Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка програмної системи для виявлення аномальних транзакцій у задачах банківського моніторингу з використанням машинного навчання

Виконав(ла): студент(ка) IV курсу, групи СП-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

	(підпис)	Великанич М. І. (прізвище та ініціали)
Керівник	(підпис)	Стоянов Ю. М. (прізвище та ініціали)
Нормоконтроль	(підпис)	Стоянов Ю. М. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Петрик М. Р. (прізвище та ініціали)
Рецензент	(підпис)	(прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія Програмного Забезпечення
(шифр і назва спеціальності)

студенту Великаничу Максиму Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмної системи для виявлення аномальних транзакцій у задачах банківського моніторингу з використанням машинного навчання

Керівник роботи Стоянов Юрій Миколайович, канд. техн. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____.

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Вимога замовника, потреба замовника

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
Аналіз вимог до програмної системи
Проєктування та розробка програмної системи
Тестування, впровадження та підтримка
Безпека життєдіяльності, основи охорони праці
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності та			
основи охорони праці			
Нормоконтроль			
Дата видачі завдання			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Ознайомлення з завданням кваліфікаційної роботи	10.04.2026	*Виконано*
2	Збір та аналіз інформації за темою дослідження (огляд	13.04.2026-	*Виконано*
	методів виявлення карткового шахрайства, алгоритмів	15.04.2026	
	машинного навчання, технологій побудови веб		
	застосунків)		
3	Формування структури пояснювальної записки	17.04.2026	*Виконано*
4	Розробка технічного завдання, проєктування архітектури	20.04.2026-	*Виконано*
	системи, бази даних та вибір методів реалізації	23.04.2026	
	інтелектуальних модулів		
5	Реалізація програмного забезпечення: ML модуля	23.04.2026-	*Виконано*
	серверної частини на FastAPI та інтеграція з веб	31.04.2026	
	інтерфейсом на React		
6	Тестування функціоналу системи, оцінка точності	1.05.2026	*Виконано*
	класифікаційної моделі та валідація відповідей API		
7	Написання розділів пояснювальної записки	1.05.2026- 5.05.2026	*Виконано*
8	Написання розділу 4 «Безпека життєдіяльності та основи	6.05.2026	*Виконано*
	охорони праці»		
9	Оформлення висновків, списку використаних джерел,	7.05.2026	*Виконано*
	додатків		
10	Перевірка роботи керівником, внесення правок		*Виконано*
11	Нормоконтроль		*Виконано*
12	Перевірка кваліфікаційної роботи на плагіат		*Виконано*
13	Попередній захист кваліфікаційної роботи		*Виконано*
14	Захист кваліфікаційної роботи		

Студент _____ Великанич М. І.
(підпис) (прізвита ініціали)

Керівник роботи _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка програмної системи для виявлення аномальних транзакцій у задачах банківського моніторингу з використанням машинного навчання // ОР «Бакалавр» // Великанич Максим Іванович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-41 // Тернопіль, 2026 // С. 79, рис. – 21, табл. – 11, додат. – 41.

Ключові слова: виявлення шахрайства, банківський моніторинг, машинне навчання, класифікація транзакцій, незбалансовані дані, веб застосунок.

У кваліфікаційній роботі вирішується задача автоматизованого виявлення шахрайських транзакцій у банківських платіжних системах. Актуальність роботи зумовлена зростанням обсягів електронних платежів та збитків від карткового шахрайства, а також недостатньою ефективністю традиційних методів виявлення аномалій в умовах великих даних.

У роботі проведено аналіз предметної області, здійснено порівняльне дослідження методів машинного навчання для задачі бінарної класифікації на незбалансованих датасетах. Розроблено програмну систему, що включає модуль машинного навчання, серверну частину з REST API та вебінтерфейс для аналітика. Проведено тестування системи та верифікацію якості побудованої класифікаційної моделі.

ABSTRACT

Development of a Software System for Anomalous Transaction Detection in Banking Monitoring Tasks Using Machine Learning // Bachelor's Degree // Velykanych Maksym Ivanovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP-41 // Ternopil, 2026 // P. 79, fig. – 21, tabl. – 11, append. – 41.

Keywords: fraud detection, banking monitoring, machine learning, transaction classification, imbalanced data, web application.

The qualification work addresses the task of automated detection of fraudulent transactions in banking payment systems. The relevance of the work is driven by the growing volume of electronic payments and losses caused by card fraud, as well as the insufficient effectiveness of traditional anomaly detection methods under big data conditions.

The work includes an analysis of the subject domain, a comparative study of machine learning methods for binary classification on imbalanced datasets. A software system has been developed comprising a machine learning module, a server-side component with a REST API, and a web interface for security analysts. The system has been tested and the quality of the built classification model has been verified.

ЗМІСТ

ВСТУП

Масштабне поширення безготівкових розрахунків і стрімке зростання кількості онлайн-платежів відкрили нові можливості як для бізнесу, так і для зловмисників. Щороку фінансові установи по всьому світу зазнають значних збитків внаслідок несанкціонованих операцій із платіжними картками, причому шахрайські схеми постійно ускладнюються та еволюціонують. Системи захисту, що ґрунтуються на фіксованих правилах або статичних порогових значеннях, виявляються нездатними своєчасно реагувати на нові патерни зловживань, оскільки вони розраховані на вже відомі сценарії атак. За таких умов особливого значення набувають методи машинного навчання, що дозволяють виявляти відхилення від типової поведінки без необхідності заздалегідь описувати кожен можливий вид шахрайства. Додаткову складність становить виражений дисбаланс класів у реальних транзакційних даних: частка підозрілих операцій зазвичай не перевищує частки відсотка від загального потоку, що суттєво ускладнює навчання моделей і вимагає застосування спеціалізованих підходів до обробки та оцінювання результатів.

Метою кваліфікаційної роботи є розробка програмної системи для автоматизованого виявлення шахрайських транзакцій у банківських платіжних системах на основі методів машинного навчання.

Досягнення поставленої мети передбачає виконання таких задач: дослідити предметну область карткового шахрайства та проаналізувати наявні підходи до його виявлення.Виконати попередню обробку транзакційних даних з урахуванням незбалансованості класів та специфіки фінансових набору даних. Здійснити порівняльний аналіз алгоритмів машинного навчання для задачі бінарної класифікації на незбалансованих вибірках. Спроєктувати архітектуру програмної системи, що поєднує компонент машинного навчання, серверну частину та веб інтерфейс. Реалізувати конвеєр обробки машинного навчання із застосуванням методів балансування класів та відбору ознак. Розробити серверну частину системи з REST API для обслуговування одиночних і пакетних запитів на класифікацію. Створити веб інтерфейс для взаємодії аналітиків із системою в

режимі реального часу. Провести тестування системи та верифікацію якості побудованої моделі.

Об'єктом дослідження є процес виявлення шахрайських транзакцій у банківських платіжних системах в умовах суттєвого дисбалансу між класами даних.

Предметом дослідження є методи та алгоритми машинного навчання для класифікації фінансових транзакцій, а також засоби їх програмної інтеграції у повноцінну веб систему моніторингу.

Практичне значення одержаних результатів. Розроблена система надає аналітикам зручний інструмент для оперативного виявлення підозрілих операцій, як в індивідуальному, так і в пакетному режимі, з автоматичним збереженням історії перевірок. Архітектурне рішення на основі контейнеризації забезпечує відтворюваність середовища та спрощує розгортання системи у виробничій інфраструктурі банківської установи. Отримані результати можуть бути використані фахівцями з фінансової безпеки для зниження ризиків, пов'язаних із несанкціонованим використанням платіжних карток.

Кваліфікаційна робота бакалавра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто предметну область та сформульовано постановку задачі. Другий розділ присвячено проектуванню архітектури та реалізації програмної системи. Третій розділ містить результати тестування та оцінювання якості моделі. Четвертий розділ охоплює питання охорони праці та безпеки життєдіяльнос

РОЗДІЛ 1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Розробка будь-якої програмної системи розпочинається з ретельного аналізу предметної області та формулювання вимог. У цьому розділі розглянуто специфіку шахрайства в банківських платіжних системах, визначено проблематику задачі виявлення аномальних транзакцій, обґрунтовано вибір методів машинного навчання та побудовано модель варіантів використання розроблюваної системи.

1.1 Аналіз предметної області

Картковое шахрайство є однією з найбільш поширених форм фінансових злочинів у цифровому середовищі. За даними аналітичних звітів Nilson Report, глобальні збитки від несанкціонованих операцій із платіжними картками перевищують 30 мільярдів доларів на рік і продовжують зростати разом із збільшенням обсягів електронної комерції [1]. Зловмисники постійно вдосконалюють свої методи від класичного фішингу та скімінгу до складних атак із використанням викрадених облікових даних і синтетичних ідентифікаційних профілів, що суттєво ускладнює завдання захисту [2].

Традиційні методи протидії шахрайству спираються на систему детерміністичних правил: якщо сума транзакції перевищує певний поріг або операція здійснюється з незвичної географічної точки, система блокує її або надсилає запит на підтвердження. Такий підхід є прозорим і легко інтерпретованим, однак має суттєві обмеження він нездатний виявляти нові схеми шахрайства, не описані в базі правил, а підтримка актуального набору правил вимагає постійного ручного втручання фахівців [3]. Крім того, зловмисники, знаючи про існування фіксованих порогів, адаптують свою поведінку, залишаючись у допустимих межах.

Альтернативою є застосування методів машинного навчання, які дозволяють автоматично виявляти приховані патерни у транзакційних даних без необхідності явно описувати кожен можливий сценарій шахрайства. Сучасні дослідження

демонструють ефективність як класичних алгоритмів логістичної регресії, методу опорних векторів, дерев рішень, так і ансамблевих підходів на основі Random Forest та градієнтного бустингу [4]. Глибинне навчання також показує перспективні результати, однак потребує значно більших обчислювальних ресурсів і менш інтерпретованих моделей, що є критичним у фінансовій сфері [5].

Для дослідження та розробки системи використано загальновідомий датасет Credit Card Fraud Detection, підготовлений групою машинного навчання Університету Вільного Брюсселя і розміщений на платформі Kaggle [9]. Датасет містить 284 807 транзакцій, здійснених власниками карток протягом двох діб у вересні 2013 року. З них лише 492 транзакції (~0.17%) позначені як шахрайські. Усі числові ознаки V1–V28 отримано методом аналізу головних компонент (PCA) з метою анонімізації конфіденційних даних; у відкритому вигляді доступні лише ознаки Time та Amount. Цільова змінна Class приймає значення 0 (легітимна операція) або 1 (шахрайство).

Огляд існуючих комерційних рішень у сфері виявлення шахрайства зокрема, Featurespace ARIC, SAS Fraud Management, IBM Safer Payments свідчить про те, що вони є закритими корпоративними продуктами з високим порогом входу та вузькою орієнтацією на великі фінансові установи [7]. Відкритих, легко відтворюваних рішень із веб інтерфейсом, придатних для використання командами аналітиків середнього рівня, практично немає. Це визначає практичну цінність розроблюваної системи.

З технологічного боку, реалізація системи виявлення аномалій потребує інтеграції кількох компонентів: бібліотек машинного навчання для побудови та збереження моделі, серверного фреймворку для обслуговування запитів на класифікацію, реляційної бази даних для зберігання результатів та веб інтерфейсу для взаємодії з аналітиком. Серед бібліотек машинного навчання на мові Python домінує scikit-learn, добре задокументований інструментарій із широким набором алгоритмів, утилітами для попередньої обробки та оцінювання якості моделей [8]. Як серверний фреймворк доцільно розглядати FastAPI сучасний асинхронний інструмент, що забезпечує автоматичну генерацію OpenAPI документації та

вбудовану валідацію даних через Pydantic [9]. Для фронтендної частини бібліотека React є де-факто стандартом побудови односторінкових вебзастосунків із компонентною архітектурою [10].

1.2 Проблематика виявлення шахрайства та аналіз задачі

Виявлення шахрайських транзакцій належить до класу задач бінарної класифікації: кожній транзакції необхідно присвоїти мітку «легітимна» (0) або «шахрайська» (1). Попри простоту формулювання, практична реалізація цієї задачі пов'язана з низкою принципових труднощів, які потребують спеціальних підходів до навчання моделей та оцінювання їхньої якості.

Ключовою проблемою є екстремальний дисбаланс класів. У використовуваному датасеті частка шахрайських транзакцій складає лише 0.17% від загальної кількості співвідношення між класами становить приблизно 1:578. За таких умов наївний класифікатор, що відносить усі транзакції до класу «легітимних», матиме точність (Accuracy) близько 99.83% і при цьому не виявить жодного шахрайства. Це унаочнює принципову непридатність метрики Accuracy для оцінювання якості в подібних задачах [11].

Для коректного оцінювання якості класифікатора на незбалансованих даних необхідно звернутися до метрик, що явно враховують розподіл між класами. Precision (точність) показує, яка частка транзакцій, позначених моделлю як шахрайські, є справді шахрайськими тобто відображає рівень хибних спрацювань. Recall (повнота) вимірює, яку частку реально шахрайських транзакцій модель виявила тобто характеризує рівень пропущених загроз. Між цими метриками існує обернена залежність: підвищення чутливості моделі (більший Recall) зазвичай супроводжується збільшенням кількості хибних спрацювань (менший Precision) [12].

Найбільш інформативною інтегральною метрикою для даної задачі є площа під кривою Precision-Recall (PR-AUC, або Average Precision). На відміну від широко застосовуваної ROC-AUC, яка будується у просторі TPR/FPR і

залишається стабільно високою навіть на незбалансованих вибірках, PR-AUC явно відображає якість роботи моделі з міноритарним класом і є значно більш чутливою до реального рівня виявлення шахрайства [13]. Метрика F1 як гармонічне середнє Precision та Recall використовується як додатковий показник збалансованості між хибними спрацюваннями та пропусками.

Окрім вибору метрик, дисбаланс класів потребує врахування на етапі навчання моделі. Стандартне навчання без будь-яких коригувань призводить до того, що модель оптимізується переважно на мажоритарному класі та ігнорує міноритарний. Серед методів боротьби з дисбалансом виокремлюють: алгоритмічні підходи, методи передискретизації SMOTE (Synthetic Minority Over-sampling Technique), що генерує синтетичні зразки міноритарного класу, та методи субдискретизації мажоритарного класу [14]. При застосуванні SMOTE критично важливо виконувати передискретизацію виключно на навчальній вибірці, не допускаючи витоку синтетичних даних у тестову так зване data leakage, що призводить до оптимістично завищених оцінок якості [15].

Таким чином, задача виявлення шахрайських транзакцій формулюється як задача бінарної класифікації на екстремально незбалансованому датасеті, де основною метрикою оптимізації є PR-AUC, а якість моделі оцінюється за сукупністю показників: PR-AUC, ROC-AUC, F1, Precision та Recall.

1.3 Обґрунтування вибору методів машинного навчання

Вибір алгоритмів машинного навчання для задачі виявлення шахрайства визначається рядом взаємопов'язаних вимог: достатньою точністю класифікації на незбалансованих даних, інтерпретованістю моделі (що є важливим у фінансовій сфері), прийнятним часом навчання та можливістю ефективного налаштування гіперпараметрів. У роботі розглядаються три алгоритми: логістична регресія, Random Forest та XGBoost.

Логістична регресія є базовим лінійним класифікатором, що моделює ймовірність належності спостереження до класу через логістичну функцію від

лінійної комбінації ознак [16]. Її переваги полягають у простоті, швидкості навчання та легкій інтерпретованості коефіцієнтів. Водночас лінійна природа моделі обмежує її здатність виявляти складні нелінійні залежності між ознаками транзакцій, що є суттєвим недоліком у задачах, де патерни шахрайства можуть бути нелінійними та складно структурованими. Логістична регресія використовується як базова модель для порівняння з більш складними підходами.

Random Forest - ансамблевий алгоритм, що будує множину дерев рішень на випадкових підвибірках даних і ознак та агрегує їхні передбачення голосуванням [17]. Така стратегія суттєво знижує дисперсію порівняно з окремим деревом і забезпечує стійкість до перенавчання. Важливою перевагою є вбудований механізм оцінки важливості ознак (feature importance), що дозволяє інтерпретувати внесок кожної змінної у класифікацію. Random Forest природно підтримує параметр, що автоматично коригує ваги класів обернено пропорційно до їхньої частоти в навчальній вибірці.

XGBoost реалізує алгоритм градієнтного бустингу - послідовного навчання дерев рішень, де кожне наступне дерево коригує помилки попередніх [18]. Алгоритм відрізняється високою предикативною якістю та широко застосовується у задачах виявлення шахрайства. Серед переваг вбудована регуляризація L1/L2, стійкість до пропущених значень та ефективна реалізація, що забезпечує швидке навчання навіть на великих датасетах. Недоліком є більша кількість гіперпараметрів порівняно з Random Forest, що ускладнює налаштування.

Для вибору оптимальних гіперпараметрів обох ансамблевих моделей застосовується RandomizedSearchCV із крос-валідацією та метрикою оптимізації average_precision, що відповідає PR-AUC. Випадковий пошук замість повного перебору (GridSearchCV) обрано з міркувань обчислювальної ефективності при достатній якості результату [19].

Глибинні нейронні мережі не розглядаються у даній роботі з кількох причин. По-перше, задокументовано, що ансамблеві методи на табличних даних демонструють результати, порівнянні з нейронними мережами або кращі за них, при значно меншій обчислювальній вартості [20]. По-друге, нейронні мережі є

менш інтерпретованими моделями, що суперечить вимогам до прозорості рішень у фінансовій сфері. По-третє, обмежений обсяг міноритарного класу (492 зразки) є недостатнім для ефективного навчання глибоких архітектур без суттєвої аугментації даних.

1.4 Моделювання варіантів використання системи

Моделювання варіантів використання є одним із ключових інструментів об'єктно-орієнтованого аналізу вимог. Воно дозволяє формалізувати взаємодію між користувачами та системою ще на етапі аналізу, до початку проєктування та реалізації. Діаграма варіантів використання відображає, які дії може виконувати кожен актор у системі та як окремі варіанти використання пов'язані між собою. Такий підхід забезпечує спільне розуміння функціональних вимог між замовником і розробником, а також формує основу для подальшого планування тестових сценаріїв [21].

Розроблювана система має одного зовнішнього актора - аналітик. Це автентифікований користувач, який безпосередньо взаємодіє з веб інтерфейсом системи: реєструється, виконує вхід, надсилає транзакції на перевірку та переглядає результати класифікації. Компонент машинного навчання є внутрішнім підсистемним елементом і не виступає самостійним зовнішнім актором, оскільки не ініціює взаємодію з системою самостійно він лише обробляє запити, що надходять від аналітика через REST API.

Система охоплює п'ять варіантів використання, діаграма яких наведена на рисунку 1. Варіант використання “Реєстрація облікового запису” включає (include) варіант “Автефікація”, оскільки є обов'язковою складовою обох сценаріїв. Варіант “Перегляд історії перевірок користувача” розширює (extend) основний робочий процес аналітика, будучи необов'язковою, але логічно пов'язаною дією після виконання перевірок.

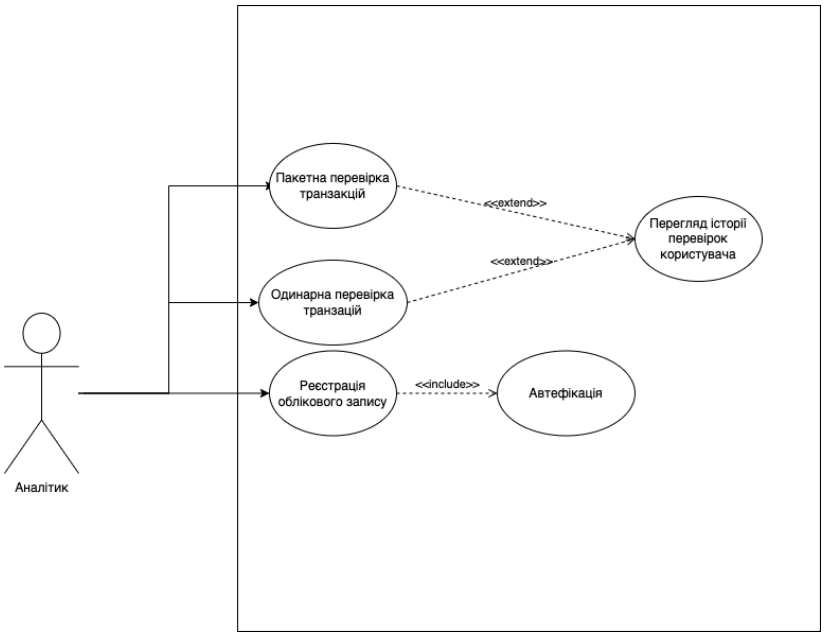

Рисунок 1.4 - Діаграма варіантів використання системи

Нижче наведено детальні специфікації кожного варіанту використання.

Варіант використання 1: “Реєстрація облікового запису”

Короткий опис: створення нового облікового запису аналітика для отримання доступу до функцій системи.

Передумови: користувач не має зареєстрованого облікового запису в системі; веб інтерфейс доступний.

Основний потік подій:

- аналітик відкриває сторінку реєстрації та заповнює форму, вказуючи електронну адресу та пароль;
- система перевіряє коректність формату введених даних засобами клієнтської валідації;
- клієнтська частина надсилає POST-запит до ендпоінту /auth/register із введеними даними;
- серверна частина перевіряє унікальність електронної адреси в базі даних;
- пароль хешується за алгоритмом bcrypt і обліковий запис зберігається в таблиці users;
- система повертає підтвердження успішної реєстрації, користувач перенаправляється на сторінку входу.

Альтернативний потік: якщо електронна адреса вже зареєстрована, сервер повертає повідомленням про помилку, яке відображається у формі.

Після умови: обліковий запис створено, аналітик може виконати автентифікацію.

Варіант використання 2: “Автентифікація”

Короткий опис: підтвердження особи аналітика та отримання токена доступу для роботи із захищеними функціями системи.

Передумови: обліковий запис аналітика існує в базі даних системи.

Основний потік подій:

- аналітик безпеки відкриває сторінку входу та вводить електронну адресу і пароль;
- клієнтська частина надсилає POST-запит до ендпоінту /auth/login;
- серверна частина знаходить запис користувача за електронною адресою та перевіряє відповідність введеного пароля збереженому bcrypt-хешу;
- у разі успіху сервер генерує JWT-токен із обмеженим терміном дії та повертає його клієнту;
- клієнтська частина зберігає токен у пам'яті застосунку та перенаправляє користувача на головну сторінку.

Альтернативний потік: якщо електронна адреса не знайдена або пароль не збігається, сервер повертає відповідь про помилку, форма відображає повідомлення про невірні облікові дані.

Після умови: аналітик безпеки автентифікований, JWT-токен отримано, доступ до захищених ендпоінтів відкрито.

Варіант використання 3: «Перевірка одиночної транзакції»

Короткий опис: надсилання ознак однієї транзакції на класифікацію та отримання результату із зазначенням ймовірності шахрайства.

Передумови: аналітик безпеки автентифікований у системі, JWT-токен дійсний.

Основний потік подій:

- користувач переходить на сторінку одиночної перевірки та вводить числові ознаки транзакції у відповідні поля форми;
- клієнтська частина надсилає POST-запит до ендпоінту /predict із даними транзакції та JWT-токеном у заголовку Authorization;
- серверна частина валідує вхідні дані черезсхему та передає їх до ML
- ML застосовує збережений StandardScaler для нормалізації ознак і передає масив до навченої моделі;
- модель повертає мітку класу (0 - легітимна, 1 - шахрайська) та ймовірність належності до класу шахрайства;
- результат класифікації зберігається в таблиці predictions бази даних із прив'язкою до ідентифікатора поточного користувача;
- клієнтська частина відображає результат: мітку класу, ймовірність та відповідне візуальне позначення.

Альтернативний потік: якщо токен прострочений або відсутній, сервер повертає відповідь із помилкою, клієнт перенаправляє користувача на сторінку входу.

Після умови: результат класифікації відображено в інтерфейсі та збережено в базі даних.

Варіант використання 4: «Пакетна перевірка транзакцій»

Короткий опис: надсилання масиву транзакцій на одночасну класифікацію та отримання зведеного звіту з результатами для кожної транзакції.

Передумови: користувач автентифікований у системі; масив містить щонайменше одну транзакцію.

Основний потік подій:

- аналітик безпеки переходить на сторінку пакетної перевірки та завантажує або вводить масив транзакцій;
- клієнтська частина надсилає POST-запит до ендпоінту /predict/batch із масивом транзакцій та JWT-токеном;

- серверна частина валідує кожен запис масиву через схему;
- pipeline обробляє всі транзакції за один виклик моделі, повертаючи масив міток і ймовірностей;
- результати зберігаються в таблиці predictions як єдиний запис типу «batch» із JSON-масивом у полі result;
- клієнтська частина відображає зведену таблицю результатів із позначенням підозрілих транзакцій.

Альтернативний потік: якщо будь-який запис масиву містить некоректні дані, сервер повертає відповідь із описом помилки валідації.

Після умови: результати класифікації всіх транзакцій відображено та збережено в базі даних.

Варіант використання 5: «Перегляд історії перевірок»

Короткий опис: перегляд хронологічного переліку раніше виконаних запитів на класифікацію із відображенням типу запиту та отриманих результатів.

Передумови: користувач автентифікований у системі.

Основний потік подій:

- користувач переходить на сторінку історії перевірок;
- клієнтська частина надсилає GET-запит до ендпоінту /predict/history із JWT-токеном у заголовку;
- серверна частина вибирає з таблиці predictions усі записи, що належать поточному користувачеві, впорядковані за датою створення у спадному порядку;
- клієнтська частина відображає список запитів із зазначенням типу (одиночний або пакетний), дати та короткого підсумку результату.

Після умови: користувач переглянув повну історію своїх запитів на класифікацію.

1.5 Висновки до розділу 1

У першому розділі проведено комплексний аналіз предметної області та сформульовано вимоги до розроблюваної програмної системи. Встановлено, що

стрімке зростання обсягів електронних платежів і постійне вдосконалення шахрайських схем роблять автоматизоване виявлення аномальних транзакцій актуальним науково-технічним завданням. Показано, що традиційні підходи на основі фіксованих правил є принципово обмеженими та не здатні адаптуватися до нових загроз, тоді як методи машинного навчання забезпечують необхідну гнучкість і здатність до узагальнення.

Детально розглянуто проблематику виявлення шахрайства: обґрунтовано непридатність метрики Accuracy для задач із екстремальним дисбалансом класів та визначено PR-AUC як основну метрику оптимізації. Розкрито роль методу SMOTE у боротьбі з дисбалансом класів та наголошено на важливості дотримання коректного порядку його застосування для уникнення витоку даних.

Здійснено порівняльний огляд трьох алгоритмів машинного навчання логістичної регресії, Random Forest та XGBoost. Для кожного алгоритму визначено переваги, обмеження та доцільність застосування у контексті задачі виявлення шахрайства. Обґрунтовано недоцільність використання глибинних нейронних мереж для табличних даних обмеженого обсягу, де ансамблеві методи демонструють порівняну або вищу ефективність при значно меншій обчислювальній вартості та кращій інтерпретованості.

Побудовано модель варіантів використання системи, що охоплює п'ять функціональних сценаріїв: реєстрацію, автентифікацію, одиночну та пакетну перевірку транзакцій і перегляд історії. Описано єдиного зовнішнього актора та зв'язки між варіантами використання з використанням стереотипів «include» та «extend». Сформовані вимоги є достатньою основою для переходу до етапу проєктування архітектури системи, що розглядається у наступному розділі.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИИ

Розділ присвячено повному циклу проєктування та розробки програмної системи виявлення шахрайських транзакцій. Послідовно розглядаються: вибір і обґрунтування архітектурних рішень, технологічного стеку, схема бази даних, проєктування серверної частини з діаграмою класів, впровадження машиного навчання з порівняльним аналізом моделей та реалізація веб інтерфейсу. Кожне проєктне рішення обґрунтовується з позиції функціональних вимог, визначених у першому розділі [22].

2.1 Проєктування архітектури системи

Архітектура програмної системи визначає спосіб організації її компонентів, характер взаємодії між ними та розподіл відповідальності. Вибір архітектурного рішення є одним із найбільш відповідальних етапів проєктування, оскільки він суттєво впливає на масштабованість, супроводжуваність і тестованість системи .

Для розроблюваної системи обрано трирівневу архітектуру, що складається з рівня представлення, рівня бізнес-логіки та рівня даних (рисунок 2). Така архітектура є загальноприйнятим стандартом для вебзастосунків, що поєднують ML-компонент із RESTful API [23].

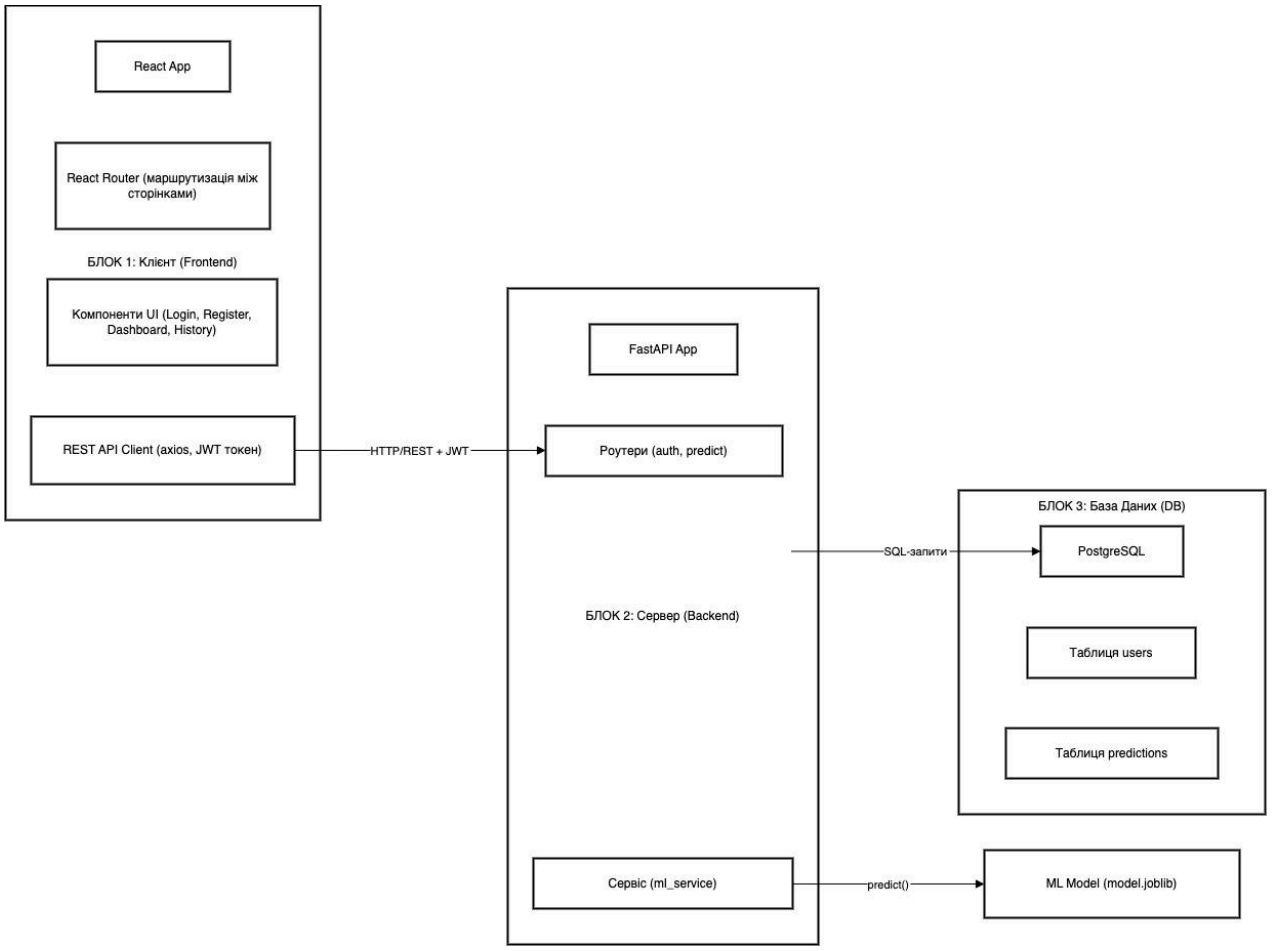

Рисунок 2.1 – Діаграма компонентів системи

Рівень представлення реалізовано як односторінковий застосунок (Single Page Application) на базі бібліотеки React. Клієнт взаємодіє з сервером виключно через HTTP-запити до REST API, що забезпечує чітке розмежування між логікою відображення та бізнес-логікою.

Рівень бізнес-логіки реалізовано на FastAPI і охоплює два функціональних модулі: маршрутизатор автентифікації та маршрутизатор класифікації. Саме на цьому рівні розміщено підсистема машинного навчання навчену модель разом із трансформером ознак. JWT Middleware відповідає за перевірку токена доступу на захищених маршрутах.

Рівень даних представлено реляційною СУБД PostgreSQL. Взаємодія з базою даних відбувається виключно через ORM SQLAlchemy 2.0, що унеможливлює SQL-ін'єкції та спрощує міграції схеми [24].

2.2 Вибір та обґрунтування технологічного стеку

Вибір технологічного стеку безпосередньо впливає на продуктивність розробки, якість кінцевого продукту та можливість подальшого розширення системи. Кожен компонент стеку обирався шляхом порівняльного аналізу альтернатив за визначеними критеріями [25].

Мова програмування для ML-компонента та серверної частини. Python є домінуючою мовою в екосистемі машинного навчання завдяки зрілості бібліотек scikit-learn, pandas, numpy та XGBoost. Порівняно з альтернативами Java (менша екосистема ML), R (орієнтований на статистику, не на продакшн-деплой) - Python забезпечує найкоротший шлях від дослідницького прототипу до розгорнутого API [26] . Серверний фреймворк

Таблиця 2.1 – Порівняння серверних фреймворків

Критерій	**FastAPI**	**Django REST**	**Flask**
Продуктивність (req/s)	Висока	Середня	Середня
Автодокументація	Вбудована	Стороння бібліотека	Стороння бібліотека
Валідація даних	Pydantic	Serializers	Marshmallow
Крива навчання	Низька	Висока	Низька
async/await	Висока	Обмежена	Обмежена

FastAPI обрано як оптимальний вибір для реалізації ML-орієнтованого REST API: нативна підтримка асинхронного виконання, автоматична генерація Swagger документації та вбудована валідація через Pydantic суттєво прискорюють розробку та зменшують кількість шаблонного коду [9].

Фронтенд бібліотека

Таблиця 2.2 – Порівняння фронтенд-бібліотек

Критерій	**React**	**Vue 3**	**Angular**
Розмір спільноти	Найбільша	Середня	Велика
Крива навчання	Середня	Низька	Велика
Гнучкість архітектури	Висока	Висока	Обмежена (opinionated)
Екосистема	Найбагатша	Достатня	Достатня
TypeScript підтримка	Добра	Відмінна	Вбудована
Доречність для SPA	Висока	Висока	Висока

React обрано завдяки найбільшій екосистемі пакетів, широкому поширенню у виробничих системах та наявності бібліотеки recharts для візуалізації результатів класифікації. СУБД

Таблиця 2.3 – Порівняння систем управління базами даних

Критерій	**PostgreSQL**	**MongoDB**	**SQLite**
Тип	Реляційна	Документоорієнтована	Реляційна
ACID транзакції	Повна підтримка	Часткова	Повна підтримка
JSON поля	Нативна	Нативна	Обмежена
Масштабованість	Висока	Висока	Низька

Продовження таблиці 2.3

Придатність для продакшн	Висока	Висока	Низька
Docker	Офіційний	Офіційний	Не потрібен

PostgreSQL обрано через повну підтримку ACID транзакцій, нативну підтримку JSON полів (необхідну для зберігання вхідних даних та результатів класифікації у таблиці predictions) та наявність офіційного Docker образу [27].

2.3 Проєктування бази даних

Схема бази даних визначає структуру зберігання даних та зв'язки між сутностями. Проєктування виконувалось у відповідності до принципів нормалізації: схема знаходиться у третій нормальній формі (3NF), що усуває транзитивні залежності та мінімізує надлишковість даних [28].

База даних містить дві таблиці: users та predictions. ER-діаграма наведена на рисунку 2.3.

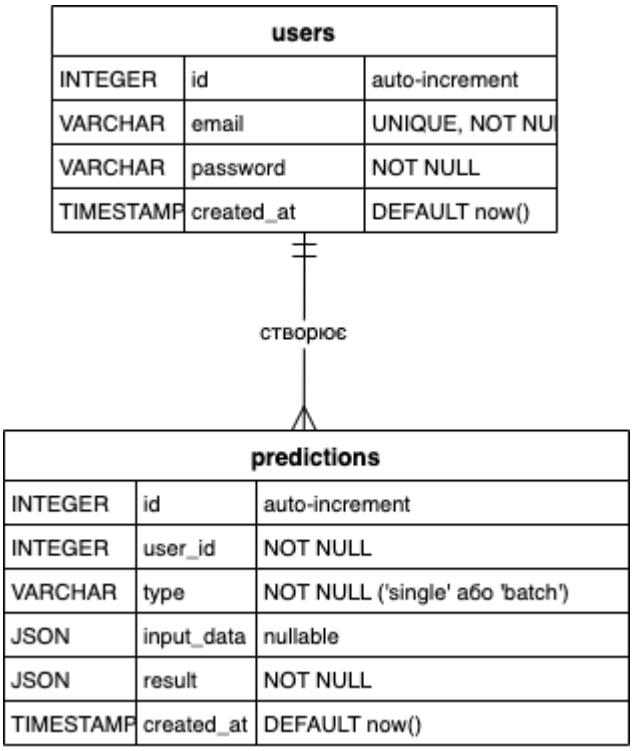

Рисунок 2.3 – ER-діаграма бази даних

Таблиця 2.4 - Структура таблиці users

Стовпець	**Тип**	**Обмеження**	**Опис**
id	INTEGER	PRIMARY KEY	Унікальний ідентифікатор
email	VARCHAR	UNIQUE, NOT NULL	Електронна адреса
password	VARCHAR	NOT NULL	bcrypt-хеш пароля
created_at	TIMESTAMP TZ	DEFAULT now()	Дата реєстрації

Таблиця 2.5 - Структура таблиці predictions

Стовпець	**Тип**	**Обмеження**	**Опис**
id	INTEGER	PRIMARY KEY	Унікальний ідентифікатор
user_id	INTEGER	FK → users.id	Власник запиту
type	VARCHAR	NOT NULL	«single» або «batch»
input_data	JSON	NOT NULL	Вхідні ознаки транзакцій
result	JSON	NOT NULL	Мітки класів та ймовірності
created_at	TIMESTAMPTZ	DEFAULT now()	Дата запиту

Зв'язок між таблицями «один-до-багатьох»: один користувач може мати довільну кількість записів про класифікацію. Поле type розрізняє одиночні та пакетні запити, що дозволяє коректно відображати їх в інтерфейсі історії. Поля input_data та result зберігаються у форматі JSON, що забезпечує гнучкість структури даних без фіксованої схеми у PostgreSQL [29].

На поле email таблиці users накладено унікальний індекс для забезпечення цілісності даних та прискорення операцій пошуку при автентифікації. На поле user_id таблиці predictions встановлено індекс для оптимізації вибірки історії запитів конкретного користувача.

2.4 Проєктування серверної частини

Серверна частина реалізована на основі FastAPI і організована за принципом розділення відповідальності: маршрутизатори визначають HTTP-інтерфейс, схеми Pydantic відповідають за валідацію, моделі SQLAlchemy

за взаємодію з базою даних, а підсистема машинного навчання за класифікацію [30].Структура модулів

Таблиця 2.6 – Ендпоінти REST API

Метод	**URL**	**Авторизація**	**Опис**
POST	/auth/register	Відкритий	Реєстрація нового користувача
POST	/auth/login	Відкритий	Автентифікація, отримання JWT
POST	/predict	JWT	Класифікація одиночної транзакції
POST	/predict/batch	JWT	Пакетна класифікація транзакцій
GET	/predict/history	JWT	Отримання історії запитів
GET	/health	Відкритий	Перевірка стану сервісу

Автентифікація реалізована на основі JSON Web Tokens (JWT). При успішному вході система генерує підписаний токен із полями sub (ідентифікатор користувача) та exp (термін дії). Захищені маршрути перевіряють токен через FastAPI Dependency Injection залежність get_current_user, яка декодує JWT та повертає об'єкт поточного користувача [31] . Паролі зберігаються виключно у вигляді bcrypt-хешів; відкритий текст пароля не зберігається і не передається після реєстрації.

Діаграма класів. На рисунку 2.4 наведено діаграму класів серверної частини, що відображає основні компоненти системи та їх зв'язки.

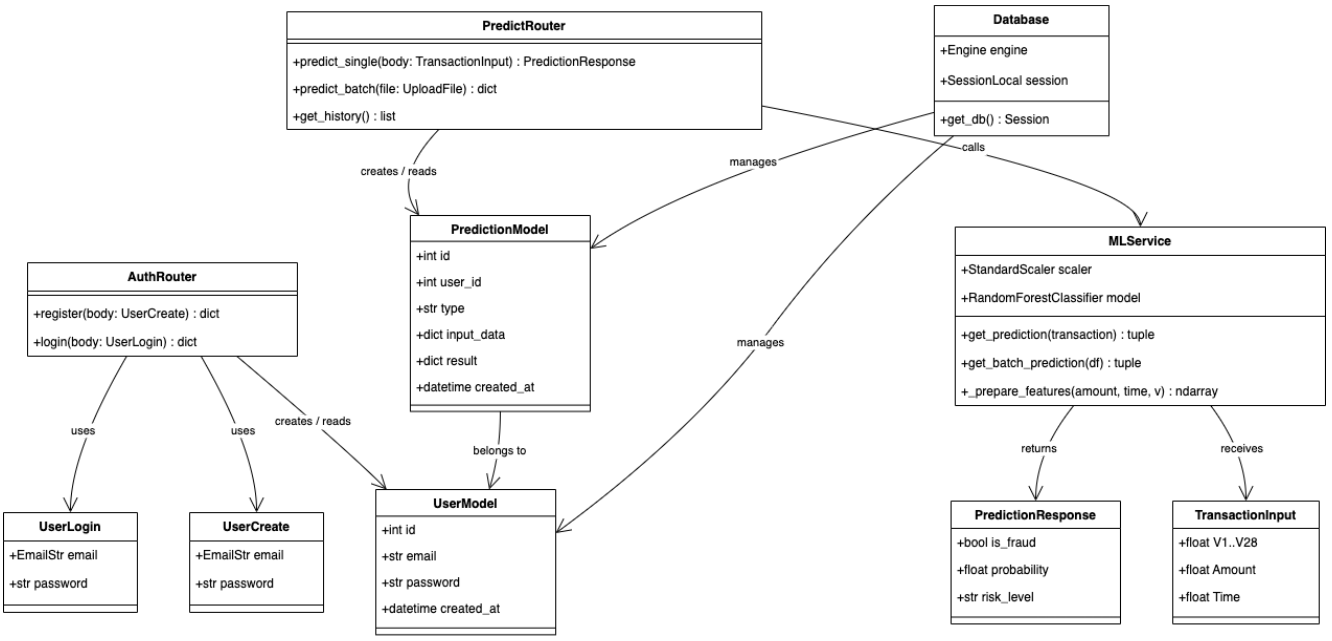

Рисунок 2.4 – Діаграма класів серверної частини

Схеми (UserCreate, UserLogin, TransactionInput, PredictionResult) виконують подвійну функцію: автоматичну валідацію вхідних даних із генерацією зрозумілих повідомлень про помилки та генерацію специфікації, яка відображається у вбудованому Swagger UI за адресою /docs.

2.5 Розробка модуля класифікації транзакцій

Модуль класифікації транзакцій є центральним компонентом системи. Він охоплює повний цикл від завантаження та дослідження датасету до серіалізації фінальної моделі для використання в API. Розробка виконувалась у середовищі Jupyter Notebook, що забезпечує відтворюваність експериментів [32].Розвідувальний аналіз даних

На першому етапі виконано розвідувальний аналіз даних (Exploratory Data Analysis, EDA). Датасет містить 284 807 транзакцій із 30 числовими ознаками та цільовою змінною Class. Виявлено суттєвий правосторонній скіс у розподілі ознаки Amount (рисунок 2.5).

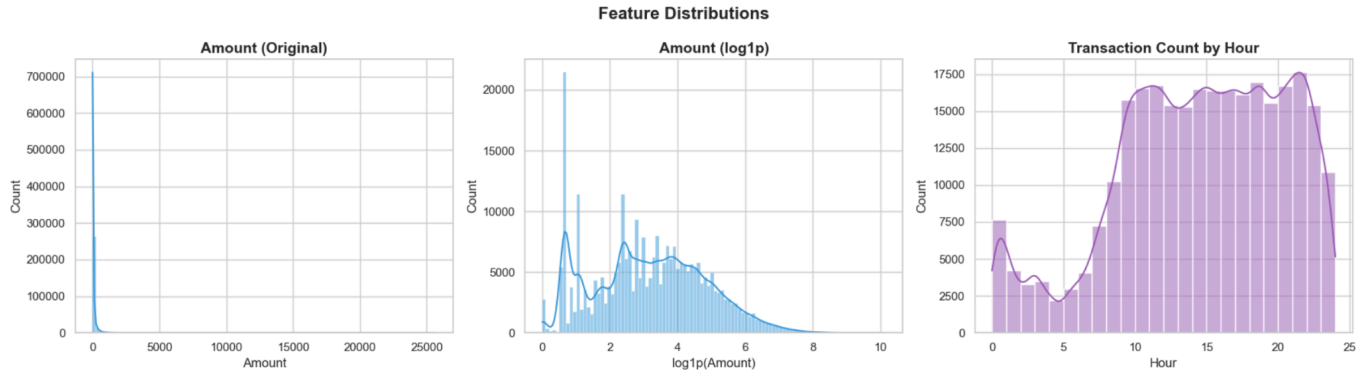

Рисунок 2.5 – Розподіл ознак Amount та Hour

Аналіз кореляцій між ознаками V1–V28 не виявив значущих попарних кореляцій, що є очікуваним результатом, оскільки ці ознаки отримано методом PCA і є лінійно незалежними за побудовою.

Конструювання ознак. На основі результатів EDA виконано конструювання двох додаткових ознак:

- Amount_log = log1p(Amount), логарифмічне перетворення суми транзакції для усунення правостороннього скісу розподілу та наближення його до нормального;
- Hour = (Time / 3600) % 24 перетворення часової мітки на годину доби для виявлення часових патернів шахрайської активності.

Після конструювання ознак вихідні стовпці Amount та Time видалено. Фінальний набір ознак містить 30 змінних: V1–V28, Amount_log, Hour.

Розбиття даних та балансування класів

Дані розбито на навчальну (75%) та тестову (25%) вибірки із застосуванням стратифікованого розбиття для збереження пропорції класів в обох підмножинах.

Нормалізацію ознак виконано за допомогою StandardScaler: параметри масштабування обчислені виключно на навчальній вибірці та застосовані до тестової це запобігає витоку статистик тестових даних у процес навчання.

Для боротьби з дисбалансом класів на навчальній вибірці застосовано SMOTE (Synthetic Minority Over-sampling Technique). Метод генерує синтетичні зразки міноритарного класу шляхом інтерполяції між наявними зразками у

просторі ознак. Після застосування SMOTE навчальна вибірка містить збалансовані класи у співвідношенні 1:1. Принципово важливо, що SMOTE застосовується виключно після розбиття на train/test застосування до всього датасету призвело б до data leakage та завищення оцінок якості.

Навчання та порівняння моделей. Навчання трьох алгоритмів: логістичної регресії, Random Forest та XGBoost, виконувалось в ідентичних умовах: однакові навчальна вибірка після SMOTE та тестова вибірка без SMOTE. Для ансамблевих моделей гіперпараметри підбирались за допомогою RandomizedSearchCV із крос-валідацією

Таблиця 2.7 – Порівняння алгоритмів машинного навчання

Алгоритм	**Переваги**	**Недоліки**	**Особливості**
Логістична регресія	Швидке навчання, інтерпретованість коефіцієнтів, стабільність	Лінійна межа рішення, низький F1 на незбалансованих даних	Використовуєтьс я як baseline
Random Forest	Стійкість до перенавчання, feature importance, підтримка class_weight	Повільніше навчання ніж LR, менш інтерпретована ніж LR	Ансамбль незалежних дерев
XGBoost	вбудована регуляризація L1/L2, стійкість до пропущених значень	Багато гіперпараметрів, схильний до перенавчання без регуляризації	Послідовне навчання з корекцією помилок

Таблиця 2.8 – Результати оцінювання моделей на тестовій вибірці

Модель	**ROC-AUC**	**PR-AUC**	**F1**	**Precision**	**Recall**
Logistic Regression	0.967	0.711	0.10	0.05	0.89
Random Forest	0.950	0.824	0.83	0.96	0.73
XGBoost	0.984	0.771	0.83	0.92	0.75

Аналіз результатів (таблиця 2.8) свідчить про принципову різницю між моделями в контексті задачі. Логістична регресія демонструє найвищий Recall (0.89), однак критично низький Precision (0.05) модель класифікує переважну більшість транзакцій як шахрайські, що є неприйнятним у виробничому середовищі через надмірну кількість хибних спрацювань. XGBoost показує найвищий ROC-AUC (0.984), однак PR-AUC (0.771) є нижчим, ніж у Random Forest.Та на рисунку 2.6 зображено матрицю помилок моделей.

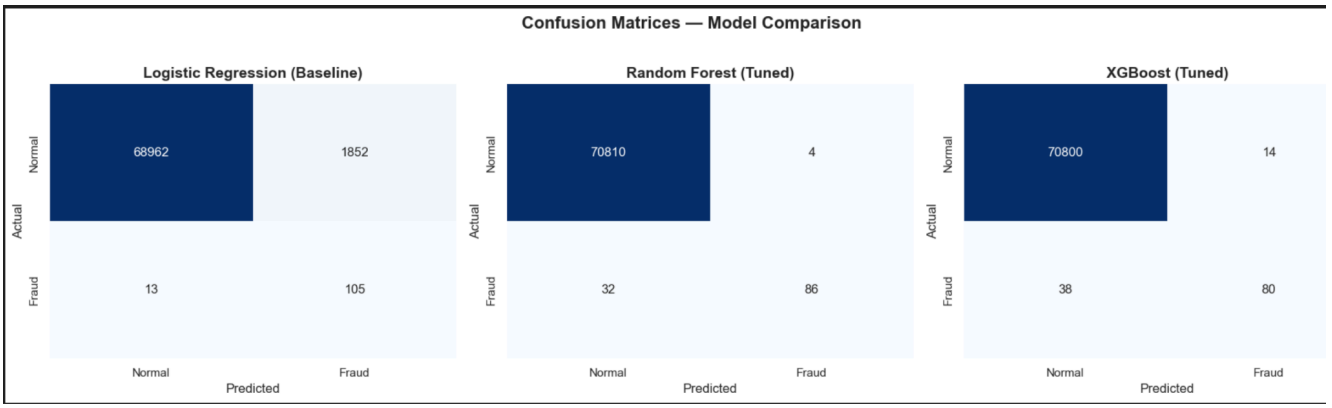

Рисунок 2.6 – Матриці помилок порівнюваних моделей

Random Forest обрано як фінальну модель на підставі найвищого PR-AUC (0.824) ключової метрики для незбалансованих даних та найкращого балансу між

Precision (0.96) і Recall (0.73), що мінімізує хибні спрацювання при прийнятному рівні виявлення шахрайства. На рисунку 2.7 наведено криву Precision-Recall для для порівнюваних моделей.

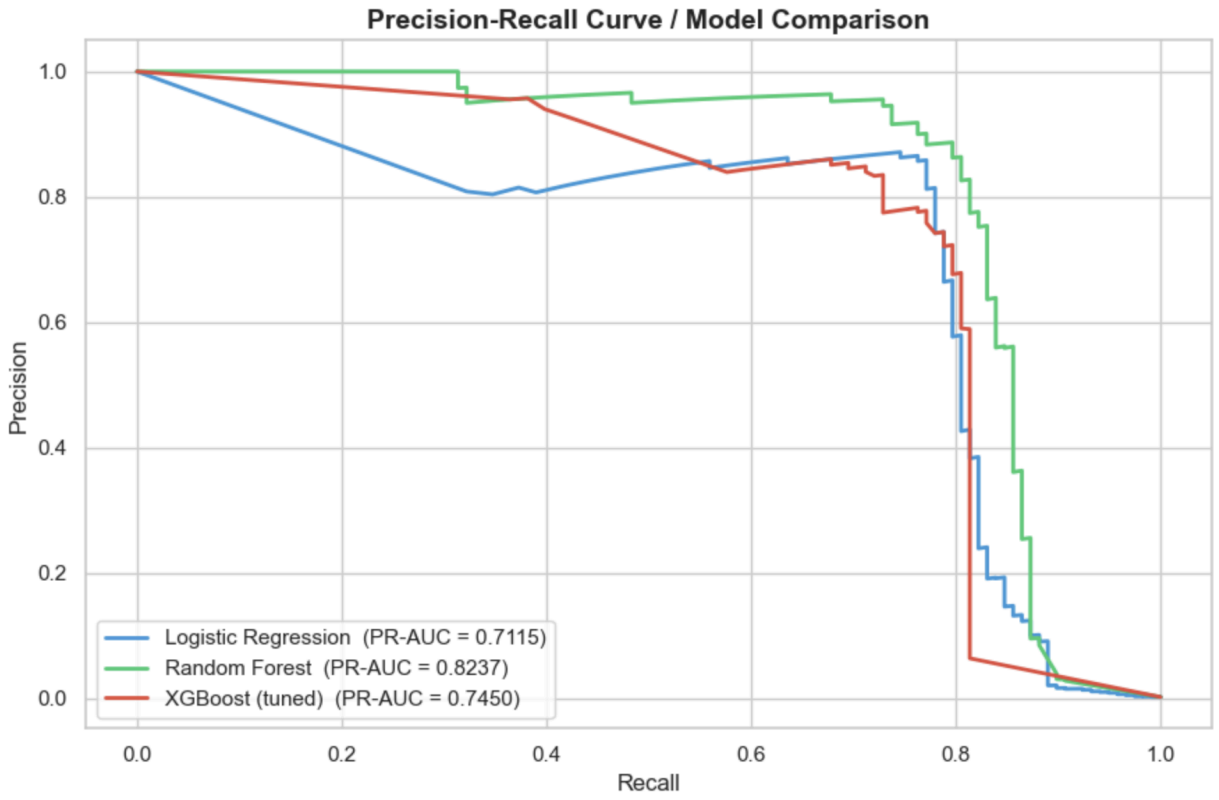

Рисунок 2.7 – Криві Precision-Recall для порівнюваних моделей

Фінальний ML модуль об'єкт, що містить StandardScaler та навчену модель RandomForestClassifier серіалізовано за допомогою бібліотеки joblib у файл .pkl. При запуску FastAPI застосунку файл завантажується одноразово в оперативну пам'ять та використовується для обслуговування всіх запитів на класифікацію, що мінімізує затримку відповіді.

2.6 Реалізація веб застосунку

Веб Інтерфейс реалізовано як односторінковий застосунок на React із використанням функціональних компонентів та хуків. Маршрутизація між сторінками реалізована бібліотекою react-router-dom; HTTP-запити до API через axios із попередньо налаштованим базовим URL та перехоплювачем (interceptor), що автоматично зчитує JWT-токен із localStorage та додає його до заголовка Authorization кожного захищеного запиту.

Застосунок містить три основні маршрути:

- LoginPage та RegisterPage форми автентифікації та реєстрації з клієнтською валідацією полів (рисунок 2.7).

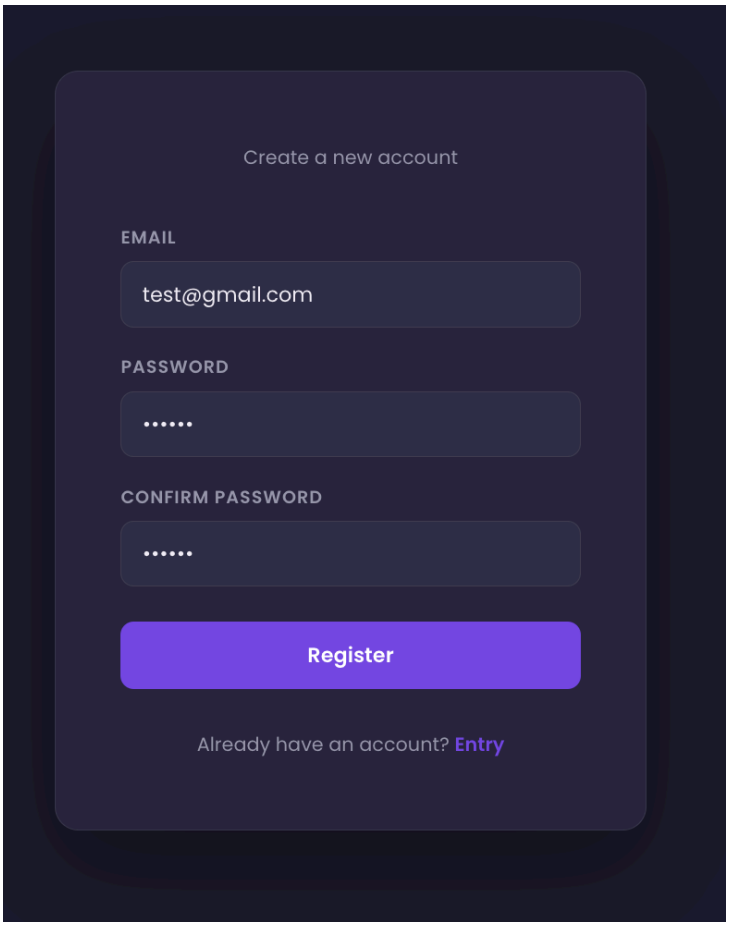

Рисунок 2.7 – Сторінка реєстрації

При успішному вході отриманий JWT-токен зберігається у localStorage, що забезпечує збереження сесії між перезавантаженнями сторінки (рисунок 2.8);

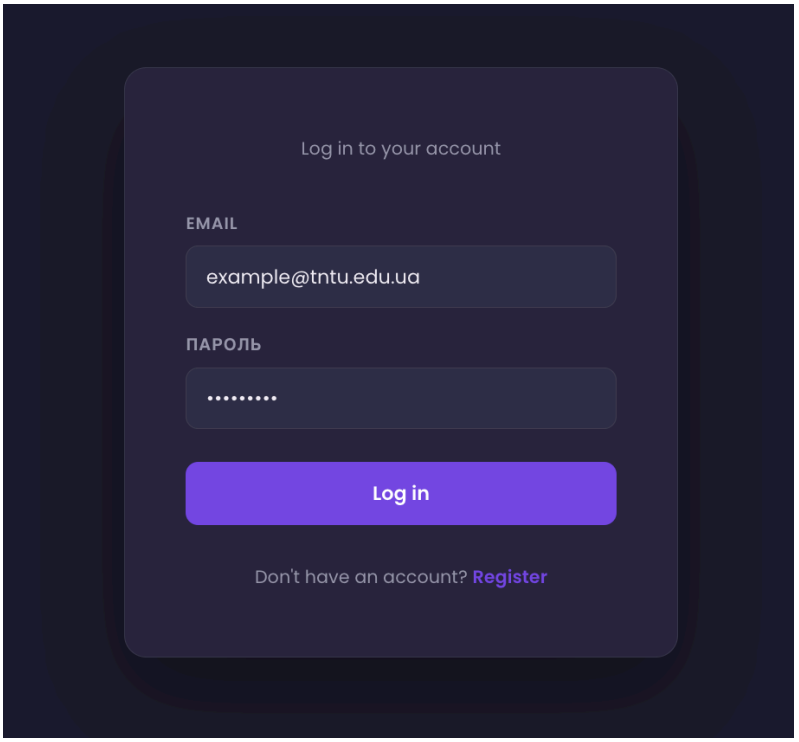

Рисунок 2.8 – Сторінка входу

- DashboardPage головна сторінка з двома вкладками : вкладка одиночної перевірки транзакції та вкладка пакетної перевірки CSV файлу. Обидва режими реалізовані як підкомпоненти в межах одного маршруту, між якими перемикається аналітик без переходу на іншу сторінку;
- HistoryPage сторінка хронологічної історії всіх запитів поточного користувача з модальним вікном деталізації результатів та вбудованою аналітикою.

Вкладка одиночної перевірки (рисунок 2.8) надає аналітику форму для введення числових ознак транзакції. Після надсилання запиту система відображає результат класифікації мітку FRAUD або LEGITIMATE разом із числовим значенням ймовірності шахрайства, що дозволяє аналітику оцінити ступінь впевненості моделі.

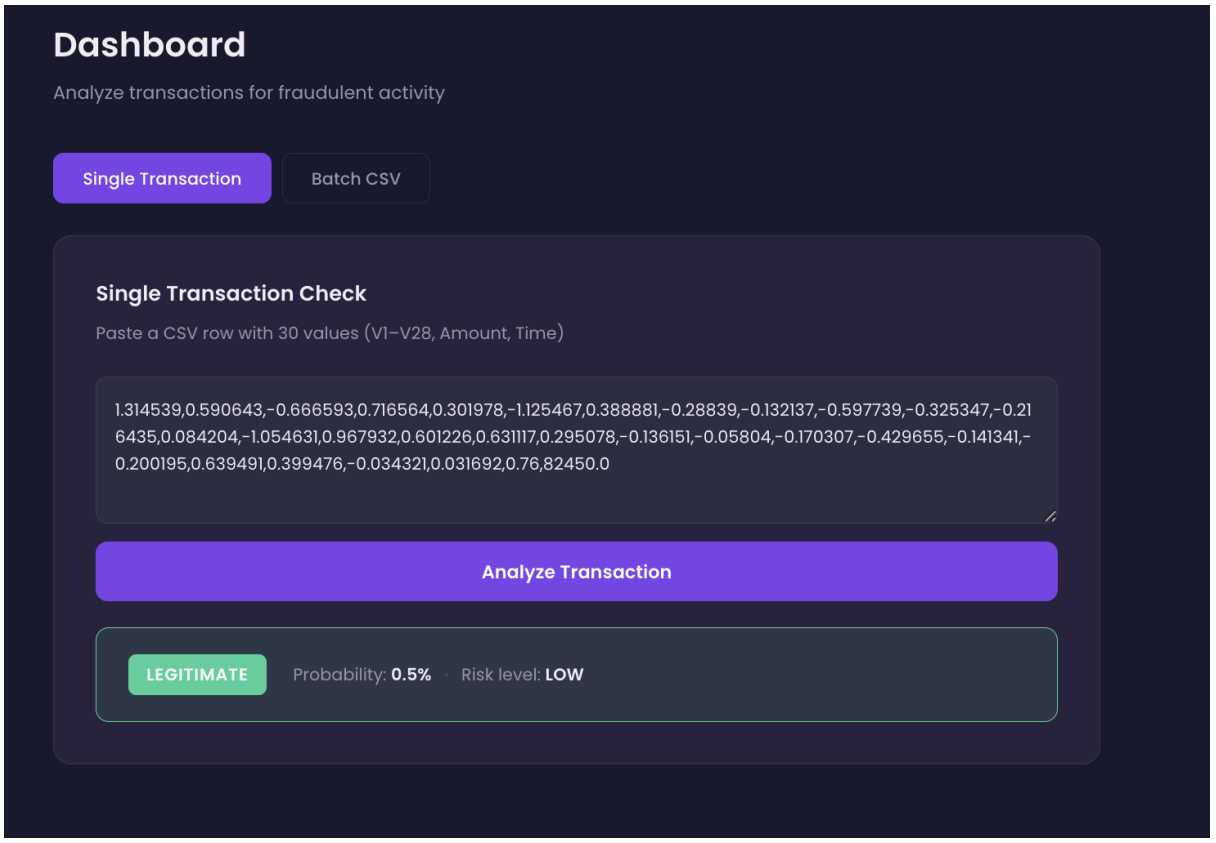

Рисунок 2.8 – Dashboard, вкладка одиночної перевірки з результатом класифікації

Вкладка пакетної перевірки (рисунок 2.9) дозволяє завантажити CSV-файл із масивом транзакцій. Після обробки система відображає зведену статистику: загальну кількість транзакцій (Total), кількість виявлених шахрайських (Fraudulent) та легітимних (Legitimate), а також відсоток шахрайства (Fraud Rate).

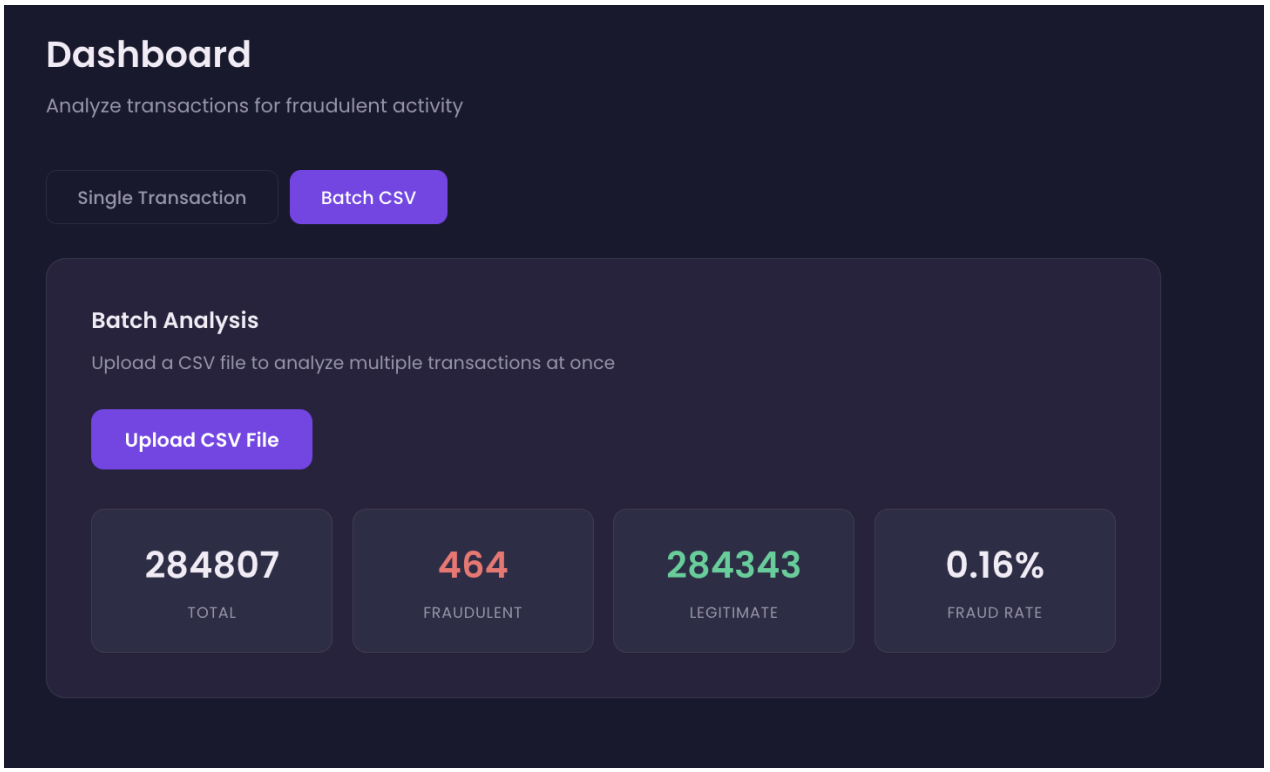

Рисунок 2.9 – Dashboard, вкладка пакетної перевірки з результатами batch-запиту

Сторінка History (рисунок 2.10) відображає хронологічний перелік усіх запитів поточного користувача у вигляді таблиці із зазначенням типу запиту (single/batch), дати та короткого підсумку.

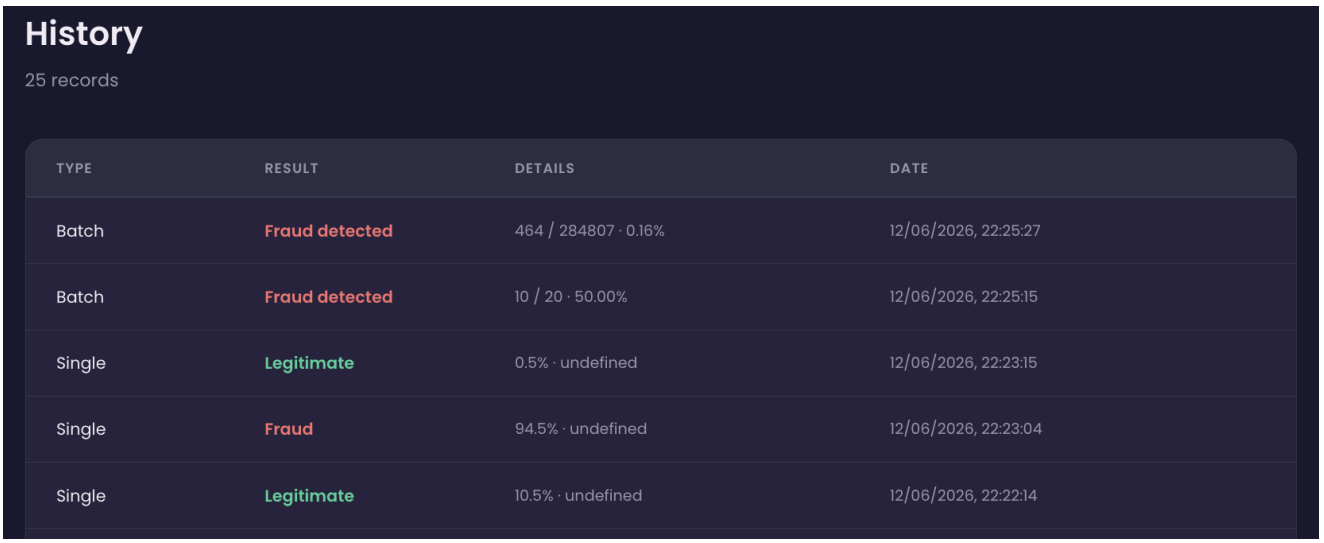

Рисунок 2.10 – Сторінка History з таблицею запитів

Клік на запис відкриває модальне вікно (рисунок 2.11) з детальними результатами та графіками аналітики, що надає аналітику повний контекст класифікаційного рішення.

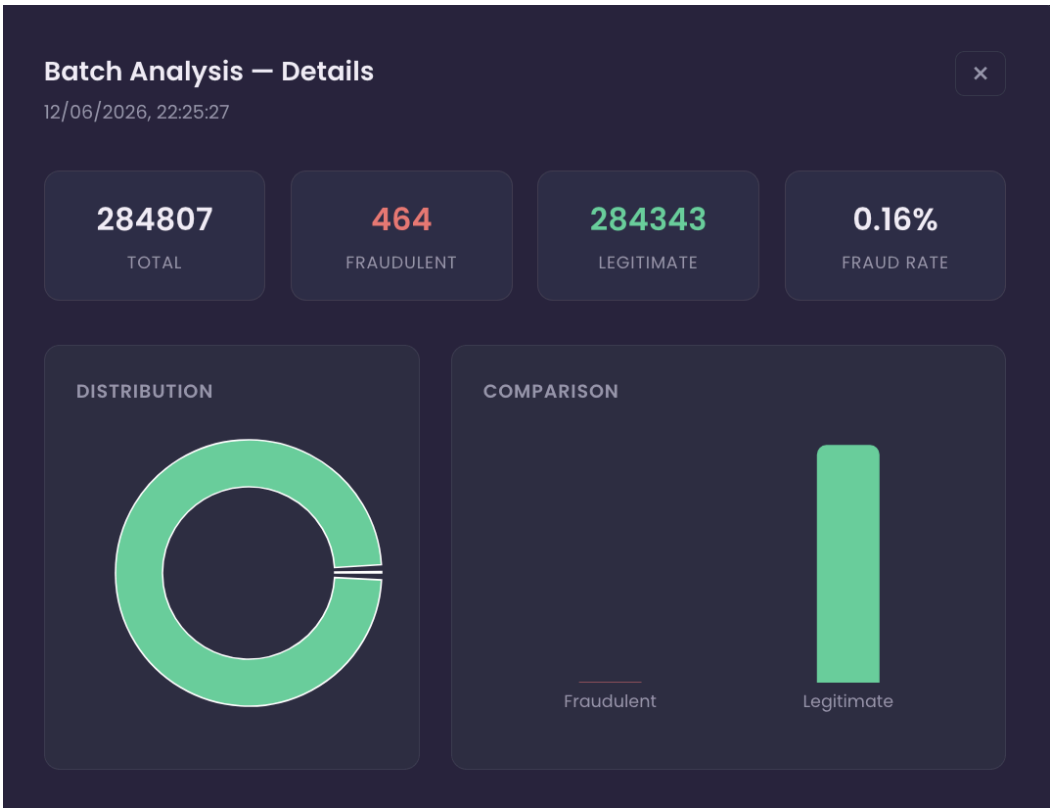

Рисунок 2.11 – Модальне вікно деталізації запису History

Захист маршрутів від неавтентифікованого доступу реалізовано через компонент PrivateRoute, який при кожному переході перевіряє наявність JWT-токена у localStorage та перенаправляє на сторінку входу у разі його відсутності або закінчення терміну дії х.

2.7 Висновки до розділу 2

У другому розділі виконано повний цикл проєктування та розробки програмної системи виявлення шахрайських транзакцій. Кожне прийняте рішення обґрунтовано з позиції функціональних вимог, визначених у першому розділі, та підкріплено порівняльним аналізом альтернатив.

Для системи обрано трирівневу архітектуру із контейнеризацією засобами Docker. Така організація забезпечує чітке розмежування відповідальності між рівнями та повну відтворюваність середовища виконання незалежно від платформи розгортання. Порівняльний аналіз технологічного стеку підтвердив перевагу обраних технологій.

Схему бази даних спроєктовано у третій нормальній формі: дві таблиці users та predictions пов'язані відношенням «один-до-багатьох». Тип JSON для полів input_data та result забезпечує гнучке зберігання результатів як одиночних, так і пакетних запитів без зміни схеми. Серверну частину реалізовано за принципом розділення відповідальності: шість ендпоінтів REST API, JWT-автентифікація та bcrypt-хешування паролів.

Модуль класифікації охоплює повний ML pipeline: конструювання ознак Amount_log та Hour, стратифіковане розбиття 75%/25%, застосування SMOTE виключно на навчальній вибірці та порівняльне навчання трьох алгоритмів. За результатами оцінювання фінальною моделлю обрано Random Forest із PR-AUC = 0.824 та Precision = 0.96 найкращий баланс між точністю виявлення та мінімізацією хибних спрацювань. Веб Інтерфейс реалізовано як React SPA із трьома маршрутами, захищеними компонентом PrivateRoute, та збереженням JWT-токена у localStorage.

Окремої уваги заслуговує підхід до безпеки системи: JWT-токени з обмеженим терміном дії, bcrypt-хешування паролів і перевірка автентифікації через механізм Dependency Injection FastAPI формують захищений периметр доступу до класифікаційних функцій. Ці рішення забезпечують не лише коректне функціонування системи у штатному режимі, а й стійкість до типових векторів атак на веб застосунки.

Прийняті архітектурні, технологічні та алгоритмічні рішення утворюють цілісний узгоджений стек, кожен компонент якого задокументований і обґрунтований. Отримані результати є достатньою основою для переходу до комплексного тестування системи та верифікації якості класифікаційної моделі, що розглядається у третьому розділі.

РОЗДІЛ 3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

Третій розділ присвячено верифікації коректності та надійності розробленої програмної системи. Описано план і методи тестування, наведено результати функціонального тестування REST API засобами Postman, тестування безпекових сценаріїв, а також порядок розгортання системи та рекомендації щодо її підтримки. Тестування охоплює як серверну частину, так і ML частину системи.

3.1 Види та план тестування

Тестування програмного забезпечення є обов'язковим етапом розробки, що дозволяє виявити дефекти до введення системи в експлуатацію та підтвердити відповідність реалізації функціональним вимогам. Для розроблюваної системи застосовано кілька взаємодоповнювальних видів тестування, вибір яких зумовлений специфікою архітектури: наявністю REST API, ML частини та механізму автентифікації.

Функціональне тестування спрямоване на перевірку відповідності поведінки системи специфікації. Для кожного ендпоінту REST API визначено позитивні сценарії (коректні вхідні дані, очікувана успішна відповідь) та негативні сценарії (некоректні дані, відсутня авторизація, дублікати). Тестування виконувалось за допомогою інструменту Postman графічного HTTP-клієнта, що дозволяє формувати запити довільного типу, передавати заголовки та тіло запиту.

Тестування безпеки охоплює перевірку механізмів захисту: неавторизованого доступу до захищених ендпоінтів, некоректного формату вхідних даних та спроби реєстрації з уже існуючою електронною адресою. Цей вид тестування є критично важливим для систем, що обробляють фінансові дані.

Верифікація ML моделі виконується окремо від функціонального тестування API. Вона полягає в аналізі метрик якості навченої моделі на тестовій вибірці матриці помилок, кривої Precision-Recall та порівняльної таблиці результатів трьох алгоритмів.

Таблиця 3.1 – План тестування системи

№	Об'єкт тестування	Вид тестування	Метод	Очікуваний результат
1	POST /auth/register	Функціональне	Postman	201, обліковий запис створено
2	POST /auth/login	Функціональне	Postman	200, JWT-токен отримано
3	POST /predict	Функціональне	Postman	200, мітка класу та ймовірність
4	POST /predict/batch	Функціональне	Postman	200, масив результатів
5	GET /predict/history	Функціональне	Postman	200, список запитів
6	GET /health	Функціональне	Postman	200, статус сервісу
7	POST /predict без токена	Безпека	Postman	401 Unauthorized
8	POST /register дублікат	Безпека	Postman	400 Bad Request
9	POST /predict некоректні дані	Безпека	Postman	422 Unprocessable Entity
10	POST /login невірний пароль	Безпека	Postman	401 Unauthorized
11	Random Forest на тестовій вибірці	Верифікація ML	Jupyter	PR-AUC ≥ 0.80, Precision ≥ 0.90

3.2 Тестування REST API засобом Postman

Функціональне тестування REST API виконувалось у середовищі Postman з попередньо налаштованим оточенням, що містить змінну token для автоматичного зберігання JWT токена після успішної автентифікації. Усі запити до захищених ендпоінтів містять заголовок Authorization: Bearer.

Позитивні сценарії

Тест 1: Реєстрація нового користувача. Надіслано POST-запит до /auth/register із тілом у форматі JSON, що містить електронну адресу та пароль. Сервер повернув відповідь із кодом 201 та підтвердженням успішної реєстрації. Результат тестування наведено на рисунку 3.1.

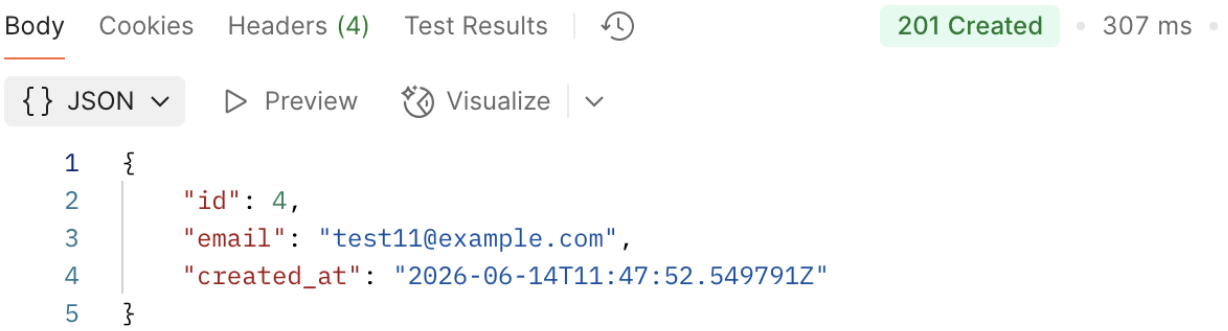

Рисунок 3.1 – Успішна реєстрація

Тест 2: Автентифікація та отримання токена. Надіслано POST-запит до /auth/login із тими самими обліковими даними. Сервер повернув відповідь із кодом 200 та JSON об'єктом, що містить поле access_token. Токен збережено у змінну оточення Postman для використання в наступних запитах. Результат наведено на рисунку 3.2.

Рисунок 3.2 – Автентифікація та отримання JWT-токена

Тест 3: Класифікація одиночної транзакції. Надіслано POST-запит до /predict із JSON-тілом, що містить 30 числових ознак транзакції (V1–V28, Amount, Time). Сервер повернув відповідь із кодом 200, міткою класу prediction: 0 (легітимна транзакція) та значенням ймовірності шахрайства probability: 0.105. Результат наведено на рисунку 3.3.

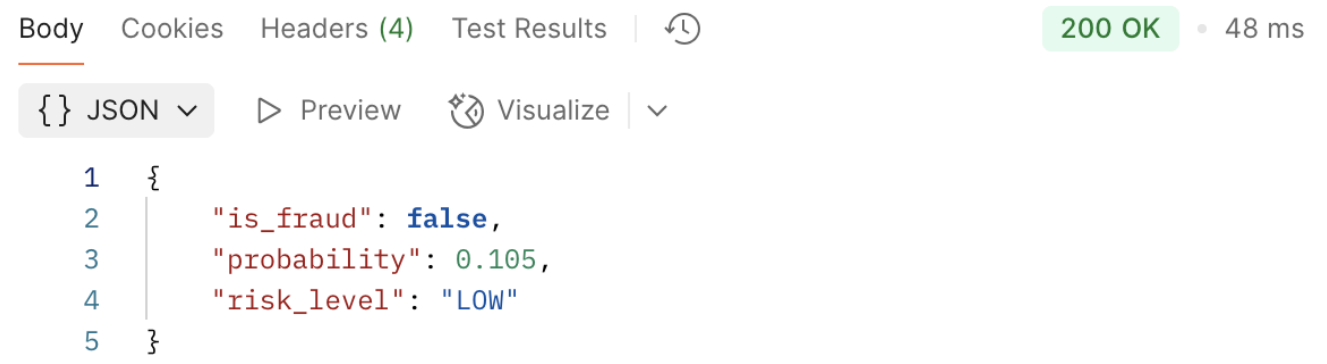

Рисунок 3.3 – Класифікація одиночної транзакції

Тест 4: Пакетна класифікація транзакцій. Надіслано POST-запит до /predict/batch у форматі multipart/form-data з CSV-файлом у полі file. Файл містив 20 транзакції. Сервер повернув відповідь із кодом 200 та масивом результатів класифікації для кожної транзакції. Результат наведено на рисунку 3.4.

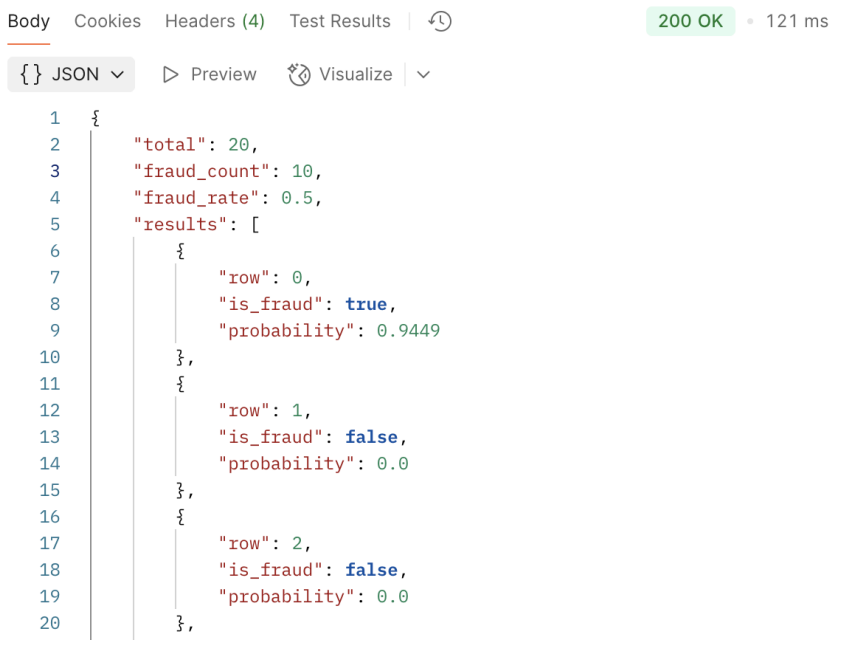

Рисунок 3.4 – Пакетна класифікація через CSV-файл

Тест 5 - Отримання історії запитів. Надіслано GET-запит до /predict/history із JWT-токеном у заголовку. Сервер повернув відповідь із кодом 200 та JSON-масивом, що містить записи про попередні запити поточного користувача з типом, датою та результатом кожного. Результат наведено на рисунку 3.5.

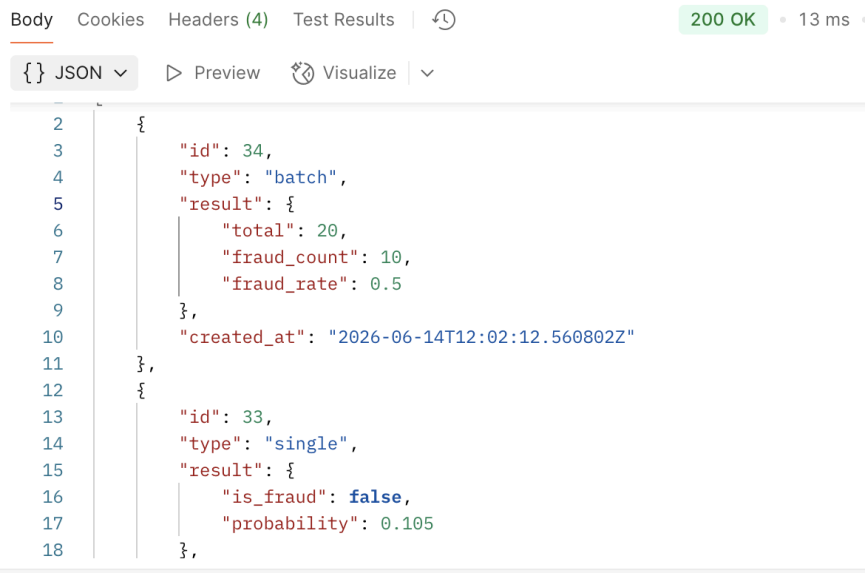

Рисунок 3.5 – Postman: отримання історії запитів

Негативні сценарії та тестування безпеки

Окрім перевірки коректної роботи у штатних умовах, виконано тестування поведінки системи при некоректних або зловмисних запитах. Це дозволяє переконатись у надійності механізмів захисту та валідації вхідних даних.

Тест 6: Доступ до захищеного ендпоінту без токена. Надіслано POST-запит до /predict без заголовка Authorization. Сервер повернув відповідь із кодом 401 Unauthorized та повідомленням "detail": "Not authenticated". Це підтверджує коректну роботу JWT Middleware: жоден захищений ендпоінт не доступний без дійсного токена (рисунок 3.6).

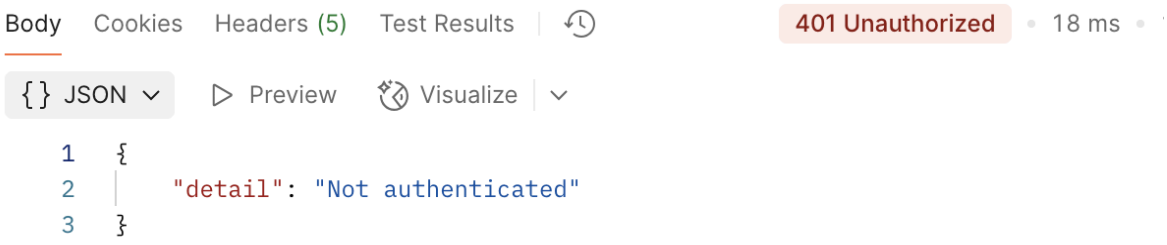

Рисунок 3.6 – Спроба доступу без токена

Тест 7: Реєстрація з дублікатом електронної адреси. Надіслано повторний POST-запит до /auth/register із вже зареєстрованою електронною адресою. Сервер повернув відповідь із кодом 400 Bad Request та повідомленням про те, що користувач із такою адресою вже існує. Це підтверджує коректність перевірки унікальності на рівні серверної логіки та унікального індексу бази даних (рисунок 3.7).

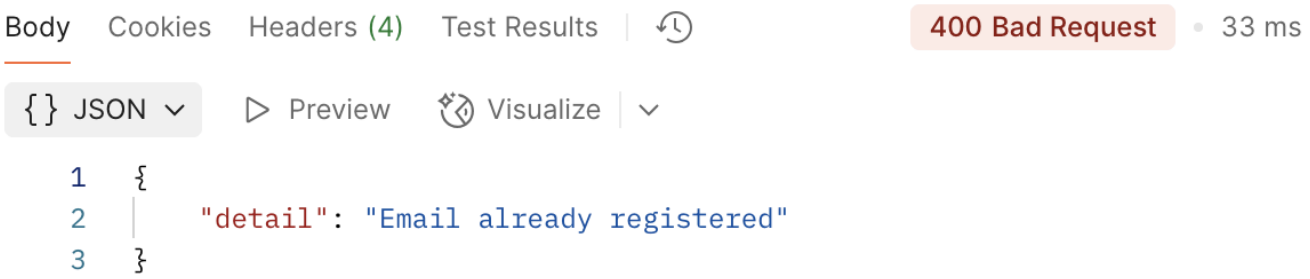

Рисунок 3.7 – Реєстрація з дублікатом email

Тест 8: Некоректний формат вхідних даних. Надіслано POST-запит до /predict із рядковим значенням замість числового у полі V1. Сервер повернув відповідь із кодом 422 Unprocessable Entity та детальним описом помилки валідації від Pydantic: поле V1, отримано тип string, очікується float. Це підтверджує коректну роботу схем валідації вхідних даних (рисунок 3.8).

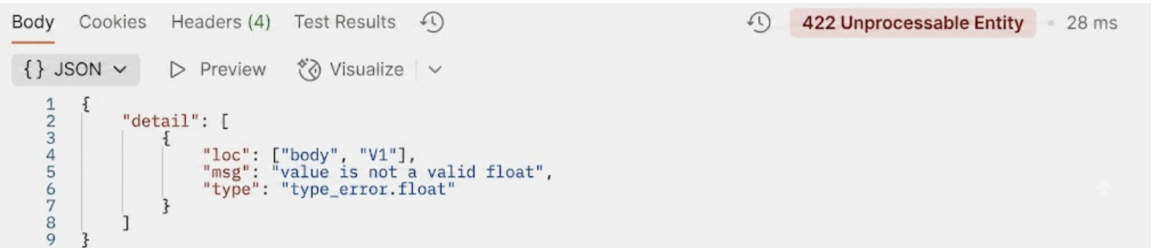

Рисунок 3.8 – Некоректний тип даних у запиті

Таблиця 3.2 – Зведені результати тестування REST API

№	Ендпоінт	Сценарій	Очікуваний код	Отриманий код	Статус
1	POST /auth/register	Нова реєстрація	201	201	Passed
2	POST /auth/login	Коректні дані	200	200	Passed

Продовження таблиці 3.2

3	POST /predict	Одиночна транзакція	200	200	Passed
4	POST /predict/batch	CSV-файл	200	200	Passed
5	GET /predict/history	З токеном	200	200	Passed
6	POST /predict	Без токена	401	401	Passed
7	POST /auth/register	Дублікат email	400	400	Passed
8	POST /predict	Рядок замість числа	422	422	Passed

За результатами тестування всі 8 перевірених сценаріїв завершились успішно: отримані коди відповідей повністю відповідають очікуваним. Система коректно обробляє як позитивні сценарії роботи, так і некоректні запити, повертаючи інформативні повідомлення про помилки.

3.3 Розгортання та системні вимоги

Розгортання програмної системи передбачає змішаний підхід: база даних PostgreSQL запускається у Docker-контейнері для забезпечення ізоляції та відтворюваності, тоді як backend і frontend запускаються безпосередньо на хост-машині [41]. Залежності серверної частини зафіксовані у файлі requirements.txt, залежності вебінтерфейсу — у package.json.

Таблиця 3.3 – Мінімальні системні вимоги для розгортання

Компонент	**Мінімальна вимога**	**Рекомендована**
Операційна система	Linux / macOS / Windows 10+	Ubuntu 22.04 LTS
CPU	2 ядра, 2.0 ГГц	4 ядра, 2.5 ГГц
RAM	4 ГБ	8 ГБ
Дисковий простір	5 ГБ	10 ГБ
Docker Engine	24.0+	Остання стабільна
Docker Compose	2.20+	Остання стабільна
Node.js	18.0+	20 LTS
Python	3.10+	3.11
Мережа	Порти 3000, 8000, 5432 вільні	—

Порядок розгортання системи складається з таких кроків:

1. Розпакування архіву проєкту. Розпакувати архів із вихідним кодом у цільову директорію та перейти до неї.
2. Налаштування змінних середовища. Скопіювати файл .env.example у .env та заповнити значення: SECRET_KEY (секретний ключ JWT), POSTGRES_PASSWORD та інші параметри підключення до бази даних.

3. Запуск бази даних: docker compose up, команда завантажить офіційний образ PostgreSQL та запустить контейнер бази даних у фоновому режимі. Перший запуск може тривати 1 - 2 хвилини.

4. Запуск серверної частини. FastAPI-застосунок запускається на хост машині та стає доступним за адресою http://localhost:8000. Прапор --reload забезпечує автоматичне перезавантаження при зміні коду.

5. Запуск веб інтерфейс: npm start React застосунок запускається на хост машині та стає доступним за адресою http://localhost:3000.

6. Перевірка стану сервісів. Після успішного запуску система доступна за адресами: Веб Інтерфейс: http://localhost:3000. REST API документація (Swagger UI): http://localhost:8000/docs. Health-check ендпоінт: http://localhost:8000/health.

7. Завершення роботи. Зупинити процеси backend (Ctrl+C) та frontend (Ctrl+C), після чого зупинити контейнер бази даних: docker compose down.

База даних PostgreSQL запускається у Docker-контейнері (образ postgres:15-alpine, порт 5432). Backend (FastAPI) та frontend (React) запускаються безпосередньо на хост-машині як окремі процеси у двох терміналах.

Для виробничого розгортання рекомендується розмістити систему за зворотним проксі-сервером Nginx із налаштованими SSL-сертифікатами (Let's Encrypt) та винести змінні середовища в систему управління секретами (Docker Secrets або HashiCorp Vault). Резервне копіювання бази даних слід виконувати щонайменше раз на добу засобами pg_dump із зберіганням резервних копій на окремому сховищі.

3.4 Висновки до розділу 3

У третьому розділі виконано комплексне тестування розробленої програмної системи виявлення шахрайських транзакцій та описано повний порядок її розгортання. Результати тестування підтверджують коректність реалізації всіх

функціональних вимог, визначених у першому розділі, та надійність механізмів захисту серверної частини.

На етапі планування тестування визначено три види перевірок: функціональне тестування REST API, тестування безпеки та розгортанні. Для кожного ендпоінту сформовано тест-кейси з чітко визначеними вхідними даними та очікуваними результатами, що забезпечило систематичність і відтворюваність процесу тестування. Загальний план охоплює 11 тест-кейсів, з яких 6 є позитивними функціональними сценаріями та 4 негативними сценаріями перевірки безпеки.

Функціональне тестування REST API засобами Postman охопило всі п'ять реалізованих ендпоінтів: реєстрацію, автентифікацію, одиночну та пакетну класифікацію транзакцій, а також отримання історії запитів. Усі 5 позитивних сценаріїв завершились успішно сервер повертав коректні HTTP-коди відповідей (201 для реєстрації, 200 для решти запитів) та JSON-об'єкти очікуваної структури. Зокрема, ендпоінт predict повернув мітку класу та ймовірність шахрайства для тестової транзакції, ендпоінт predict/batch коректно обробив CSV-файл із 20 транзакціями та повернув масив результатів, а predict/history надав хронологічний перелік збережених запитів поточного користувача.

Тестування безпеки охопило три критичні сценарії захисту. Спроба доступу до захищеного ендпоінту без JWT-токена коректно відхиляється із кодом 401 Unauthorized, що підтверджує працездатність механізму перевірки автентифікації через Dependency Injection FastAPI. Спроба повторної реєстрації з вже існуючою електронною адресою повертає код 400 Bad Request, що свідчить про коректну перевірку унікальності на рівні серверної бізнес-логіки та відповідного індексу бази даних. Передача рядкового значення замість числового у полі ознаки транзакції призводить до відповіді з кодом 422 Unprocessable Entity з детальним описом помилки валідації Pydantic. Усі 8 перевірених сценаріїв як позитивних, так і негативних завершились із результатом Passed.

Розгортання системи реалізовано за змішаним підходом: база даних PostgreSQL запускається у Docker контейнері, що гарантує ізольоване та

відтворюване середовище зберігання даних, тоді як серверна (FastAPI) та клієнтська (React) частини запускаються безпосередньо на хост-машині у двох окремих терміналах. Такий підхід спрощує процес розгортання в середовищі розробки та знижує вимоги до попереднього налаштування. Визначено мінімальні та рекомендовані системні вимоги до апаратного і програмного забезпечення, а також надано рекомендації щодо виробничого розгортання за зворотним проксі-сервером Nginx із SSL-сертифікатами та регулярного резервного копіювання бази даних засобами pg_dump.

Отримані результати тестування свідчать про те, що розроблена програмна система функціонує відповідно до специфікації, коректно обробляє як штатні, так і позаштатні ситуації, та готова до практичного використання аналітиками безпеки у банківській сфері.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Четвертий розділ присвячено питанням безпеки життєдіяльності та охорони праці в контексті розробки програмної системи виявлення шахрайських транзакцій. Розглянуто методологію моделювання та прогнозування небезпечних ситуацій, що виникають під час функціонування інформаційних систем, а також вимоги до організації праці розробника програмного забезпечення, що тривалий час працює з відеодисплейними терміналами.

4.1 Безпека життєдіяльності

Моделювання та прогнозування небезпечних ситуацій. Моделювання та прогнозування небезпечних ситуацій є одним із ключових інструментів превентивної безпеки в сучасних умовах розвитку інформаційних технологій і складних технічних систем. Під небезпечною ситуацією розуміють стан, за якого створюється реальна можливість виникнення нещасного випадку, аварії або надзвичайної події, що може призвести до матеріальних збитків, шкоди здоров'ю людей або порушення нормального функціонування об'єктів [6]. У контексті розробки та експлуатації програмної системи виявлення шахрайських транзакцій небезпечні ситуації можуть виникати як на рівні технічної інфраструктури (відмова серверів, витік бази даних, кібератаки), так і на рівні організації праці розробників та операторів системи.

Моделювання небезпек передбачає побудову формалізованих описів можливих сценаріїв розвитку негативних подій з метою кількісної або якісної оцінки їхньої ймовірності та наслідків. Виокремлюють кілька підходів до моделювання: детерміністичний, що базується на причинно-наслідкових зв'язках між подіями; імовірнісний, що враховує стохастичну природу більшості реальних загроз; та системний, що розглядає об'єкт захисту як складну взаємопов'язану структуру елементів [6]. Для інформаційних систем, подібних до розроблюваної, найбільш адекватним є системний імовірнісний підхід, оскільки збої можуть

виникати внаслідок складної взаємодії апаратних, програмних та людських чинників.

Прогнозування небезпечних ситуацій здійснюється на основі аналізу ретроспективних даних, статистики інцидентів та побудови математичних моделей розвитку негативних сценаріїв. Ефективним інструментом є метод «дерева відмов» (Fault Tree Analysis, FTA), що дозволяє систематично ідентифікувати комбінації подій, які призводять до критичного стану системи. Для програмної системи обробки фінансових транзакцій небезпечними сценаріями вищого рівня є: несанкціонований доступ до бази даних із персональними даними користувачів, відмова ML-компонента, що призводить до некоректної класифікації транзакцій, та недоступність сервісу внаслідок DDoS-атаки або апаратної відмови.

Стосовно розробленої системи, серед ідентифікованих небезпек на рівні технічної інфраструктури виокремлюються такі: витік JWT-секретного ключа, що компрометує механізм автентифікації всіх користувачів; SQL-ін'єкції через недостатню валідацію вхідних даних; пошкодження або втрата даних бази даних PostgreSQL внаслідок апаратної відмови; деградація точності ML-моделі внаслідок концептуального дрейфу даних (concept drift), тобто зміни статистичних властивостей транзакцій у часі.

Прогнозування розвитку небезпечних ситуацій в інформаційних системах вимагає запровадження системи безперервного моніторингу ключових показників: доступності сервісу, частоти помилкових відповідей API, аномальних піків навантаження та відхилень у розподілі вхідних даних. Для автоматизованого виявлення аномалій у роботі самої системи доцільно застосовувати статистичні методи контролю якості зокрема, контрольні карти Шюхарта, що фіксують вихід показників за межі допустимих відхилень і сигналізують про необхідність втручання [6].

Запобігання небезпечним ситуаціям на організаційному рівні передбачає розробку та впровадження плану реагування на інциденти (Incident Response Plan), що визначає послідовність дій команди при виявленні порушення безпеки або відмови системи. Такий план має включати процедури сповіщення

відповідальних осіб, ізоляції ураженого компонента, відновлення з резервної копії та проведення post-mortem аналізу для запобігання повторенню інциденту. Регулярне проведення навчань і симуляцій кризових ситуацій підвищує готовність команди до ефективного реагування в реальних умовах.

Важливим аспектом прогнозування небезпечних ситуацій є оцінка ризиків, пов'язаних із людським чинником. Статистика інцидентів у сфері інформаційної безпеки свідчить, що понад 80% порушень безпеки даних зумовлено помилками персоналу: використанням слабких паролів, переходом за фішинговими посиланнями, ненавмисним розкриттям конфіденційної інформації або неправомірним налаштуванням прав доступу [1]. У контексті розроблюваної системи це означає необхідність проведення регулярних навчань із кібергігієни для всіх членів команди, застосування принципу мінімальних привілеїв при налаштуванні доступу до виробничої бази даних та обов'язкового використання багатофакторної автентифікації для адміністративних облікових записів.

Кількісна оцінка ризику здійснюється як добуток ймовірності реалізації небезпечної події та величини потенційних збитків: $R = P \times C$, де R — рівень ризику, P — ймовірність настання події, C — величина збитків. Така матриця ризиків дозволяє пріоритизувати заходи захисту: спочатку усуваються або мінімізуються ризики з найвищим добутком $P \times C$, а ризики з низькою ймовірністю і незначними наслідками приймаються як залишкові. Для програмної системи обробки фінансових транзакцій найвищий пріоритет мають ризики витоку персональних даних користувачів та некоректної класифікації шахрайських транзакцій, що може призвести до фінансових збитків клієнтів банківської установи [6].

Таким чином, моделювання та прогнозування небезпечних ситуацій є невід'ємною складовою забезпечення надійності та безпеки програмних систем, що обробляють фінансові дані. Системний підхід до ідентифікації загроз, кількісна оцінка їхньої ймовірності через матрицю ризиків та завчасна розробка сценаріїв реагування суттєво знижують ризики виникнення критичних інцидентів і мінімізують їхні наслідки для користувачів та організації в цілому.

4.2 Основи охорони праці

Вимоги до режимів праці і відпочинку при роботі з ВДТ. Робота з відеодисплейними терміналами (ВДТ) є основним видом діяльності розробника програмного забезпечення і характеризується специфічним поєднанням фізичних та психоемоційних навантажень. Тривала безперервна робота з монітором комп'ютера спричиняє зорову втому (астенопію), статичне перевантаження м'язово-скелетної системи, психоемоційне напруження та порушення циркадних ритмів організму [8]. Нормативне регулювання режимів праці й відпочинку при роботі з ВДТ здійснюється відповідно до вимог ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» та НПАОП 0,00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями».

Категорія важкості робіт із ВДТ визначається характером виконуваних операцій, тривалістю зорового навантаження та рівнем психоемоційного напруження. Розробка програмного забезпечення належить до третьої категорії складності - роботи з творчим характером, введенням та редагуванням тексту й коду, читанням технічної документації, що вимагає тривалої концентрації уваги та напруженої зорової роботи [8]. Для цієї категорії встановлено найбільш жорсткі вимоги до режиму праці й відпочинку.

Відповідно до нормативних вимог, загальний час роботи з ВДТ не повинен перевищувати 50% тривалості робочої зміни. Протягом восьмигодинної зміни обов'язковими є регламентовані перерви: при 8-годинній зміні для третьої категорії робіт рекомендовано дві перерви тривалістю 15 хвилин через 2 години після початку роботи та через 2 години після обідньої перерви, а також 10-хвилинна перерва кожну годину безперервної роботи з дисплеєм. Сукупна тривалість регламентованих перерв не повинна бути меншою за 90 хвилин на зміну [8]. Перерви доцільно використовувати для виконання комплексу фізичних вправ, що знімають статичне навантаження з шийного та поперекового відділів хребта, а також вправ для очей.

Організація робочого місця розробника програмного забезпечення має відповідати ергономічним вимогам, що безпосередньо впливають на ефективність режиму праці й відпочинку. Монітор слід розташовувати на відстані 60–70 см від очей, його верхній край - на рівні очей або на 10–15 см нижче. Клавіатура розміщується на висоті 70–75 см від підлоги, що відповідає положенню зігнутих під прямим кутом ліктів. Крісло має забезпечувати можливість регулювання висоти та підтримку поперекового вигину хребта. Освітленість робочої поверхні повинна становити 300–500 лк при загальному освітленні та відсутності прямого потрапляння сонячного світла на екран монітора, що унеможливлює відблиски й зниження контрасту зображення [8].

Окремої уваги заслуговує організація психогігієни праці розробника. Монотонна відлагоджувальна робота (debugging), тривале очікування результатів компіляції або навчання ML-моделі, а також виконання дедлайнів є джерелами хронічного психоемоційного стресу, що знижує продуктивність та підвищує ризик помилок у коді. Для профілактики професійного вигорання рекомендується чергування різнохарактерних задач протягом робочого дня: поєднання написання коду, проведення код-рев'ю, вивчення документації та адміністративних завдань дозволяє знизити монотонність і підтримувати рівень когнітивної залученості [8].

Важливим елементом охорони здоров'я при роботі з ВДТ є дотримання вимог до мікроклімату приміщення. Температура повітря в приміщенні, де розташовані робочі місця з комп'ютерами, має підтримуватись у межах 22–24°C у теплий і 20–22°C у холодний період року при відносній вологості 40–60%. Рівень шуму не повинен перевищувати 50 дБА. Ці параметри суттєво впливають на тепловий та акустичний комфорт, а отже - на продуктивність та функціональний стан оператора ВДТ [8]. Недотримання мікрокліматичних норм призводить до підвищеної втомлюваності, зниження концентрації уваги та збільшення кількості помилок при виконанні складних творчих задач, до яких належить розробка програмного забезпечення.

Окремою складовою охорони праці при роботі з ВДТ є захист від електромагнітного випромінювання. Сучасні монітори на основі технологій IPS та

OLED генерують значно менший рівень електромагнітного поля порівняно з застарілими CRT-дисплеями, однак вимога щодо мінімальної відстані між монітором та оператором залишається актуальною. Рекомендується розміщувати бокові та задні панелі моніторів на відстані не менше 1,2 м від робочих місць сусідніх операторів, що мінімізує вплив бічного випромінювання. Використання захисних окулярів з фільтром синього спектру (blue light blocking glasses) є додатковим засобом зниження зорового навантаження при тривалій роботі з екраном [8].

Контроль за дотриманням режимів праці й відпочинку при роботі з ВДТ покладається на роботодавця та службу охорони праці організації. Обов'язковим є проведення інструктажів із питань безпечної організації робочого місця, попередніх і щорічних медичних оглядів працівників, що систематично використовують ВДТ, а також атестації робочих місць за умовами праці. Своєчасне виявлення ознак зорової втоми, порушень постави та психоемоційного перевантаження дозволяє вчасно скоригувати режим роботи та запобігти розвитку професійних захворювань опорно-рухового апарату, органів зору та нервової системи розробника програмного забезпечення.

ВИСНОВКИ

У кваліфікаційній роботі вирішено задачу розробки програмної системи для автоматизованого виявлення шахрайських транзакцій у банківських платіжних системах із застосуванням методів машинного навчання. Усі поставлені задачі виконано в повному обсязі, мету роботи досягнуто.

У ході аналізу предметної області встановлено, що традиційні системи захисту на основі детерміністичних правил принципово не здатні адаптуватися до нових шахрайських схем, оскільки розраховані виключно на заздалегідь відомі сценарії. Це обумовлює практичну необхідність переходу до інтелектуальних систем виявлення аномалій, які виявляють відхилення від типової поведінки без явного опису кожного можливого виду шахрайства. Огляд існуючих комерційних рішень виявив відсутність доступних відкритих інструментів із зручним веб інтерфейсом для команд аналітиків середнього рівня, що визначило практичну цінність розроблюваної системи.

Дослідження проблематики задачі дозволило встановити, що виявлення шахрайства є задачею бінарної класифікації на екстремально незбалансованих даних частка шахрайських транзакцій у використаному датасеті складає лише 0.17% від загальної кількості (492 зі 284 807 транзакцій). Обґрунтовано непридатність метрики Accuracy для таких задач та визначено PR-AUC як основну метрику оптимізації, що явно відображає якість роботи моделі. Доведено необхідність застосування SMOTE виключно на навчальній вибірці для уникнення data leakage та завищення оцінок якості.

Порівняльне дослідження трьох алгоритмів машинного навчання: логістичної регресії, Random Forest та XGBoost виконувалось в ідентичних умовах на однакових навчальній та тестовій вибірках. За сукупністю ключових метрик фінальною моделлю обрано Random Forest із підібраними гіперпараметрами: алгоритм досяг PR-AUC = 0.824 та Precision = 0.96, що означає лише 4 хибних спрацювання на 100 позначених транзакцій. Такий результат є прийнятним для

виробничого застосування у банківській сфері, де надмірне блокування легітимних операцій негативно впливає на клієнтський досвід.

Спроєктовано та реалізовано трирівневу архітектуру системи: React SPA як рівень представлення, FastAPI як рівень бізнес-логіки та PostgreSQL як рівень даних.

Серверна частина реалізує шість ендпоінтів REST API, захищених механізмом JWT-автентифікації через FastAPI Dependency Injection. ML pipeline включає конструювання двох додаткових ознак (Amount_log та Hour), стратифіковане розбиття вибірки, застосування SMOTE та RandomizedSearchCV для підбору гіперпараметрів. Серіалізований pipeline завантажується в пам'ять одноразово при старті застосунку, забезпечуючи мінімальну затримку відповіді. Веб Інтерфейс реалізовано як React SPA з трьома маршрутами: автентифікація, Dashboard із двома вкладками (одиночна та пакетна перевірка CSV-файлу) і History із модальним вікном деталізації результатів.

Функціональне тестування системи засобами Postman охопило 8 сценаріїв: 5 позитивних та 3 негативних. Усі 8 тест-кейсів завершились із результатом Passed отримані HTTP-коди відповідей повністю відповідають очікуваним. Розгортання системи реалізовано за таким підходом: PostgreSQL у Docker контейнері, FastAPI та React безпосередньо на хост-машині, що спрощує середовище розробки та знижує вимоги до попереднього налаштування.

Практичне значення роботи полягає у тому, що розроблена система надає аналітикам безпеки зручний та функціональний інструмент для оперативної перевірки підозрілих транзакцій як індивідуально, так і пакетно через CSV-файл із автоматичним збереженням повної історії запитів. Відтворюваність середовища та задокументований порядок розгортання забезпечують можливість практичного впровадження системи в інфраструктуру фінансової установи. Отримані результати підтверджують, що поєднання методів машинного навчання з повноцінною серверною та клієнтською частиною є ефективним підходом до вирішення задач автоматизованого фінансового моніторингу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Global Card Fraud Losses at $33 Billion [Електронний ресурс]. – Режим доступу:URL:https://www.globenewswire.com/news-release/2026/01/07/3214821/0/en/Global-Card-Fraud-Losses-at-33-Billion.html (дата звернення: 10.05.2026).

2. Learned Lessons in Credit Card Fraud Detection from a Practitioner Perspective [Електронний ресурс]. – Режим доступу: URL: https://www.sciencedirect.com/science/article/abs/pii/S095741741400089X (дата звернення: 10.05.2026).

3. Credit Card Fraud and Detection Techniques :A Review [Електронний ресурс]. – Режим Доступу:URL: https://knowledge.lancashire.ac.uk/id/eprint/25005/1/25005%20BBS_en_2009_2_Delamaire.pdf (дата звернення: 10.05.2026).

4. Credit Card Fraud Detection Using Machine Learning Techniques: A Comparative Analysis [Електронний ресурс]. – Режим доступу: URL: https://doi.org/10.1109/ICCNI.2017.8123782 (дата звернення: 10.05.2026).

5. Tabular Data: Deep Learning Is Not All You Need [Електронний ресурс]. – Режим доступу: URL: https://arxiv.org/abs/2106.03253 (дата звернення: 11.05.2026).

6. Желібо Є.П., Зацарний В.В. Безпека життєдіяльності: підручник. Київ: Каравела, 2023. 344 с.

7. ARIC Risk Hub: Adaptive Behavioural Analytics for Fraud Prevention [Електронний ресурс]. – Режим доступу: URL: https://www.featurespace.com/aric-risk-hub (дата звернення: 12.05.2026).

8. Жидецький В.Ц. Охорона праці користувачів комп’ютерів : підручник. Львів : Афіша, 2020. 176 с.

9. Credit Card Fraud Detection [Електронний ресурс]. – Режим доступу: URL: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.

10. React Документація [Електронний ресурс]. – Режим доступу: URL: https://react.dev/ (дата звернення: 03.05.2026).

11. A Systematic Analysis of Performance Measures for Classification Tasks

[Електронний ресурс]. – Режим доступу: URL: https://www.sciencedirect.com/science/article/abs/pii/S0306457309000259 (дата звернення: 11.05.2026).

12. The Relationship Between Precision-Recall and ROC Curves [Електронний ресурс]. – Режим доступу: URL: https://dl.acm.org/doi/10.1145/1143844.1143874 (дата звернення: 11.05.2026).

13. The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets [Електронний ресурс]. – Режим доступу: URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0118432 (дата звернення: 11.05.2026).

14. SMOTE: Synthetic Minority Over-sampling Technique [Електронний ресурс]. – Режим доступу: URL: https://www.jair.org/index.php/jair/article/view/10302 (дата звернення: 11.05.2026).

15. Leakage in Data Mining: Formulation, Detection, and Avoidance [Електронний ресурс]. – Режим доступу: URL: https://dl.acm.org/doi/10.1145/2382577.2382579 (дата звернення: 11.05.2026).

16. Applied Logistic Regression, 3rd Edition [Електронний ресурс]. – Режим доступу:URL:https://www.wiley.com/en-us/Applied+Logistic+Regression%2C+3rd+Edition-p-9780470582473 (дата звернення: 12.05.2026).

17. Random Forests [Електронний ресурс]. – Режим доступу: URL: https://link.springer.com/article/10.1023/A:1010933404324 (дата звернення: 11.05.2026).

18. XGBoost: A Scalable Tree Boosting System [Електронний ресурс]. – Режим доступу: URL: https://arxiv.org/abs/1603.02754 (дата звернення: 11.05.2026).

19. Random Search for Hyper-Parameter Optimization [Електронний ресурс]. – Режим доступу: URL: https://jmlr.org/papers/v13/bergstra12a.html (дата звернення: 11.05.2026).

20. Tabular Data: Deep Learning Is Not All You Need [Електронний ресурс]. – Режим доступу: URL: https://arxiv.org/abs/2106.03253 (дата звернення: 11.05.2026).

21. Writing Effective Use Cases [Електронний ресурс]. – Режим доступу: URL: https://www.oreilly.com/library/view/writing-effective-use/0201702258/ (дата звернення: 12.05.2026).

22. Software Architecture in Practice, 4th Edition [Електронний ресурс]. – Режим доступу:URL:https://www.oreilly.com/library/view/software-architecture-in/9780136886099/ (дата звернення: 12.05.2026).

23. Microservices Patterns: With Examples in Java [Електронний ресурс]. – Режим доступу: URL: https://www.manning.com/books/microservices-patterns (дата звернення: 12.05.2026).

24. Docker Compose Документація [Електронний ресурс]. – Режим доступу: URL: https://docs.docker.com/compose/ (дата звернення: 05.05.2026).

25. Patterns of Enterprise Application Architecture [Електронний ресурс]. – Режим доступу: URL: https://martinfowler.com/eaaCatalog/ (дата звернення: 12.05.2026).

26. Scikit-learn: Machine Learning in Python [Електронний ресурс]. – Режим доступу: URL: https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html (дата звернення: 12.05.2026).

27. PostgreSQL 15 Документація [Електронний ресурс]. – Режим доступу: URL: https://www.postgresql.org/docs/15/ (дата звернення: 06.05.2026).

28. An Introduction to Database Systems, 8th Edition [Електронний ресурс]. – Режим доступу: URL: https://archive.org/details/introductiontoda0000date (дата звернення: 12.05.2026).

29. PostgreSQL JSON Types Документація [Електронний ресурс]. – Режим доступу: URL: https://www.postgresql.org/docs/current/datatype-json.html (дата звернення: 06.05.2026).

30. Clean Architecture: A Craftsman's Guide to Software Structure and Design [Електронний ресурс]. – Режим доступу: URL: https://www.oreilly.com/library/view/clean-architecture-a/9780134494272/ (дата звернення: 12.05.2026).

31. RFC 7519: JSON Web Token (JWT) [Електронний ресурс]. – Режим доступу: URL: https://www.rfc-editor.org/rfc/rfc7519.html (дата звернення: 07.05.2026).

32.Jupyter Notebook Документація [Електронний ресурс]. – Режим доступу: URL: https://jupyter-notebook.readthedocs.io/ (дата звернення: 08.05.2026).

33.Joblib Документація Електронний ресурс]. – Режим доступу: URL: https://joblib.readthedocs.io/ (дата звернення: 08.05.2026).

34.React Router Документація [Електронний ресурс]. – Режим доступу: URL: https://reactrouter.com/ (дата звернення: 08.05.2026).

35.React Документація: Routing Patterns [Електронний ресурс]. – Режим доступу: URL: https://react.dev/learn/managing-state (дата звернення: 08.05.2026).

36.The Art of Software Testing, 3rd Edition [Електронний ресурс]. – Режим доступу: URL: https://www.wiley.com/en-us/The+Art+of+Software+Testing%2C+3rd+Edition-p-9781118031964 (дата звернення: 13.05.2026).

37.Postman Learning Center [Електронний ресурс]. – Режим доступу: URL: https://learning.postman.com/ (дата звернення: 09.05.2026).

38.OWASP API Security Project [Електронний ресурс]. – Режим доступу: URL: https://owasp.org/www-project-api-security/ (дата звернення: 09.05.2026).

39.Hidden Technical Debt in Machine Learning Systems [Електронний ресурс]. – Режим доступу: URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf (дата звернення: 13.05.2026).

40.Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation [Електронний ресурс]. – Режим доступу: URL: https://arxiv.org/abs/2010.16061 (дата звернення: 13.05.2026).

41.Compose in Production [Електронний ресурс]. – Режим доступу: URL: https://docs.docker.com/compose/how-tos/production/ (дата звернення: 10.05.2026).

ДОДАТКИ

ДОДАТОК А

Тези конференції

Міністерство освіти і науки України
Тернопільський національний технічний університет
імені Івана Пулюя
Маріборський університет (Словенія)
Технічний університет в Кошице (Словаччина)
Каунаський технологічний університет (Литва)
Львівський національний університет
імені Івана Франка
Гірничо-металургійна академія ім. Станіслава Сташиця (Польща)
Луцький національний технічний університет
Чернівецький національний університет
імені Юрія Федьковича
Вроцлавський економічний університет (Польща)
Університет технологій та економіки
імені Хелени Ходковської (Польща)
Донбаська державна машинобудівна академія

Студентське наукове товариство

IX МІЖНАРОДНА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

24-25 квітня 2026 р.

(збірник тез конференції)

Тернопіль 2026

УДК 00000
Великанич М.І. - ст. гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ ДЛЯ ВИЯВЛЕННЯ АНОМАЛЬНИХ ТРАНЗАКЦІЙ У ЗАДАЧАХ БАНКІВСЬКОГО МОНІТОРИНГУ З ВИКОРИСТАННЯМ МАШИННОГО НАВЧАННЯ

Науковий керівник: к.т.н., доц. Стоянов Ю.М.

Velykanych M.I.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF A SOFTWARE SYSTEM FOR DETECTING ANOMALOUS TRANSACTIONS IN BANKING MONITORING TASKS USING MACHINE LEARNING

Supervisor: Ph.D., Assoc. Stoyanov Y.M.

Ключові слова: машинне навчання, банківський моніторинг, аномалії.
Keywords: machine learning, banking monitoring, anomalies.

В умовах глобальної цифровізації фінансових послуг та постійного збільшення кількості безготівкових переказів, швидке виявлення шахрайських дій стає першочерговим завданням для банків. Класичні системи моніторингу, що працюють за наперед заданими правилами, часто дають збої, блокуючи легітимні операції, повільно реагують на нові види кіберзлочинів.

Основною складністю при побудові таких моделей є природний дисбаланс даних частка аномалій зазвичай не перевищує 1% від усього потоку платежів. Крім того, для практичного використання ML-алгоритмів необхідна зручна програмна оболонка, яка дозволить операторам служби безпеки швидко аналізувати результати роботи моделі.

Метою даної роботи є проектування та реалізація повноцінної програмної системи, що поєднує інтелектуальний аналітичний бекенд для пошуку аномалій та інтерактивний клієнтський додаток для моніторингу в реальному часі.

Архітектура рішення базується на клієнт-серверній моделі.Для детекції відхилень та класифікації транзакцій застосовуються моделі та алгоритми ізоляції, а проблема дисбалансу вирішується за допомогою технік ресемплінгу. Клієнтська частина будується на базі сучасних веб-фреймворків і містить інтерактивний дашборд, інструменти для завантаження історичних вибірок та стрічку інцидентів. Важливою функцією є наявність модулю пояснення рішень моделі, що деталізує причини блокування кожної підозрілої операції. Запропонована система забезпечує вищу точність детекції завдяки застосуванню ML-алгоритмів і одночасно пропонує спеціалістам візуально зрозумілий інструмент для аудиту фінансових даних. Таке комплексне рішення повністю задовольняє сучасні потреби індустрії у сфері фінансової безпеки.

ДОДАТОК Б

Лістинг колу навчання та оцінювання моделей класифікації шахрайських транзакцій

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
import os
import warnings
import xgboost as xgb
warnings.filterwarnings('ignore')

from    sklearn.model_selection               import    train_test_split,
RandomizedSearchCV
from sklearn.preprocessing      import StandardScaler
from sklearn.linear_model       import LogisticRegression
from sklearn.ensemble           import RandomForestClassifier
from sklearn.metrics            import (
    classification_report, confusion_matrix,
    roc_auc_score, precision_recall_curve, auc,
    precision_score, recall_score, f1_score
)

sns.set_theme(style='whitegrid', palette='muted')
RANDOM_STATE = 42


# 1. Завантаження даних та конструювання ознак

df = pd.read_csv('../data/raw/creditcard.csv')
df = df.drop_duplicates()

# Amount_log усуває правосторонній скіс розподілу суми транзакції
```

```
df['Amount_log'] = np.log1p(df['Amount'])
# Hour виявляє часові патерни шахрайської активності
df['Hour'] = (df['Time'] / 3600) % 24

X = df.drop(columns=['Class', 'Time', 'Amount'])
y = df['Class']

print(f"Ознак: {X.shape[1]}  |  Транзакцій: {X.shape[0]:,}")
print(f"Шахрайських: {y.sum()}  |  Легітимних: {(y == 0).sum():,}")


# 2. Розбиття на навчальну/тестову вибірки та нормалізація

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=RANDOM_STATE, stratify=y
)

# fit ТІЛЬКИ на навчальній вибірці — запобігання data leakage
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)


# Модель 1 — Logistic Regression (baseline)

lr = LogisticRegression(
    max_iter=1000,
    class_weight='balanced',
    random_state=RANDOM_STATE
)
lr.fit(X_train_scaled, y_train)

y_pred_lr  = lr.predict(X_test_scaled)
y_proba_lr = lr.predict_proba(X_test_scaled)[:, 1]


#  Модель 2 — Random Forest

param_dist = {
```

```
    'n_estimators': [100, 200],
    'max_depth':    [10, 20],
    'max_features': ['sqrt', 'log2'],
}

search = RandomizedSearchCV(
    RandomForestClassifier(
        class_weight='balanced',
        random_state=RANDOM_STATE,
        n_jobs=-1
    ),
    param_distributions=param_dist,
    n_iter=6,
    scoring='average_precision',   # оптимізація саме PR-AUC
    cv=3,
    random_state=RANDOM_STATE,
    n_jobs=-1
)
search.fit(X_train_scaled, y_train)

rf_tuned         = search.best_estimator_
y_pred_rf_tuned  = rf_tuned.predict(X_test_scaled)
y_proba_rf_tuned = rf_tuned.predict_proba(X_test_scaled)[:, 1]

print(f"Найкращі параметри RF: {search.best_params_}")


#  Модель 3 — XGBoost

X_tr, X_val, y_tr, y_val = train_test_split(
    X_train_scaled, y_train,
    test_size=0.2, random_state=RANDOM_STATE, stratify=y_train
)

dtrain = xgb.DMatrix(X_tr, label=y_tr)
dvalid = xgb.DMatrix(X_val, label=y_val)
dtest  = xgb.DMatrix(X_test_scaled, label=y_test)

xgb_params = {
```

```
    'objective':        'binary:logistic',
    'eta':              0.039,
    'max_depth':        2,
    'subsample':        0.8,
    'colsample_bytree': 0.9,
    'eval_metric':      'auc',
    'seed':             RANDOM_STATE,
}

xgb_tuned = xgb.train(
    xgb_params, dtrain,
    num_boost_round=500,
    evals=[(dtrain, 'train'), (dvalid, 'valid')],
    early_stopping_rounds=50,
    maximize=True,
    verbose_eval=False
)

y_proba_xgb_tuned = xgb_tuned.predict(dtest)
y_pred_xgb_tuned  = (y_proba_xgb_tuned >= 0.5).astype(int)


# Порівняння моделей за метриками

def get_metrics(y_true, y_pred, y_proba, name):
    p, r, _ = precision_recall_curve(y_true, y_proba)
    return {
        'Model':     name,
        'ROC-AUC':   round(roc_auc_score(y_true, y_proba), 4),
        'PR-AUC':    round(auc(r, p), 4),
                    'Precision':  round(precision_score(y_true,  y_pred,
zero_division=0), 4),
        'Recall':    round(recall_score(y_true, y_pred, zero_division=0),
4),
        'F1':        round(f1_score(y_true, y_pred, zero_division=0), 4),
    }

results = pd.DataFrame([
```

```
    get_metrics(y_test, y_pred_lr,         y_proba_lr,         'Logistic
Regression'),
      get_metrics(y_test, y_pred_rf_tuned,   y_proba_rf_tuned,   'Random
Forest (tuned)'),
      get_metrics(y_test, y_pred_xgb_tuned, y_proba_xgb_tuned, 'XGBoost
(tuned)'),
])
print(results)

 7. Збереження фінальної моделі


os.makedirs('../saved_models', exist_ok=True)

joblib.dump(rf_tuned, '../saved_models/best_fraud_model.joblib')
joblib.dump(scaler,   '../saved_models/scaler.joblib')

print('Модель збережено: best_fraud_model.joblib')
print('Scaler збережено: scaler.joblib')
```

Лістинг коду серверної частини:

ml_service.py:

```
# Завантаження моделі та виконання класифікації

import numpy as np
import pandas as pd
import joblib

model  = joblib.load('saved_models/best_fraud_model.joblib')
scaler = joblib.load('saved_models/scaler.joblib')

FEATURE_ORDER = [
    'V1','V2','V3','V4','V5','V6','V7','V8','V9','V10',
    'V11','V12','V13','V14','V15','V16','V17','V18','V19','V20',
    'V21','V22','V23','V24','V25','V26','V27','V28',
    'Amount_log', 'Hour'
]
```

```
THRESHOLD = 0.5


def _prepare_features(amount: float, time: float, v_features: dict) ->
np.ndarray:
    """Виконує те саме конструювання ознак, що і при навчанні."""
    amount_log = np.log1p(amount)
    hour       = (time / 3600) % 24
    row = {**v_features, 'Amount_log': amount_log, 'Hour': hour}
    df  = pd.DataFrame([row])[FEATURE_ORDER]
    return scaler.transform(df)


def get_prediction(transaction) -> tuple[float, int]:
    """Класифікація однієї транзакції."""
    v_features = {
        f'V{i}': getattr(transaction, f'V{i}') for i in range(1, 29)
    }
    X = _prepare_features(transaction.Amount, transaction.Time,
v_features)
    probability = float(model.predict_proba(X)[0][1])
    label = int(probability >= THRESHOLD)
    return probability, label
```

app/auth.py -Генерація та валідація JWT-токена

```
from datetime import datetime, timedelta
from jose import JWTError, jwt
import bcrypt
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer

SECRET_KEY = "..."
ALGORITHM  = "HS256"

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/auth/login")
```

```
def hash_password(password: str) -> str:
    return bcrypt.hashpw(
        password.encode("utf-8"), bcrypt.gensalt()
    ).decode("utf-8")


def verify_password(plain: str, hashed: str) -> bool:
    return bcrypt.checkpw(plain.encode("utf-8"), hashed.encode("utf-8"))


def create_access_token(data: dict) -> str:
    payload = data.copy()
    payload["exp"] = datetime.utcnow() + timedelta(minutes=60)
    return jwt.encode(payload, SECRET_KEY, algorithm=ALGORITHM)


def get_current_user(token: str = Depends(oauth2_scheme),
db=Depends(get_db)):
    """Dependency для захищених ендпоінтів — перевіряє JWT-токен."""
    credentials_error = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Invalid credentials"
    )
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        user_id = payload.get("sub")
        if user_id is None:
            raise credentials_error
    except JWTError:
        raise credentials_error

    user = db.query(models.User).filter(models.User.id ==
int(user_id)).first()
    if user is None:
        raise credentials_error
    return user
```

app/models.py - моделі таблиць users та predictions

```
from sqlalchemy import Column, Integer, String, DateTime, ForeignKey,
JSON
from sqlalchemy.orm import relationship
from sqlalchemy.sql import func


class User(Base):
    __tablename__ = "users"

    id         = Column(Integer, primary_key=True, index=True)
    email      = Column(String, unique=True, index=True, nullable=False)
    password   = Column(String, nullable=False)
    created_at = Column(DateTime(timezone=True),
server_default=func.now())

    predictions = relationship("Prediction", back_populates="user")


class Prediction(Base):
    __tablename__ = "predictions"

    id         = Column(Integer, primary_key=True, index=True)
    user_id    = Column(Integer, ForeignKey("users.id"), nullable=False)
    type       = Column(String, nullable=False)       # 'single' або
'batch'
    input_data = Column(JSON, nullable=True)
    result     = Column(JSON, nullable=False)
    created_at = Column(DateTime(timezone=True),
server_default=func.now())

    user = relationship("User", back_populates="predictions")
```

predict.py Ендпоінт класифікації одиночної транзакції

```
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session

router = APIRouter(prefix="/predict", tags=["predict"])


def risk_level(probability: float) -> str:
    if probability < 0.3:
        return "LOW"
    if probability < 0.7:
        return "MEDIUM"
    return "HIGH"


@router.post("/", response_model=schemas.PredictResponse)
def predict_single(
    body: schemas.TransactionInput,
    db: Session = Depends(get_db),
    current_user: models.User = Depends(get_current_user)
):
    probability, label = get_prediction(body)

    prediction = models.Prediction(
        user_id=current_user.id,
        type="single",
        input_data=body.model_dump(),
        result={"is_fraud": bool(label), "probability":
round(probability, 4)}
    )
    db.add(prediction)
    db.commit()

    return schemas.PredictResponse(
        is_fraud=bool(label),
        probability=round(probability, 4),
        risk_level=risk_level(probability)
    )
```

Частина веб додатку:

api.js

```
// HTTP-клієнт з автоматичним додаванням JWT-токена

import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:8000',
  headers: { 'Content-Type': 'application/json' },
});

// Перехоплювач додає токен до кожного запиту
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

export const login = (email, password) =>
  api.post('/auth/login', { email, password });

export const predictSingle = (transactionData) =>
  api.post('/predict/', transactionData);

export const predictBatch = (formData) =>
  api.post('/predict/batch', formData, {
    headers: { 'Content-Type': 'multipart/form-data' },
  });

export const getHistory = () => api.get('/predict/history');
```

App.js

```
// Маршрутизація та захист приватних сторінок

import { BrowserRouter, Routes, Route, Navigate } from
'react-router-dom';
import LoginContainer      from './pages/Login/LoginContainer';
import DashboardContainer  from './pages/Dashboard/DashboardContainer';
import HistoryContainer    from './pages/History/HistoryContainer';

const PrivateRoute = ({ children }) => {
  const token = localStorage.getItem('token');
  return token ? children : <Navigate to="/login" replace />;
};

function App() {
  return (
```

```
    <BrowserRouter>
      <Routes>
        <Route path="/login" element={<LoginContainer />} />
        <Route
          path="/dashboard"
          element={<PrivateRoute><DashboardContainer /></PrivateRoute>}
        />
        <Route
          path="/history"
          element={<PrivateRoute><HistoryContainer /></PrivateRoute>}
        />
        <Route path="*" element={<Navigate to="/login" replace />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;


// DashboardContainer.jsx
// Контейнер з бізнес-логікою класифікації транзакції

import React, { Component } from 'react';
import Dashboard from './Dashboard';
import { predictSingle, predictBatch } from '../../services/api';

class DashboardContainer extends Component {
  constructor(props) {
    super(props);
    this.state = {
      singleResult:  null,
      singleLoading: false,
      singleError:   '',
    };
  }

  // Перетворює рядок CSV у структурований об'єкт транзакції
  parseCsvRow = (row) => {
    const values = row.trim().split(',').map(Number);
    const keys = [
      'V1','V2','V3','V4','V5','V6','V7','V8','V9','V10',
      'V11','V12','V13','V14','V15','V16','V17','V18','V19','V20',
      'V21','V22','V23','V24','V25','V26','V27','V28',
      'Amount', 'Time'
    ];
    if (values.length !== 30) {
      throw new Error(`Очікувалось 30 значень, отримано
${values.length}`);
    }
```

```
    return Object.fromEntries(keys.map((k, i) => [k, values[i]]));
  };

  handleSingleSubmit = async (e) => {
    e.preventDefault();
    this.setState({ singleLoading: true, singleError: '' });
    try {
      const transaction = this.parseCsvRow(e.target.csvRow.value);
      const { data } = await predictSingle(transaction);
      this.setState({ singleResult: data });
    } catch (err) {
      this.setState({ singleError: err.message || 'Помилка перевірки' });
    } finally {
      this.setState({ singleLoading: false });
    }
  };

  render() {
    return (
      <Dashboard
        singleResult={this.state.singleResult}
        singleLoading={this.state.singleLoading}
        singleError={this.state.singleError}
        onSingleSubmit={this.handleSingleSubmit}
      />
    );
  }
}

export default DashboardContainer;


// History.jsx
// Модальне вікно деталізації запису з графіками

import { PieChart, Pie, Cell, Tooltip } from 'recharts';

const COLORS = ['#F87171', '#34D399'];

const PredictionModal = ({ record, onClose }) => {
  if (!record) return null;

  const pieData = [
    { name: 'Fraud probability', value: record.result.probability },
    { name: 'Safe probability',  value: 1 - record.result.probability },
  ];

  return (
    <div className="overlay" onClick={onClose}>
      <div className="modal" onClick={(e) => e.stopPropagation()}>
```

```
        <h3>{record.type === 'single' ? 'Single Transaction' : 'Batch
Analysis'}</h3>

        <PieChart width={220} height={200}>
          <Pie
            data={pieData}
            innerRadius={55}
            outerRadius={85}
            dataKey="value"
          >
            {pieData.map((_, i) => (
              <Cell key={i} fill={COLORS[i]} />
            ))}
          </Pie>
          <Tooltip formatter={(v) => `${(v * 100).toFixed(1)}%`} />
        </PieChart>
      </div>
    </div>
  );
};
```