МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н. Каразіна

Факультет: **ННІ Каразінський банківський інститут**
Кафедра: **Інформаційних технологій та математичного моделювання**
Спеціальність: **122 Комп'ютерні науки**
Освітня програма: **Комп'ютерні науки та інформаційні технології в бізнесі**

Група: АК-41Б **денна форма навчання**

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

на тему:

«РОЗУМНІ ФІНАНСИ: СТВОРЕННЯ ФІНАНСОВОГО ДОДАТКУ НА ОСНОВІ МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ»

ЗА НАКАЗОМ № 4601-5/335 ВІД 07 ЛЮТОГО 2025 РОКУ

здобувача вищої освіти **Шабалтаса Владислава Ярославовича**

Робота допущена до захисту в ЕК
протокол кафедри ІТММ № 13 від 31.05.2025

Завідувач кафедри ІТММ
к. п. н., доцент
_____ Н. І. Стяглик

Науковий керівник
к. ф.-м. н. доцент
_____ Л. Д. Філатова

м. Харків 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В. Н. Каразіна

Факультет навчально-науковий інститут "Каразінський банківський інститут"
Кафедра інформаційних технологій та математичного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки та інформаційні технології в бізнесі

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ **Н. І. Стяглик**
08 лютого 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЄКТ)

Шабалтаса Владислава Ярославовича
(прізвище, ім'я, по батькові студента)

1. Тема роботи «Розумні фінанси: створення фінансового додатку на основі методів інтелектуального аналізу даних»

Керівник роботи к. ф.-м. н. доцент Л. Д. Філатова
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 08 лютого 2025 року № 4601-5/335

2. Строк подання студентом роботи 15 травня 2025 року

3. Перелік питань, які потрібно розробити:

У розділі 1: Проаналізувати основи інтелектуальних фінансових додатків.

У розділі 2: Розробити архітектуру додатка.

У розділі 3: Реалізувати прототип інтелектуального додатка.

4. План роботи

№ з/п	Назви етапів роботи
1	Вибір здобувачем теми кваліфікаційної бакалаврської роботи
2	Затвердження плану і завдання кваліфікаційної бакалаврської роботи
3	Здача кваліфікаційної бакалаврської роботи керівнику
4	Підпис кваліфікаційної бакалаврської роботи керівника
5	Підпис кваліфікаційної бакалаврської роботи у нормоконтролера
6	Допуск завідувачем кафедри до захисту кваліфікаційної бакалаврської роботи
7	Захист кваліфікаційної бакалаврської роботи

5. Дата видачі завдання 08 лютого 2025 року

Студент _____ **В. Я. Шабалтас**
підпис ініціали, прізвище

Керівник роботи _ **Л. Д. Філатова**
підпис ініціали, прізвище

РЕФЕРАТ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ
«РОЗУМНІ ФІНАНСИ: СТВОРЕННЯ ФІНАНСОВОГО ДОДАТКУ НА ОСНОВІ МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ»
Шабалтаса Владислава Ярославовича

Кваліфікаційна бакалаврська робота містить 57 сторінка, 7 таблиць, 16 рисунків, список літератури з 14 найменувань.

Об'єктом дослідження є процеси управління персональними фінансами за допомогою методів інтелектуального аналізу даних у цифрових додатках.

Предметом дослідження є методи інтелектуального аналізу даних, технології платформи .NET та фреймворку ML.NET з можливістю застосування їх для підвищення функціональності фінансових додатків.

Мета кваліфікаційної бакалаврської роботи полягає у дослідженні можливостей методів інтелектуального аналізу даних для створення фінансових додатків та їх управління з використанням ML.NET та оцінці ефективності цього підходу порівняно з базовими методами.

Завданнями кваліфікаційної бакалаврської роботи є:

- у першому розділі розглянути теоретичні можливості створення додатків за допомогою методів інтелектуального аналізу даних, проаналізувати ключові частини сучасних фінансових додатків, вивчити й зрозуміти платформу .NET та її фреймворк ML.NET, ознайомитись з вимогами до безпеки і законодавством у цій сфері
- у другому розділі провести щільний аналіз існуючих фінансових додатків, спроектувати архітектуру фінансового додатку з використанням методів інтелектуального аналізу даних, спланувати процес розробки та тестування прототипу.
- у третьому розділі відтворити програмну реалізацію прототипу фінансового додатку з методами інтелектуального аналізу даних, провести порівняльний аналіз базового підходу та обраних інтелектуальних функцій, зробити висновки щодо перспектив даного методу після отримання результатів.

Актуальність дослідження:

пов’язана з розповсюдженням цифрових технологій та швидким розвитком технологій інтелектуального аналізу даних, що дозволяє створювати будь-які додатки, які перевершують можливості базових методів.

За результатами дослідження:

розроблено прототип додатку з використанням методів інтелектуального аналізу даних за допомогою платформи .NET, підтвердження переваги інтелектуального методу над базовими методами.

Практична новизна:

полягає у створенні та реалізації методів інтелектуального аналізу даних на ML.NET у структуру фінансового додатка на платформі .NET, а також у

наглядній демонстрації переваги цього методу.

Одержані результати можуть бути використані у фінансовій сфері, де застосовуються основні методи інтелектуального аналізу даних для створення сучасних фінансових додатків.

КЛЮЧОВІ СЛОВА: ФІНАНСОВИЙ ДОДАТОК, МАШИННЕ НАВЧАННЯ, C#, ML.NET, .NET, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ, УПРАВЛІННЯ ФІНАНСАМИ, АНАЛІЗ ДАНИХ, FINTECH.

ABSTRACT

AT QUALIFICATION BACHELOR WORK

«SMART FINANCE: CREATION OF A FINANCIAL APPLICATION BASED ON DATA MINING METHODS»

Shabaltas Vladyslav

The bachelor's thesis contains 57 pages, 7 tables, 16 drawings, a list of references of 14 titles.

The object of the of research is the processes of personal finance management using data mining methods in digital applications.

The subject of the study is data mining methods, technologies of the .NET platform and the ML.NET framework, with the possibility of using them to improve the functionally of financial applications.

The purpose of a bachelor's thesis is to study the possibilities of data mining methods for creating and managing financial applications using ML.NET and to evaluate the effectiveness of this approach in comparison with basic methods.

The tasks of a bachelor's degree are:

- in the first section to consider the theoretical possibilities of creating applications using data mining methods, analyze the key parts of modern financial applications, study and understand the .NET platform and its ML.NET framework and familiarize yourself with security requirements and legislation in this area;
- in the second section, conduct a thorough analysis of existing financial applications, design the architecture of a financial application using data mining methods and plan the process of developing and testing a prototype;
- in the third section, we will reproduce the software implementation of the financial application prototype with data mining methods, conduct a comparative analysis of the basic approach and selected intelligent functions and draw conclusions about prospects of this method after obtaining the results.

The relevance of the study is associated with the proliferation of digital technologies and the rapid development of data mining technologies which allows creating any application that exceeds the capabilities of basic methods.

According to the results of the research a prototype application using data mining methods using the .NET platform was developed confirming the superiority of the intelligent method over the basic methods.

Main theoretical provisions on the topic of the practical relevance of the constructions is to create and implement ML.NET data mining methods in the structure of a financial application on the .NET platform, as well as to demonstrate the benefits of this method.

The results obtained can be used in the financial sector where the basic methods of data mining are used to create modern financial applications.

KEYWORDS: FINANCIAL APPLICATION, MACHINE LEARNING, C#,

ML.NET, .NET, INTELLIGENT DATA ANALYSIS, FINANCE MANAGEMENT, DATA ANALYSIS, FINTECH.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК, СИМВОЛІВ І ТЕРМІНІВ

ІАД – Інтелектуальний Аналіз Даних

ML – Machine Learning

UI – User Interface

MVP – Minimum Viable Product

СУБД – Системи управління базами даних

FinTech – Financial Technology, сектор фінансових технологій

ВСТУП

Управління особистими фінансами завжди було непростим завданням – хтось веде записи в зошиті, хтось використовує Excel, а хтось узагалі покладається лише на пам’ять. Але в останні роки ситуація стрімко змінюється. З появою сучасних фінансових застосунків усе більше людей переходять до цифрового контролю над своїми витратами й доходами. І це вже не просто зручність, це нова культура поводження з грошима [1-3].

Те, що раніше вимагало багато часу й зусиль, тепер можна зробити за невеликий проміжок часу, наприклад, проаналізувати витрати, спрогнозувати майбутні витрати чи отримати пораду, як краще розподілити бюджет. Саме тут починає працювати те, що раніше здавалося фантастикою – методи інтелектуального аналізу даних з використанням машинного навчання. Вони не тільки допомагають з математичними розрахунками, а й вчаться на прикладах, аналізують поведінку і адаптуються під конкретного користувача й пропонують корисні рішення для втілення у життя.

Актуальність теми дослідження

Зростаючий попит людей на ефективні рішення в області фінансового сектору та збільшення користування технологіями інтелектуального аналізу даних (ІАД) дає змогу працювати більше та краще, завдяки методам інтелектуального аналізу даних, фінансовий сектор стає прозорішим та доступним для новачків у цій сфері. Актуальність обраної теми полягає в тому, що поєднання машинного навчання та методів ІАД відкриває нові можливості для глибокого аналізу наданої інформації та підвищення продуктивності прогнозування фінансових сервісів.

Мета і завдання дослідження

Метою даної кваліфікаційної бакалаврської роботи є **дослідження та практичне застосування методів інтелектуального аналізу даних** для ефективного управління власними фінансами за допомогою відтвореного прототипу додатку на платформі .NET з використанням популярного

фреймворку ML.NET та порівнянням з базовим методом.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. **Дослідити теоретичні основи** створення фінансових додатків за допомогою інтелектуального аналізу даних, зокрема проаналізувати ключові аспекти фінансових додатків.

2. **Проаналізувати наявні фінансові додатки** для визначення методів роботи до власного додатка, розробити архітектуру та описати функціональні вимоги, спланувати процес створення додатку та його тестування.

3. **Здійснити програмну реалізацію** прототипу фінансового додатка з методами інтелектуального аналізу даних, додати обрані функції, провести порівняльну характеристику базового методу та методу з інтелектуальним аналізом, зробити висновки за отриманими результатами тестування щодо практичної необхідності методів.

Об’єкт дослідження – процеси управління фінансами за допомогою цифрових додатків

Предмет дослідження – методи інтелектуального аналізу даних та машинного навчання, зокрема робота з платформою .NET та фреймворком ML.NET у варіанті їх застосування задля збільшення ефективності функціональності фінансових додатків.

Методи дослідження – у роботі використовуються методи системного аналізу, порівняльного аналізу, також методи машинного навчання для побудови програмної реалізації прототипу за допомогою аналітичних моделей, а також статистичний аналіз для обробки та виведення кінцевих кількісних результатів оцінювання.

Отриманні результати можуть бути важливими при розробці сучасних фінансових додатків, допомагаючи збільшити ефективність та підвищити точність подальших досліджень у сфері фінансового сектору з використанням методів інтелектуального аналізу даних та машинного навчання.

РОЗДІЛ 1. ТЕОРІЯ ТА ТЕХНОЛОГІЇ ІНТЕЛЕКТУАЛЬНИХ ФІНАНСІВ

1.1. Основні сучасні фінансові додатки

На сьогодні складно уявити життя людини без смартфона, де зберігається величезна кількість критично важливої для нас інформації. Один з таких важливих видів інформації – фінансові додатки, не має значення, чи це онлайн-банкінг, чи криптовалютна біржа, все, що їх поєднує – це фінанси. Фінансові додатки дуже швидко поширюються світом, ще швидше поширюються онлайн-банкінги з можливістю доступу з телефону без потреби фізично відвідувати відділення [4, 5]. Така тенденція почала зростати з початком COVID-19 та збільшується щодня (Табл. 1.1 – 1.2).

Таблиця 1.1

Вплив пандемії COVID-19 на використання онлайн-банкінгу в Україні

Основний показник	**Кінець 2019 р.**	**Кінець 2020 р.**	**Кінець 2021 р.**	**Кінцевий висновок**
Відсоток дорослого населення, яке користується онлайн-банкінгом	45-50%	55-60%	60-70%+	Через карантинні обмеження відбулось прискорення переходу до онлайн-банкінгу.

Такі таблиці дають змогу побачити темпи росту популярності фінансових додатків за два роки. Питання збільшення кількості цих додатків вже нікого не цікавить, головне, що важливо в популярності додатків – якість та новітні технології. Кожний раз, коли з’являється щось нове на ринку, воно повинно ставати краще та брати з старих платформ щось вже справне та вдосконалювати.

На ринку існує наразі, велике розмаїття фінансових додатків, найпоширенішими з них є додатки мобільного та інтернет-банкінгу. Завдяки простоті роботи цих банків, ми можемо з легкістю дізнатися про наш

поточний стан балансу, подивитись історію транзакцій будь-коли, без проблем оплатити комунальні послуги, за декілька простих кроків відкрити собі депозитний рахунок чи оформити кредит – це все надається нам з можливістю не знаходитись фізично у відділені банків та не спілкуватись з персоналом.

Таблиця 1.2

Вплив пандемії COVID-19 на використання онлайн-банкінгу в Україні

Основний показник	**Кінець 2019 р.**	**Кінець 2020 р.**	**Кінець 2021 р.**	**Кінцевий висновок**
Кількість активних користувачів банківських додатків	Помірне зростання	Значний стрибок росту	Продовження росту	Пандемія стала каталізатором для освоєння мобільного банкінгу
Обсяг безготівкових операцій	Помірне зростання	Швидке збільшення обсягу	Зростання продовжується	Зменшення використання готівки, зручність оплати карткою
Обсяг онлайн-платежів	Помірне зростання	Великий темп росту (Приват Банк звітував про +27%)	Зростання продовжується	Наслідок переходу до онлайн-покупок
Обсяг роботи цифрових гаманців	Малий темп росту	Прискорення освоєння технології	Широке розповсюдження на маси	Зручність безконтактної оплати з телефону без наявності картки

В Україні найпоширенішими з цієї категорії є Приват24, мобільний додаток від Monobank і SenseSuperApp від Sense Bank. Ці додатки мають шалений попит, тому їх вдосконалення навіть не обговорюється. Вони є гарними інструментами для виконання нескладних операцій, але не допомагають глибоко проаналізувати дані чи прогнозувати майбутнє на основі поведінки користувача.

Ці питання допоможуть вирішити додатки для управління особистими фінансами (PFM). Їхня основна концепція – допомогти користувачу отримати повне уявлення про його фінансове становище, якісно зібравши та оформивши інформацію з усіх джерел в одному місці. Ці додатки полегшують рутину справу, допомагають вести облік доходів і витрат,

розподіляти за категоріями, створювати цілі, надавати бонуси за їх виконання. Серед популярних додатків, знаходяться такі: Mint, YNAB, Pocket Guard. За допомогою таких додатків, людині набагато легше розуміти, куди витрачаються гроші, а також збільшує фінансову грамотність. Але, навіть ці додатки вимагають від користувача багато складних дій, такі як, ручне введення готівкових операцій або корекції категоризації, виходячи з цього зрозуміло, що людям похилого віку, ці речі покажуться незрозумілими та вони просто не погодяться працювати з такими сервісами.

Таким чином, всупереч красивим словам у розвитку функціональності фінансових додатків, більшості з них бракує глибокого інтелектуального аналізу, який би допоміг перетворити їх на чудових помічників, без складних пояснень чи налаштувань. Саме цей варіант впровадження інтелектуалізації фінансових послуг є основним напрямком розвитку, що надає нашій темі роботи сенс та важливість.

1.2. Інтелектуальний аналіз даних та машинне навчання у фінансах

Інтелектуальний аналіз даних – це процес виявлення в сирих даних раніше невідомих, нетривіальних, фактично корисних і доступних інтерпретації знань, необхідних для прийняття рішень у різних сферах людської діяльності [6, 7].

Машинне навчання (Machine Learning, ML) – розділ штучного інтелекту, присвячений розумінню та розробленню методів, що дозволяють машинам навчатися завдяки методам, у яких використовують дані для покращення продуктивності комп’ютера в певному наборі завдань [8, 10].

Фінансова область дуже зміна, дані обробляються та оновлюються швидкими темпами у величезних обсягах. Працювати з такою швидкістю завдяки простим статистичним методам неуявно. Тому на допомогу в таких непростих задачах приходять методи ІАД та ML, які дозволяють автоматизувати складні процеси, знаходити можливості для потенційного

прибутку і виявляти приховані ризики.

Загалом, існує велика кількість напрямків де застосовується ІАД та ML у фінансовому секторі. Одна з найвідоміших та найкритичніших областей є виявлення шахрайства. Методи виявлення аномалій допомагають розпізнати підозрілі транзакції, бо коли платіжні системи кожен день обробляють великі кількості транзакцій, неможливо перевірити вручну кожен платіж на ознаки шахрайства, це просто неможливо з погляду швидкості та часу. Тому даний метод швидко орієнтується у просторі і швидко аналізує операції так присвоює їм статус ризику, що одразу сигналізує про необхідність обов’язкової та своєчасної перевірки чи блокування.

Іншою важливою задачею є оцінювання кредитних ризиків, де потрібно точно і швидко розраховувати ймовірності того, що позичальник зможе повернути кредит вчасно і повному об’ємові. Методи класифікації часто використовуються для побудови моделей кредитного аналізу, які перевіряють багато факторів: кредитну історію позичальника, сімейний стан, рівень доходу та інші. Після того як модель побудована, вона присвоює кожному позичальнику свою оцінку, що допомагає автоматизувати процес відсіювання заявок або прийняття угод щодо видачі кредиту та визначення остаточних умов.

У сфері вкладання інвестицій активно просовуються методи прогнозування для передбачення вартості акцій, динаміки ринкових показників тощо. Це основна задача регресійного аналізу, де моделі можуть аналізувати неймовірну велику кількість даних про цінову політику, обсяги торгівлі, новини, пов’язані з економікою та інше, щоб будувати та вдосконалювати моделі для прогнозу.

Методи кластеризації надають можливість пов’язувати клієнтів за потребами, фінансовими поведінками. Такий метод дає змогу створювати персоналізовані маркетингові кампанії, згідно з бажанням клієнтів, пропонувати продукти, які максимально відповідають потребам конкретної людини та адаптувати стратегічні кроки.

За аналізом поведінки користувачів можна побудувати будь-яку систему, яка може пропонувати фінансові послуги, які максимально зможуть зацікавити клієнта, як приклад, це може бути високий відсоток прибутковості з депозиту чи інвестиційні можливості або кредитну картку з цікавими бонусами та перевагами. Все це створюється за допомогою рекомендаційних систем.

Для кращого розуміння методів, які широко використовуються в інтелектуальному аналізі даних та машинному навчанні у фінансах, продемонструємо наступним чином (Рис. 1.1). На ньому наочно представлено основні методи ІАД та їх можливість застосування у інтелектуальному фінансовому додатку, що дає повне розуміння та уявлення для чого методи потрібні та де вони продемонструють себе найкраще.

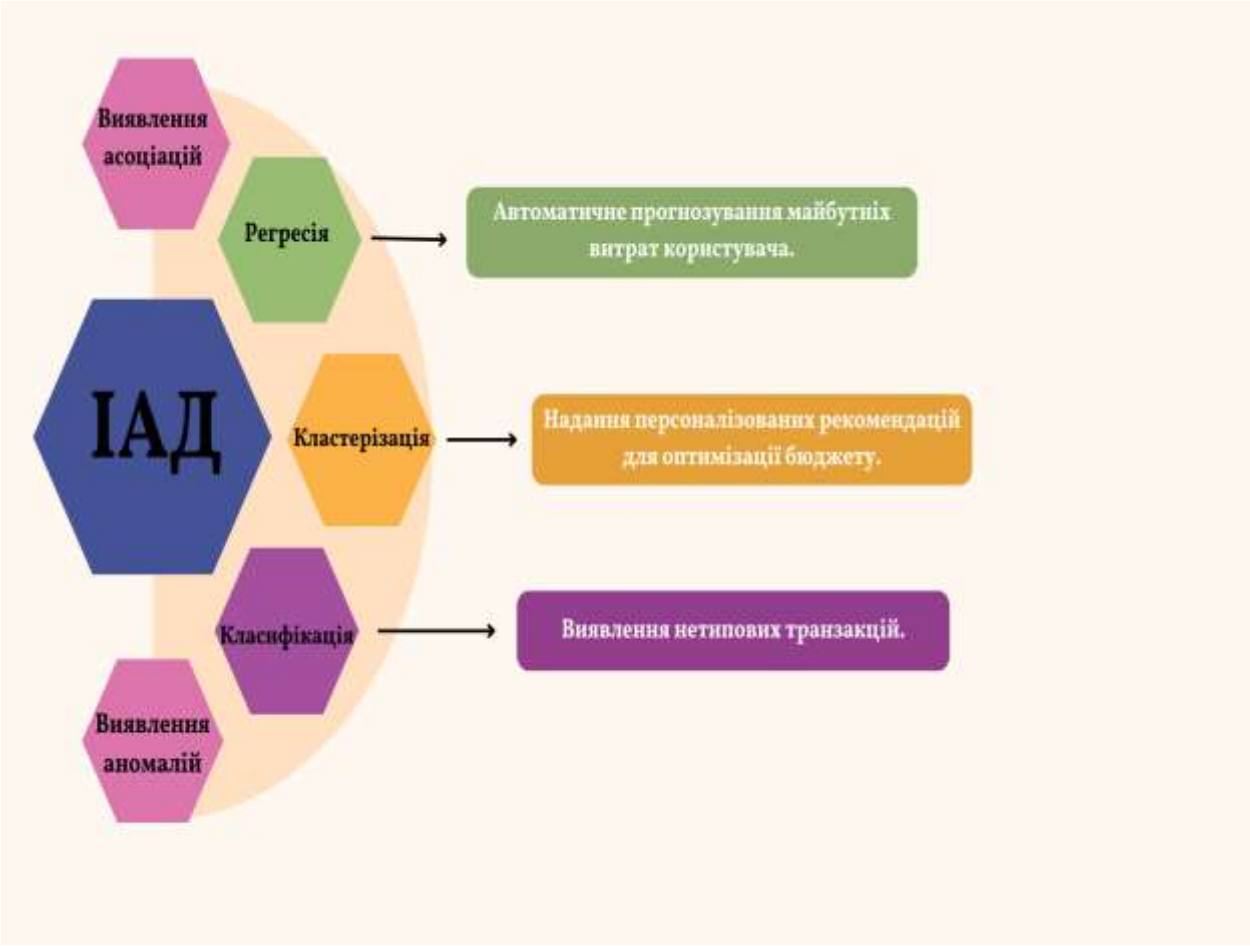

Рис. 1.1. Методи ІАД та їх призначення у фінансовому додатку

Цей рисунок візуалізує, як ІАД слугує основою для застосування його

методів, які допомагають вирішити складні завдання та зробити їх простішими в роботі. Від трьох основних методів прямують стрілки до прямокутників з текстом, що позначають конкретні задачі, які вирішуються в інтелектуальному фінансовому додатку, що розробляється в рамках кваліфікаційної бакалаврської роботи. Ці зв’язки демонструють, який саме метод використовується для реалізації тієї чи іншої задачі.

– Метод «Регресія» використовується для прогнозування майбутніх витрат користувача, тим самим допомагаючи додатку передбачити, де ви можете помилитись, та, ймовірно, скільки грошей витратите, аналізуючи ваші минулі операції.

– Метод «Кластеризація» допомагає додатку надавати поради для бюджету посилаючись на персоналізовані налаштування. Він групує подібні витрати, щоб проаналізувати звички і запропонувати, як краще заощадити на тих чи інших операціях, чи навпаки витратити гроші.

– Метод «Класифікація» використовується для знаходження підозрілих фінансових операцій, які були виконані неодноразово. Метод допомагає додатку помітити транзакції, які виглядають підозрілими або можуть бути ознакою шахрайства з сторони інших людей.

– Метод «Виявлення аномалій» та метод «Виявлення асоціацій» подібні до методів «Класифікація» та «Кластеризація» відповідно.

1.3. Технологічна платформа .NET/ML.NET та забезпечення безпеки фінансових даних

За останній час немало технологічних нововведень додалось у повсякденне життя, які допомагають нам рухатись вперед до повноцінно технологічного світу, тому обирати щось найкраще стало легше. Але якщо питання стосується створення гарного та ефективного фінансового додатка, то потрібно віднестися до цього дуже відповідально, незалежно від складності та розміру задачі. Від того, наскільки правильну буде

проаналізована та підібрана технологія, залежить остаточна реалізація крутого функціоналу, яка охоплює можливості інтелектуального аналізу даних та машинного навчання, які згадували раніше. Не можна забувати про те, що у фінансовій сфері особливий статус має питання захищеності даних, їх окремого надійного зберігання та відповідність додатку до чинних законодавств.

Для створення функціоналу інтелектуального фінансового додатка в даній роботі було обрано технологічну основу, що базується на мові програмування C# та універсальній платформі для багатьох задач .NET. Вибір впав на мову програмування C# непросто так, він зумовлений її сучасністю, зрозумілістю, продуктивністю, підтримкою ООП та наявністю розвиненої екосистеми. Типізація мови сприяє безперебійному виявленню помилок на етапі компіляції, що гарно відображається на загальну надійність коду.

Платформа .NET становить потужне середовище розробки, завдяки підтримці створення різноманітних типів додатків: від серверних рішень (Back-end) з використанням ASP.NET Core до інтерфейсу користувача (Front-end), який реалізовується також як веб додаток або мобільний додаток з використанням відповідних до цієї задачі .NET фреймворків. Це надає перевагу саме мові програмування C#, бо завдяки платформі, можна побудувати весь додаток на єдиній технологічній базі, що одразу спрощує розробку, налагодження коду та подальшу ефективну підтримку на всіх рівнях. Надійність .NET підтверджена багатьма проєктами, серед них: хмарна платформа Microsoft Azure, пошукова система Bing, сайт для розробників Stack Overflow.

Ключовим компонентом для реалізації задуманих функцій машинного навчання в обраних технологічних збірках є основна бібліотека ML.NET. Цей фреймворк від Microsoft стає в пригоді гарним інструментом для .NET розробників, завдяки можливості інтегрувати моделі машинного навчання прямо в додатки, які написані на C#. Це усуває необхідність окремих

платформ або іншим мов програмування для розробки ML-моделей та подальшої складної роботи з інтеграції з основною частиною коду додатка. ML.NET має зрозумілий API для проходження повністю всіх етапів моделі машинного навчання в рамках роботи з .NET середовищем.

Робочий процес з ML.NET складається з логічних, послідовних дій. Першою дією необхідно завантажити дані, яку будуть використовуватися для навчання моделі. Це можуть бути дані про доходи користувача, фінансові транзакції тощо. Наступною дією йде етап підготовки даних. Дані можуть мати різний формат, тому ML.NET безпосередньо допомагає та надає набір компонентів для виконання цих операцій в рамках конвеєра даних.

Після підготовки даних визначається конвеєр навчання, який включає кроки підготовки даних та вибір алгоритму машинного навчання, що повинен повноцінно відповідати поставленій задачі. Далі, на цьому конвеєрі проходить тренування моделі на підготовлених до неї даних. В результаті з’являється навчена модель, яка потім може бути використана в додатку як варіант отримання прогнозів або інших результатів на нових, раніше невідомих даних. ML.NET за своєю особливістю, підтримує різноманітні алгоритми для розв’язання задач, актуальних до нашого проєкту, описаного до кваліфікаційної бакалаврської роботи, включаючи регресії, класифікації, кластеризації, виявлення аномалій.

Розробка інтелектуального фінансового додатка, потребує щільної уваги до характеристик фінансових даних та принципів їх надійного та таємного зберігання. Саме тому постає питання про безпеку, бо фінансові дані стали надзвичайно чутливими: вони містять інформацію про доходи і витрати, активи й борги користувача, що становить основу банківської таємниці клієнта. Ці дані, як правило, мають неосяжний обсяг, маючи особливість постійно оновлюватись, тому іноді можуть бути неповними або містити помилки. Для зберігання таких даних найчастіше використовують системи управління базами даних (СУБД). Вибір постає між реляційними базами даних (PostgreSQL, SQL Server), які забезпечують постійну цілісність

даних та підтримають можливість обробки складних запитів, та NoSQL базами даних (MongoDB), що можуть бути більш масштабованими для великих обсягів неструктурованих даних. Однак, незалежно від типу СУБД, основою є надійність зберігання цих даних, цілісність, можливість резервного копіювання в будь-який момент, відновлення для запобігання втрати даних, а також безпечний доступ.

Забезпечення безпеки у фінансових додатках є основною успіху, а також критичною необхідністю для збільшення довіри з сторони клієнтів, бо користувачі довіряють додатку свої конфіденційні дані, і будь-який злив в загальний доступ інформації може призвести до неймовірних фінансових витрат та стати болючим ударом в репутаційному полі. Основні види безпеки, які необхідно врахувати при розробці, щоб мінімізувати несанкціонований вхід від сторонніх людей чи шахраїв (Табл 1.3).

Таблиця 1.3

Ключові види забезпечення безпеки фінансових додатків

Вид безпеки	**Опис роботи**
Автентифікація та Авторизація	Перевірка особи користувача при вході з багатофакторною перевіркою. Контроль доступу до даних та функцій після входу.
Шифрування даних	Захист важливих даних шляхом їх шифрування як під час передачі, так і під час зберігання.
Контроль вхідних даних	Повна перевірка всіх даних, що надходять до системи.
Безпечне кодування	Дотримання принципів написання коду, що максимально зменшує можливість вразливості. Використання виключно безпечних бібліотек та фреймворків.
Моніторинг та аудит	Ведення журналу дій користувачів та системних подій. Виявлення підозрілої активності в умовах реального часу і проведення розслідування у разі злому.

Джерело: розробка автора

Окрім технічних норм, важливим є дотримання регуляторних вимог. Діяльність фінансових додатків, регулюється законодавством України

(закони про захист персональних даних, про фінансові послуги та державне регулювання ринків фінансових послуг тощо) та міжнародними стандартами (якщо додаток орієнтований на міжнародний ринок або використовує сервіси, що підпадають під їх дію). Відповідність цим вимогам є обов'язковою для безперебійної та легальної роботи додатка та побудови довіри з боку користувачів цього продукту. Це включає і отримання згоди користувача на обробку даних, забезпечення його прав щодо доступу та видалення своєї інформації, а також неминуче реагування на витоки даних відповідно до процедур.

Таким чином, вибір технологічної основи у вигляді C#/.NET/ML.NET надає надійні інструменти для розробки архітектури та функціоналу додатка. Однак не завжди на нашому шляху успіх та прийняття користувачами такого додатка залежить від того, наскільки відповідально розробники поставилися до своєї роботи і забезпечили безпеку фінансової частини, відповідності регуляторним вимогам та врахування етичних аспектів. Ці вимоги є обов'язковою частиною процесу проектування та реалізації фінансової системи подібній тій, що розглядається у кваліфікаційній бакалаврській роботі.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОГО ФІНАНСОВОГО ДОДАТКА ТА ПЛАНУВАННЯ ДОСЛІДЖЕННЯ

2.1. Аналіз ринку наявних рішень та визначення концепції власного додатка

Початок роботи над будь-яким новим продуктом, який потрібно довести до кінцевого стану, тим паче в такій галузі, як FinTech та все, що з ними зв’язано, вимагають глибокого занурення у наявні технології. Неможливо створити щось справді революційне та цінне, не розуміючи, що вже пропонується на ринку користувачам, які їхні поточні потреби задоволені, а де все ще існують недоліки або неочевидні можливості.

Тому перший і основний крок – це ретельний аналіз доступних FinTech додатків на ринку [11, 12]. Такий аналіз допомагає не створювати з нуля проєкт з невідомими для себе технологіями, які можуть погано себе демонструвати на виході, а будувати на досягненнях інших, виявляючи при цьому свою унікальність серед інших.

Проаналізувавши певну низку фінансових додатків, які наразі є популярними та представляють різні підходи у сфері управління особистими фінансами на мобільних платформах [13]. Серед таких додатків були: Monobank, Sense SuperApp, Приват24, Revolut, ПУМБ. В них зацікавило не тільки їхній базовий функціонал та сучасний інтерфейс, а також можливість аналізувати історію операцій та прогнозувати майбутні витрати клієнта, які властиві методам інтелектуального аналізу даних та машинного навчання.

Для кожного з обраних додатків, які були проаналізовані завдяки ознайомленню та вивченню з їх офіційними описами на головних сайтах та в магазинах додатків (Google Play Market), а також на сторінках відомих сайтів, де у статтях зазначено про FinTech ринок, згідно з цим було проведено оцінювання наявності реалізації даних функцій: автоматична категоризація витрат для розподілу транзакцій за певними категоріями (транспорт, розваги,

їжа), персоналізовані рекомендації, що пропонують провести оптимізацію витрат для заощаджень клієнта, виявлення шахрайських дій та попередження про підозрілі транзакції, або звітність, в якій наочно демонструються дані про фінансову активність за допомогою графіків чи діаграм.

Результати детального аналізу функціоналу та наявності методів інтелектуального аналізу даних зібрано та представлено у таблиці 2.1 [14].

Ця таблиця наочно демонструє відмінності у функціональності та рівні інтеграції окремих методів інтелектуального аналізу серед обраних додатків, які найбільш популярні на просторах інтернету в Україні. Представлена таблиця була виконана для спроби оцінити проаналізовані додатки за допомогою джерел доступних в інтернеті, виходячи з цього можемо заявити, що з базовими речами (переказами, виписками, оплатами) сучасні фінансові

Таблиця 2.1

Порівняльний аналіз обраних фінансових додатків на наявність методів інтелектуального аналізу даних

Назва додатка	**Тип додатка**	**Базовий функціонал**	**Наявність ІАД/МН**	**Опис ІАД та МН функцій**	**Обмеження ІАД та МН функцій**
Revolut	Банкінг (PFM)	Платежі, перекази, депозити, інвестиції, бюджет, картки	Бюджет, аналітика та безпека	Аналіз транзакцій, безпека (протоколи)	Відсутня категоризація, прогноз, рекомендації
Monobank	Необанкінг	SEPA, перекази, депозити, кешбек, платежі, картки	Бюджет, аналітика та безпека	Аналіз транзакцій, безпека (протоколи)	Без 2FA, немає категоризації, прогнозу, рекомендації
Приват 24	Класичний державний банкінг	Перекази, вклади, кредити, РКО/бізнес	Часткова (безпека антифрод)	Антифрод: система (включає моніторинг і блокування). Безпека (протоколи і 2FA)	Немає категоризації, прогнозу.

Продовження таблиці 2.1

Sense Bank	Державний банкінг	Картки, платежі, перекази, депозити, кредити, державні програми, інвестиції	(Безпека/ Моніторинг)	Високий рівень безпеки (протоколи, ФГВ, 2FA). Моніторинг(блокування за підозрою	Немає категоризації, прогнозу, бюджету.
ПУМБ	Класичний банкінг	Кредити, перекази під 0%, депозити, картки, кешбек, бізнес	Часткова (безпека)	Високий рівень безпеки (протоколи, 2FA). Високий рівень надійності.	Немає категоризації, бюджету, прогнозу та рекомендацій

Джерело: сайт MixFin [14].

фінансові додатки справляться максимально ефективно. Це вже такий обов'язковий пункт, без якого уявити успішний додаток неможливо. Однак, занурення у їхні можливості, особливо ті, що стосуються використання даних користувача для складних, інтелектуальних функцій, виявило цікаву особливість. Виходячи з аналізу, здається, що величезний потенціал який може бути використаний для реального збільшення швидкості розрахуванн у фінансовому плануванні та контролі, залишається доки нерозкритим.

Як приклад, обрати ту саму автоматичну категоризацію витрат. Вона присутня у більшості додатків, і це вже є великим плюсом. Але її точність бажає кращого, бо вона максимально неопрацьована. Багато користувачів все ще стикаються з необхідністю вручну виправляти категорії транзакцій, підтверджувати нові типи категорій. Це, може втомлювати з часом і псувати враження від такої «технології».

Звертаючи увагу на бюджетування та планування, де ця технологія вважається потрібною та цікавою, вона виглядає дуже сирою та статичною.

Можливість встановлення лімітів на місяць являється гарною додатковою функцією, яка полегшує життя та допомагає заощадити на деяких не потрібних покупках – має добрий вигляд, але для того, щоб

відносити себе до прикладу сучасних фінансових додатків, цього вкрай замало. Для зручної роботи дуже не вистачає підказок, чому саме такі ліміти встановлюються, покроковий план дотримання цього ліміту та опис дій, якщо витрати вийдуть за встановлений ліміт.

Якщо дивитися на наступні технології, то мова йде про персональні рекомендації в яких рідко коли бувають до кінця пропрацьовано поради згідно з зацікавленістю клієнта, які б будувалися на глибокому аналізі конкретних звичок або історії надходження грошей до гаманця. Частіше всього рекомендації просуваються як невеличкі пропозиції щодо порад для користувача, чи навіть отримання налаштованих під кожного клієнта бонусів в додатку.

Особливо, коли найважливішою частиною в фінансовому додатку є безпеки, без якої неможливо уявити доброчесне та захищене функціонування банку з його клієнтами, і тут попри наявність протоколів захисту і антифродових систем, які встановлені у Приват24, якість підтримки захищеності і її активне вдосконалення, викликає багато питань на які нечасто знаходиться часткова або повна відповідь.

Після ознайомлення та аналізу ринку, стає зрозумілим, що для надання цій сфері сучасності потрібен додаток, який зможе ефективно використовувати та вчасно додавати величезний масив даних користувача для активного прогнозування його фінансового майбутнього. Потреба у додатку, на яку довго очікують – стати активним помічником у фінансовому житті людини.

Саме тому ідея створення дипломного проєкту має широкий сенс. Прототип додатка який створюється, запропонує користувачу дещо цікаве, а саме використовувати його потужності як обчислювальний центр, за допомогою якого стане набагато легше прогнозувати наступні витрати, виходячи з даних які взяті з історії транзакцій, надавати рекомендації згідно з вподобаннями людини, які дійсно можуть допомогти заощадити та правильно розподілити кошти.

Підхід до роботи, заснований на основних методах інтелектуального аналізу даних та машинного навчання, допоможе зробити управління особистими фінансами більш прозорими та ефективними для сучасної людини, в якої і без цього багато зайнятості в різних сферах життя. Тому саме ця концепція була довгий час обдумана, повторно проаналізована і згідно з кінцевим рішенням вона стала опорою для подальшого проектування прототипу додатка.

2.2. Архітектура додатка та функціональні можливості

Ідея інтелектуального фінансового додатка стає все більш зрозумілою, виходячи з аналізу ринку фінансового сектора та ознайомлення з тим чого найбільше хоче користувач. Діло дійшло до етапу, коли потрібно розуміти як цю ідею розширити до варіанту відтворення у справжній світ.

При проєктуванні слід розуміти, як найкраще організувати та з’єднати окремі частини додатка, щоб після цього вони працювали воєдино, були простими для розробки і могли постійно змінюватись для масштабності проєкту у майбутньому. Додаток, має окремий блок, який відповідає за окрему частину процесу, і тому щоб робота була злагоджена, потрібно ці блоки оптимізувати, описати та поєднати.

З цією задачею найкраще справляється багатошарова архітектура, яка ділить відповідальність між кожними рівнями націло. Така технологія активно використовується у великих компаніях задля досягнення найкрутішого результату.

Перший шар, що найближчий до користувача – це рівень представлення. Власне кажучи, обличчя нашого додатка – інтерфейс, з яким користувач безпосередньо весь час взаємодіє. Його задача – зрозуміло відобразити для користувача всю інформацію, кнопки, графіки та результати роботи методів інтелектуального аналізу даних – передбачення, рекомендації тощо. І, звісно зібрати від користувача дані (логін, суми витрат,

налаштування) та передати їх далі.

За попереднім шаром криється рівень бізнес-логіки (Business Logic Layer) – це мозок застосунку. Саме тут відбуваються всі важливі обчислення, перевірки, приймаються рішення згідно з встановленими правилами. Тут реалізується логіка управління бюджетами, цілями, обробляються фінансові операції. І найважливіше, саме цей рівень звертається до «інтелектуального» двигуна, аби отримати прогнози чи рекомендації, та після цього передає їх назад рівню представлення.

Далі йде рівень доступу до даних (Data Access Layer). Його задача – бути надійним «посередником» між бізнес-логікою та сховищем усіх наших фінансових даних – базою даних. Цей шар знає, як коректно запросити дані з бази, як їх туди записати чи оновити, і при цьому ховає від очей усі технічні аспекти роботи з конкретною системою.

Після опису попередніх частин багатошарової архітектури, додаємо інформацію про останній важливий складник в нашій роботі – модуль інтелектуального аналізу та машинного навчання. Який виступає у ролі головної частини прототипу застосунку, що дає можливість опрацьовувати неструктуровані дані інтелектуальними методами, зазначеними раніше. Його можливо інтегрувати з бізнес-логікою, але основна задача машинного навчання – підготовка даних, навчання моделей (прогнозування, класифікація, кластеризація) та формування інтелектуальних висновків по запиту клієнта. Цей модуль має на меті активно взаємодіяти з рівнем доступу до даних для отримання давніх транзакцій та інших операцій для аналізу в роботі додатка. Як ці головні елементи взаємодіють між собою для утворення злагодженої системи, наочно представлено на рисунку 2.1.

Щоб перевести ідею з голови в реалізацію на комп’ютері, потрібно мати не тільки кмітливість, а й сміливість відтворити, щось, що раніше не вдавалося нікому. Після основних кроків, коли ретельно вивчено ринок, зрозуміло, чого не вистачає користувачам, саме завдяки цьому окреслено головні цілі майбутньої системи в якій складний функціонал буде розбито на

прості, керовані частини. Питання створення системи, де кожен блок має своє значення та чітке завдання, не здається вже неймовірною задачею, знаючи конкретні дії для досягнення запланованого. Натомість ми зупинились на зарекомендованому себе підході у світі розробки протягом багатьох років – багатошаровій архітектурі, яка дозволяє організувати компоненти, відділяючи ті, що відповідають за взаємодію з користувачем, від тих, що займаються складною розрахунковою роботою та в додаток зберіганням даних.

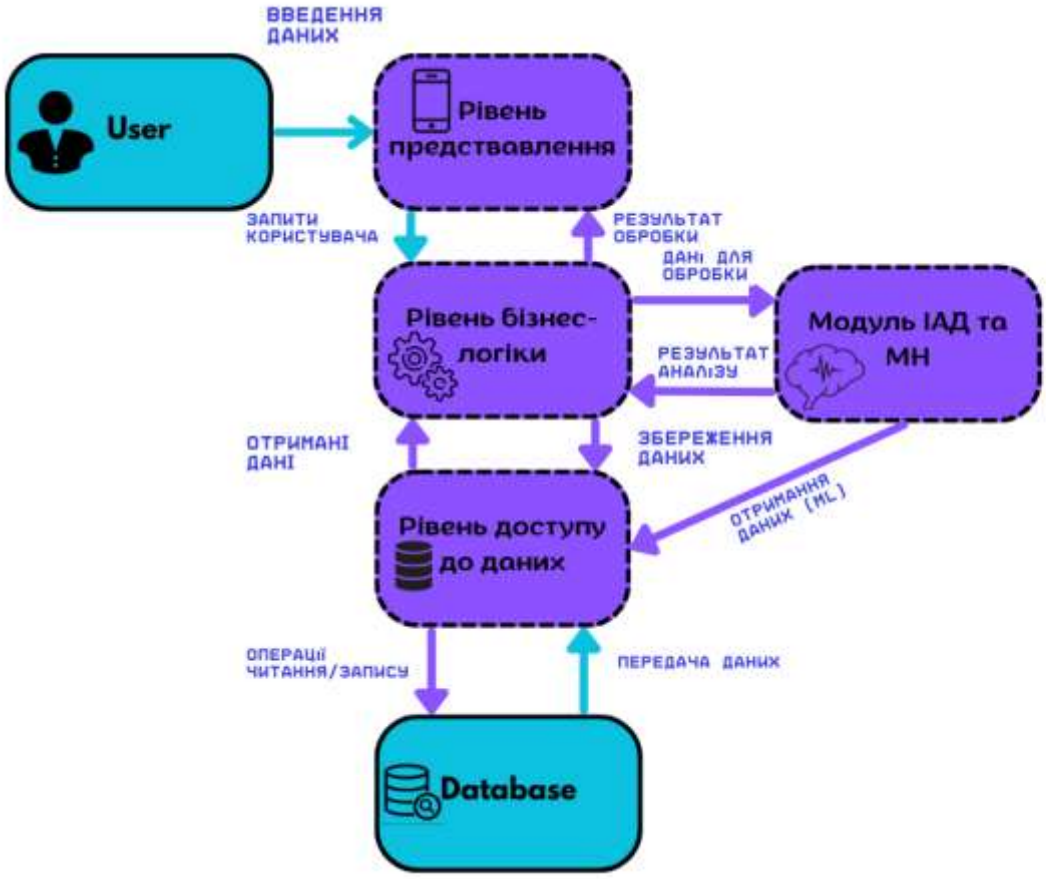

Рис. 2.1. Архітектура інтелектуального фінансового додатка

Розглянувши детально наш рисунок, побудовану систему всередині додатка, можна перейти до описання його функціональних можливостей, які він зможе виконувати задля допомоги користувачеві для швидкої роботи з фінансовими потоками даних.

Рішення які мають бути реалізовані бездоганно з можливістю використовувати методи інтелектуального аналізу даних які мають перетворити додаток на революційний (Табл 2.2):

Таблиця 2.2

Функціональні можливості інтелектуального фінансового додатка

Функціональна можливість	**Реальне застосування**
Керування транзакціями	Ручне введення операцій, перегляд історії доходів чи витрат
Автоматична категоризація витрат	Застосування категоризації до транзакції на основі даних з можливістю коригування
Прогнозування майбутніх витрат	Передбачення витрат користувача на майбутні періоди часу (в нашому випадку — на день) на основі аналізу історичних даних
Виявлення нетипових витрат	Моніторинг фінансових операцій та своєчасне повідомлення про нетипові дії з гаманцем
Звіти та візуалізації	Представлення фінансової активності користувача за запитом

Джерело: розробка автора

За інформацією поданою в таблиці стає зрозумілим те, які функціональні можливості будуть реалізовані в прототипі додатка для повноцінної демонстрації можливостей ІАД та ML в об’єднаній задачі, яка потребує повноцінного рішення, виходячи з теоретичних дописів. Демонстрація цих методів буде додана в невеликому об’ємі для наглядної ефективності даної технології в додатку.

2.3. Проєктування інтелектуальних функцій та їх опис

Після ретельного опрацювання загальної архітектури додатка та визначення його функціональних можливостей, включаючи ключові інтелектуальні методи, що були деталізовані у підрозділі 2.2., постає завдання глибокого занурення у проєктування безпосередньо алгоритмічної

частини та планування всього життєвого циклу розробки програмного продукту. Даний етап є критично важливим, оскільки саме тут визначаються конкретні методи та інструменти реалізації інтелектуального потенціалу системи, а також окреслюються шляхи досягнення поставлених цілей у реальних умовах розробки та впровадження.

Виходячи з обсягу функціоналу, визначеного для реалізації у прототипі фінансового додатка (Табл. 2.1.), ключовий акцент на етапі проєктування інтелектуальних функцій було зроблено на трьох основних задачах, що потребують застосування методів інтелектуального аналізу даних та машинного навчання: автоматична категоризація витрат, прогнозування майбутніх витрат та виявлення нетипових транзакцій. Реалізація цих функцій покликана надати користувачеві ті проактивні інструменти управління фінансами, відсутність або обмежена реалізація яких була виявлена на етапі аналізу ринку (підрозділ 2.1.).

Перш ніж перейти до інтегрування функцій в додаток, потрібно вирішити декілька незрозумілостей. Одна з яких – який метод машинного навчання буде найефективнішим, дізнатися яку саму інформацію ми використовуємо, як обробляємо та, що видаляємо. Після того, як проробили дані пункти, перевіряємо, чи дійсно все працює як нам потрібно, і очікує нас позитивний результат в кінцевому тестуванні чи ні.

Вибір та обґрунтування алгоритмів машинного навчання ML.NET. Для реалізації визначених інтелектуальних функцій було обрано потужну бібліотеку ML.NET, яка надає широкий спектр алгоритмів та інструментів для інтеграції можливостей машинного навчання у .NET додатки. Вибір ML.NET зумовлений тим, що він дозволяє використовувати знання у середовищі .NET (який працює на пряму з мовою програмування C#),

Забезпечує високу продуктивність, можливість розгортання моделей як локально, так і в хмарному середовищі. Для кожної із задач інтелектуального аналізу було здійснено вибір найбільш релевантних алгоритмів з урахуванням типу даних, що будуть оброблятися, та специфіки самої задачі.

Для кожної інтелектуальної функції було розроблено базовий підхід:

– Для Автоматичної категоризації витрат. Де базовим підходом буде система з правилом, що базується на пошуку ключових слів або фраз в описі транзакції та назві торгівельної точки. Наприклад, якщо опис містить «Кава» або назва містить «Кав’ярня», транзакція належить до категорії Кава або Їжа. Цей підхід є простим у реалізації, але менш гнучким та не здатним адаптуватися до нових патернів чи синонімів без ручного оновлення правил

– Для Прогнозування майбутніх витрат. Базовим підходом буде виступати простий метод екстраполяції часових рядів, наприклад, розрахунок середніх витрат за певний попередній період (середні тижневі витрати за останні два тижні) та використання цього середнього значення як прогнозу на наступний період. Цей підхід не враховує складні патерни, сезонність або вплив інших факторів.

– Для Виявлення нетипових транзакцій де базовим підходом буде порогове значення за сумою транзакції. Наприклад, транзакція перестає бути типовою, якщо її сума перевищує певне значення яке зафіксоване до цього або значно відхиляється від середньої суми транзакцій за певною категорією. Цей підхід може бути неефективним для виявлення складніших аномалій, що не пов’язані лише з сумою, або призводити до великої кількості хибних спрацьовувань.

Планування процесу розробки та тестування додатка. Розробка фінансового прототипу додатка проходить поетапно, включаючи в себе етапи, які починаються з створення до кінцевого тестування перед повноцінним запуском. Для цього обирається зручний підхід, в якому робота розподілиться на невеликі частини. Після успішного виконання кожного з них буде готова частина функціоналу, яку можна одразу протестувати й там же покращити за потреби. Такий формат дозволяє рухатися поступово й бачити результат наших дій на кожному окремому результаті, що і є правильним рішенням для даного завдання.

Ключові етапи процесу розробки включатимуть:

1. Налаштування середовища розробки, яке містить в собі встановлення необхідного програмного забезпечення (Microsoft Visual Studio 2022), налаштування .NET та ML.NET, а також системи управління базами даних.

2. Реалізація базового функціоналу, з розробкою компонентів, що відповідають за управління обліковим записом (за спрощеною схемою), роботу з фінансовими джерелами (імітація роботи або імпорт даних), ручне введення та зберігання транзакцій у базі даних.

3. Реалізація Модуля доступу до даних з написанням коду для взаємодії з обраною базою даних, реалізація функцій збереження, отримання, оновлення транзакцій та іншої необхідної інформації.

4. Реалізація Модуля інтелектуального аналізу з розробкою відповідного коду для завантаження та підготовки даних з бази даних (реалізація розроблених трансформацій та виділення головних ознак), реалізація процесу навчання обраних алгоритмів ML.NET для категоризації, прогнозування та виявлення аномалій на тренувальній вибірці, розробка коду для використання навчених моделей для отримання прогнозів, категоризації нових транзакцій та виявлення аномалій у реальному часі.

5. Реалізація Рівня бізнес-логіки з написанням коду, що зв’язує Рівень представлення та Модуль доступу до даних, реалізує логіку управління функціоналом, викликає функції Модуля інтелектуального аналізу та обробляє його результати.

6. Реалізація Рівня представлення (UI), в якому головне – розробка користувацького інтерфейсу (прототип мобільного додатка або веб інтерфейсу), що дозволяє користувачеві взаємодіяти з системою, вводити дані, переглядати історію, отримувати візуалізації, прогнози, рекомендації та сповіщення про аномалії.

Планування тестування залишається частиною процесу після розробки для забезпечення якості програмного продукту. Для цього буде застосовано комплексний підхід до тестування, виходячи з масштабів роботи:

1. Модульне тестування для перевірки коректності роботи окремих функцій та компонентів коду.

2. Інтеграційне тестування для перевірки взаємодії між різними модулями та шарами системи (наприклад, як Рівень бізнес-логіки взаємодіє з Модулем доступу до даних).

3. Функціональне тестування з перевіркою відповідності реалізованого функціоналу сформульованим функціональним вимогам.

4. Тестування інтелектуальних методів з перевіркою коректності роботи функцій прогнозування, категоризації та виявлення аномалій на тестових сценаріях. А також аналіз результатів порівняння з базовим підходом.

Також у процесі розробки буде проводитись налагодження (debugging) для виправлення виявлених помилок. Після завершення реалізації та тестування буде здійснено розгортання прототипу для демонстрації його роботи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ДОДАТКА ТА ОЦІНКА ЕФЕКТИВНОСТІ

3.1. Створення прототипу додатка

У сучасних фінансових додатках головна цінність – не просто зберігати та відображати дані, а допомагати користувачу розуміти їх. Саме для цього й використовуються методи інтелектуального аналізу даних (ІАД) та машинного навчання. У рамках цього проєкту було реалізовано низку функцій, які спрощують взаємодію з фінансами, роблять додаток корисним щодня, а не лише при перегляді витрат.

Усі функції будувалися з урахуванням реальних потреб користувачів. Спочатку було визначено, які завдання найбільше дратують та забирають час. Потім – підібрано найбільш відповідні моделі бібліотеки ML.NET, які можна інтегрувати в C# додаток. Окремо увагу приділено тому, щоб усе працювало швидко та не вимагало доступу до зовнішніх серверів, тобто навчання та аналіз відбуваються локально.

Реалізація прототипу здійснюється шляхом послідовної імплементації створених архітектурних шарів та компонентів, їх інтеграції та наповнення конкретною логікою згідно з визначеними функціональними вимогами. Практичний опис методів може бути розширеним і відрізнятися від кінцевого виду програмного коду.

Центральним елементом ініціалізації є налаштування необхідних залежностей та служб, які будуть доступні протягом життєвого циклу додатка. У представленому коді, роль основного додатка виконує клас App, який успадковується від Xamarin.Forms.Application і є точкою входу для виконання початкових операцій при запуску програми. Першим кроком в конструкторі класу App є ініціалізація ключових компонентів, необхідних для функціонування системи, таких як: ML.NET контексту та налаштування шару доступу до даних. ML.NET контекст (MLContext) є потоково безпечним і стає відправною точкою для всіх операцій ML.NET, таких як завантаження

даних, визначення конвеєрів підготовки даних та навчання моделей. Шар доступу до даних (DataAccess) відповідає за абстрагування роботи з базою даних, надаючи бізнес-логікі зручний інтерфейс для взаємодії зі сховищем даних.

Фрагмент коду, що демонструє ініціалізацію ML.NET контексту та шару доступу даних представлено на рис. 3.1.

```
1  public class App : Application
2  {
3      public static MLContext MlContext { get; private set; }
4      public static ITransformer CategoryModel { get; private set; }
5      public static ITransformer ExpensePredictionModel { get; private
           set; }
6      public static ITransformer AnomalyDetectionModel { get; private
           set; }
7      public static DataAccess DataAccess { get; private set; }
8
9      public App()
10     {
11         // Ініціалізація компонентів
12         MlContext = new MLContext(seed: 1);
13         DataAccess = new DataAccess("finances.db");
14         InitializeDatabase(); // Виклик методу ініціалізації БД
15
16         // ... інші виклики методів ініціалізації та тренування
               моделей ...
17     }
18     // ... інші методи класу App ...
19 }
```

Рис. 3.1. Ініціалізація ML.NET контексту та шару доступу до даних

В даному фрагменті коду створюється статичний екземпляр MLContext з фіксованим значенням seed для забезпечення відтворюваності результатів при навчанні моделі. Також створюється статичний екземпляр класу DataAccess, якому при ініціалізації передається ім’я файлу бази даних.

Використання статичних властивостей для MLContext та DataAccess забезпечує їх доступність з будь-якої частини додатка без необхідності повторного створення або передачі через конструктори, що є поширеним підходом у простих додатках або прототипах.

Наступним важливим кроком ініціалізації додатка є забезпечення наявності та коректної структури бази даних. Це реалізується викликом методу InitializeDatabase(). Цей метод відповідає за створення файлу бази даних, якщо він ще не існує, ти визначення схеми даних шляхом виконання SQL-скриптів для створення необхідних таблиць Accounts, Categories, Transactions. Реалізація схеми бази даних здійснена згідно з проєктуванням, представленим у підрозділі 2.2., включаючи визначення зв’язків між таблицями.

Код методу InitializeDatabase(), що демонструє створення таблиць бази даних, представлено на рис. 3.2.

Реалізація шару доступу до даних (DataAccess) включає методи для виконання SQL-запитів до бази даних SQLite. Ці методи інкапсулюють логіку підключення, виконання команд та обробки результатів, надаючи бізнес-логікі чистий інтерфейс для взаємодії з БД.

Ключовими методами в цьому є ExecuteNonQuery (для виконання команд, що повертають одне значення) та ExecuteQuery<T> (для виконання запитів SELECT та відображення результатів у колекцію об’єктів заданого типу). Ці методи активно використовуються як бізнес-логікою, так і методами навчання моделей для отримання даних.

Реалізація Модуля інтелектуального аналізу включає визначення класів даних (таких як TransactionData, ExpenseData, AnomalyData та відповідних класів для прогнозу), що описують структуру даних, з якими працюють моделі ML.NET. Ці класи використовуються для завантаження даних та створення методів прогнозування.

Код, що ілюструє структуру класів для ML.NET, представлено на рис. 3.3.

```
private void InitializeDatabase()
{
    // Перевірка наявності файлу БД та його створення
    if (!File.Exists("finances.db"))
    {
        SQLiteConnection.CreateFile("finances.db");
    }

    // Створення підключення до БД та виконання команд створення таблиць
    using (var connection = new SQLiteConnection($"Data Source=finances.db;Version=3;"))
    {
        connection.Open();
        using (var command = connection.CreateCommand())
        {
            command.CommandText = @"
                CREATE TABLE IF NOT EXISTS Accounts (
                    Id INTEGER PRIMARY KEY AUTOINCREMENT,
                    Name TEXT NOT NULL,
                    Balance REAL NOT NULL
                )";
            command.ExecuteNonQuery();

            command.CommandText = @"
                CREATE TABLE IF NOT EXISTS Categories (
                    Id INTEGER PRIMARY KEY AUTOINCREMENT,
                    Name TEXT NOT NULL,
                    Type TEXT NOT NULL
                )";
            command.ExecuteNonQuery();

            command.CommandText = @"
                CREATE TABLE IF NOT EXISTS Transactions (
                    Id INTEGER PRIMARY KEY AUTOINCREMENT,
                    AccountId INTEGER NOT NULL,
                    CategoryId INTEGER,
                    Amount REAL NOT NULL,
                    Description TEXT,
                    Date TEXT NOT NULL,
                    FOREIGN KEY(AccountId) REFERENCES Accounts(Id),
                    FOREIGN KEY(CategoryId) REFERENCES Categories(Id)
                )";
            command.ExecuteNonQuery();
        }
    }

    // Додавання демонстраційних даних, якщо БД порожня
    using (var connection = new SQLiteConnection($"Data Source=finances.db;Version=3;"))
    {
        connection.Open();
         using (var command = connection.CreateCommand())
        {
            command.CommandText = "SELECT EXISTS(SELECT 1 FROM Categories LIMIT 1)";
            var categoriesExist = (long)command.ExecuteScalar() > 0; // SQLite returns long for count/exists

            if (!categoriesExist)
            {
                SeedDemoData(); // Виклик методу для наповнення демонстраційними даними
            }
        }
    }
```

Рис. 3.2. Реалізація методу InitializeDatabase()

Ці класи визначають схему вхідних та вихідних даних для моделей ML.NET. Атрибути [LoadColumn] та [ColumnName] використовуються для відображення властивостей C# класів на стовпці даних, що обробляються конвеєром ML.NET.

Після налаштування середовища та визначення класів даних здійснюється тренування моделей машинного навчання. У даному прототипі тренування моделей категоризації, прогнозування витрат та виявлення аномалій відбувається при запуску додатка у відповідних приватних методах класу App. Ці методи реалізують створені конвеєри ML.NET, що включають етапи підготовки даних та навчання обраних алгоритмів.

```
// Клас даних для навчання моделі категоризації та її використання
public class TransactionData
{
    [LoadColumn(0)]
    public string Description { get; set; } // Опис транзакції

    [LoadColumn(1)]
    [ColumnName("Label")] // Мітка для навчання
    public string Category { get; set; } // Категорія транзакції
}

public class TransactionPrediction
{
    [ColumnName("PredictedLabel")] // Прогнозована мітка після моделі
    public string Category { get; set; } // Прогнозована категорія
}

// Клас даних для навчання моделі прогнозування витрат
public class ExpenseData
{
    [LoadColumn(0)]
    public string Month { get; set; } // Місяць
    [LoadColumn(1)]
    public string Category { get; set; } // Категорія
    [LoadColumn(2)]
    [ColumnName("Label")] // Мітка для навчання
    public float TotalAmount { get; set; } // Сума витрат
}

public class ExpensePrediction
{
    [ColumnName("Score")] // Результат прогнозу
    public float PredictedAmount { get; set; } // Прогнозована сума витрат
}

// Клас даних для навчання моделі виявлення аномалій
public class AnomalyData
{
    [LoadColumn(0)]
    public float Amount { get; set; } // Сума транзакції (ознака)
    [LoadColumn(1)]
    public string Category { get; set; } // Категорія (ознака)
}

public class AnomalyPrediction
{
    [ColumnName("Score")] // Оцінка аномальності
    public float Score { get; set; }
    [ColumnName("PredictedLabel")] // Бінарна мітка аномалії
    public bool IsAnomaly { get; set; }
}
```

Рис. 3.3. Фрагмент класів даних для ML моделей

Реалізація навчання моделі автоматичної категоризації витрат, процес навчання якого реалізовано у методі TrainCategoryModel(). Він отримує дані з бази даних (опис транзакції та категорія), потім завантажує їх у IDataView, визначає конвеєр, що включає трансформацію тексту, кодування міток та алгоритм багатоскладової класифікації SdcaMaximumEntropy, і викликає

метод Fit() для навчання моделі.

Код методу TrainCategoryModel, що наочно ілюструє процес, представлено на рис. 3.4.

```
private void TrainCategoryModel()
{
    // Отримання даних для навчання з бази даних
    var transactions = DataAccess.ExecuteQuery<TransactionData>(@"
        SELECT t.Description, c.Name as Category
        FROM Transactions t
        JOIN Categories c ON t.CategoryId = c.Id
        WHERE t.Description IS NOT NULL");

    if (!transactions.Any())
        return;

    // Завантаження даних у формат IDataView для ML.NET
    var trainingData = MlContext.Data.LoadFromEnumerable(transactions);

    // Визначення конвеєру обробки даних та навчання:
    var pipeline = MlContext.Transforms.Text.FeaturizeText("DescriptionFeatures", "Description") // Трансформація
        тексту
        .Append(MlContext.Transforms.Concatenate("Features", "DescriptionFeatures")) // Об'єднання ознак
        .Append(MlContext.Transforms.Conversion.MapValueToKey("Label", "Category")) // Кодування міток
        .Append(MlContext.MulticlassClassification.Trainers.SdcaMaximumEntropy()) // Алгоритм класифікації
        .Append(MlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel", "Label")); // Декодування прогнозу

    // Навчання моделі
    CategoryModel = pipeline.Fit(trainingData);
}
```

Рис. 3.4. Реалізація навчання моделі для автоматичної категоризації витрат

Реалізація навчання моделі прогнозування майбутніх витрат, де навчання реалізовано у методі TrainExpensePredictionModel(). Дані про агреговані місячні витрати завантажуються прямо з БД. Конвейєр включає підготовку ознак (кодування категорій, обробка місяця) та алгоритми регресії FastTreeRegressor. Метод Fit() навчає модель.

Код методу TrainExpensePredictionModel, що ілюструє цей процес представлено на рис. 3.5.

Реалізація навчання моделі виявлення нетипових транзакцій створено у методі TrainAnomalyDetectionModel(). Дані про витрати завантажуються з БД. Конвейєр включає підготовку ознак (кодування категорії, об’єднання з сумою) та алгоритм виявлення аномалій RandomizedPca. Метод Fit() навчає

модель.

```
private void TrainExpensePredictionModel()
{
    // Отримання агрегованих даних про витрати за місяцями по категоріях з БД
    var expenseData = DataAccess.ExecuteQuery<ExpenseData>(@"
        SELECT
            strftime('%Y-%m', Date) as Month,
            c.Name as Category,
            SUM(ABS(t.Amount)) as TotalAmount
        FROM Transactions t
        JOIN Categories c ON t.CategoryId = c.Id
        WHERE c.Type = 'Витрати'
        GROUP BY strftime('%Y-%m', Date), c.Name
        ORDER BY strftime('%Y-%m', Date)");

    if (!expenseData.Any())
        return;

    // Завантаження даних у формат IDataView
    var trainingData = MlContext.Data.LoadFromEnumerable(expenseData);

    // Визначення конвеєру обробки даних та навчання:
    var pipeline = MlContext.Transforms.CopyColumns("Label", "TotalAmount") // Копіювання
        мітки
        .Append(MlContext.Transforms.Categorical.OneHotEncoding("CategoryEncoded",
            "Category")) // Кодування категорії
        .Append(MlContext.Transforms.Text.FeaturizeText("MonthFeatures", "Month")) //
            Обробка місяця як тексту
        .Append(MlContext.Transforms.Concatenate("Features", "CategoryEncoded",
            "MonthFeatures")) // Об'єднання ознак
        .Append(MlContext.Regression.Trainers.FastTree()); // Алгоритм регресії

    // Навчання моделі
    ExpensePredictionModel = pipeline.Fit(trainingData);
}
```

Рис. 3.5. Реалізація навчання моделі прогнозування майбутніх витрат

Код методу TrainAnomalyDetectionModel, що ілюструє цей процес представлено на рис. 3.6.

Після тренування, навчені моделі (об’єкти ITransformer) зберігаються у статичних властивостях класу App для подальшого використання іншими компонентами додатка, зокрема Рівнем бізнес-логіки.

Реалізація Рівня бізнес-логіки продемонстрована у класі FinancialService. Цей клас містить методи, що інкапсулюють основну логіку

функціонування додатка, що взаємодіє з шаром доступу до даних та модулем інтелектуального аналізу. Конструктор класу отримує посилання на необхідні сервіси та навчені моделі зі статичних властивостей класу App.

```
private void TrainAnomalyDetectionModel()
{
    // Отримання даних про витрати (сума, категорія) з БД
    var anomalyData = DataAccess.ExecuteQuery<AnomalyData>(@"
        SELECT
            t.Amount,
            c.Name as Category
        FROM Transactions t
        JOIN Categories c ON t.CategoryId = c.Id
        WHERE c.Type = 'Витрати'
        ORDER BY t.Date");

    if (!anomalyData.Any())
        return;

    // Завантаження даних у формат IDataView
    var trainingData = MlContext.Data.LoadFromEnumerable(anomalyData);

    // Визначення конвеєру обробки даних та навчання:
    var pipeline = MlContext.Transforms.Categorical.OneHotEncoding("CategoryEncoded",
        "Category") // Кодування категорії
        .Append(MlContext.Transforms.Concatenate("Features", "Amount", "CategoryEncoded"))
            // Об'єднання ознак
        .Append(MlContext.AnomalyDetection.Trainers.RandomizedPca()); // Алгоритм виявлення
            аномалій

    // Навчання моделі
    AnomalyDetectionModel = pipeline.Fit(trainingData);
}
```

Рис. 3.6. Реалізація навчання моделі виявлення нетипових транзакцій

Код конструктора FinancialService представлено на рис. 3.7. Методи класу FinancialService реалізують конкретні функціональні можливості додатка, визначені у таблиці 2.1. Наприклад, метод AddTransaction реалізує логіку додавання нової фінансової транзакції. В його реалізації демонструється інтеграція з інтелектуальними функціями.

Перед збереженням у базу даних викликається метод PredictCategory (що використовує навчену модель категоризації) для автоматичного визначення категорії транзакції, а після збереження здійснюється перевірка

на аномалію за допомогою методу IsAnomalyTransaction (що використовує навчену модель виявлення аномалій).

```
public class FinancialService
{
    private readonly DataAccess _dataAccess; // Шар доступу до даних
    private readonly MLContext _mlContext; // ML.NET контекст
    private readonly ITransformer _categoryModel; // Модель категоризації
    private readonly ITransformer _expensePredictionModel; // Модель прогнозування
    private readonly ITransformer _anomalyDetectionModel; // Модель аномалій

    public FinancialService()
    {
        // Отримання посилань на ініціалізовані компоненти з класу App
        _dataAccess = App.DataAccess;
        _mlContext = App.MlContext;
        _categoryModel = App.CategoryModel;
        _expensePredictionModel = App.ExpensePredictionModel;
        _anomalyDetectionModel = App.AnomalyDetectionModel;
    }

    // ... інші методи класу (напр., AddTransaction, PredictCategory тощо) ...
}
```

Рис. 3.7. Конструктор класу FinancialService (Бізнес-логіка)

Код методу AddTransaction, що ілюструє цю інтеграцію, представлено на рис. 3.8.

Методи класу FinancialService, що відповідають за використання навчених ML-моделей для отримання прогнозів (PredictCategory, PredictNextMonthExpenses, IsAnomalyTransaction) та генерації прогнозування (PredictionEngine) на основі відповідних навчених моделей (ITransformer) та передачі вхідних даних для отримання результату. Код цих методів є важливою реалізацією бізнес-логіки, що демонструє застосування інтелектуальних функцій.

Реалізація користувацького інтерфейсу (UI). Кінцева частина цього підрозділу має демонстрацію користувацького інтерфейсу для інтелектуального методу та для базового методу. Для повного розуміння

ситуації надаються скриншоти відтворення інтерфейсу згідно з умовами завдання на рис. 3.9.

```
public void AddTransaction(int accountId, string description, decimal amount, DateTime date)
{
    // Автоматична класифікація категорії за описом
    var categoryId = PredictCategory(description); // Виклик методу прогнозування категорії

    // Збереження транзакції в БД
    _dataAccess.ExecuteNonQuery(@"
        INSERT INTO Transactions (AccountId, CategoryId, Amount, Description, Date)
        VALUES (@AccountId, @CategoryId, @Amount, @Description, @Date)",
        new Dictionary<string, object>
        {
            { "@AccountId", accountId },
            { "@CategoryId", categoryId },
            { "@Amount", amount },
            { "@Description", description },
            { "@Date", date.ToString("yyyy-MM-dd HH:mm:ss") }
        });

    // Оновлення балансу рахунку
    _dataAccess.ExecuteNonQuery(@"
        UPDATE Accounts
        SET Balance = Balance + @Amount
        WHERE Id = @AccountId",
        new Dictionary<string, object>
        {
            { "@AccountId", accountId },
            { "@Amount", amount }
        });

    // Перевірка на аномалію для витрат
    if (amount < 0)
    {
         var categoryName = GetCategoryName(categoryId); // Отримання назви категорії
        if (IsAnomalyTransaction(Math.Abs((float)amount), categoryName)) // Виклик методу виявлення аномалій
        {
            // Логіка сповіщення користувача про аномалію
            System.Diagnostics.Debug.WriteLine($"Аномалія: {amount} грн, {description} ({categoryName})");
        }
    }
}
```

Рис. 3.8. Реалізація методу AddTransaction (Інтеграція категоризації та аномалій)

На початку скриншоту бачимо напис з привітанням користувача який зайде до додатка та побачить його першим. Праворуч від напису знаходиться на фіолетовому кругу форма білого плюса, яка відповідає за швидкий доступ додавання нової операції.

Далі бачимо поточний баланс створений у вигляді картки (форма прямокутника із закругленими кутами). Сума балансу написана жирним текстом у вигляді цифр, привертаючи максимальну увагу до себе, бо це один

з важливих параметрів в інтерфейсі. Нижче видно останню історію транзакцій за якою інтуїтивно зрозуміло, що зелений колір – дохід, а червоний – витрати.

Під балансом знаходиться прямокутник з жовтим елементом та знаком підвищеної уваги для попередження користувача про виявлення нетипових

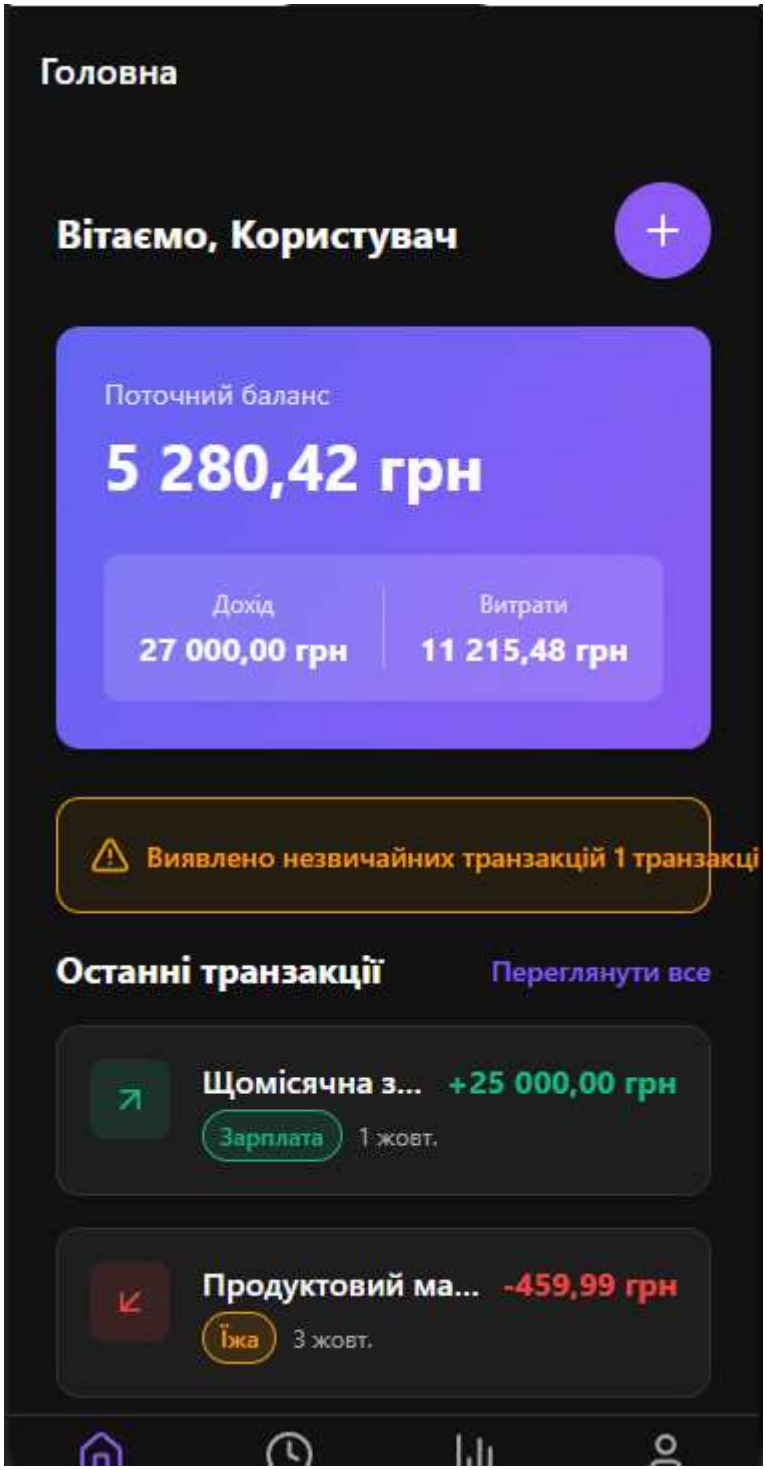

Рис. 3.9. Головний екран прототипу з ІАД

транзакцій. Такий яскравий візуальний акцент допомагає одразу звернути увагу користувача.

Наступне, що ми обговоримо – це розбиття операцій на категорії, які знаходяться на рис. 3.10.

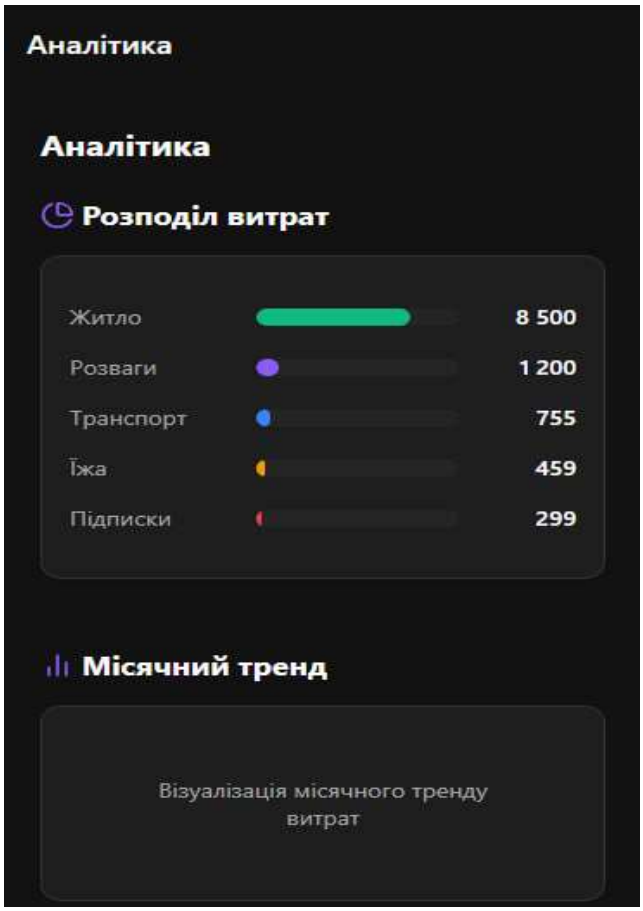

Рис. 3.10. Категорії та розподіл витрат

За даним рисунком чітко бачимо розподіл витрат на кожні категорії, які виділенні окремим кольором задля відмінності між собою. Частина «Аналітика», дозволяє ознайомитись з категоріями в які грошей потрачено найбільше та найменше за допомогою смуги заповнення. Кінцеве заповнення означає вихід за ліміт суми зазначеної раніше. Далі йдуть фінансові поради щодо зменшення витрать, які розглянуті на рис. 3.11.

Дані поради допоможуть зменшити витрати на такі непотрібні речі, як підписки на стрімінгові сервіси задля розваг, чи покупки продуктів у

мережах дорогих магазинів. Завдяки цим порадам можна постійно тримати себе в ліміті який встановлено та не боятись вийти за його межі.

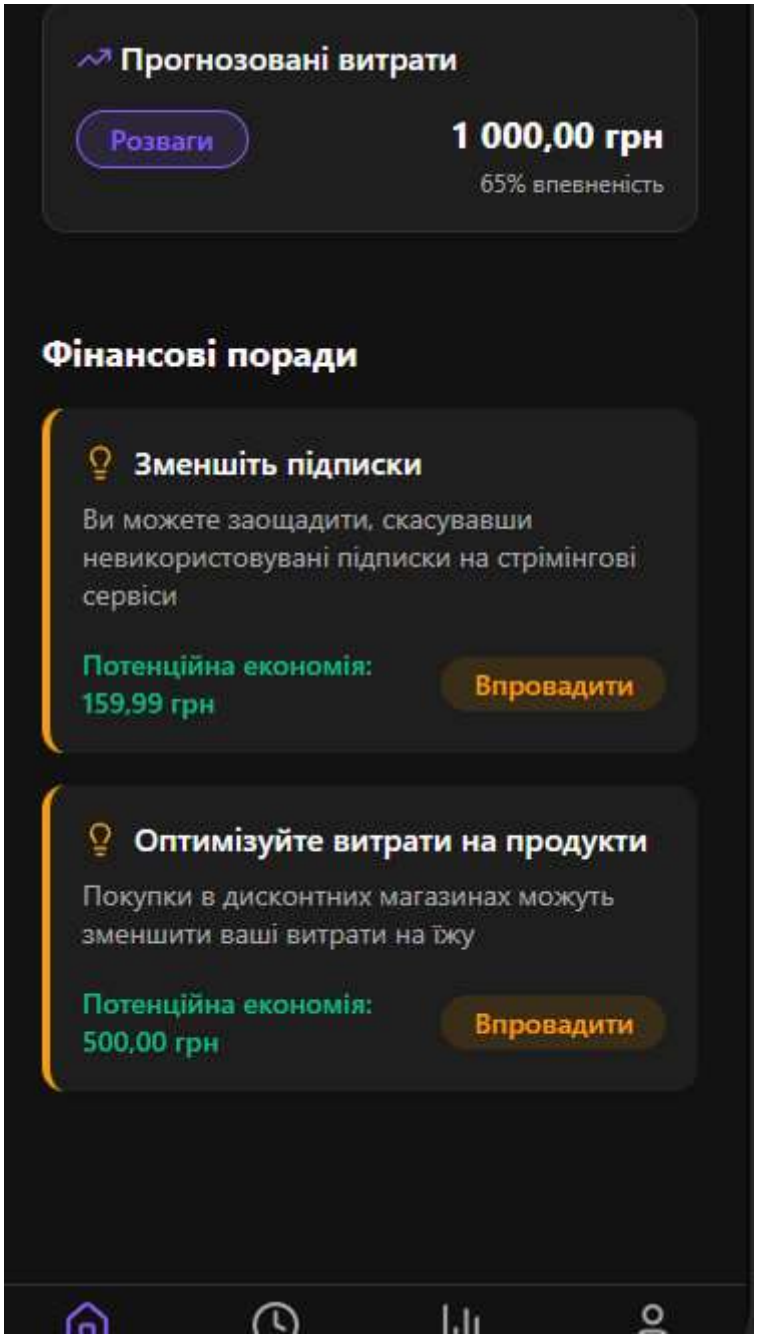

Рис. 3.11. Фінансові поради для зменшення витрат

3.2. Реалізація та оцінювання базового методу

У цьому розділі було реалізовано базовий підхід до аналізу фінансових транзакцій без використання складних алгоритмів машинного навчання або

глибоких методів інтелектуального аналізу даних. Метою роботи є забезпечення початкового рівня автоматизованої обробки фінансових операцій користувача шляхом простих, але ефективних правил і слідування логіки.

Реалізовані Функціональні компоненти:

- Категоризація транзакцій з використанням методу CategorizeTransaction аналізує текстовий опис транзакції та визначає її категорію на основі ключових слів. Наприклад, якщо у назві міститься слово «Аптека», транзакція буде віднесена до категорії «Медицина» . Це реалізовано через просту логіку пошуку ключових слів у нижньому регістрі. Завдяки такому підходу, можливо швидко класифікувати операції, але є обмеження у точності через залежність від формулювань у тексті.

- Прогноз витрат з методом ForecastNextMonth виконує обчислення середнього значення за останні три місяці витрат користувача та використовує його як прогноз на наступний місяць. Ця проста лінійна модель підраховується на основі середнього арифметичного.

- Виявлення аномалій з методом IsAnomaly визначає аномальну транзакцію, якщо її ума вдвічі перевищує середнє значення історичних витрат. Наприклад, при середніх витратах 1200 грн, операція на суму 3000 грн буде визнана аномальною. Цей підхід хоч і простий, але він

допомагає виявити грубі відхилення.

- Фінансові поради в яких використовується метод GetAdvice формує типову пораду залежно від категорії транзакції. Наприклад, для категорії «Транспорт» пропонується зменшити витрати шляхом переходу на громадський транспорт. Поради є статичними, але за своїм призначенням спрямовані на загальну оптимізацію витрат.

Ознайомитись з ефективністю базового методу можна за допомогою таблиці 3.2.

Таблиця 3.2

Оцінювання ефективності базового методу аналізу транзакцій

Критерій	**Висновок**
Простота реалізації	Метод легко реалізується у вигляді правил і базових функцій
Швидкість обробки	Алгоритми працюють миттєво без додаткових обчислень
Зрозумілість для користувачів	Результат аналізу легко розуміється для кожного
Точність категоризації	Залежить від ключових слів, не підтримуючи неоднозначні випадки
Адаптивність до змін	Метод не враховує індивідуальні звички користувача
Виявлення аномалій	Базується лише на простому пороговому порівнянні
Прогнозування витрат	Враховується лише середнє значення без трендів і сезонності

Джерело: розробка автора.

Реалізований базовий метод демонструє мінімально життєздатний функціонал системи аналізу фінансових транзакцій. Він може бути використаний як стартова точка в умовах обмежених ресурсів чи розробки MVP проєкту для попереднього ознайомлення перед використанням складних методів інтелектуального аналізу даних.

Використання базових методів для побудови програмного коду показано на рис.3.12.

На цьому рисунку зображений вивід в консольну частину опису транзакції та її суму. Виходячи з даних, які були записані, результат аналізу доходить висновку, що за програмним кодом «Аптека» належить до категорії «Медицина», прогноз витрат на наступній місяць згідно з середнім значенням дорівнює 1216.6666 грн. Сума яка введена в параметр транзакції (6500) викликає аномалію і згідно з цим поступає рекомендаційна порада з підготовленим текстом згідно з наданими даними.

Відносно з даною оцінкою базового методу, зроблена додаткова перевірка з використанням інших даних, які структуруються у категорії зазначені в програмному коді на рис. 3.13.

```
    public static void Main()
    {
        // Тестова транзакція
        Console.WriteLine("Введіть опис транзакції:");
        string description = Console.ReadLine();

        Console.WriteLine("Введіть суму транзакції:");
        double amount = double.Parse(Console.ReadLine());

        // Приклад історії витрат (останні 3 місяці)
        List<double> history = new List<double> { 1200, 1350, 1100 };

        string category = CategorizeTransaction(description);
        double forecast = ForecastNextMonth(history);
        bool isAnomaly = IsAnomaly(amount, history);
        string advice = GetAdvice(category);

        Console.WriteLine($"\n Результати аналізу:");
        Console.WriteLine($"Категорія: {category}");
        Console.WriteLine($"Прогноз витрат на наступний місяць: {forecast} грн");
        Console.WriteLine($"Аномалія: {(isAnomaly ? "Так" : "Ні")}");
        Console.WriteLine($"Порада: {advice}");
    }

```

```
Введіть опис транзакції:
аптека
Введіть суму транзакції:
6500

 Результати аналізу:
Категорія: Медицина
Прогноз витрат на наступний місяць: 1216.6666666666667 грн
Аномалія: Так
Порада: Регулярна перевірка здоров'я допоможе уникнути раптових витрат.
```

Рис. 3.12. Використання базових методів для написання коду

Було змінено частину коду: List<double> history = new List<double> {1500, 1700, 2500};. Використовуючи інші значення, відміні від першого тестування з більшим діапазоном від 1500 до 2500 відповідно. Опис транзакції змінився на варіант з покупкою бензину на суму в 5500 грн і за результатом, в нас категорія: Транспорт, прогноз витрат на наступний місяць за середнім значенням – 1900 грн. Сума залишається бути аномальною, рекомендації додані.

Розглядаємо останній варіант, де інформація щодо опису транзакції не буде належати до певної категорії в коді і сума її буде меншою за зазначену (рис. 3.14)

```
13          Console.WriteLine("Введіть суму транзакції:");
14          double amount = double.Parse(Console.ReadLine());
15
16          // Приклад історії витрат (останні 3 місяці)
17          List<double> history = new List<double> { 1500, 1700, 2500 };
18
19          string category = CategorizeTransaction(description);
20          double forecast = ForecastNextMonth(history);
21          bool isAnomaly = IsAnomaly(amount, history);
22          string advice = GetAdvice(category);
23
24          Console.WriteLine($"\n Результати аналізу:");
```

```
Введіть опис транзакції:
бензин
Введіть суму транзакції:
5500

 Результати аналізу:
Категорія: Транспорт
Прогноз витрат на наступний місяць: 1900 грн
Аномалія: Так
Порада: Розгляньте варіанти громадського транспорту для економії.
```

Рис. 3.13. Використання інших даних для оцінювання роботи базового методу

```
31      public static string CategorizeTransaction(string description)
32      {
33          description = description.ToLower();
34
35          if (description.Contains("магазин") || description.Contains("супермаркет"))
36              return "Продукти";
37          if (description.Contains("аптека") || description.Contains("ліки"))
38              return "Медицина";
39          if (description.Contains("заправка") || description.Contains("бензин"))
40              return "Транспорт";
41          if (description.Contains("кафе") || description.Contains("ресторан"))
42              return "Харчування поза домом";
43
44          return "Інше";
45      }
46
47      public static double ForecastNextMonth(List<double> history)
48      {
```

```
Введіть опис транзакції:
іграшки
Введіть суму транзакції:
300

 Результати аналізу:
Категорія: Інше
Прогноз витрат на наступний місяць: 600 грн
Аномалія: Ні
Порада: Перевіряйте свої витрати регулярно для кращого контролю.
```

Рис. 3.14. Використання нестандартних даних в базовому методі

3.3. Аналіз ефективності інтелектуального та базового методів

Порівняємо два реалізованих підходи для аналізу фінансових транзакцій. Метод базовий і метод, що використовує інтелектуальний аналіз даних і машинне навчання. В обох випадках задача стоїть однакова: категоризація транзакцій, прогнозування витрат та виявлення аномалій з рекомендаціями.

Базовий підхід реалізовано у вигляді простих функцій на C#, які працюють за принципом ключових слів та арифметичних розрахунків. Наприклад, для визначення категорії витрати застосовуються прості умовні оператори: якщо в описі транзакції є слово аптека – це належить до медицини, якщо бензин, то транспорт і так далі. Прогнозування витрат ґрунтується лише на середньому значенні минулих періодів. Аномалії виявляється порівнянням поточної суми з подвійним середнім – якщо перевищено, значить ситуація нетипова.

Даний метод відкриває можливості створення простих у реалізації методів, коли потрібно створити швидко і вчасно роботу – він підходить краще за все, тим паче даючи зрозумілу відповідь для користувача, не маючи складних алгоритмів обчислення. Такий варіант підходить для тих, кому важлива мінімалістична аналітика без зайвих складних речей.

Другий підхід – це використання машинного навчання та інтелектуального аналізу даних. Тут модель навчена на реальних прикладах транзакцій, що дозволяє їй краще розуміти контекст і точно класифікувати витрати за складними або незвичними описами. Прогноз витрат базується на алгоритмах регресії, що враховують не лише середнє, а й тренди, сезонність та такі фактори, як: дохід, кількість покупок, дні тижня тощо.

Аномалії виявляються вже не просто порогом деякої суми, а через аналіз відхилення від поведінкових шаблонів користувача. Система може враховувати, що для однієї людини витрата розміром в 3000 грн – це цілком нормально, а для іншої – сигнал зменшити витрати. Також на основі категорії

витрати та історії поведінки можна надавати більш персоналізовані поради.

Такі підходи вже використовуються в банківських додатках, фінансових помічниках. Вони складніші в реалізації, але результат – ширший, точніший і корисніший для кінцевого користувача.

Ось один приклад: користувач протягом трьох місяців витрачає кожний місяць приблизно 5000 грн на харчування. Одного місяця з'являється транзакція на 9500 грн і базовий метод одразу сигналізує про аномалію, а інтелектуальний метод дивиться глибше і бачить, що було багато маленьких транзакцій і ця одна велика просто замінили їх, після чого робиться висновок щодо зміни поведінки, але в межах норми

Другий приклад пов'язаний з використанням порад. В базовому методі для цього можна просто надати скриптовану фразу щодо великих витрат на розваги чи на ліки. Але інтелектуальний аналіз даних допоможе виявити, що саме припадає на витрати для розваг і чому порівняно з попереднім місяцем зросли витрати на цю категорію. Фраза буде інакшою, в стилі пояснення, що витрати зросли порівняно з попередніми місяцями та потрібно ретельно перевірити які підписки на розваги для нас більше не є актуальними, виходячи зі зменшення активності перегляду телепередач, гра в ігри тощо. Наочне порівняння згідно з повноцінним описом кожного методу табл. 3.3.

Таблиця 3.3

Табличне порівняння методів за показниками

Показник	**Базовий метод**	**Інтелектуальний метод**
Реалізація	Ключові слова	Машинне навчання
Гнучкість відносно параметрів	Фіксовані правила	Адаптація під дані задані користувачем
Точність категоризації	Невелика при незвичних словах	Висока, навіть коли формулювання інакші
Прогноз витрат	Середнє значення за останні періоди	Регресійна модель з вирахуванням
Виявлення аномалій	Порівняння з середнім значенням	Аналіз шаблонів поведінки
Надання рекомендацій	Загальні фрази	Персоналізовані рекомендації
Швидкість	Миттєва	Висока після навчання моделі

Джерело: розробка автора

Обидва підходи мають повне право на існування, але вони розв’язують різні задачі. Базовий підхід – це старт, зручний для прототипу, MVP чи демонстраційної версії додатка. Він не потребує часу на навчання, дає миттєвий результат і підходить для тих, хто хочу побачити загальну картину за мінімальну кількість часу.

Інтелектуальний підхід – це вже серйозний інструмент для глибокого аналізу, який реально допомагає керувати фінансами. Його сила – в адаптації, точності та персоналізації. Такі методи використовуються в сучасних банківських системах, особистих фінансових помічниках і навіть в автоматизованих системах прийняття рішень.

Для нашого прототипу вирішено було реалізувати обидва варіанти для демонстрування різниці між ними. Це дозволяє не просто обрати кращий варіант, а побачити, як з часом можна еволюціонувати від простого інтелектуального.

ВИСНОВКИ

У межах даного дослідження було здійснено глибокий аналіз сучасних підходів до розробки інтелектуальних фінансових систем, які поєднують у собі інструменти машинного навчання, аналізу даних і програмування на мові C#. Робота охоплює не лише теоретичні засади, але й практичну реалізацію повноцінного прототипу додатка з інтелектуальними методами, що підтвердило можливість створення ефективних рішень на перетині фінансів і сучасних IT-технологій.

На першому етапі дослідження було розглянуто, які саме фінансові додатки сьогодні мають найбільше значення для користувачів – від мобільних банків до сервісів ведення бюджету й автоматичного аналізу витрат. Особливу увагу приділено ролі інтелектуального аналізу даних, адже саме він дозволяє системам не просто зберігати інформацію, а й розуміти її, виявляючи при цьому аномалії, прогнозувати витрати, давати корисні поради користувачеві. Вивчення платформи .NET і бібліотеки ML.NET дало змогу побачити, що сучасна екосистема C# відкриває широкі можливості для створення гнучких і потужних моделей машинного навчання прямо всередині застосунку.

Другий розділ роботи був присвячений проєктуванню і плануванню самого фінансового додатка. Було проведено аналіз ринку, що дозволило сформулювати чітке бачення продукту: додаток, який не просто збирає дані, а й допомагає користувачу краще зрозуміти свої фінансові звички та керувати витратами. Ретельне проєктування архітектури дозволило організувати структуру застосунку так, щоб кожна його частина – від збору даних до аналітики – працювала злагоджено та логічно, не викликаючи ускладнень. Було окреслено ключові функції, зокрема: класифікація транзакцій, виявлення
аномальних витрат, прогнозування майбутніх витрат і надання персоналізованих порад.

У третьому розділі було реалізовано порівняння базового методу з інтелектуальним, підставлені їх відмінності та позитивні речі. За допомогою багаторівневої архітектури було розділено прототип додатка на логічні компоненти, які забезпечують зручність у підтримці та масштабуванні. Також було описано та продемонстровано на рисунках яскравий користувацький інтерфейс з навігацією та методами інтелектуального аналізу зазначеного у завдані.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Топ-5 фінансових додатків для управління бюджетом. [Електронний ресурс]. – Режим доступу: https://ideabank.ua/uk/experts/top-5-finansovykh-dodatkiv-dlya-upravlinnya-byudzhetom#1

2. П'ять українських додатків для контролю витрат. [Електронний ресурс]. – Режим доступу: https://rubryka.com/article/dlya-kontrolyu-vytrat/

3. Топ додатків для обліку фінансів. [Електронний ресурс]. – Режим доступу: https://e-zubkovskiy.com/blog/top-dodatkiv-dlya-obliku-finansiv

4. Мобільні застосунки для планування бюджету та контролю витрат: які найбільш популярні. [Електронний ресурс]. – Режим доступу: https://epravda.com.ua/publications/2023/02/22/697326/

5. 10 найкращих інструментів та програм для управління фінансами (травень 2025 р.). [Електронний ресурс]. – Режим доступу: https://www.securities.io/uk/financial-management-platforms/

6. Мокін В. Б., Дратований М. В. Наука про дані: машинне навчання та інтелектуальний аналіз даних. Вінниця : ВНТУ, 2024. – 263 с.

7. Ланде Д.В., Субач І.Ю., Бояринова Ю.Є. Основи теорії і практики інтелектуального аналізу даних у сфері кібербезпеки. ІСЗЗІ КПІ ім. Ігоря Сікорського», 2018. - 300 с.

8. Machine Learning, ML. [Електронний ресурс]. – Режим доступу: https://www.it.ua/knowledge-base/technology-innovation/machine-learning

9. Дроздов Д. А. Машинне навчання та штучний інтелект: можливості та виклики / Д. А. Дроздов, Н. С. Калайда // Радіоелектроніка та молодь у XXI столітті : матеріали 29-го Міжнар. молодіж. форуму, 16–19 квітня 2025 р. – Харків : ХНУРЕ, 2025. – Т. 6 – С. 410-412. [Електронний ресурс]. – Режим доступу: https://openarchive.nure.ua/handle/document/30890

10. Дослідження особливостей використання Data Mining у сфері банківських послуг // Науковий вісник ХНУ. Серія: Економіка. – 2020. – № 6. – С. 125–130.

11. Vartsaba V., Zaslavska O. FinTech industry in Ukraine: problems and prospects for the implementation of innovative solutions Baltic Journal of Economic Studies. 2020. Vol. 6, No. 4. pp.46-55. DOI: https://doi.org/10.30525/2256-0742/2020-6-4-46-55

12. Стратегія розвитку фінтеху в Україні до 2025 року. Сталий розвиток інновацій, кешлес та фінграмотність. [Електронний ресурс]. – Режим доступу: https://bank.gov.ua/admin_uploads/article/Strategy_finteh2025.pdf?v=4

13. Веселий І. О. Аналіз цифровізації роздрібного сегменту української банківської системи. Наукові праці ДонНТУ. Серія: «Економічна» №2(28), 2023. С. 76-83.

14. Mixfin.com. Фінансові послуги та інструменти [Електронний ресурс]. – Режим доступу: https://mixfin.com/ua/

ДОДАТОК

КОД ПРОГРАМИ З UI-ПРИКЛАДУ

Файл AppShell.xaml (Основна навігація)

```
<Shell xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:YourAppName"
    x:Class="YourAppName.AppShell"
    BackgroundColor="#121212"
    TitleColor="White"
    UnselectedColor="Gray"
    SelectedColor="White">

  <TabBar>
    <ShellContent Title="Головна"
                  Icon="home_icon.png" ContentTemplate="{DataTemplate local:HomePage}" />
    <ShellContent Title="Історія"
                  Icon="history_icon.png"
                  ContentTemplate="{DataTemplate local:HistoryPage}" /> <ShellContent
Title="Аналітика"
                  Icon="analytics_icon.png"
                  ContentTemplate="{DataTemplate local:AnalyticsPage}" />
    <ShellContent Title="Профіль"
                  Icon="profile_icon.png"
                  ContentTemplate="{DataTemplate local:ProfilePage}" /> </TabBar>
</Shell>
```

AppShell.xaml.cs (Реєстрація маршруту)

```
namespace YourAppName;

public partial class AppShell : Shell
{
   public AppShell()
   {
```

```
            InitializeComponent();

            // Реєструємо маршрут для сторінки деталей транзакції
            // Навіть без ViewModel ми використовуємо Shell для навігації та передачі параметрів
            Routing.RegisterRoute("TransactionDetailsPage", typeof(TransactionDetailsPage));

            // TODO: Зареєструйте інші маршрути, якщо є
        }
    }
```

HomePage.xaml (Головна сторінка)

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:YourAppName"
             x:Class="YourAppName.HomePage"
             Title="Головна"
             BackgroundColor="#1E1E1E">
    <ScrollView>
        <VerticalStackLayout Spacing="15" Padding="15">

            <Grid ColumnDefinitions="*,Auto">
                <Label Text="Вітаємо, Користувач" FontSize="24" TextColor="White" VerticalOptions="Center"/>
                <Button Grid.Column="1" Text="+" FontSize="20" FontAttributes="Bold" BackgroundColor="#333333" TextColor="White" CornerRadius="25" WidthRequest="50" HeightRequest="50"
                        Clicked="AddTransactionButton_Clicked"/>
            </Grid>

            <Frame CornerRadius="15" Padding="20" Background="{LinearGradientBrush EndPoint='1,0'}">
                <Frame.Background>
                    <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
                        <GradientStop Color="#8A2BE2" Offset="0.0" />
                        <GradientStop Color="#4B0082" Offset="1.0" />
                    </LinearGradientBrush>
                </Frame.Background>
                <VerticalStackLayout Spacing="10">
                    <Label Text="Поточний баланс" FontSize="16" TextColor="#E0E0E0"/>
                    <Label Text="5 280,42 грн" FontSize="36" FontAttributes="Bold" TextColor="White"/>
                    <Grid ColumnDefinitions="*,*">
                        <VerticalStackLayout>
                            <Label Text="Дохід" FontSize="14" TextColor="#E0E0E0"/>
                            <Label Text="27 000,00 грн" FontSize="18" TextColor="White"/>
                        </VerticalStackLayout>
                        <VerticalStackLayout Grid.Column="1" HorizontalOptions="End">
                            <Label Text="Витрати" FontSize="14" TextColor="#E0E0E0"/>
                            <Label Text="11 215,48 грн" FontSize="18" TextColor="White"/>
                        </VerticalStackLayout>
```

```
            </Grid>
        </VerticalStackLayout>
    </Frame>
    <Frame CornerRadius="10" Padding="10" BorderColor="#FFA500"
BackgroundColor="#40FFA500" x:Name="UnusualTransactionsWarningFrame"
IsVisible="True"> <HorizontalStackLayout Spacing="10">
            <Image Source="warning_icon.png" HeightRequest="24" WidthRequest="24" />
            <Label Text="Виявлено незвичайних транзакцій: 1" TextColor="#FFA500"
VerticalOptions="Center"/>
        </HorizontalStackLayout>
    </Frame>

    <Grid ColumnDefinitions="*,Auto" Margin="0,10,0,0">
        <Label Text="Останні транзакції" FontSize="20" FontAttributes="Bold"
TextColor="White"/>
        <Label Grid.Column="1" Text="Переглянути все" TextColor="#ADD8E6"
TextDecorations="Underline" VerticalOptions="Center">
            <Label.GestureRecognizers>
                <TapGestureRecognizer Tapped="ViewAllTransactionsLabel_Tapped"/>
            </Label.GestureRecognizers>
        </Label>
    </Grid>

    <CollectionView x:Name="RecentTransactionsCollectionView" ItemsSource="{Binding
RecentTransactions}" SelectionMode="Single"
                    SelectionChanged="RecentTransactionsCollectionView_SelectionChanged">
<CollectionView.ItemTemplate>
            <DataTemplate>
                <Frame Padding="15" Margin="0,5" CornerRadius="10"
BackgroundColor="#2C2C2C">
                    <Grid ColumnDefinitions="Auto,*,Auto" RowDefinitions="Auto,Auto"
ColumnSpacing="10">
                        <Image Source="{Binding TransactionTypeIcon}" HeightRequest="40"
WidthRequest="40" VerticalOptions="Center" Grid.RowSpan="2"/>
                        <Label Grid.Column="1" Text="{Binding Description}"
FontAttributes="Bold" TextColor="White" LineBreakMode="TailTruncation"/>
                        <Label Grid.Column="2" Text="{Binding Amount, StringFormat='{0:N2}
грн'}" TextColor="{Binding AmountColor}" FontAttributes="Bold" FontSize="16"
VerticalOptions="Center" HorizontalTextAlignment="End" Grid.RowSpan="2"/>

                        <HorizontalStackLayout Grid.Column="1" Grid.Row="1" Spacing="5"
VerticalOptions="Center">
                            <Border StrokeShape="RoundRectangle 5" BackgroundColor="{Binding
CategoryTagColor}" Padding="6,3">
                                <Label Text="{Binding Category}" TextColor="White"
FontSize="10"/>
                            </Border>
                            <Label Text="{Binding Date, StringFormat='{0:d MMM}'}"
FontSize="12" TextColor="Gray"/>
                        </HorizontalStackLayout>
                    </Grid>
                </Frame>
```

```
            </DataTemplate>
        </CollectionView.ItemTemplate>
    </CollectionView>
    <Frame CornerRadius="10" Padding="15" BackgroundColor="#2C2C2C"
Margin="0,10,0,0">
        <VerticalStackLayout Spacing="8">
            <HorizontalStackLayout Spacing="10">
                <Image Source="chart_bar_icon.png" HeightRequest="24"
WidthRequest="24"/> <Label Text="Прогнозовані витрати" FontSize="18"
FontAttributes="Bold" TextColor="White"/>
            </HorizontalStackLayout>
            <Grid ColumnDefinitions="Auto,*,Auto">
                <Border StrokeShape="RoundRectangle 8" BackgroundColor="Purple"
Padding="10,5">
                    <Label Text="Розваги" TextColor="White" FontSize="14"/>
                </Border>
                <Label Grid.Column="2" Text="1 000,00 грн" FontSize="20"
FontAttributes="Bold" TextColor="White" VerticalOptions="Center"
HorizontalOptions="End"/>
            </Grid>
            <Label Text="65% впевненість" FontSize="12" TextColor="LightGray"
HorizontalOptions="End"/>
        </VerticalStackLayout>
    </Frame>
    <Label Text="Фінансові поради" FontSize="20" FontAttributes="Bold"
TextColor="White" Margin="0,15,0,5"/>
    <Frame CornerRadius="10" Padding="15" BackgroundColor="#2C2C2C">
        <VerticalStackLayout Spacing="10">
            <HorizontalStackLayout Spacing="10">
                <Image Source="lightbulb_icon.png" HeightRequest="24" WidthRequest="24"/>
<Label Text="Зменшіть підписки" FontAttributes="Bold" FontSize="16" TextColor="White"
VerticalOptions="Center"/>
            </HorizontalStackLayout>
            <Label Text="Ви можете заощадити, скасувавши невикористовувані підписки
на стрімінгові сервіси." TextColor="LightGray" FontSize="14"
LineBreakMode="WordWrap"/>
            <Label Text="Потенційна економія: 159,99 грн" TextColor="#90EE90"
FontSize="14" FontAttributes="Bold"/>
            <Button Text="Впровадити" BackgroundColor="#FFA500"
TextColor="#1E1E1E" CornerRadius="8" FontAttributes="Bold"
                    Clicked="ImplementAdvice1Button_Clicked"/>
        </VerticalStackLayout>
    </Frame>
    <Frame CornerRadius="10" Padding="15" BackgroundColor="#2C2C2C"
Margin="0,10,0,0">
        <VerticalStackLayout Spacing="10">
            <HorizontalStackLayout Spacing="10">
                <Image Source="lightbulb_icon.png" HeightRequest="24" WidthRequest="24"/>
                <Label Text="Оптимізуйте витрати на продукти" FontAttributes="Bold"
FontSize="16" TextColor="White" VerticalOptions="Center"/>
            </HorizontalStackLayout>
```

```
                <Label Text="Покупки в дисконтних магазинах можуть зменшити ваші
витрати на їжу." TextColor="LightGray" FontSize="14" LineBreakMode="WordWrap"/>
                <Label Text="Потенційна економія: 500,00 грн" TextColor="#90EE90"
FontSize="14" FontAttributes="Bold"/>
                <Button Text="Впровадити" BackgroundColor="#FFA500"
TextColor="#1E1E1E" CornerRadius="8" FontAttributes="Bold"
                        Clicked="ImplementAdvice2Button_Clicked"/>
            </VerticalStackLayout>
        </Frame>

    </VerticalStackLayout>
  </ScrollView>
</ContentPage>
```

HomePage.xaml.cs (Логіка до HomePage.xaml)

```
using System.Collections.ObjectModel; // Потрібно для ObservableCollection
using YourAppName.Models; // Потрібно для використання моделі Transaction

namespace YourAppName;

public partial class HomePage : ContentPage
{
  // Колекція транзакцій тепер у code-behind
  public ObservableCollection<Transaction> RecentTransactions { get; set; }

  public HomePage()
  {
    InitializeComponent();

    Title = "Головна"; // Встановлюємо заголовок сторінки

    // Ініціалізуємо колекцію та додаємо тестові дані
    RecentTransactions = new ObservableCollection<Transaction>
    {
      new Transaction { Id = "T123", Description = "Покупка продуктів в супермаркеті",
Amount = -755.50m, Date = DateTime.Now.AddDays(-1), Category = "Їжа", IsUnusual = false
},
      new Transaction { Id = "T124", Description = "Зарплата за місяць", Amount =
27000.00m, Date = DateTime.Now.AddDays(-2), Category = "Дохід", IsUnusual = false },
      new Transaction { Id = "T125", Description = "Щомісячна підписка на стрімінг",
Amount = -459.00m, Date = DateTime.Now.AddDays(-3), Category = "Підписки", IsUnusual =
true }, // Приклад "незвичайної"
       new Transaction { Id = "T126", Description = "Оплата комунальних послуг", Amount
= -3200.00m, Date = DateTime.Now.AddDays(-4), Category = "Житло", IsUnusual = false },
      new Transaction { Id = "T127", Description = "Квитки в кіно", Amount = -500.00m,
Date = DateTime.Now.AddDays(-5), Category = "Розваги", IsUnusual = false },
      new Transaction { Id = "T128", Description = "Проїзд в транспорті", Amount = -
50.00m, Date = DateTime.Now.AddDays(-6), Category = "Транспорт", IsUnusual = false },
    };

    // Встановлюємо контекст даних для прив'язки (ItemsSource CollectionView)
```

```
        BindingContext = this; // Прив'язуємо XAML до самого класу Page code-behind

        // Логіка для IsVisible попередження (встановлюємо в code-behind)
        UnusualTransactionsWarningFrame.IsVisible = RecentTransactions.Any(t => t.IsUnusual);
         // В реальності тут би була складніша логіка перевірки "незвичайності"
    }

    // Обробник події Clicked для кнопки "+"
    private async void AddTransactionButton_Clicked(object sender, EventArgs e)
    {
        System.Diagnostics.Debug.WriteLine("Add Transaction button clicked!");
        await DisplayAlert("Дія", "Натиснуто кнопку Додати транзакцію", "OK");
        // TODO: Навігація на сторінку додавання транзакції
    }

    // Обробник події Tapped для посилання "Переглянути все"
    private async void ViewAllTransactionsLabel_Tapped(object sender, EventArgs e)
    {
        System.Diagnostics.Debug.WriteLine("View All Transactions label tapped!");
        await DisplayAlert("Дія", "Натиснуто посилання Переглянути все", "OK");
        // TODO: Навігація на сторінку Історії всіх транзакцій
    }

    // Обробник події Clicked для першої кнопки "Впровадити" (порада 1)
    private async void ImplementAdvice1Button_Clicked(object sender, EventArgs e)
    {
        System.Diagnostics.Debug.WriteLine("Implement Advice 1 button clicked!");
        await DisplayAlert("Порада", "Натиснуто кнопку Впровадити (Зменшити підписки)",
"OK");
        // TODO: Логіка обробки поради 1
    }

    // Обробник події Clicked для другої кнопки "Впровадити" (порада 2)
    private async void ImplementAdvice2Button_Clicked(object sender, EventArgs e)
    {
        System.Diagnostics.Debug.WriteLine("Implement Advice 2 button clicked!");
        await DisplayAlert("Порада", "Натиснуто кнопку Впровадити (Оптимізувати витрати
на продукти)", "OK");
        // TODO: Логіка обробки поради 2
    }

    // Обробник події SelectionChanged для CollectionView
    private async void RecentTransactionsCollectionView_SelectionChanged(object sender,
SelectionChangedEventArgs e)
    {
        // Перевіряємо, чи був вибраний елемент, і чи це наша модель Transaction
        if (e.CurrentSelection.FirstOrDefault() is Transaction selectedTransaction)
        {
            System.Diagnostics.Debug.WriteLine($"Transaction selected:
{selectedTransaction.Description}");

            // Навігація на сторінку деталей, передаємо ID як параметр запиту URL
```

```
            var navigationParameter = new ShellNavigationQueryParameters
            {
                { "id", selectedTransaction.Id } // Передаємо ID як параметр "id"
            };
            await Shell.Current.GoToAsync($"TransactionDetailsPage", navigationParameter);

            // Скидаємо вибраний елемент, щоб можна було знову його вибрати
            ((CollectionView)sender).SelectedItem = null;
        }
    }
}
```

TransactionDetailsPage.xaml (Деталі транзакції)

```
        <ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
                     xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                     x:Class="YourAppName.TransactionDetailsPage"
                     Title="Деталі транзакції"
                     BackgroundColor="#1E1E1E"
                     x:DataType="local:TransactionDetailsPage" >
            <VerticalStackLayout Spacing="15" Padding="15">

                <Frame CornerRadius="15" Padding="20" BackgroundColor="#8B0000">
                    <VerticalStackLayout Spacing="5" HorizontalOptions="CenterAndExpand">
                        <Label Text="Витрата" FontSize="16" TextColor="#E0E0E0"
HorizontalTextAlignment="Center"/>
                        <Label Text="{Binding Transaction.Amount, StringFormat='{0:N2} грн'}"
FontSize="40" FontAttributes="Bold" TextColor="White"
HorizontalTextAlignment="Center"/> </VerticalStackLayout>
                </Frame>

                <Border Stroke="#FFA500" StrokeThickness="1" Padding="10"
BackgroundColor="#40FFA500" IsVisible="{Binding Transaction.IsUnusual}">
<Border.StrokeShape>
                        <RoundRectangle CornerRadius="10"/>
                    </Border.StrokeShape>
                    <HorizontalStackLayout Spacing="10">
                        <Image Source="warning_icon.png" HeightRequest="24" WidthRequest="24"
/>
                        <Label Text="Цю транзакцію позначено як незвичайну"
TextColor="#FFA500" VerticalOptions="Center"/>
                    </HorizontalStackLayout>
                </Border>

                <Frame CornerRadius="10" Padding="15" BackgroundColor="#2C2C2C">
                    <VerticalStackLayout Spacing="12">
                        <Grid ColumnDefinitions="Auto,*" RowSpacing="10">
                            <Label Grid.Row="0" Grid.Column="0" Text="Опис:" TextColor="Gray"
FontSize="14"/>
                            <Label Grid.Row="0" Grid.Column="1" Text="{Binding
Transaction.Description}" TextColor="White" FontSize="16" FontAttributes="Bold"/> <Label
```

```
Grid.Row="1" Grid.Column="0" Text="Категорія:" TextColor="Gray" FontSize="14"
VerticalOptions="Center"/>
                        <Border Grid.Row="1" Grid.Column="1" StrokeShape="RoundRectangle 8"
BackgroundColor="Blue" Padding="10,5" HorizontalOptions="Start">
                            <Label Text="{Binding Transaction.Category}" TextColor="White"
FontSize="14"/> </Border>

                        <Label Grid.Row="2" Grid.Column="0" Text="Дата:" TextColor="Gray"
FontSize="14"/>
                        <Label Grid.Row="2" Grid.Column="1" Text="{Binding Transaction.Date,
StringFormat='{0:dddd, d MMMM yyyy p.}'}" TextColor="White" FontSize="16"/> <Label
Grid.Row="3" Grid.Column="0" Text="ID транзакції:" TextColor="Gray" FontSize="14"/>
                        <Label Grid.Row="3" Grid.Column="1" Text="{Binding Transaction.Id}"
TextColor="White" FontSize="16"/> </Grid>
                    </VerticalStackLayout>
                </Frame>

                <Button Text="Видалити транзакцію"
                        TextColor="#FF6347" BackgroundColor="#2C2C2C"
                        BorderColor="#FF6347"
                        BorderWidth="1"
                        CornerRadius="10"
                        ImageSource="trash_icon.png"
                        Clicked="DeleteTransactionButton_Clicked" Margin="0,20,0,0"
                        HeightRequest="50"
                        FontSize="16"/>
            </VerticalStackLayout>
        </ContentPage>
```

TransactionDetailsPage.xaml.cs (Логіка та параметри навігації)

```
using YourAppName.Models; // Потрібно для використання моделі Transaction

namespace YourAppName;

// Реалізуємо IQueryAttributable для отримання параметрів навігації
public partial class TransactionDetailsPage : ContentPage, IQueryAttributable
{
    // Властивість для зберігання даних транзакції, до якої прив'язаний UI
    public Transaction Transaction { get; set; }

    public TransactionDetailsPage()
    {
        InitializeComponent();
        // BindingContext буде встановлений в ApplyQueryAttributes після
завантаження даних
    }

    // Цей метод викликається автоматично, коли сторінка отримує параметри запиту
через Shell
```

```
        public void ApplyQueryAttributes(IDictionary<string, object> query)
        {
            // Перевіряємо, чи є параметр "id" і чи він є рядком
            if (query.TryGetValue("id", out object idValue) && idValue is string transactionId)
            {
                System.Diagnostics.Debug.WriteLine($"Received Transaction ID in Details Page: {transactionId}");
                // TODO: Завантажити дані транзакції з джерела даних за отриманим transactionId
                // Для прикладу створюємо фіктивну транзакцію
                Transaction = new Transaction
                {
                    Id = transactionId,
                    Description = $"Деталі транзакції з ID: {transactionId}", // Приклад
                    Amount = -123.45m, // Приклад
                    Date = DateTime.Now.AddHours(-5), // Приклад
                    Category = "Тест Категорія", // Приклад
                    IsUnusual = true // Приклад
                };

                // Встановлюємо контекст даних ПІСЛЯ завантаження даних транзакції
                BindingContext = this; // Прив'язуємо XAML до цього класу Page code-behind
            }
        }

        // Обробник події Clicked для кнопки "Видалити транзакцію"
        private async void DeleteTransactionButton_Clicked(object sender, EventArgs e)
        {
            System.Diagnostics.Debug.WriteLine($"Delete Transaction button clicked for ID: {Transaction?.Id}");

            // Запит на підтвердження
            bool confirm = await DisplayAlert(
                "Підтвердження",
                "Ви впевнені, що хочете видалити цю транзакцію?",
                "Так", "Ні");

            if (confirm)
            {
                System.Diagnostics.Debug.WriteLine("Transaction deletion confirmed.");
                // TODO: Реалізуйте логіку видалення транзакції з джерела даних за Transaction.Id
                await DisplayAlert("Видалено", "Транзакцію видалено (імітація)", "OK");

                // Повертаємося назад
                await Shell.Current.GoToAsync("..");
            }
            else
            {
                System.Diagnostics.Debug.WriteLine("Transaction deletion cancelled.");
            }
        }
```

```
        // Важливо: скидати BindingContext або дані при виході зі сторінки, якщо вона не Transient
        // Хоча для Transient сторінок це менш критично, але гарна практика
        // protected override void OnNavigatedFrom(NavigatedFromEventArgs args)
        // {
        //     base.OnNavigatedFrom(args);
        //     Transaction = null; // Скидаємо дані
        //     BindingContext = null; // Скидаємо контекст прив'язки
        // }
    }
```

AnalyticsPage.xaml (Аналітика)

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="YourAppName.AnalyticsPage"
             Title="Аналітика"
             BackgroundColor="#1E1E1E">
    <ScrollView>
        <VerticalStackLayout Spacing="20" Padding="15">

            <Frame CornerRadius="10" Padding="15" BackgroundColor="#2C2C2C">
                <VerticalStackLayout Spacing="12">
                    <HorizontalStackLayout Spacing="10">
                        <Image Source="pie_chart_icon.png" HeightRequest="24" WidthRequest="24"/> <Label Text="Розподіл витрат" FontSize="20" FontAttributes="Bold" TextColor="White"/>
                    </HorizontalStackLayout>

                    <Grid ColumnDefinitions="Auto,*,Auto" RowDefinitions="Auto" ColumnSpacing="10" VerticalOptions="Center">
                        <Label Grid.Column="0" Text="Житло" TextColor="White" FontSize="16" VerticalTextAlignment="Center"/>
                        <ProgressBar Grid.Column="1" Progress="0.60" ProgressColor="Green" BackgroundColor="#444444" HeightRequest="12" VerticalOptions="Center"/> <Label Grid.Column="2" Text="8 500" TextColor="White" FontSize="16" VerticalTextAlignment="Center"/>
                    </Grid>
                    <Grid ColumnDefinitions="Auto,*,Auto" ColumnSpacing="10" VerticalOptions="Center">
                        <Label Grid.Column="0" Text="Розваги" TextColor="White" FontSize="16" VerticalTextAlignment="Center"/>
                        <ProgressBar Grid.Column="1" Progress="0.20" ProgressColor="Purple" BackgroundColor="#444444" HeightRequest="12" VerticalOptions="Center"/>
                        <Label Grid.Column="2" Text="1 200" TextColor="White" FontSize="16" VerticalTextAlignment="Center"/>
                    </Grid>
                     <Grid ColumnDefinitions="Auto,*,Auto" ColumnSpacing="10" VerticalOptions="Center">
                        <Label Grid.Column="0" Text="Транспорт" TextColor="White" FontSize="16" VerticalTextAlignment="Center"/>
```

```
                        <ProgressBar Grid.Column="1" Progress="0.10" ProgressColor="Blue"
BackgroundColor="#444444" HeightRequest="12" VerticalOptions="Center"/>
                        <Label Grid.Column="2" Text="755" TextColor="White" FontSize="16"
VerticalTextAlignment="Center"/>
                    </Grid>
                     <Grid ColumnDefinitions="Auto,*,Auto" ColumnSpacing="10"
VerticalOptions="Center">
                        <Label Grid.Column="0" Text="Їжа" TextColor="White" FontSize="16"
VerticalTextAlignment="Center"/>
                        <ProgressBar Grid.Column="1" Progress="0.05" ProgressColor="Orange"
BackgroundColor="#444444" HeightRequest="12" VerticalOptions="Center"/>
                        <Label Grid.Column="2" Text="459" TextColor="White" FontSize="16"
VerticalTextAlignment="Center"/>
                    </Grid>
                     <Grid ColumnDefinitions="Auto,*,Auto" ColumnSpacing="10"
VerticalOptions="Center">
                        <Label Grid.Column="0" Text="Підписки" TextColor="White"
FontSize="16" VerticalTextAlignment="Center"/>
                        <ProgressBar Grid.Column="1" Progress="0.03" ProgressColor="Red"
BackgroundColor="#444444" HeightRequest="12" VerticalOptions="Center"/>
                        <Label Grid.Column="2" Text="299" TextColor="White" FontSize="16"
VerticalTextAlignment="Center"/>
                    </Grid>
                    </VerticalStackLayout>
                </Frame>

                <Frame CornerRadius="10" Padding="15" BackgroundColor="#2C2C2C">
                   <VerticalStackLayout Spacing="10">
                      <HorizontalStackLayout Spacing="10">
                         <Image Source="line_chart_icon.png" HeightRequest="24"
WidthRequest="24"/> <Label Text="Місячний тренд" FontSize="20" FontAttributes="Bold"
TextColor="White"/>
                      </HorizontalStackLayout>
                      <Border HeightRequest="200" BackgroundColor="#333333"
Stroke="#555555" StrokeThickness="1">
                         <Border.StrokeShape>
                            <RoundRectangle CornerRadius="8"/>
                         </Border.StrokeShape>
                         <Label Text="Візуалізація місячного тренду витрат" TextColor="Gray"
HorizontalOptions="Center" VerticalOptions="Center"/>
                      </Border>
                   </VerticalStackLayout>
                </Frame>

             </VerticalStackLayout>
          </ScrollView>
       </ContentPage>
```

MauiProgram.cs

```
using Microsoft.Extensions.Logging;
```

```
namespace YourAppName;

public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .UseMauiCommunityToolkit()
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
            });

#if DEBUG
        builder.Logging.AddDebug();
#endif

        // !!! ВИДАЛЯЄМО РЕЄСТРАЦІЮ PAGES ТА VIEWMODELS !!!
        // builder.Services.AddSingleton<HomePage>();
        // builder.Services.AddSingleton<HomePageViewModel>();
        // builder.Services.AddTransient<TransactionDetailsPage>();
        // builder.Services.AddTransient<TransactionDetailsPageViewModel>();
        // builder.Services.AddSingleton<AnalyticsPage>();
        // builder.Services.AddSingleton<AnalyticsPageViewModel>();

        // TODO: Залишимо приклади сервісів, якщо потрібно або ні.

        return builder.Build();
    }
}
```