

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Розробка інтелектуального модуля для
динамічного навчання моделей і рекомендації
параметрів для комерційних платформ»**

Виконав студент групи ІПЗм-24-1
ВЕРБОВСЬКИЙ Олександр Юрійович

Керівник роботи:
ЛОКТИКОВА Тамара Миколаївна

Рецензент:
МОРОЗОВ Андрій Васильович

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ВЕРБОВСЬКИЙ Олександр Юрійович**

Керівник роботи: **ЛОКТИКОВА Тамара Миколаївна**

Тема роботи: **«Розробка інтелектуального модуля для динамічного навчання моделей і рекомендації параметрів для комерційних платформ»,**

затверджена Наказом закладу вищої освіти від **«18» вересня 2025 р., №491/с**

Вихідні дані для роботи: **об'єктом дослідження є процес побудови та функціонування динамічних інтелектуальних модулів прогнозування даних, а предметом дослідження є методи та алгоритми машинного навчання, а також принципи їхньої інтеграції в архітектуру комерційних платформ для прогнозування даних і автоматизації тренування моделей.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Локтікова Т.М.	22.09.2025р.	22.09.2025р.
2	Локтікова Т.М.	20.10.2025р.	20.10.2025р.
3	Локтікова Т.М.	10.11.2025р.	10.11.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота магістра присвячена розробці інтелектуального модуля для динамічного навчання моделей і рекомендації параметрів для комерційних платформ. Пояснювальна записка до випускної роботи містить текстову частину, викладену на 75 сторінках друкованого тексту. Список використаних джерел містить 45 найменувань і займає 5 сторінок. У роботі наведено 37 рисунків та 3 таблиці. Також у роботі є 12 сторінок додатків. Загальний обсяг роботи – 87 сторінок.

Метою роботи є розробка інтелектуального модуля для динамічного навчання моделей та наданні рекомендованих параметрів для кращого тренування моделей навіть користувачами без спеціальної освіти та досвіду.

Для реалізації модуля динамічного тренування моделей було обрано алгоритм випадкового лісу, для проєктування системи рекомендацій полів з сильними залежностями було використано кореляційну матрицю та χ^2 тест.

Для розробки інтелектуального модуля було використано мову програмування Python та систему керування базами даних PostgreSQL для побудови швидких вибірок. Демонстрація простоти інтеграції була здійснена за допомогою розробленої вебплатформи для оренди одягу, побудованої за допомогою технологій Next.js, та Node.js.

Застосунок пропонує зрозумілий інтерфейс, який забезпечує простий процес створення та тренування моделей для прогнозування ймовірності виникнення суперечок, а також на етапі тренування допомагає визначати адміністраторам параметри з найбільшим впливом на ризик.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ, АЛГОРИТМ ВИПАДКОВОГО ЛІСУ, ДИНАМІЧНІ ТРЕНУВАННЯ, ПРОГНОЗУВАННЯ, СИЛЬНА ЗАЛЕЖНІСТЬ ПАРАМЕТРІВ.

ABSTRACT

The master's thesis is dedicated to the development of an intelligent module for dynamic model training and parameter recommendation for commercial platforms. The explanatory note to the thesis contains 75 pages of printed text. The list of references includes 45 sources and occupies 5 pages. The work features 37 figures, 3 tables, and 12 pages of appendices. The total volume of the thesis is 87 pages.

The purpose of the research is to develop an intelligent module for dynamic model training and to provide recommended parameters that enable users without specialized education or experience to effectively train predictive models.

To implement the dynamic training module, the Random Forest algorithm was chosen. To design a system for recommending fields with strong dependencies, a correlation matrix and a χ^2 test were used.

The intelligent module was developed using the Python programming language and the PostgreSQL database management system to ensure fast data retrieval. The demonstration of integration simplicity was implemented through a developed clothing rental web platform built with Next.js and Node.js technologies.

The application offers a user-friendly interface that ensures a straightforward process of creating and training models to predict the likelihood of disputes. During training, it also helps administrators identify the parameters with the greatest influence on risk.

KEYWORDS: AUTOMATION, DYNAMIC TRAINING, PREDICTION, RANDOM FOREST ALGORITHM, STRONG PARAMETER DEPENDENCY.

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП.....	8
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ.....	10
1.1 Особливості інтеграції інноваційних технологій у комерційні процеси.....	10
1.2 Аналіз існуючих реалізацій прогнозування даних у сучасних інтерактивних платформах.....	11
1.3 Постановка задачі дослідження.....	17
1.4 Вибір програмних засобів для реалізації та інтеграції модуля динамічного тренування моделей прогнозування даних.....	18
Висновки до першого розділу.....	21
РОЗДІЛ 2 РОЗРОБКА ІНТЕЛЕКТУАЛЬНОГО МОДУЛЯ ДЛЯ ДИНАМІЧНОГО ТРЕНУВАННЯ МОДЕЛЕЙ ПРОГНОЗУВАННЯ ДАНИХ ТА РЕКОМЕНДАЦІЙ ПАРАМЕТРІВ.....	23
2.1 Огляд наявних методів та алгоритмів прогнозування даних	23
2.2 Реалізація алгоритму для тренування моделі прогнозування даних методом випадкового лісу.....	31
2.3 Реалізація механізму прогнозування даних за допомогою натренованої моделі.....	36
2.4 Алгоритм визначення релевантних параметрів для прогнозування суперечок	38
2.5 Опис бази даних демонстративного застосунку та визначення можливих параметрів для тренування моделі прогнозування ризику виникнення суперечки.....	41
Висновки до другого розділу.....	46

РОЗДІЛ 3 ІНТЕГРАЦІЯ ІНТЕЛЕКТУАЛЬНОГО МОДУЛЯ У	
ДЕМОНСТРАТИВНУ ПЛАТФОРМУ.....	48
3.1 Опис демонстративного застосунку	48
3.2 Опис адміністративної панелі демонстративного застосунку	56
3.3 Інтеграція розробленого API прогнозування ризиків в основну частину платформи	59
Висновки до третього розділу	67
ВИСНОВКИ	68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	70
ДОДАТКИ.....	75

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АПЗ – архітектура програмного забезпечення

БД – база даних

ВЛ – випадковий ліс

ПЗ – програмне забезпечення

РРЧ – режим реального часу

ШІ – штучний інтелект

API – інтерфейс застосунків (англ. Application Programming Interface)

HTTP – протокол передачі гіпертексту (англ. HyperText Transfer Protocol)

JSON – формат обміну даними JavaScript (англ. JavaScript Object Notation)

LMS – система управління навчанням (англ. Learning Management System)

ML – машинне навчання (англ. Machine Learning)

SPA – односторінкові застосунки (англ. Single Page Applications)

UI – інтерфейс користувача (англ. User Interface)

UX – досвід користувача (англ. User Experience)

ВСТУП

Актуальність теми зумовлена стрімким зростанням популярності та інтересу замовників до інтеграції сучасних рішень на основі штучного інтелекту, а також їхньою ціною. У середньому, замовник платить 40-50 доларів за годину роботи одного спеціаліста, що зазвичай робить рішення на базі ШІ за участі фахівців недоступними для багатьох малих бізнесів, які не потребують складних рішень. Крім того, нерідко на старті проєктів відбуваються зміни напряму діяльності, а також не відразу отримують достатню кількість даних, що може призводити до неточності моделі з часом, що, в свою чергу, призводитиме до потреби повторного аналізу та перетреноувати моделі прогнозування.

Також важливо розуміти, що подібні замовлення не є довготривалими, через це ймовірність знайти відповідального та якісного спеціаліста стає значно нижчою. Це пов'язано з низьким особистим інтересом розробників до виконання простих та шаблонних проєктів, а також із фінансовою винагородою.

Таким чином, оптимальним рішенням є розробка дешевого механізму, здатного динамічно тренувати моделі, а також автоматизовано надавати рекомендації щодо вибору параметрів для якісного тренування, що зменшить витрати замовників, а також дозволить зменшити кількість нецікавих типових замовлень для розробників.

Метою даної роботи є розробка інтелектуального модуля для динамічного навчання моделей та надання рекомендованих параметрів для їхнього кращого тренування навіть користувачами без спеціальної освіти та досвіду.

Для реалізації поставленої мети потрібно виконати наступні **завдання**:

1. Здійснити аналіз сучасних підходів до прогнозування даних.
2. Здійснити вибір засобів проєктування, ретельно оцінюючи доступні інструменти та технології.

3. Визначити універсальні порогові значення для системи рекомендацій.

4. Розробити API частину для побудови моделей та використання їхніх прогнозів із застосуванням алгоритму випадкового лісу.

5. Реалізувати інтеграцію автоматизованої системи тренування моделей прогнозування даних у комерційну платформу.

Об'єктом дослідження є процес побудови та функціонування динамічних інтелектуальних модулів прогнозування даних.

Предметом дослідження є методи та алгоритми машинного навчання, а також принципи їхньої інтеграції в архітектуру комерційних платформ для прогнозування даних і автоматизації тренування моделей.

Наукова новизна полягає у поєднанні вебтехнологій зі моделями прогнозування даних, що дозволяє створювати гнучкі платформи, здатні адаптуватися до різних видів комерційної діяльності.

Практичне значення полягає у можливості реального впровадження описаного модуля для зниження вартості та пришвидшення реалізації модулів прогнозування даних у комерційних платформах.

Апробація. За темою випускної кваліфікаційної роботи магістра:

1) опубліковано тези доповіді Verbovskiy O. Yu., Loktikova T.M. Using Data Prediction Methods and Algorithms to Improve the Interactive Platform User Experience. Тези доповідей Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки, 12-17 травня 2025 року. Житомир: “Житомирська політехніка”, 2025. С. 52-53 [1];

2) подано та прийнято до опублікування статтю Вербовський О., Локтікова Т., Кушнір Н., Лисогор Ю. Дослідження принципів побудови та проектування інтерактивної платформи з використанням методів та алгоритмів прогнозування даних. Вісник Хмельницького національного університету. Серія: технічні науки. 2026. №1 [2].

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Особливості інтеграції інноваційних технологій у комерційні процеси

Протягом усієї історії розвитку людство прагне автоматизувати рутинні процеси, підвищуючи продуктивність праці та точність прийняття рішень. Тепер це прагнення еволюціонувало від створення різних механічних пристроїв і промислових фабрик до розробки складних сучасних програмних систем. Однак, попри значний прогрес у сфері інформаційних технологій та велику кількість реалізованих автоматизованих рішень, усе ще існує велика кількість процесів, які можна оптимізувати для покращення рівня життя людей усіх верств населення.

Гарним прикладом такого процесу є інтеграція популярних та модних реалізацій у власні проєкти особливо, у теперішній час. До прикладів належать інтеграції різних сучасних технологій, по типу штучного інтелекту, автоматизованих систем для ведення бізнесу та алгоритмів для створення персоналізованих рекомендацій, що відкривають нові горизонти для досягнення ефективності. Крім того, сучасні розробки дозволяють не лише автоматизувати рутинні операції, але й робити їх більш адаптивними та гнучкими, що знижує людську участь, мінімізує помилки і головне – досить часто зменшує фінансові витрати власників.

Однак, щоразу, як потрібно оновити модель з різних причин, наприклад, бо вона застаріла, чи більше не відповідає потребам бізнесу, компаніям потрібно тимчасово наймати фахівців. Враховуючи високі витрати на наймання спеціалістів, які мають знання та досвід у сфері аналізу даних та машинного навчання, це стає серйозною перепорою для більшості малих компаній, які не завжди можуть дозволити собі витратити такі фінанси навіть нетривалий період.

Крім того, щоразу коли відбувається найм робітника, є певний ризик, що співпраця не буде успішною. Буває, що просто спеціаліст не відповідає поставленій задачі, буває, що задача поставлена не зрозуміло і задачу виконано відповідно, а буває, що взагалі спеціаліст не зацікавлений у виконанні завдання, що можна назвати рутинним. Крім того, є ще одна важлива проблема при виконанні роботи людиною, а саме – людський фактор, який неможливо повністю передбачити. Неналаштовані процеси комунікації, недооцінка ризиків однією зі сторін, та навіть просто особисті конфлікти можуть призвести до значних труднощів у проєкті. У деяких випадках, такі проблеми можуть призвести не лише до витраченого часу та фінансів, а і до катастрофічних наслідків, по типу, повного закриття бізнесу, якщо проєкт занадто сильно залежний від тренуваних моделей.

Таким чином, гарним рішенням проблеми є механізм динамічного тренування моделей, адже будь-хто будь-коли зможе створити власну модель та перевірити її ефективність у реальних умовах. Однак, навіть щоб тренувати моделі, користувачі повинні опиратись на дані, та їх візуалізації, що все одно потребує спеціальних знань на достатньому рівні і може призвести до неприємних ситуацій пов'язаних безпосередньо з низькою кваліфікацією користувача у сфері прогнозування даних. Тому для простішого використання такого механізму, варто розробити автоматизовану систему рекомендації параметрів з сильною залежністю.

1.2 Аналіз існуючих реалізацій прогнозування даних у сучасних інтерактивних платформах

Під час розробки модуля було здійснено детальний аналіз та порівняння різних впроваджених реалізацій систем прогнозування даних, які мають подібне призначення, функціонал та механізми взаємодії з користувачами. Враховувалися різні аспекти, зокрема точність прогнозів, можливість прогнозування в режимі реального часу, а також адаптивність до

різних сценаріїв і типів даних. Особливу увагу було приділено тому, як ці рішення реалізують автоматизовані процеси прийняття рішень, з якою кількістю даних працюють, а також які типи алгоритмів машинного навчання використовують для отримання ймовірності виникнення певної події. Це дозволило визначити найефективніше рішення для початкової стадії, яке б задовольняло поставлені завдання і забезпечувало високу точність роботи. До списку розглянутих платформ належать PandaDoc, Salesforce Einstein та Gong.

Першою розглянутою платформою, яка використовує подібні рішення, є PandaDoc (<https://www.pandadoc.com>). Дана платформа є інструментом для автоматизації документообігу, що допомагає компаніям створювати, відправляти та підписувати документи різного типу, наприклад: комерційні пропозиції, угоди та контракти. Вона спрощує робочий процес, надаючи інструменти для створення інтерактивних документів із шаблонів, можливості спільної роботи та відстеження активності в реальному часі.

На платформі наявна можливість прогнозування успіху. Аналіз даних відбувається на основі інформації про взаємодію з документами, наприклад, час перегляду, частоту відкриття та дії, що вживаються одержувачем. Використовуючи ці дані, платформа надає аналітику щодо ймовірності укладення угоди, допомагаючи сейлз-менеджерам приймати обґрунтовані рішення.

Це дозволяє компаніям глибше аналізувати поведінку потенційних клієнтів, виявляючи ключові патерни та тренди, що допомагають краще розуміти їхні потреби та вподобання. Завдяки цьому бізнеси можуть точно налаштовувати свої маркетингові стратегії та пропозиції, персоналізуючи контент і пропозиції для кожного сегмента клієнтів, що в результаті підвищує ефективність продажів та загальні прибутки. Оцінка ймовірності відображається у звітах та дашбордах.

На рис. 1.1 представлено приклад згенерованого звіту на платформі PandaDoc.



PandaDoc

Annual recurring revenue (ARR)

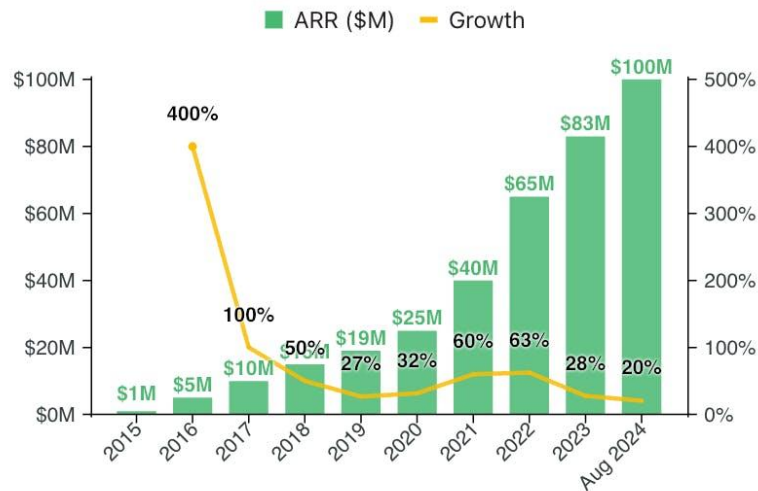


Рисунок 1.1 – Приклад звіту PandaDoc

Другою розглянутою реалізацією є Salesforce Einstein (<https://www.salesforce.com/>). Це вбудоване у CRM Salesforce рішення на основі штучного інтелекту, яке надає можливість аналізувати поведінку клієнтів і прогнозувати результати угод. Платформа використовує історичні дані, такі як електронні листи, дзвінки, активність у соціальних мережах і демографічні показники, щоб надавати персоналізовані рекомендації для продажів та маркетингу. Це допомагає командам ефективніше залучати клієнтів, підвищувати продуктивність і прогнозувати майбутні доходи.

На додачу, таке рішення підвищує ефективність управління продажами, а також сприяє кращій взаємодії з клієнтами, дозволяючи компаніям приймати більш обґрунтовані рішення на основі аналізу даних. Прогнозування на основі штучного інтелекту дозволяє оптимізувати процеси, знижуючи ймовірність людських помилок та суб'єктивних рішень, що в свою чергу забезпечує більш високий рівень довіри клієнтів до платформи.

Ймовірність успіху кожної угоди формується на основі машинного навчання та аналізу великих обсягів даних. Функціонал включає прогнозування продажів, автоматичне визначення пріоритетних лідів та можливостей, а також виявлення ризиків у процесі укладення угод.

На рис. 1.2 представлено приклад звітності за допомогою рішення Salesforce Einstein.

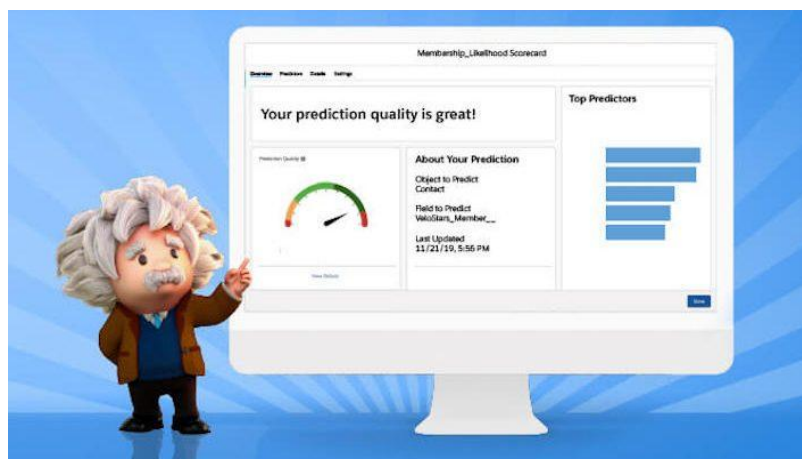


Рисунок 1.2 – Приклад звітності Salesforce Einstein

Останнім розглянутим рішенням є аналітична платформа Gong (<https://www.gong.io/>). Вона розроблена для управління продажами. Її основна специфіка – це запис та аналіз взаємодії команд із клієнтами через дзвінки, електронну пошту та відеозустрічі. Платформа допомагає компаніям зрозуміти, які аспекти комунікації найбільш ефективні для закриття угод, визначаючи найкращі стратегії взаємодії з клієнтами. Gong використовує штучний інтелект для розпізнавання мовлення та аналізу тональності, що дозволяє визначити настрій і зацікавленість клієнта.

Такий підхід дозволяє не лише прогнозувати ймовірність успіху угод, але й визначати слабкі місця у комунікації, що можуть стати причиною провалу. Таким чином, визначивши точний настрій клієнта працівники можуть змінювати стратегію взаємодії в реальному часі, підвищуючи ефективність своїх продажів, та не витрачати час на порожню балаканину, яка не принесе користі компанії.

Щодо прогнозування успіху, Gong аналізує розмови та дії клієнтів, виявляючи патерни, що корелюють із успішними продажами. Система надає детальні аналітичні звіти про те, які фрази чи теми сприяють укладенню угод, а також підкреслює потенційні ризики у спілкуванні.

На рис. 1.3 представлено головну сторінку платформи Gong.

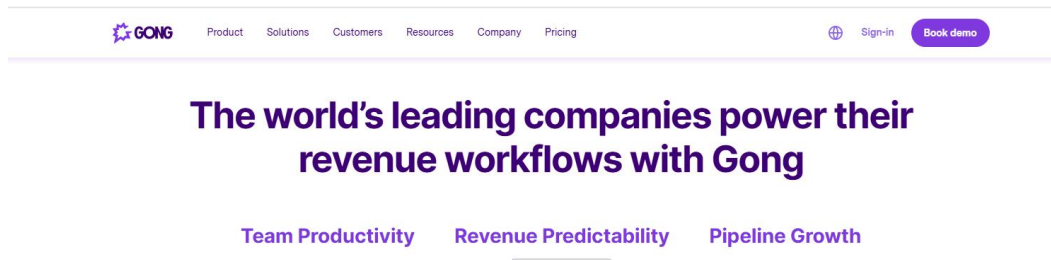


Рисунок 1.3 – Головна сторінка Gong

У табл. 1.1 представлена порівняльна характеристика розглянутих реалізацій за основними параметрами.

Таблиця 1.1

Порівняльна характеристика реалізацій прогнозування даних

Платформа	PandaDoc	Salesforce Einstein	Gong
1	2	3	4
Методи передбачення успіху	Аналіз взаємодії з документами (час перегляду, частота відкриття, дії)	Аналіз історичних даних про угоди, поведінку клієнтів, та їх взаємодії	Аналіз взаємодії з клієнтами, вивчення дзвінків, листувань та зустрічей
Відображення прогнозованих даних	Надання аналітики щодо ймовірності укладення угоди через панель управління	Ймовірність успіху угоди відображається як прогнозовані на дашбордах	Прогнозовані ймовірності успіху угоди на основі комунікацій та сигналів

Продовження табл. 1.1

1	2	3	4
Кількість даних, з якою працює	Працює навіть з обмеженою кількістю документів	Підходить для середніх та великих об'ємів даних	Найкраще працює при наявності великого обсягу історичних комунікацій
Особливості	Аналіз активності користувачів для підвищення конверсії	Інтеграція з CRM-системою для прогнозування результатів	Використання понад 300 сигналів для прогнозування результатів

Аналіз реалізацій дозволив визначити основні напрями та моделі в області прогнозування даних. Серед них – текстовий аналіз (наприклад, аналіз текстової частини опису пропозиції, замовлення, його назви чи аналізу листувань у внутрішньому чаті), аналіз часових факторів (таких як час від створення замовлення до його підтвердження, час між комунікацією замовника і виконавця, та час який проходить між завершенням замовлення та його оцінкою користувачами), а також аналіз ключових параметрів (таких як кількість успішно виконаних замовлень виконавцем, кількість успішно оплачених оренд замовником, кількість коментарів у рейтингах користувачів, тривалість виконання замовлення та його ціна). Порівнявши ці приклади реалізацій, було обрано формат аналізу параметрів, адже він популярніший, його легше розробити та інтегрувати.

1.3 Постановка задачі дослідження

Проаналізувавши існуючі рішення в сфері прогнозування даних, можна визначити, що основним недоліком існуючих платформ є відсутність автоматизованого механізму для тренування нових моделей прогнозування даних. Багато платформ покладаються на ручне тренування моделей спеціалістами, що значно збільшує фінансові витрати навіть при незначній зміні напрямку діяльності платформи.

Для вирішення цієї проблеми необхідно розробити систему автоматизації, яка спростить процеси аналізу даних та тренування моделей, знижуючи необхідність у постійному залученні спеціалістів. Крім того, передбачається, що описана система повинна мати можливість інтеграції у вже існуючі платформи для обробки даних, що можливо зробити за допомогою API-ендпоїнтів та спільної бази даних з проєктом. Це дозволить побудувати ізольовану систему, що дозволить значно зменшувати час, необхідний для адаптації коду модуля до нових проєктів, і підвищити гнучкість тренувань підбираючи деякі параметри моделей спеціально для проєкту.

Щодо спрощення процесу тренування, то варто розробити механізм для визначення сильно залежних параметрів, що буде надавати користувачам рекомендацію щодо використання їх при тренуванні. Таке рішення не лише дасть змогу користувачам швидше обирати важливі параметри, а й зменшить ймовірність виникнення помилок у зв'язку з їх некваліфікованістю.

Таким чином, дана робота має на меті розробку інтелектуального модуля для динамічного навчання моделей та надання рекомендованих параметрів для їх кращого тренування навіть користувачами без спеціальної освіти та досвіду, водночас заощаджуючи кошти для невеликих компаній і дозволяючи спеціалістам працювати лише над складними та унікальними завданнями.

Крім того, дана робота включає в себе вивчення можливих методів та алгоритмів у цьому напрямі, вибір алгоритму, який буде підходити для максимально широкого спектру проєктів, визначення граничних значень для системи автоматичного визначення параметрів із сильною залежністю, а також інтеграцію розробленого модуля до демонстративної платформи. На додачу, у роботі приділено увагу оновленню структури бази даних при інтеграції розробленого сервісу, а також прикладу реалізації фронтенд-частини для зручного використання механізму тренування моделей адміністраторами.

Також варто зауважити, що розробка цього модуля не лише дозволить оптимізувати описані процеси, а й відкриває можливості для подальшого розвитку. У майбутньому система може бути адаптована для роботи з іншими алгоритмами машинного навчання, а також інтегрувати новітні технології, такі як глибинне навчання, що дозволить значно підвищити точність прогнозів і забезпечить гнучкість, чим розширить ринок, на якому можна буде використовувати цей модуль.

Таким чином, після виконання раніше описаних завдань, буде отримано надійний динамічний модуль для навчання моделей та надання рекомендованих параметрів для їх кращого тренування з можливістю швидкої інтеграції у існуючі комерційні платформи, по типу застосунку для оренди одягу.

1.4 Вибір програмних засобів для реалізації та інтеграції модуля динамічного тренування моделей прогнозування даних

Оскільки розроблюваний модуль буде використовуватись для прогнозування даних, у проєкті буде використано окремий сервер, розроблений за допомогою високорівневої мови Python, адже ця мова має велику кількість бібліотек для реалізацій високоякісних моделей для прогнозування даних [3 – 5].

Однією з них є `sklearn`, яка надає широкий набір інструментів для прогнозування даних [6, 7]. Також у мові Python розроблено бібліотеку `Pandas` [8, 9], завдяки якій можна ефективно обробляти та аналізувати великі обсяги даних. Це стає гарним доповненням, до раніше згаданої `sklearn`, адже у парі вони надають можливість швидкого та якісного натренування моделі [10]. На додачу, інструменти дозволяють обмежуватись невеликим об'ємом коду, що також зменшує ймовірність помилок під час розробки, заощаджуючи час на тестуванні.

Ще однією позитивною якістю Python є засоби для роботи з HTTP-протоколами. Наприклад, завдяки мікрофреймворку можна легко створити безпечні API-ендпоїнти, які забезпечуватимуть зручний доступ до функціоналу створення, перемикавання та отримання прогнозів розробленого алгоритму для сторонніх серверів [11].

Приклад схеми взаємодії розробленого модуля з комерційною платформою зображено на рис. 1.4.

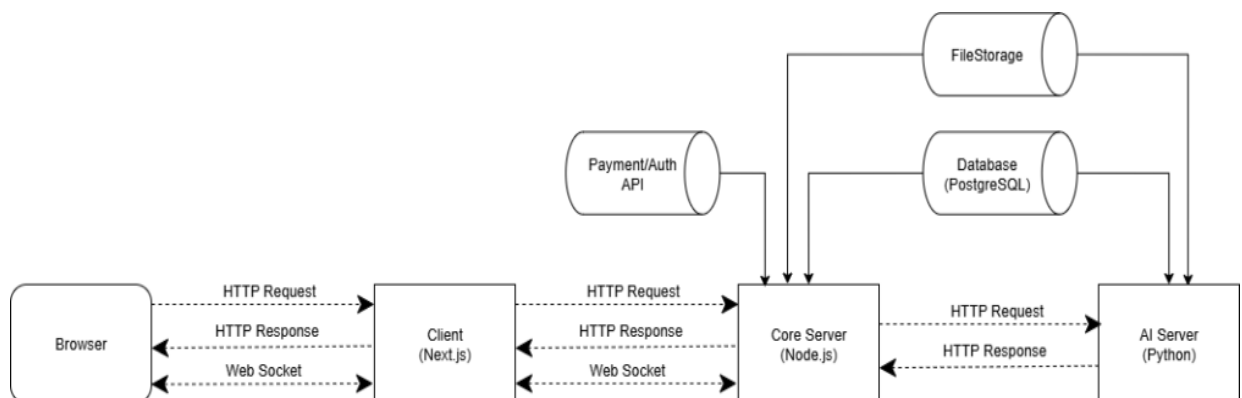


Рисунок 1.4 – Приклад схеми взаємодії розробленого модуля з комерційною платформою

Також варто зауважити, що у мові Python передбачена багатопотокова реалізація. Це можливо завдяки стандартному модулю `threading`, який дозволяє запускати окремі потоки виконання всередині одного процесу [12, 13]. Кожен потік виконує власну задачу незалежно від головного циклу

програми, тому такий підхід дає змогу не блокувати основний сервер Flask під час виконання тривалих обчислень.

Крім того, для управління потоками використовується черга завдань з модуля `queue`, яка забезпечує синхронізований обмін повідомлень між потоками – головний потік додає завдання у чергу, а робочий потік їх обробляє [14]. Це гарантує безпечну взаємодію між потоками та запобігає конфліктам при одночасному доступі до спільних ресурсів.

Також не можна не згадати бібліотеку `psycopg2` у мові Python, адже вона є найпопулярнішою для роботи з PostgreSQL. На етапі збирання даних для тренування моделі, ця бібліотека буде дуже корисною, адже вона дозволяє отримувати реальні дані безпосередньо з бази даних проєкту. Таким чином, буде завчасно уникнено будь-яких помилок, пов'язаних саме з пошкодженням даних. Наприклад, при переносі даних у файл в ручному режимі, могла десь трапитись опечатка, або ще якийсь людський фактор. Крім того, при написанні ручного скрипта, є ймовірність виникнення помилки пов'язаної з погано продуманим сценарієм, або не оптимізованим запитом. Тому, пряме підключення є найкращим варіантом для отримання даних задля тренування моделей прогнозування даних.

Рішення використовувати PostgreSQL для збереження даних було прийнято через його високу продуктивність при роботі з великою кількістю даних [15, 16]. Також ця СУБД підтримує складні типи даних, такі як JSON, який дозволяє зберігати напівструктуровані дані з можливістю швидкого пошуку та індексації по них, що є дуже зручним для збереження та обробки динамічної структури вибірки.

Як згадувалось раніше, для коректної інтеграції розроблюваного модуля, важливо, щоб основна частина платформи, у яку відбуватиметься інтеграція, мала можливість роботи зі сторонніми API [17]. Це важливо, через те, що є можливість взаємодії із будь-якими сторонніми сервісами, не залежно від стеку, що ідеально підходить для розроблюваного за допомогою Python модуля.

Враховуючи раніше описану вимогу до веб-застосунку для легкої інтеграції, буде розроблено просту демонстративну платформу, де для серверної частини буде використовуватись Node.js, а для клієнтської – Next.js [18 – 20].

Для виконання розробки веб-частини проєкту обрано Visual Studio Code. Це інтегроване середовище розробки, яке надає редактор з можливостями аналізу коду, автоматизованими засобами відловлення та виправлення помилок, а також широким спектром розширень для підтримки різних мов та технологій, включно з JavaScript та Python. Також до середовища було додано деякі розширення, що зробило застосунок ще зручнішим для розробки та дозволило пришвидшити роботу над розробкою, зокрема завдяки розширенню для автоматичного доповнення коду Tabnine.

Для швидкої взаємодії з базою даних, буде використано програму DBeaver, адже це зручний та швидкий інструмент для роботи з PostgreSQL. Також даний застосунок має графічний інтерфейс для перегляду таблиць, фільтрації вибірки, її сортування, виконання SQL-запитів та адміністрування бази даних. Саме описані переваги та наявність автодоповнення скрипта дозволяє значно прискорювати процеси розробки та тестування структури сховища даних та самих запитів.

Висновки до першого розділу

У цьому розділі було проведено детальний аналіз популярних реалізацій систем прогнозування даних, які використовують максимально різні підходи до обробки та аналізу даних. Завдяки цьому було виділено три основні підходи – текстовий аналіз, аналіз часових показників, а також аналіз основних параметрів, який і було прийнято для поточної реалізації, оскільки він дозволяє охопити більшу кількість проєктів, у які можна буде інтегрувати розроблюваний модуль.

Також у цьому розділі було сформовано завдання для розробки модуля для динамічного тренування моделей прогнозування даних, який включає в себе вивчення можливих методів та алгоритмів у цьому напрямі, вибір алгоритму, який буде підходити для максимально широкого спектру проєктів, визначення граничних значень для системи автоматичного визначення параметрів із сильною залежністю, а також інтеграцію розробленого модуля до демонстративної платформи.

На завершення, у даному розділі було прийнято рішення щодо використання мови програмування Python, через наявність зручних бібліотек для розробки та тренування моделей прогнозування даних. Прикладами є scikit-learn для реалізації алгоритмів машинного навчання та Pandas для ефективної обробки даних. Для взаємодії з базою даних було обрано PostgreSQL, через її високу продуктивність при роботі з великими масивами даних і підтримку складних типів даних. Крім того, для забезпечення зручності інтеграції та доступу до модуля, буде створено API за допомогою фреймворку Flask.

РОЗДІЛ 2 РОЗРОБКА ІНТЕЛЕКТУАЛЬНОГО МОДУЛЯ ДЛЯ ДИНАМІЧНОГО ТРЕНУВАННЯ МОДЕЛЕЙ ПРОГНОЗУВАННЯ ДАНИХ ТА РЕКОМЕНДАЦІЇ ПАРАМЕТРІВ

2.1 Огляд наявних методів та алгоритмів прогнозування даних

На сьогодні існує безліч методів прогнозування даних. Одні з них призначені для виявлення залежностей між числовими величинами, інші – для класифікації, а деякі здатні аналізувати текст і знаходити закономірності навіть у ньому. Таким чином, для реалізації функціоналу для передбачення, є багато підходів і інструментів, тож важливим процесом є прийняття рішення щодо використання методу, який найкраще підійде для поточного завдання.

Отже, першим розглянутим методом є один з найпростіших для розуміння та реалізації, а саме – поліноміальна регресія. Це статистичний метод, який використовується для пошуку взаємозв'язку між змінними, за допомогою рівняння певного порядку [21].

Метод є досить простим, оскільки для його тренування достатньо перевести всі дані в систему координат і підібрати рівняння, яке найкраще описує ці дані. Також модель такого методу легко перевірити, за допомогою візуалізації, адже достатньо зобразити координати і домалювати підібрану криву. Чим вищий ступінь рівняння, тим точніше буде прогноз даних, однак, через те, що такий підхід підлаштовує рівняння до тренувальних даних, то без певної нормалізації та фільтрації даних є досить великий ризик перенавчання, що може призвести до отримання некоректних результатів.

Також важливо зауважити, що даний метод може працювати з декількома параметрами, однак він досить погано працює з великою кількістю параметрів. Це може призвести до перенавчання моделі при використанні категоріальних ознак, оскільки вона намагається підлаштуватися під конкретні дані, а не знаходити загальні закономірності, що не підходить для поточної реалізації.

Далі розглянемо дерева рішень – це один з найбільш поширених алгоритмів для класифікації та регресії. Для реалізації моделі, будується загальне дерево, де розписують кроки залежно від даних на основі певної ознаки [22 – 24]. Листками в цих деревах є кінцеві прогнози. При кожному новому кейсі, будується нова гілка разом з листком, що робить такий метод дуже гнучким і здатним працювати з новими кейсами у майбутньому. Таку модель легко тестувати, адже достатньо отримати зображення дерева, щоб переглянути хід прогнозу.

Кожен рівень дерева зменшує кількість елементів для подальшої обробки вдвічі, тому часова складність дерев рішень зазвичай складає $O(\log n)$, де n – кількість елементів у наборі даних. Таким чином, загальний час на побудову прогнозу зростає значно повільніше, ніж лінійно з кількістю елементів, що робить метод ефективним навіть для великих наборів даних.

Однак, такі дерева також схильні до перенавчання. Дерево може стати занадто глибоким і підлаштовуватиметься під тренувальні дані, навіть якщо маленька зміна призведе до перебудови значного шматка. У подальшому це може почати призводити до помилкових відповідей.

Тому варто зауважити, що даний алгоритм має вдосконалення. Першим із них є метод випадкового лісу, що комбінує багато дерев рішень [25 – 27]. Це призводить до отримання більш стабільного та точного прогнозу, адже кожне дерево створюється на основі випадково обраних підмножин даних. Фінальний результат можна отримати не лише у вигляді відповіді “Так” чи “Ні”, або 0/1 чи “True”/“False”, а й у відсотках, завдяки тому, що фінальний результат можна отримати шляхом усереднення прогнозів.

Іншим вдосконаленням дерева рішень є градієнтний метод XGBoost [28 – 30]. У ньому також використовується множина дерев рішень, однак на відміну від випадкового лісу, дерева будуються послідовно. Кожне наступне дерево навчається з урахуванням помилок, допущених попередніми моделями, що дозволяє поступово покращувати точність прогнозу. Метод оптимізує задану функцію втрат із використанням градієнтного спуску та

містить механізми регуляризації, які запобігають перенавчанню. Фінальний прогноз формується шляхом зваженого підсумовування результатів усіх дерев, що також дозволяє отримувати не лише бінарні значення, а й оцінку у відсотковому вигляді.

Як можна побачити, обидва вдосконалення можна використати в даному проєкті, тому ще одним важливим завдання стало експериментальне визначення модифікації алгоритму випадкового дерева, що найліпше підходить для реалізації у даній роботі. Для цього було використано три різних за кількістю та типами даних датасети з платформи www.kaggle.com,

Тестові тренування моделей відбувались в однакових умовах, для визначення числових параметрів з міцними залежностями було використано кореляційну матрицю, а для категоріальних – тест χ^2 . Пороговими значеннями було обрано значення 0.6 та 0.05 для відповідних перевірок, як загальноприйняті для оцінки сили зв'язків між параметрами. Таким чином, було гарантовано, що для однакових датасетів буде визначено однакові важливі параметри, на основі яких будуть відбуватись тестові тренування, а також визначено точність прогнозів таких моделей.

У першому наборі даних містилась інформація про справжні та фейкові оголошення щодо найму робітників. У ньому було аж 17880 рядків даних, що є досить близьким числом до об'ємів, з якими очікується, що буде працювати розроблений модуль. У даному датасеті були не лише категоріальні та числові параметри, а й текстові, однак було прийнято рішення уникати їх використання, так як для цього потрібні складніші алгоритми.

Результати тренування зображено на рис. 2.1 та 2.2.

```
(venv) PS E:\dplome\new-restart\forest-server> py field_prediction_test.py --file_name=recruitment-scam --numerical_fields=salary_range --categorical_fields=department,location,telecommuting,has_company_logo,has_questions,employment_type,required_experience,required_education,industry,function,in_balanced_dataset --check_field=fraudulent
correlation_threshold 0.6
p_value_threshold 0.05
dependencies: {'correlation_dict': {'min_price': np.float64(0.0020752388430255657), 'max_price': np.float64(0.001842742927288229)}, 'chi2_dict': [{'department': 0.0726097485993974}, {'has_company_logo': 3.5943691337772625e-268}, {'has_questions': 2.515984961488263e-34}, {'location': 0.06937788859133931}, {'employment_type': 2.4377014440608537e-13}, {'required_experience': 3.79020423434021e-19}, {'telecommuting': 0.78986891319539}, {'required_education': 7.684947749658579e-101}, {'industry': 3.8619427145446395e-41}, {'function': 9.284874065372859e-97}, {'in_balanced_dataset': 0.0}]}
Filtered numeric fields based on correlation threshold: []
Filtered categorical fields based on p-value threshold: ['has_company_logo', 'has_questions', 'employment_type', 'required_experience', 'required_education', 'industry', 'function', 'in_balanced_dataset']
accuracy 95.71428571428572
(venv) PS E:\dplome\new-restart\forest-server> 
```

Рисунок 2.1 – Тестове тренування RF моделі на основі датасету з оголошеннями про найм на роботу

```
(venv) PS E:\dyp\lone\new-restart\forest-server> py field_prediction_test.py --file_name=recruitment-scam --numerical_fields=salary_range --categorical_fields=department,location,telecommuting,has_company_logo,has_questions,employment_type,required_experience,required_education,industry,function,in_balanced_dataset --check_field=fraudulent
correlation_threshold 0.6
p_value_threshold 0.05
dependencies: {'correlation_dict': {'min_price': np.float64(0.0020752388430255657), 'max_price': np.float64(0.001842742927288229)}, 'chi2_dict': [{'department': 0.0726097485993974}, {'has_company_logo': 3.5943691337772625e-268}, {'has_questions': 2.515984961488263e-34}, {'location': 0.06937788859133931}, {'employment_type': 2.437701444068537e-13}, {'required_experience': 3.7902042343021e-19}, {'telecommuting': 0.78986891319539}, {'required_education': 7.684947749658579e-101}, {'industry': 3.8619427145446395e-41}, {'function': 9.284874065372859e-97}, {'in_balanced_dataset': 0.0}]}
Filtered numeric fields based on correlation threshold: []
Filtered categorical fields based on p-value threshold: ['has_company_logo', 'has_questions', 'employment_type', 'required_experience', 'required_education', 'industry', 'function', 'in_balanced_dataset']
accuracy 97.14285714285714
(venv) PS E:\dyp\lone\new-restart\forest-server>
```

Рисунок 2.2 – Тестове тренування Xgboost моделі на основі датасету з оголошеннями про найм на роботу

Наступний датасет містив інформацію про успішні інвестиції. Даний датасет містив лише 4000 рядків даних, однак для експерименту цього достатньо, адже розроблюваний модуль повинен показувати себе якнайкраще навіть з невеликим набором даних. У ньому були лише категоріальні та числові параметри.

Результат експериментального визначення точності моделі випадкового лісу на основі датасету з успішним інвестуванням зображено на рис. 2.3.

```
(venv) PS E:\dyp\lone\new-restart\forest-server> py field_prediction_test.py --file_name=movie-gross --numerical_fields=score,votes,budget,runtime,gross --categorical_fields=year,genre,director,writer,star,country,company --check_field=is_earn_money
correlation_threshold 0.6
p_value_threshold 0.05
dependencies: {'correlation_dict': [{'score': 0.20567634738923632}, {'votes': 0.6087187440018885}, {'budget': 0.5838483648885644}, {'runtime': 0.2233065762257989}], 'chi2_dict': {'rating': np.float64(0.17117823855409273), 'year': np.float64(0.0018421166712555923), 'genre': np.float64(0.02350050403633423), 'director': np.float64(0.002998234407429595), 'writer': np.float64(0.0001099361773747445), 'star': np.float64(0.01674151216306155), 'country': np.float64(0.2535092197796832), 'company': np.float64(3.8598835784962986e-05)}}
Filtered numeric fields based on correlation threshold: ['votes']
Filtered categorical fields based on p-value threshold: ['year', 'genre', 'director', 'writer', 'star', 'company']
accuracy 65.26315789473685
(venv) PS E:\dyp\lone\new-restart\forest-server>
```

Рисунок 2.3 – Тестове тренування RF моделі на основі датасету з успішними інвестуваннями

На рис. 2.4 зображено результат експериментального визначення точності натренованої моделі з використанням алгоритму Xgboost на основі даних з успішним інвестуванням.

```
(venv) PS E:\dyp\lone\new-restart\forest-server> py field_prediction_test.py --file_name=movie-gross --numerical_fields=score,votes,budget,runtime,gross --categorical_fields=year,genre,director,writer,star,country,company --check_field=is_earn_money
correlation_threshold 0.6
p_value_threshold 0.05
dependencies: {'correlation_dict': [{'score': 0.20567634738923632}, {'votes': 0.6087187440018885}, {'budget': 0.5838483648885644}, {'runtime': 0.2233065762257989}], 'chi2_dict': {'rating': np.float64(0.17117823855409273), 'year': np.float64(0.0018421166712555923), 'genre': np.float64(0.02350050403633423), 'director': np.float64(0.002998234407429595), 'writer': np.float64(0.0001099361773747445), 'star': np.float64(0.01674151216306155), 'country': np.float64(0.2535092197796832), 'company': np.float64(3.8598835784962986e-05)}}
Filtered numeric fields based on correlation threshold: ['votes']
Filtered categorical fields based on p-value threshold: ['year', 'genre', 'director', 'writer', 'star', 'company']
accuracy 70.52631578947368
(venv) PS E:\dyp\lone\new-restart\forest-server>
```

Рисунок 2.4 – Тестове тренування Xgboost моделі на основі датасету успішності інвестування

Останнім розглянутим датасетом був набір, що містив перелік записів із затримками авіарейсів. Даний датасет містив аж 539382 рядків даних, що також дозволило перевірити швидкість та якість тренування моделей при великій кількості даних. Він містив як категоріальні так і числові параметри.

На рис. 2.5 представлено результат експериментального визначення точності натренованої моделі з використанням алгоритму випадкового лісу на основі датасету з затримками авіарейсів.

```
(venv) PS E:\dyp\lome\new-restart\forest-server> py field_prediction_test.py --file_name=airlines-delay --numerical_fields=Time,Length --categorical_fields=Airline,AirportFrom,Airpo
rtTo,DayOfWeek --check_field=Class
correlation_threshold 0.6
p_value_threshold 0.05
dependencies: {'correlation_dict': {'Time': np.float64(0.15045680780283274), 'Length': np.float64(0.04048828550851612)}, 'chi2_dict': {'Airline': np.float64(0.0), 'AirportFrom': np
.float64(2.041368758698839e-98), 'AirportTo': np.float64(0.0), 'DayOfWeek': np.float64(2.735034036561441e-251)}}
Filtered numeric fields based on correlation threshold: []
Filtered categorical fields based on p-value threshold: ['Airline', 'AirportFrom', 'AirportTo', 'DayOfWeek']
accuracy 68.57142857142857
(venv) PS E:\dyp\lome\new-restart\forest-server>
```

Рисунок 2.5 – Результат тренування RF моделі на основі датасету з затримками авіарейсів

Результат обрахунку точності моделі на основі алгоритму Xgboost з використанням датасету затримок авіарейсів представлено на рис. 2.6.

```
(venv) PS E:\dyp\lome\new-restart\forest-server> py field_prediction_test.py --file_name=airlines-delay --numerical_fields=Time,Length --categorical_fields=Airline,AirportFrom,Airpo
rtTo,DayOfWeek --check_field=Class
correlation_threshold 0.6
p_value_threshold 0.05
dependencies: {'correlation_dict': {'Time': np.float64(0.15045680780283274), 'Length': np.float64(0.04048828550851612)}, 'chi2_dict': {'Airline': np.float64(0.0), 'AirportFrom': np
.float64(2.041368758698839e-98), 'AirportTo': np.float64(0.0), 'DayOfWeek': np.float64(2.735034036561441e-251)}}
Filtered numeric fields based on correlation threshold: []
Filtered categorical fields based on p-value threshold: ['Airline', 'AirportFrom', 'AirportTo', 'DayOfWeek']
accuracy 62.857142857142854
(venv) PS E:\dyp\lome\new-restart\forest-server>
```

Рисунок 2.6 – Результат тренування Xgboost моделі на основі датасету з затримками авіарейсів

Як можна побачити з рис. 2.1 – 2.4 алгоритм Xgboost давав трохи кращу точність, однак з рис. 2.5 та 2.6 видно, що з датасетом для прогнозування затримки авіарейсів він справився значно гірше (більше ніж на 5% нижча точність). Після виконання певного аналізу літературних джерел щодо цієї ситуації, було визначено, що це зумовлено тим, що через свою перевагу послідовного алгоритму, XGBoost є чутливим до шуму, корельованих ознак та специфіки розподілу даних, тоді як алгоритм випадкового лісу, завдяки паралельній побудові незалежних дерев, забезпечує вищу стабільність на шумних або гетерогенних датасетах [31].

Саме через це і було прийнято рішення обрати саме алгоритм випадкового лісу серед описаних покращень, адже він хоч і дає менш точні моделі, однак працює стабільніше.

Далі розглянемо логістичну регресію [32]. Це статистичний метод для бінарної класифікації, що використовує сигмоїдну функцію (тобто повертає число на проміжку від нуля до одиниці) для прогнозування ймовірності належності сукупності характеристик до певного класу. Логіка роботи даної регресії дуже подібна до раніше описаної поліноміальної. При тренуванні моделі підбираються найкращі коефіцієнти для рівняння, що описує залежність між вхідними змінними та ймовірністю належності до певного класу. Модель легко протестувати, адже, так само як і при тестуванні поліноміальної регресії, достатньо зобразити точки на графіку та побудувати криву, що утворюється в процесі тренування.

Крім того, час тренування моделі залежить прямо пропорційно до кількості елементів у наборі даних, тобто $O(n)$. Це робить метод досить ефективним для малих та середніх наборів даних, але з великими обсягами даних час на тренування буде значно збільшуватися.

Тепер розглянемо метод найближчих сусідів. Це непараметричний метод керованого навчання, який використовується для класифікації та регресії і базується на ідеї, що схожі об'єкти знаходяться ближче один до одного на площині ознак [33 – 35]. Під час тренування такого методу не відбувається жодного навчання, оскільки модель не будує якусь явну функцію або рівняння. Цей алгоритм зберігає всі дані, і коли потрібно зробити прогноз, шукає найближчі сусіди за певними ознаками для нового елемента.

Основним недоліком, через який не було використано цей метод, стала потреба в експериментуванні з визначенням числа сусідів, які потрібно перевіряти біля елемента, для ідентифікації класу елемента. Це рішення призвело б до частого повторного тренування моделі для визначення кращого рішення, на відміну від того ж випадкового лісу.

Наостанок варто розглянути найпопулярніший та найпотужніший метод, а саме – нейронні мережі [36 – 38]. Це метод глибокого навчання, що використовує багатошарові штучні нейрони для навчання складних нелінійних функцій, чим дуже подібний до роботи людського мозку. Основна ідея нейронних мереж полягає в тому, що кожен нейрон у мережі отримує вхідні дані, обробляє їх за допомогою певної функції активації і передає результат наступному шару. Навчання зазвичай здійснюється за допомогою алгоритмів оптимізації, таких як градієнтний спуск, які дозволяють поступово коригувати ваги нейронів для зменшення помилки між передбаченнями мережі та фактичними результатами.

Головною перевагою є те, що моделі побудовані за допомогою нейронних мереж зазвичай дуже гнучкі та ефективні для задач, які вимагають вивчення складних структур у великих наборах даних, таких як виявлення об'єктів на зображеннях, автоматичний переклад тексту, прогнозування на основі часових рядів та інші.

Часова складність нейронних мереж зазвичай становить $O(n * m)$, де n – кількість елементів у навчальній вибірці, а m – кількість вхідних ознак. Це означає, що час тренування мережі зростає пропорційно до кількості даних та змінних, що використовуються для навчання. Для складних моделей і великих наборів даних час тренування може бути значним, що робить нейронні мережі вимогливими до обчислювальних ресурсів, що у свою чергу може бути дуже затратним для стартапу.

На додачу, такий метод потребує великої кількості інформації, значно більшої ніж раніше згаданий випадковий ліс, що є проблемою для невеликих проєктів, які ще не мають достатнього набору даних.

У табл. 2.1 наведено підсумок порівняння раніше згаданих методів, які можуть бути використані для реалізації прогнозування даних.

Таблиця 2.1

Порівняння методів прогнозування

Метод	Логістична регресія	Випадковий ліс	Метод найближчих сусідів	Нейронні мережі
Легкий для розуміння	Так	Так	Ні	Ні
Гнучкість	Обмежена лише для лінійних залежностей	Висока, дозволяє працювати з кількома варіантами даних	Дуже висока, легко застосувати до нових даних	Дуже висока, може працювати з будь-якими складними структурами
Стійкість до перенавчання	Високий ризик перенавчання	Стійкий до перенавчання через об'єднання дерев	Високий ризик перенавчання	Може перенавчитися при малих наборах даних
Наобхідна кількість даних	Невелика	Середня	Велика	Дуже велика
Часова складність	$O(n)$	$O(\log n)$	$O(k * n)$	$O(n * m)$

Таким чином, на основі проаналізованих даних про різні методи та алгоритми прогнозування даних, було прийнято рішення щодо реалізації задачі за допомогою методу випадкового лісу. Адже цей метод, на відміну від більшості розглянутих, дозволяє отримувати прогнози у відсотках. Також він здатний забезпечувати високу точність прогнозів без використання великого об'єму даних та значних обчислювальних ресурсів.

2.2 Реалізація алгоритму для тренування моделі прогнозування даних методом випадкового лісу

Як зазначалось раніше, основною метою реалізації є створення моделі, яка на основі історичних даних про взаємодію користувачів зможе автоматично оцінювати ймовірність виникнення потенційних суперечок ще до моменту створення угоди. Таким чином, платформа зможе завчасно попереджати обидві сторони про можливі проблеми та зменшувати кількість конфліктів між користувачами.

Як було згадано раніше, реалізацію моделі виконано за допомогою мови програмування Python, яка є одним із найпоширеніших інструментів у галузі машинного навчання.

Це логічно, адже на додачу до раніше згаданих бібліотек для побудови та тренування виключно моделей прогнозування даних, є такі бібліотеки, як numpy – для виконання чисельних операцій, joblib – для збереження навченої моделі у вигляді .pkl файлу та scіru для використання уже розроблених функцій отримання складних важливих характеристик на кшталт значення імовірності на основі χ^2 тесту [39 – 41]. Саме використання цих інструментів забезпечує швидку реалізацію, високу швидкодію та легку інтеграцію модуля для динамічного навчання моделей у веб-середовища комерційних платформ.

Як згадувалось раніше, щоб забезпечити можливість інтеграції розробленої моделі у повноцінну платформу, алгоритм було реалізовано у вигляді окремого API-сервера, який відповідає за процес тренування та прогнозування. Такий підхід дозволяє звертатися до моделі з основного сервера через HTTP-запити, що спрощує масштабування системи та її подальше оновлення.

Варто зауважити, що на цьому етапі вже було виконано базові заходи для підвищення безпеки API-сервера [42, 43]. Наприклад, було реалізовано базовий механізм авторизації запитів за допомогою API-ключів. Його суть полягала у перевірці кожного запиту до сервера на наявність спеціального

ключу доступу у його body частині. Цей ключ перевіряється перед виконанням будь-якого API-запиту, якщо ключ не відповідає збереженому у системі, сервер повертає повідомлення про помилку доступу.

Після реалізації базової частини розробки, було проведено аналіз архітектури процесу тренування моделі з метою визначення можливих вузьких місць та оптимізації обчислень. Виявилось, що процес тренування моделі є досить ресурсоємним і може займати тривалий час, особливо при збільшенні обсягу даних.

Щоб уникнути затримок у роботі веб-інтерфейсу та забезпечити стабільну взаємодію для адміністраторів, було прийнято рішення виконувати тренування у фоновому режимі, використовуючи багатопоточність. Для цього було реалізовано окремий робочий потік – `random_forest_worker`. Ця функція відповідає за постійне прослуховування черги завдань та виконання відповідних дій після отримання нового елемента. При надходженні команди на тренування моделі запускається процес вибірки даних, їх попередньої обробки та безпосередньо – навчання. Реалізація наведена нижче:

```
def random_forest_worker(task_queue, props):  
    app = create_app()  
    with app.app_context():  
        while True:  
            task = task_queue.get()  
            if task == "EXIT":  
                print("Worker thread finished.")  
                break  
            if task == "Train Random Forest Model":  
                train_random_forest(props['model_id'], task_queue)
```

Після реалізації механізму виклику та управління процесом тренування моделі, наступним етапом стало безпосереднє впровадження алгоритму прогнозування. Як зазначалося раніше, для побудови моделі було обрано метод випадкового лісу – один із найефективніших алгоритмів машинного

навчання, який поєднує високу точність прогнозів і стійкість до перенавчання.

Однак, спершу було дано відповідь на найважливіші питання під час попередньої обробки даних, а саме – чи потребують дані нормалізації та як варто поводитися з нульовими або відсутніми значеннями.

Враховуючи особливості методу випадкового лісу, який не чутливий до масштабу числових ознак, було вирішено не виконувати додаткову нормалізацію даних. Алгоритм самостійно визначає оптимальні порогові значення для розбиття ознак під час побудови дерев рішень, що робить його стійким до різних масштабів вхідних параметрів.

Щодо обробки пропущених значень, то замість видалення неповних записів або глобального заповнення середніми величинами, було реалізовано динамічний механізм підстановки значень за замовчуванням, визначених безпосередньо у структурі моделі адміністраторами. Кожне поле, яке містить нульове значення, обробляється за допомогою створеної функції `get_model_field_value`, що підставляє значення за замовчуванням, яке вказується зі сторони адміністраторів перед безпосереднім тренуванням моделі. Такий підхід гарантує цілісність даних, дозволяє уникнути втрати записів, підвищує стабільність навчання моделі, а також надає адміністраторам гнучкість у роботі з даними, дозволяючи контролювати процес тренування, відстежувати точність прогнозів і оперативно коригувати параметри моделі без втручання у код платформи. Реалізація функції та залежних наведена нижче:

```
def get_field_default_value(field, structure_body):  
    for field_details in structure_body:  
        if field_details.get("pseudonym") == field:  
            return field_details.get("defaultValue")
```

Реалізація алгоритму випадкового лісу представлена у вигляді функції `train_random_forest`, у якій в відбувається повний цикл побудови моделі – від

вибірки даних із бази до збереження натренованої моделі та оновлення інформації у системі.

Функція реалізована з урахуванням інтеграції у вебсередовище, тому процес тренування моделі виконується в межах контексту Flask. Це забезпечує можливість безпосередньої взаємодії з базою даних через ORM, доступ до системних змінних і коректну роботу під час асинхронних викликів [44, 45]. На початковому етапі створюється контекст застосунку, який ініціалізує всі необхідні сервіси та дозволяє керувати станом навчання в реальному часі.

Усередині контексту визначається допоміжна функція `update_progress`, яка фіксує поточний стан виконання навчання у відсотках, що дає змогу адміністраторам платформи у реальному часі спостерігати за перебігом процесу тренування. Такий підхід значно підвищує зручність роботи з моделлю та забезпечує прозорість усіх етапів тренування.

Далі здійснюється завантаження конфігурації поточної моделі з таблиці `DisputePredictionModels`, що містить параметри для тренування – зокрема, перелік вибраних ознак (`selected_fields`), цільову змінну (`check_field`) та структуру даних (`body`). На основі цих параметрів формується SQL-запит, який отримує релевантні записи для подальшої обробки. Лістинг функції генерування запиту вибірки полів для аналізу наведено у Додатку А.

Після виконання запиту дані перетворюються у список словників, що містить лише необхідні поля. Кожне значення проходить попередню обробку через функцію `get_model_field_value`, яка підставляє дефолтні значення у разі відсутності даних, забезпечуючи цілісність вибірки.

Далі дані конвертуються у формат `DataFrame` за допомогою бібліотеки `pandas`, що спрощує подальшу роботу з ними. На наступному етапі дані розділяються на дві вибірки – X (вхідні ознаки, дані про договори) та Y (цільова змінна, тобто чи сталась суперечка за цих даних), а потім поділяються на тренувальну та тестову множини за допомогою функції `train_test_split` із бібліотеки `scikit-learn`. Співвідношення тренувальних і

тестових даних визначається динамічно – воно також зберігається у системних налаштуваннях і може бути змінено адміністратором. Причина такого рішення така ж, як і з механізмом значень за замовчуванням, адміністратори можуть власноруч змінювати значення і аналізувати зміну точності створених моделей.

Оскільки метод випадкового лісу ґрунтується на побудові великої кількості незалежних дерев рішень, кожне з яких формує власне передбачення, то фінальний результат визначається шляхом усереднення прогнозів усіх дерев. На додачу, кожне дерево приймає рішення на основі різних підмножин даних і ознак, що дозволяє моделі уникати перенавчання та краще узагальнювати результати.

На цьому етапі формується процесор попередньої обробки даних, який відповідає за підготовку вхідних ознак до навчання моделі. Для категоріальних параметрів використовується механізм one-hot кодування з ігноруванням невідомих значень, а числові ознаки передаються без змін.

Для кодування категоріальних ознак було обрано саме це, а не label-кодування, оскільки воно є більш універсальним і придатним для використання в більшості задач машинного навчання, забезпечуючи коректну інтерпретацію категоріальних даних незалежно від конкретного датасету.

Після створення процесору, будується об'єкт моделі `RandomForestClassifier`, параметри якого, такі як кількість дерев (`n_estimators`) та стан генератора випадкових чисел (`random_state`), задаються динамічно через параметри, якими можуть керувати адміністратори.

Після цього починається процес тренування моделі, а потім виконується перевірка точності прогнозів на тестовій вибірці за допомогою методу `assurasy_score`, а отриманий відсоток точності записується у базу даних для подальшого моніторингу результатів. Далі навчена модель серіалізується за допомогою бібліотеки `joblib` та зберігається у вигляді `.pkl` файлу в директорії, що відповідає ідентифікатору моделі. Такий формат

дозволяє швидко завантажувати модель без повторного тренування і використовується безпосередньо під час виклику API для отримання прогнозу щодо ризику.

Після завершення основного процесу тренування може автоматично запустити додатковий процес перепрогнозування усіх наявних ордерів у системі, за умови, що адміністратор обрав цю опцію перед тренуванням моделі. Даний механізм потрібен для того, щоб оновити ймовірності ризику за вже існуючими записами, використовуючи новостворену, більш точну модель. Запуск перепрогнозу виконується у тому ж робочому середовищі, але в окремому потоці і починається із виклику функції `reevaluation_worker`, завдяки передачі завдання "Revaluation Model Started" у чергу.

Також у адміністраторів передбачена можливість керування активністю моделей – після завершення навчання вони можуть самостійно вирішити, чи зробити новостворену модель активною для прогнозування нових замовлень, чи залишити поточну версію без змін. Це забезпечує гнучкість та контроль над процесом оновлення системи, адже не кожна нова модель обов'язково має ставати робочою одразу після тренування, адже адміністратори можуть спершу перевірити точність та стабільність нової моделі, порівняти результати з попередніми версіями, а вже потім прийняти рішення щодо її активації. Таким чином, було реалізовано механізм, який дозволяє уникнути ризику некоректних прогнозів у випадку низької точності моделі.

Лістинг основної функції тренування моделі наведено у Додатку Б.

2.3 Реалізація механізму прогнозування даних за допомогою натренованої моделі

Після завершення тренування моделі її можна використовувати для швидкого отримання прогнозів. Для цього реалізовано окремий API-ендпоінт. У випадку демонстративної платформи це `/api/predict-dispute`, який

дозволяє за ідентифікатором конкретного замовлення, або його чернетки, майже миттєво отримувати оцінку ймовірності виникнення суперечки.

Варто зауважити, що у проєкті було створено допоміжну таблицю `temp_orders`, у якій зберігаються деталі чернеток замовлень до того, як вони потраплять у основну таблицю `orders` і стануть доступними ключовим учасникам угоди. Така архітектура дозволяє виконувати попередню оцінку ризику ще до повноцінного створення замовлення. Крім того, використання таблиці `temp_orders` позитивно впливає на продуктивність і масштабованість системи, адже, це дає змогу розвантажити основну таблицю, оскільки тестові або проміжні дані не потрапляють до таблиці `orders`, де виконуються бізнес-критичні операції. Також варто зауважити, що таблиця `temp_orders` регулярно очищується, що запобігає накопиченню зайвої інформації та забезпечує стабільну роботу бази даних. І на додачу, такий підхід забезпечує гнучкість у налаштуванні алгоритму прогнозування, адже адміністратори можуть експериментувати з даними, оптимізуючи параметри моделі без ризику порушення цілісності основних записів.

Логіка ендпоінта починається із формування динамічного SQL-запиту, який отримує необхідні параметри. Це дає змогу сформувати вхідний набір ознак, який далі передається до функції `get_dispute_probability`, у якій здійснюється підготовка даних для прогнозу, а саме – вибираються всі поля, що були визначені під час навчання моделі, а далі формується вектор ознак для передачі в модель. Для кожного відсутнього значення система автоматично підставляє визначене адміністратором значення, що гарантує коректність прогнозу навіть за неповних даних.

Наступним етапом з файлової системи завантажується актуальна навчена модель `Random Forest`, збережена у форматі `.pkl`, після чого виконується обчислення прогнозу та ймовірності виникнення суперечки.

Завдяки автоматичному використанню поточної актуальної моделі, а не через передачу її ідентифікатора у запиті з основного сервера, система залишається незалежною від логіки керування версіями на бекенді та

гарантує, що всі прогнози виконуються виключно на основі останньої затвердженої моделі. Такий підхід дозволяє уникнути помилок, пов'язаних із використанням застарілих чи несумісних моделей, зменшує кількість параметрів у запиті та спрощує інтеграцію.

Крім того, це підвищує надійність архітектури, адже оновлення моделі відбувається шляхом створення нового файлу у файловій системі, після чого всі майбутні запити автоматично починають використовувати оновлений алгоритм без необхідності модифікувати серверний код або конфігурацію API.

Після обчислення прогнозованого значення, а також супутньої інформації про це значення, воно повертається до основного сервера у вигляді JSON-відповіді.

Також було враховано, що в реальних умовах повна відсутність чи абсолютна впевненість у виникненні суперечки є статистично малоймовірними сценаріями, тому у моделі передбачено додаткові обмеження, а саме – ймовірність ризику не може перевищувати 99% та не може дорівнювати 0%.

Лістинг основної функції, яка на основі даних угоди повертає ймовірність суперечки, наведено у Додатку В.

2.4 Алгоритм визначення релевантних параметрів для прогнозування суперечок

Процес аналізу залежностей між параметрами, які можуть впливати на результат оцінки ймовірності виникнення суперечки, є громіздким і недоступним для недосвідченої людини, тому для автоматизації цього етапу було реалізовано функціонал рекомендації параметрів, від яких є сильна залежність обраної ознаки.

Це рішення значно спрощує вибір ключових параметрів для подальшого навчання моделі, знижує залежність моделей від людських

факторів, підвищує ефективність та точність прогнозів, а також дає змогу швидко адаптувати модель до нових умов та даних, адже автоматично враховується специфіка даних. Крім того, ці рекомендації супроводжуються графіками, на яких візуально зображено причини результатів, що дозволяє адміністраторам краще зрозуміти логіку, аналізувати взаємозв'язки між параметрами та робити більш обґрунтовані рішення щодо подальшої оптимізації моделі.

Щоб реалізувати таке рішення, необхідно було дослідити наявні алгоритми, завдяки яким можна визначити залежність між ознаками. Першим із них стала матриця кореляції. Вона дозволяє виявити сильні залежності між числовими ознаками та результатом і виділити найбільш залежні параметри. Ще однією перевагою цього алгоритму є те, що можна виявити як прямі, так і зворотні залежності, це допомагає краще зрозуміти, як різні ознаки взаємодіють між собою.

Після визначення алгоритму, який дозволяє аналізувати залежності між числовими та категоріальними ознаками, постала потреба у методі для оцінки взаємозв'язку між двома категоріальними змінними. Таким методом став χ^2 тест, адже він дозволяє оцінити, наскільки спостережуваний розподіл частот відрізняється від очікуваного, що допомагає визначити, які категорії мають суттєвий вплив і заслуговують на подальший аналіз.

Після визначення ключових алгоритмів для аналізу залежностей між ознаками було розпочато етап розробки. Виклик відповідного функціоналу реалізовано в API-ендпоінті `/api/predict-training`. Спочатку функція отримує інформацію про поля, які планують використати для тренування моделі. Далі викликається функція `model_dependencies`, яка повертає інформацію про залежності між полями.

Усередині функції `model_dependencies` спочатку створюється відповідна директорія для графіків, що будуть генеруватися для кожного поля. Далі функція виконує запит на отримання необхідних даних для

тренування. Результати запиту конвертуються в об'єкт `DataFrame` за допомогою раніше згаданої бібліотеки `pandas`.

Наступним етапом відбувається обробка числових ознак за допомогою `MinMaxScaler` з бібліотеки `sklearn`, що дозволяє нормалізувати значення в межах заданого діапазону. Після цього розраховується кореляційна матриця для числових змінних, що дозволяє виявити сильні залежності між ознаками та результатом. Для кожної числової ознаки також генерується візуалізація у вигляді скрипкової діаграми, що надає зрозуміле уявлення про розподіл значень і їх взаємозв'язок з результатом.

Після отримання інформації про числові ознаки, функція аналізує категоріальні ознаки. Спочатку вони групуються, після чого проводиться обчислення відносних частот для кожної категорії. Далі, для оцінки статистичної значущості категоріальних змінних виконується χ^2 тест за допомогою функції `chi2_contingency` з бібліотеки `scipy.stats`. Після отримання результату відбувається візуалізація даних завдяки стовпчиковому типу графіків.

Повний лістинг функції, яка визначає рівень залежності між ознаками моделі та ймовірністю виникнення суперечки, наведено в Додатку Г.

Після отримання інформації про кореляційну матрицю та результати χ^2 тесту відбувається порівняння значень кожної з ознак з установленими пороговими значеннями. Якщо кореляція між числовою ознакою та результатом перевищує встановлений поріг, ця ознака додається до списку рекомендованих. Аналогічно, якщо р-значення для категоріальної ознаки менше порогового значення, така ознака також включається до списку важливих для моделювання. Такий підхід дозволяє автоматично відсіяти ознаки з низьким впливом на прогнозування ймовірності виникнення суперечки.

Таким чином, на цьому етапі важливо було визначити які порогові значення краще використовувати для кореляційної матриці та χ^2 тесту, які б

забезпечували коректне виявлення залежностей і були придатними для застосування в якомога ширшому спектрі проєктів.

Для цього було проведено невеликий експеримент із використанням описаних у пункті 2.1 датасетів. Його результати подано у табл. 2.2.

Таблиця 2.2

Залежність точності моделей від порогових значень

Датасет	cor = 0.3 p_value = 0.1	cor = 0.5 p_value = 0.08	cor = 0.6 p_value = 0.05	cor = 0.8 p_value = 0.01
Шахрайські оголошення	94.29	95.71	95.71	95.71
Вдалість інвестицій	60	60	65.26	65.98
Затримка авіарейсів	61.31	65.24	68.57	68.59

Як можна побачити, точність збільшувалась значно при порогових значеннях коефіцієнту кореляції рівному 0.6, а ймовірності випадковості – 0.05. Також варто зауважити, що при більш жорстких значеннях спостерігалось незначне додаткове зростання точності, однак кількість відібраних ознак у тестових датасетах суттєво зменшувалася. Це, у свою чергу, могло призвести до значної ймовірності відсутності рекомендованих параметрів під час застосування інтеграції модуля до проєктів замовників.

2.5 Опис бази даних демонстративного застосунку та визначення можливих параметрів для тренування моделі прогнозування ризику виникнення суперечки

Для коректної роботи з даними, демонстративна платформа містить 28 основні таблиці та дві системні для коректної роботи міграцій та сидерів. На

Наступною важливою таблицею є оголошення (listings). У ній міститься детальна інформація про предмети, що можна орендувати на платформі. До ключових полів належать: ідентифікатор (id), назва предмету (name), який здають в оренду, його опис (description), координати пункту (lat, lng), де користувач зможе забрати орендований предмет, відстань (radius), на яку власник може відійти від пункту видачі, щоби зменшити замовнику шлях, ідентифікатор власника предмету (owner_id), ціна за день оренди (price) та ідентифікатор категорії (category_id), до якої належить річ. Для прогнозування виникнення суперечок важливими можуть бути такі поля: категорія, ціна та ідентифікатор власника, оскільки, як згадувалось раніше, на основі ідентифікаторів можна отримати додаткову інформацію з інших таблиць, наприклад, про відгуки власника.

Далі розглянемо основну таблицю проєкту, від якої можна побудувати зв'язки ледь не з кожною іншою, а саме – замовлення (orders). У цій таблиці важливими є такі поля: звичайний статус (status), статус скасування замовлення (cancel_status), комісії користувачів (renter_fee, owner_fee), ідентифікатор замовника (renter_id), ідентифікатор оголошення (listing_id), ціна оренди за день (price), яку погодили користувачі, дати початку (start_time) та кінця замовлень (finish_time). З цієї таблиці можна отримати наступні важливі параметри для моделі прогнозування даних: статус на якому виникла суперечка, податок орендарю, податок замовнику на момент оренди, тривалість замовлення, а також раніше згадані поля для під'єднання інших таблиць - ідентифікатори оголошення та замовника.

Також на платформі є можливість коригування ціни та періоду замовлення. Для цього існує окрема таблиця запитів оновлення замовлень (order_update_requests). Завдяки ній, у модель прогнозування даних можна додати наступні параметри: різниця цін, різниця між початковою та остаточною тривалістю, а також скільки разів взагалі було висунуто пропозиції опонентам.

Для визначення результату потрібно отримувати інформацію про те, чи закінчилося замовлення успішно чи суперечкою, що можна отримати за допомогою приєднання відповідної таблиці до замовлення.

Таблицями, на основі яких будуються рейтинги користувачів, є коментарі власників (`owner_comments`) та коментарі орендарів (`renter_comments`). Вони дуже схожі загальною структурою, містять описи (`description`), ідентифікатори (`id`), особисті відгуки від опонента (`leave_feedback`), підтвердження адміністраторів (`approved`), час створення (`created_at`), ідентифікатор замовлення, на яке залишили відгук (`order_id`) а також числові поля для оцінки кожної характеристики замовника чи власника. Для моделі тренування даних важливими можуть бути наступні значення, наприклад середня загальна оцінка користувача на момент початку замовлення, середня оцінка користувача у якості замовника чи власника, або ж середня оцінка за кожною з характеристик.

Також у платформі реалізовано механізм чатів, за дані якого відповідають таблиці чатів (`chats`), зв'язків з користувачами (`chat_relations`), повідомлень (`chat_messages`) та контентів повідомлень (`chat_message_contents`). У подальшому, за потреби, можливо буде реалізувати ще один модуль прогнозування даних на основі великих датасетів з використанням нейронних мереж, для якого можна буде використати дані з цих чатів для тренування моделі та прогнозування виникнення суперечок.

Таким чином, після опису структури бази даних проєкту стало зрозуміло, які числові та категоріальні параметри можуть бути використані для тренування моделі. До них належать: середні оцінки замовника та власника, ціна за день оренди оголошення, кількість успішно виконаних замовлень власником, кількість успішно завершених оренд замовником, кількість внесених оновлень до поточного замовлення, перевірка чи замовник є власником інших речей, чи власник користувався платформою для оренди речей, а також – поточна запропонована тривалість оренди речі.

Простими параметрами з цього списку є категорія речі та її ціна за день оренди. Ці параметри є простими, адже їх можна отримати прямим запитом вибірки конкретного поля з конкретної таблиці.

Інший тип параметрів – це складені. Вони визначаються на основі певних формул та комбінації з іншими табличними значеннями. До них належать усі інші згадані параметри. При перетворенні у SQL-запит, вони вираховують за допомогою підзапитів, для зручності керування якими варто додавати псевдоніми полям та під'єднанням таблицям.

Для визначення тривалості замовлення, потрібно отримати різницю між початком та кінцем замовлення у днях, для чого в PostgreSQL існує відповідна функція.

Кількість виконаних та завершених замовлень, а також кількість створених пропозицій для поточного замовлення, варто також визначити за допомогою використання відповідної вбудованої функції, однак варто також додати умову, що дата створення кожного з розглянутих записів повинна бути меншою за дату створення поточного замовлення.

Для визначення наявності замовлень у власника, де він є замовником, потрібно виконати просте під'єднання відповідної таблиці, де користувач має відповідну роль і перевірити чи отримана кількість більше нуля. Для іншого схожого параметру, а саме – визначення кількості виконаних замовлень замовником – потрібно виконати під'єднання оголошень, власником яких є замовник, а потім підключити до них замовлення і також перевірити чи знайдена кількість більша за нуль.

Середні значення оцінок користувачів, у поточній ролі, на момент створення замовлення, можна вирахувати на основі усередненого середнього значення усіх критеріїв серед усіх відгуків до дати створення поточного замовлення.

На завершення, задля визначення того, чи була створена суперечка для замовлення потрібно виконати просте під'єднання відповідної таблиці до

таблиці замовлень і перевірити, чи кількість таких елементів є більшою за нуль.

Також, варто зауважити, що при інтеграції розробленого модуля дана структура бази даних трохи зміниться. Для коректної роботи модуля та зручної взаємодії з моделями передбачення суперечок буде створено таблицю, що відповідає за їх архів (`dispute_prediction_models`). Вона дозволить перемикатися між натренованими моделями, на відповідній панелі керування. Такий підхід дозволив більше експериментувати з різними полями, що передаються на сервер з прогнозуванням, і уникати ситуацій, за яких повертатиметься неправильна інформація. Гарним наприклад користі цього функціоналу можуть слугувати випадки, коли нова модель має низьку точність. У таких ситуаціях користувачі зможуть переключитися на стару модель, що дозволить зберегти контроль над ситуацією.

Висновки до другого розділу

У цьому розділі було проведено огляд і аналіз сучасних методів машинного навчання, що надало можливість обґрунтовано обрати оптимальний алгоритм для досягнення поставленої мети. У результаті аналізу було визначено, що найкращим методом для виконання поточного завдання проєкту є метод випадкового лісу, адже він поєднує високу точність прогнозів, стійкість до перенавчання та здатність ефективно працювати без значних обчислювальних витрат.

Реалізація алгоритму була здійснена засобами мови програмування Python з використанням бібліотек `scikit-learn`, `pandas`, `numpy` та `joblib`, що забезпечило ефективну та надійну побудову моделі, її навчання та подальшу інтеграцію у вебсередовище. Для взаємодії з комерційними платформами розроблено API-сервер за допомогою фреймворку `Flask`, який підтримує багатопоточність і дозволяє виконувати процес тренування моделей у

фоновому режимі без навантаження на головний застосунок, що підвищило стабільність роботи сервера модуля та комфорт користувачів.

Також у межах розділу реалізовано систему автоматичного визначення релевантних параметрів для моделі, яка ґрунтується на аналізі кореляційної матриці та χ^2 тесту. Також було експериментально визначено оптимальні порогові значення для цих перевірок. Цей механізм спростив процес вибору даних, що зробило його більш об'єктивним і забезпечило адаптивність алгоритму до зміни вхідних даних. Завдяки цьому функціоналу модуль може не лише прогнозувати ризики, а й самонавчатися через вдосконалення вхідних параметрів на основі статистичних закономірностей.

Таким чином, у результаті виконання другого розділу було створено гнучкий, масштабований і безпечний модуль прогнозування, який може інтегруватися в інші платформи та слугувати основою для подальших розробок у напрямі інтелектуальної аналітики ризиків.

РОЗДІЛ 3 ІНТЕГРАЦІЯ ІНТЕЛЕКТУАЛЬНОГО МОДУЛЯ У ДЕМОНСТРАТИВНУ ПЛАТФОРМУ

3.1 Опис демонстративного застосунку

Після того, як було виконано основну мету проєкту, а саме було розроблено інтелектуальний модуль для динамічного тренування моделей і механізм рекомендації параметрів для комерційних платформ, було виконано розробку демонстративної платформи. Дана платформа мала на меті дозволити користувачам безпечно орендувати та здавати в оренду різні елементи одягу, що стало ідеальною задачею для використання розробленого модуля. Адже завдяки ньому можна швидко створювати структуру та виконувати тренування моделі для визначення ймовірності виникнення суперечок, забезпечуючи вищий рівень безпеки та впевненості користувачів, будь-ким, у будь-який час та з будь-якого місця.

Розроблена платформа має модулі авторизації та реєстрації, адже завдяки ним користувачі отримують можливість створювати оголошення, приймати їх, отримувати винагороди та створювати суперечки. Для зручності взаємодії з системою реалізовано можливість входу в акаунт як через електронну пошту, так і за допомогою сторонніх сервісів Google та Facebook.

На рис. 3.1 зображено основну сторінку платформи з активною формою авторизації.

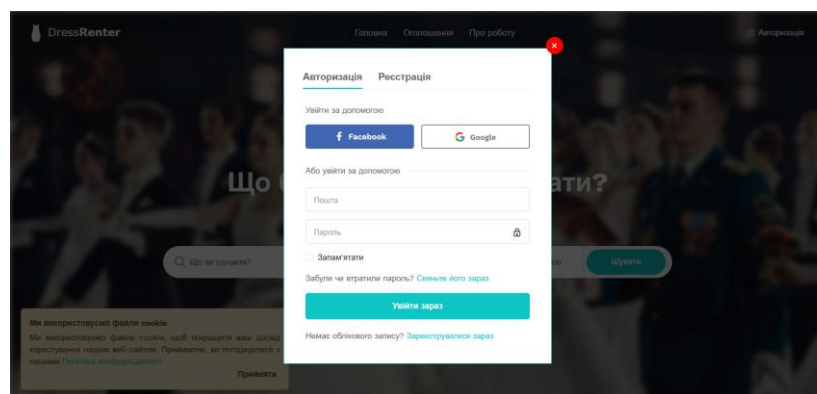
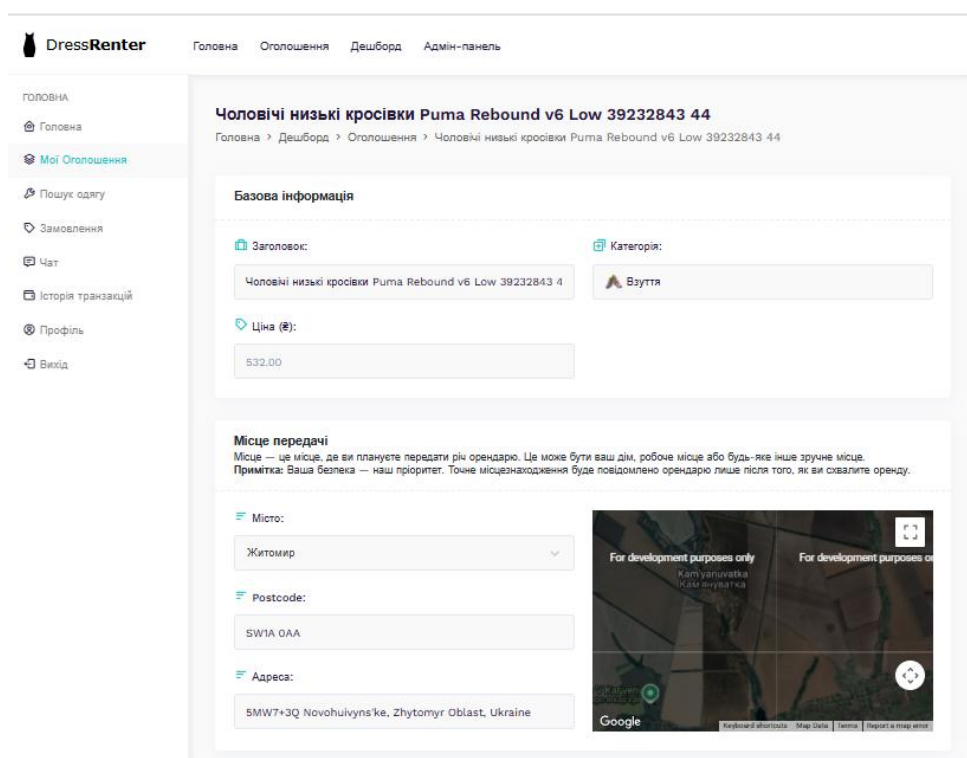


Рисунок 3.1 – Форма авторизації на фоні основної сторінки

Також на платформі реалізовано модуль оголошень, які стали ключовим елементом взаємодії користувачів із платформою. Кожен лістинг містить детальну інформацію про предмет, що хочуть здати в оренду, а саме – назву, опис, адресу, ціну, координати зустрічі для передачі, категорію та фотографії. Для створення та редагування оголошень на платформі реалізовано зручну форму, яка містить поля для базової інформації, категорії, місця розташування через інтеграцію з Google Maps, а також секцію для завантаження фотографій.

На рис. 3.2 зображено сторінку створення/редагування оголошення.



The screenshot displays the 'DressRenter' web application interface for creating or editing a listing. The top navigation bar includes links for 'Головна' (Home), 'Оголошення' (Listings), 'Дешборд' (Dashboard), and 'Адмін-панель' (Admin Panel). A left sidebar contains a menu with options like 'Головна', 'Міє Оголошення', 'Пошук одягу', 'Замовлення', 'Чат', 'Історія транзакцій', 'Профіль', and 'Вихід'. The main content area is titled 'Чоловічі низькі кросівки Puma Rebound v6 Low 39232843 44'. Below the title, there's a breadcrumb trail: 'Головна > Дешборд > Оголошення > Чоловічі низькі кросівки Puma Rebound v6 Low 39232843 44'. The form is divided into sections: 'Базова інформація' (Basic Information) with fields for 'Заголовок:' (Title), 'Категорія:' (Category), and 'Ціна (€):' (Price); 'Місце передачі' (Delivery Location) with fields for 'Місто:' (City), 'Postcode:', and 'Адреса:' (Address); and a map section showing a Google Map of the location. The map area also contains text: 'For development purposes only' and 'Kam yanyavka, Kam yanyavka'.

Рисунок 3.2 – Сторінка створення та редагування оголошення

Після створення оголошення та підтвердження його публікації адміністраторами платформи, інші користувачі можуть швидко знайти даний предмет. Для зручності користувачів на сторінці пошуку є фільтри за категоріями, містами та цінами. Крім того, на ній є інтерактивна карта, що дозволяє легко визначити, в якому районі користувач може отримати замовлений предмет, а також зрозуміти, чи підходить місце зустрічі замовнику. На рис. 3.3 зображено сторінку пошуку речей.

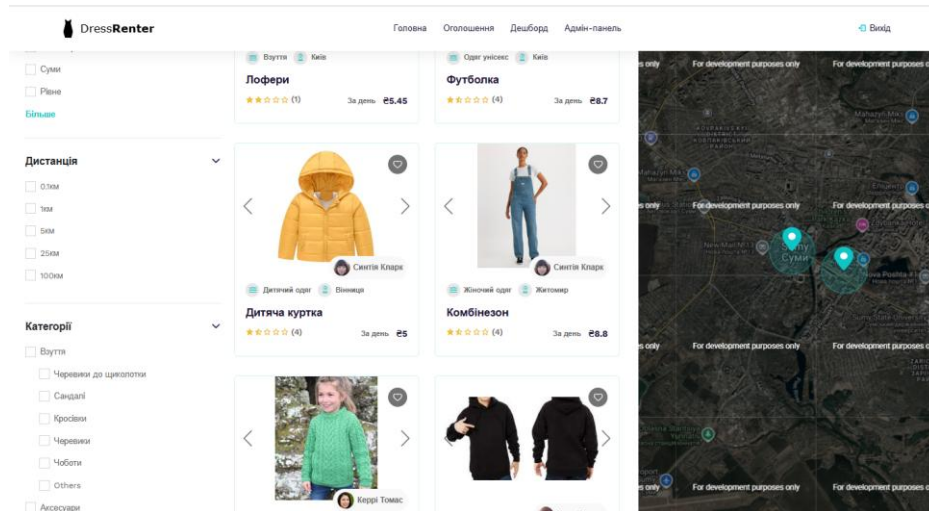


Рисунок 3.3 – Сторінка пошуку речей з фільтрами

На самій сторінці оголошення, користувачі можуть детальніше переглянути пропозицію, її опис та світлини. Це дозволяє їм ознайомитись з усіма деталями, які можуть бути важливими для прийняття рішення щодо оренди. Окрім цього, сторінка містить додаткові відгуки попередніх орендарів, а також функції для зручного зв'язку з автором оголошення.

На рис. 3.4 представлено сторінку перегляду оголошення.

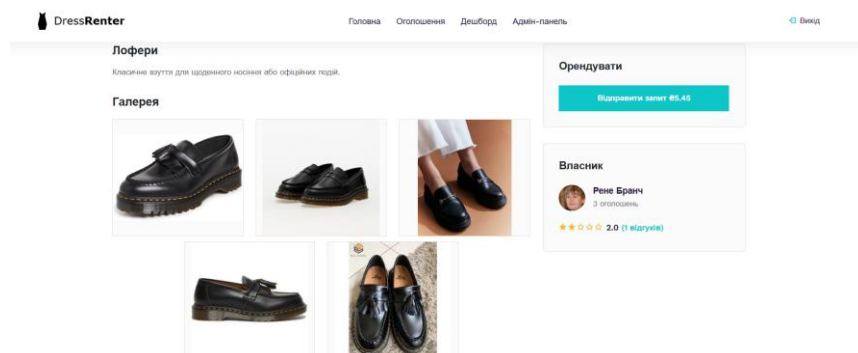


Рисунок 3.4 – Сторінка перегляду оголошення

Також на цій сторінці є можливість оренди предмета, що розглядається на дати, які підходять користувачу. Це можна зробити за допомогою календаря, який з'являється при натисканні кнопки початку оренди. Під час створення нової угоди система автоматично перевіряє поточні виконувані замовлення з обраною річчю і не дає користувачу створити замовлення на уже вибрані дати, що запобігає виникненню невалідних або дубльованих операцій.

На рис. 3.5 зображено розроблену форму створення замовлення.

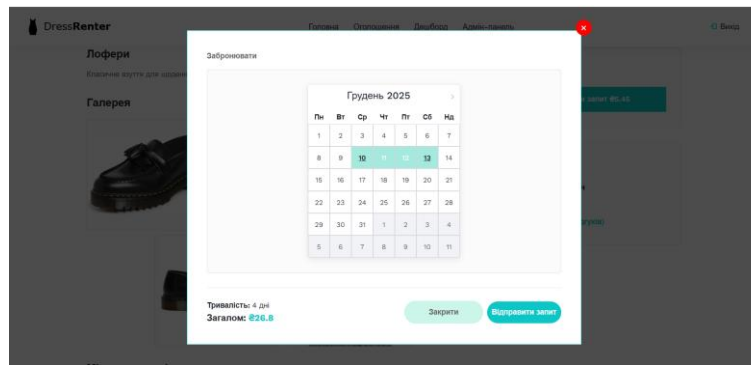


Рисунок 3.5 – Створення замовлення

Після створення запиту на оренду, власник отримує повідомлення про нього. На цьому етапі опоненти можуть комунікувати один з одним, пропонувати інші дати, ціни та умови і закріплювати зміни за допомогою зворотних запитів підтвердження угоди.

На рис. 3.6 зображено сторінку перегляду замовлення та його деталей.

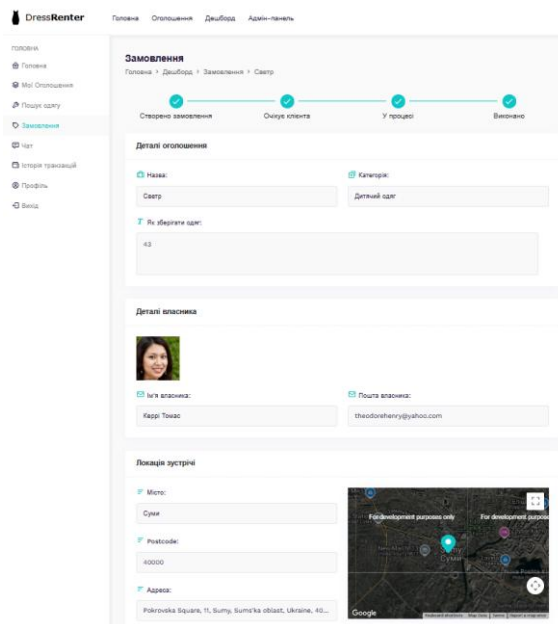


Рисунок 3.6 – Сторінка перегляду замовлення

Загалом, замовлення має 8 основних статусів та 1 додатковий. До основних належать статуси “Очікує підтвердження власника”, “Очікує підтвердження замовника”, що відповідають за етапи узгодження, “Очікує оплати” – відповідає за усі процеси виконання оплати, її звичайного та повторного підтвердження. Статус “В процесі” означає, що предмет було

передано замовнику, а “Очікує підтвердження завершення” та “Завершено” – вказують, що було розпочато процес завершення. У разі відхилення або дострокового скасування угоди застосовуються статуси “Відхилено” та “Скасовано”.

Особливим для замовлень є статус “Суперечка”, адже якщо він виникає, то будь-які зміни в замовленні користувачами відхиляються до моменту, коли адміністратори її вирішать та або повернуть кошти замовнику, або переведуть їх на рахунок власника.

Оплату на платформі можна виконати у два способи – за допомогою кредитної картки, а також за прямим банківським переказом коштів, однак для цього користувачі повинні вручну надати доказ виконання оплати, а адміністратори мають перевірити та підтвердити цей платіж.

На рис. 3.7 зображено розроблену сторінку прикріплення підтвердження виконаного переказу коштів користувачем.

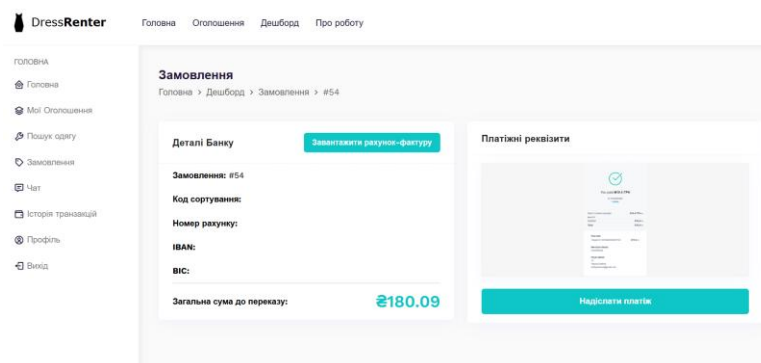


Рисунок 3.7 – Сторінка підтвердження виконання переказу коштів користувачем

Після виконання платежів та отримання коштів платформа фіксує цю інформацію в базі даних і користувачі можуть переглядати її на спеціальній сторінці історії транзакцій, зображеній на рис. 3.8.

Це дуже зручно, оскільки користувачі можуть забути про виконану операцію або очікувані кошти, а також можуть спробувати звинуватити платформу в привласненні їхніх коштів, або захочуть краще прогнозувати

свій прибуток за місяць. У такому випадку наявність повної історії оплат стає надзвичайно корисною.

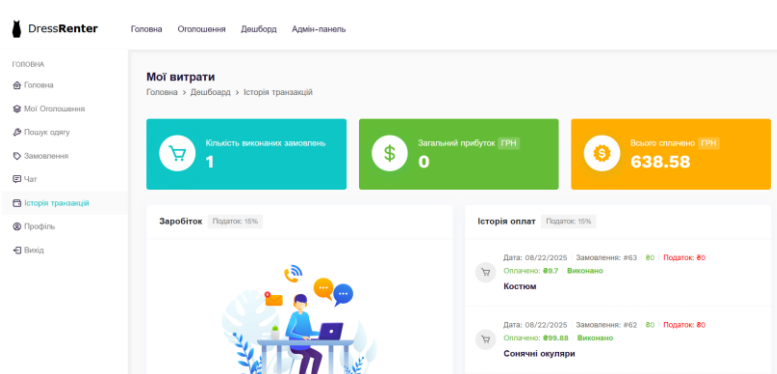


Рисунок 3.8 – Сторінка історії транзакцій у фінансовому модулі платформи

Як згадувалось раніше, користувачі можуть комунікувати між собою на різних стадіях замовлення. Для цього було реалізовано внутрішній чат, що тісно інтегрований з модулем угод – усі зміни статусів, оновлення пропозицій, підтвердження або відхилення запитів відображаються в реальному часі за допомогою WebSocket-з'єднання. Завдяки цьому користувачі миттєво отримують повідомлення про перебіг угоди, що створює відчуття реальної взаємодії та спрощує процес узгодження. Крім того, чат підтримує надсилання системних повідомлень про ключові події, наприклад – створення, оплату, завершення або скасування угоди, що підвищує зручність використання платформи та забезпечує прозору комунікацію на всіх етапах оренди. На рис. 3.9 зображено функціонал чату демонстративної платформи.

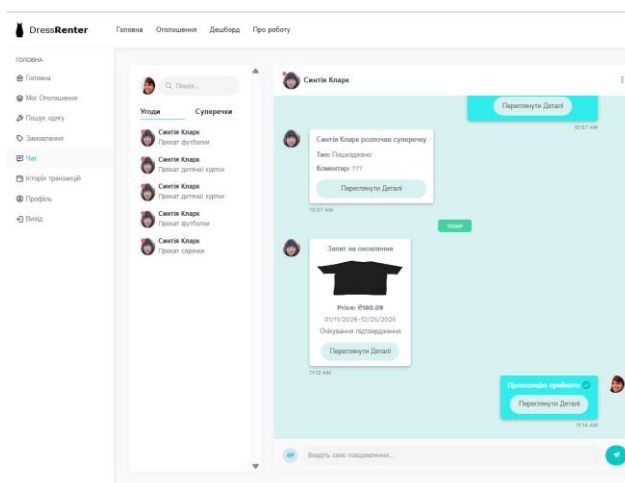


Рисунок 3.9 – Демонстрація комунікації між користувачами на платформі

Також користувачі можуть комунікувати і зі службою підтримки. Завдяки цьому типу чатів користувачі можуть отримати допомогу з технічними питаннями, уточненням деталей послуг, а також вирішенням проблем із оплатою або місцем зустрічі. Наявність такого функціоналу забезпечує користувачам відчуття підтримки та безпеки, що підвищує рівень їхньої довіри до сервісу та сприяє збереженню лояльності.

Приклад такої комунікації зображено на рис. 3.10.

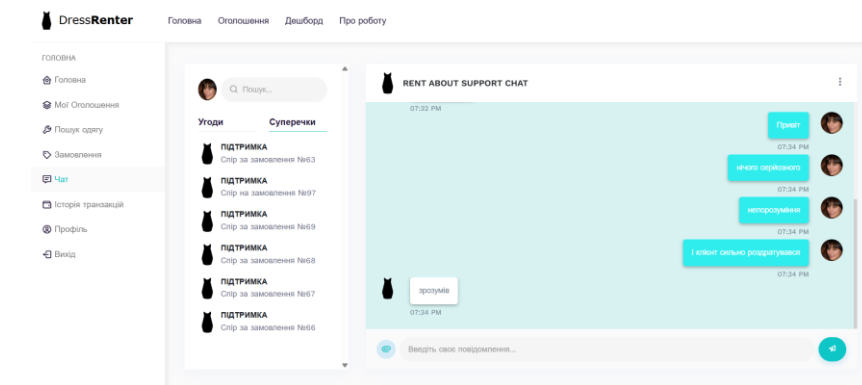


Рисунок 3.10 – Демонстрація комунікації між користувачем та службою підтримки під час суперечки

Важливо зауважити, що користувачі можуть створювати суперечки лише після успішної оплати. Така умова зумовлена потребою у витрат ресурсів компанії, що є власником платформи, для виконання лише серйозних завдань, що їм приносять користь та фінанси. Для самого створення суперечки користувачі можуть натиснути на відповідну кнопку як зі списку замовлень, так і з самого замовлення або чату. Після цього користувачів перекидає на відповідну форму зображену на рис. 3.11.

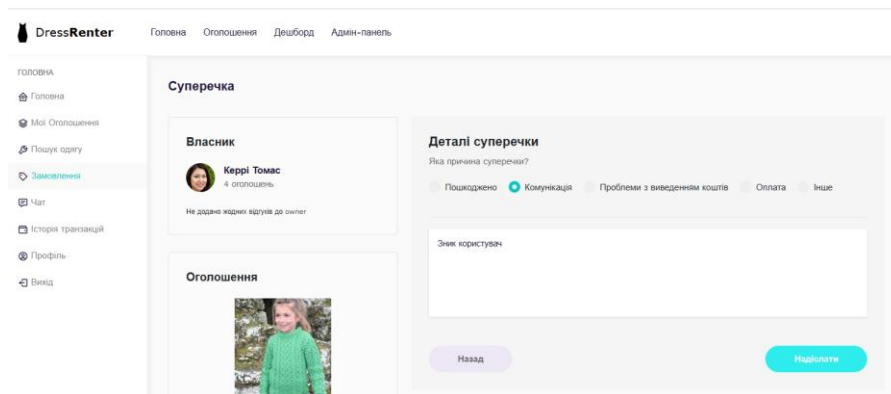


Рисунок 3.11 – Створення суперечки користувачем

Під час створення суперечки користувач повинен чітко визначити основну причину суперечки та вказати її в переліку доступних варіантів, а також описати свою думку щодо ситуації та очікувані дії від адміністраторів платформи у відповідному полі.

Наступними важливими структурами на платформі є відгуки. Під час їх розробки було реалізовано розширену багатофакторну модель оцінювання, яка дозволяє формувати більш точне уявлення про якість взаємодії між користувачами. Для власників передбачено вісім критеріїв оцінки, серед яких – точність опису речі, відповідність фото, стан, чистота, швидкість реагування, зрозумілість комунікації, гнучкість у плануванні та вміння вирішувати спірні ситуації.

Дану реалізацію зображено на рис. 3.12.

DressRenter Головна Оголошення Дешборд Про роботу

голова
Головна
Мої Оголошення
Пошук одягу
Замовлення
Чат
Історія транзакцій
Профіль
Вихід

Власник
Синтія Кларк
4 оголошень
★☆☆☆ 1.6 (4 відгуки)

Додати публічний відгук
Вашу адресу електронної пошти не буде опубліковано. Обов'язкові поля позначені *

Точність опису товару * Наскільки точно товар відповідає опису, наданому в оголошення?	★★★★★
Точність фотографій * Наскільки точними були фотографії товару?	★★★★★
Стан при отриманні * Як ви оцінюєте стан товару при отриманні?	★★★★★
Чистота * Як ви оцінюєте чистоту товару при отриманні?	★★★★★
Чутливість * Як швидко власник відповів на повідомлення та запити?	★★★★★
Чіткість * Наскільки чіткими та корисними, на вашу думку, були інструкції та відповіді в асінга?	★★★★★
Гнучкість у плануванні * Як ви оцінюєте гнучкість власника щодо часу отримання та повернення?	★★★★★
Вирішення проблеми * Наскільки ефективно власник вирішив будь-які проблеми чи заперечення?	★★★★★

Рисунок 3.12 – Форма оцінки власника предмета

У свою чергу для орендарів створено дев'ять параметрів: дбайливість у користуванні, пунктуальність, швидкість відповіді, зрозумілість спілкування, дотримання правил використання, повага до умов оренди, чесність, надійність та загальний рівень задоволеності. Усі ці показники обробляються на рівні сервера платформи, де система автоматично обчислює середні значення за кожним параметром та загальний рейтинг користувача, що дозволяє відображати реальну репутацію при перегляді деталей про користувачів.

Описаний перелік зображено на рис. 3.13.

Замовник

Укстер
0 оцінок

Немає наданих відгуків до цього замовника

Додати публічний відгук
Вашу адресу електронної пошти не буде опубліковано. Обов'язкові поля позначені *

Догма *
Наскільки ретельно, на вашу думку, орендар поводився з товаром і повернув його в тому ж стані? ★ ★ ★ ★ ★

Специфічність *
Як ви оцінюєте орендаря стосовно періоду повернення? ★ ★ ★ ★ ★

Чувствителі *
Як оцінюєте орендаря відносно надання інструкцій та запиту? ★ ★ ★ ★ ★

Чіткість *
Наскільки зрозумілими та корисними, на вашу думку, були інструкції та відповіді власника? ★ ★ ★ ★ ★

Інструкції з використання *
Наскільки добре, на вашу думку, власник дотримувалася наданих інструкцій з використання? ★ ★ ★ ★ ★

Умови надання послуг *
Наскільки зрозумілими та корисними, на вашу думку, були інструкції та відповіді власника були? ★ ★ ★ ★ ★

Чесність *
Наскільки чесно був орендар щодо стану товару після повернення? ★ ★ ★ ★ ★

Надійність *
Як ви оцінюєте надійність орендаря з точки зору отримання, використання та повернення товару? ★ ★ ★ ★ ★

Задовolenість *
Наскільки ви були задоволені загальним досвідом, наданим орендарем? ★ ★ ★ ★ ★

Рисунок 3.13 – Форма оцінки замовника

3.2 Опис адміністративної панелі демонстративного застосунку

Оскільки у попередньому пункті було описано основний функціонал застосунку, то варто зацентувати увагу на не менш важливій частині – а саме секція для користувачів із додатковими правами. Ця частина системи відповідає за контроль та перевірку процесів, що відбуваються на платформі. Зокрема, адміністраторам надано можливість перегляду та керування оголошеннями, угодами, суперечками, відгуками, а також чатами.

На рис. 3.14 зображено сторінку менеджменту замовлень.

Оренди

Активних 1 Завершено 52 Скасовано 0 У суперечці 47 **Усього 100**

Шукати за ідентифікатором Минулого року

Усього замовлень 100

ID	ТОВАР	ПОЧАТОК ЗАМОВЛЕННЯ	КІНЕЦЬ ЗАМОВЛЕННЯ	ЗАГАЛЬНА СУМА	СТАТУС
#100	Слідиця	09/02/2025	11/13/2025	\$8.69	Завершено
#99	Кільце	08/26/2025	08/26/2025	\$5.1	В процесі
#98	Костюм	09/03/2025	09/09/2025	\$7.1	Завершено
#97	Сонячні окуляри	09/01/2025	09/03/2025	\$6.99	В процесі
#96	Вечірня сукня	09/04/2025	09/12/2025	\$7	Завершено
#95	Футболка	09/06/2025	10/14/2025	\$8.7	Завершено
#94	Дитяча куртка	08/26/2025	09/03/2025	\$7	Завершено

Рисунок 3.14 – Сторінка контролю замовлень адміністратором

Також адміністратори мають секцію, для раніше згаданої оплати за допомогою прямого банківського переказу. У разі підтвердження факту виконання операції, адміністратори змінюють статус на “Схвалено”, у

протилежному – на “Відхилено” та додають опис причини, щоб користувачі могли зрозуміти, які дії їм потрібно виконати, для виправлення ситуації.

На рис. 3.15 зображено сторінку керування платежами.

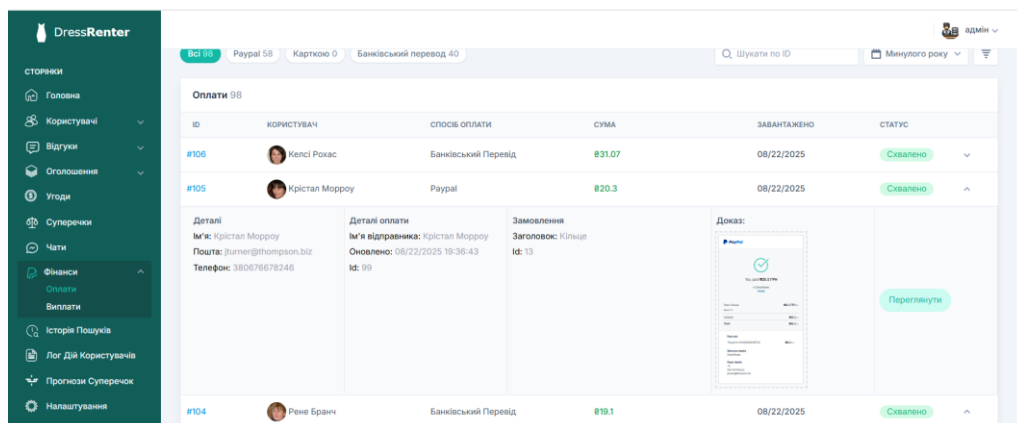


Рисунок 3.15 – Сторінка перегляду платежів адміністратором

Також у секції адміністраторів наявний повноцінний модуль роботи з суперечками, який надає їм можливості аналізу, модерації та вирішення конфліктних ситуацій. На додачу, у даній секції інтегровано спеціалізовані чати для кожної суперечки, що дозволяє адміністраторам спілкуватися окремо з кожним учасником суперечки, уточнюючи деталі та запитуючи докази. Завдяки цьому платформа має прозорий механізм вирішення конфліктів, який поєднує автоматизовану обробку даних і живу комунікацію, що значно підвищує рівень довіри користувачів до платформи.

На рис. 3.16 зображено комунікацію зі сторони адміністратора.

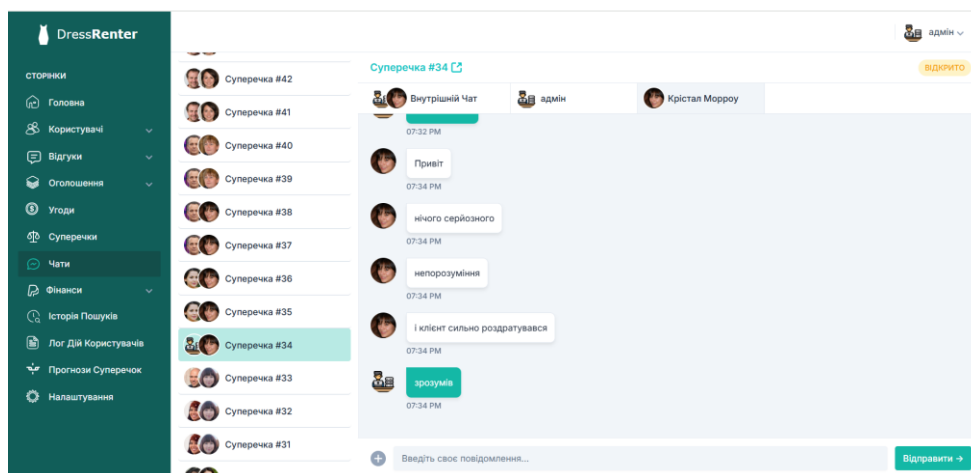


Рисунок 3.16 – Чат суперечки адміністратора з користувачем

На рис. 3.17 зображено можливість перегляду адміністраторами чату між користувачами. Така функція дозволяє адміністраторам здійснювати моніторинг взаємодії між учасниками чату, що важливо для забезпечення порядку та контролю за дотриманням правил використання сервісу. Історія чату відображається повністю, навіть з видаленими повідомленнями, що дає можливість виявляти та реагувати на шахрайства, непорозуміння або конфлікти, що можуть виникнути під час комунікації.

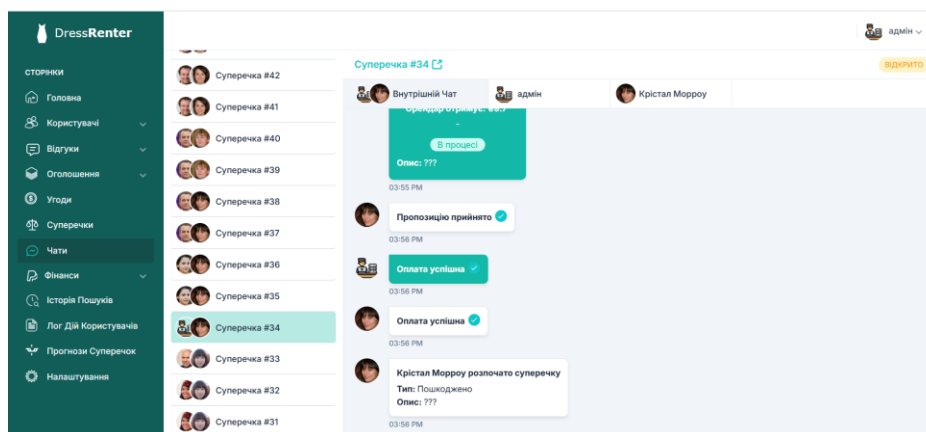


Рисунок 3.17 – Перегляд чату користувачів адміністратором

Як можна побачити, на демонстративній платформі реалізовано якісний механізм для вирішення суперечок, однак на кожную суперечку витрачається багато часу адміністраторів, тобто, власники платформи витрачають на це ресурси, та фактично, втрачають кошти на кожній такій ситуації, тому є сенс звести кількість таких випадків до мінімуму.

Таким чином, можна побачити, що зменшення кількості суперечок – це ідеальне завдання для застосування реалізованого модуля, адже демонстративна платформа є невеликою, у будь-який момент може змінитись тип продукції, наприклад з одягу на техніку, або навіть змінитись формат оплати одноразових платежів до регулярних. У такому випадку власникам платформи доведеться не наймати спеціаліста для аналізу даних та тренування нової моделі, а просто підкоригувати поточну структуру та перетренувати її у декілька кліків.

3.3 Інтеграція розробленого API прогнозування ризиків в основну частину платформи

Етап розпочався з інтеграції розроблених API для тренування моделей прогнозування ймовірності виникнення суперечки, отримання рекомендованих для тренування параметрів на основі створеної адміністратором структури та отримання прогнозів.

Оскільки функціонал пов'язаний з тренуванням знаходиться на окремому сервері, то першочерговим завданням було підключити базу даних демонстративного проєкту до нього за допомогою env-змінних.

Далі було реалізовано нову секцію для адміністраторів з назвою “Disputes Predictions” у клієнтській частині програми. У ній відображається найважливіша інформація про модель, наприклад – статуси моделей, а також їх прогрес тренування.

Статус “Pending Started” відображається, коли сервер для тренування моделей повернув інформацію про рекомендовані поля і адміністратор може на основі них виконати тренування, “Started” – коли адміністратор відправляє запит на сервер тренування моделей запит для початку тренування, “Stopped” відображається якщо під час тренування моделі адміністратор зупиняє цей процес, та “Finished”, який означає, що тренування моделі було повністю завершено та її можна використовувати.

Ще однією важливою складовою інтерфейсу є колонка активності моделі, яка визначає, яка з моделей наразі використовується системою для прогнозування. На платформі може бути активною лише одна модель, саме вона застосовується для розрахунку ймовірності виникнення суперечки.

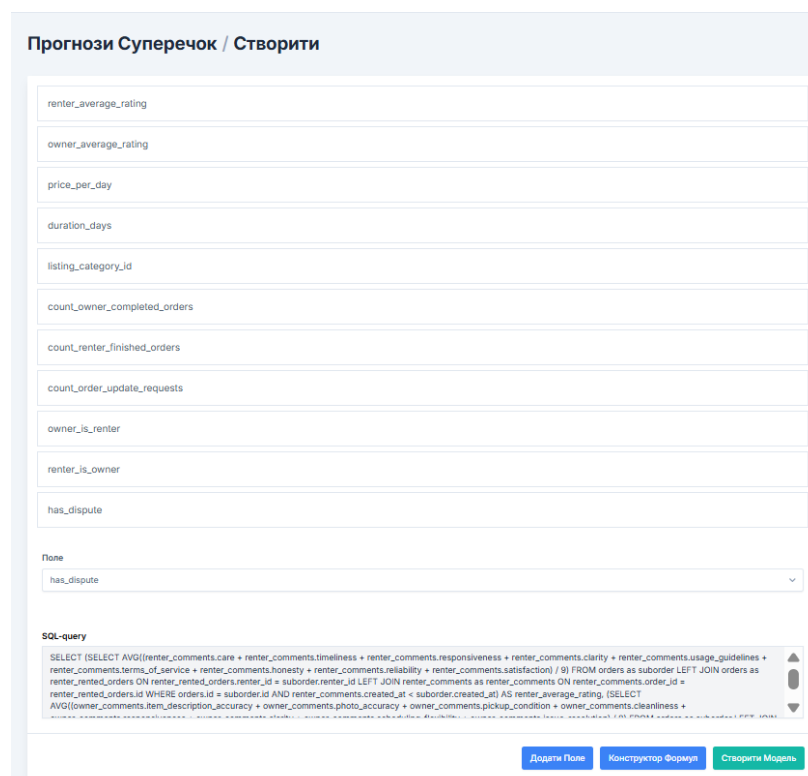
На додачу, у таблиці передбачено колонку точності моделі, значення якої повертає сервер прогнозування після завершення тренування. Цей показник дозволяє адміністраторам оцінити ефективність новоствореної моделі та прийняти рішення – активувати її для використання або залишити попередню версію, якщо вона демонструє кращі результати.

Також на сторінці перегляду списку моделей адміністратори одразу бачать, які поля були використані для формування вибірки перед тренуванням, а також дату початку тренування. Наявність цієї інформації на головній сторінці значно полегшує пошук потрібної моделі, дозволяючи швидко порівнювати їх параметри без необхідності переходити до детального перегляду.

Крім того, на сторінці з переліком моделей передбачено зручний функціонал для створення нових моделей та редагування чернеток, які містять опис структури даних і перелік характеристик, що використовуються під час тренування. Це дозволяє адміністраторам гнучко налаштовувати параметри майбутніх моделей без потреби щоразу починати з нуля.

Додатково реалізовано можливість клонування існуючих моделей, що значно прискорює процес створення нових версій на основі вже перевірених конфігурацій і вибірок даних.

На рис. 3.18 зображено сторінку створення нової моделі адміністратором.



Прогнози Суперечок / Створити

renter_average_rating

owner_average_rating

price_per_day

duration_days

listing_category_id

count_owner_completed_orders

count_renter_finished_orders

count_order_update_requests

owner_is_renter

renter_is_owner

has_dispute

None

has_dispute

SQL-query

```
SELECT (SELECT AVG(renter_comments.care + renter_comments.timeliness + renter_comments.responsiveness + renter_comments.clarity + renter_comments.usage_guidelines + renter_comments.terms_of_service + renter_comments.honesty + renter_comments.reliability + renter_comments.satisfaction) / 9) FROM orders as suborder LEFT JOIN orders as renter_rented_orders ON renter_rented_orders.renter_id = suborder.renter_id LEFT JOIN renter_comments ON renter_comments.order_id = renter_rented_orders.id WHERE orders.id = suborder.id AND renter_comments.created_at < suborder.created_at AS renter_average_rating, (SELECT AVG(owner_comments.item_description_accuracy + owner_comments.photo_accuracy + owner_comments.pickup_condition + owner_comments.cleanliness +
```

Додати Поле Конструктор Формул Створити Модель

Рисунок 3.18 – Створення нової моделі

Під час створення нової моделі адміністратори повинні налаштувати вибір полів даних і задати для них псевдоніми. Це дозволяє зручніше розрізняти параметри при подальшому аналізі результатів прогнозування. Для цього передбачено два механізми вибору: простий та складений. У простому режимі адміністратор може швидко обрати потрібне поле з однієї з таблиць бази даних, тоді як складений режим дає змогу попередньо обробляти дані, поєднувати декілька полів і перетворювати їх у числові або категоріальні ознаки. Такий підхід забезпечує гнучкість під час формування навчальної вибірки та дозволяє створювати більш точні моделі. Відповідні механізми вибору полів зображено на рис. 3.19 та 3.20.

Після формування списку вибраних полів адміністратори мають обрати категоріальне поле, для якого будуть генеруватись прогнози тренованої моделі. Такий підхід реалізовано з метою підвищення гнучкості системи та забезпечення зручності роботи під час подальших етапів налаштування.

На додачу, під час процесу побудови запиту для вибірки даних передбачено можливість його попереднього тестування. У спеціальному полі SQL-query автоматично формується запит, який ідентичний до того, що буде використано під час тренування моделі. Завдяки цьому адміністратори можуть перевірити коректність структури запиту, переконатися, що він повертає очікувані результати, та уникнути неочікуваних помилок під час виконання тренування.

Рисунок 3.19 – Додавання простого поля категорії оголошення у перелік характеристик тренування моделі

Під час створення простої характеристики адміністратори можуть використати базову таблицю ордерів або під'єднати до неї інші таблиці, що мають прямі зв'язки. Таке під'єднання можна виконувати послідовно кілька разів, доки не буде знайдено потрібну таблицю та поле. Для кожної вибраної характеристики користувачі задають псевдонім, а також можуть визначити значення за замовчуванням для випадків, коли поле не має значення. Після цього адміністратори повинні обрати тип створеного поля – категоріальний або числовий, залежно від характеру даних та ролі, яку вони відіграватимуть у процесі тренування моделі.



Рисунок 3.20 – Додавання складеного поля тривалості замовлення на основі першої та останньої дат у перелік характеристик тренування моделі

Під час створення інтерфейсу для побудови складених характеристик було використано бібліотеки `@dnd-kit/core` та `@dnd-kit/sortable`, що дозволило реалізувати механізм drag-and-drop для зручного формування вибірки даних. Такий підхід забезпечує користувачам можливість самостійно створювати унікальні запити шляхом переміщення блоків у межах модального вікна. Лістинг реалізованого модального компонента наведено у Додатку Е.

Модальне вікно надає можливість виконувати як арифметичні, так і SQL-операції. Серед них – обчислення середнього значення, отримання дня тижня або години з поля, а також звичайні операції, як-от групування у дужки, підстановка значення з таблиці чи задання константи.

При підстановці значень із таблиці адміністратор використовує подібну форму, що й при створенні простої вибірки. Крім того, при розробці цього компоненту було враховано особливість роботи зі складеними запитами: усі приєднання таблиць виконуються у межах одного підзапиту та стають спільно доступними для всіх полів. Це означає, що після приєднання певної таблиці її дані залишаються доступними й під час вибору наступних полів, що суттєво підвищує гнучкість побудови складних характеристик.

На рис. 3.21 зображено форму вибору поля у складеній формулі.

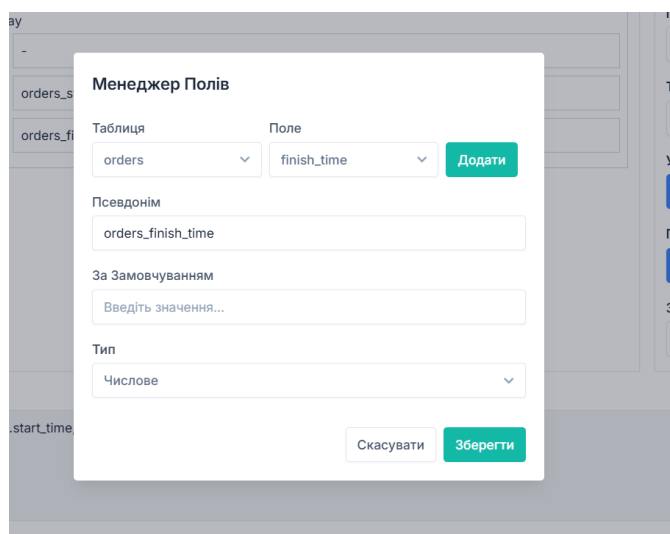


Рисунок 3.21 – Вибір поля дати завершення замовлення для підставлення у формулу обрахунку тривалості замовлення

Під час побудови підзапиту користувачі мають можливість налаштовувати додаткові параметри вибірки, зокрема додавати умови фільтрації, задавати значення за замовчуванням безпосередньо для результату або виконувати групування даних у разі потреби. Така гнучкість дозволяє точніше формувати набір даних для подальшого тренування моделі.

Як і при перегляді повного запиту вибірки даних, адміністратори можуть протестувати підзапит, у відповідному полі SQL-query. Отриманий код можна скопіювати та виконати вручну для перевірки коректності результатів і впевненості, що запит повертає саме ті дані, які очікуються для моделі.

На етапі отримання прогнозу, щодо вибірки полів для тренування моделей, користувачам також доступний розділ з рекомендаціями системи, який містить графіки та числові показники на основі яких було прийнято рішення рекомендувати чи не рекомендувати певну ознаку. Це допомагає адміністраторам глибше розуміти логіку побудови моделі та обґрунтованість запропонованих змінних.

Інтерфейс перегляду детальної аналітики та графіків представлено на рис. 3.22.

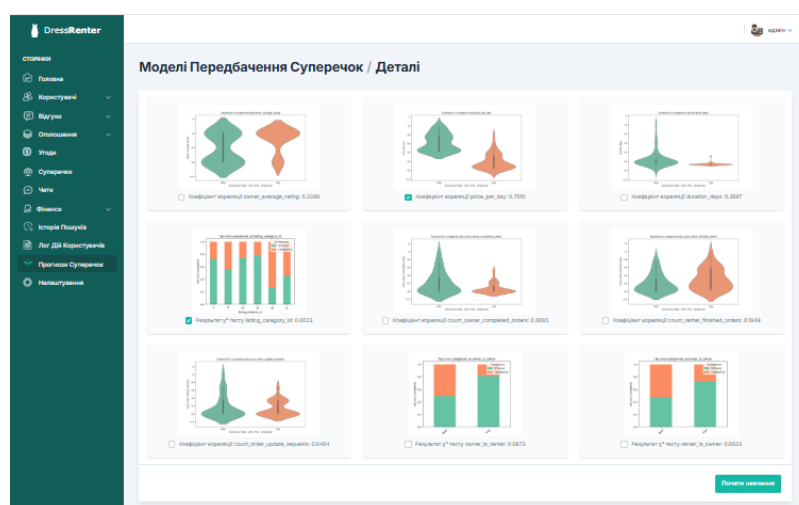
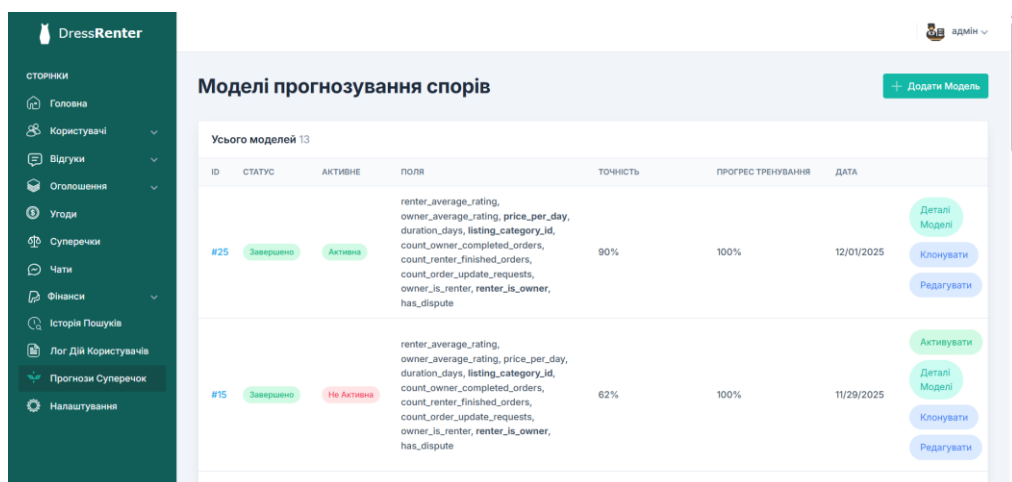


Рисунок 3.22 – Перегляд деталей системи рекомендацій

Після проведення кількох тестових тренувань було виявлено, що точність моделей прогнозування виникнення суперечок значною мірою залежить від кількості та якості використаних вхідних параметрів. Найкращу точність значенням 84% продемонструвала модель, для тренування якої використано дані, запропоновані рекомендаційною системою. До них належать такі параметри: ціна замовлення, чи здає замовник власні речі в оренду, а також категорія предмету, що здається.

Натомість моделі з більшою кількістю логічних ознак показували лише до 60% точності. Це свідчить про те, що для підвищення ефективності прогнозування доцільно орієнтуватися на рекомендаційну систему, яка враховує поведінкові та контекстні зв'язки, а не лише кількісні характеристики.

На рис. 3.23 зображено результати тренування кількох тестових моделей.



The screenshot shows the 'Деталі Моделі' (Model Details) page for model #25. The table lists the following data:

ID	СТАТУС	АКТИВНЕ	ПОЛЯ	ТОЧНІСТЬ	ПРОГРЕС ТРЕНУВАННЯ	ДАТА
#25	Завершено	Активне	renter_average_rating, owner_average_rating, price_per_day, duration_days, listing_category_id, count_owner_completed_orders, count_renter_finished_orders, count_order_update_requests, owner_is_renter, renter_is_owner, has_dispute	90%	100%	12/01/2025
#15	Завершено	Не Активне	renter_average_rating, owner_average_rating, price_per_day, duration_days, listing_category_id, count_owner_completed_orders, count_renter_finished_orders, count_order_update_requests, owner_is_renter, renter_is_owner, has_dispute	62%	100%	11/29/2025

Рисунок 3.23 – Порівняльні результати тренування тестових моделей прогнозування суперечок між користувачами

Після створення та успішного тренування моделей адміністратори мають можливість керувати їхньою активністю – перемикатися між різними моделями та виконувати перезапис прогнозів ймовірності виникнення суперечок у наявних замовленнях. Це дає змогу оперативно оновлювати дані на основі більш точної або новоствореної моделі без необхідності повторного ручного аналізу, а також повертатись до існуючих надійних моделей, у випадку виникнення непередбачуваних ситуацій.

Приклад реалізації цього механізму наведено на рис. 3.24.

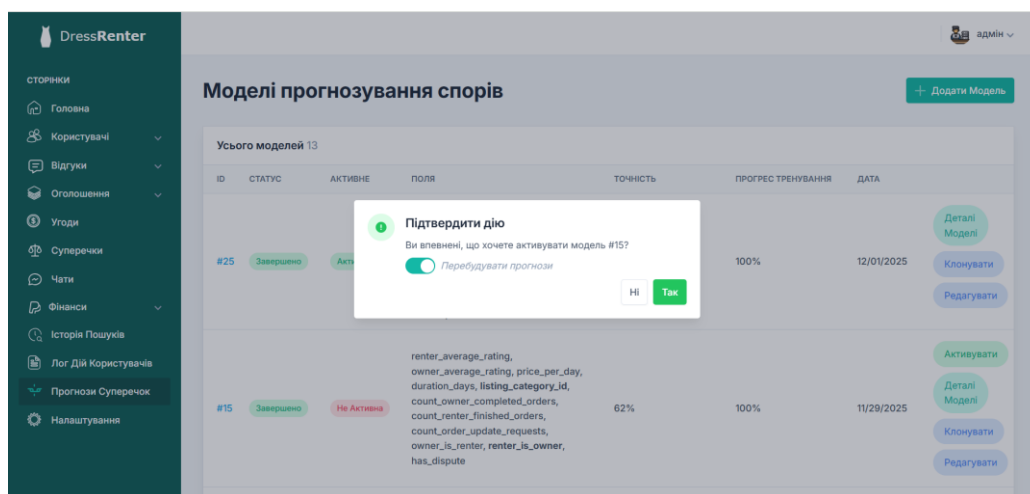


Рисунок 3.24 – Позначення розробленої моделі активною та перезапис ймовірностей у всіх замовленнях

Після активації моделі користувачі отримують прогнози щодо ризику виникнення суперечки. Система автоматично аналізує введені дані, які були визначені адміністратором під час етапу тренування моделі, та обчислює відповідну ймовірність ризику. Результати прогнозу відображаються у зрозумілому для користувача відсотковому форматі, а для зручності сприйняття впроваджено кольорову індикацію рівня ризику: низький ризик (від 1 до 33%) – зелений колір, середній ризик (від 34 до 66%) – жовтий, високий ризик (від 67 до 99%) – червоний. Такий підхід забезпечує інтуїтивне розуміння результатів прогнозу та допомагає користувачам приймати більш зважені рішення перед підтвердженням угоди.

На рис. 3.25 зображено відображення ймовірності виникнення суперечки у користувачів при створенні замовлення.

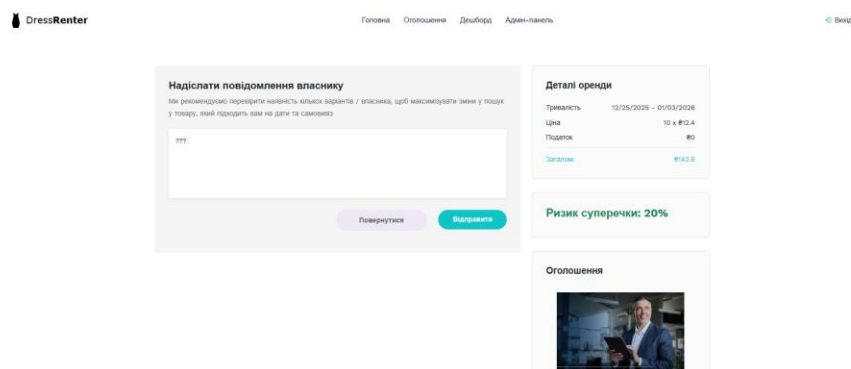


Рисунок 3.25 – Відображення ймовірності виникнення суперечки до створення замовлення

На рис. 3.26 зображено відображення ймовірності виникнення суперечки у користувачів на етапі підтвердження замовлення. Даний прогноз на цьому етапі є важливим, адже теоретично, на цьому етапі ймовірність виникнення суперечки може змінитись через зміни дат та ціни в пропозиції. Завдяки повторній оцінці ймовірності виникнення суперечки після кожної пропозиції, користувач може вчасно отримати інформацію про можливу проблему, що дозволить йому вчасно вжити необхідних заходів для уникнення неприємних ситуацій.

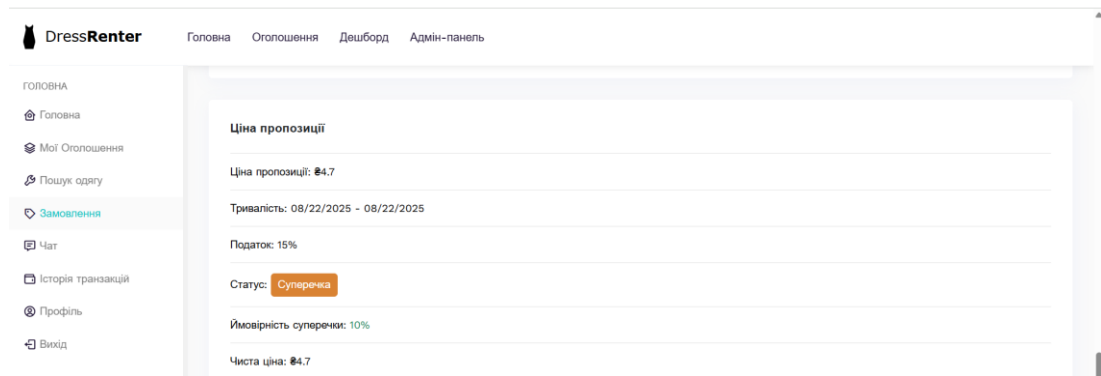


Рисунок 3.26 – Відображення ймовірності виникнення суперечки на етапі підтвердження замовлення

Висновки до третього розділу

У цьому розділі було проведено повний цикл інтеграції розробленого інтелектуального модуля для динамічного навчання моделей і рекомендації параметрів для платформи оренди одягу, що містить систему управління користувачами, замовленнями, фінансовими операціями, комунікацією між користувачами та систему вирішення суперечок.

У базі даних платформи було створено таблицю для керування моделями. Для цього було розроблено окрему міграцію на основному сервері проєкту. Після аналізу структури бази даних було визначено можливу вибірку параметрів для тренування моделі прогнозування даних. Після цього було виконано тестове тренування на основі рекомендованих параметрів зі створених, під час якого було продемонстровано простоту роботи з модулем. У результаті модель продемонструвала точність 84%, що підтверджує доцільність інтеграції та є прийнятним показником для проєкту з невеликим обсягом даних.

Таким чином, завдяки інтеграції розробленого інтелектуального модуля для динамічного тренування моделей прогнозування даних було довершено багатофункціональну комерційну платформу для оренди одягу, яка поєднує зручність користування, автоматизацію бізнес-процесів та високий рівень безпеки.

ВИСНОВКИ

У результаті виконання випускної кваліфікаційної роботи магістра було розроблено інтелектуальний модуль для динамічного навчання моделей та рекомендації параметрів для їхнього кращого тренування навіть користувачами без спеціальної освіти та досвіду. Дана реалізація дозволяє зменшити витрати на аналіз інформації та тренування моделей для прогнозування даних невеликих комерційних платформ, адже завдяки розробленому модулю такі завдання зможуть виконувати не лише кваліфіковані спеціалісти, а і звичайні люди.

Важливим етапом розробки стало дослідження та обґрунтування вибору методу випадкового лісу, як оптимального алгоритму прогнозування, здатного працювати з різними типами даних, зберігаючи стабільність та точність результатів навіть у випадку обмеженого набору навчальних даних.

Під час аналізу було прийнято рішення не використовувати логістичну регресію через те, що вона недостатньо орієнтована на роботу з категоріальними параметрами, метод найближчих сусідів не підійшов через потребу в експериментуванні з визначенням оптимальної кількості сусідів, що призводило б до частого перетреноування моделі, а нейронні мережі не було використано, оскільки вони мають високу часову складність тренування та потребують великої кількості даних, що є проблемою для невеликих проєктів.

Також метод випадкового лісу було експериментально порівняно з одним із подібних методів, а саме з XGBoost. Для цього було використано три різних набори даних з різною кількістю записів. Виявилось, що більшу точність прогнозу зазвичай має XGBoost, однак його не було обрано, оскільки різниця у точності була незначною, а метод працював нестабільно на одному із використовуваних датасетів, що могло б призвести до подібних проблем на нових проєктах.

Ще однією важливою реалізацією у даному модулі є система рекомендації параметрів. Завдяки ній користувачі можуть якісно натренувати модель, не маючи для цього потрібних умінь. Достатньо просто погодитись із рекомендаціями системи, які визначаються на основі коефіцієнту кореляції для числових параметрів, а також на основі тесту χ^2 для категоріальних. На основі раніше згаданих датасетів було експериментально визначено оптимальні граничні значення за замовчуванням. Ними стали 0.6 для числових, що відповідає середньому рівню залежності, та 0.05 для категоріальних, що відповідає високому. Було вирішено використовувати саме ці значення, оскільки зі збільшенням значень коефіцієнтів для фільтрації даних значно зменшувалась кількість можливих даних у датасетах, а точність моделей майже не змінювалась, що могло б стати проблемою для невеликих проєктів.

Після завершення розробки модуля для динамічного навчання моделей і рекомендації параметрів його було інтегровано у демонстративну комерційну платформу для оренди одягу задля швидкої реалізації прогнозування ризиків виникнення суперечок. Завдяки розробленому модулю, функціонал вдалося реалізувати швидко, дешево та якісно. Для тренування моделі було побудовано структуру параметрів за допомогою відповідної секції в адмін панелі. Перед тренуванням було відправлено запит для побудови графіків, а також визначення рекомендованих параметрів для тренування. Далі було виконано тренування, у результаті якого було отримано модель, точність якої становила 84%, що є гарним показником для проєкту з невеликим набором даних.

Робота над кожним етапом керувалася метою створення дешевого та ефективного інструменту для її користувачів. Після завершення проєкту можна зробити висновок, що пропонований програмний продукт готовий до впровадження в реальні проєкти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Verbovskiy O. Y., Loktikova T. M. Using Data Prediction Methods and Algorithms to Improve the Interactive Platform User Experience. Тези доповідей Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки, 12-17 травня 2025 року. Житомир, 2025. С. 52–53.
2. Дослідження принципів побудови та проєктування інтерактивної платформи з використанням методів та алгоритмів прогнозування даних / О. Вербовський та ін. Вісник Хмельницького національного університету. Серія: технічні науки. 2026. № 1 (подано та прийнято до опублікування).
3. Prateek J. Artificial Intelligence with Python. Birmingham: Packt Publishing, 2017. 446 p.
4. Raschka S., Patterson J., Nolet C. Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence. Information. 2020. Vol. 11, no. 4. P. 1–44. URL: <https://doi.org/10.3390/info11040193>.
5. Hoffmann M., Scherer M., Hempel T. Deeptime: a Python library for machine learning dynamical models from time series data. Machine Learning: Science and Technology. 2021. Vol. 3, no. 1. P. 1–28. URL: <https://doi.org/10.1088/2632-2153/ac3de0>.
6. Hao J., Ho T. K. Machine Learning Made Easy: A Review of Scikit-learn Package in Python Programming Language. Journal of Educational and Behavioral Statistics. 2019. Vol. 44, no. 3. URL: <https://doi.org/10.3102/1076998619832248>.
7. Python-based scikit-learn machine learning models for thermal and electrical performance prediction of high-capacity lithium-ion battery / M.-K. Tran et al. International Journal of Energy Research. 2021. URL: <https://doi.org/10.1002/er.7202>.
8. An Empirical Study on the Energy Usage and Performance of Pandas and Polars Data Analysis Python Libraries / F. Nahrstedt et al. Association for

Computing Machinery. 2024. P. 58–68. URL: <https://doi.org/10.1145/3661167.3661203>.

9. Hagedorn S., Kläbe S., Sattler K.-U. Putting Pandas in a Box. Conference on Innovative Data Systems Research. 2021. URL: <https://doi.org/10.22032/dbt.51534>.

10. Testas A. Distributed Machine Learning with PySpark. Berkeley, CA : Apress, 2023. 490 p. URL: https://doi.org/10.1007/978-1-4842-9751-3_10.

11. Relan K. Building REST APIs with Flask. Apress Berkeley, CA: Apress, 2019. 199 p. URL: <https://doi.org/10.1007/978-1-4842-5022-8>.

12. Nguyen Q. Mastering Concurrency in Python. Birmingham: Packt Publishing, 2018. 446 p.

13. Wicaksono D., Soewito B. Application of the Multi-Threading Method and Python Script for the Automate of Network. Syntax Literate: Jurnal Ilmiah Indonesia. 2024. Vol. 9, no. 6. URL: <https://doi.org/10.36418/syntax-literate.v9i6.16345>.

14. Prasad S. V., Donthi R., Challa M. K. A Precise Study on Solving the Multi Server Queuing Model through Lpp By Using Python Program. IOP Conference Series: Materials Science and Engineering. 2020. Vol. 933. URL: <https://doi.org/10.1088/1757-899X/993/1/012109>.

15. Zapata J. G. MySQL vs PostgreSQL: A Comparative Analysis of Relational Database Management Systems (RDBMS) Technologies Response Time in Web-based E-commerce. 2024. URL: <https://doi.org/10.13140/RG.2.2.24791.69288>.

16. Vadlamani V. Introduction to PostgreSQL Database Management. PostgreSQL Skills Development on Cloud. – Berkeley, CA. 2024. URL: https://doi.org/10.1007/979-8-8688-0817-3_1.

17. Singh U. REST API Framework: Designing and Developing Web Services. International Research Journal of Engineering and Technology. 2021. T. 8. C. 815–817.

18. Freeman A. Mastering Node.js Web Development: Go on a comprehensive journey from the fundamentals to advanced web development with Node.js. Packt Publishing. 2024.
19. Halili F., Ramadani E. Web Services: A Comparison of Soap and Rest Services. Modern Applied Science. 2018. T. 12, № 3. C. 175. URL: <https://doi.org/10.5539/mas.v12n3p175>.
20. Patel V. Analyzing the Impact of Next.JS on Site Performance and SEO. International Journal of Computer Applications Technology and Research. 2023. T. 12, № 10. C. 24–27. URL: <https://doi.org/10.7753/IJCATR1210.1004>.
21. Forecasting of Rice Production Using a Linear Regression and Polynomial Regression / B. Wibisana et al. Journal of Informatics and Communication Technology. 2024. Vol. 6, no. 1. P. 44–41. URL: <https://doi.org/10.52661/jict.v6i1.232>.
22. Yang F. J. An Extended Idea about Decision Trees. 2019 International Conference on Computational Science and Computational Intelligence (CSCI). 2019. P. 349–354. URL: <https://doi.org/10.1109/CSCI49370.2019.00068>.
23. SONG Y.-y., LU Y. Decision tree methods: applications for classification and prediction. Shanghai Archives of Psychiatry. 2015. Vol. 27, no. 2. P. 130–135. URL: <https://doi.org/10.11919/j.issn.1002-0829.215044>.
24. Ville B. Decision trees. WIREs Computational Statistics. 2013. Vol. 5, no. 6. P. 448–455. URL: <https://doi.org/10.1002/wics.1278>.
25. Sapra V., Sapra L., Srivastava P. Identification of Brain Stroke using Boosted Random Forest. 2022 International Conference on Advances in Computing, Communication and Materials (ICACCM). 2022. P. 1–5. URL: <https://doi.org/10.1109/ICACCM56405.2022.10009527>.
26. Belgiu M., Drăguț L. Random forest in remote sensing: A review of applications and future directions. ISPRS Journal of Photogrammetry and Remote Sensing. 2016. Vol. 114. P. 24–31. URL: <https://doi.org/10.1016/j.isprsjprs.2016.01.011>.

27. Salman H. A., Kalakech A., Steiti A. Random Forest Algorithm Overview. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2024. Vol. 2024. P. 69–79. URL: <https://doi.org/10.58496/BJML/2024/007>.
28. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016. P. 785–794. URL: <https://doi.org/10.1145/2939672.2939785>.
29. Applications of XGBoost in water resources engineering: A systematic literature review (Dec 2018–May 2023) / M. Niazkar et al. *Environmental Modelling & Software*. 2024. Vol. 174. URL: <https://doi.org/10.1016/j.envsoft.2024.105971>.
30. XGBoost Model for Chronic Kidney Disease Diagnosis / M. Niazkar et al. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. 2020. Vol. 17, no. 6. P. 2131–2140. URL: <https://doi.org/10.1109/TCBB.2019.2911071>.
31. Nalluri M., Pentela M., Eluri N. R. A Scalable Tree Boosting System: XG Boost. *International Journal of Research Studies in Science, Engineering and Technology*. 2020. Vol. 7, no. 12. P. 36–51. URL: <https://doi.org/10.22259/2349-476X.0712005>.
32. Logistic Regression / Y. Geng et al. *Practical Machine Learning Illustrated with KNIME*. 2024. P. 99–132. URL: https://doi.org/10.1007/978-981-97-3954-7_4.
33. Hou P., Zhou L., Yang Y. Density clustering method based on k-nearest neighbor propagation. *Journal of Physics: Conference Series*. 2024. Vol. 2858, no. 1. URL: <https://doi.org/10.1088/1742-6596/2858/1/012041>.
34. Cunningham P., Delany S. J. k-Nearest Neighbour Classifiers - A Tutorial. *ACM Computing Surveys (CSUR)*. 2021. Vol. 54, no. 6. P. 1–25. URL: <https://doi.org/10.1145/3459665>.
35. Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review / H. A. Abu Alfeilat et al. *Big Data*. 2019. Vol. 7, no. 4. URL: <https://doi.org/10.1089/big.2018.0175>.

36. Brunton S. L., Kutz J. N. Data Driven Science & Engineering Machine Learning, Dynamical Systems, and Control. Brunton & Kutz, Cambridge University Press, 2019. URL: <https://doi.org/10.1017/9781108380690>.
37. Islam M., Chen G., Jin S. An Overview of Neural Network. American Journal of Neural Networks and Applications. 2019. Vol. 5, no. 1. P. 7–11. URL: <https://doi.org/10.11648/j.ajnna.20190501.12>.
38. Enzo G., Massimo B. Introduction to artificial neural networks. European Journal of Gastroenterology & Hepatology. 2019. Vol. 19, no. 12. P. 1046–1054. URL: <https://doi.org/10.1097/MEG.0b013e3282f198a0>.
39. Scikit-learn: Machine Learning in Python / P. Pedregosa et al. Journal of Machine Learning Research 12. 2011. P. 2825–2830.
40. McKinney W. Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference (SciPy). 2010. P. 56–61. URL: <https://doi.org/10.25080/Majora-92bf1922-00a>.
41. Array programming with NumPy. Nature. 2020. Vol. 585. P. 357–362. URL: <https://doi.org/10.1038/s41586-020-2649-2>.
42. Sun R. Communications in Computer and Information Science. 1506th ed. Springer, Singapore: Communications in Computer and Information Science, 2022. URL: https://doi.org/10.1007/978-981-16-9229-1_11.
43. Gupta A. Advancing API Security: A Comprehensive Evaluation of Authentication Mechanisms and Their Implications for Cybersecurity. 2024. URL: <https://doi.org/10.21428/e90189c8.406d2328>.
44. Testas A. Distributed Machine Learning with PySpark. Berkeley, CA : Apress, 2023. 490 p. URL: https://doi.org/10.1007/978-1-4842-9751-3_10.
45. Lee K.-H. A Security Threats in Wireless Charger Systems in M2M. Journal of the Korea Convergence Society. 2013. Vol. 4, no. 1. URL: <https://doi.org/10.15207/JKCS.2013.4.1.027>.

ДОДАТКИ

Код функції-генератора запиту вибірки

```

def get_query_to_train(body, mainTable='orders'):
    query = 'SELECT '
    joins = {}

    for index, field in enumerate(body):
        if index > 0:
            query += ", "

        if field["type"] == "field":
            query +=
f'{field["content"]["tableName"]}.{field["content"]["fieldName"]}'

            if field.get("pseudonym"):
                query += f' as {field["pseudonym"]}'

            for join in field["content"]["joins"]:
                joins[join["pseudonym"]] = (
                    f'LEFT JOIN {join["joinedTable"]} as {join["pseudonym"]} '
                    f'ON {join["pseudonym"]}.{join["joinedField"]} = '
                    f'{join["baseTable"]}.{join["baseField"]}'
                )

        if field["type"] == "template":
            inner_joins = {}
            inner_query_parts = ["SELECT "]

            def add_query_by_item(item):
                date_part_keys = {...}
                function_wrappers = {...}
                for key, part in date_part_keys.items():
                    function_wrappers[key] = [f'date_part('{part}', ", ')]
                wrapper = function_wrappers.get(item["key"])

                if wrapper:
                    inner_query_parts.append(wrapper[0])
                    for sub_item_index, sub_item in enumerate(item["subItems"]):
                        if sub_item_index != 0 and not
is_operation_key(item["subItems"][sub_item_index - 1]["key"]) and not
is_operation_key(sub_item["key"]) and not (sub_item_index !=
len(item["subItems"]) - 1 and is_operation_key(item["subItems"][sub_item_index +
1]["key"])):
                            inner_query_parts.append(', ')
                            add_query_by_item(sub_item)
                            inner_query_parts.append(wrapper[1])

            if is_operation_key(item["key"]):
                inner_query_parts.append(f' {item["key"]} ')

```

```

        if item["key"] == 'table_selects':

            inner_query_parts.append(f'{swap_sub_orders_table(item["content"]["tableName"])}
            {item["content"]["fieldName"]}')
            for join in item["content"]["joins"]:
                inner_joins[join["pseudonym"]] = (
                    f'LEFT JOIN {join["joinedTable"]} as
{join["pseudonym"]} '
                    f'ON {join["pseudonym"]}.{join["joinedField"]} =
' f'{swap_sub_orders_table(join["baseTable"])}.{join["baseField"]}')

            if item["key"] == 'custom_value':
                inner_query_parts.append(item["content"]["value"])

        for content_item in field["content"]:
            add_query_by_item(content_item)

        inner_query_parts.append(' FROM orders as suborder')

        for join_key in inner_joins:
            inner_query_parts.append(f' {inner_joins[join_key]}')
            inner_query_parts.append(' WHERE orders.id = suborder.id')

        conditions = field.get('conditions', [])

        if len(conditions) > 0:
            for index, condition in enumerate(conditions):
                inner_query_parts.append(
                    f' AND
{swap_sub_orders_table(condition["baseTable"])}.{condition["baseField"]}
{condition["joinCondition"]}
{swap_sub_orders_table(condition["joinedTable"])}.{condition["joinedField"]}'
                )
        groups = field.get('groups', [])

        if len(groups) > 0:
            inner_query_parts.append(' GROUP BY ')
            for index, group in enumerate(groups):
                if index > 0:
                    inner_query_parts.append(', ')
            inner_query_parts.append(f'{swap_sub_orders_table(group["baseTable"])}.{group["baseField"]}')

        orders = field.get('orders', [])

        if len(orders) > 0:
            inner_query_parts.append(' ORDER BY ')
            for index, order in enumerate(orders):
                if index > 0:

```

```

        inner_query_parts.append(', ')
inner_query_parts.append(f'{swap_sub_orders_table(order["baseTable"])}.{order["baseField"]}')
    query += f'({"".join(inner_query_parts)}) AS {field["pseudonym"]}'

    query += ' , orders.id as id FROM ' + mainTable + ' as orders'
    for join_key in joins:
        query += f' {joins[join_key]}'
    return query

```

Додаток Б

Код функції, що тренує модель

```

def train_random_forest(model_id, task_queue = None):
    app = create_app()

    with app.app_context():
        def update_progress(progress):
            model = DisputePredictionModels.query.get(model_id)
            if model:
                model.progress_percent = progress
                db.session.commit()

        prediction_model = DisputePredictionModels.query.get(model_id)
        query = get_query_to_train(prediction_model.body)

        result = db.session.execute(text(query))
        rows = result.fetchall()
        fields = result.keys()
        selected_fields = prediction_model.selected_fields if
prediction_model.selected_fields else []

        filtered_fields = [field for field in fields if field in selected_fields
or field == prediction_model.check_field]

        categorical_fields = []
        numeric_fields = []

        for field in prediction_model.body:
            if field.get("pseudonym") in selected_fields:
                if field.get("comparisonType") == "categorical":
                    categorical_fields.append(field["pseudonym"])
                else:
                    numeric_fields.append(field["pseudonym"])

        preprocessor = ColumnTransformer(transformers=[ ('cat',
OneHotEncoder(handle_unknown='ignore', _output=False),
categorical_fields), ('num', 'passthrough', numeric_fields)])

        update_progress(10)

```

```

        data_to_generate = [{field: get_model_field_value(row, field,
prediction_model.body) for field in filtered_fields} for row in rows]

    df = pd.DataFrame(data_to_generate)

    X = df.drop(prediction_model.check_field, axis=1)
    y = df[prediction_model.check_field]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=System.get_train_test_split(), random_state=System.get_random_state())

    model = Pipeline(steps=[('preprocess', preprocessor),
        ('rf', RandomForestClassifier(n_estimators=System.get_n_estimators(),
            random_state=System.get_random_state(), n_jobs=-1))])

    update_progress(20)
    model.fit(X_train, y_train)
    update_progress(50)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred) * 100
    update_progress(90)
    model_save_path = f"images/{model_id}/random_forest_model.pkl"
    os.makedirs(os.path.dirname(model_save_path), exist_ok=True)
    joblib.dump(model, model_save_path)

    model_entry=DisputePredictionModels.query.filter_by(id=model_id).first()
    model_entry.accuracy = accuracy
    db.session.commit()

    if task_queue:
        if prediction_model.after_finish_rebuild:
            worker_thread = threading.Thread(target=revaluation_worker,
args=(task_queue,), kwargs={'props': {}})
            worker_thread.start()
            task_queue.put("Revaluation Model Started")

        if prediction_model.after_finish_active:
            DisputePredictionModels.query.filter(DisputePredictionModels.active ==
True).update({DisputePredictionModels.active: False}
                )
            prediction_model.active = True
            db.session.commit()
            update_progress(100)
            prediction_model.finished = True
            prediction_model.started = False
            db.session.commit()

    return model

```

Код функції, що передбачає ризик виникнення суперечки на основі
набору даних

```
def get_dispute_probability(data_to_generate, model_id=None):
    query = DisputePredictionModels.query

    if model_id:
        query = query.filter_by(id=model_id)
    else:
        query = query.filter_by(active=True)

    generated_model = query.first()
    row_data = {}

    for field in generated_model.selected_fields:
        row_data[field] = data_to_generate.get(field,
get_field_default_value(field, generated_model.body))

    model_path = f"images/{generated_model.id}/random_forest_model.pkl"
    if not os.path.exists(model_path):
        return {'prediction': 0, 'probability_of_delay': 50}

    model = joblib.load(model_path)
    print(type(model))
    X = pd.DataFrame([row_data])

    prediction = model.predict(X)
    prediction_proba = model.predict_proba(X)
    delay_probability = prediction_proba[0][1]
    delay_probability = round(int(delay_probability * 100))

    if delay_probability > 99:
        delay_probability = 99
    if delay_probability == 0:
        delay_probability = 1
    return {'prediction': int(prediction[0]), 'probability_of_delay':
delay_probability}
```

Код функції, що визначає ключові ознаки моделі та їх значимість для
виникнення суперечок

```
def model_dependencies(model):
    folder = 'images/'+str(model.id)

    if model.checked and model.prediction_details:
        return model.prediction_details
```

```

if not os.path.exists(folder):
    os.makedirs(folder)

query = get_query_to_train(model.body)
result = db.session.execute(text(query))
data = pd.DataFrame(result.fetchall(), columns=result.keys())

numerical_fields = [field["pseudonym"] for field in model.body if
field.get("comparisonType") == "numerical" and field.get("pseudonym") !=
model.check_field]
categorical_fields = [field["pseudonym"] for field in model.body if
field.get("comparisonType") == "categorical" and field.get("pseudonym") !=
model.check_field]

scaler = MinMaxScaler()

for field in numerical_fields:
    data[field] = scaler.fit_transform(data[[field]])
    plt.figure(figsize=(10, 6))
    sns.violinplot(x=data[model.check_field], y=data[field], palette='Set2')
    plt.title(f"Relation between {field} and Dispute")
    plt.xlabel('Dispute (1 = Yes, 0 = No)')
    plt.ylabel(field)
    plt.savefig(f'{folder}/{field}.png')
    plt.close()

for field in categorical_fields:
    category_dispute_freq = data.groupby([field,
model.check_field]).size().unstack(fill_value=0)
    category_dispute_freq =
category_dispute_freq.div(category_dispute_freq.sum(axis=1),
                           axis=0)

    plt.figure(figsize=(10, 6))

    category_dispute_freq.plot(kind='bar', stacked=True, colormap='Set2')

    no_dispute_patch = mpatches.Patch(color='#66c2a5', label='No Dispute
(0)')
    dispute_patch = mpatches.Patch(color='#fc8d62', label='Dispute (1)')
    plt.legend(handles=[no_dispute_patch, dispute_patch], title='Dispute',
loc='upper right')

    plt.title(f"Frequency of Dispute by {field}")
    plt.xlabel(field)
    plt.ylabel('Frequency of Dispute')
    plt.xticks(rotation=45)

    plt.savefig(f'{folder}/{field}.png')

```

```

plt.close()

correlation_matrix = data[numerical_fields + [model.check_field]].corr()

chi2_results = {}

for field in categorical_fields:
    contingency_table = pd.crosstab(data[field], data[model.check_field])
    # Perform the chi-square test
    chi2, p, dof, expected = chi2_contingency(contingency_table)

    chi2_results[field] = p

dependencies = {
    'correlation_dict': {field: correlation_matrix[field][model.check_field]
for field in numerical_fields},
    'chi2_dict': chi2_results
}

model.prediction_details = dependencies
model.checked = True
db.session.commit()

return dependencies

```

Додаток Г

Код сторінки створення структури моделі для прогнозування
ймовірності виникнення суперечок

```

import React, { useRef, useState } from "react";
import { DndContext, DragOverlay } from "@dnd-kit/core";
import { arrayMove } from "@dnd-kit/sortable";
import { cloneObject,
    generateRandomString,
    isItemKeyDraggable,
} from "../../utils";
import Sidebar from "../Sidebar";
import Content from "../Content";
import ContentItem from "../ContentItem";
import OverlayItem from "../OverlayItem";
import DraggableItem from "../DraggableItem";
import STATIC from "../../static";
import Query from "../Query";
import Where from "../Where";
import ModelParamFieldModal from "../ModelParamFieldModal";

const Builder = ({ tableStructure, dopProps }) => {
    const contentRef = useRef(null);
    const [activeDrag, setActiveDrag] = useState(null);
    const [activeTableDetails, setActiveTableDetails] = useState(null);

```

```

const handleDragStart = (event) => {
  setActiveDrag(event.active);
};

const getDroppableParent = (
  itemId,
  checkableItems = null,
  needCheck = true
) => {
  if (checkableItems === null) {
    checkableItems = dopProps.content;
  }

  for (let i = 0; i < checkableItems.length; i++) {
    const checkableItem = checkableItems[i];

    if (isItemKeyDraggable(checkableItem.key)) {
      for (let j = 0; j < checkableItem.subItems.length; j++) {
        const checkableChild = checkableItem.subItems[j];

        if (needCheck && !isItemKeyDraggable(checkableChild.key)) {
          break;
        }

        if (checkableChild.id === itemId) {
          return checkableItem;
        }
      }

      const resChildCheck = getDroppableParent(
        itemId,
        checkableItem.subItems,
        needCheck
      );

      if (resChildCheck) {
        return resChildCheck;
      }
    }
  }

  return null;
};

const insertItemById = (items, targetId, itemToInsert) => {
  return items.map((item) => {
    if (item.id === targetId) {
      return {
        ...item,

```

```

        subItems: [...(item.subItems || []), itemToInsert],
    };
}

if (item.subItems?.length) {
    return {
        ...item,
        subItems: insertItemById(item.subItems, targetId, itemToInsert),
    };
}

return item;
});
};

const findAndRemoveItem = (items, targetId) => {
    let found = null;

    const filtered = items
        .map((item) => {
            if (item.id === targetId) {
                found = item;
                return null;
            }

            if (item.subItems?.length) {
                const [cleanedChildren, removed] = findAndRemoveItem(
                    item.subItems,
                    targetId
                );
                if (removed) {
                    found = removed;
                    return { ...item, subItems: cleanedChildren };
                }
            }

            return item;
        })
        .filter(Boolean);

    return [filtered, found];
};

const getItemById = (id, items = null) => {
    if (items === null) {
        items = dopProps.content;
    }

    for (let item of items) {
        if (item.id === id) {

```

```

        return item;
    }

    if (item.subItems && item.subItems.length > 0) {
        const foundItem = getItemById(id, item.subItems);
        if (foundItem) {
            return foundItem;
        }
    }
}
return null;
};

const setTableDetails = ({ tableName, fieldName, joins, pseudonym }) => {
    dopProps.setContent((prevItems) =>
        updateItemRecursively(prevItems, activeTableDetails.id, {
            content: { tableName, fieldName, joins, pseudonym },
        })
    );
};

const setCustomValue = ({ id, value }) => {
    dopProps.setContent((prevItems) =>
        updateItemRecursively(prevItems, id, {
            content: { value },
        })
    );
};

return (
    <>
        <DndContext onDragStart={handleDragStart} onDragEnd={handleDragEnd}>
            <div className="flex space-x-4 w-full h-full h-max-full overflow-y-hidden
mb-4">
                <Sidebar
                    id={sidebarId}
                    items={[...] }
                    keyField="key"
                    bodyField="body"
                    Component={({ item }) => (
                        <DraggableItem item={item} example>
                            {item.body}
                        </DraggableItem>
                    )}
                />

                <Content
                    ref={contentRef}
                    id={dropdownId}
                    items={dopProps.content}

```

```

        keyField="id"
        Component=(({ item }) => (
            <ContentItem
                tableStructure={tableStructure}
                item={item}
                getDroppableParent={getDroppableParent}
                key={item.id}
                activeDrag={activeDrag}
                setActiveTableDetails={setActiveTableDetails}
                setCustomValue={setCustomValue}
            />
        ))
    />

    <Where
        tableStructure={tableStructure}
        items={dopProps.content}
        {...dopProps}
    />
</div>

<Query
    tableStructure={tableStructure}
    items={dopProps.content}
    {...dopProps}
/>

<DragOverlay dropAnimation={null}>
    {activeDrag ? (
        <OverlayItem
            active={activeDrag}
            examples={[
                ...withChildrenItems,
                ...operationItems,
                ...customItems,
            ]}
            items={dopProps.content}
        />
    ) : null}
</DragOverlay>
</DndContext>

<ModelParamFieldModal
    tableStructure={tableStructure}
    onSaveClick={setTableDetails}
    modalOpen={!!activeTableDetails}
    closeModal={() => setActiveTableDetails(null)}
    pseudonym={activeTableDetails?.content?.pseudonym ?? ""}
    fieldName={activeTableDetails?.content?.fieldName ?? ""}
    tableName={activeTableDetails?.content?.tableName ?? ""}

```

```
        joins={activeTableDetails?.content?.joins ?? []}  
        needComparisonType={false}  
      />  
    </>  
  );  
};  
  
export default Builder;
```