НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Методи виявлення аномалій трафіку в SDN мережах

Виконав : студент __4__ курсу, групи ІО-16
(шифр групи)

_____Козленко Євгеній Юрійович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент каф. ОТ Коренко Д. В._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) асистент д. ф. Міщенко Л. Д. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент __ ст. викл. каф. СПСКС Радченко К. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Козленка Євгенія Юрійовича

1. Тема проєкту *Методи виявлення аномалій трафіку в SDN мережах*
керівник проєкту Коренко Дмитро Володимирович, асистент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 № 1705-с
2. Термін здачі студентом закінченого проєкту 6 червня 2025

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд методів виявлення аномалій трафіку в SDN мережах.

Розділ 2. Огляд алгоритмів та технологій для розробки системи.

Розділ 3. Деталі розробки системи.

Розділ 4. Дослідження та аналіз розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Міщенко Л. Д.		

7. Дата видачі завдання _10 лотого 2025 р._____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	10.02.2025 – 14.02.2025	
2.	*Вивчення та аналіз завдання*	15.02.2025 – 04.04.2025	
3.	*Розробка архітектури та загальної структури системи*	04.04.2025 – 25.04.2025	
4.	*Розробка структур окремих підсистем*	26.04.2025 – 14.05.2025	
5.	*Програмна реалізація системи*	15.05.2025 - 21.05.2025	
6.	*Оформлення пояснювальної записки*	22.05.2025-05.06.2025	
7.	*Захист програмного продукту*	06.06.2025	
8.	*Передзахист*	13.06.2025	
9.	*Захист*	20.06.2025	

Студент-дипломник _____ Євгеній КОЗЛЕНКО
(підпис)

Керівник проєкту _____ Дмитро КОРЕНКО
(підпис)

АНОТАЦІЯ

У дипломній роботі розглянуто методи виявлення аномального трафіку в SDN-мережах. Проведено аналіз підходів із трьох категорій: статистичної (CUSUM), класичного ML (Isolation Forest) та глибокого навчання (Autoencoder, GNN). Визначено їхні переваги та недоліки. Для реалізації обрано CUSUM та Isolation Forest. Розроблена система працює за архітектурою з серверним модулем обробки та клієнтськими сенсорами. Реалізацію виконано з використанням Elixir, Python та ML-бібліотек.

Ключові слова: SDN, аномалії трафіку, CUSUM, Isolation Forest, Elixir, Python.

ANNOTATION

This thesis explores methods for anomaly detection in SDN networks. It analyzes three categories: statistical (CUSUM), classical ML (Isolation Forest), and deep learning (Autoencoder, GNN). Their pros and cons are evaluated. CUSUM and Isolation Forest were selected for implementation. The system uses a server-based processing module with client-side sensors and is implemented in Elixir and Python with ML libraries.

Key words: SDN, traffic anomalies, CUSUM, Isolation Forest, Elixir, Python.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Методи виявлення	4		
			аномалій трафіку в SDN			
			мережах			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Методи виявлення	55		
			аномалій трафіку в SDN			
			мережах			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Методи виявлення	1		
			аномалій трафіку в SDN			
			мережах			
			Діаграма залежності			
			компонентів системи			
			(структурна схема)			
	А4	***ІАЛЦ.4672008.005 Д2***	Методи виявлення	1		
			аномалій трафіку в SDN			
			мережах			
			Діаграма залежності			
			модулів AnomalyHub			
			(функціональна схема)			
	А4	***ІАЛЦ.467200.006 Д3***	Методи виявлення	1		
			аномалій трафіку в SDN			
			мережах			
			Алгоритм дій програмного			
			забезпечення			
			(принципові схема)			
	А4	***ІАЛЦ.467200.007 Д4***	Методи виявлення	90		
			аномалій трафіку в SDN			
			мережах			
			Текст програмного коду			

Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	

ІАЛЦ.467200.001 ОА

Розроб		Козленко Є.Ю.			Методи виявлення аномалій трафіку в SDN мережах Опис альбому	Літ. | | | Аркуш 1 | Аркушів 1
Перев		Коренко Д.В.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-16

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Методи виявлення аномалій трафіку в SDN мережах*»

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи аналізу мережевого трафіку на основі методів виявлення аномалій трафіку в SDN мережах, а також на подальшу підтримку та вдосконалення системи.

Областю застосування цієї системи є моніторинг та забезпечення інформаційної безпеки комп'ютерних мереж підприємств, дата-центрів, телекомунікаційних операторів, а також дослідницьких та критично важливих інформаційних інфраструктур, де використовується SDN-архітектура. Система може бути застосована для виявлення атак типу DDoS, несанкціонованого доступу, сканування портів, а також інших аномальних сценаріїв трафіку в реальному часі з метою оперативного реагування та мінімізації ризиків порушення цілісності, доступності та конфіденційності даних.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», який був затверджений факультетом "Інформатики та обчислювальної техніки" кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка системи моніторингу трафіку програмно-конфігурованих мереж із використанням методів виявлення аномалій трафіку, що дозволить ефективно вирішувати дану задачу.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий і інтуїтивно-зрозумілий інтерфейс системи.
- Надати можливість користувачам отримувати інфографіку щодо свого трафіку.
- Надати можливість користувачам власноруч отримувати повідомлення про виявлені аномалії трафіку.
- Надати вичерпну та зрозумілу документацію для розробленого продукту.

5.2. Вимоги до програмного забезпечення

- ОС Linux.

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i3-2100T.
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.02.2025 – 14.02.2025
Вивчення та аналіз завдання	15.02.2025 – 04.04.2025
Розробка архітектури та загальної структури системи	04.04.2025 – 25.04.2025
Розробка структур окремих частин системи	26.04.2025 – 14.05.2025
Програмна реалізація системи	15.05.2025 - 21.05.2025
Виправлення помилок	22.05.2025-05.06.2025
Оформлення пояснювальної записки	10.02.2025 – 14.02.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Методи виявлення аномалій трафіку в SDN мережах*»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

SDN	Software-Defined Network
CUSUM	Cumulative Sum
IF	Isolation Forest
SVM	Support Vector Machine
API	Application Programming Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

ВСТУП

На тлі цих викликів все більше уваги привертають програмно-конфігуровані мережі (Software-Defined Networking), які докорінно змінюють підходи до проєктування та управління мережевими структурами. Основна концепція SDN побудована на розділенні контрольної площини та площини пересилання трафіку, що дозволяє централізувати управління, спрощує реалізацію політик безпеки, дає можливість оперативно налаштовувати маршрутизацію і гнучко реагувати на будь-які зміни в мережевому середовищі.

Однак впровадження SDN, попри очевидні переваги, породжує нові виклики. Централізація управлінської логіки, вразливість контролера, уніфіковані API та потенційна компрометація сенсорних вузлів роблять таку архітектуру вразливішою. У зв’язку з цим фокус безпеки SDN-мереж поступово зміщується на виявлення аномалій у трафіку. Такі аномальні патерни поведінки можуть свідчити про атаки, спроби проникнення, сканування вразливостей, витоки даних.

Традиційні системи виявлення вторгнень, що зазвичай працюють на основі аналізу сигнатур або заздалегідь визначених правил, часто виявляються недостатньо ефективними у масштабних мережах SDN із високими темпами змін. Це підкреслює важливість використання інтелектуальних методів аналізу трафіку. Технології на основі машинного навчання здатні автоматично ідентифікувати нові моделі аномальної активності завдяки аналізу поведінкових характеристик мережевого трафіку.

Дослідження зосереджується на аналізі сучасних методів виявлення аномального трафіку в SDN-мережах із фокусом на статистичні та машинні методи, включаючи оцінку їхньої ефективності. У межах роботи планується розробити прототип інтелектуальної системи, здатної в реальному часі проводити аналіз трафіку, ідентифікувати потенційно

небезпечну активність і оперативно повідомляти про загрози. Особливу увагу буде приділено питанням масштабованості системи, її адаптивності, захисту компонентів та інтеграції в загальну архітектуру SDN.

РОЗДІЛ 1
ОГЛЯД МЕТОДІВ ВИЯВЛЕННЯ АНОМАЛІЙ ТРАФІКУ В SDN МЕРЕЖАХ

1.1 Програмно-конфігуровані мережі (SDN)

1.1.1 Основні поняття

У широкому сенсі, програмно-конфігурована мережа (SDN, Software-Defined Networking) — це підхід до проєктування, розгортання та управління мережами, що дозволяє централізовано контролювати логіку маршрутизації та обробки трафіку за допомогою програмного забезпечення. Основна ідея полягає у розділенні мережі на рівні: передачі мережі, управління мережі та прикладних додатків. Така декомпозиція дозволяє підвищити гнучкість, масштабованість та керованість сучасних мережевих систем.

Введення програмно-конфігурованих мереж вносить кардинальні зміни у підхід до проектування та управління мережами. По-перше, концепція SDN відокремлює площину управління мережею від площини передачі даних, дозволяючи централізовано керувати маршрутизацією трафіку. По-друге, SDN об'єднує площину управління, дозволяючи одній керуючій програмі ефективно керувати багатьма пристроями на площині даних за допомогою стандартизованого інтерфейсу API, такого як OpenFlow.

Архітектура програмно-конфігурованих мереж, включає в себе шість основних компонентів (рис. 1.1). Перший компонент, розташований на верхньому рівні, відомий як рівень додатків або прикладний рівень, використовується для реалізації різних служб, таких як системи виявлення і запобігання вторгнення, забезпечення якості обслуговування, контроль доступу і проксі-сервер.

Другий компонент, рівень управління, абстрагує топологію мережі від рівня додатків через northbound API. Контролер SDN, що є ключовим елементом на цьому рівні, координує роботу мережевих пристроїв на рівні передачі даних, використовуючи southbound API, такий як протокол OpenFlow. Тут також існують eastbound і westbound API, які дозволяють обмінюватися інформацією між контролерами мережі. Третій компонент, транспортний рівень, відповідає за обробку і передачу пакетів відповідно до інструкцій, отриманих від рівня управління. Північні та південні API-інтерфейси (Northbound і Southbound interfaces) представляють важливі засоби зв'язку для додатків та контролерів SDN, де північні API дозволяють додаткам використовувати мережеві сервіси, а південні API забезпечують ефективне управління ресурсами мережі. Східні/Західні API-інтерфейси забезпечують комунікацію між різними SDN-контролерами, дозволяючи їм обмінюватися інформацією, пов'язаною з обробкою трафіку на рівні передачі даних [1].

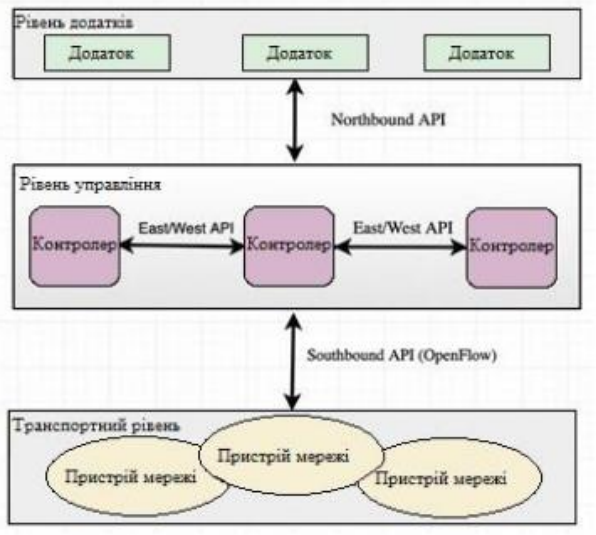


Рисунок 1.1 – Структура мережі SDN

1.1.2 Опис та класифікація аномалій трафіку

Проблеми в комп'ютерних мережах виявляються як аномалії трафіку, які вони спричиняють. Загалом, аномалія - це те, що йде врозріз з очікуваннями. Наприклад, пошкоджений комутатор може створити неочікуваний трафік в іншій частині мережі, або нові коди помилок починають з'являтися, коли служба не працює. Пошук та усунення несправностей в мережі ґрунтується на мережевих аномаліях.

Перший метод класифікації аномалій базується на тому, як вони відрізняються від звичайного зв'язку. Аномалії можуть відрізнятися або за типом переданих даних (поведінкові), або за кількістю переданих даних (за обсягом), або за обома критеріями. Ще один спосіб класифікувати аномалії - за їхньою причиною:

- Помилки, не пов'язані з людським фактором - наприклад, вихід з ладу обладнання або переривання радіозв'язку через погодні умови;
- Помилки, пов'язані з людським фактором - наприклад, збій у роботі мережі, спричинений неправильною конфігурацією або випадково від'єднаним мережевим кабелем;
- Зловмисна людська діяльність - наприклад, інсайдерська атака, коли незадоволений працівник компанії пошкоджує мережевий принтер, або зовнішня атака, коли зловмисник намагається вивести з ладу мережу і завдати шкоди репутації [2].

1.2 Огляд методів виявлення аномалій

Загалом методи аномалій трафіку можна класифікувати за різними критеріями, котрі представлені на рисунку 1.2 .

Рисунок 1.2 – Класифікація методів виявлення аномалій.

1.2.1 Статистичні методи

CUSUM (Cumulative Sum Control Chart) — це класичний статистичний метод виявлення змін у потоці даних, що базується на накопиченій сумі відхилень від очікуваного середнього значення. У контексті виявлення аномалій у трафіку SDN-мереж CUSUM дозволяє ефективно виявляти точку, в якій поведінка мережі відхиляється від нормальної — тобто момент появи аномалії. Основним припущенням алгоритму CUSUM є те, що середнє значення випадкової послідовності є від’ємним за нормальних умов та стає додатним, коли відбувається зміна. Отже, необхідно перетворити $\{X_n\}$ у нову послідовність $\{Z_n\}$, яка задається формулою $Z_n = X_n - \beta$, де β - константа. Параметр β встановлюється відповідно до нормальних умов роботи мережі і гарантує, що більша частина значень послідовності Z_n є від'ємною за нормальних умов і стає додатною, коли відбувається зміна.

На практиці для виявлення аномальних проявів у мережі використовується рекурсивний алгоритм що визначається послідовністю $\{Y_n\}$:

$$\begin{cases} Y_n = (Y_{n-1} + X_n - \beta)^+ \\ Y_0 = 0 \end{cases}, \text{де } x^+ = \begin{cases} x, & x > 0 \\ 0 \end{cases} \quad (1)$$

де β задано таким чином, що значення $X_n - \beta$ залишається злегка від'ємними під час нормальних операцій. Як наслідок, очікується, що збільшення метрики буде виявлено, як тільки значення перевищать β. Тривале надходження значень більших за β, призведе до подальшого збільшення функції CUSUM, поки не буде досягнуто рівня сигналу тривоги.

Велике значення Y_n є ознакою атаки. Виходячи з цього, ми визначаємо ймовірність атаки p для вимірювання ступеня аномальності початкової послідовності X_n:

$$p = \begin{cases} \frac{Y_n}{\alpha * \beta}, & Y_n < \alpha * \beta \\ 1.0 \end{cases}, \quad (2)$$

де p – ймовірність атаки для послідовності X_n; α - регулювальний параметр, який використовується для підсилення значення β і задається як константа (1, 2, …); Y_n - значення CUSUM для послідовності X_n [3].

1.2.2 Методи машинного навчання

Support Vector Machines (SVM) — це потужний метод класифікації та регресії, який застосовується для виявлення аномалій у мережевому трафіку, зокрема у SDN-мережах. Метод працює шляхом побудови гіперплощини, яка максимально відокремлює нормальні зразки від аномальних, або шляхом використання One-Class SVM — для випадків, коли доступні лише нормальні приклади. Тобто ідея полягає у побудові гіперповерхні, що охоплює більшість навчальних даних та реакції на всі дані що не потрапляють до ціжї площини як на аномальні. До особливостів методу можна віднести велику чутливість до вхідних даний, що при

великий натренованості системи, може оцінювати навіть мінімальні відхилення від норми як аномалії [4].

У контексті програмно конфігурованих мереж може ефективно виявляти аномалії за допомогою формування матриць ознак трафіку, наприклад довжини та частоти надходження пакетів, типів протоколу або часу життя потоку, кількості байтів.

Isolation Forest (IF) — це ефективний алгоритм для виявлення аномалій, що ґрунтується на принципі ізоляції аномальних зразків, а не на моделюванні нормального трафіку. Цей метод добре підходить для виявлення аномалій при великому об’ємі даних та високій вимірності простору ознак. Алгоритм може визначати різні види аномалій від окремих ізольованих точок з низькою локальною щільністю так і цілі кластери аномалій [5].

Ідея алгоритму полягає у створенні ансомблю дерев ізоляції, кожне з яких побудоване на ввипадковій підмножині даних, де для кожної точки визначається середня глибина ізоляції та чим менша середня довжина шляху до ізоляції, тим більша ймовірність, що точка є аномальною. Оцінку «аномальності»

$$s(x,n) = 2^{\frac{-E(h(x))}{c(n)}}, \quad (3)$$

де E(h(x)) – середня довжина шляху до ізоляції точки *x, c(n)* – гармонічне число, що нормалізує шлях:

$$c(n) = \ln(n) + \gamma \quad (4)$$

Тобто алгоритм полягає у діленні кожного рівню та відсіканні негативних розподілень. Таким чином графіку буде виглядати як певне дерево (рис. 1.3).

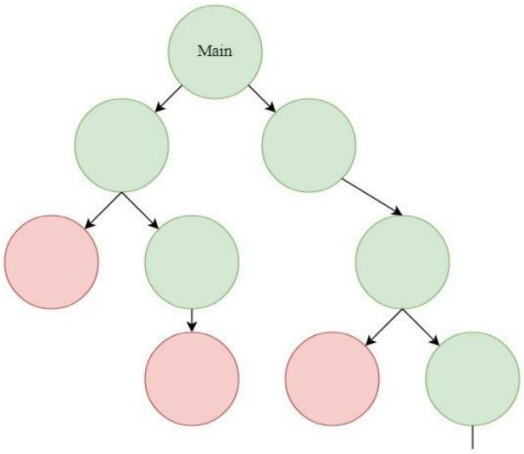


Рисунок 1.3 – Принцип роботи моделі Isolation Forest

1.2.3 Методи глибокого начання (DEEP Learning)

Autoencoder — це тип нейронної мережі, що використовується для виявлення аномалій, особливо в задачах, де необхідно обробляти великі обсяги даних або високо-вимірні особливості. Основною метою autoencoder є відновлення вхідних даних через проміжне стиснуте представлення, що дозволяє ефективно виявляти аномалії на основі великої різниці між відновленими і вихідними даними (рис. 1.4).

Кодер перетворює вхідний вектор X у приховане представлення H:

$$H = \sigma(W_{xh}X + b_{xh}), \quad (5)$$

де σ - функція активації, наприклад, сигмоїдна функція або випрямлена лінійна одиниця, W - вагова матриця, а b - вектор зсуву. Операція перетворення застосовується до прихованого представлення H для відновлення початкового вхідного простору за допомогою декодера [5].

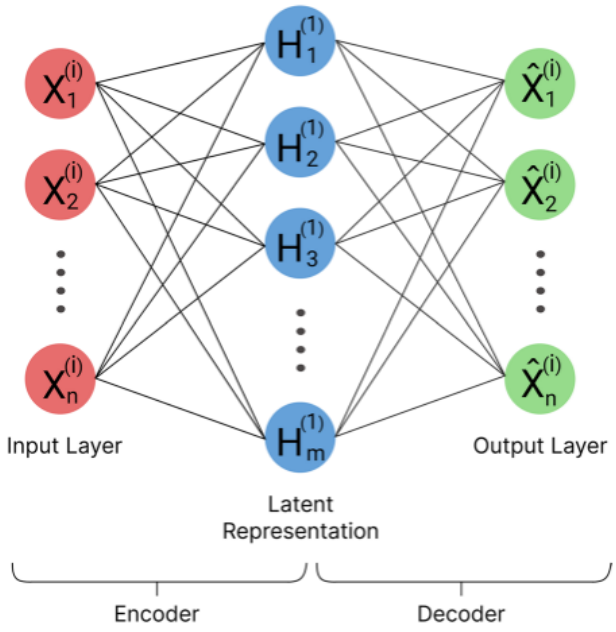


Рисунок 1.4 – Принцип роботи моделі Autoencoder.

У контексті виявлення аномалій у програмно конфігурованих мережах може агрегувати великі обсяги телеметричних даних, де завдяки гнучкості моделі може бути використана для перевірки таблиць або сирих даних отриманих з контролера, підтримує високу розмірність вхідних даних та автоматично вивчає та запам'ятовує структури трафіку.

1.2.4 Методи на основі графів

Graph Neural Networks (GNN) — це клас глибоких моделей навчання, що оперують графовими структурами, де вузли представляють об'єкти, а ребра — взаємозв'язки між ними. Метод GNN дозволяє враховувати не тільки індивідуальні характеристики вузлів, але і структуру зв'язків між ними. Граф є певною сукупністю пножин вузлів, ребер та ознак: $G = (V, E, X \in R^{n*d})$. Система оновл.є представлення кожнаго вузла за допомогою інформації від сусідів:

$$h_v^{(l+1)} = \sigma(\sum_{u \in N(v)} \frac{1}{c_{vu}} W^{(l)} h_u^{(l)}), \quad (6)$$

де $N(v)$ – сусіди вузла u, $W^{(l)}$- матриця ваг на шарі l, σ – функція активації.

У контексті виявлення аномалій, GNN навчається відтворювати або класифікувати властивості графу. Аномалії визначаються як вузли або підграфи, чия поведінка/структура значно відрізняється від решти [5].

Для програмно- конфігурованих мереж дані, забрані з контролера можуть бути зібрані у вигляді таблиць, отриманих від комутаторів, та представленні у вигляді динамічних графів для навчання та використання даної моделі.

Серед підходів до виявлення аномалій можна виокремити:

- Node-level – виявлення вузлів за незвичною поведінкою. Наприклад хости чи комутатори, що генерують невзично велику кількість трафіку чи аномальні типи з'єднань.
- Edge-level – виявлення аномальних зв'язків. Наприклад з'єднання між раніше зв'язаними вузлами.
- Subgraph – аналіз структурних патернів. Наприклад дерева з підозрілою глибиною чи кластери вузлів з незвичайним ступенем взяємодії.

1.3 Порівняльний аналіз методів

Далі будуть описані у табл. 1.1. такі переваги та недоліки алгоритмів, що використовуються для виявлення типів аномалій трафіку.

Таблиця 1.1 – Переваги та недоліки методів виявлення аномалій трафіку.

Алгоритм	Переваги	Недоліки
CUSUM	1. Простота реалізації 2. Висока чутливість до змін у тренді. 3. Адаптивність до різних типів даних. 4. Добре працює у режимі реального часу	1. Чутливість до параметрів порогу. 2. Низька ефективність за складних або нелінійних типах аномалій.

Кінець таблиці 1.1

Алгоритм	Переваги	Недоліки
SVM	1. Навчання без учителя 5. Ефективний у простих випадках з лінійним або слабко-нелінійним розділенням.	1. Погана масштабованість 2. Вразливий до параметрів 3. Не враховує структуру зв’язків.
Isolation Forest	1. Ефективний при роботі з великими даними 2. Не потребує масштабування ознак 6. Працює без учителя	1. Погана продуктивність при роботі з щільними кластерами 4. Менш ефективний для залежних від часу аномалій
Autoencoder	1. Хороша ефективність зі складними та високорозмірними даними 2. Можна адаптувати до часових рядів, графів, зображень 7. Самонавчання	1. Вимагає багато даних для навчання 2. Може «перенавчитися» на нормальні патерни 5. Складна архітектура
GNN	1. Враховує структурні залежності між об’єктами 2. Ефективний для графових даних 8. Може обробляти гетерогенні дані	1. Високі обчислювальні витрати 2. Потребує графової репрезентації даних 6. Чутливий до зміни структури графу

Як видно із таблиці 1.1 кожен метод виявлення аномалій трафіку має свої переваги та недоліки. Проаналізувавши всі позитивні та негативні сторони алгоритмів було прийнято рішення використати поєднання алгоритмів різних рівнів для кращого аналізу даних.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було проведено аналіз сучасних методів виявлення аномалій у трафіку програмно-конфігурованих мереж (SDN). Було визначено, що в умовах постійного зростання складності мережевого трафіку та підвищення вимог до безпеки, особливу увагу необхідно приділяти вибору ефективного алгоритму виявлення нетипової активності. Методи, які розглядалися в межах розділу, охоплюють основні категорії підходів до задачі виявлення аномалій: статистичні методи, класичне машинне навчання та глибоке навчання.

До категорії статистичних методів було віднесено алгоритм CUSUM, який демонструє високу чутливість до змін у потоці трафіку та ефективність у задачах з часовими рядами. Його основними перевагами є низька обчислювальна складність та простота реалізації. Водночас метод є чутливим до вибору параметрів та недостатньо ефективним у випадках складних багатовимірних залежностей у даних.

У категорії методів класичного машинного навчання були розглянуті One-Class Support Vector Machine (SVM) та Isolation Forest. Перший метод базується на принципі побудови оптимальної гіперплощини для відокремлення нормальних даних від потенційно аномальних. Метод є точним, однак потребує значних обчислювальних ресурсів при роботі з великими наборами даних. Isolation Forest, у свою чергу, орієнтований на ізоляцію аномалій через побудову випадкових дерев. Він має добру масштабованість та швидкодію, але менш точний у випадках, коли аномалії мають складну внутрішню структуру.

У категорії глибокого навчання було розглянуто Autoencoder — нейронну мережу, яка навчається реконструювати нормальний трафік і виявляє аномалії на основі похибки реконструкції. Цей підхід добре працює з нелінійними та високовимірними даними, проте потребує великої кількості навчальних прикладів та ресурсомісткого процесу навчання.

Окрім того, було проаналізовано метод Graph Neural Networks (GNN) — сучасний підхід, що дозволяє моделювати мережевий трафік у вигляді графів. GNN ураховує топологію мережі, взаємодії між вузлами, що підвищує точність виявлення аномалій, проте вимагає складної реалізації та значного обсягу якісно підготовлених даних.

Було проведено порівняння описаних методів за рядом ключових характеристик, таких як точність, обчислювальна складність, здатність працювати з часовими залежностями та обсяг потребуваних даних. З огляду на отримані результати, для реалізації системи виявлення аномалій було обрано два методи — Isolation Forest та CUSUM. Перший — як легший за обчисленнями метод класичного машинного навчання, придатний до роботи в режимі реального часу та другий – метод статистичного аналізу для швидкого аналізу. Подальші етапи роботи зосереджено на реалізації, експериментальному тестуванні та порівнянні ефективності обраних моделей у контексті виявлення аномалій у трафіку SDN-мереж.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1 Мова програмування

Для розробки системи в даній роботі було обрано мови програмування Elixir та Python, чому сприяло багато факторів, які будуть послідовно надаватися в наступних пунктах.

2.1.1 Мова Elixir

З усіх мов програмування Elixir, безперечно, вирізняється як одна з найсучасніших і найефективніших для побудови масштабованих, розподілених та відмовостійких систем. Elixir базується на мові Erlang, яка була створена ще у 1980-х роках компанією Ericsson спеціально для потреб телекомунікаційних систем. Erlang проєктувався для роботи в умовах високих навантажень, безперервної доступності та великого обсягу одночасних з’єднань. Саме тому платформа BEAM, на якій працює Elixir, має унікальні властивості, які забезпечують виняткову стійкість і масштабованість [6].

BEAM (Bogdan/Björn's Erlang Abstract Machine) — це віртуальна машина, яка дозволяє запускати сотні тисяч одночасних lightweight-процесів. Ці процеси не блокують один одного, споживають мінімальні ресурси та повністю ізольовані. Завдяки цьому збій одного процесу не призводить до краху всієї системи. Кожен процес у BEAM є незалежним і легким, на відміну від потоків операційної системи. Модель акторів, яку реалізує BEAM, передбачає обмін повідомленнями між процесами без спільної пам’яті, що значно спрощує паралельну обробку та виключає типові помилки конкурентного програмування [7].

Ключовою концепцією в Elixir є супервізори — спеціальні процеси, які слідкують за підлеглими процесами та перезапускають їх у разі збою. Це дозволяє системі самостійно відновлювати працездатність після помилок, без втручання адміністратора. Такі підходи формують архітектуру, що орієнтована не на уникнення збоїв, а на їх правильну локалізацію та автоматичне усунення [8].

Відомі компанії, як-от Discord, Pinterest, PepsiCo та Moz, обрали Elixir для критичних частин своїх сервісів. Наприклад, Discord використовує Elixir для обробки мільйонів одночасних з’єднань у голосових чатах. Це підтверджує, що мова придатна до масштабування у промислових умовах.

Станом на 2024 рік Elixir стабільно зростає в популярності за результатами опитувань Stack Overflow та GitHub, особливо серед DevOps-фахівців, бекенд-інженерів та розробників систем реального часу. Її здатність масштабуватися горизонтально, без шкоди для продуктивності, робить Elixir одним із найкращих рішень для сучасних архітектур мікросервісів та обробки великого трафіку.

На рисунку 2.1 можна наглядно побачити рейтинг мов програмування за даними TIOBE Index.

Position	Programming Language	Ratings
21	Ruby	0.77%
22	Prolog	0.77%
23	Swift	0.77%
24	Lisp	0.77%
25	Kotlin	0.72%
26	Classic Visual Basic	0.67%
27	SAS	0.57%
28	(Visual) FoxPro	0.49%
29	Scala	0.49%
30	Haskell	0.46%
31	Dart	0.46%
32	VBScript	0.41%
33	Lua	0.40%
34	Objective-C	0.37%
35	Julia	0.36%
36	Bash	0.28%
37	TypeScript	0.26%
38	ABAP	0.26%
39	RPG	0.21%
40	PL/SQL	0.21%
41	Solidity	0.20%
42	GAMS	0.19%
43	V	0.19%
44	D	0.17%
45	PowerShell	0.16%
46	ML	0.15%
47	Elixir	0.14%
48	Awk	0.14%

Рисунок 2.1 – Популярність мов програмування за даними TIOBE Index [15]

2.1.2 Мова Python

Серед мов програмування Python вирізняється своєю універсальністю, простотою синтаксису та широкою екосистемою бібліотек, що робить її ідеальним вибором для реалізації систем аналізу даних, зокрема — систем виявлення аномалій трафіку в SDN-мережах. Python розроблявся з фокусом на читабельність коду та швидкість розробки, що особливо важливо в умовах обмежених термінів та експериментального характеру дослідницьких проєктів.

Однією з ключових переваг Python є наявність потужних бібліотек для роботи з машинним навчанням та аналізом даних, таких як scikit-learn, TensorFlow, PyTorch, pandas, NumPy та matplotlib. Завдяки цим інструментам можливо швидко будувати, навчати та валідувати моделі, що

критично важливо для проєктів з виявлення аномалій, де часто використовується підхід "data-driven".

Python також активно використовується у науковому та інженерному середовищі, що підтверджується його високими позиціями у рейтингах мов програмування (). Наприклад, згідно з даними Stack Overflow та IEEE Spectrum за 2024 рік, Python стабільно входить до трійки найпопулярніших мов серед розробників і науковців [16].

Для роботи з SDN Python забезпечує підтримку низки бібліотек і фреймворків, таких як POX, RYU та Faucet, які дозволяють взаємодіяти з контролерами OpenFlow, емулювати трафік та будувати кастомізовану логіку маршрутизації. У зв’язку з цим Python є природним вибором для створення прототипу системи виявлення аномалій на основі поведінкових характеристик мережевого трафіку.

Окрім того, Python має зручну інтеграцію з іншими інструментами: системами візуалізації (наприклад, Grafana через API), веб-серверами (Flask, FastAPI), а також платформами зберігання даних (InfluxDB, MongoDB, PostgreSQL). Це дозволяє будувати повноцінні аналітичні пайплайни — від збору даних до їх обробки, аналізу та подальшої інтеграції у графічні інтерфейси адміністратора.

Усе це робить Python оптимальним вибором для системи, де необхідно швидко і надійно реалізувати інтелектуальні методи аналізу трафіку, з мінімальними затратами на розгортання та підтримку. Компанії, як-от Google, Netflix, NASA та Dropbox, широко використовують Python для створення складних аналітичних та автоматизованих систем, що свідчить про його придатність до реальних задач високого рівня складності.

2.1.3 Підсумок щодо вибору мови

Узагальнюючи всі зважені аргументи щодо вибору мов програмування для реалізації даного проєкту, було прийнято рішення використати комбінований підхід, що поєднує Python та Elixir. Така архітектура дозволяє максимально ефективно використати сильні сторони кожної мови: Elixir — для реалізації високонадійного серверного компонента, здатного обробляти велику кількість з’єднань у реальному часі, та Python — для побудови і навчання моделей машинного навчання, а також гнучкого аналізу трафіку. Комбіноване використання цих технологій дозволяє досягти балансу між продуктивністю, масштабованістю та інтелектуальністю системи, а також забезпечити гнучкість під час розробки та підтримки.

2.2 Технології та бібліотеки

Далі описуються технології та бібліотеки, які будуть використовуватися у проекті.

2.2.1 Phoenix Framework

Phoenix — це сучасний веб-фреймворк для мови програмування Elixir, створений з акцентом на високу продуктивність, масштабованість і зручність розробки веб-додатків. Його архітектура орієнтована на одночасне обслуговування великої кількості з'єднань і надання високої доступності в умовах навантажених систем.

На відміну від традиційних веб-фреймворків, Phoenix використовує можливості віртуальної машини BEAM, що дозволяє одночасно обробляти сотні тисяч з'єднань без шкоди для стабільності. Використовуючи легкі

процеси та модель акторів, Phoenix легко масштабується горизонтально, забезпечуючи реальну конкурентність при мінімальних витратах ресурсів.

Одна з ключових особливостей Phoenix — це підтримка Phoenix Channels, яка забезпечує двосторонню комунікацію в реальному часі через WebSocket. Цей підхід ідеально підходить для створення чатів, live-панелей, ігор, фінансових дашбордів та будь-яких додатків, що потребують миттєвого оновлення даних.

Phoenix LiveView — ще одна унікальна можливість фреймворку, яка дозволяє створювати інтерактивні фронтенд-компоненти без написання JavaScript. Завдяки LiveView розробники можуть будувати динамічні веб-додатки з мінімальною складністю, передаючи логіку у бекенд, що спрощує супровід і покращує продуктивність.

Phoenix забезпечує надійні механізми захисту від атак CSRF, XSS і фіктивних запитів. Крім того, чітка структура проектів і прозорий код сприяють легкому тестуванню, відлагодженню та підтримці.

Phoenix активно використовується компаніями, які обробляють великий трафік у режимі реального часу. Наприклад, платформа для трейдингу Financial Times, стартапи типу Podium та сервіси, що працюють із мільйонами користувачів щодня, демонструють успішне впровадження Phoenix у високонавантажених умовах [10].

2.2.2 Ecto

Ecto — це офіційна бібліотека для роботи з базами даних у середовищі Elixir, яка надає засоби для створення, запитів, міграцій та перевірки цілісності даних. Вона є невід’ємною частиною екосистеми Phoenix і забезпечує чисту, декларативну модель взаємодії з реляційними базами даних.

Ecto пропонує потужний Domain-Specific Language (DSL), який дозволяє будувати SQL-запити у вигляді Elixir-коду. Це забезпечує

статичну перевірку на етапі компіляції, автодоповнення та підвищену безпеку порівняно з сирими SQL-рядками.

За допомогою Ecto розробники можуть легко створювати та відслідковувати міграції бази даних, а також підтримувати схему в актуальному стані. Механізм змін схеми дозволяє контролювати версії структури даних у різних середовищах.

Ecto підтримує визначення змінюваних структур (changesets), які використовуються для перевірки, фільтрації та валідації даних перед їх збереженням. Крім того, легко реалізуються зв’язки між сутностями: one-to-one, one-to-many, many-to-many.

Ecto підтримує підключення до різних СУБД (PostgreSQL, MySQL, MSSQL тощо) через адаптери. Робота з базою даних є асинхронною та оптимізованою для паралельного виконання завдяки можливостям BEAM [11].

2.2.3 PostgreSQL

PostgreSQL — це потужна об'єктно-реляційна система управління базами даних з відкритим кодом, яка часто використовується в екосистемі Elixir як основна СУБД через свою надійність, масштабованість і функціональність. Вона підтримується спільнотою вже понад 30 років і є одним із найбільш стабільних рішень для промислового використання.

PostgreSQL дотримується стандарту SQL і має ACID-властивості, що гарантує цілісність даних навіть у випадку аварій. Завдяки цьому вона ідеально підходить для фінансових, державних та інших критичних систем.

Система підтримує розширення (наприклад, PostGIS для геоданих), типи JSONB для зберігання напівструктурованих даних, а також складні індекси, тригери, збережені процедури та інші механізми для побудови складних запитів і логіки всередині бази.

Через адаптер Postgrex бібліотека Ecto забезпечує повну підтримку PostgreSQL, включаючи транзакції, batch-операції, асинхронну обробку запитів та роботу з JSONB-полями. Це дає змогу ефективно працювати з великим обсягом даних у реальному часі.

PostgreSQL залишається рекомендованим вибором для більшості проєктів на Elixir через свою гнучкість, продуктивність і активну підтримку спільноти [13, 14].

2.2.4 Scikit-learn

Scikit-learn — це одна з найпопулярніших і найпотужніших бібліотек машинного навчання у мові програмування Python. Вона надає зручний інтерфейс до великої кількості класичних алгоритмів машинного навчання, що охоплюють як задачі класифікації, регресії, кластеризації, так і зниження розмірності, вибір ознак, побудову моделей та їх оцінку. Scikit-learn є високорівневою обгорткою над такими бібліотеками, як NumPy, SciPy та matplotlib, що забезпечує високу продуктивність та інтеграцію з іншими інструментами обробки даних.

Основні можливості Scikit-learn:

- Класифікація: SVM, Decision Trees, Random Forest, k-NN, Naive Bayes та інші.
- Регресія: Linear Regression, Ridge, Lasso, SVR.
- Кластеризація: k-means, DBSCAN, агломеративна кластеризація.
- Зниження розмірності: PCA, t-SNE.
- Оцінка моделей: крос-валідація, матриця помилок, метрики якості (accuracy, precision, recall, F1-score).
- Побудова пайплайнів (pipeline) для автоматизації процесу підготовки даних та тренування моделей.

Scikit-learn ідеально підходить для задач попереднього аналізу, розробки прототипів і інтеграції моделей у промислові системи [9, 12].

2.2.5 REST API

REST API (англ. Representational State Transfer Application Programming Interface) — це архітектурний стиль створення веб-сервісів, який дозволяє системам обмінюватися інформацією через стандартні HTTP-запити. REST API широко використовується для побудови сучасних веб-додатків і клієнт-серверної взаємодії.

Ключові принципи REST API:

- Клієнт-серверна архітектура: клієнт і сервер є незалежними; клієнт надсилає запити, сервер обробляє їх і повертає відповіді.
- Статус без стану (stateless): кожен запит містить усю необхідну інформацію для його обробки, сервер не зберігає стан клієнта.
- Уніфікований інтерфейс: використовуються стандартні HTTP-методи:
  - GET — для отримання ресурсу;
  - POST — для створення нового ресурсу;
  - PUT — для повного оновлення ресурсу;
  - PATCH — для часткового оновлення;
  - DELETE — для видалення ресурсу.
- Адресація через URI: кожен ресурс доступний за унікальною URL-адресою.
- Передача даних у форматі JSON або XML (найчастіше використовується JSON).

REST API активно використовується для створення інтерфейсів до сервісів машинного навчання, зокрема — для передачі зібраних даних на сервер, запуску обробки або отримання результатів прогнозу. У контексті цього проєкту REST API забезпечує взаємодію між клієнтом, який передає трафік, і сервером, який виконує аналіз за допомогою моделі, реалізованої у Python [17].

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було детально розглянуто та обґрунтовано вибір мов програмування та допоміжних технологій, які будуть використані для реалізації проєкту. У фокусі були мови Python та Elixir, а також супутні фреймворки та інструменти, що забезпечують взаємодію між компонентами та зберігання даних.

Перш за все було обґрунтовано доцільність використання мови Python — потужного інструменту для побудови моделей машинного навчання. Зокрема, було описано застосування таких бібліотек як scikit-learn, TensorFlow, які надають широкі можливості для обробки даних, побудови автоенкодерів і виконання статистичного аналізу трафіку. Python також використовується для створення REST API за допомогою Flask, що забезпечує зручну взаємодію з іншими частинами системи.

Далі було детально розглянуто використання мови програмування Elixir як основного інструменту для реалізації високонавантаженої серверної логіки. Завдяки платформі BEAM та підтримці модель акторів, Elixir забезпечує масштабованість, конкурентність і відмовостійкість. Для реалізації веб-інтерфейсу та обробки запитів у реальному часі було обрано фреймворк Phoenix, який є одним із найпродуктивніших рішень у світі Elixir. Також у проєкті буде використано бібліотеку Ecto — потужний інструмент для роботи з базами даних у середовищі Elixir, що забезпечує зручну ORM-абстракцію, валідацію даних та ефективну інтеграцію з базою даних PostgreSQL.

СУБД PostgreSQL була обрана як надійне, розширюване та продуктивне рішення для зберігання як історичних даних про трафік, так і результатів аналізу аномалій. PostgreSQL добре масштабується, підтримує складні запити, транзакції, а також сумісна як з Python-інструментами, так і з Ecto.

Таким чином, обраний стек технологій дозволяє реалізувати надійну, масштабовану та інтелектуальну систему. Комбінування переваг Python для реалізації складних обчислень і машинного навчання з надійністю Elixir та Phoenix для обробки трафіку в реальному часі забезпечує високу ефективність і гнучкість усієї архітектури. Усі обрані інструменти мають чітко визначені ролі, які взаємодоповнюють одна одну і сприяють досягненню поставленої мети.

РОЗДІЛ 3 ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

Відповідно до проведеного аналізу, була розроблена інтелектуальна система виявлення аномального трафіку в SDN-мережі, яка базується на використанні методів машинного навчання. Враховуючи архітектурні особливості SDN, було прийнято рішення про модульну побудову системи з поділом на три основні компоненти: сенсорний агент, центральний сервер (Elixir Hub) та мікросервіс штучного інтелекту для детекції.

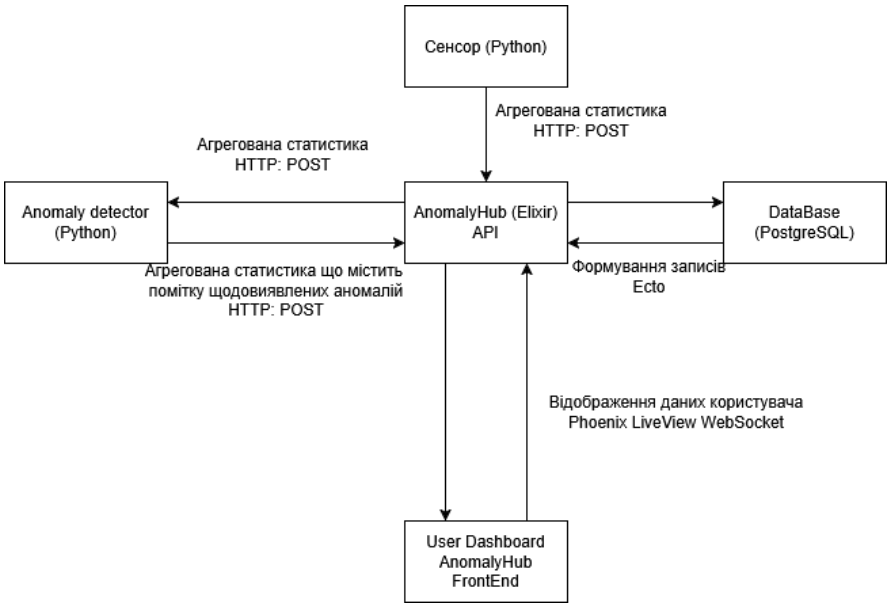
3.1 Архітектура системи

У ході проектування та розробки системи виявлення аномалій було розглянуто загальні підходи до розробки веб-додатків такі як моноліт та мікросервісна архітектура. Було вирішено поєднати обидва підходи для досягнення максимальної продуктивності, поєднання хороших сторін та мінімізації негативних аспектів обох підходів.

Система складається з набору сенсорів, що встановлюються локально на машину клієнта та підключається до мережевого інтерфейсу пристрою, збирає та агрегує інформацію яку отримує (кількість пакетів, цільові порти, ip-адреси), надсилає до розгорнутого на сервері монолітного проекту (AnomalyHub), виконаного мовою програмування Elixir, за допомогою HTTP протоколів. AnomalyHub за допомогою RestAPI поєднаний з мікросервісом аналізу трафіку надсилає та отримує записи отримані від сенсору, та у вигляді графіків статистики відправляє на веб-сторінку користувача, що дозволяє йому контролювати роботу свого пристрою не маючи фізичного доступу до нього.

Розроблена архітектурна модель має хорошу модульність, адже дозволяє вільно змінювати роботу сенсорів, підключати та відключати нові модулі аналізу трафіку, проте уникає недоліків мікросервісних архітектур, як велика роздробленість проекту, ускладнене розгортання, тощо.

На рисунку 3.1 описано схему роботи проєкту та використані технології.



Рисенок 3.1 – Структурна схема організації додатку.

3.2 Огляд сенсорного агенту (Sensor)

Сенсор — це невеликий автономний Python-додаток, який розгортається безпосередньо на клієнтських пристроях та виконує роль агента збору телеметричних даних про мережевий трафік, де здійснює пасивне прослуховування мережевого інтерфейсу, агрегує ключові характеристики трафіку та передає їх на центральний сервер.

З технічної точки зору сенсор підключається безпосередньо до мережевого інтерфейсу пристрою, використовуючи можливості бібліотеки Scapy, що дозволяє здійснювати захоплення, аналіз та фільтрацію мережевих пакетів на низькому рівні. Такий підхід забезпечує гнучкість у

роботі з різними мережевими протоколами та форматами пакетів без потреби в додатковому програмному забезпеченні чи драйверах.

Сенсор працює у фоновому режимі та виконує збір таких метрик, як (рис 3.2):

- загальна кількість перехоплених пакетів;
- кількість унікальних IP-адрес джерел;
- розподіл пакетів за мережевими протоколами (TCP, UDP тощо);
- кількість TCP SYN-запитів (що потенційно може вказувати на спроби сканування портів);
- статистику найбільш часто використовуваних портів призначення.

```
def packet_callback(packet):
    if packet.haslayer(scapy.IP):
        with lock:
            stats["packet_count"] += 1
            stats["unique_src_ips"].add(packet[scapy.IP].src)

            proto = packet[scapy.IP].proto
            stats["proto_counter"][proto] += 1

            if packet.haslayer(scapy.TCP):
                tcp_layer = packet[scapy.TCP]
                # Перевірка лише SYN флагу
                if tcp_layer.flags == "S":
                    stats["tcp_syn_count"] += 1
                stats["dst_ports"][tcp_layer.dport] += 1

            elif packet.haslayer(scapy.UDP):
                udp_layer = packet[scapy.UDP]
                stats["dst_ports"][udp_layer.dport] += 1
```

Рисунок 3.2 – Реалізація методу сканування мережевого трафіку.

Вся зібрана інформація агрегується у реальному часі та з інтервалом у кілька секунд (типово — 5) у форматі JSON надсилається на центральний сервер за допомогою HTTP-запитів. Авторизація запитів відбувається за

допомогою API-ключів, що дозволяє забезпечити автентифікацію кожного сенсора та прив'язку його до конкретного користувача в системі (рис. 3.3).

```
def send_stats(config):
    while True:
        time.sleep(SEND_INTERVAL)
        with lock:
            payload = {
                "timestamp": int(time.time()),
                "packet_count": stats["packet_count"],
                "unique_src_ip_count": len(stats["unique_src_ips"]),
                "proto_counter": dict(stats["proto_counter"]),
                "tcp_syn_count": stats["tcp_syn_count"],
                "top_dst_ports": stats["dst_ports"].most_common(10)
            }

            headers = {
                "Authorization": f"Bearer {config['api_key']}",
                "Content-Type": "application/json"
            }

            try:
                response = requests.post(SERVER_URL, json=payload, headers=headers)
                print(f"[+] Надіслано {payload['packet_count']} пакетів. Статус: {response.status_code}")
            except Exception as e:
                print(f"[!] Помилка надсилання: {e}")

            # Очистка
            stats["packet_count"] = 0
            stats["unique_src_ips"].clear()
            stats["proto_counter"].clear()
            stats["tcp_syn_count"] = 0
            stats["dst_ports"].clear()
```

Рисунок 3.3 – Реалізація методу відправлення даних до серверу.

Конфігурація сенсора є мінімалістичною: при першому запуску користувач вводить унікальний API-ключ, після чого він зберігається у локальному конфігураційному файлі (рис. 3.4). Це спрощує повторні запуски та підтримку агента. Надсилання даних виконується у паралельному потоці, що дозволяє сенсору працювати асинхронно, не блокуючи процес збору нових пакетів.

```python
# Завантаження .env
load_dotenv()
SERVER_URL = os.getenv("SERVER_URL")

if not SERVER_URL:
    raise RuntimeError("[!] SERVER_URL не знайдено у .env файлі!")

def load_or_create_config():
    if os.path.exists(CONFIG_FILE):
        with open(CONFIG_FILE, 'r') as f:
            config = json.load(f)
            print("[*] Конфігурацію сенсора завантажено.")
    else:
        print("[*] Введіть ваш унікальний API ключ (наданий адміністратором):")
        api_key = input("API ключ: ").strip()
        config = {"api_key": api_key}
        with open(CONFIG_FILE, 'w') as f:
            json.dump(config, f)
            print(f"[*] Конфігурацію збережено у {CONFIG_FILE}.")
    return config
```

Рисунок 3.4 – Реалізація конфігурування сенсору.

Окремо варто відзначити, що структура сенсора розроблена з урахуванням розширюваності — при потребі можливо легко додати нові типи метрик або змінити логіку обробки трафіку без необхідності повної перебудови програми.

Таким чином, сенсор забезпечує локальне перехоплення та попередню обробку трафіку без впливу на продуктивність клієнтського пристрою та виконує роль сполучної ланки між фізичною мережею користувача та аналітичним модулем, розміщеним на сервері. Його легкість, автономність і здатність до швидкого розгортання роблять його ефективним засобом моніторингу у розподіленому середовищі.

3.3 Огляд центрального серверу AnomalyHub

Центральним елементом архітектури розробленої системи є AnomalyHub — серверний компонент, відповідальний за прийом, обробку та подальшу маршрутизацію телеметричних даних, отриманих від сенсорів. Його реалізовано як монолітний веб-додаток на мові програмування Elixir з використанням фреймворку Phoenix, що забезпечує

високу стабільність, масштабованість і обробку великої кількості одночасних з'єднань завдяки конкурентній моделі Erlang/BEAM.

Хаб виступає центральною точкою зв'язку в системі, координуючи обмін інформацією між сенсорами, користувачами та мікросервісами аналізу. Основні функції хабу поділяються на кілька логічних компонентів

3.2.1 Обробка трафіку від сенсорів

Хаб приймає HTTP POST-запити, що надсилаються сенсорами через API. Кожен запит містить у собі агреговані мережеві дані та унікальний API-ключ, який дозволяє автентифікувати джерело трафіку та прив'язати його до відповідного користувача. Завдяки вбудованим можливостям Phoenix для роботи з сокетами та асинхронною обробкою запитів, хаб здатен ефективно обробляти дані з великої кількості джерел у реальному часі (рис. 3.5).

```
def receive_data(conn, params) do
  case get_req_header(conn, "authorization") do
    ["Bearer " <> api_key] ->
      case Sensors.get_sensor_by_api_key(api_key) do
        nil ->
          send_resp(conn, 401, "Invalid API Key")

        %Sensor{} = sensor ->
          Task.start(fn ->
            forward_and_handle_anomalies(sensor, params)
          end)

          send_resp(conn, 202, "Accepted")
      end

    _ ->
      send_resp(conn, 401, "Missing API Key")
  end
end
```

Рисунок 3.5 – Реалізація контролеру для отримання даних.

Після автентифікації дані перевіряються на коректність та зберігаються у базі даних для подальшого використання. Тут застосовується ORM бібліотека Ecto, яка спрощує роботу з реляційними

базами даних (типово PostgreSQL), забезпечуючи зручний інтерфейс для побудови запитів та ведення історії отриманого трафіку.

Паралельно, отримані записи надсилаються до мікросервісу машинного навчання, який виконує аналіз трафіку на предмет аномалій. Комунікація з аналітичним модулем реалізується через REST API, що забезпечує слабку зв'язаність між компонентами та полегшує майбутнє масштабування системи (рис. 3.6).

```
defp forward_and_handle_anomalies(sensor, params) do
  headers = [
    {"Content-Type", "application/json"},
    {"Authorization", "Bearer #{sensor.api_key}"}
  ]

  body = Jason.encode!(params)

  case HTTPoison.post(@detector_url, body, headers) do
    {:ok, %HTTPoison.Response{status_code: 200, body: body}} ->
      case Jason.decode(body) do
        {:ok, %{} = data} ->
          TrafficRecords.store_report(sensor, data)

        _ ->
          IO.warn("[!] Could not parse AI module response")
      end

    {:error, err} ->
      IO.inspect(err, label: "Error contacting AI module")

    err ->
      IO.inspect(err)
      IO.warn("[!] Unknown error from AI module")
  end
end
```

Рисунок 3.6 – Реалізація методу надсилання даних до сервісу виявлення аномалій.

3.2.3 Збереження та візуалізація даних

Всі прийняті та оброблені дані зберігаються у базі даних із прив'язкою до відповідного користувача та сенсора. На основі цієї інформації формуються графіки та діаграми, доступні користувачеві через веб-інтерфейс. Для зручності реалізовано особистий кабінет, у якому кожен користувач має змогу бачити історію активності свого пристрою, рівень підозрілої активності тощо.

3.2.2 Реактивна взаємодія з користувачем

AnomalyHub також підтримує WebSocket-з’єднання, завдяки чому можлива реактивна трансляція оновлень — як-от появи нових аномалій чи оновлень трафіку — у реальному часі на фронтенд без необхідності ручного оновлення сторінки.

Таким чином, AnomalyHub виступає як координаційний центр усієї системи, забезпечуючи обробку та маршрутизацію даних, авторизацію користувачів, інтеграцію з інтелектуальним аналізом трафіку та зручний користувацький інтерфейс. Його реалізація на базі Elixir дозволила досягти високої продуктивності, ефективної роботи в умовах великої кількості паралельних з’єднань та готовності до масштабування без повної перебудови архітектури. Також будь-який із функціональних блоків (зберігання, візуалізація, API, логіка взаємодії з ML) може бути відокремлений в окремий мікросервіс при зростанні навантаження або зміні вимог до проєкту.

3.4 Огляд мікросервісу виявлення аномалій

У складі системи виявлення аномального трафіку важливу роль відіграє сервіс аналізу трафіку — незалежний мікросервіс, відповідальний за обробку та аналіз агрегованих статистичних даних, які надходять від сенсорів через центральний хаб (AnomalyHub). Цей сервіс виконує автоматичне виявлення аномалій у мережевому трафіку в режимі реального часу, забезпечуючи раннє виявлення потенційних кіберзагроз.

Сервіс аналізу приймає агреговану інформацію про мережевий трафік (кількість пакетів, унікальні IP-адреси, SYN-пакети, порти тощо) у вигляді JSON-запитів від центрального хабу. Його основне завдання — оцінити ці дані та визначити, чи є вони типовими (нормальними), чи відхиляються від очікуваних шаблонів поведінки, сигналізуючи про можливу аномалію.

Сервіс реалізовано за допомогою Python — мови, яка часто використовується в області аналізу даних та машинного навчання. У якості API-інтерфейсу використовується легкий фреймворк FastAPI, який дозволяє швидко та ефективно будувати REST API, що приймає запити від Elixir-хабу та повертає результат аналізу у зручному вигляді.

Сервіс аналізу поєднує два підходи до виявлення аномалій, що дозволяє покрити як довгострокові статистичні відхилення, так і миттєві сплески активності.

Було використано модель машинного навчання Isolation Forest, що вивчає типові патерни трафіку (кількість унікальних IP-адрес, SYN-запити, порти) та згодом виявляє відхилення. Такий підхід добре підходить для високорівневого аналізу та виявлення складних аномалій на основі трендів.

Доповнює систему статистичний метод детектування змін CUSUM, який відстежує послідовні коливання значень (наприклад, кількість пакетів за інтервал часу) та виявляє сталі зсуви в середньому значенні. Його застосування дає можливість швидко реагувати на сплески активності, такі як різке зростання SYN-запитів, сканування портів або спроби DOS-атак. Реалізація алгоритму зображена на рисунку 3.7.

```python
class CusumDetector:
    def __init__(self, threshold: float, drift: float = 0.0):
        self.threshold = threshold
        self.drift = drift
        self.pos_sum = 0.0
        self.neg_sum = 0.0
        self.last_mean = None

    def update(self, value: float) -> bool:
        if self.last_mean is None:
            self.last_mean = value
            return False

        diff = value - self.last_mean - self.drift
        self.pos_sum = max(0, self.pos_sum + diff)
        self.neg_sum = min(0, self.neg_sum + diff)

        if self.pos_sum > self.threshold or abs(self.neg_sum) > self.threshold:
            self.pos_sum = 0
            self.neg_sum = 0
            return True

        return False
```

Рисунок 3.7 – Реалізація класу CUSUM для виявлення відхилень у трафіку.

Комунікація з хабом відбувається за допомого контролера який проводить аналіз ключа та запускає метод перевірки отриманих агрегованих даних на основі імплементованих алгоритмів та помічає пакет за допомогою прапорця *is_anomaly*. Оброблені дані повертаються до хабу та записуються в базу даних. Реалізацію контролеру зображено на рисунку 3.8.

```python
@app.post("/api/detect")
async def receive_aggregate(request: Request, payload: AggregatedTraffic):
    auth = request.headers.get("Authorization")
    if not auth or not auth.startswith("Bearer "):
        raise HTTPException(status_code=403, detail="Missing or invalid Authorization header")

    user_key = auth.split()[1]

    feature_vector = extract_features_from_aggregate(payload)

    if user_key not in user_buffers:
        user_buffers[user_key] = []
    user_buffers[user_key].append(feature_vector)

    if user_key not in user_cusums:
        user_cusums[user_key] = CusumDetector(threshold=500.0, drift=10.0)
    cusum_anomaly = user_cusums[user_key].update(payload.tcp_syn_count)

    if user_key not in user_models:
        load_user_model(user_key)

    if user_key not in user_models and len(user_buffers[user_key]) >= MIN_TRAIN_SIZE:
        data = np.array(user_buffers[user_key])
        scaler = StandardScaler()
        scaled_data = scaler.fit_transform(data)
        model = IsolationForest(contamination=0.05, random_state=42)
        model.fit(scaled_data)
        user_models[user_key] = model
        user_scalers[user_key] = scaler
        await save_models_async()  # Автоматичне збереження після тренування
        return {**payload.dict(), "anomaly": False, "message": "Model trained"}

    isolation_anomaly = False
    if user_key in user_models:
        scaled = user_scalers[user_key].transform([feature_vector])
        pred = user_models[user_key].predict(scaled)[0]
        isolation_anomaly = (pred == -1)

    is_anomaly = cusum_anomaly or isolation_anomaly

    return {
        **payload.dict(),
        "anomaly": bool(is_anomaly),
        "cusum_anomaly": bool(cusum_anomaly),
        "isolation_anomaly": bool(isolation_anomaly)
    }
```

Рисунок 3.8 – Реалізація контролеру сервісу аналізу трафіку.

ВИСНОВОК ДО РОЗДІЛУ 3

У цьому розділі було детально описано архітектуру інтелектуальної системи виявлення аномального трафіку в SDN-мережах. Система побудована з урахуванням сучасних вимог до масштабованості, модульності та адаптивності та умов реального середовища. Архітектура поєднує елементи монолітного та мікросервісного підходів, що дозволяє досягти оптимального балансу між простотою обслуговування, продуктивністю та гнучкістю розвитку системи в майбутньому.

Ключовими компонентами системи є локальні сенсори, центральний сервер (AnomalyHub) та сервіс аналізу трафіку. Сенсори, реалізовані на основі бібліотеки Scapy, виконують агрегацію статистичних характеристик з мережевого інтерфейсу клієнтської машини, зберігаючи дані на проміжному рівні, перш ніж надіслати їх до центрального вузла обробки. Передача даних відбувається через захищені HTTP-запити з використанням API-ключів, що забезпечує базовий рівень автентифікації та контролю доступу.

Центральний компонент системи, AnomalyHub, реалізований мовою Elixir на базі фреймворку Phoenix, виступає в ролі координатора системи. Він виконує функції обробки запитів, маршрутизації, зберігання даних у СКБД PostgreSQL, а також взаємодії з користувачем через веб-інтерфейс. Така реалізація забезпечує високу продуктивність і здатність обробляти одночасні з’єднання завдяки використанню неблокуючої архітектури Erlang VM.

Окрему роль відіграє сервіс аналізу трафіку — AI-модуль, який використовує алгоритм Isolation Forest для виявлення відхилень у багатовимірному просторі ознак, що характеризують трафік. Для виявлення поступових змін у поведінці системи застосовується також метод CUSUM. Комбінація цих підходів дозволяє досягти високої точності

у виявленні як різких, так і прихованих аномалій, пов’язаних із мережевими атаками або відхиленнями від нормальної поведінки.

Особливістю архітектури є також збереження даних для подальшого аналізу, побудови графіків і виявлення довготривалих трендів. Кожен компонент системи реалізований таким чином, щоб забезпечити низьку зв'язаність і високу відповідальність, що відповідає принципам побудови сучасних розподілених систем. Таким чином, розроблена архітектура системи відповідає сучасним вимогам до систем мережевого моніторингу та безпеки. Запропоноване рішення може ефективно використовуватись як у лабораторних умовах, так і в реальних SDN-мережах для підвищення рівня безпеки та контролю.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1 Огляд інтерфейсу розробленої системи.

Інтерфейс розробленої системи складається з двох копонентів: інтерфейсу сенсору та інтерфейсу хабу. Сенсор є консольною програмо, я виконується у середовищі Python та запускається на кінцевому пристрої користувача або віртуальній машині та збирає морежевий трафік. Його інтерфейс є максимально мінімалістичним – вивід у консоль запиту на введення ключа підключення, кількість надісланих пакетів та статус їх надсилання. Сенсор дозволяє зручно відстежувати свою роботу без використання додаткових ресурсів, що особливо актуально для розгортання на слабких пристроях або пристроях Iot (Internet of things) (рис. 4.1).

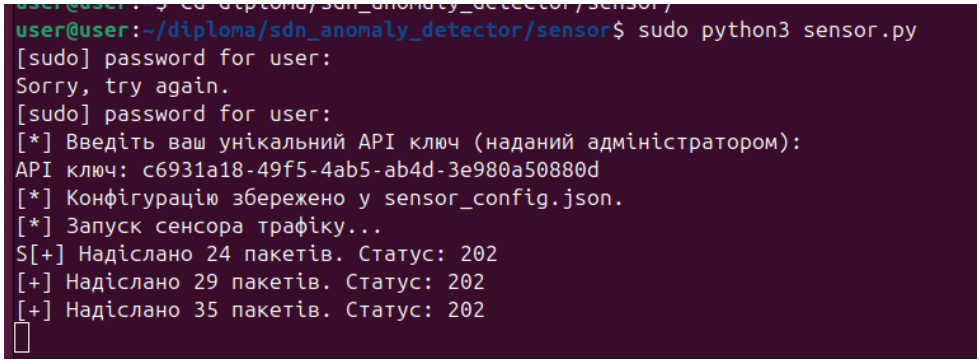


Рисунок 4.1 – Інтерфейс сенсору.

Інтерфейс хабу (AnomalyHub) має набагато ширший функціонал, що реалізовано як повноцінний веб-додаток із сучасним адаптивним інтерфейсом. Його дизайн побудований з урахуванням принципів UI/UX (User Interface/ User Experience): інтуїтивна навігація, контрастність кольорів, швидке оновлення даних.

Інтерфейс хабу можна поділити на дві основні зони:

- Зона керування користувачем – реєстрація, вхід, налаштування облікового запису(зміна пароля, пошти)

- Зона трафіку та аналітики – призначена для взаємодії з сенсорами та візуалізації даних.

Форма реєстрації (рис. 4.2) налічує валідацію введених користувачем даних, що при помилці виведе повідомлення (некоректна електронна пошта аьо пароль має налічувати 12 символів).

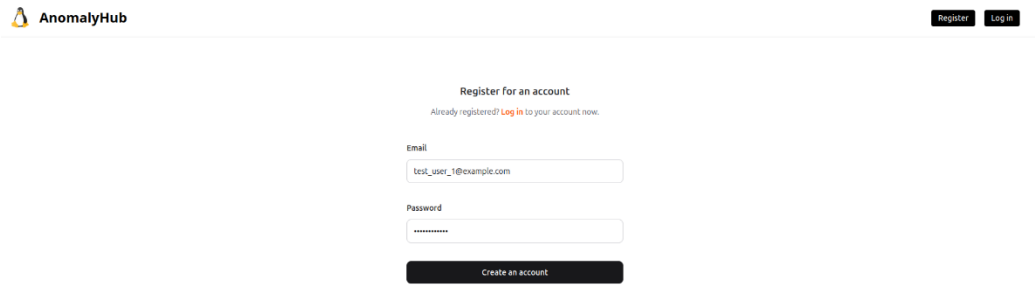


Рисунок 4.2 – Інтерфейс сторінки реєстрації.

Після реєстрації або входу в систему користува потрапляє на головну сторіну де може додати новий сенсор. Для додавання нового сенсору до свого облікового запису користувач натискає кнопку додати та вводить назву нового серсору, отримує унікальний згенерований API_key, що при першому запуску сенсору вводить у консоль (наступні рази програма підвантажує збережений ключ з файлу).

Кожний сенсор знаходиться в табличці, при натискання на рядок відкривається детальна інформація щодо активності сенсору та даних, що були зібрані. Також присутнє випадаюче вікно вибору часу, за який користувач хоче переглянути статистику (рис. 4.3).

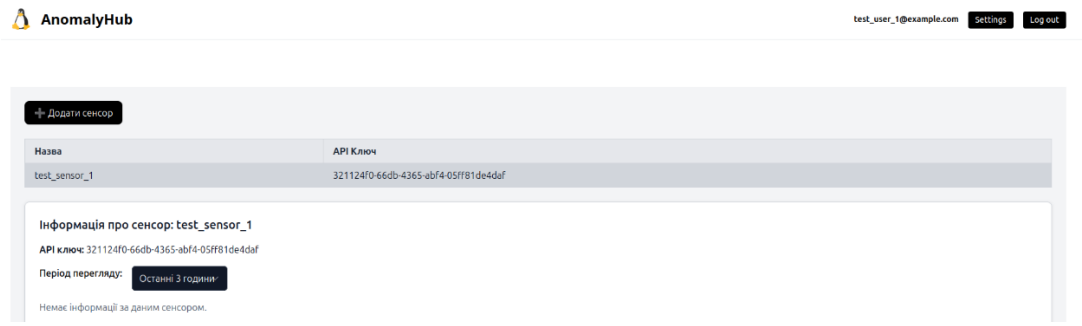


Рисунок 4.3 – Інтерфейс додавання та інформації по сенсору.

Детальна інформація про роботу сенсора реалізована у вигляді графіків що оновлюються кожні 60 секунд:

- графіку трафіку та аномалій – показує кількість отриманих пакетів у реальному часі та кількість виявлених аномалбних записів (рис. 4.4);

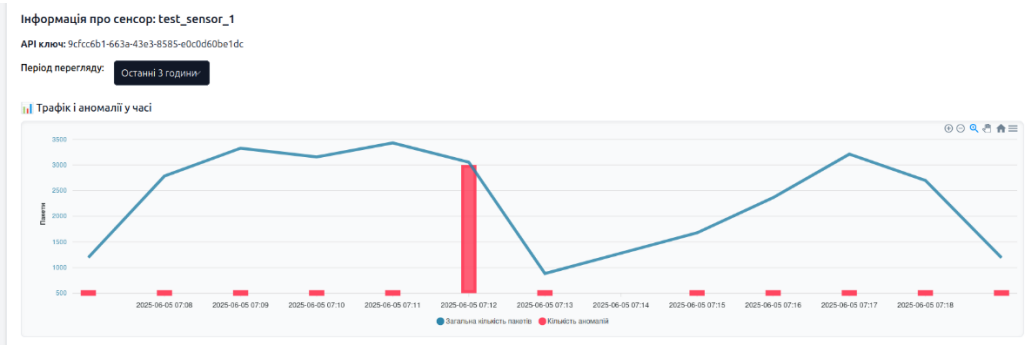


Рисунок 4.4 – Графік трафіку та аномалій.

- графіку типів пакетів – дозволяє оцінити який протокол домінує (TCP, UDP, ICMP) (рис. 4.5);

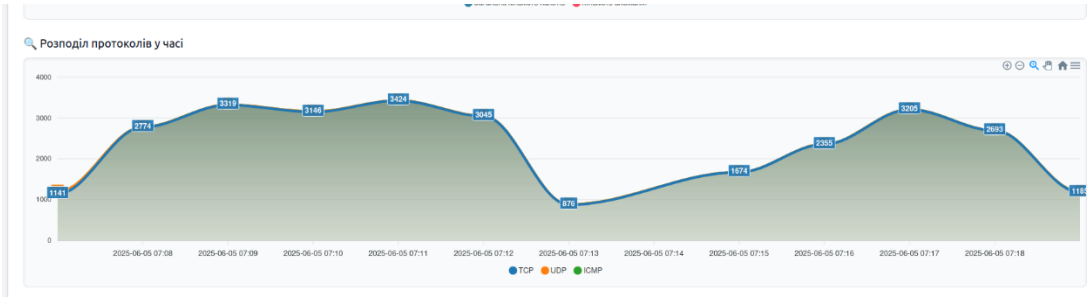


Рисунок 4.5 – Графік кількості отриманих пакетів відповідно до протоколу.

- гістограми цільових портів – допомагає виявити, на які порти було здійснено найбільше звернень (рис. 4.6);

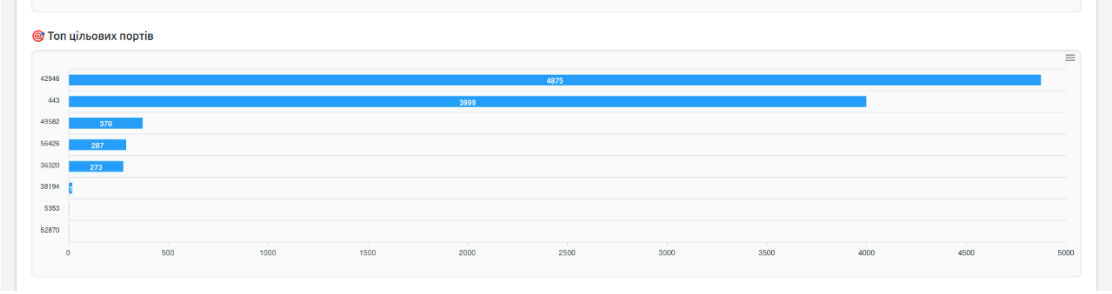


Рисунок 4.6 – Графік цільових портів.

- графік аномалій – ілюструє час та алгоритм виявлення певної підозрілої активності (рис. 4.7);

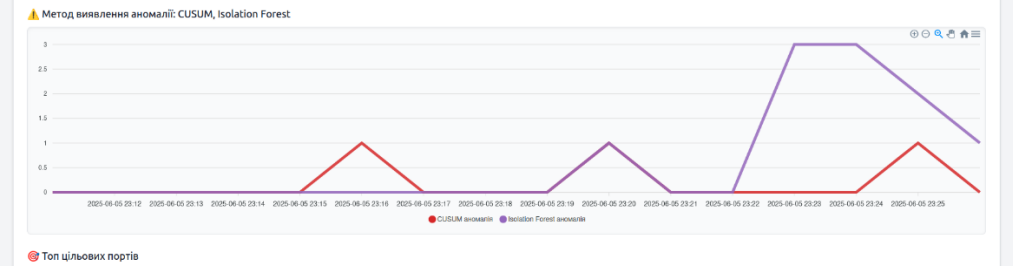


Рисунок 4.7 – Графік виявлених у час аномалій відповідно до методу.

- графік унфкальних IP-адрес – показує загальну кількість взаємодіючих пристроїв у мережі, що є корисним індикатором про ботнет0фтфках (рис. 4.8).

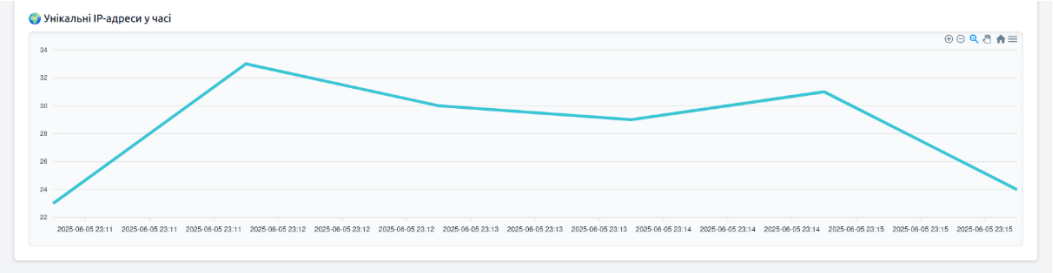


Рисунок 4.8 – Графік виявлених унікальних ip-адрес.

4.2 Огляд бази даних додатку

Для збереження структурованої інформації про користувачів, сенсори та записи трафіку було обрано СКБД PostgreSQL, адже вона відзнаяається високою надійністю та широким функціоналом аналітичних запитів. Також до переваг PostgreSQL можна віднести:

- підтримка транзакцій ACID (Atomicity, Consistency, Isolation, Durability);
- широкий набір функцій для роботи з JSON, часовими рядами, агрегацією.
- Висока масштабованість та оптимізація запитів;
- Розширення для обробки телеметричних даних (TimescaleDB).

База даних має чітку логічну структуру та містить 3 основні таблиці Users, Sensors та TrafficRecords та 1 домоміжну UsersTokens.

Таблиця Users зберігає облікові дані користувача системи та має таку структуру описану в таблиці 4.1.

Таблиця 4.1 – Структура полів таблиці Users.

Назва поля	Тип Даних	Опис
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор користувача в системі
email	VARCHAR UNIQUE NOT NULL	Адреса електроннох пошти користувача.
hashed_password	VARCHAR NOT NULL	Хеш паролю (збережений із використанням bcrypt)
confirmed_at	TIMESTAMP	Час підтвердження облікового запису за допомогою електронного листа
updated_at	TIMESTAMP	Час останньох зміни в обліковому записі
inserted_at	TIMESTAMP	Час створення облікового запису

Таблиця UsersToken зберігає токени користувача для скидання паролю чи зміни пошти чи її підтвердження та має структуру описану в таблиці 4.2.

Таблиця 4.2 – Структура полів таблиці UsersTokens.

Назва поля	Тип Даних	Опис
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор токена
user_id	INTEGER NOT NULL REFERENCES Users(id)	Посилання на відповідного користувача.
token	VARCHAR NOT NULL	Хеш токена (збережений із використанням bcrypt)
context	VARCHAR	Контексти призначення токену
sent_to	VARCHAR	Електроння пошта на яку було надіслано токен.
inserted_at	TIMESTAMP	Час створення створення токену.

Таблиця Sensors зберігає назву сенсору та його API ключ, також має прив’язку до користувача та має структуру описану в таблиці 4.3.

Таблиця 4.3 – Структура полів таблиці Sensors.

Назва поля	Тип Даних	Опис
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор сенсору.
user_id	INTEGER NOT NULL REFERENCES Users(id)	Посилання на відповідного користувача.
name	VARCHAR	Назва сенсору.
api_key	VARCHAR NOT NULL	Ключ сенсору.
updated_at	TIMESTAMP	Час остянньої зміни в сутності сенсору.
inserted_at	TIMESTAMP	Час створення створення сенсору.

Таблиця TrafficRecords зберігає метрики мережевого трафіку від кожного сенсору у певному часовому вікні (5 секунд). Ці записи використовуються для візуалізації трафіку для користувача та має структуру описану в таблиці 4.4.

Таблиця 4.4 – Структура полів таблиці Sensors.

Назва поля	Тип Даних	Опис
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор запису.
sensor_id	INTEGER NOT NULL REFERENCES Sensors(id)	Посилання на відповідний сенсор.
Timestamp	INTEGER	Час зібраного трафіку.
packet_count	INTEGER	Кількість отриманих пакетів за період.
unique_src_ip_count	INTEGER	Кількість унікальних ip-адрес у потоці трафіку.
proto_counter	JSONB	Кільківсть пакетів за протоколами.
tcp_syn_count	INTEGER	Кількість SYN-пакетів.

Кінець таблиці 4.4.

Назва поля	Тип Даних	Опис
top_dst_ports	ARRAY[]::INTEGER	Масив портів, що отримали найбільше запитів.
anomaly	BOOL	Прапорець, що показує чи були виявлені яномалії.
cusum_anomaly	BOOL	Працорець методу виявлення CUSUM.
isolation_anomaly	BOOL	Працорець методу виявлення Isolation Forest.
inserted_at	TIMESTAMP	Час створення створення запису.
updated_at	TIMESTAMP	Час остянньої зміни в сутності запису.

Зв’язки між таблицями можна переглянути на рисунках 4.9 та 4.10.

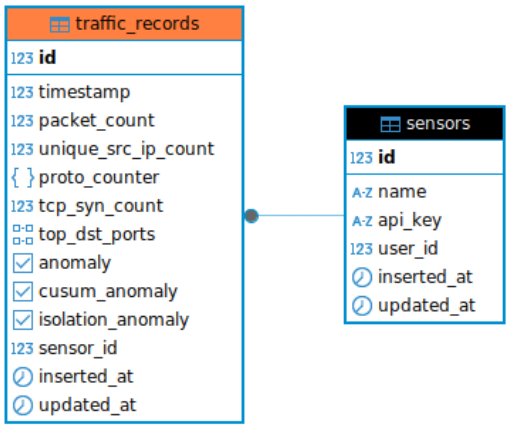


Рисунок 4.9 – Зв’язок таблиць Sensors та TrafficRecords.

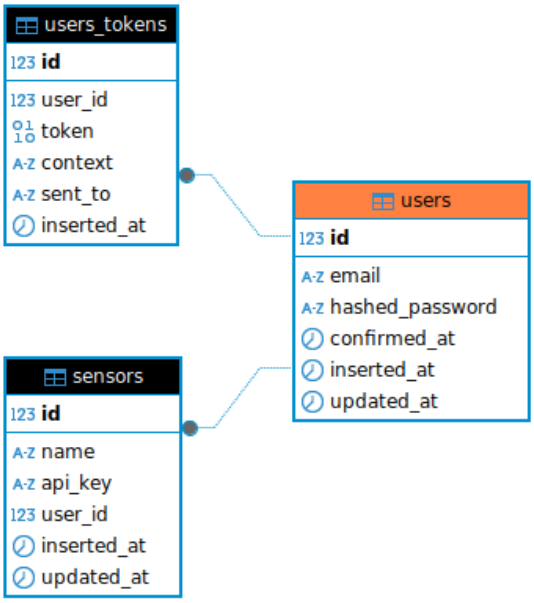


Рисунок 4.10 – Зв’язок таблиць Users та Sensors.

4.3 Тестування розробленої системи

Для тестування розробленої системи було обрано підхід ручного тестування (manual quality assurance) або ж поведінкове тестування (behavior tests). Для цього були змодельовані атаки на мережу: Port scanning, UDP flood, TCP flood. Реалізація скрипту атаки зображена на рисунку 4.11.

```
from scapy.all import IP, TCP, send
import random
import time

target_ip = "192.168.1.102"      # IP цілі (наприклад, твій хост Mininet)
target_port = 80            # Порт цілі (можеш змінити на потрібний)
packet_count = 2000         # Кількість пакетів для відправки
delay = 0.01                # Затримка між пакетами (щоб не задудосити дуже швидко)

print(f"Starting SYN flood attack on {target_ip}:{target_port}...")

for i in range(packet_count):
    ip_layer = IP(src=f"10.0.0.{random.randint(2, 254)}", dst=target_ip)
    tcp_layer = TCP(sport=random.randint(1024, 65535), dport=target_port, flags="S", seq=random.randint(1000, 9000))
    packet = ip_layer / tcp_layer
    send(packet, verbose=False)
    time.sleep(delay)

print("Attack finished.")
```

Рисунок 4.11 – Реалізація простого скрипу атаки на мережу.

Детектор аномалій проявив очікувану поведінку та помітив записи як аномальні (рис. 4.12)

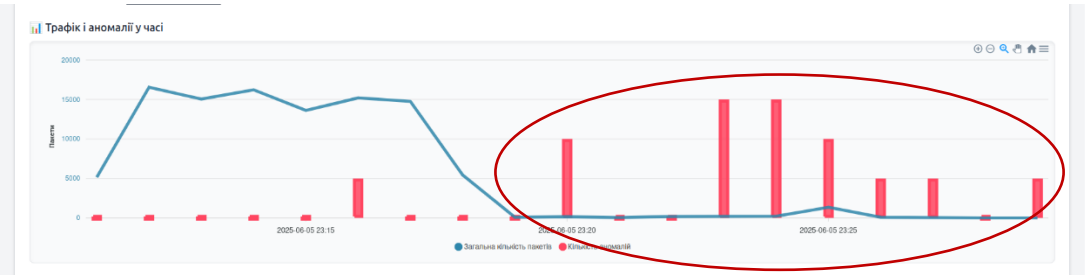


Рисунок 4.12 – Реакція системи на атаку.

Також на графіку можна побачити приклад хибного спрацювання системи (рис. 4.13). Це очікувано відбулось через підвищену кількість пакетів що проходять через мережевий протягом суттєвого часу, що спровокували статистичну аномалію виявлено за допомогою методу CUCUM.

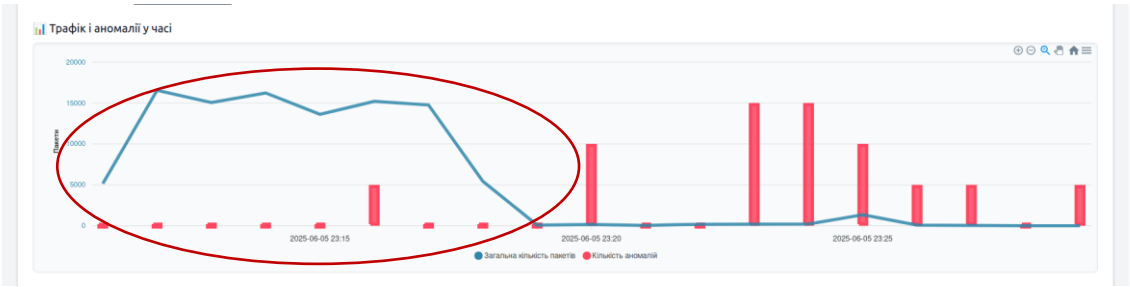


Рисунок 4.13 – Хибне спрацювання детектору аномалій.

Тим часом графік аномалій показує якими методами було виявленно аномалії (рис. 4.14).

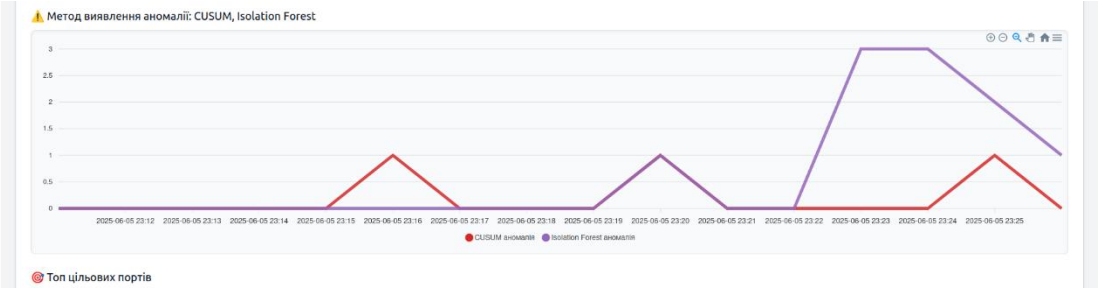


Рисунок 4.14 – Графік аномалій відповідно до методу виявлення.

Виходячи з отриманих результатів можна зробити висновок, що система виконує очікувані функції та сигналізує про появу аномального трафіку, проте може хибно спрацьовувати. Покращити роботу системи можна за допомогою використання більш глибоких методів аналізу.

4.4 Рекомендації щодо розвитку та вдосконалення додатку

Існує щонайменше 3 напрями покращення додатку: покращення сенсора, модуля та хабу.

Для покращення сенсору доцільно підключити складні бібліотеки для аналізу вмісту пакетів (payload) та flow-level statistics (NetFlow, sFlow). Для покращення захищеності з'єднання між сенсорами та хабом доцільно ввести шифрування повідомлень або генерацію підпису за допомогою RSA ключів.

Покращення модуля виявлення аномалій може розвиватись як горизонтально так і вертикально. Це можу бути збільшення кількості різних методів виявлення аномалій: моделі Autoencoder, SVM або поглиблення вже імплементованих: навчання моделі на великих об'ємах типового та атипового трафіку отриманих з пристроїв або знайдених у вільному доступі. Також для покращення захищеності модуля виявлення аномалій є додавання до чорного списку усіх ip-адрес, окрім хабу, що дає змогу не перейматися про безпеку модулю.

З точки зору масштабованості доцільно винести обробку запитів від сенсорів у окрему чергу (через Redis, RabbitMQ або Kafka), що дозволить розділити прийом даних та їх подальшу обробку, зменшити навантаження на основний процес та забезпечити буферизацію при пікових навантаженнях. Для досягнення високої відмовостійкості та можливості горизонтального масштабування хабу можна використати OTP-механізми мови Elixir (супервізори, кластери) або інтегрувати систему з Kubernetes, що дозволить автоматично масштабувати інстанси хабу залежно від навантаження.

Функціональність хабу також можна покращити шляхом впровадження гнучкої системи налаштувань для кожного сенсора окремо: зміна частоти надсилання даних, фільтрування трафіку за IP чи портами тощо. Крім того, доцільно реалізувати історію взаємодії з кожним сенсором та статистику стану його підключення для кращого контролю.

На рівні інтерфейсу адміністратора варто додати можливість перегляду результатів аналізу в динаміці: графіки, теплові карти, порівняння активності в різні періоди. Це дозволить виявляти аномалії не лише у моменті, але й відстежувати довгострокові тенденції.

Важливим елементом системи можестати система реагування на аномалії що може включати в себе блокування підозрілих потоків, додавання ір-адрес що провокують аномалії у чорний список, сповіщення користувача про виявлену аномалію.

ВИСНОВОК ДО РОЗДІЛУ 4

У даному розділі було проаналізовано реалізацію та функціональні можливості розробленої системи виявлення аномального трафіку в SDN-мережах. Було здійснено огляд користувацького інтерфейсу, структуру бази даних.

У межах тестування було проведено ручну перевірку системи шляхом моделювання кількох сценаріїв мережевих атак, що дозволило оцінити працездатність сенсора, взаємодію з хабом та реакцію модуля виявлення аномалій.

Також було сформульовано практичні рекомендації для подальшого розвитку системи в трьох основних напрямках: удосконалення сенсора, хабу та модуля аналізу трафіку. Запропоновано конкретні шляхи інтеграції нових технологій, розширення функціоналу, а також підвищення рівня безпеки та надійності системи.

ВИСНОВКИ

У рамках виконання дипломної роботи було створено інтелектуальну систему для виявлення аномального трафіку в SDN-мережах. Ця система спрямована на ідентифікацію потенційно небажаних чи шкідливих дій у мережі в реальному часі. Вона оснащена аналітичним інтерфейсом і має автоматизовані інструменти оцінки трафіку.

Перший розділ був присвячений аналізу архітектури SDN-мереж та основних загроз, характерних для таких середовищ. Також розглянуто сучасні підходи до виявлення аномалій, зокрема алгоритми машинного навчання.

У другому розділі були обрані й детально описані технології, використані для розробки системи виявлення аномалій трафіку. Здійснено аналіз їхньої доцільності, визначено переваги та недоліки.

Третій розділ зосереджувався на створенні окремих компонентів системи: сенсора, сервер-хабу та мікросервісу аналізу трафіку. Розкрито механізм збору даних із мережевого інтерфейсу, передачу цієї інформації до серверної частини та застосування методів для ідентифікації аномалій. Реалізація системи базувалася на сучасних інструментах, таких як Python, Scapy, FastAPI, Elixir.

Четвертий розділ висвітлює функціональні аспекти користувацького інтерфейсу та організацію бази даних. Було проведено ручне тестування системи для виявлення аномальної активності в змодельованих умовах роботи мережі. Також запропоновано напрямки подальшого вдосконалення системи, включаючи розширення функціоналу сенсора, покращення якості моделей аналізу та підвищення рівня безпеки серверної частини.

Таким чином, мета дипломної роботи була успішно досягнута. Розроблена система продемонструвала ефективність у взаємодії з мережею, аналізі трафіку та виявленні аномалій. Її модульна архітектура

дозволяє легко масштабувати рішення та адаптувати його до реальних умов експлуатації, забезпечуючи відповідність зростаючим вимогам безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дакова Л.В., Даков С.Ю. Аналіз концепції програмно-конфігурованих мереж та протоколу OpenFlow. Наукові записки ДУТ №2(4). 2023. ISSN 2786-8362. ст. 66-72.
2. Pecha P. Science of Network Anomalies - Progress Flowmon. Progress Blogs. URL: https://www.progress.com/blogs/science-of-network-anomalies (дата звернення: 14.05.2025).
3. Wei Lu, Tong Hengjian Detecting Network Anomalies Using CUSUM and EM Clustering. Conference Paper in Lecture Notes in Computer Science. 2009. pg. 302-303.
4. Hasan Torabi, Seyedeh Leili Mirtaheri, Sergio Greco. Practical autoencoder based anomaly detection by using vector reconstruction error. Cybersecurity. pg. 4-5. 2023.
5. Jianheng Tang, Jiajin Li, Ziqi Gao, Jia Li. Rethinking Graph Neural Networks for Anomaly Detection. Proceedings of the 39th International Conference on Machine Learning. 2022.
6. Elixir: scalability and concurrency in an elegant syntax | LLlnformatics. LLInformatics | Software Development and Product Consulting. URL: https://www.llinformatics.com/blog/elixir-programming-language (дата звернення: 14.05.2025).
7. What is Elixir? - Erlang Solutions. Erlang Solutions. URL: https://www.erlang-solutions.com/blog/what-is-elixir (дата звернення: 14.05.2025).
8. Inc B. C. Mastering Elixir Processes: A Deep Dive for Software Engineers. Medium. URL: https://bluetickconsultants.medium.com/mastering-elixir-processes-a-deep-dive-for-software-engineers-dfb7ddfbb283 (дата звернення: 14.05.2025).

9. Contributors to Wikimedia projects. scikit-learn - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Scikit-learn (дата звернення: 14.05.2025).
10. Phoenix Framework. Phoenix Framework. URL: https://www.phoenixframework.org/ (date of access: 06.06.2025).
11. GitHub - elixir-ecto/ecto: A toolkit for data mapping and language integrated query. GitHub. URL: https://github.com/elixir-ecto/ecto (дата звернення: 14.05.2025).
12. scikit-learn: machine learning in Python – scikit-learn 0.16.1 documentation. scikit-learn: machine learning in Python – scikit-learn 0.16.1 documentation. URL: https://scikit-learn.org (дата звернення: 14.05.2025).
13. PostgreSQL. PostgreSQL. URL: https://www.postgresql.org/ (дата звернення: 14.05.2025).
14. Contributors to Wikimedia projects. PostgreSQL - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/PostgreSQL (дата звернення: 14.05.2025).
15. TIOBE Index - TIOBE. TIOBE. URL: https://www.tiobe.com/tiobe-index/ (дата звернення: 14.05.2025).
16. Contributors to Wikimedia projects. Python (programming language) - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Python_(programming_language) (дата звернення: 14.05.2025).
17. Що таке rest api: основні принципи та практики застосування. FoxmindEd. URL: https://foxminded.ua/shcho-take-rest-api/ (дата звернення: 16.05.2025).

ДОДАТОК 1

Методи виявлення аномалій трафіку SDN мережах

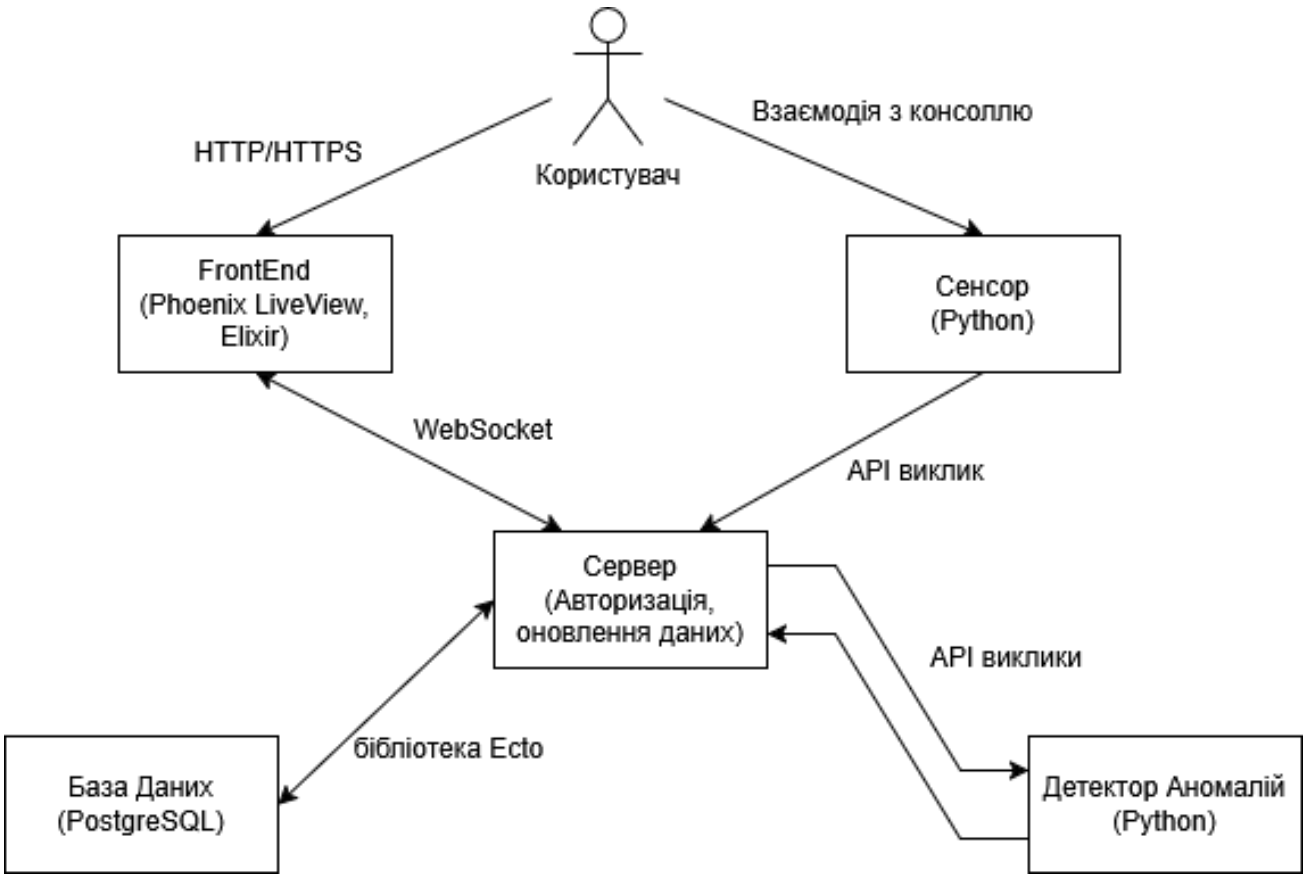
Діаграма залежності компонентів системи

(Стуктурна схема)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.04 Д1	
		№ докум.	Підпис	Дата		
Розробив		Козленко Є. Ю.			**Методи виявлення аномалій трафіку в SDN мережах** Діаграма залежності компонентів системи (структурна схема)	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Коренко Д. В.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-16
Н. Контр.		Міщенко Л. Д.				
Затвердив						

ДОДАТОК 2

Методи виявлення аномалій трафіку SDN мережах

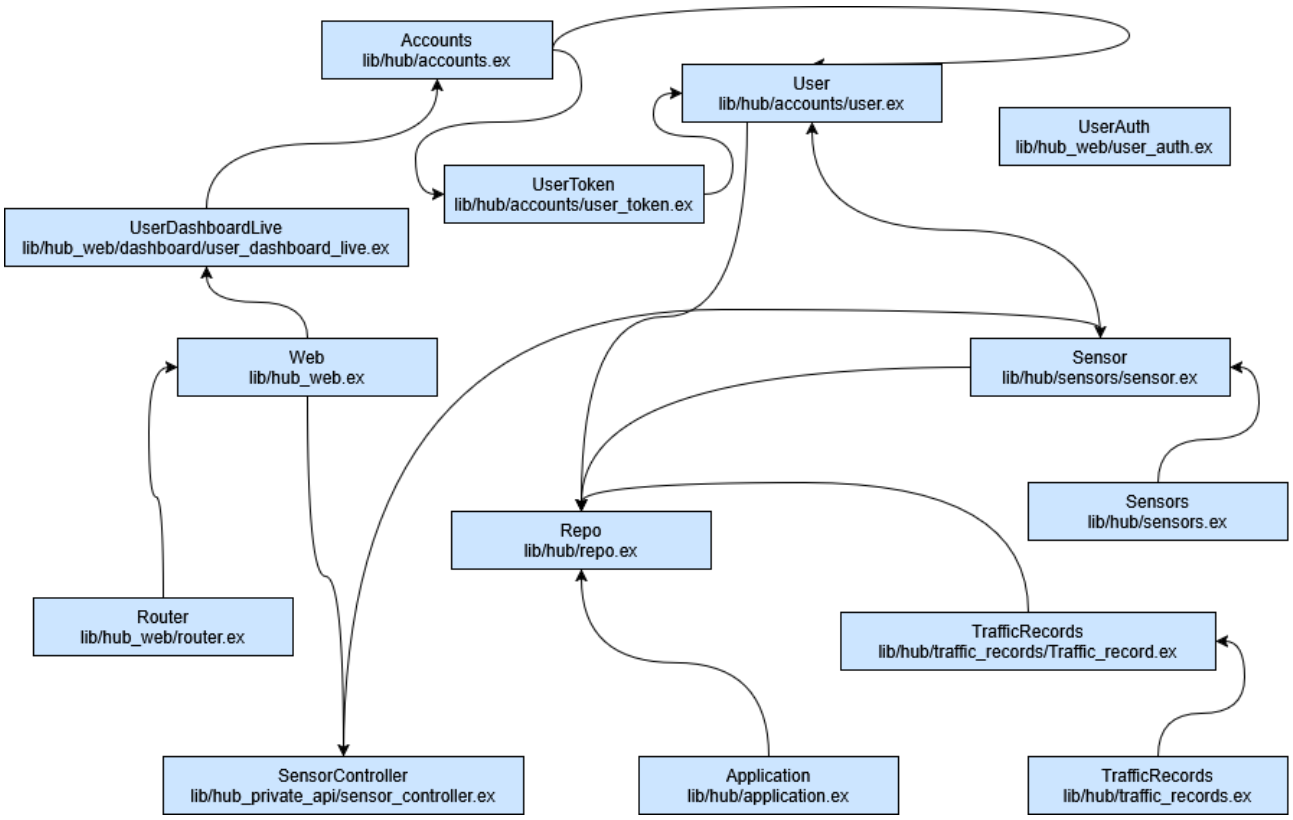
Діаграма залежності модулів серверу

(Функціональна схема)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.05 Д2		
		№ докум.	Підпис	Дата			
Розробив		Козленко Є. Ю.			**Методи виявлення аномалій трафіку в SDN мережах** Діаграма залежності модулів серверу (функціональна схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Коренко Д. В.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-16	
Н. Контр.		Міщенко Л. Д.					
Затвердив							

ДОДАТОК З

Методи виявлення аномалій трафіку SDN мережах

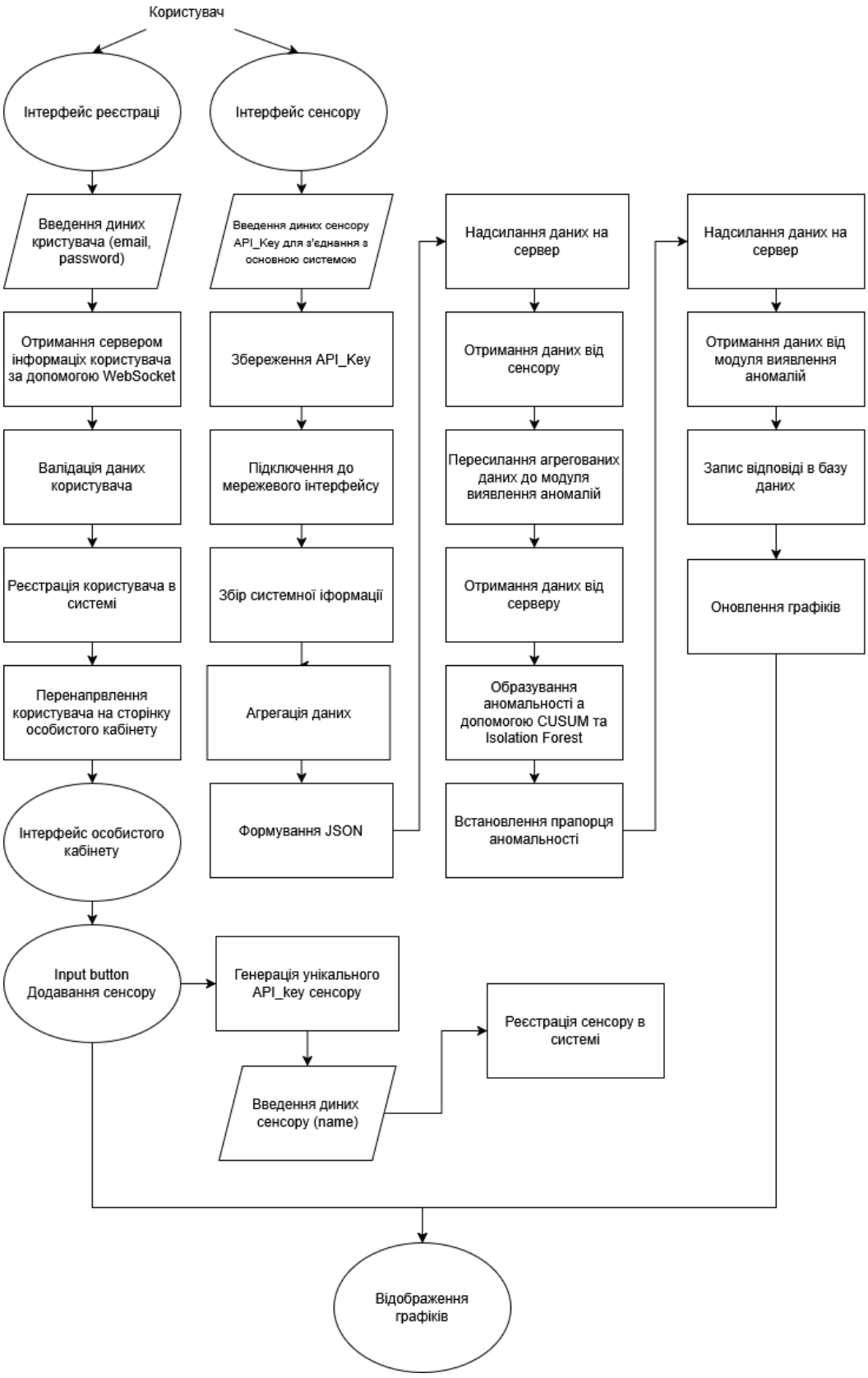
Алгоритм дій користувача та програмного забезпечення

(Принципова схема)

ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.06 Д3		
		№ докум.	Підпис	Дата			
Розробив		Козленко Є. Ю.			**Методи виявлення аномалій трафіку в SDN мережах** Алгоритм дій користувача та програмного задезпечення (принципова схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Коренко Д. В.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-16	
Н. Контр.		Міщенко Л. Д.					
Затвердив							

ДОДАТОК 4

Методи виявлення аномалій трафіку SDN мережах

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 89

Київ 2025 р

```
from fastapi import FastAPI, Request, HTTPException
from pydantic import BaseModel
from typing import List, Dict, Tuple
import numpy as np
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
import os
import joblib
import atexit
import asyncio

app = FastAPI()

MODEL_DIR = "models"
os.makedirs(MODEL_DIR, exist_ok=True)

user_models: Dict[str, IsolationForest] = {}
user_scalers: Dict[str, StandardScaler] = {}
user_buffers: Dict[str, List[List[float]]] = {}
user_cusums: Dict[str, 'CusumDetector'] = {}

MIN_TRAIN_SIZE = 100

class CusumDetector:
    def __init__(self, threshold: float, drift: float = 0.0):
        self.threshold = threshold
        self.drift = drift
        self.pos_sum = 0.0
        self.neg_sum = 0.0
        self.last_mean = None

    def update(self, value: float) -> bool:
        if self.last_mean is None:
            self.last_mean = value
            return False
```

```
        diff = value - self.last_mean - self.drift
        self.pos_sum = max(0, self.pos_sum + diff)
        self.neg_sum = min(0, self.neg_sum + diff)

        if self.pos_sum > self.threshold or abs(self.neg_sum) > self.threshold:
            self.pos_sum = 0
            self.neg_sum = 0
            return True

        return False

class AggregatedTraffic(BaseModel):
    timestamp: int
    packet_count: int
    unique_src_ip_count: int
    proto_counter: Dict[str, int]
    tcp_syn_count: int
    top_dst_ports: List[Tuple[int, int]]

def extract_features_from_aggregate(data: AggregatedTraffic) -> List[float]:
    proto_tcp = data.proto_counter.get("6", 0)
    proto_udp = data.proto_counter.get("17", 0)
    proto_icmp = data.proto_counter.get("1", 0)
    top_ports_count = sum([pair[1] for pair in data.top_dst_ports[:3]])
    return [
        data.packet_count,
        data.unique_src_ip_count,
        data.tcp_syn_count,
        proto_tcp,
        proto_udp,
        proto_icmp,
        top_ports_count
    ]

def get_model_path(user_key: str) -> str:
    return os.path.join(MODEL_DIR, f"{user_key}_model.pkl")

def get_scaler_path(user_key: str) -> str:
    return os.path.join(MODEL_DIR, f"{user_key}_scaler.pkl")

def load_user_model(user_key: str):
    model_path = get_model_path(user_key)
```

```
    scaler_path = get_scaler_path(user_key)
    if os.path.exists(model_path) and os.path.exists(scaler_path):
        user_models[user_key] = joblib.load(model_path)
        user_scalers[user_key] = joblib.load(scaler_path)

async def save_models_async():
    for user_key in user_models:
        joblib.dump(user_models[user_key], get_model_path(user_key))
        joblib.dump(user_scalers[user_key], get_scaler_path(user_key))
    print("[ ✔ ] Models saved asynchronously.")

def save_models_sync():
    asyncio.create_task(save_models_async())

atexit.register(save_models_sync)

@app.on_event("startup")
async def startup_event():
    asyncio.create_task(periodic_model_saver())

async def periodic_model_saver(interval_minutes: int = 10):
    while True:
        await asyncio.sleep(interval_minutes * 60)
        await save_models_async()

@app.post("/api/detect")
async def receive_aggregate(request: Request, payload: AggregatedTraffic):
    auth = request.headers.get("Authorization")
    if not auth or not auth.startswith("Bearer "):
        raise HTTPException(status_code=403, detail="Missing or invalid Authorization header")

    user_key = auth.split()[1]

    feature_vector = extract_features_from_aggregate(payload)

    if user_key not in user_buffers:
        user_buffers[user_key] = []
    user_buffers[user_key].append(feature_vector)

    if user_key not in user_cusums:
        user_cusums[user_key] = CusumDetector(threshold=500.0, drift=10.0)
    cusum_anomaly = user_cusums[user_key].update(payload.tcp_syn_count)
```

```
    if user_key not in user_models:
        load_user_model(user_key)

    if user_key not in user_models and len(user_buffers[user_key]) >= MIN_TRAIN_SIZE:
        data = np.array(user_buffers[user_key])
        scaler = StandardScaler()
        scaled_data = scaler.fit_transform(data)
        model = IsolationForest(contamination=0.05, random_state=42)
        model.fit(scaled_data)
        user_models[user_key] = model
        user_scalers[user_key] = scaler
        await save_models_async()  # Автоматичне збереження після тренування
        return {**payload.dict(), "anomaly": False, "message": "Model trained"}

    isolation_anomaly = False
    if user_key in user_models:
        scaled = user_scalers[user_key].transform([feature_vector])
        pred = user_models[user_key].predict(scaled)[0]
        isolation_anomaly = (pred == -1)

    is_anomaly = cusum_anomaly or isolation_anomaly

    return {
        **payload.dict(),
        "anomaly": bool(is_anomaly),
        "cusum_anomaly": bool(cusum_anomaly),
        "isolation_anomaly": bool(isolation_anomaly)
    }
```

```
# Вихідне Python середовище
FROM python:3.11-slim

# Уникаємо питань при встановленні бібліотек
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# Створення директорії для коду
WORKDIR /app

# Копіюємо файли проєкту
COPY . .
```

```
# Встановлення залежностей
RUN pip install --no-cache-dir -r requirements.txt

# Створення директорії для моделей (на всяк випадок)
RUN mkdir -p models

# Відкриваємо порт (за замовчуванням uvicorn — 8000)
EXPOSE 8000

# Команда запуску FastAPI
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

import scapy.all as scapy
import requests
import json
import time
import threading
import os
from collections import Counter
from dotenv import load_dotenv

CONFIG_FILE = "sensor_config.json"
SEND_INTERVAL = 5
lock = threading.Lock()

# Агреговані дані
stats = {
    "packet_count": 0,
    "unique_src_ips": set(),
    "proto_counter": Counter(),
    "tcp_syn_count": 0,
    "dst_ports": Counter()
}

# Завантаження .env
load_dotenv()
SERVER_URL = os.getenv("SERVER_URL")

if not SERVER_URL:
    raise RuntimeError("[!] SERVER_URL не знайдено у .env файлі!")
```

```
def load_or_create_config():
    if os.path.exists(CONFIG_FILE):
        with open(CONFIG_FILE, 'r') as f:
            config = json.load(f)
            print("[*] Конфігурацію сенсора завантажено.")
    else:
        print("[*] Введіть ваш унікальний API ключ (наданий адміністратором):")
        api_key = input("API ключ: ").strip()
        config = {"api_key": api_key}
        with open(CONFIG_FILE, 'w') as f:
            json.dump(config, f)
            print(f"[*] Конфігурацію збережено у {CONFIG_FILE}.")
    return config

def packet_callback(packet):
    if packet.haslayer(scapy.IP):
        with lock:
            stats["packet_count"] += 1
            stats["unique_src_ips"].add(packet[scapy.IP].src)

            proto = packet[scapy.IP].proto
            stats["proto_counter"][proto] += 1

            if packet.haslayer(scapy.TCP):
                tcp_layer = packet[scapy.TCP]
                # Перевірка лише SYN флагу
                if tcp_layer.flags == "S":
                    stats["tcp_syn_count"] += 1
                stats["dst_ports"][tcp_layer.dport] += 1

            elif packet.haslayer(scapy.UDP):
                udp_layer = packet[scapy.UDP]
                stats["dst_ports"][udp_layer.dport] += 1

def send_stats(config):
    while True:
        time.sleep(SEND_INTERVAL)
        with lock:
            payload = {
                "timestamp": int(time.time()),
                "packet_count": stats["packet_count"],
                "unique_src_ip_count": len(stats["unique_src_ips"]),
```

```
            "proto_counter": dict(stats["proto_counter"]),
            "tcp_syn_count": stats["tcp_syn_count"],
            "top_dst_ports": stats["dst_ports"].most_common(10)
        }

        headers = {
            "Authorization": f"Bearer {config['api_key']}",
            "Content-Type": "application/json"
        }

        try:
            response = requests.post(SERVER_URL, json=payload, headers=headers)
            print(f"[+] Надіслано {payload['packet_count']} пакетів. Статус: {response.status_code}")
        except Exception as e:
            print(f"[!] Помилка надсилання: {e}")

        # Очистка
        stats["packet_count"] = 0
        stats["unique_src_ips"].clear()
        stats["proto_counter"].clear()
        stats["tcp_syn_count"] = 0
        stats["dst_ports"].clear()

def main():
    config = load_or_create_config()
    print("[*] Запуск сенсора трафіку...")
    threading.Thread(target=send_stats, args=(config,), daemon=True).start()
    scapy.sniff(prn=packet_callback, store=False)

if __name__ == "__main__":
    main()



# This file is responsible for configuring your application
# and its dependencies with the aid of the Config module.
#
# This configuration file is loaded before any dependency and
# is restricted to this project.

# General application configuration
import Config
```

```
config :hub,
  ecto_repos: [Hub.Repo],
  generators: [timestamp_type: :utc_datetime]

# Configures the endpoint
config :hub, HubWeb.Endpoint,
  url: [host: "localhost"],
  adapter: Bandit.PhoenixAdapter,
  render_errors: [
    formats: [html: HubWeb.ErrorHTML, json: HubWeb.ErrorJSON],
    layout: false
  ],
  pubsub_server: Hub.PubSub,
  live_view: [signing_salt: "VpHRteUD"]

# Configures the mailer
#
# By default it uses the "Local" adapter which stores the emails
# locally. You can see the emails in your browser, at "/dev/mailbox".
#
# For production it's recommended to configure a different adapter
# at the `config/runtime.exs`.
config :hub, Hub.Mailer, adapter: Swoosh.Adapters.Local

# Configure esbuild (the version is required)
config :esbuild,
  version: "0.17.11",
  hub: [
    args:
      ~w(js/app.js --bundle --target=es2017 --outdir=../priv/static/assets --external:/fonts/* --external:/images/*),
    cd: Path.expand("../assets", __DIR__),
    env: %{"NODE_PATH" => Path.expand("../deps", __DIR__)}
  ]

# Configure tailwind (the version is required)
config :tailwind,
  version: "3.4.3",
  hub: [
    args: ~w(
      --config=tailwind.config.js
      --input=css/app.css
      --output=../priv/static/assets/app.css
```

```
    ),
    cd: Path.expand("../assets", __DIR__)
  ]

# Configures Elixir's Logger
config :logger, :console,
  format: "$time $metadata[$level] $message\n",
  metadata: [:request_id]

# Use Jason for JSON parsing in Phoenix
config :phoenix, :json_library, Jason

# Import environment specific config. This must remain at the bottom
# of this file so it overrides the configuration defined above.
import_config "#{config_env()}.exs"

import Config

# Configure your database
config :hub, Hub.Repo,
  username: "postgres",
  password: "postgres",
  hostname: "localhost",
  database: "hub_dev",
  stacktrace: true,
  show_sensitive_data_on_connection_error: true,
  pool_size: 10

# For development, we disable any cache and enable
# debugging and code reloading.
#
# The watchers configuration can be used to run external
# watchers to your application. For example, we can use it
# to bundle .js and .css sources.
config :hub, HubWeb.Endpoint,
  # Binding to loopback ipv4 address prevents access from other machines.
  # Change to `ip: {0, 0, 0, 0}` to allow access from other machines.
  http: [ip: {127, 0, 0, 1}, port: 4000],
  check_origin: false,
  code_reloader: true,
  debug_errors: true,
  secret_key_base: "2LCfI7UCbSk7ZahLAqhtEivp+suOuovQsiUohIJ0u+NpfoyTQAEQm+p41zyv/njM",
```

```
  watchers: [
    esbuild: {Esbuild, :install_and_run, [:hub, ~w(--sourcemap=inline --watch)]},
    tailwind: {Tailwind, :install_and_run, [:hub, ~w(--watch)]}
  ]

# ## SSL Support
#
# In order to use HTTPS in development, a self-signed
# certificate can be generated by running the following
# Mix task:
#
#     mix phx.gen.cert
#
# Run `mix help phx.gen.cert` for more information.
#
# The `http:` config above can be replaced with:
#
#     https: [
#       port: 4001,
#       cipher_suite: :strong,
#       keyfile: "priv/cert/selfsigned_key.pem",
#       certfile: "priv/cert/selfsigned.pem"
#     ],
#
# If desired, both `http:` and `https:` keys can be
# configured to run both http and https servers on
# different ports.

# Watch static and templates for browser reloading.
config :hub, HubWeb.Endpoint,
  live_reload: [
    patterns: [
      ~r"priv/static/(?!uploads/).*(js|css|png|jpeg|jpg|gif|svg)$",
      ~r"priv/gettext/.*(po)$",
      ~r"lib/hub_web/(controllers|live|components)/.*(ex|heex)$"
    ]
  ]

# Enable dev routes for dashboard and mailbox
config :hub, dev_routes: true

# Do not include metadata nor timestamps in development logs
```

```
config :logger, :console, format: "[$level] $message\n"

# Set a higher stacktrace during development. Avoid configuring such
# in production as building large stacktraces may be expensive.
config :phoenix, :stacktrace_depth, 20

# Initialize plugs at runtime for faster development compilation
config :phoenix, :plug_init_mode, :runtime

config :phoenix_live_view,
  # Include HEEx debug annotations as HTML comments in rendered markup
  debug_heex_annotations: true,
  # Enable helpful, but potentially expensive runtime checks
  enable_expensive_runtime_checks: true

# Disable swoosh api client as it is only required for production adapters.
config :swoosh, :api_client, false
import Config

# Note we also include the path to a cache manifest
# containing the digested version of static files. This
# manifest is generated by the `mix assets.deploy` task,
# which you should run after static files are built and
# before starting your production server.
config :hub, HubWeb.Endpoint, cache_static_manifest: "priv/static/cache_manifest.json"

# Configures Swoosh API Client
config :swoosh, api_client: Swoosh.ApiClient.Finch, finch_name: Hub.Finch

# Disable Swoosh Local Memory Storage
config :swoosh, local: false

# Do not print debug messages in production
config :logger, level: :info

# Runtime production configuration, including reading
# of environment variables, is done on config/runtime.exs.
import Config

# config/runtime.exs is executed for all environments, including
# during releases. It is executed after compilation and before the
# system starts, so it is typically used to load production configuration
```

```
# and secrets from environment variables or elsewhere. Do not define
# any compile-time configuration in here, as it won't be applied.
# The block below contains prod specific runtime configuration.

# ## Using releases
#
# If you use `mix release`, you need to explicitly enable the server
# by passing the PHX_SERVER=true when you start it:
#
#     PHX_SERVER=true bin/hub start
#
# Alternatively, you can use `mix phx.gen.release` to generate a `bin/server`
# script that automatically sets the env var above.
if System.get_env("PHX_SERVER") do
  config :hub, HubWeb.Endpoint, server: true
end

if config_env() == :prod do
  database_url =
    System.get_env("DATABASE_URL") ||
      raise """
      environment variable DATABASE_URL is missing.
      For example: ecto://USER:PASS@HOST/DATABASE
      """

  maybe_ipv6 = if System.get_env("ECTO_IPV6") in ~w(true 1), do: [:inet6], else: []

  config :hub, Hub.Repo,
    # ssl: true,
    url: database_url,
    pool_size: String.to_integer(System.get_env("POOL_SIZE") || "10"),
    socket_options: maybe_ipv6

  # The secret key base is used to sign/encrypt cookies and other secrets.
  # A default value is used in config/dev.exs and config/test.exs but you
  # want to use a different value for prod and you most likely don't want
  # to check this value into version control, so we use an environment
  # variable instead.
  secret_key_base =
    System.get_env("SECRET_KEY_BASE") ||
      raise """
      environment variable SECRET_KEY_BASE is missing.
```

```
    You can generate one by calling: mix phx.gen.secret
    """

host = System.get_env("PHX_HOST") || "example.com"
port = String.to_integer(System.get_env("PORT") || "4000")

config :hub, :dns_cluster_query, System.get_env("DNS_CLUSTER_QUERY")

config :hub, HubWeb.Endpoint,
  url: [host: host, port: 443, scheme: "https"],
  http: [
    # Enable IPv6 and bind on all interfaces.
    # Set it to  {0, 0, 0, 0, 0, 0, 0, 1} for local network only access.
    # See the documentation on https://hexdocs.pm/bandit/Bandit.html#t:options/0
    # for details about using IPv6 vs IPv4 and loopback vs public addresses.
    ip: {0, 0, 0, 0, 0, 0, 0, 0},
    port: port
  ],
  secret_key_base: secret_key_base

# ## SSL Support
#
# To get SSL working, you will need to add the `https` key
# to your endpoint configuration:
#
#     config :hub, HubWeb.Endpoint,
#       https: [
#         ...,
#         port: 443,
#         cipher_suite: :strong,
#         keyfile: System.get_env("SOME_APP_SSL_KEY_PATH"),
#         certfile: System.get_env("SOME_APP_SSL_CERT_PATH")
#       ]
#
# The `cipher_suite` is set to `:strong` to support only the
# latest and more secure SSL ciphers. This means old browsers
# and clients may not be supported. You can set it to
# `:compatible` for wider support.
#
# `:keyfile` and `:certfile` expect an absolute path to the key
# and cert in disk or a relative path inside priv, for example
# "priv/ssl/server.key". For all supported SSL configuration
```

```
  # options, see https://hexdocs.pm/plug/Plug.SSL.html#configure/1
  #
  # We also recommend setting `force_ssl` in your config/prod.exs,
  # ensuring no data is ever sent via http, always redirecting to https:
  #
  #     config :hub, HubWeb.Endpoint,
  #       force_ssl: [hsts: true]
  #
  # Check `Plug.SSL` for all available options in `force_ssl`.

  # ## Configuring the mailer
  #
  # In production you need to configure the mailer to use a different adapter.
  # Also, you may need to configure the Swoosh API client of your choice if you
  # are not using SMTP. Here is an example of the configuration:
  #
  #     config :hub, Hub.Mailer,
  #       adapter: Swoosh.Adapters.Mailgun,
  #       api_key: System.get_env("MAILGUN_API_KEY"),
  #       domain: System.get_env("MAILGUN_DOMAIN")
  #
  # For this example you need include a HTTP client required by Swoosh API client.
  # Swoosh supports Hackney and Finch out of the box:
  #
  #     config :swoosh, :api_client, Swoosh.ApiClient.Hackney
  #
  # See https://hexdocs.pm/swoosh/Swoosh.html#module-installation for details.
End

import Config

# Only in tests, remove the complexity from the password hashing algorithm
config :bcrypt_elixir, :log_rounds, 1

# Configure your database
#
# The MIX_TEST_PARTITION environment variable can be used
# to provide built-in test partitioning in CI environment.
# Run `mix help test` for more information.
config :hub, Hub.Repo,
  username: "postgres",
  password: "postgres",
```

```
  hostname: "localhost",
  database: "hub_test#{System.get_env("MIX_TEST_PARTITION")}",
  pool: Ecto.Adapters.SQL.Sandbox,
  pool_size: System.schedulers_online() * 2

# We don't run a server during test. If one is required,
# you can enable the server option below.
config :hub, HubWeb.Endpoint,
  http: [ip: {127, 0, 0, 1}, port: 4002],
  secret_key_base: "lU7fW0AyHFbOeElcVM/YrfAL7fo/BONgNOfz+Y/A/ttGfV1on6UkT/lsHgkihv9M",
  server: false

# In test we don't send emails
config :hub, Hub.Mailer, adapter: Swoosh.Adapters.Test

# Disable swoosh api client as it is only required for production adapters
config :swoosh, :api_client, false

# Print only warnings and errors during test
config :logger, level: :warning

# Initialize plugs at runtime for faster test compilation
config :phoenix, :plug_init_mode, :runtime

# Enable helpful, but potentially expensive runtime checks
config :phoenix_live_view,
  enable_expensive_runtime_checks: true

defmodule HubWeb do
  @moduledoc """
  The entrypoint for defining your web interface, such
  as controllers, components, channels, and so on.

  This can be used in your application as:

      use HubWeb, :controller
      use HubWeb, :html

  The definitions below will be executed for every controller,
  component, etc, so keep them short and clean, focused
  on imports, uses and aliases.
```

```
  Do NOT define functions inside the quoted expressions
  below. Instead, define additional modules and import
  those modules here.
  """

  def static_paths, do: ~w(assets fonts images favicon.ico robots.txt)

  def router do
    quote do
      use Phoenix.Router, helpers: false

      # Import common connection and controller functions to use in pipelines
      import Plug.Conn
      import Phoenix.Controller
      import Phoenix.LiveView.Router
    end
  end

  def channel do
    quote do
      use Phoenix.Channel
    end
  end

  def controller do
    quote do
      use Phoenix.Controller,
        formats: [:html, :json],
        layouts: [html: HubWeb.Layouts]

      use Gettext, backend: HubWeb.Gettext

      import Plug.Conn

      unquote(verified_routes())
    end
  end

  def live_view do
    quote do
      use Phoenix.LiveView,
        layout: {HubWeb.Layouts, :app}
```

```
    unquote(html_helpers())
  end
end

def live_component do
  quote do
    use Phoenix.LiveComponent

    unquote(html_helpers())
  end
end

def html do
  quote do
    use Phoenix.Component

    # Import convenience functions from controllers
    import Phoenix.Controller,
      only: [get_csrf_token: 0, view_module: 1, view_template: 1]

    # Include general helpers for rendering HTML
    unquote(html_helpers())
  end
end

defp html_helpers do
  quote do
    # Translation
    use Gettext, backend: HubWeb.Gettext

    # HTML escaping functionality
    import Phoenix.HTML
    # Core UI components
    import HubWeb.CoreComponents

    # Shortcut for generating JS commands
    alias Phoenix.LiveView.JS

    # Routes generation with the ~p sigil
    unquote(verified_routes())
  end
```

```
  end

  def verified_routes do
    quote do
      use Phoenix.VerifiedRoutes,
        endpoint: HubWeb.Endpoint,
        router: HubWeb.Router,
        statics: HubWeb.static_paths()
    end
  end

  @doc """
  When used, dispatch to the appropriate controller/live_view/etc.
  """
  defmacro __using__(which) when is_atom(which) do
    apply(__MODULE__, which, [])
  end
end
defmodule Hub.Accounts do
  @moduledoc """
  The Accounts context.
  """

  import Ecto.Query, warn: false
  alias Hub.Repo

  alias Hub.Accounts.{User, UserToken, UserNotifier}

  ## Database getters

  @doc """
  Gets a user by email.

  ## Examples

      iex> get_user_by_email("foo@example.com")
      %User{}

      iex> get_user_by_email("unknown@example.com")
      nil

  """
```

```
def get_user_by_email(email) when is_binary(email) do
  Repo.get_by(User, email: email)
end

@doc """
Gets a user by email and password.

## Examples

    iex> get_user_by_email_and_password("foo@example.com", "correct_password")
    %User{}

    iex> get_user_by_email_and_password("foo@example.com", "invalid_password")
    nil

"""
def get_user_by_email_and_password(email, password)
    when is_binary(email) and is_binary(password) do
  user = Repo.get_by(User, email: email)
  if User.valid_password?(user, password), do: user
end

@doc """
Gets a single user.

Raises `Ecto.NoResultsError` if the User does not exist.

## Examples

    iex> get_user!(123)
    %User{}

    iex> get_user!(456)
    ** (Ecto.NoResultsError)

"""
def get_user!(id), do: Repo.get!(User, id)

## User registration

@doc """
Registers a user.
```

```
## Examples

    iex> register_user(%{field: value})
    {:ok, %User{}}

    iex> register_user(%{field: bad_value})
    {:error, %Ecto.Changeset{}}

"""
def register_user(attrs) do
  %User{}
  |> User.registration_changeset(attrs)
  |> Repo.insert()
end

@doc """
Returns an `%Ecto.Changeset{}` for tracking user changes.

## Examples

    iex> change_user_registration(user)
    %Ecto.Changeset{data: %User{}}

"""
def change_user_registration(%User{} = user, attrs \\ %{}) do
  User.registration_changeset(user, attrs, hash_password: false, validate_email: false)
end

## Settings

@doc """
Returns an `%Ecto.Changeset{}` for changing the user email.

## Examples

    iex> change_user_email(user)
    %Ecto.Changeset{data: %User{}}

"""
def change_user_email(user, attrs \\ %{}) do
  User.email_changeset(user, attrs, validate_email: false)
```

```
end

@doc """
Emulates that the email will change without actually changing
it in the database.

## Examples

    iex> apply_user_email(user, "valid password", %{email: ...})
    {:ok, %User{}}

    iex> apply_user_email(user, "invalid password", %{email: ...})
    {:error, %Ecto.Changeset{}}

"""
def apply_user_email(user, password, attrs) do
  user
  |> User.email_changeset(attrs)
  |> User.validate_current_password(password)
  |> Ecto.Changeset.apply_action(:update)
end

@doc """
Updates the user email using the given token.

If the token matches, the user email is updated and the token is deleted.
The confirmed_at date is also updated to the current time.
"""
def update_user_email(user, token) do
  context = "change:#{user.email}"

  with {:ok, query} <- UserToken.verify_change_email_token_query(token, context),
       %UserToken{sent_to: email} <- Repo.one(query),
       {:ok, _} <- Repo.transaction(user_email_multi(user, email, context)) do
    :ok
  else
    _ -> :error
  end
end

defp user_email_multi(user, email, context) do
  changeset =
```

```
    user
    |> User.email_changeset(%{email: email})
    |> User.confirm_changeset()

  Ecto.Multi.new()
  |> Ecto.Multi.update(:user, changeset)
  |> Ecto.Multi.delete_all(:tokens, UserToken.by_user_and_contexts_query(user, [context]))
 end

 @doc ~S"""
 Delivers the update email instructions to the given user.

 ## Examples

    iex> deliver_user_update_email_instructions(user, current_email,
&url(~p"/users/settings/confirm_email/#{&1}"))
    {:ok, %{to: ..., body: ...}}

 """
 def deliver_user_update_email_instructions(%User{} = user, current_email, update_email_url_fun)
    when is_function(update_email_url_fun, 1) do
  {encoded_token, user_token} = UserToken.build_email_token(user, "change:#{current_email}")

  Repo.insert!(user_token)
  UserNotifier.deliver_update_email_instructions(user, update_email_url_fun.(encoded_token))
 end

 @doc """
 Returns an `%Ecto.Changeset{}` for changing the user password.

 ## Examples

    iex> change_user_password(user)
    %Ecto.Changeset{data: %User{}}

 """
 def change_user_password(user, attrs \\ %{}) do
  User.password_changeset(user, attrs, hash_password: false)
 end

 @doc """
 Updates the user password.
```

```
## Examples

    iex> update_user_password(user, "valid password", %{password: ...})
    {:ok, %User{}}

    iex> update_user_password(user, "invalid password", %{password: ...})
    {:error, %Ecto.Changeset{}}

"""
def update_user_password(user, password, attrs) do
  changeset =
    user
    |> User.password_changeset(attrs)
    |> User.validate_current_password(password)

  Ecto.Multi.new()
  |> Ecto.Multi.update(:user, changeset)
  |> Ecto.Multi.delete_all(:tokens, UserToken.by_user_and_contexts_query(user, :all))
  |> Repo.transaction()
  |> case do
    {:ok, %{user: user}} -> {:ok, user}
    {:error, :user, changeset, _} -> {:error, changeset}
  end
end

## Session

@doc """
Generates a session token.
"""
def generate_user_session_token(user) do
  {token, user_token} = UserToken.build_session_token(user)
  Repo.insert!(user_token)
  token
end

@doc """
Gets the user with the given signed token.
"""
def get_user_by_session_token(token) do
  {:ok, query} = UserToken.verify_session_token_query(token)
```

```
  Repo.one(query)
end

@doc """
Deletes the signed token with the given context.
"""
def delete_user_session_token(token) do
  Repo.delete_all(UserToken.by_token_and_context_query(token, "session"))
  :ok
end

## Confirmation

@doc ~S"""
Delivers the confirmation email instructions to the given user.

## Examples

    iex> deliver_user_confirmation_instructions(user, &url(~p"/users/confirm/#{&1}"))
    {:ok, %{to: ..., body: ...}}

    iex> deliver_user_confirmation_instructions(confirmed_user, &url(~p"/users/confirm/#{&1}"))
    {:error, :already_confirmed}

"""
def deliver_user_confirmation_instructions(%User{} = user, confirmation_url_fun)
    when is_function(confirmation_url_fun, 1) do
  if user.confirmed_at do
    {:error, :already_confirmed}
  else
    {encoded_token, user_token} = UserToken.build_email_token(user, "confirm")
    Repo.insert!(user_token)
    UserNotifier.deliver_confirmation_instructions(user, confirmation_url_fun.(encoded_token))
  end
end

@doc """
Confirms a user by the given token.

If the token matches, the user account is marked as confirmed
and the token is deleted.
"""
```

```
def confirm_user(token) do
  with {:ok, query} <- UserToken.verify_email_token_query(token, "confirm"),
       %User{} = user <- Repo.one(query),
       {:ok, %{user: user}} <- Repo.transaction(confirm_user_multi(user)) do
    {:ok, user}
  else
    _ -> :error
  end
end

defp confirm_user_multi(user) do
  Ecto.Multi.new()
  |> Ecto.Multi.update(:user, User.confirm_changeset(user))
  |> Ecto.Multi.delete_all(:tokens, UserToken.by_user_and_contexts_query(user, ["confirm"]))
end

## Reset password

@doc ~S"""
Delivers the reset password email to the given user.

## Examples

    iex> deliver_user_reset_password_instructions(user, &url(~p"/users/reset_password/#{&1}"))
    {:ok, %{to: ..., body: ...}}

"""
def deliver_user_reset_password_instructions(%User{} = user, reset_password_url_fun)
    when is_function(reset_password_url_fun, 1) do
  {encoded_token, user_token} = UserToken.build_email_token(user, "reset_password")
  Repo.insert!(user_token)
  UserNotifier.deliver_reset_password_instructions(user, reset_password_url_fun.(encoded_token))
end

@doc """
Gets the user by reset password token.

## Examples

    iex> get_user_by_reset_password_token("validtoken")
    %User{}
```

```
    iex> get_user_by_reset_password_token("invalidtoken")
    nil

  """
  def get_user_by_reset_password_token(token) do
    with {:ok, query} <- UserToken.verify_email_token_query(token, "reset_password"),
         %User{} = user <- Repo.one(query) do
      user
    else
      _ -> nil
    end
  end

  @doc """
  Resets the user password.

  ## Examples

    iex> reset_user_password(user, %{password: "new long password", password_confirmation: "new long password"})
    {:ok, %User{}}

    iex> reset_user_password(user, %{password: "valid", password_confirmation: "not the same"})
    {:error, %Ecto.Changeset{}}

  """
  def reset_user_password(user, attrs) do
    Ecto.Multi.new()
    |> Ecto.Multi.update(:user, User.password_changeset(user, attrs))
    |> Ecto.Multi.delete_all(:tokens, UserToken.by_user_and_contexts_query(user, :all))
    |> Repo.transaction()
    |> case do
      {:ok, %{user: user}} -> {:ok, user}
      {:error, :user, changeset, _} -> {:error, changeset}
    end
  end
end
defmodule Hub.Accounts do
  @moduledoc """
  The Accounts context.
  """
```

```
import Ecto.Query, warn: false
alias Hub.Repo

alias Hub.Accounts.{User, UserToken, UserNotifier}

## Database getters

@doc """
Gets a user by email.

## Examples

    iex> get_user_by_email("foo@example.com")
    %User{}

    iex> get_user_by_email("unknown@example.com")
    nil

"""
def get_user_by_email(email) when is_binary(email) do
  Repo.get_by(User, email: email)
end

@doc """
Gets a user by email and password.

## Examples

    iex> get_user_by_email_and_password("foo@example.com", "correct_password")
    %User{}

    iex> get_user_by_email_and_password("foo@example.com", "invalid_password")
    nil

"""
def get_user_by_email_and_password(email, password)
    when is_binary(email) and is_binary(password) do
  user = Repo.get_by(User, email: email)
  if User.valid_password?(user, password), do: user
end

@doc """
```

```
  Gets a single user.

  Raises `Ecto.NoResultsError` if the User does not exist.

  ## Examples

      iex> get_user!(123)
      %User{}

      iex> get_user!(456)
      ** (Ecto.NoResultsError)

  """
  def get_user!(id), do: Repo.get!(User, id)

  ## User registration

  @doc """
  Registers a user.

  ## Examples

      iex> register_user(%{field: value})
      {:ok, %User{}}

      iex> register_user(%{field: bad_value})
      {:error, %Ecto.Changeset{}}

  """
  def register_user(attrs) do
    %User{}
    |> User.registration_changeset(attrs)
    |> Repo.insert()
  end

  @doc """
  Returns an `%Ecto.Changeset{}` for tracking user changes.

  ## Examples

      iex> change_user_registration(user)
      %Ecto.Changeset{data: %User{}}
```

```
"""
def change_user_registration(%User{} = user, attrs \\ %{}) do
  User.registration_changeset(user, attrs, hash_password: false, validate_email: false)
end

## Settings

@doc """
Returns an `%Ecto.Changeset{}` for changing the user email.

## Examples

    iex> change_user_email(user)
    %Ecto.Changeset{data: %User{}}

"""
def change_user_email(user, attrs \\ %{}) do
  User.email_changeset(user, attrs, validate_email: false)
end

@doc """
Emulates that the email will change without actually changing
it in the database.

## Examples

    iex> apply_user_email(user, "valid password", %{email: ...})
    {:ok, %User{}}

    iex> apply_user_email(user, "invalid password", %{email: ...})
    {:error, %Ecto.Changeset{}}

"""
def apply_user_email(user, password, attrs) do
  user
  |> User.email_changeset(attrs)
  |> User.validate_current_password(password)
  |> Ecto.Changeset.apply_action(:update)
end

@doc """
```

```
  Updates the user email using the given token.

  If the token matches, the user email is updated and the token is deleted.
  The confirmed_at date is also updated to the current time.
  """
  def update_user_email(user, token) do
    context = "change:#{user.email}"

    with {:ok, query} <- UserToken.verify_change_email_token_query(token, context),
         %UserToken{sent_to: email} <- Repo.one(query),
         {:ok, _} <- Repo.transaction(user_email_multi(user, email, context)) do
      :ok
    else
      _ -> :error
    end
  end

  defp user_email_multi(user, email, context) do
    changeset =
      user
      |> User.email_changeset(%{email: email})
      |> User.confirm_changeset()

    Ecto.Multi.new()
    |> Ecto.Multi.update(:user, changeset)
    |> Ecto.Multi.delete_all(:tokens, UserToken.by_user_and_contexts_query(user, [context]))
  end

  @doc ~S"""
  Delivers the update email instructions to the given user.

  ## Examples

      iex> deliver_user_update_email_instructions(user, current_email,
  &url(~p"/users/settings/confirm_email/#{&1}"))
      {:ok, %{to: ..., body: ...}}

  """
  def deliver_user_update_email_instructions(%User{} = user, current_email, update_email_url_fun)
      when is_function(update_email_url_fun, 1) do
    {encoded_token, user_token} = UserToken.build_email_token(user, "change:#{current_email}")
```

```
  Repo.insert!(user_token)
  UserNotifier.deliver_update_email_instructions(user, update_email_url_fun.(encoded_token))
end

@doc """
Returns an `%Ecto.Changeset{}` for changing the user password.

## Examples

    iex> change_user_password(user)
    %Ecto.Changeset{data: %User{}}

"""
def change_user_password(user, attrs \\ %{}) do
  User.password_changeset(user, attrs, hash_password: false)
end

@doc """
Updates the user password.

## Examples

    iex> update_user_password(user, "valid password", %{password: ...})
    {:ok, %User{}}

    iex> update_user_password(user, "invalid password", %{password: ...})
    {:error, %Ecto.Changeset{}}

"""
def update_user_password(user, password, attrs) do
  changeset =
    user
    |> User.password_changeset(attrs)
    |> User.validate_current_password(password)

  Ecto.Multi.new()
  |> Ecto.Multi.update(:user, changeset)
  |> Ecto.Multi.delete_all(:tokens, UserToken.by_user_and_contexts_query(user, :all))
  |> Repo.transaction()
  |> case do
    {:ok, %{user: user}} -> {:ok, user}
    {:error, :user, changeset, _} -> {:error, changeset}
```

```
  end
end

## Session

@doc """
Generates a session token.
"""
def generate_user_session_token(user) do
  {token, user_token} = UserToken.build_session_token(user)
  Repo.insert!(user_token)
  token
end

@doc """
Gets the user with the given signed token.
"""
def get_user_by_session_token(token) do
  {:ok, query} = UserToken.verify_session_token_query(token)
  Repo.one(query)
end

@doc """
Deletes the signed token with the given context.
"""
def delete_user_session_token(token) do
  Repo.delete_all(UserToken.by_token_and_context_query(token, "session"))
  :ok
end

## Confirmation

@doc ~S"""
Delivers the confirmation email instructions to the given user.

## Examples

    iex> deliver_user_confirmation_instructions(user, &url(~p"/users/confirm/#{&1}"))
    {:ok, %{to: ..., body: ...}}

    iex> deliver_user_confirmation_instructions(confirmed_user, &url(~p"/users/confirm/#{&1}"))
    {:error, :already_confirmed}
```

```
"""
def deliver_user_confirmation_instructions(%User{} = user, confirmation_url_fun)
    when is_function(confirmation_url_fun, 1) do
  if user.confirmed_at do
    {:error, :already_confirmed}
  else
    {encoded_token, user_token} = UserToken.build_email_token(user, "confirm")
    Repo.insert!(user_token)
    UserNotifier.deliver_confirmation_instructions(user, confirmation_url_fun.(encoded_token))
  end
end

@doc """
Confirms a user by the given token.

If the token matches, the user account is marked as confirmed
and the token is deleted.
"""
def confirm_user(token) do
  with {:ok, query} <- UserToken.verify_email_token_query(token, "confirm"),
       %User{} = user <- Repo.one(query),
       {:ok, %{user: user}} <- Repo.transaction(confirm_user_multi(user)) do
    {:ok, user}
  else
    _ -> :error
  end
end

defp confirm_user_multi(user) do
  Ecto.Multi.new()
  |> Ecto.Multi.update(:user, User.confirm_changeset(user))
  |> Ecto.Multi.delete_all(:tokens, UserToken.by_user_and_contexts_query(user, ["confirm"]))
end

## Reset password

@doc ~S"""
Delivers the reset password email to the given user.

## Examples
```

```
    iex> deliver_user_reset_password_instructions(user, &url(~p"/users/reset_password/#{&1}"))
    {:ok, %{to: ..., body: ...}}

  """
  def deliver_user_reset_password_instructions(%User{} = user, reset_password_url_fun)
    when is_function(reset_password_url_fun, 1) do
   {encoded_token, user_token} = UserToken.build_email_token(user, "reset_password")
   Repo.insert!(user_token)
   UserNotifier.deliver_reset_password_instructions(user, reset_password_url_fun.(encoded_token))
  end

  @doc """
  Gets the user by reset password token.

  ## Examples

    iex> get_user_by_reset_password_token("validtoken")
    %User{}

    iex> get_user_by_reset_password_token("invalidtoken")
    nil

  """
  def get_user_by_reset_password_token(token) do
   with {:ok, query} <- UserToken.verify_email_token_query(token, "reset_password"),
       %User{} = user <- Repo.one(query) do
    user
   else
    _ -> nil
   end
  end

  @doc """
  Resets the user password.

  ## Examples

    iex> reset_user_password(user, %{password: "new long password", password_confirmation: "new long password"})
    {:ok, %User{}}

    iex> reset_user_password(user, %{password: "valid", password_confirmation: "not the same"})
```

```
    {:error, %Ecto.Changeset{}}

  """
  def reset_user_password(user, attrs) do
    Ecto.Multi.new()
    |> Ecto.Multi.update(:user, User.password_changeset(user, attrs))
    |> Ecto.Multi.delete_all(:tokens, UserToken.by_user_and_contexts_query(user, :all))
    |> Repo.transaction()
    |> case do
      {:ok, %{user: user}} -> {:ok, user}
      {:error, :user, changeset, _} -> {:error, changeset}
    end
  end
end
defmodule Hub.Release do
  @moduledoc """
  Used for executing DB release tasks when run in production without Mix
  installed.
  """
  @app :hub

  def migrate do
    load_app()

    for repo <- repos() do
      {:ok, _, _} = Ecto.Migrator.with_repo(repo, &Ecto.Migrator.run(&1, :up, all: true))
    end
  end

  def rollback(repo, version) do
    load_app()
    {:ok, _, _} = Ecto.Migrator.with_repo(repo, &Ecto.Migrator.run(&1, :down, to: version))
  end

  defp repos do
    Application.fetch_env!(@app, :ecto_repos)
  end

  defp load_app do
    Application.load(@app)
  end
end
```

```
defmodule Hub.Repo do
  use Ecto.Repo,
    otp_app: :hub,
    adapter: Ecto.Adapters.Postgres
end
defmodule Hub.Sensors do
  import Ecto.Query

  alias Hub.Repo
  alias Hub.Sensors.Sensor

  def list_sensors_by_user(user_id) do
    Repo.all(from(s in Sensor, where: s.user_id == ^user_id))
  end

  def get_sensor_by_api_key(api_key) do
    Repo.get_by(Sensor, api_key: api_key)
  end



  def create_sensor(user, attrs) do
    user
    |> Ecto.build_assoc(:sensors)
    |> Sensor.changeset(Map.put(attrs, "api_key", UUID.uuid4()))
    |> Repo.insert()
  end
end
defmodule Hub.TrafficBroadcast do
  use GenServer
  alias Phoenix.PubSub

  @interval :timer.minutes(1)

  def start_link(_opts), do: GenServer.start_link(__MODULE__, %{}, name: __MODULE__)

  def init(_) do
    schedule_broadcast()
    {:ok, %{}}
  end

  def handle_info(:broadcast, state) do
    PubSub.broadcast(Hub.PubSub, "pubsub_refresh", :refresh)
```

```
    schedule_broadcast()
    {:noreply, state}
  end

  defp schedule_broadcast() do
    Process.send_after(self(), :broadcast, @interval)
  end
end
defmodule Hub.TrafficRecords do
  import Ecto.Query
  alias Hub.Repo
  alias Hub.TrafficRecords.TrafficRecord

  def get_sensor_stats(sensor_id, period) do
    interval = "minute"
    now_unix = Timex.now() |> Timex.to_unix()
    from_unix = now_unix - String.to_integer(period) * 3600

    base_query =
      from(tr in TrafficRecord,
        where: tr.sensor_id == ^sensor_id and tr.timestamp >= ^from_unix,
        select: %{
          truncated_time: fragment("date_trunc(?, to_timestamp(?))", ^interval, tr.timestamp),
          packet_count: tr.packet_count,
          unique_src_ip_count: tr.unique_src_ip_count,
          tcp_syn_count: tr.tcp_syn_count,
          anomaly: tr.anomaly,
          cusum_anomaly: tr.cusum_anomaly,
          isolation_anomaly: tr.isolation_anomaly,
          proto_counter: tr.proto_counter
        }
      )

    query =
      from(sq in subquery(base_query),
        group_by: sq.truncated_time,
        order_by: sq.truncated_time,
        select: %{
          time: sq.truncated_time,
          total_packet_count: sum(sq.packet_count),
          total_unique_src_ips: sum(sq.unique_src_ip_count),
```

```
      total_tcp_syn_count: sum(sq.tcp_syn_count),
      anomaly_count: fragment("SUM(CASE WHEN ? = TRUE THEN 1 ELSE 0 END)", sq.anomaly),
      cusum_anomaly_count:
        fragment("SUM(CASE WHEN ? = TRUE THEN 1 ELSE 0 END)", sq.cusum_anomaly),
      isolation_anomaly_count:
        fragment("SUM(CASE WHEN ? = TRUE THEN 1 ELSE 0 END)", sq.isolation_anomaly),
      proto_tcp_count: fragment("SUM(COALESCE((? ->> '6')::int, 0))", sq.proto_counter),
      proto_udp_count: fragment("SUM(COALESCE((? ->> '17')::int, 0))", sq.proto_counter),
      proto_icmp_count: fragment("SUM(COALESCE((? ->> '1')::int, 0))", sq.proto_counter)
    }
  )

  Repo.all(query)
end

def get_top_ports(sensor_id, period) do
  now_unix = Timex.now() |> Timex.to_unix()
  from_unix = now_unix - String.to_integer(period) * 3600

  from(t in TrafficRecord,
    where: t.sensor_id == ^sensor_id and t.timestamp >= ^from_unix,
    group_by: t.top_dst_ports,
    select: %{port: t.top_dst_ports, count: count(t.id)},
    order_by: [desc: count(t.id)],
    limit: 10
  )
  |> Repo.all()
  |> Enum.flat_map(& &1.port)
  |> Enum.reduce(%{}, fn [port, count], acc ->
    Map.update(acc, port, count, &(&1 + count))
  end)
  |> Enum.map(fn {port, count} -> %{port: port, count: count} end)
  |> Enum.sort_by(& &1.count, :desc)
  |> Enum.take(10)
end

def store_report(sensor, attrs) do
  sensor
  |> Ecto.build_assoc(:traffic_records)
  |> TrafficRecord.changeset(attrs)
  |> Repo.insert()
end
```

```
end
defmodule Hub.Accounts.User do
  use Ecto.Schema
  import Ecto.Changeset

  schema "users" do
    field(:email, :string)
    field(:password, :string, virtual: true, redact: true)
    field(:hashed_password, :string, redact: true)
    field(:current_password, :string, virtual: true, redact: true)
    field(:confirmed_at, :utc_datetime)
    has_many(:sensors, Hub.Sensors.Sensor)
    timestamps(type: :utc_datetime)
  end

  @doc """
  A user changeset for registration.

  It is important to validate the length of both email and password.
  Otherwise databases may truncate the email without warnings, which
  could lead to unpredictable or insecure behaviour. Long passwords may
  also be very expensive to hash for certain algorithms.

  ## Options

    * `:hash_password` - Hashes the password so it can be stored securely
      in the database and ensures the password field is cleared to prevent
      leaks in the logs. If password hashing is not needed and clearing the
      password field is not desired (like when using this changeset for
      validations on a LiveView form), this option can be set to `false`.
      Defaults to `true`.

    * `:validate_email` - Validates the uniqueness of the email, in case
      you don't want to validate the uniqueness of the email (like when
      using this changeset for validations on a LiveView form before
      submitting the form), this option can be set to `false`.
      Defaults to `true`.
  """
  def registration_changeset(user, attrs, opts \\ []) do
    user
    |> cast(attrs, [:email, :password])
    |> validate_email(opts)
```

```
  |> validate_password(opts)
end

defp validate_email(changeset, opts) do
  changeset
  |> validate_required([:email])
  |> validate_format(:email, ~r/^[^\s]+@[^\s]+$/, message: "must have the @ sign and no spaces")
  |> validate_length(:email, max: 160)
  |> maybe_validate_unique_email(opts)
end

defp validate_password(changeset, opts) do
  changeset
  |> validate_required([:password])
  |> validate_length(:password, min: 12, max: 72)
  # Examples of additional password validation:
  # |> validate_format(:password, ~r/[a-z]/, message: "at least one lower case character")
  # |> validate_format(:password, ~r/[A-Z]/, message: "at least one upper case character")
  # |> validate_format(:password, ~r/[!?@#$%^&*_0-9]/, message: "at least one digit or punctuation character")
  |> maybe_hash_password(opts)
end

defp maybe_hash_password(changeset, opts) do
  hash_password? = Keyword.get(opts, :hash_password, true)
  password = get_change(changeset, :password)

  if hash_password? && password && changeset.valid? do
    changeset
    # If using Bcrypt, then further validate it is at most 72 bytes long
    |> validate_length(:password, max: 72, count: :bytes)
    # Hashing could be done with `Ecto.Changeset.prepare_changes/2`, but that
    # would keep the database transaction open longer and hurt performance.
    |> put_change(:hashed_password, Bcrypt.hash_pwd_salt(password))
    |> delete_change(:password)
  else
    changeset
  end
end

defp maybe_validate_unique_email(changeset, opts) do
  if Keyword.get(opts, :validate_email, true) do
    changeset
```

```
    |> unsafe_validate_unique(:email, Hub.Repo)
    |> unique_constraint(:email)
  else
    changeset
  end
end

@doc """
A user changeset for changing the email.

It requires the email to change otherwise an error is added.
"""
def email_changeset(user, attrs, opts \\ []) do
  user
  |> cast(attrs, [:email])
  |> validate_email(opts)
  |> case do
    %{changes: %{email: _}} = changeset -> changeset
    %{} = changeset -> add_error(changeset, :email, "did not change")
  end
end

@doc """
A user changeset for changing the password.

## Options

  * `:hash_password` - Hashes the password so it can be stored securely
    in the database and ensures the password field is cleared to prevent
    leaks in the logs. If password hashing is not needed and clearing the
    password field is not desired (like when using this changeset for
    validations on a LiveView form), this option can be set to `false`.
    Defaults to `true`.
"""
def password_changeset(user, attrs, opts \\ []) do
  user
  |> cast(attrs, [:password])
  |> validate_confirmation(:password, message: "does not match password")
  |> validate_password(opts)
end

@doc """
```

```
  Confirms the account by setting `confirmed_at`.
  """
  def confirm_changeset(user) do
    now = DateTime.utc_now() |> DateTime.truncate(:second)
    change(user, confirmed_at: now)
  end

  @doc """
  Verifies the password.

  If there is no user or the user doesn't have a password, we call
  `Bcrypt.no_user_verify/0` to avoid timing attacks.
  """
  def valid_password?(%Hub.Accounts.User{hashed_password: hashed_password}, password)
      when is_binary(hashed_password) and byte_size(password) > 0 do
    Bcrypt.verify_pass(password, hashed_password)
  end

  def valid_password?(_, _) do
    Bcrypt.no_user_verify()
    false
  end

  @doc """
  Validates the current password otherwise adds an error to the changeset.
  """
  def validate_current_password(changeset, password) do
    changeset = cast(changeset, %{current_password: password}, [:current_password])

    if valid_password?(changeset.data, password) do
      changeset
    else
      add_error(changeset, :current_password, "is not valid")
    end
  end
end
defmodule Hub.Accounts.UserNotifier do
  import Swoosh.Email

  alias Hub.Mailer

  # Delivers the email using the application mailer.
```

```
defp deliver(recipient, subject, body) do
  email =
    new()
    |> to(recipient)
    |> from({"Hub", "contact@example.com"})
    |> subject(subject)
    |> text_body(body)

  with {:ok, _metadata} <- Mailer.deliver(email) do
    {:ok, email}
  end
end

@doc """
Deliver instructions to confirm account.
"""
def deliver_confirmation_instructions(user, url) do
  deliver(user.email, "Confirmation instructions", """

  ==============================

  Hi #{user.email},

  You can confirm your account by visiting the URL below:

  #{url}

  If you didn't create an account with us, please ignore this.

  ==============================
  """)
end

@doc """
Deliver instructions to reset a user password.
"""
def deliver_reset_password_instructions(user, url) do
  deliver(user.email, "Reset password instructions", """

  ==============================

  Hi #{user.email},
```

```
    You can reset your password by visiting the URL below:

    #{url}

    If you didn't request this change, please ignore this.

    ==============================
    """)
  end

  @doc """
  Deliver instructions to update a user email.
  """
  def deliver_update_email_instructions(user, url) do
    deliver(user.email, "Update email instructions", """

    ==============================

    Hi #{user.email},

    You can change your email by visiting the URL below:

    #{url}

    If you didn't request this change, please ignore this.

    ==============================
    """)
  end
end
defmodule Hub.Accounts.UserToken do
  use Ecto.Schema
  import Ecto.Query
  alias Hub.Accounts.UserToken

  @hash_algorithm :sha256
  @rand_size 32

  # It is very important to keep the reset password token expiry short,
  # since someone with access to the email may take over the account.
  @reset_password_validity_in_days 1
```

```
@confirm_validity_in_days 7
@change_email_validity_in_days 7
@session_validity_in_days 60

schema "users_tokens" do
  field :token, :binary
  field :context, :string
  field :sent_to, :string
  belongs_to :user, Hub.Accounts.User

  timestamps(type: :utc_datetime, updated_at: false)
end

@doc """
Generates a token that will be stored in a signed place,
such as session or cookie. As they are signed, those
tokens do not need to be hashed.

The reason why we store session tokens in the database, even
though Phoenix already provides a session cookie, is because
Phoenix' default session cookies are not persisted, they are
simply signed and potentially encrypted. This means they are
valid indefinitely, unless you change the signing/encryption
salt.

Therefore, storing them allows individual user
sessions to be expired. The token system can also be extended
to store additional data, such as the device used for logging in.
You could then use this information to display all valid sessions
and devices in the UI and allow users to explicitly expire any
session they deem invalid.
"""
def build_session_token(user) do
  token = :crypto.strong_rand_bytes(@rand_size)
  {token, %UserToken{token: token, context: "session", user_id: user.id}}
end

@doc """
Checks if the token is valid and returns its underlying lookup query.

The query returns the user found by the token, if any.
```

```
The token is valid if it matches the value in the database and it has
not expired (after @session_validity_in_days).
"""
def verify_session_token_query(token) do
  query =
    from token in by_token_and_context_query(token, "session"),
      join: user in assoc(token, :user),
      where: token.inserted_at > ago(@session_validity_in_days, "day"),
      select: user

  {:ok, query}
end

@doc """
Builds a token and its hash to be delivered to the user's email.

The non-hashed token is sent to the user email while the
hashed part is stored in the database. The original token cannot be reconstructed,
which means anyone with read-only access to the database cannot directly use
the token in the application to gain access. Furthermore, if the user changes
their email in the system, the tokens sent to the previous email are no longer
valid.

Users can easily adapt the existing code to provide other types of delivery methods,
for example, by phone numbers.
"""
def build_email_token(user, context) do
  build_hashed_token(user, context, user.email)
end

defp build_hashed_token(user, context, sent_to) do
  token = :crypto.strong_rand_bytes(@rand_size)
  hashed_token = :crypto.hash(@hash_algorithm, token)

  {Base.url_encode64(token, padding: false),
   %UserToken{
     token: hashed_token,
     context: context,
     sent_to: sent_to,
     user_id: user.id
   }}
end
```

```
@doc """
Checks if the token is valid and returns its underlying lookup query.

The query returns the user found by the token, if any.

The given token is valid if it matches its hashed counterpart in the
database and the user email has not changed. This function also checks
if the token is being used within a certain period, depending on the
context. The default contexts supported by this function are either
"confirm", for account confirmation emails, and "reset_password",
for resetting the password. For verifying requests to change the email,
see `verify_change_email_token_query/2`.
"""
def verify_email_token_query(token, context) do
  case Base.url_decode64(token, padding: false) do
    {:ok, decoded_token} ->
      hashed_token = :crypto.hash(@hash_algorithm, decoded_token)
      days = days_for_context(context)

      query =
        from token in by_token_and_context_query(hashed_token, context),
          join: user in assoc(token, :user),
          where: token.inserted_at > ago(^days, "day") and token.sent_to == user.email,
          select: user

      {:ok, query}

    :error ->
      :error
  end
end

defp days_for_context("confirm"), do: @confirm_validity_in_days
defp days_for_context("reset_password"), do: @reset_password_validity_in_days

@doc """
Checks if the token is valid and returns its underlying lookup query.

The query returns the user found by the token, if any.

This is used to validate requests to change the user
```

```
email. It is different from `verify_email_token_query/2` precisely because
`verify_email_token_query/2` validates the email has not changed, which is
the starting point by this function.

The given token is valid if it matches its hashed counterpart in the
database and if it has not expired (after @change_email_validity_in_days).
The context must always start with "change:".
"""
def verify_change_email_token_query(token, "change:" <> _ = context) do
  case Base.url_decode64(token, padding: false) do
    {:ok, decoded_token} ->
      hashed_token = :crypto.hash(@hash_algorithm, decoded_token)

      query =
        from token in by_token_and_context_query(hashed_token, context),
          where: token.inserted_at > ago(@change_email_validity_in_days, "day")

      {:ok, query}

    :error ->
      :error
  end
end

@doc """
Returns the token struct for the given token value and context.
"""
def by_token_and_context_query(token, context) do
  from UserToken, where: [token: ^token, context: ^context]
end

@doc """
Gets all tokens for the given user for the given contexts.
"""
def by_user_and_contexts_query(user, :all) do
  from t in UserToken, where: t.user_id == ^user.id
end

def by_user_and_contexts_query(user, [_ | _] = contexts) do
  from t in UserToken, where: t.user_id == ^user.id and t.context in ^contexts
end
end
```

```
defmodule Hub.Sensors.Sensor do
  use Ecto.Schema
  import Ecto.Changeset

  schema "sensors" do
    field(:name, :string)
    field(:api_key, :string)
    belongs_to(:user, Hub.Accounts.User)

    has_many(:traffic_records, Hub.TrafficRecords.TrafficRecord)

    timestamps(type: :utc_datetime)
  end

  @doc false
  def changeset(sensor, attrs) do
    sensor
    |> cast(attrs, [:name, :api_key])
    |> validate_required([:name, :api_key])
    |> unique_constraint(:api_key)
  end
end
defmodule Hub.TrafficRecords.TrafficRecord do
  use Ecto.Schema
  import Ecto.Changeset

  schema "traffic_records" do
    field(:timestamp, :integer)
    field(:packet_count, :integer)
    field(:unique_src_ip_count, :integer)
    field(:proto_counter, :map)
    field(:tcp_syn_count, :integer)
    field(:top_dst_ports, {:array, {:array, :integer}}) # масив масивів [port, count]
    field(:anomaly, :boolean, default: false)
    field(:cusum_anomaly, :boolean, default: false)
    field(:isolation_anomaly, :boolean, default: false)

    belongs_to(:sensor, Hub.Sensors.Sensor)

    timestamps()
  end
```

```
  def changeset(record, attrs) do
    record
    |> cast(attrs, [
      :timestamp,
      :packet_count,
      :unique_src_ip_count,
      :proto_counter,
      :tcp_syn_count,
      :top_dst_ports,
      :anomaly,
      :cusum_anomaly,
      :isolation_anomaly,
      :sensor_id
    ])
    |> validate_required([:timestamp, :sensor_id])
    |> foreign_key_constraint(:sensor_id)
  end
end
defmodule HubPrivateAPI.Controllers.SensorController do
  use HubWeb, :controller

  alias Hub.Sensors
  alias Hub.Sensors.Sensor
  alias Hub.TrafficRecords

  @detector_url "http://0.0.0.0:10000/api/detect"

  def receive_data(conn, params) do
    case get_req_header(conn, "authorization") do
      ["Bearer " <> api_key] ->
        case Sensors.get_sensor_by_api_key(api_key) do
          nil ->
            send_resp(conn, 401, "Invalid API Key")

          %Sensor{} = sensor ->
            Task.start(fn ->
              forward_and_handle_anomalies(sensor, params)
            end)

            send_resp(conn, 202, "Accepted")
        end
```

```
      _ ->
        send_resp(conn, 401, "Missing API Key")
    end
  end

  defp forward_and_handle_anomalies(sensor, params) do
    headers = [
      {"Content-Type", "application/json"},
      {"Authorization", "Bearer #{sensor.api_key}"}
    ]

    body = Jason.encode!(params)

    case HTTPoison.post(@detector_url, body, headers) do
      {:ok, %HTTPoison.Response{status_code: 200, body: body}} ->
        case Jason.decode(body) do
          {:ok, %{} = data} ->
            TrafficRecords.store_report(sensor, data)

          _ ->
            IO.warn("[!] Could not parse AI module response")
        end

      {:error, err} ->
        IO.inspect(err, label: "Error contacting AI module")

      err ->
        IO.inspect(err)
        IO.warn("[!] Unknown error from AI module")
    end
  end
end
defmodule HubWeb.UserAuth do
  use HubWeb, :verified_routes

  import Plug.Conn
  import Phoenix.Controller

  alias Hub.Accounts

  # Make the remember me cookie valid for 60 days.
  # If you want bump or reduce this value, also change
```

```
# the token expiry itself in UserToken.
@max_age 60 * 60 * 24 * 60
@remember_me_cookie "_hub_web_user_remember_me"
@remember_me_options [sign: true, max_age: @max_age, same_site: "Lax"]

@doc """
Logs the user in.

It renews the session ID and clears the whole session
to avoid fixation attacks. See the renew_session
function to customize this behaviour.

It also sets a `:live_socket_id` key in the session,
so LiveView sessions are identified and automatically
disconnected on log out. The line can be safely removed
if you are not using LiveView.
"""
def log_in_user(conn, user, params \\ %{}) do
  token = Accounts.generate_user_session_token(user)
  user_return_to = get_session(conn, :user_return_to)

  conn
  |> renew_session()
  |> put_token_in_session(token)
  |> maybe_write_remember_me_cookie(token, params)
  |> redirect(to: user_return_to || signed_in_path(conn))
end

defp maybe_write_remember_me_cookie(conn, token, %{"remember_me" => "true"}) do
  put_resp_cookie(conn, @remember_me_cookie, token, @remember_me_options)
end

defp maybe_write_remember_me_cookie(conn, _token, _params) do
  conn
end

# This function renews the session ID and erases the whole
# session to avoid fixation attacks. If there is any data
# in the session you may want to preserve after log in/log out,
# you must explicitly fetch the session data before clearing
# and then immediately set it after clearing, for example:
#
```

```
#    defp renew_session(conn) do
#      preferred_locale = get_session(conn, :preferred_locale)
#
#      conn
#      |> configure_session(renew: true)
#      |> clear_session()
#      |> put_session(:preferred_locale, preferred_locale)
#    end
#
defp renew_session(conn) do
  delete_csrf_token()

  conn
  |> configure_session(renew: true)
  |> clear_session()
end

@doc """
Logs the user out.

It clears all session data for safety. See renew_session.
"""
def log_out_user(conn) do
  user_token = get_session(conn, :user_token)
  user_token && Accounts.delete_user_session_token(user_token)

  if live_socket_id = get_session(conn, :live_socket_id) do
    HubWeb.Endpoint.broadcast(live_socket_id, "disconnect", %{})
  end

  conn
  |> renew_session()
  |> delete_resp_cookie(@remember_me_cookie)
  |> redirect(to: ~p"/")
end

@doc """
Authenticates the user by looking into the session
and remember me token.
"""
def fetch_current_user(conn, _opts) do
  {user_token, conn} = ensure_user_token(conn)
```

```
  user = user_token && Accounts.get_user_by_session_token(user_token)
  assign(conn, :current_user, user)
end

defp ensure_user_token(conn) do
  if token = get_session(conn, :user_token) do
    {token, conn}
  else
    conn = fetch_cookies(conn, signed: [@remember_me_cookie])

    if token = conn.cookies[@remember_me_cookie] do
      {token, put_token_in_session(conn, token)}
    else
      {nil, conn}
    end
  end
end

@doc """
Handles mounting and authenticating the current_user in LiveViews.

## `on_mount` arguments

  * `:mount_current_user` - Assigns current_user
    to socket assigns based on user_token, or nil if
    there's no user_token or no matching user.

  * `:ensure_authenticated` - Authenticates the user from the session,
    and assigns the current_user to socket assigns based
    on user_token.
    Redirects to login page if there's no logged user.

  * `:redirect_if_user_is_authenticated` - Authenticates the user from the session.
    Redirects to signed_in_path if there's a logged user.

## Examples

Use the `on_mount` lifecycle macro in LiveViews to mount or authenticate
the current_user:

    defmodule HubWeb.PageLive do
      use HubWeb, :live_view
```

```
      on_mount {HubWeb.UserAuth, :mount_current_user}
      ...
    end

Or use the `live_session` of your router to invoke the on_mount callback:

    live_session :authenticated, on_mount: [{HubWeb.UserAuth, :ensure_authenticated}] do
      live "/profile", ProfileLive, :index
    end
"""
def on_mount(:mount_current_user, _params, session, socket) do
  {:cont, mount_current_user(socket, session)}
end

def on_mount(:ensure_authenticated, _params, session, socket) do
  socket = mount_current_user(socket, session)

  if socket.assigns.current_user do
    {:cont, socket}
  else
    socket =
      socket
      |> Phoenix.LiveView.put_flash(:error, "You must log in to access this page.")
      |> Phoenix.LiveView.redirect(to: ~p"/users/log_in")

    {:halt, socket}
  end
end

def on_mount(:redirect_if_user_is_authenticated, _params, session, socket) do
  socket = mount_current_user(socket, session)

  if socket.assigns.current_user do
    {:halt, Phoenix.LiveView.redirect(socket, to: signed_in_path(socket))}
  else
    {:cont, socket}
  end
end

defp mount_current_user(socket, session) do
  Phoenix.Component.assign_new(socket, :current_user, fn ->
```

```
    if user_token = session["user_token"] do
      Accounts.get_user_by_session_token(user_token)
    end
  end)
end

@doc """
Used for routes that require the user to not be authenticated.
"""
def redirect_if_user_is_authenticated(conn, _opts) do
  if conn.assigns[:current_user] do
    conn
    |> redirect(to: signed_in_path(conn))
    |> halt()
  else
    conn
  end
end

@doc """
Used for routes that require the user to be authenticated.

If you want to enforce the user email is confirmed before
they use the application at all, here would be a good place.
"""
def require_authenticated_user(conn, _opts) do
  if conn.assigns[:current_user] do
    conn
  else
    conn
    |> put_flash(:error, "You must log in to access this page.")
    |> maybe_store_return_to()
    |> redirect(to: ~p"/users/log_in")
    |> halt()
  end
end

defp put_token_in_session(conn, token) do
  conn
  |> put_session(:user_token, token)
  |> put_session(:live_socket_id, "users_sessions:#{Base.url_encode64(token)}")
end
```

```
  defp maybe_store_return_to(%{method: "GET"} = conn) do
    put_session(conn, :user_return_to, current_path(conn))
  end

  defp maybe_store_return_to(conn), do: conn

  defp signed_in_path(_conn), do: ~p"/"
end
defmodule HubWeb.Telemetry do
  use Supervisor
  import Telemetry.Metrics

  def start_link(arg) do
    Supervisor.start_link(__MODULE__, arg, name: __MODULE__)
  end

  @impl true
  def init(_arg) do
    children = [
      # Telemetry poller will execute the given period measurements
      # every 10_000ms. Learn more here: https://hexdocs.pm/telemetry_metrics
      {:telemetry_poller, measurements: periodic_measurements(), period: 10_000}
      # Add reporters as children of your supervision tree.
      # {Telemetry.Metrics.ConsoleReporter, metrics: metrics()}
    ]

    Supervisor.init(children, strategy: :one_for_one)
  end

  def metrics do
    [
      # Phoenix Metrics
      summary("phoenix.endpoint.start.system_time",
        unit: {:native, :millisecond}
      ),
      summary("phoenix.endpoint.stop.duration",
        unit: {:native, :millisecond}
      ),
      summary("phoenix.router_dispatch.start.system_time",
        tags: [:route],
        unit: {:native, :millisecond}
```

```
),
summary("phoenix.router_dispatch.exception.duration",
  tags: [:route],
  unit: {:native, :millisecond}
),
summary("phoenix.router_dispatch.stop.duration",
  tags: [:route],
  unit: {:native, :millisecond}
),
summary("phoenix.socket_connected.duration",
  unit: {:native, :millisecond}
),
sum("phoenix.socket_drain.count"),
summary("phoenix.channel_joined.duration",
  unit: {:native, :millisecond}
),
summary("phoenix.channel_handled_in.duration",
  tags: [:event],
  unit: {:native, :millisecond}
),

# Database Metrics
summary("hub.repo.query.total_time",
  unit: {:native, :millisecond},
  description: "The sum of the other measurements"
),
summary("hub.repo.query.decode_time",
  unit: {:native, :millisecond},
  description: "The time spent decoding the data received from the database"
),
summary("hub.repo.query.query_time",
  unit: {:native, :millisecond},
  description: "The time spent executing the query"
),
summary("hub.repo.query.queue_time",
  unit: {:native, :millisecond},
  description: "The time spent waiting for a database connection"
),
summary("hub.repo.query.idle_time",
  unit: {:native, :millisecond},
  description:
    "The time the connection spent waiting before being checked out for the query"
```

```
      ),

      # VM Metrics
      summary("vm.memory.total", unit: {:byte, :kilobyte}),
      summary("vm.total_run_queue_lengths.total"),
      summary("vm.total_run_queue_lengths.cpu"),
      summary("vm.total_run_queue_lengths.io")
    ]
  end

  defp periodic_measurements do
    [
      # A module, function and arguments to be invoked periodically.
      # This function must call :telemetry.execute/3 and a metric must be added above.
      # {HubWeb, :count_users, []}
    ]
  end
end
defmodule HubWeb.Router do
  use HubWeb, :router

  import HubWeb.UserAuth

  pipeline :browser do
    plug(:accepts, ["html"])
    plug(:fetch_session)
    plug(:fetch_live_flash)
    plug(:put_root_layout, html: {HubWeb.Layouts, :root})
    plug(:protect_from_forgery)
    plug(:put_secure_browser_headers)
    plug(:fetch_current_user)
  end

  pipeline :api do
    plug(:accepts, ["json"])
  end

  scope "/api/", HubPrivateAPI.Controllers do
    pipe_through([:api])

    post("/sensor", SensorController, :receive_data)
  end
```

```
# Enable LiveDashboard and Swoosh mailbox preview in development
if Application.compile_env(:hub, :dev_routes) do
  import Phoenix.LiveDashboard.Router

  scope "/dev" do
    pipe_through(:browser)

    live_dashboard("/dashboard", metrics: HubWeb.Telemetry)
    forward("/mailbox", Plug.Swoosh.MailboxPreview)
  end
end

## Authentication routes

scope "/", HubWeb do
  pipe_through([:browser, :redirect_if_user_is_authenticated])

  live_session :redirect_if_user_is_authenticated,
    on_mount: [{HubWeb.UserAuth, :redirect_if_user_is_authenticated}] do
    live("/users/register", UserRegistrationLive, :new)
    live("/users/log_in", UserLoginLive, :new)
    live("/users/reset_password", UserForgotPasswordLive, :new)
    live("/users/reset_password/:token", UserResetPasswordLive, :edit)
  end

  post("/users/log_in", UserSessionController, :create)
end

scope "/", HubWeb do
  pipe_through([:browser, :require_authenticated_user])

  live_session :require_authenticated_user,
    on_mount: [{HubWeb.UserAuth, :ensure_authenticated}] do
    live("/users/settings", UserSettingsLive, :edit)
    live("/users/settings/confirm_email/:token", UserSettingsLive, :confirm_email)
    live("/dashboard", UserDashboardLive, :show)
    live("/", UserDashboardLive, :show)
  end
end

scope "/", HubWeb do
```

```
    pipe_through([:browser])

    delete("/users/log_out", UserSessionController, :delete)

    live_session :current_user,
      on_mount: [{HubWeb.UserAuth, :mount_current_user}] do
      live("/users/confirm/:token", UserConfirmationLive, :edit)
      live("/users/confirm", UserConfirmationInstructionsLive, :new)
    end
  end
end
defmodule HubWeb.Endpoint do
  use Phoenix.Endpoint, otp_app: :hub

  # The session will be stored in the cookie and signed,
  # this means its contents can be read but not tampered with.
  # Set :encryption_salt if you would also like to encrypt it.
  @session_options [
    store: :cookie,
    key: "_hub_key",
    signing_salt: "Dn6jQFDy",
    same_site: "Lax"
  ]

  socket "/live", Phoenix.LiveView.Socket,
    websocket: [connect_info: [session: @session_options]],
    longpoll: [connect_info: [session: @session_options]]

  # Serve at "/" the static files from "priv/static" directory.
  #
  # You should set gzip to true if you are running phx.digest
  # when deploying your static files in production.
  plug Plug.Static,
    at: "/",
    from: :hub,
    gzip: false,
    only: HubWeb.static_paths()

  # Code reloading can be explicitly enabled under the
  # :code_reloader configuration of your endpoint.
  if code_reloading? do
    socket "/phoenix/live_reload/socket", Phoenix.LiveReloader.Socket
```

```
    plug Phoenix.LiveReloader
    plug Phoenix.CodeReloader
    plug Phoenix.Ecto.CheckRepoStatus, otp_app: :hub
  end

  plug Phoenix.LiveDashboard.RequestLogger,
    param_key: "request_logger",
    cookie_key: "request_logger"

  plug Plug.RequestId
  plug Plug.Telemetry, event_prefix: [:phoenix, :endpoint]

  plug Plug.Parsers,
    parsers: [:urlencoded, :multipart, :json],
    pass: ["*/*"],
    json_decoder: Phoenix.json_library()

  plug Plug.MethodOverride
  plug Plug.Head
  plug Plug.Session, @session_options
  plug HubWeb.Router
end
defmodule HubWeb.UserDashboardLive do
  use HubWeb, :live_view
  alias Hub.Sensors

  alias Hub.Accounts
  alias Hub.TrafficRecords
  alias Phoenix.PubSub
  @impl true
  def mount(_params, %{"user_token" => user_token}, socket) do
    user = Accounts.get_user_by_session_token(user_token)
    sensors = Sensors.list_sensors_by_user(user.id)
    selected_sensor = List.first(sensors)

    # Завантажуємо початкову статистику для обраного сенсора
    stats = load_stats(selected_sensor, "3")
    top_ports = load_top_ports(selected_sensor, "3")

    if connected?(socket) && selected_sensor,
      do: PubSub.subscribe(Hub.PubSub, "pubsub_refresh")
```

```
  {:ok,
   socket
   |> assign(sensors: sensors)
   |> assign(selected_sensor: selected_sensor)
   |> assign(show_add_form: false)
   |> assign(stats: stats)
   |> assign(top_ports: top_ports)
   |> assign(search_period: "3")}
end

defp load_stats(nil, _), do: %{}

defp load_stats(sensor, search_period),
  do: TrafficRecords.get_sensor_stats(sensor.id, search_period)

defp load_top_ports(nil, _), do: []

defp load_top_ports(sensor, search_period) do
  sensor.id
  |> TrafficRecords.get_top_ports(search_period)
end

def handle_event("select_sensor", %{"sensor_id" => sensor_id}, socket) do
  sensor = Enum.find(socket.assigns.sensors, &(&1.id == String.to_integer(sensor_id)))

  stats = load_stats(sensor, socket.assigns.search_period)
  top_ports = load_top_ports(sensor, socket.assigns.search_period)

  {:noreply,
   socket
   |> assign(selected_sensor: sensor)
   |> assign(stats: stats)
   |> assign(top_ports: top_ports)}
end

def handle_event("add_sensor", _params, socket) do
  {:noreply, assign(socket, :show_add_form, true)}
end

def handle_event(
      "save_sensor",
      %{"sensor" => %{"name" => name}},
```

```
      %{assigns: %{current_user: user}} = socket
    ) do
  case Hub.Sensors.create_sensor(user, %{"name" => name}) do
    {:ok, sensor} ->
      sensors = Hub.Sensors.list_sensors_by_user(user.id)
      stats = load_stats(sensor, socket.assigns.search_period)
      top_ports = load_top_ports(sensor, socket.assigns.search_period)

      {:noreply,
       socket
       |> assign(sensors: sensors)
       |> assign(selected_sensor: sensor)
       |> assign(stats: stats)
       |> assign(:top_ports, top_ports)
       |> assign(show_add_form: false)}

    {:error, changeset} ->
      {:noreply, assign(socket, changeset: changeset)}
  end
end

def handle_event("change_period", %{"period" => search_period}, socket) do
  stats = load_stats(socket.assigns.selected_sensor, search_period)
  top_ports = load_top_ports(socket.assigns.selected_sensor, search_period)

  {:noreply,
   socket
   |> assign(:search_period, search_period)
   |> assign(:stats, stats)
   |> assign(:top_ports, top_ports)}
end

def handle_info(:refresh, socket) do
  stats = load_stats(socket.assigns.selected_sensor, socket.assigns.search_period)
  top_ports = load_top_ports(socket.assigns.selected_sensor, socket.assigns.search_period)

  {:noreply,
   socket
   |> assign(stats: stats)
   |> assign(top_ports: top_ports)}
 end
end
```

```
<div class="space-y-6 bg-gray-100 min-h-screen p-6 text-gray-900">

  <!-- Кнопка додавання сенсора -->
  <div>
    <button phx-click="add_sensor" class="bg-black text-white px-4 py-2 rounded-lg hover:bg-gray-800 transition">
      ➕ Додати сенсор
    </button>
  </div>

  <!-- Форма додавання -->
  <%= if @show_add_form do %>
    <div class="p-6 border border-gray-300 rounded-lg shadow-md bg-white">
      <h3 class="text-lg font-semibold mb-4">Новий сенсор</h3>
      <form phx-submit="save_sensor">
        <input type="text" name="sensor[name]" placeholder="Назва сенсора"
          class="border border-gray-300 px-3 py-2 rounded w-full mb-4 focus:outline-none focus:ring-2 focus:ring-black" />
        <button type="submit" class="bg-gray-900 text-white px-4 py-2 rounded hover:bg-gray-700 transition">💾
          Зберегти</button>
      </form>
    </div>
    <% end %>

      <!-- Таблиця сенсорів -->
      <div class="overflow-x-auto">
        <table class="table-auto w-full border-collapse border border-gray-300 shadow-sm bg-white rounded-lg">
          <thead>
            <tr class="bg-gray-200 text-gray-800">
              <th class="border border-gray-300 px-4 py-2 text-left">Назва</th>
              <th class="border border-gray-300 px-4 py-2 text-left">API Ключ</th>
            </tr>
          </thead>
          <tbody>
            <%= for sensor <- @sensors do %>
              <tr phx-click="select_sensor" phx-value-sensor_id={sensor.id} class={"cursor-pointer transition
                hover:bg-gray-100 " <> if @selected_sensor && sensor.id == @selected_sensor.id, do: " bg-gray-300",
                else: "" }>
                <td class="border border-gray-300 px-4 py-2">
                  <%= sensor.name %>
```

```
            </td>
            <td class="border border-gray-300 px-4 py-2">
              <%= sensor.api_key || "–" %>
            </td>
          </tr>
          <% end %>
        </tbody>
      </table>
    </div>

    <!-- Деталі сенсора -->
    <%= if @selected_sensor do %>
      <div class="p-6 border border-gray-300 rounded-lg shadow bg-white">
        <h2 class="text-xl font-semibold mb-4">Інформація про сенсор: <%= @selected_sensor.name %>
        </h2>
        <p class="mb-4"><strong>API ключ:</strong>
          <%= @selected_sensor.api_key || "не встановлено" %>
        </p>

        <form phx-change="change_period" class="mb-4">
          <div class="flex  gap-4">
            <label class="block text-l font-medium text-black mb-4" for="period">
              Період перегляду:
            </label>
            <select name="period" id="period"
              class="bg-gray-900 text-white border border-gray-600 px-3 py-2 rounded-md focus:outline-none
focus:ring-2 focus:ring-white transition">
              <option value="1" selected={@search_period=="1" }>Остання година</option>
              <option value="3" selected={@search_period=="3" }>Останні 3 години</option>
              <option value="6" selected={@search_period=="6" }>Останні 6 годин</option>
              <option value="12" selected={@search_period=="12" }>Останні 12 години</option>
              <option value="24" selected={@search_period=="24" }>Останні 24 години</option>
            </select>
          </div>
        </form>

        <%= if @stats !=[] do %>

          <section class="mt-6">
            <h3 class="text-lg font-medium mb-2">📊 Трафік і аномалії у часі</h3>
            <div  id="trafficAnomalyChart"  phx-hook="TrafficAnomalyChart"  data-chart={Jason.encode!(%{
categories:
```

```
            Enum.map(@stats, &Timex.format!(&1.time, "{ISO:Extended}" )), total_packets:
Enum.map(@stats, &
            &1.total_packet_count), anomaly_flags: Enum.map(@stats, &(&1.anomaly_count || 0)) })}
            class="w-full h-64 bg-gray-50 border border-gray-200 rounded-lg shadow-inner">
          </div>
        </section>

        <section class="mt-6">
          <h3 class="text-lg font-medium mb-2">🔍 Розподіл протоколів у часі</h3>
          <div id="protocolDistributionChart" phx-hook="ProtocolDistributionChart" data-
chart={Jason.encode!(%{
            categories: Enum.map(@stats, &Timex.format!(&1.time, "{ISO:Extended}" )), tcp:
Enum.map(@stats,
            &(&1.proto_tcp_count || 0)), udp: Enum.map(@stats, &(&1.proto_udp_count || 0)), icmp:
Enum.map(@stats,
            &(&1.proto_icmp_count || 0)) })}
            class="w-full h-64 bg-gray-50 border border-gray-200 rounded-lg shadow-inner">
          </div>
        </section>

        <section class="mt-6">
          <h3 class="text-lg font-medium mb-2">⚠ Метод виявлення аномалії: CUSUM, Isolation
Forest</h3>
          <div id="anomalyComparisonChart" phx-hook="AnomalyComparisonChart" data-
chart={Jason.encode!(%{
            categories: Enum.map(@stats, &Timex.format!(&1.time, "{ISO:Extended}" )), cusum:
Enum.map(@stats, &
            &1.cusum_anomaly_count), isolation: Enum.map(@stats, & &1.isolation_anomaly_count) })}
            class="w-full h-64 bg-gray-50 border border-gray-200 rounded-lg shadow-inner">
          </div>
        </section>

        <section class="mt-6">
          <h3 class="text-lg font-medium mb-2">🎯 Топ цільових портів</h3>
          <div id="topPortsChart" phx-hook="TopPortsChart" data-chart={Jason.encode!(%{ ports:
Enum.map(@top_ports,
            & &1.port), counts: Enum.map(@top_ports, & &1.count) })}
            class="w-full h-64 bg-gray-50 border border-gray-200 rounded-lg shadow-inner">
          </div>
        </section>

        <section class="mt-6">
```

```
            <h3 class="text-lg font-medium mb-2">🌍 Унікальні IP-адреси у часі</h3>
            <div id="uniqueIpsChart" phx-hook="UniqueIpsChart" data-chart={Jason.encode!(%{ categories:
              Enum.map(@stats, &Timex.format!(&1.time, "{ISO:Extended}" )), unique_src_ips:
Enum.map(@stats, &
              &1.total_unique_src_ips) })}
              class="w-full h-64 bg-gray-50 border border-gray-200 rounded-lg shadow-inner">
            </div>
          </section>

          <% else %>
            <p class="text-gray-600">Немає інформації за даним сенсором.</p>
            <% end %>
        </div>
        <% else %>
          <%= if @sensors !=[] do %>

          <p class="text-gray-700">Оберіть сенсор зі списку вище, щоб переглянути деталі.</p>
          <% else %>
            <p class="text-gray-700">Додайте сенсор, щоб отримати унікальний коюч для свого
пристрою.</p>
            <% end %>
              <% end %>
</div>
defmodule HubWeb.UserConfirmationInstructionsLive do
  use HubWeb, :live_view

  alias Hub.Accounts

  def render(assigns) do
    ~H"""
    <div class="mx-auto max-w-sm">
      <.header class="text-center">
        No confirmation instructions received?
        <:subtitle>We'll send a new confirmation link to your inbox</:subtitle>
      </.header>

      <.simple_form for={@form} id="resend_confirmation_form" phx-submit="send_instructions">
        <.input field={@form[:email]} type="email" placeholder="Email" required />
        <:actions>
          <.button phx-disable-with="Sending..." class="w-full">
            Resend confirmation instructions
          </.button>
```

```
      </:actions>
    </.simple_form>

    <p class="text-center mt-4">
      <.link href={~p"/users/register"}>Register</.link>
      | <.link href={~p"/users/log_in"}>Log in</.link>
    </p>
  </div>
  """
 end

 def mount(_params, _session, socket) do
  {:ok, assign(socket, form: to_form(%{}, as: "user"))}
 end

 def handle_event("send_instructions", %{"user" => %{"email" => email}}, socket) do
  if user = Accounts.get_user_by_email(email) do
   Accounts.deliver_user_confirmation_instructions(
    user,
    &url(~p"/users/confirm/#{&1}")
   )
  end

  info =
   "If your email is in our system and it has not been confirmed yet, you will receive an email with instructions
shortly."

  {:noreply,
   socket
   |> put_flash(:info, info)
   |> redirect(to: ~p"/")}
 end
end
defmodule HubWeb.UserConfirmationLive do
 use HubWeb, :live_view

 alias Hub.Accounts

 def render(%{live_action: :edit} = assigns) do
  ~H"""
  <div class="mx-auto max-w-sm">
   <.header class="text-center">Confirm Account</.header>
```

```
    <.simple_form for={@form} id="confirmation_form" phx-submit="confirm_account">
      <input type="hidden" name={@form[:token].name} value={@form[:token].value} />
      <:actions>
        <.button phx-disable-with="Confirming..." class="w-full">Confirm my account</.button>
      </:actions>
    </.simple_form>

    <p class="text-center mt-4">
      <.link href={~p"/users/register"}>Register</.link>
      | <.link href={~p"/users/log_in"}>Log in</.link>
    </p>
  </div>
  """
end

def mount(%{"token" => token}, _session, socket) do
  form = to_form(%{"token" => token}, as: "user")
  {:ok, assign(socket, form: form), temporary_assigns: [form: nil]}
end

# Do not log in the user after confirmation to avoid a
# leaked token giving the user access to the account.
def handle_event("confirm_account", %{"user" => %{"token" => token}}, socket) do
  case Accounts.confirm_user(token) do
    {:ok, _} ->
      {:noreply,
       socket
       |> put_flash(:info, "User confirmed successfully.")
       |> redirect(to: ~p"/")}

    :error ->
      # If there is a current user and the account was already confirmed,
      # then odds are that the confirmation link was already visited, either
      # by some automation or by the user themselves, so we redirect without
      # a warning message.
      case socket.assigns do
        %{current_user: %{confirmed_at: confirmed_at}} when not is_nil(confirmed_at) ->
          {:noreply, redirect(socket, to: ~p"/")}

        %{} ->
          {:noreply,
```

```
            socket
            |> put_flash(:error, "User confirmation link is invalid or it has expired.")
            |> redirect(to: ~p"/")}
        end
    end
  end
end
defmodule HubWeb.UserForgotPasswordLive do
  use HubWeb, :live_view

  alias Hub.Accounts

  def render(assigns) do
    ~H"""
    <div class="mx-auto max-w-sm">
      <.header class="text-center">
        Forgot your password?
        <:subtitle>We'll send a password reset link to your inbox</:subtitle>
      </.header>

      <.simple_form for={@form} id="reset_password_form" phx-submit="send_email">
        <.input field={@form[:email]} type="email" placeholder="Email" required />
        <:actions>
          <.button phx-disable-with="Sending..." class="w-full">
            Send password reset instructions
          </.button>
        </:actions>
      </.simple_form>
      <p class="text-center text-sm mt-4">
        <.link href={~p"/users/register"}>Register</.link>
        | <.link href={~p"/users/log_in"}>Log in</.link>
      </p>
    </div>
    """
  end

  def mount(_params, _session, socket) do
    {:ok, assign(socket, form: to_form(%{}, as: "user"))}
  end

  def handle_event("send_email", %{"user" => %{"email" => email}}, socket) do
    if user = Accounts.get_user_by_email(email) do
```

```
      Accounts.deliver_user_reset_password_instructions(
        user,
        &url(~p"/users/reset_password/#{&1}")
      )
    end

    info =
      "If your email is in our system, you will receive instructions to reset your password shortly."

    {:noreply,
     socket
     |> put_flash(:info, info)
     |> redirect(to: ~p"/")}
  end
end
defmodule HubWeb.UserLoginLive do
  use HubWeb, :live_view

  def render(assigns) do
    ~H"""
    <div class="mx-auto max-w-sm">
      <.header class="text-center">
        Log in to account
        <:subtitle>
          Don't have an account?
          <.link navigate={~p"/users/register"} class="font-semibold text-brand hover:underline">
            Sign up
          </.link>
          for an account now.
        </:subtitle>
      </.header>

      <.simple_form for={@form} id="login_form" action={~p"/users/log_in"} phx-update="ignore">
        <.input field={@form[:email]} type="email" label="Email" required />
        <.input field={@form[:password]} type="password" label="Password" required />

        <:actions>
          <.input field={@form[:remember_me]} type="checkbox" label="Keep me logged in" />
          <.link href={~p"/users/reset_password"} class="text-sm font-semibold">
            Forgot your password?
          </.link>
        </:actions>
```

```
      <:actions>
        <.button phx-disable-with="Logging in..." class="w-full">
          Log in <span aria-hidden="true">→</span>
        </.button>
      </:actions>
    </.simple_form>
  </div>
  """
 end

 def mount(_params, _session, socket) do
  email = Phoenix.Flash.get(socket.assigns.flash, :email)
  form = to_form(%{"email" => email}, as: "user")
  {:ok, assign(socket, form: form), temporary_assigns: [form: form]}
 end
end
defmodule HubWeb.UserRegistrationLive do
 use HubWeb, :live_view

 alias Hub.Accounts
 alias Hub.Accounts.User

 def render(assigns) do
  ~H"""
  <div class="mx-auto max-w-sm">
    <.header class="text-center">
      Register for an account
      <:subtitle>
        Already registered?
        <.link navigate={~p"/users/log_in"} class="font-semibold text-brand hover:underline">
          Log in
        </.link>
        to your account now.
      </:subtitle>
    </.header>

    <.simple_form
      for={@form}
      id="registration_form"
      phx-submit="save"
      phx-change="validate"
      phx-trigger-action={@trigger_submit}
```

```
      action={~p"/users/log_in?_action=registered"}
      method="post"
    >
      <.error :if={@check_errors}>
        Oops, something went wrong! Please check the errors below.
      </.error>

      <.input field={@form[:email]} type="email" label="Email" required />
      <.input field={@form[:password]} type="password" label="Password" required />

      <:actions>
        <.button phx-disable-with="Creating account..." class="w-full">Create an account</.button>
      </:actions>
    </.simple_form>
  </div>
  """
end

def mount(_params, _session, socket) do
  changeset = Accounts.change_user_registration(%User{})

  socket =
    socket
    |> assign(trigger_submit: false, check_errors: false)
    |> assign_form(changeset)

  {:ok, socket, temporary_assigns: [form: nil]}
end

def handle_event("save", %{"user" => user_params}, socket) do
  case Accounts.register_user(user_params) do
    {:ok, user} ->
      {:ok, _} =
        Accounts.deliver_user_confirmation_instructions(
          user,
          &url(~p"/users/confirm/#{&1}")
        )

      changeset = Accounts.change_user_registration(user)
      {:noreply, socket |> assign(trigger_submit: true) |> assign_form(changeset)}

    {:error, %Ecto.Changeset{} = changeset} ->
```

```
      {:noreply, socket |> assign(check_errors: true) |> assign_form(changeset)}
    end
  end

  def handle_event("validate", %{"user" => user_params}, socket) do
    changeset = Accounts.change_user_registration(%User{}, user_params)
    {:noreply, assign_form(socket, Map.put(changeset, :action, :validate))}
  end

  defp assign_form(socket, %Ecto.Changeset{} = changeset) do
    form = to_form(changeset, as: "user")

    if changeset.valid? do
      assign(socket, form: form, check_errors: false)
    else
      assign(socket, form: form)
    end
  end
end
defmodule HubWeb.UserResetPasswordLive do
  use HubWeb, :live_view

  alias Hub.Accounts

  def render(assigns) do
    ~H"""
    <div class="mx-auto max-w-sm">
      <.header class="text-center">Reset Password</.header>

      <.simple_form
        for={@form}
        id="reset_password_form"
        phx-submit="reset_password"
        phx-change="validate"
      >
        <.error :if={@form.errors != []}>
          Oops, something went wrong! Please check the errors below.
        </.error>

        <.input field={@form[:password]} type="password" label="New password" required />
        <.input
          field={@form[:password_confirmation]}
```

```
      type="password"
      label="Confirm new password"
      required
    />
    <:actions>
      <.button phx-disable-with="Resetting..." class="w-full">Reset Password</.button>
    </:actions>
  </.simple_form>

  <p class="text-center text-sm mt-4">
    <.link href={~p"/users/register"}>Register</.link>
    | <.link href={~p"/users/log_in"}>Log in</.link>
  </p>
 </div>
 """
end

def mount(params, _session, socket) do
 socket = assign_user_and_token(socket, params)

 form_source =
  case socket.assigns do
   %{user: user} ->
    Accounts.change_user_password(user)

   _ ->
    %{}
  end

 {:ok, assign_form(socket, form_source), temporary_assigns: [form: nil]}
end

# Do not log in the user after reset password to avoid a
# leaked token giving the user access to the account.
def handle_event("reset_password", %{"user" => user_params}, socket) do
 case Accounts.reset_user_password(socket.assigns.user, user_params) do
  {:ok, _} ->
   {:noreply,
    socket
    |> put_flash(:info, "Password reset successfully.")
    |> redirect(to: ~p"/users/log_in")}
```

```
      {:error, changeset} ->
        {:noreply, assign_form(socket, Map.put(changeset, :action, :insert))}
    end
  end

  def handle_event("validate", %{"user" => user_params}, socket) do
    changeset = Accounts.change_user_password(socket.assigns.user, user_params)
    {:noreply, assign_form(socket, Map.put(changeset, :action, :validate))}
  end

  defp assign_user_and_token(socket, %{"token" => token}) do
    if user = Accounts.get_user_by_reset_password_token(token) do
      assign(socket, user: user, token: token)
    else
      socket
      |> put_flash(:error, "Reset password link is invalid or it has expired.")
      |> redirect(to: ~p"/")
    end
  end

  defp assign_form(socket, %{} = source) do
    assign(socket, :form, to_form(source, as: "user"))
  end
end
defmodule HubWeb.UserSettingsLive do
  use HubWeb, :live_view

  alias Hub.Accounts

  def render(assigns) do
    ~H"""
    <.header class="text-center">
      Account Settings
      <:subtitle>Manage your account email address and password settings</:subtitle>
    </.header>

    <div class="space-y-12 divide-y">
      <div>
        <.simple_form
          for={@email_form}
          id="email_form"
          phx-submit="update_email"
```

```
    phx-change="validate_email"
  >
    <.input field={@email_form[:email]} type="email" label="Email" required />
    <.input
      field={@email_form[:current_password]}
      name="current_password"
      id="current_password_for_email"
      type="password"
      label="Current password"
      value={@email_form_current_password}
      required
    />
    <:actions>
      <.button phx-disable-with="Changing...">Change Email</.button>
    </:actions>
  </.simple_form>
</div>
<div>
  <.simple_form
    for={@password_form}
    id="password_form"
    action={~p"/users/log_in?_action=password_updated"}
    method="post"
    phx-change="validate_password"
    phx-submit="update_password"
    phx-trigger-action={@trigger_submit}
  >
    <input
      name={@password_form[:email].name}
      type="hidden"
      id="hidden_user_email"
      value={@current_email}
    />
    <.input field={@password_form[:password]} type="password" label="New password" required />
    <.input
      field={@password_form[:password_confirmation]}
      type="password"
      label="Confirm new password"
    />
    <.input
      field={@password_form[:current_password]}
      name="current_password"
```

```
          type="password"
          label="Current password"
          id="current_password_for_password"
          value={@current_password}
          required
        />
        <:actions>
          <.button phx-disable-with="Changing...">Change Password</.button>
        </:actions>
      </.simple_form>
    </div>
  </div>
  """
end

def mount(%{"token" => token}, _session, socket) do
  socket =
    case Accounts.update_user_email(socket.assigns.current_user, token) do
      :ok ->
        put_flash(socket, :info, "Email changed successfully.")

      :error ->
        put_flash(socket, :error, "Email change link is invalid or it has expired.")
    end

  {:ok, push_navigate(socket, to: ~p"/users/settings")}
end

def mount(_params, _session, socket) do
  user = socket.assigns.current_user
  email_changeset = Accounts.change_user_email(user)
  password_changeset = Accounts.change_user_password(user)

  socket =
    socket
    |> assign(:current_password, nil)
    |> assign(:email_form_current_password, nil)
    |> assign(:current_email, user.email)
    |> assign(:email_form, to_form(email_changeset))
    |> assign(:password_form, to_form(password_changeset))
    |> assign(:trigger_submit, false)
```

```
  {:ok, socket}
end

def handle_event("validate_email", params, socket) do
  %{"current_password" => password, "user" => user_params} = params

  email_form =
    socket.assigns.current_user
    |> Accounts.change_user_email(user_params)
    |> Map.put(:action, :validate)
    |> to_form()

  {:noreply, assign(socket, email_form: email_form, email_form_current_password: password)}
end

def handle_event("update_email", params, socket) do
  %{"current_password" => password, "user" => user_params} = params
  user = socket.assigns.current_user

  case Accounts.apply_user_email(user, password, user_params) do
    {:ok, applied_user} ->
      Accounts.deliver_user_update_email_instructions(
        applied_user,
        user.email,
        &url(~p"/users/settings/confirm_email/#{&1}")
      )

      info = "A link to confirm your email change has been sent to the new address."
      {:noreply, socket |> put_flash(:info, info) |> assign(email_form_current_password: nil)}

    {:error, changeset} ->
      {:noreply, assign(socket, :email_form, to_form(Map.put(changeset, :action, :insert)))}
  end
end

def handle_event("validate_password", params, socket) do
  %{"current_password" => password, "user" => user_params} = params

  password_form =
    socket.assigns.current_user
    |> Accounts.change_user_password(user_params)
    |> Map.put(:action, :validate)
```

```
      |> to_form()

    {:noreply, assign(socket, password_form: password_form, current_password: password)}
  end

  def handle_event("update_password", params, socket) do
    %{"current_password" => password, "user" => user_params} = params
    user = socket.assigns.current_user

    case Accounts.update_user_password(user, password, user_params) do
      {:ok, user} ->
        password_form =
          user
          |> Accounts.change_user_password(user_params)
          |> to_form()

        {:noreply, assign(socket, trigger_submit: true, password_form: password_form)}

      {:error, changeset} ->
        {:noreply, assign(socket, password_form: to_form(changeset))}
    end
  end
end
<!DOCTYPE html>
<html lang="en" class="[scrollbar-gutter:stable]">

<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <meta name="csrf-token" content={get_csrf_token()} />
  <.live_title default="AnomalyHub">
    AnomalyHub
  </.live_title>
  <link phx-track-static rel="stylesheet" href={~p"/assets/app.css"} />
  <script defer phx-track-static type="text/javascript" src={~p"/assets/app.js"}>
  </script>
</head>

<body class="bg-white">
  <div class="flex items-center justify-between px-4 sm:px-6 lg:px-8 py-4 border-b border-gray-200">
    <div class="flex items-center gap-4">
      <a href="/">
```

```
      <img src={~p"/images/logo.png"} width="36" />
    </a>
    <span class="text-2xl font-extrabold text-black" style="font-family: 'Poppins', sans-serif;">
      AnomalyHub
    </span>
  </div>
  <ul class="flex items-center gap-4">
    <%= if @current_user do %>
      <li class="text-sm font-bold text-black">
        <%= @current_user.email %>
      </li>
      <li>
        <.link href={~p"/users/settings"}
          class="text-sm font-medium text-white bg-black px-3 py-1.5 rounded hover:bg-gray-800 transition">
          Settings
        </.link>
      </li>
      <li>
        <.link href={~p"/users/log_out"} method="delete"
          class="text-sm font-medium text-white bg-black px-3 py-1.5 rounded hover:bg-gray-800 transition">
          Log out
        </.link>
      </li>
      <% else %>
        <li>
          <.link href={~p"/users/register"}
            class="text-sm font-medium text-white bg-black px-3 py-1.5 rounded hover:bg-gray-800 transition">
            Register
          </.link>
        </li>
        <li>
          <.link href={~p"/users/log_in"}
            class="text-sm font-medium text-white bg-black px-3 py-1.5 rounded hover:bg-gray-800 transition">
            Log in
          </.link>
        </li>
        <% end %>
    </ul>
  </div>

  {@inner_content}
</body>
```

```
</html>
defmodule Hub.Repo.Migrations.CreateUsersAuthTables do
  use Ecto.Migration

  def change do
    execute "CREATE EXTENSION IF NOT EXISTS citext", ""

    create table(:users) do
      add :email, :citext, null: false
      add :hashed_password, :string, null: false
      add :confirmed_at, :utc_datetime

      timestamps(type: :utc_datetime)
    end

    create unique_index(:users, [:email])

    create table(:users_tokens) do
      add :user_id, references(:users, on_delete: :delete_all), null: false
      add :token, :binary, null: false
      add :context, :string, null: false
      add :sent_to, :string

      timestamps(type: :utc_datetime, updated_at: false)
    end

    create index(:users_tokens, [:user_id])
    create unique_index(:users_tokens, [:context, :token])
  end
end
defmodule Hub.Repo.Migrations.CreateSensors do
  use Ecto.Migration

  def change do
    create table(:sensors) do
      add :name, :string
      add :api_key, :string
      add :user_id, references(:users, on_delete: :delete_all), null: false

      timestamps(type: :utc_datetime)
    end
```

```
    create unique_index(:sensors, [:api_key])
    create index(:sensors, [:user_id])
  end
end
defmodule Hub.Repo.Migrations.CreateTrafficRecords do
  use Ecto.Migration

  def change do
    create table(:traffic_records) do
      add(:timestamp, :integer, null: false)
      add(:packet_count, :integer, null: false)
      add(:unique_src_ip_count, :integer, null: false)
      add(:proto_counter, :jsonb, null: false, default: "{}")
      add(:tcp_syn_count, :integer, null: false, default: 0)
      add(:top_dst_ports, {:array, {:array, :integer}}, null: false, default: [])

      add(:anomaly, :boolean, null: false, default: false)
      add(:cusum_anomaly, :boolean, null: false, default: false)
      add(:isolation_anomaly, :boolean, null: false, default: false)

      add(:sensor_id, references(:sensors, on_delete: :delete_all), null: false)

      timestamps()
    end

    create(index(:traffic_records, [:sensor_id]))
    create(index(:traffic_records, [:timestamp]))
  end
end
# Script for populating the database. You can run it as:
#
#     mix run priv/repo/seeds.exs
#
# Inside the script, you can read and write to any of your
# repositories directly:
#
#     Hub.Repo.insert!(%Hub.SomeSchema{})
#
# We recommend using the bang functions (`insert!`, `update!`
# and so on) as they will fail if something goes wrong.
```

```
alias Hub.Repo
alias Hub.Accounts.User
alias Hub.Sensors.Sensor
alias Hub.TrafficRecords.TrafficRecord

# 1. Створення користувача
user =
  Repo.insert!(%User{
    email: "test@example.com",
    hashed_password: Bcrypt.hash_pwd_salt("password123")
  })

# 2. Створення сенсора
sensor =
  Repo.insert!(%Sensor{
    name: "Test Sensor 1",
    api_key: UUID.uuid4(),
    user_id: user.id
  })

# 3. Генерація фейкових traffic_records
Enum.each(1..100, fn i ->
  timestamp = Timex.now() |> Timex.shift(minutes: -i * 5) |> Timex.to_unix()

  proto_counter = %{
    "6" => Enum.random(300..400),
    "17" => Enum.random(50..100),
    "1" => Enum.random(10..30)
  }

  top_dst_ports = [
    [80, Enum.random(150..250)],
    [443, Enum.random(80..150)],
    [22, Enum.random(30..70)]
  ]

  Repo.insert!(%TrafficRecord{
    sensor_id: sensor.id,
    timestamp: timestamp,
    packet_count: Enum.random(400..600),
    unique_src_ip_count: Enum.random(80..120),
    proto_counter: proto_counter,
```

```
    tcp_syn_count: Enum.random(300..400),
    top_dst_ports: top_dst_ports,
    anomaly: Enum.random([true, false, false]),
    cusum_anomaly: Enum.random([true, false]),
    isolation_anomaly: Enum.random([true, false])
  })
end)

IO.puts("☑ Seeds вставлено: test@example.com / password123")
defmodule Hub.MixProject do
  use Mix.Project

  def project do
    [
      app: :hub,
      version: "0.1.0",
      elixir: "~> 1.14",
      elixirc_paths: elixirc_paths(Mix.env()),
      start_permanent: Mix.env() == :prod,
      aliases: aliases(),
      deps: deps()
    ]
  end

  # Configuration for the OTP application.
  #
  # Type `mix help compile.app` for more information.
  def application do
    [
      mod: {Hub.Application, []},
      extra_applications: [:logger, :runtime_tools]
    ]
  end

  # Specifies which paths to compile per environment.
  defp elixirc_paths(:test), do: ["lib", "test/support"]
  defp elixirc_paths(_), do: ["lib"]

  # Specifies your project dependencies.
  #
  # Type `mix help deps` for examples and options.
  defp deps do
```

```
  [
    {:bcrypt_elixir, "~> 3.0"},
    {:phoenix, "~> 1.7.21"},
    {:phoenix_ecto, "~> 4.5"},
    {:ecto_sql, "~> 3.10"},
    {:postgrex, ">= 0.0.0"},
    {:phoenix_html, "~> 4.1"},
    {:phoenix_live_reload, "~> 1.2", only: :dev},
    {:phoenix_live_view, "~> 1.0"},
    {:floki, ">= 0.30.0", only: :test},
    {:phoenix_live_dashboard, "~> 0.8.3"},
    {:esbuild, "~> 0.8", runtime: Mix.env() == :dev},
    {:tailwind, "~> 0.2.0", runtime: Mix.env() == :dev},
    {:heroicons,
     github: "tailwindlabs/heroicons",
     tag: "v2.1.1",
     sparse: "optimized",
     app: false,
     compile: false,
     depth: 1},
    {:swoosh, "~> 1.5"},
    {:finch, "~> 0.13"},
    {:telemetry_metrics, "~> 1.0"},
    {:telemetry_poller, "~> 1.0"},
    {:gettext, "~> 0.26"},
    {:jason, "~> 1.2"},
    {:dns_cluster, "~> 0.1.1"},
    {:bandit, "~> 1.5"},
    {:uuid, "~> 1.1"},
    {:httpoison, "~> 1.8"},
    {:timex, "~> 3.7"}
  ]
end

# Aliases are shortcuts or tasks specific to the current project.
# For example, to install project dependencies and perform other setup tasks, run:
#
#     $ mix setup
#
# See the documentation for `Mix` for more info on aliases.
defp aliases do
  [
```

```
      setup: ["deps.get", "ecto.setup", "assets.setup", "assets.build"],
      "ecto.setup": ["ecto.create", "ecto.migrate", "run priv/repo/seeds.exs"],
      "ecto.reset": ["ecto.drop", "ecto.setup"],
      test: ["ecto.create --quiet", "ecto.migrate --quiet", "test"],
      "assets.setup": ["tailwind.install --if-missing", "esbuild.install --if-missing"],
      "assets.build": ["tailwind hub", "esbuild hub"],
      "assets.deploy": [
        "tailwind hub --minify",
        "esbuild hub --minify",
        "phx.digest"
      ]
    ]
  end
end
# Find eligible builder and runner images on Docker Hub. We use Ubuntu/Debian
# instead of Alpine to avoid DNS resolution issues in production.
#
# https://hub.docker.com/r/hexpm/elixir/tags?page=1&name=ubuntu
# https://hub.docker.com/_/ubuntu?tab=tags
#
# This file is based on these images:
#
#   - https://hub.docker.com/r/hexpm/elixir/tags - for the build image
#   - https://hub.docker.com/_/debian?tab=tags&page=1&name=bullseye-20250428-slim - for the release image
#   - https://pkgs.org/ - resource for finding needed packages
#   - Ex: hexpm/elixir:1.18.3-erlang-27.3.4-debian-bullseye-20250428-slim
#
ARG ELIXIR_VERSION=1.18.3
ARG OTP_VERSION=27.3.4
ARG DEBIAN_VERSION=bullseye-20250428-slim

ARG BUILDER_IMAGE="hexpm/elixir:${ELIXIR_VERSION}-erlang-${OTP_VERSION}-debian-
${DEBIAN_VERSION}"
ARG RUNNER_IMAGE="debian:${DEBIAN_VERSION}"

FROM ${BUILDER_IMAGE} as builder

# install build dependencies
RUN apt-get update -y && apt-get install -y build-essential git \
  && apt-get clean && rm -f /var/lib/apt/lists/*_*

# Установка Node.js і npm у фазі builder
```

```
RUN apt-get update && \
    apt-get install -y curl && \
    curl -fsSL https://deb.nodesource.com/setup_18.x | bash - && \
    apt-get install -y nodejs

# prepare build dir
WORKDIR /app

# install hex + rebar
RUN mix local.hex --force && \
    mix local.rebar --force

# set build ENV
ENV MIX_ENV="prod"

# install mix dependencies
COPY mix.exs mix.lock ./
RUN mix deps.get --only $MIX_ENV
RUN mkdir config

# copy compile-time config files before we compile dependencies
# to ensure any relevant config change will trigger the dependencies
# to be re-compiled.
COPY config/config.exs config/${MIX_ENV}.exs config/
RUN mix deps.compile

COPY priv priv

COPY lib lib

COPY assets assets

# install deps
RUN npm install --prefix ./assets
# compile assets
RUN mix assets.deploy

# Compile the release
RUN mix compile

# Changes to config/runtime.exs don't require recompiling the code
COPY config/runtime.exs config/
```

```
COPY rel rel
RUN mix release

# start a new build stage so that the final image will only contain
# the compiled release and other runtime necessities
FROM ${RUNNER_IMAGE}

RUN apt-get update -y && \
  apt-get install -y libstdc++6 openssl libncurses5 locales ca-certificates \
  && apt-get clean && rm -f /var/lib/apt/lists/*_*

# Set the locale
RUN sed -i '/en_US.UTF-8/s/^# //g' /etc/locale.gen && locale-gen

ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV LC_ALL en_US.UTF-8

WORKDIR "/app"
RUN chown nobody /app

# set runner ENV
ENV MIX_ENV="prod"

# Only copy the final release from the build stage
COPY --from=builder --chown=nobody:root /app/_build/${MIX_ENV}/rel/hub ./

USER nobody

# If using an environment that doesn't automatically reap zombie processes, it is
# advised to add an init process such as tini via `apt-get install`
# above and adding an entrypoint. See https://github.com/krallin/tini for details
# ENTRYPOINT ["/tini", "--"]

CMD ["/app/bin/server"]
```