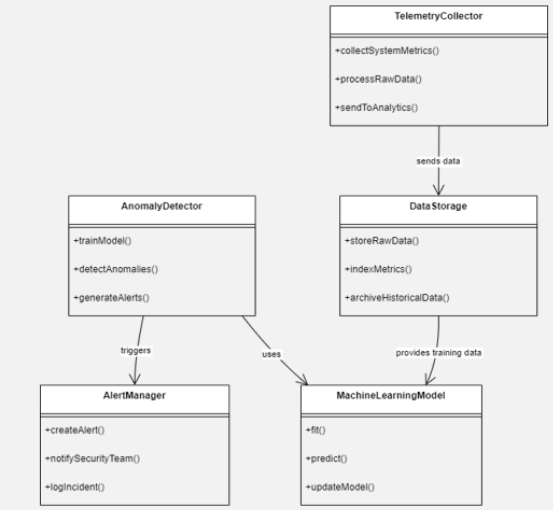
ДІАГРАМА ПРЕЦЕДЕНТІВ

Діаграма прецедентів – це графічне зображення взаємодії акторів (користувачів або інших систем) із програмною системою через прецеденти (сценарії використання).Вона показує хто взаємодіє із системою (актори), що саме робить система для актора (прецеденти) та як вони пов’язані між собою. Діаграма допомагає зрозуміти, яку поведінку очікують від системи користувачі та інші зовнішні сутності.

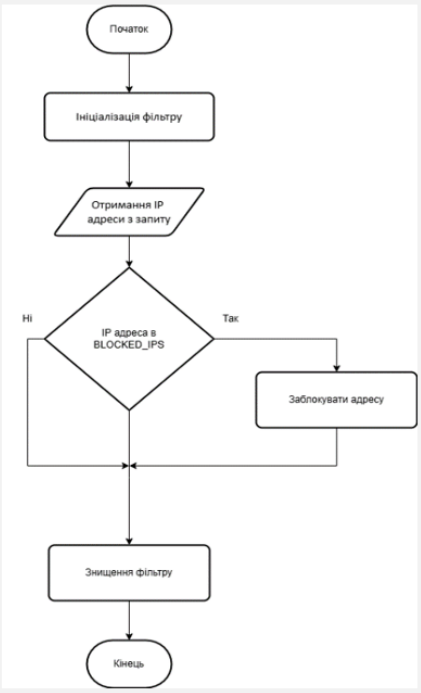
Діаграма прецедентів описує взаємодію користувача з системою з погляду зовнішньої поведінки, але не показує, як реалізована система зсередини. Вона не дає уявлення про структуру модулів, компоненти ШІ чи потоки телеметричних даних. Також планується автоматизувати більшість процесів і не залучати акторів.



ДІАГРАМИ ПЗ

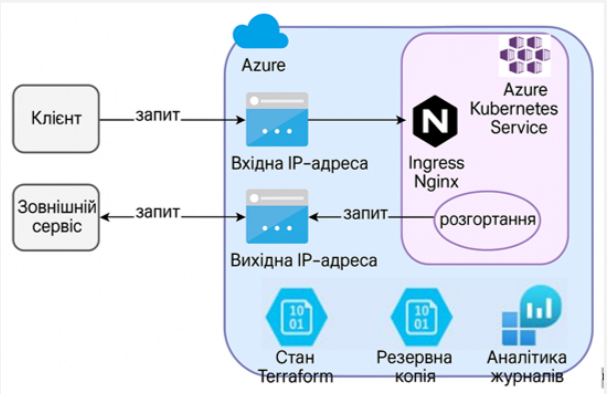


ДІАГРАМИ ПЗ



ІНФРАСТРУКТУРА

Завдяки співпраці з Центром інформаційних технологій і захисту інформації (ЦІТЗІ), студенти мають безкоштовний доступ до Office 365 for Education, що знижує фінансові бар'єри для використання хмарних сервісів. У зв'язку з цим було обрано середовище Microsoft Azure — гнучку хмарну платформу, яка підтримує розгортання контейнеризованих рішень та забезпечує масштабованість системи. Для цього використано Azure Kubernetes Service (AKS), що спрощує управління інфраструктурою через Terraform.

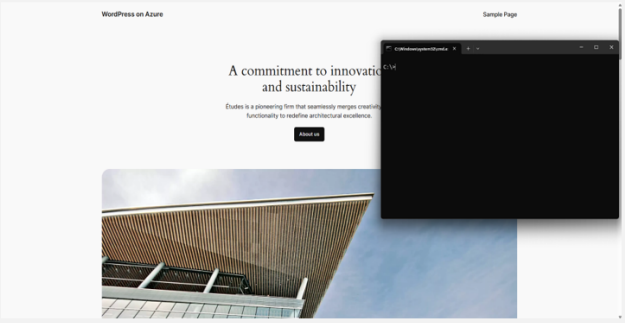


ТЕСТУВАННЯ

Для тестування запропоновано сценарій:

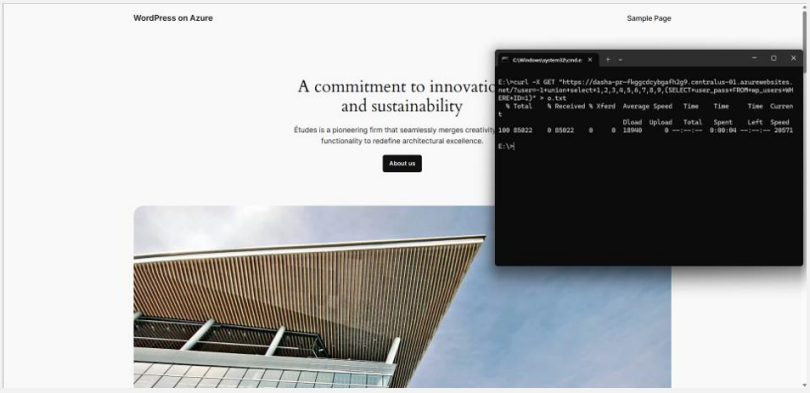
1. Користувач перевіряє з'єднання та доступність веб ресурсу.
2. Користувач відправляє запит з аномальною активністю, наприклад з вмістом SQL ін'єкції.
3. Система виявляє аномальну активність та блокує користувача по IP.
4. Користувач спостерігає проблеми з'єднання, веб ресурс не доступний.

Отже, маємо веб ресурс «https://dasha-pr-fkggcdcybgafh2g9.centralus-01.azurewebsites.net», та перевіряємо з'єднання з ним.



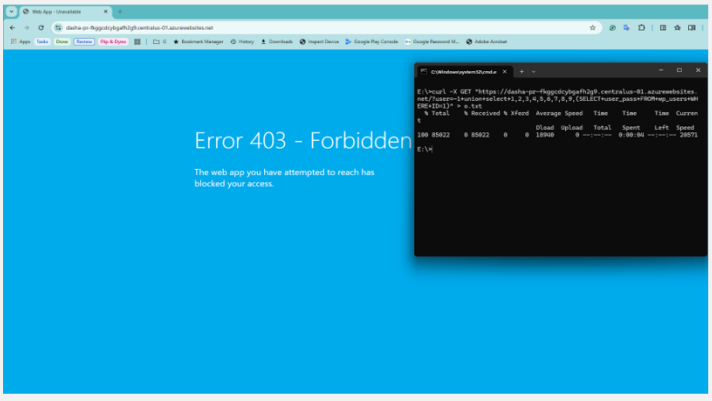
ТЕСТУВАННЯ

Використовуючи утиліту curl, відправимо аномальний запит який містить SQL ін'єкцію



ТЕСТУВАННЯ

Після відправки аномального запиту, можна спостерігати що доступ до ресурсу заблоковано



Отже, в результаті тестування ми переконались в коректності роботи програмного засобу. Доступ до ресурсу було заблоковано, час спрацювання менше 150мс що є чудовим показником. Додатково варто зауважити що система здатна до самовдосконалення, за рахунок навчання. Тобто, можна очікувати на меншу кількість хибних спрацювань, та обробка виключень будь-якого типу нових, невідомих (Zero-Day) атак.

ВИСНОВКИ

У межах бакалаврської роботи реалізовано програмний засіб телеметричного моніторингу аномальної активності з використанням ШІ та ELK Stack. Спроєктована архітектура охоплює повний цикл роботи з телеметрією — від збору до реагування. Впроваджено модуль машинного навчання, що забезпечує кластеризацію та аналіз часових рядів. Візуалізація здійснюється через Kibana, зберігання — в Elasticsearch, де час відповіді не перевищує 150 мс. Механізм сповіщень реагує за 3 с. Рішення придатне до впровадження в мережах підприємств. Подальші дослідження доцільно спрямувати на вдосконалення системи обробки інцидентів за допомогою систем обробки черг на зразок Apache Kafka та Rabbit MQ.

Додаток Г. Протокол перевірки на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка програмного засобу телеметричного моніторингу аномальних активностей програмних компонентів для захисту інформаційних систем з використанням технології ШІ та ELK stack,

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. 1КІТС-21б

Керівник доцент Карпінець В.В. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	94,04%
Загальна схожість	5,97%

Аналіз звіту подібності (відмітити потрібне)

- ☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Присяжна Д.О.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

☐ **Допустити до захисту**

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)