МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ АНАЛІЗУ ТА ПРОГНОЗУВАННЯ БІЗНЕС-ПРОЦЕСІВ»**

Виконав: студент 2 курсу, групи 8.1264
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)
освітньої програми Інформаційні системи та штучний інтелект
(назва освітньої програми)
Жмурко Олександр Валерійович
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н., Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, к.ф.-м.н., доцент Лісняк А.О
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 126 Інформаційні системи та технології
(шифр і назва)

Освітня програма Інформаційні системи та штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

" ____ " _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Жмурку Олександру Валерійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка інтелектуальної системи для автоматизації аналізу та прогнозування бізнес-процесів

керівник роботи Матвіїшина Надія Вікторівна, доцент, к.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 326-с

2. Строк подання студентом роботи 02.12.2025 р

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

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

1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та розробка системи для автоматизації аналізів та прогнозування бізнес процесів на основі банківської системи

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 07.03.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	02.04.2025	
2.	Збір вихідних даних.	10.04.2025	
3.	Обробка методичних та теоретичних	10.05.2025	
	джерел.		
4.	Розробка першого та другого розділу.	20.08.2025	
5.	Розробка третього розділу.	24.11.2025	
6.	Оформлення та нормоконтроль	10.11.2025	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	16.12.2025	

Студент _____ О.В. Жмурко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.В. Матвіїшна
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка інтелектуальної системи для автоматизації аналізу та прогнозування бізнес-процесів»: 80 с., 23 рис., 19 джерел.

АНАЛІТИКА ДАНИХ, БІЗНЕС-ПРОЦЕСИ, ЕКОНОМІЧНИЙ ЕФЕКТ, ІНТЕЛЕКТУАЛЬНА СИСТЕМА, МАШИННЕ НАВЧАННЯ, ПРОГНОЗУВАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, ERP-ІНТЕГРАЦІЯ, LSTM-МОДЕЛІ.

Об’єкт дослідження. Бізнес-процеси підприємств, які характеризуються великою кількістю взаємопов’язаних даних, що підлягають аналізу та прогнозуванню.

Мета роботи. Розробити архітектуру та програмну реалізацію інтелектуальної системи, яка забезпечує автоматизацію аналітичних і прогностичних процесів у бізнес-середовищі.

Методи дослідження. Методи машинного навчання, статистичного аналізу, нормалізації даних, обробки часових рядів, нейронні мережі (LSTM, GRU), методи оптимізації та економічного моделювання (ROI, TCO, NPV).

У роботі представлено інтелектуальну систему для автоматизації аналізу та прогнозування бізнес-процесів підприємства. Проведено огляд існуючих підходів до прогнозування та побудови аналітичних систем, узагальнено принципи застосування машинного навчання у сфері управління бізнесом. Розроблено концептуальну архітектуру системи, що включає модулі збору, очищення та підготовки даних, аналітичне ядро на базі нейронних мереж, підсистему візуалізації результатів і інтеграційний інтерфейс для взаємодії з ERP-системами.

У результаті експериментальної перевірки підтверджено ефективність моделей прогнозування попиту й фінансових показників це середня похибка MAPE знижена на 23% порівняно з базовими моделями. Розроблена система

може забезпечити скорочення часу формування звітів, підвищення точності управлінських рішень та економічну доцільність впровадження. Отримані результати можуть бути використані під час розробки корпоративних аналітичних рішень, цифрової трансформації підприємств і впровадження інтелектуальних систем підтримки прийняття рішень.

SUMMARY

Master's Qualifying Theses "Development of an Intelligent System for Automating the Analysis and Forecasting of Business Processes": 80 pages, 23 figures, 19 sources.

ARTIFICIAL INTELLIGENCE, BUSINESS PROCESSES, DATA ANALYTICS, ECONOMIC EFFECT, FORECASTING, INTELLIGENT SYSTEM, MACHINE LEARNING, TIME SERIES, ERP INTEGRATION, LSTM MODELS.

Object of study. Business processes of enterprises are characterized by large volumes of interrelated data that are subject to analysis and forecasting.

Aim of the study. To develop the architecture and software implementation of an intelligent system that automates analytical and forecasting processes in a business environment.

Method of research. Machine learning, statistical data analysis, data normalization, time series processing, neural networks (LSTM, GRU), optimization methods, and economic modeling techniques (ROI, TCO, NPV).

The thesis presents the design and implementation of an intelligent system for automating the analysis and forecasting of business processes. The study includes a review of existing forecasting approaches and analytical systems, as well as an overview of machine learning applications in business process management. A conceptual system architecture has been developed, consisting of data acquisition, cleaning, and preprocessing modules, an analytical core based on neural networks, a visualization subsystem, and an integration interface for ERP systems.

Experimental evaluation confirmed the effectiveness of forecasting models for demand and financial indicators, achieving a 23% reduction in MAPE compared to baseline models. The developed system improves reporting

efficiency, enhances the accuracy of management decisions, and demonstrates economic feasibility through ROI and TCO indicators. The results can be applied in corporate analytics solutions, digital transformation initiatives, and the implementation of intelligent decision support systems.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 6
Вступ 9
1 Теоретичні основи побудови інтелектуальних систем 11
1.1 Поняття та класифікація інтелектуальних систем 14
1.2 Методи прогнозування та аналізу даних 17
1.3 Визначення стеку технологій для побудови системи 20
1.4 Огляд фреймворка Go та його можливостей 23
1.5 Огляд фреймворка React та його можливостей 24
1.6 Порівняння з іншими аналогами 26
1.7 Висновки до розділу 1 27
2 Проєктування інтелектуальної системи 29
2.1 Аналіз функціональних та нефункціональних вимог до системи 29
2.2 Структура бази даних вебзастосунку 31
2.3 Проєктування архітектури вебзастосунку 36
2.4 Проєктування серверної частини 45
2.5 Висновки до розділу 2 51
3 Програмна реалізація та тестування системи 52
3.1 Розробка архітектури та налаштування серверного середовища 52
3.2 Реалізація основних модулів системи 55
3.3 Розробка клієнтської частини та інтерфейсів 61
3.4 Тестування та оптимізація системи 68
3.5 Висновки до розділу 3 75
Висновки 77
Перелік посилань 79

ВСТУП

Сучасний розвиток технологій штучного інтелекту та обробки даних відкриває нові горизонти для підвищення ефективності управління підприємствами. Однією з ключових задач є перетворення великих обсягів операційних даних на стратегічні інсайти для прийняття обґрунтованих рішень. Прикладом таких рішень є платформи бізнес-аналітики, такі як Tableau, Power BI або SAP Analytics Cloud, які дозволяють візуалізувати показники, виявляти тенденції та будувати прогнозні моделі.

Метою кваліфікаційної роботи є проектування та реалізація інтелектуальної системи для автоматизації аналізу та прогнозування бізнес-процесів з використанням Go та фреймворку Gin для реалізації серверної частини, React та TypeScript для розробки клієнтського інтерфейсу, Python та FastAPI для створення ML-мікросервісів, а також PostgreSQL та Redis як систем зберігання даних.

Для досягнення поставленої мети сформульовано такі задачі:

- проаналізувати сучасні підходи та інструменти бізнес-аналітики та прогнозування;
- сформулювати функціональні та нефункціональні вимоги до інтелектуальної системи;
- спроєктувати мікросервісну архітектуру системи з чітким розмежуванням функціоналу;
- реалізувати основний API-шлюз та бізнес-логіку на Go з використанням фреймворку Gin;
- розробити інтерфейс користувача з використанням React та TypeScript;
- створити ML-мікросервіси на Python з FastAPI для аналітики та прогнозування;

- реалізувати систему кешування та обробки реального часу з використанням Redis;
- організувати зберігання та управління даними в PostgreSQL;
- виконати тестування окремих компонентів та провести інтеграційне тестування системи.

Структурно робота складається з трьох розділів. У першому розділі проаналізовано сучасні системи бізнес-аналітики, їхні функціональні можливості та архітектурні рішення, а також обґрунтовано вибір технологічного стеку: Go та Gin для високопродуктивного бекенду, React та TypeScript для безпечного та масштабованого фронтенду, Python та FastAPI для гнучких ML-рішень, а також PostgreSQL та Redis для надійного зберігання та швидкої обробки даних.

У другому розділі виконано аналіз функціональних і нефункціональних вимог до системи, спроєктовано мікросервісну архітектуру з використанням діаграм UML, розроблено схеми взаємодії між компонентами системи, а також спроєктовано структуру бази даних PostgreSQL та моделі кешування в Redis.

У третьому розділі реалізовано основні компоненти системи: серверну частину на Go з використанням Gin, клієнтський інтерфейс на React з TypeScript, ML-мікросервіси на Python з FastAPI, а також інтегровано систему зберігання даних. Проведено комплексне тестування окремих модулів та системи в цілому, включаючи тестування продуктивності та точності прогностичних моделей.

1 ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

У сучасних умовах цифровізації підприємства генерують значні обсяги даних про продажі, запаси, виробництво, фінанси, клієнтські взаємодії та операційну діяльність. Ці дані формують цифровий слід бізнесу, що містить цінну інформацію про всі аспекти його функціонування. Однак наявність даних сама по собі не гарантує ефективної підтримки управлінських рішень. Часто підприємства стикаються з ситуацією "даних багато, але інформації мало" – неструктурованість, розрізненість джерел та відсутність ефективних інструментів аналізу перешкоджають отриманню стратегічних переваг [1].

Виникає потреба в системах, які здатні не лише накопичувати інформацію, але й здійснювати аналіз, виявляти закономірності та виконувати прогнозування ключових бізнес-показників. Такі системи повинні інтегрувати різнорідні джерела даних, забезпечувати їхню очистку та попередню обробку, а також надавати можливості для застосування сучасних аналітичних методів [1].

Інтелектуальні системи у цій сфері виступають надбудовою над існуючими інформаційними системами (ERP, CRM, WMS тощо) та забезпечують перехід від традиційної звітності до проактивної аналітики, що дозволяє:

- **своєчасно виявляти тренди й відхилення** – автоматизований моніторинг ключових показників ефективності (KPI) дозволяє ідентифікувати аномалії в реальному часі, виявляти сезонні закономірності та довгострокові тенденції, що дає змогу оперативно реагувати на зміни у бізнес-середовищі;
- **будувати прогнози попиту, доходів, навантажень, запасів** – використання статистичних методів та алгоритмів машинного навчання дозволяє створювати точні прогнозні моделі для різних

бізнес-сценаріїв, оптимізуючи таким чином управління ресурсами та ланцюгами поставок;

- **оцінювати наслідки управлінських рішень** – за допомогою технологій "what-if" аналізу та симуляційних моделей система може прогнозувати вплив різних управлінських рішень на бізнес-показники, надаючи керівництву обґрунтовану основу для вибору оптимальної стратегії;
- **зменшувати ризики, пов'язані з невизначеністю ринку** – інтелектуальні системи дозволяють кількісно оцінювати ризики та невизначеність, розраховуючи ймовірнісні сценарії розвитку подій, що особливо важливо в умовах волатильного ринкового середовища.

У цьому розділі розглянуто базові поняття та класифікацію інтелектуальних систем, а також основні методи прогнозування та аналізу даних, що доцільно використовувати для побудови інтелектуальної системи автоматизації аналізу й прогнозування бізнес-процесів [1].

Крім того, у розділі представлено короткий огляд технологічних рішень, обраних для реалізації системи:

Go та фреймворк Gin – мова програмування з високою продуктивністю та ефективним використанням ресурсів, ідеально підходить для побудови масштабованих бекенд-систем. Фреймворк Gin надає мінімалістичний та швидкий інструмент для створення API, обробки запитів та маршрутизації, що забезпечує високу пропускну здатність системи [2].

React з TypeScript – сучасний підхід до розробки інтерфейсів користувача, де React забезпечує компонентний підхід та ефективне оновлення UI, а TypeScript додає статичну типізацію, підвищуючи надійність коду та полегшуючи розробку складних інтерфейсів візуалізації аналітичних даних.

Класифікація інтелектуальних систем бізнес-аналітики включає:

За рівнем складності аналітики:

- описальна аналітика (descriptive) – аналіз історичних даних;
- діагностична аналітика (diagnostic) – виявлення причин явищ;

– предиктивна аналітика (predictive) – прогнозування майбутніх подій;
– prescriptive аналітика – рекомендації щодо дій.

За типом використовуваних методів:

– статистичні системи;
– системи машинного навчання;
– гібридні системи.

Основні методи прогнозування та аналізу, що розглядаються:

Статистичні методи:

– часові ряди (ARIMA, SARIMA, ETS);
– регресійний аналіз;
– кластерний аналіз;
– факторний аналіз.

Методи машинного навчання:

– дерева рішень та випадкові ліси;
– градієнтний бустинг (XGBoost, LightGBM);
– рекурентні нейронні мережі (LSTM, GRU);
– методи ансамблювання моделей.

Спеціалізовані бізнес-методики:

– прогнозування попиту;
– ABC-XYZ аналіз;
– аналіз життєвого циклу продукту;
– Customer Lifetime Value прогнозування.

Розгляд цих аспектів становить теоретичну основу для подальшого проектування та реалізації інтелектуальної системи, що забезпечить комплексний підхід до аналізу та прогнозування бізнес-процесів з використанням сучасних технологічних рішень [2].

1.1 Поняття та класифікація інтелектуальних систем

Поняття інтелектуальної системи в контексті бізнес-аналітики визначається як програмно-апаратний комплекс, здатний до навчання, адаптації та автономного прийняття рішень на основі аналізу великих обсягів структурованих і неструктурованих даних. Основна відмінність інтелектуальних систем від традиційних інформаційних систем полягає в здатності не тільки обробляти дані, але й генерувати знання, виявляти складні залежності та формувати прогностичні моделі [2].

Існують моделі за рівнем автономності, які визначають ступінь самостійності системи в процесі прийняття рішень та виконання аналітичних операцій:

- інформаційно-довідкові системи – базовий рівень автономності, де система функціонує як розширений довідковий механізм. Вона забезпечує користувачів структурованою інформацією, гістограмами, трендами та основними метриками, але не формує готових висновків чи рекомендацій. Наприклад, система може відображати динаміку продажів за останній квартал, але аналіз причин змін залишається за фахівцем;
- експертні системи – середній рівень автономності, що включають базу знань і механізми логічного виводу. Ці системи здатні відтворювати логіку прийняття рішень досвідчених фахівців галузі. Вони аналізують вхідні дані, порівнюють їх із закладеними правилами та шаблонами, і формують обґрунтовані рекомендації. Наприклад, експертна система може пропонувати оптимальний рівень товарних запасів на основі аналізу історичних даних, сезонності та зовнішніх факторів;
- автономні інтелектуальні агенти – найвищий рівень автономності, де система здатна до самостійних дій та адаптації в реальному часі. Такі

агенти використовують методи машинного навчання для постійного вдосконалення своїх моделей, можуть ініціювати дії без прямого втручання людини та ефективно функціонують в умовах неповноти даних. Наприклад, автономна система управління запасами може автоматично корегувати рівні запасів на основі прогнозів попиту, аналізу ринкових тенденцій та логістичних обмежень.

Також системи бізнес-аналітики часто поєднують елементи різних рівнів автономності, забезпечуючи гнучкий підхід до автоматизації аналітичних процесів. За характером обробки даних інтелектуальні системи поділяються на кілька типів, кожен з яких має свої особливості застосування в бізнес-аналітиці [2].

Детерміновані системи працюють за чіткими алгоритмами та правилами, де результат обробки даних повністю передбачуваний і залежить від вхідних параметрів. Ці системи використовують формалізовані математичні моделі та логічні правила для аналізу даних.

В контексті бізнес-аналітики такі системи особливо ефективні для задач з чітко визначеними вхідними даними та алгоритмами обробки, наприклад, для розрахунку стандартних фінансових показників, формування регламентованої звітності або автоматизації рутинних аналітичних процесів.

Їхня перевага полягає у високій надійності та відтворюваності результатів, що є критично важливим для фінансової та операційної звітності [3].

Ймовірнісні системи використовують статистичні методи та теорію ймовірностей для аналізу даних, що дозволяє їм ефективно працювати з неповними або зашумленими даними. Такі системи особливо корисні для прогнозування ринкових тенденцій, аналізу ризиків та виявлення складних залежностей у бізнес-даних. Вони можуть оцінювати ймовірність різних сценаріїв розвитку подій, що є цінним інструментом для стратегічного планування та управління ризиками. Наприклад, ймовірнісні моделі можуть

прогнозувати зростання попиту на продукцію з певним рівнем довіри, враховуючи різні фактори впливу [3].

Нейромережеві системи представляють найбільш просунутий підхід до обробки даних, заснований на принципах роботи штучних нейронних мереж. Ці системи здатні навчатися на великих обсягах історичних даних, виявляти складні нелінійні залежності та адаптуватися до змінних умов. В бізнес-аналітиці нейромережеві підходи особливо ефективні для таких задач, як прогнозування продажів з урахуванням множини факторів, класифікація клієнтів за поведінковими паттернами, аналіз sentiment тональності відгуків та виявлення аномалій у фінансових операціях. Їхня здатність до самонавчання та узагальнення знань робить їх незамінними для роботи з великими даними та складними аналітичними завданнями [4].

Ще цікавим можна виділити те, що системи бізнес-аналітики демонструють значну диференціацію за функціональною спеціалізацією, що обумовлено специфікою різних напрямів діяльності підприємств та різноманіттям бізнес-процесів, які потребують автоматизації аналітичних функцій [4].

Системи прогнозування попиту становлять один з найбільш поширених класів інтелектуальних систем, що застосовуються в галузі бізнес-аналітики. Ці системи характеризуються здатністю до аналізу широкого спектру факторів, що впливають на потреби ринку, включаючи історичні дані про продажі, сезонні коливання, економічні індикатори, маркетингові активності та зовнішні чинники. Специфікою таких систем є використання передових алгоритмів машинного навчання, таких як рекурентні нейронні мережі та методи ансамблювання, що дозволяють досягати високої точності прогнозів на різних часових горизонтах. Ефективне функціонування систем прогнозування попиту забезпечує оптимізацію управління запасами, виробничим плануванням та логістичними процесами.

Системи управління ризиками реалізують складний комплекс аналітичних функцій, спрямованих на ідентифікацію, оцінку та мінімізацію

різноманітних ризиків діяльності підприємства. Особливістю даного класу систем є інтеграція різнорідних джерел даних, включаючи фінансову звітність, операційні показники, ринкову інформацію та зовнішні економічні дані. Сучасні системи управління ризиками використовують методи предиктивної аналітики та сценарного моделювання для прогнозування ймовірності настання ризикових подій та оцінки їх потенційного впливу на бізнес. Технології штучного інтелекту в таких системах дозволяють виявляти складні закономірності та кореляції, що не є очевидними при традиційному аналізі [4].

Системи аналізу клієнтської поведінки спеціалізуються на дослідженні паттернів поведінки споживачів та виявленні факторів, що впливають на їхні рішення. Ці системи аналізують дані з систем управління взаєминами з клієнтами (CRM), транзакційні дані, інформацію з соціальних мереж та інших каналів взаємодії. Вони використовують методи кластеризації для сегментації клієнтської бази, алгоритми асоціативних правил для аналізу кошиків покупок, а також методи прогнозування відтоку клієнтів. Особливістю сучасних систем аналізу клієнтської поведінки є використання технологій обробки природної мови для аналізу текстових відгуків та методів глибокого навчання для прогнозування індивідуальних уподобань клієнтів.

Кожен з цих спеціалізованих класів систем вимагає використання специфічних методів аналізу даних, алгоритмів машинного навчання та архітектурних рішень, що обумовлює необхідність ретельного вибору технологічного стеку та методології розробки при створенні інтелектуальної системи для конкретної бізнес-сфери [4].

1.2 Методи прогнозування та аналізу даних

Сучасна бізнес-аналітика спирається на цілий спектр методів прогнозування, які можна умовно поділити на дві великі категорії: традиційні статистичні підходи та методи машинного навчання. Кожен з цих підходів має

свої переваги та обмеження, а їхнє практичне застосування значною мірою залежить від специфіки бізнес-завдань та характеру наявних даних [5].

Статистичні методи залишаються фундаментом аналітики, особливо коли мова йде про прогнозування на основі історичних даних. Аналіз часових рядів, зокрема використання моделей ARIMA та SARIMA, дозволяє будувати точні прогнози для показників з чітко вираженою сезонністю та тенденціями. Наприклад, у роздрібній торгівлі ці методи успішно застосовуються для прогнозування попиту на товари з урахуванням сезонних коливань. Регресійний аналіз, з іншого боку, дає змогу виявляти складні взаємозв'язки між різними бізнес-показниками, що особливо цінно при аналізі впливу маркетингових активностей на обсяги продажів.

Однак з ростом обсягів даних та ускладненням бізнес-процесів все більшого значення набувають методи машинного навчання. Дерева рішень та ансамблеві алгоритми, такі як випадкові ліси та градієнтний бустинг, демонструють високу ефективність у задачах класифікації та прогнозування. Вони здатні автоматично виявляти складні нелінійні залежності в даних, що робить їх незамінними при аналізі клієнтської поведінки чи оцінці кредитних ризиків [5].

Для роботи з послідовними даними, як-от часові ряди фінансових показників або користувацькі сесії на веб-сайтах, найбільш підходящими виявляються рекурентні нейронні мережі, особливо архітектури LSTM та GRU. Вони здатні "запам'ятовувати" довгострокові залежності, що є ключовим для точного прогнозування таких складних явищ, як динаміка ринкового попиту чи відтоку клієнтів.

Особливе місце в бізнес-аналітиці займають спеціалізовані методи, розроблені для вирішення конкретних бізнес-завдань. ABC-XYZ аналіз дозволяє оптимізувати управління запасами, класифікуючи товари за обсягами продажів та стабільністю попиту. Прогнозування Customer Lifetime Value допомагає визначати найбільш цінних клієнтів і будувати ефективні стратегії їх

утримання. Кластерний аналіз знаходить застосування в сегментації клієнтської бази, що дозволяє розробляти персоналізовані маркетингові кампанії [5].

Важливим аспектом сучасної аналітики є інтеграція методів обробки природної мови (NLP) для аналізу текстових даних. Ці методи дозволяють автоматизувати обробку клієнтських відгуків, аналізувати тональність висловлювань у соціальних мережах, витягувати ключові теми з документів. Наприклад, аналіз текстів відгуків про продукти може виявити приховані проблеми якості або визначити фактори, що впливають на задоволеність клієнтів.

Для роботи з великими обсягами даних все частіше застосовуються розподілені обчислювальні системи, такі як Apache Spark, які дозволяють обробляти інформацію в реальному часі та будувати прогнози на основі актуальних даних. Це особливо важливо для таких завдань, як виявлення аномалій у фінансових операціях або моніторинг ринкових тенденцій [6].

Сучасним трендом є використання автоматизованого машинного навчання (AutoML), яке дозволяє значно прискорити процес побудови та налаштування прогностичних моделей. AutoML-системи можуть автоматично підбирати оптимальні алгоритми, виконувати feature engineering та налаштовувати гіперпараметри моделей, що робить аналітику доступною для фахівців без глибоких знань у машинному навчанні.

Вибір конкретних методів для реалізації в інтелектуальній системі залежить від багатьох факторів, включаючи характер доступних даних, вимоги до точності прогнозів, обчислювальні ресурси та досвід аналітиків. На практиці найкращих результатів часто вдається досягти за рахунок комбінування різних підходів, коли статистичні методи використовуються для базового аналізу, а алгоритми машинного навчання – для вирішення більш складних завдань. Крім того, важливим є питання інтерпретованості моделей – для прийняття управлінських рішень необхідно не тільки отримувати точні прогнози, але й розуміти, які фактори впливають на результати прогнозування [6].

1.3 Визначення стеку технологій для побудови системи

Розроблена інтелектуальна система знаходить широке застосування в банківській сфері, де вимоги до обробки даних, безпеки та надійності є особливо високими. У контексті фінансових установ система може бути ефективно використана для вирішення ряду ключових завдань.

Прогнозування кредитних ризиків становить одну з найважливіших сфер застосування. Система здатна аналізувати історичні дані про позичальників, включаючи інформацію про платоспроможність, кредитну історію, соціально-демографічні характеристики та економічні показники. Використовуючи алгоритми машинного навчання, система може оцінювати ймовірність дефолту за кожним кредитом, що дозволяє банку оптимізувати кредитний портфель та зменшити втрати [7].

Виявлення шахрайських операцій є іншим критично важливим напрямком. За допомогою аномалії детекшн та алгоритмів класифікації система може ідентифікувати підозрілі транзакції в реальному часі. Аналізуючи шаблони поведінки клієнтів, незвичайні операції та інші фактори ризику, система формує попередження для служби безпеки банку.

Оптимізація клієнтського сервісу досягається через аналіз поведінки клієнтів та прогнозування їх потреб. Система може рекомендувати індивідуальні фінансові продукти, оптимізувати роботу відділень та кол-центру, а також покращувати якість обслуговування через персоналізацію пропозицій [7].

Управління ліквідністю вимагає точної прогностичної аналітики. Система допомагає банку прогнозувати грошові потоки, оптимізувати залишки коштів та ефективно керувати активами і пасивами, що є ключовим для забезпечення фінансової стабільності установи.

Вимоги безпеки та відмовостійкості в банківському сегменті визначають необхідність реалізації додаткових заходів захисту. Система забезпечує

шифрування даних, багаторівневу автентифікацію, аудит операцій та резервне копіювання. Архітектура мікросервісів дозволяє ізолювати критично важливі компоненти та забезпечити їхню стабільну роботу навіть у разі збоїв в окремих модулях.

Інтеграція з існуючими банківськими системами є обов'язковою вимогою. Розроблена система може бути інтегрована з ядром банку, CRM, системами інтернет-банкінгу та іншими платформами через стандартизовані API. Це забезпечує єдиний інформаційний простір та дозволяє використовувати всі доступні дані для аналітики та прогнозування [8].

Відповідність регуляторним вимогам є критично важливим аспектом. Система допомагає банку виконувати вимоги Національного банку України щодо управління ризиками, протидії відмиванню коштів та формування звітності. Механізми логування та аудиту забезпечують прозорість роботи системи та дозволяють відстежувати всі операції.

Вибір технологічного стеку для розробки інтелектуальної системи аналізу та прогнозування ґрунтувався на комплексному аналізі вимог до продуктивності, масштабованості та ефективності обробки даних. Архітектура системи формувалася з урахуванням необхідності забезпечення високої пропускної здатності при роботі з великими обсягами даних, можливості паралельної обробки запитів та інтеграції різних підсистем.

Для серверної частини системи обрано мову програмування Go та фреймворк Gin, що обумовлено низкою критично важливих факторів. Go демонструє виняткову продуктивність при роботі з високими навантаженнями, що є ключовим вимогою для системи аналітики, яка має обробляти тисячі запитів одночасно. Механізм горутин дозволяє ефективно використовувати ресурси процесора при паралельній обробці даних, а статична типізація забезпечує стабільність та безпеку коду. Фреймворк Gin пропонує мінімалістичний, але потужний інструментарій для створення API, що дозволяє реалізувати швидку обробку HTTP-запитів та ефективну маршрутизацію. Це

особливо важливо для забезпечення відгуку системи в умовах реального часу при обробці аналітичних запитів [9].

Клієнтська частина реалізована з використанням React у поєднанні з TypeScript. Даний вибір обумовлений необхідністю створення динамічного та інтуїтивно зрозумілого інтерфейсу для візуалізації складних аналітичних даних. React дозволяє ефективно керувати станом інтерфейсу та забезпечує швидке оновлення компонентів при зміні даних, що є критично важливим для відображення реальних показників та прогнозів. TypeScript додає статичну типізацію, що значно підвищує надійність коду та спрощує розробку складних інтерфейсів, зменшуючи кількість помилок на етапі розробки [9].

Для машинного навчання та аналітичних модулів використано Python з фреймворком FastAPI. Python є стандартом де-факто в галузі машинного навчання завдяки широкому спектру бібліотек, таких как scikit-learn, pandas, NumPy та TensorFlow. FastAPI як сучасний фреймворк дозволяє швидко створювати продуктивні API для ML-моделей, забезпечуючи автоматичну генерацію документації та вбудовану валідацію даних. Архітектура мікросервісів на основі FastAPI дозволяє ізолювати ML-компоненти, що забезпечує гнучкість у їхньому розвитку та масштабуванні [9].

Система керування базами даних реалізована на основі PostgreSQL, що обумовлено потребою у надійному зберіганні структурованих даних та підтримці складних SQL-запитів. PostgreSQL забезпечує високу продуктивність при роботі з великими обсягами даних, підтримує транзакційність та цілісність даних, що є критично важливим для фінансової та операційної аналітики. Для кешування та зберігання тимчасових даних використано Redis, який забезпечує високу швидкість доступу до даних, що часто використовуються, та дозволяє реалізувати механізми сесій та тимчасового зберігання результатів проміжних обчислень.

Інтеграція всіх компонентів забезпечує високу продуктивність, масштабованість та надійність системи. Go обробляє бізнес-логіку та API-запити, Python відповідає за складні аналітичні алгоритми, React забезпечує

зручний інтерфейс для користувачів, а PostgreSQL та Redis гарантують цілісність та швидкість доступу до даних. Така архітектура дозволяє ефективно розподіляти навантаження між компонентами та забезпечує гнучкість у розвитку системи [9].

1.4 Огляд фреймворка Go та його можливостей

Мова програмування Go була обрана для реалізації серверної частини системи як оптимальне рішення, що поєднує високу продуктивність, простоту розробки та ефективне управління ресурсами. Використання Go дозволяє забезпечити стабільну роботу системи в умовах високих навантажень, що є критично важливим для системи бізнес-аналітики, яка має обробляти великі обсяги даних у реальному часі [10].

Основним фреймворком для розробки серверної частини обрано Gin, який пропонує мінімалістичний, але потужний інструментарій для створення високопродуктивних API. Gin забезпечує швидку обробку HTTP-запитів, ефективну маршрутизацію та підтримку middleware, що дозволяє реалізувати всі необхідні функції системи без надмірного ускладнення кодової бази.

Ключовою перевагою Go є вбудована підтримка конкурентного програмування за допомогою горутин та каналів. Ця можливість дозволяє системі ефективно обробляти паралельні запити до бази даних, одночасно виконувати кілька аналітичних операцій та обслуговувати велику кількість користувачів без істотного збільшення споживання ресурсів. Механізм горутин є значно легшим за традиційні потоки операційної системи, що дозволяє створювати десятки тисяч одночасних виконавчих одиниць.

Статична типізація та строга компіляція в Go забезпечують високу надійність коду, що особливо важливо для фінансової та бізнес-аналітики, де помилки можуть призвести до серйозних наслідків. Компілятор Go виявляє більшість потенційних помилок на етапі компіляції, а строга типізація запобігає

багатьом типовим проблемам, що виникають у динамічних мовах програмування [10].

Система пакунків та модулів в Go спрощує управління залежностями та сприяє створенню чистої архітектури. Проєкт може бути організований у вигляді набору незалежних модулів, що покращує підтримку та розвиток системи. Велика стандартна бібліотека Go надає готові рішення для більшості типових завдань, таких як робота з мережею, криптографія, обробка JSON та XML.

Для взаємодії з базами даних Go пропонує ефективний драйвер для PostgreSQL, який оптимально підходить для роботи з великими обсягами структурованих даних. Підтримка пулінгу з'єднань, транзакцій та підготовлених запитів дозволяє забезпечити високу продуктивність при роботі з базою даних.

Інтеграція з мікросервісами Python реалізована через REST API, що забезпечує гнучкість архітектури та дозволяє оптимально розподілити навантаження між різними компонентами системи. Go ефективно обробляє запити до ML-модулів, кешує результати та забезпечує їх швидку доставку клієнтській частині.

Вбудовані інструменти тестування та профілювання дозволяють підтримувати високу якість коду та оптимізувати продуктивність критичних ділянок системи. Профайлери CPU та пам'яті допомагають виявляти вузькі місця, а вбудована підтримка бенчмаркінгів дозволяє точно оцінювати продуктивність окремих компонентів [10].

1.5 Огляд фреймворка React та його можливостей

Для розробки клієнтської частини системи бізнес-аналітики було обрано бібліотеку React у поєднанні з TypeScript, що сформувало сучасний та ефективний технологічний підхід для побудови складних аналітичних

інтерфейсів. Основною перевагою React є компонентно-орієнтована архітектура, яка дозволяє модульно структурувати інтерфейс на незалежні, багаторазово використовувані блоки. Це забезпечує чітку організацію коду, прискорює процес розробки та спрощує подальшу підтримку та масштабування системи [10].

Важливу роль відіграє використання віртуального DOM, що істотно підвищує продуктивність застосунку шляхом мінімізації прямих маніпуляцій з реальним DOM – оновлюються виключно ті елементи інтерфейсу, які зазнали реальних змін. Цей підхід, поєднаний з декларативним стилем програмування, дозволяє розробникам зосередитись на бізнес-логіці відображення аналітичних даних, замість низькорівневої роботи з елементами сторінки.

У процесі реалізації використовуються як функціональні, так і класові компоненти. Функціональні компоненти застосовуються переважно для візуалізації статичних елементів та простих інтерфейсних блоків – таких як панелі інструментів, легенди графіків, інформаційні картки. Класові компоненти використовуються для реалізації складної бізнес-логіки – робота з інтерактивними діаграмами, динамічне оновлення даних у реальному часі, обробка фільтрів та параметрів аналітики, валідація вхідних даних та інтеграція з серверною частиною через REST API.

TypeScript забезпечує статичну типізацію, що значно підвищує надійність коду та спрощує розробку складних інтерфейсів візуалізації даних. Система типів допомагає уникнути помилок на етапі розробки та покращує підтримку коду в довгостроковій перспективі [11].

Такий архітектурний підхід дозволяє не лише забезпечити високу якість та модульність коду, але й значно полегшує тестування окремих компонентів системи візуалізації та дає змогу швидко впроваджувати новий аналітичний функціонал без ризику порушити існуючу логіку роботи.

React у поєднанні з TypeScript активно використовується у багатьох провідних системах бізнес-аналітики, зокрема у таких продуктах як Microsoft Power BI, Salesforce Analytics та IBM Cognos Analytics, що підтверджує його

ефективність для реалізації складних аналітичних платформ з високими вимогами до продуктивності, масштабованості та якості користувацького досвіду. Саме тому його використання в рамках даного проєкту є цілком обґрунтованим технічним рішенням.

1.6 Порівняння з іншими аналогами

У процесі розробки інтелектуальної системи аналізу та прогнозування бізнес-процесів важливо оцінити її конкурентоспроможність порівняно з сучасними аналогічними рішеннями на ринку, такими як SAP Analytics Cloud, Tableau та Microsoft Power BI [11].

SAP Analytics Cloud, як частина екосистеми SAP, пропонує комплексне рішення для бізнес-аналітики з інтеграцією з іншими продуктами компанії. Його перевагами є глибока інтеграція з SAP-системами, розширені можливості планування та прогнозування. Проте рішення має значні обмеження: висока вартість ліцензування, складність налаштування та кастомізації, залежність від екосистеми SAP, що робить його менш привабливим для середнього та малого бізнесу.

Tableau відзначається потужними можливостями візуалізації даних та інтуїтивним інтерфейсом для створення звітів. Сильними сторонами платформи є підтримка різноманітних джерел даних, розширені функції drag-and-drop, мобільні додатки. Однак Tableau має обмежену функціональність для створення власних прогностичних моделей, високу вартість та закриту архітектуру, що ускладнює глибоку інтеграцію з існуючими бізнес-системами [11].

Microsoft Power BI вигідно відрізняється інтеграцією з продуктами Microsoft, зрозумілою ціновою політикою та значною спільнотою користувачів. Платформа пропонує потужні інструменти для створення звітів та дашбордів. До недоліків можна віднести обмежену підтримку передових методів

машинного навчання, закритість архітектури для кастомних розробок та залежність від екосистеми Microsoft.

Порівняно з цими платформами, наша інтелектуальна система, реалізована на основі Go, React та Python, пропонує унікальні переваги. Відкрита архітектура дозволяє глибоку інтеграцію з будь-якими бізнес-системами, а модульна структура забезпечує гнучкість у розширенні функціоналу. Використання сучасних технологій машинного навчання на базі Python забезпечує більш точні та адаптивні прогнози порівняно зі стандартними статистичними методами, що використовуються в комерційних рішеннях.

Важливою перевагою нашої системи є можливість повного контролю над даними та алгоритмами, що критично важливо для банківської сфери та компаній з високими вимогами до безпеки. На відміну від комерційних рішень, наша система не залежить від зовнішніх хмарних сервісів і може розгортатися в приватній інфраструктурі клієнта.

З точки зоронти продуктивності, використання Go забезпечує високу швидкість обробки даних та масштабованість, що дозволяє системі ефективно працювати з великими обсягами даних у реальному часі. React інтерфейс забезпечує інтуїтивну взаємодію з складними аналітичними даними, а модульна архітектура дозволяє легко адаптувати систему під конкретні бізнес-потреби.

Таким чином, наша система пропонує унікальне поєднання відкритої архітектури, передових методів аналітики, високої продуктивності та гнучкості, що робить її конкурентноздатною [11].

1.7 Висновки до розділу 1

У першому розділі проведно аналіз сучасних підходів до побудови інтелектуальних систем бізнес-аналітики та технологій, що використовуються для їх реалізації. Обґрунтовано вибір технологічного стеку, що включає React з TypeScript для клієнтської частини, Go з фреймворком Gin для серверної

логіки, Python з FastAPI для ML-мікросервісів, а також PostgreSQL та Redis для роботи з даними.

Вибір React як основної технології для клієнтської частини обумовлений його компонентно-орієнтованою архітектурою, зручним механізмом стану, підтримкою TypeScript для забезпечення типобезпеки, а також можливістю створення складних інтерфейсів візуалізації аналітичних даних. Використання Go на серверній стороні дозволяє забезпечити високу продуктивність обробки запитів, ефективне використання ресурсів за рахунок горутин та конкурентності, а також створення масштабованих API. Застосування Python з FastAPI для ML-компонентів забезпечує гнучкість у розробці та інтеграції алгоритмів машинного навчання, що є ключовим для реалізації аналітичних функцій системи.

Таким чином, в результаті проведеного аналізу було сформовано технічну основу для розробки інтелектуальної системи аналізу та прогнозування бізнес-процесів, яка враховує сучасні вимоги до продуктивності, масштабованості, безпеки та ефективної обробки даних. Обраний стек технологій дозволяє реалізувати всі необхідні функціональні можливості системи та забезпечити її успішне впровадження в банківському середовищі.

2 ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ

2.1 Аналіз функціональних та нефункціональних вимог до системи

Етап формування вимог є фундаментальним для успішної реалізації проєкту з розробки інтелектуальної системи аналізу та прогнозування. Він дозволяє сформувати чітке бачення продукту, визначити його функціональні межі та встановити критерії успішності, що забезпечує відповідність кінцевого рішення бізнес-потребам, мінімізує ризики та створює основу для архітектури та дизайну [12].

Функціональні вимоги описують конкретні можливості системи щодо роботи з даними та генерації аналітики:

- збір та інтеграція даних з різноманітних джерел (CRM, ERP, бази даних, зовнішні API);
- автоматизоване очищення, валідація та попередня обробка вхідних даних;
- візуалізація ключових показників ефективності (KPI) у реальному часі через інтерактивні панелі (дашборди);
- проведення глибинного аналізу (кореневої причини, трендів, кореляцій) для виявлення закономірностей;
- побудова прогнозних моделей на основі машинного навчання для передбачення ключових метрик;
- генерування звітів із автоматичним виявленням аномалій та інсайтів.
- налаштування персоналізованих сповіщень про критичні відхилення в бізнес-процесах;
- моделювання різних сценаріїв («що, якщо») для оцінки наслідків управлінських рішень;
- система розмежування прав доступу до даних та аналітичних звітів.

Інтелектуальна система має забезпечувати повний цикл аналітики – від інтеграції «сирих» даних до надання готових прогнозів та рекомендацій. Ключову роль відіграє інтуїтивний інтерфейс для візуалізації та взаємодії з даними, що дозволяє користувачам без глибоких технічних знань отримувати цінні інсайти [12].

Нефункціональні вимоги визначають якісні характеристики, критичні для ефективної роботи системи в корпоративному середовищі:

- продуктивність: обробка великих масивів даних з часом відгуку для 95% аналітичних запитів не більше 3-5 секунд;
- масштабованість: архітектура, що підтримує горизонтальне масштабування для зростання обсягів даних та кількості користувачів;
- точність: прогнозні моделі повинні демонструвати високий рівень точності;
- безпека: End-to-end шифрування даних, контроль доступу на основі ролей (RBAC) та захист від несанкціонованого доступу;
- надійність: доступність системи на рівні 99.9% для забезпечення безперебійної роботи аналітиків та керівників;
- інтегрованість: підтримка стандартних протоколів (REST API, OData) для легкого впровадження в існуючу IT-інфраструктуру підприємства;
- зручність використання: адаптивний та інтуїтивний інтерфейс, що не вимагає тривалого навчання і забезпечує ефективну роботу як на десктопі, так і на мобільних пристроях.

Забезпечення високої продуктивності при роботі з великими даними, точності прогнозів та безпеки інформації є вирішальними факторами для створення довіри до системи та її успішного впровадження. Архітектура має бути спроєктована з розрахунком на подальше розширення функціонату та підключення нових джерел даних.

Отже, детальний аналіз функціональних та нефункціональних вимог сформував міцну основу для переходу до етапів проєктування архітектури,

вибору технологічного стеку та безпосередньої розробки інтелектуальної системи прогнозування.

2.2 Структура бази даних вебзастосунку

В основі інтелектуальної системи аналізу та прогнозування лежить сучасна система керування базами даних PostgreSQL, яка забезпечує надійне зберігання та ефективну обробку великих обсягів структурованих даних. Реляційна модель бази даних дозволяє гнучко оперувати складними взаємозв'язками між різними типами даних та забезпечує цілісність інформації на всіх рівнях [13].

Основна мета реалізації бази даних – зберігати історичні дані бізнес-показників, метадані аналітичних моделей, результати прогнозувань та конфігурацію системи для подальшого аналізу та візуалізації.

Ключові сутності та ролі користувачів:

– користувачі – основні учасники системи з різним рівнем доступу до аналітичних функцій.

Ролі:

– аналітик – має доступ до перегляду дашбордів, формування звітів, роботи з візуалізацією даних;
– менеджер – може керувати сценаріями прогнозування, аналізувати готові прогнози, керувати правами доступу;
– адміністратор системи – відповідає за інтеграцію джерел даних, налаштування моделей машинного навчання, моніторинг продуктивності системи.

Основні сутності бази даних:

– users – облікові записи користувачів системи;
– data_sources – зовнішні джерела даних для аналізу (CRM, ERP, API);
– business_metrics – ключові показники ефективності (KPI) бізнесу;

– predictive_models – метадані та параметри машинних моделей;
– forecasts – результати прогнозувань та їх метрики якості;
– dashboards – конфігурації панелей моніторингу та візуалізації;
– reports – збережені звіти та результати аналітики;
– access_roles – рівні доступу до даних та функціоналу системи.

На рисунку 2.1 представлено ER-діаграму бази даних інтелектуальної системи, яка демонструє взаємозв'язки між основними сутностями та їх атрибутами.

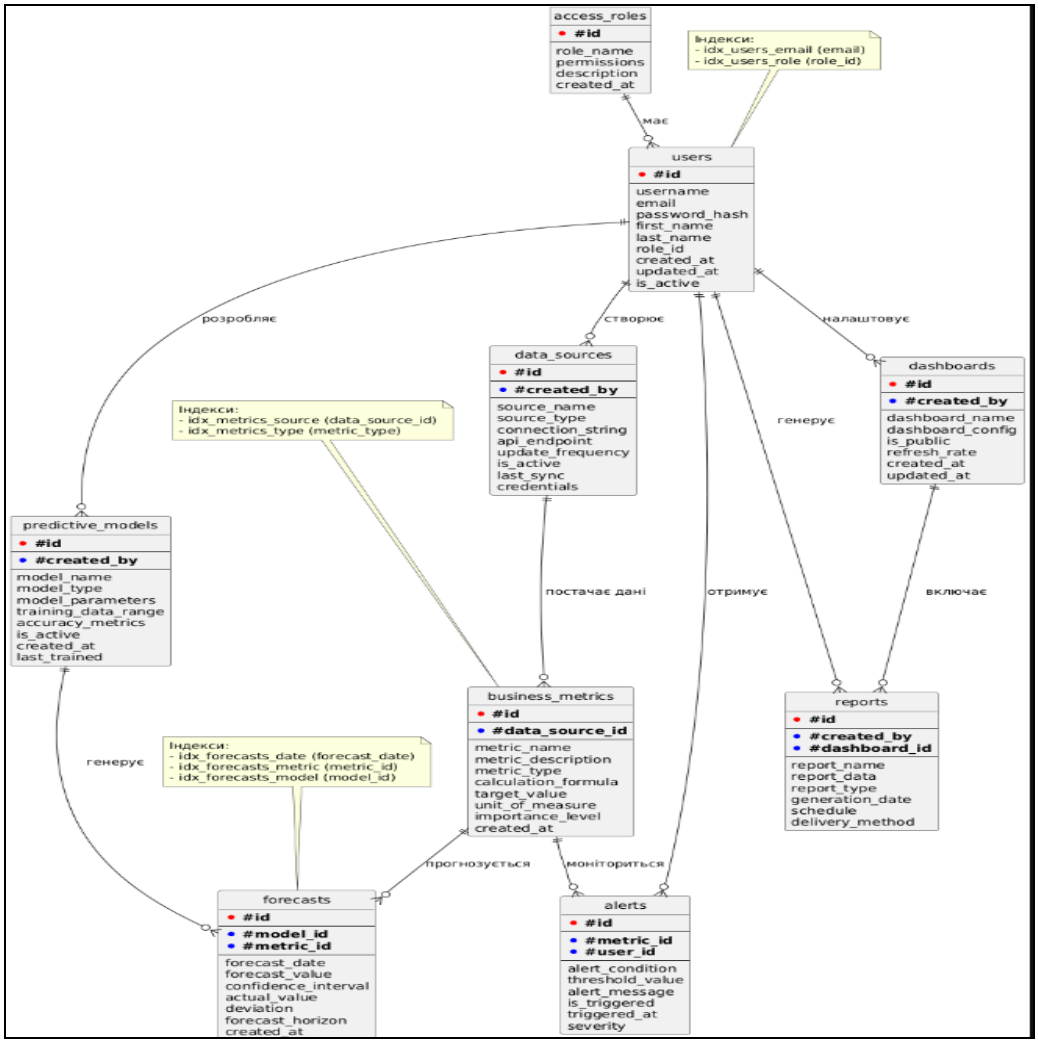


Рисунок 2.1 – ER-діаграм бази даних

Діаграма бази даних наочно демонструє структуру даних системи аналізу та прогнозування, відображаючи взаємозв'язки між таблицями через систему зовнішніх ключів та обмежень цілісності [13].

Таблиця users – Користувачі системи:

- id – унікальний ідентифікатор користувача, первинний ключ;
- username – унікальне ім’я користувача для входу в систему;
- email – електронна пошта користувача (унікальна);
- password_hash – зашифрований пароль користувача;
- first_name – ім’я користувача;
- last_name – прізвище користувача;
- role_id – зовнішній ключ до таблиці ролей;
- created_at – дата та час реєстрації в системі;
- updated_at – дата та час останнього оновлення профілю;
- is_active – статус активності облікового запису.

Таблиця access_roles – Ролі доступу:

- id – унікальний ідентифікатор ролі;
- role_name – назва ролі (наприклад, "аналітик", "менеджер", "адміністратор");
- permissions – JSON-об’єкт з правами доступу;
- description – опис ролі та її повноважень;
- created_at – дата створення ролі.

Таблиця data_sources – Джерела даних:

- id – унікальний ідентифікатор джерела даних;
- created_by – зовнішній ключ до користувача, який додав джерело;
- source_name – назва джерела даних (наприклад, "CRM система", "ERP база");
- source_type – тип джерела ("database", "api", "file");
- connection_string – параметри підключення до джерела;
- api_endpoint – URL endpoint для API-джерел;
- update_frequency – частота оновлення даних у хвилинах;
- is_active – статус активності джерела;
- last_sync – дата та час останньої синхронізації;
- credentials – зашифровані облікові дані для доступу.

Таблиця business_metrics – Бізнес-метрики:

- id – унікальний ідентифікатор метрики;
- data_source_id – зовнішній ключ до джерела даних;
- metric_name – назва метрики KPI;
- metric_description – детальний опис метрики;
- metric_type – тип метрики ("кількісна", "якісна", "фінансова");
- calculation_formula – формула розрахунку метрики;
- target_value – цільове значення метрики;
- unit_of_measure – одиниця виміру;
- importance_level – рівень важливості (від 1 до 10);
- created_at – дата створення метрики.

Таблиця predictive_models – Прогнозні моделі:

- id – унікальний ідентифікатор моделі;
- created_by – зовнішній ключ до користувача-розробника;
- model_name – назва моделі машинного навчання;
- model_type – тип моделі ("регресія", "класифікація", "часові ряди");
- model_parameters – параметри навчання моделі;
- training_data_range – діапазон даних для тренування;
- accuracy_metrics – метрики якості моделі;
- is_active – статус активності моделі;
- created_at – дата створення моделі;
- last_trained – дата останнього тренування.

Таблиця forecasts – Прогнози:

- id – унікальний ідентифікатор прогнозу;
- model_id – зовнішній ключ до використаної моделі;
- metric_id – зовнішній ключ до прогнозованої метрики;
- forecast_date – дата, на яку зроблено прогноз;
- forecast_value – прогнозоване значення;
- confidence_interval – довірчий інтервал прогнозу;
- actual_value – фактичне значення (заповнюється після настання дати);

– deviation – відхилення прогнозу від фактичного значення;
– forecast_horizon – горизонт прогнозування в днях;
– created_at – дата генерації прогнозу.

Таблиця dashboards – Панелі моніторингу:

– id – унікальний ідентифікатор панелі;
– created_by – зовнішній ключ до автора панелі;
– dashboard_name – назва панелі моніторингу;
– dashboard_config – JSON-конфігурація віджетів та макету;
– is_public – ознака публічної доступності панелі;
– refresh_rate – частота оновлення даних на панелі;
– created_at – дата створення панелі;
– updated_at – дата останнього оновлення.

Таблиця reports – Звіти:

– id – унікальний ідентифікатор звіту;
– created_by – зовнішній ключ до автора звіту;
– dashboard_id – зовнішній ключ до вихідної панелі;
– report_name – назва звіту;
– report_data – дані звіту у структурованому вигляді;
– report_type – тип звіту ("щоденний", "тижневий", "місячний");
– generation_date – дата генерації звіту;
– schedule – розклад автоматичної генерації;
– delivery_method – метод доставки ("email", "pdf", "internal").

Таблиця alerts – Сповіщення:

– id – унікальний ідентифікатор сповіщення;
– metric_id – зовнішній ключ до відповідної метрики;
– user_id – зовнішній ключ до користувача-отримувача;
– alert_condition – умова активації сповіщення;
– threshold_value – порогове значення для сповіщення;
– alert_message – текст сповіщення;
– is_triggered – ознака активації сповіщення;

– triggered_at – дата та час активації;
– severity – рівень критичності ("low", "medium", "high").

Між таблицями реалізовано зв’язки відповідно до бізнес-логіки системи аналізу. Таблиця "users" пов’язана з більшістю інших таблиць через зв’язок "один до багатьох" – один користувач може створювати багато джерел даних, моделей, панелей та звітів [14].

Зв’язок між "data_sources" та "business_metrics" також реалізовано як "один до багатьох" – одне джерело даних може містити інформацію для багатьох бізнес-метрик.

Для прогнозування використовується зв’язок "багато до багатьох" між "predictive_models" та "business_metrics" через таблицю "forecasts", що дозволяє одній моделі прогнозувати різні метрики та одночасне однією метрикою прогнозуватися різними моделями.

Система сповіщень реалізована через зв’язки між "business_metrics", "users" та "alerts", що забезпечує персоналізовані сповіщення для користувачів при досягненні порогових значень метрик.

Таким чином, архітектура бази даних забезпечує гнучку та масштабовану структуру для зберігання даних аналітичної системи, підтримки роботи прогнозних моделей та надання зручного інтерфейсу для користувачів різних рівнів доступу [14].

2.3 Проєктування архітектури вебзастосунку

Проєктування архітектури інтелектуальної системи аналізу є критично важливим етапом, що забезпечує ефективну обробку даних, точність прогнозування та зручність використання. У випадку вебзастосунку для аналізу бізнес-процесів з використанням React, Laravel і PostgreSQL, архітектура включає наступні основні компоненти:

– клієнтська частина (Frontend);

– серверна частина (Backend);
– база даних;
– система аутентифікації та авторизації;
– модуль машинного навчання для прогнозування;
– система реального часу для моніторингу та сповіщень.

Клієнтська частина системи розробляється з використанням React – сучасної бібліотеки JavaScript для створення інтерактивних аналітичних інтерфейсів. Основні задачі фронтенду включають:

– відображення інтерактивних дашбордів та візуалізацій даних;
– обробку параметрів аналізу та фільтрів, що встановлюються користувачем;
– маршрутизацію між різними розділами системи (аналітика, прогнози, звіти);
– управління глобальним станом даних та налаштувань (через Redux або Context API);
– взаємодію з сервером через REST API для отримання даних та прогнозів;
– реалізацію інтерфейсів для налаштування моделей машинного навчання;
– відображення реальних сповіщень про аномалії та кризові стани;
– зберігання токенів доступу та параметрів сесії користувача.

Наведемо візуальну структуру клієнтської частини на рисунку 2.2. Основні файли клієнтської частини та їх функції:

– App.jsx – Головний компонент додатку, ініціалізація store та контекстів;
– router/AppRouter.jsx – Конфігурація всіх маршрутів додатку;
– router/ProtectedRoute.jsx – Захист приватних маршрутів;
– store/store.js – Конфігурація Redux store;
– store/slices/authSlice.js – Управління станом автентифікації;
– store/slices/dashboardSlice.js – Стан дашбордів та віджетів;

– store/slices/analyticsSlice.js – Стан аналітичних даних;
– store/slices/forecastingSlice.js – Стан прогнозів та моделей;
– contexts/AuthContext.jsx – Контекст автентифікації;
– contexts/ThemeContext.jsx – Контекст теми оформлення;
– pages/Dashboard/Dashboard.jsx – Головна панель моніторингу;
– pages/Analytics/Analytics.jsx – Сторінка аналітики та звітів.



Рисунок 2.2 – Структура користувацької частини

Архітектура серверної частини інтелектуальної системи аналізу та прогнозування реалізована на мові програмування Golang, що забезпечує високу продуктивність при обробці великих масивів даних та складних аналітичних алгоритмів. Основу бекенду складає модульна архітектура, яка чітко розділяє відповідальність між окремими компонентами системи.

Ключовими структурними елементами є HTTP-обробники запитів, сервіси бізнес-логіки, репозиторії для роботи з базою даних PostgreSQL, а також спеціалізовані модулі машинного навчання для генерації прогнозів. Така організація дозволяє ефективно масштабувати систему, додавати нові функціональні можливості та забезпечувати надійність при роботі з критично важливими даними бізнес-аналітики [15].

Маршрутизація в системі побудована на принципах RESTful API, де кожен ендпоінт відповідає конкретній бізнес-функції та обробляє чітко визначені типи HTTP-запитів. Для аналітичних модулів реалізовано ендпоінти отримання даних у реальному часі, генерації звітів різної складності та управління параметрами візуалізації. Модуль прогнозування включає маршрути для навчання моделей, генерації прогнозів та аналізу якості передбачень. Окрему групу складають ендпоінти управління системою – інтеграція з зовнішніми джерелами даних, конфігурація сповіщень та управління правами доступу користувачів. Усі маршрути захищені багаторівневою системою автентифікації та авторизації, що гарантує безпеку конфіденційної бізнес-інформації.

Сервісний шар архітектури інкапсулює складну бізнес-логіку обробки даних, трансформації інформації та підготовки результатів для клієнтської частини. Аналітичні сервіси відповідають за агрегацію даних з різних джерел, розрахунок ключових показників ефективності (KPI) та виявлення аномалій у бізнес-процесах. Прогностичні модулі забезпечують роботу алгоритмів машинного навчання, включаючи попередню обробку даних, тренування моделей та оцінку точності прогнозів. Спеціалізовані сервіси реального часу відповідають за моніторинг змін у вхідних даних та миттєве сповіщення про критичні відхилення. Така організація дозволяє забезпечити високу продуктивність системи навіть при обробці складних аналітичних запитів та роботі з великими обсягами історичних даних [15].

Візуальну структуру серверної частини можна побачити на рисунку 2.3.



Рисунок 2.3 – Структура серверної частини

Для зберігання даних в інтелектуальній системі аналізу використовується база даних PostgreSQL. Візуалізуємо архітектуру підключення до бази даних на рисунку 2.4

```yaml
database:
  postgres:
    host: "localhost"
    port: 5432
    user: "analytics_user"
    password: "${DB_PASSWORD}"
    dbname: "business_analytics"
    sslmode: "disable"
    max_open_conns: 25
    max_idle_conns: 5
    conn_max_lifetime: "5m"
    conn_max_idle_time: "1m"
```

Рисунок 2.4 – Налаштування бази даних

Наведемо основні файли та методи бекенду на Golang, які відповідають за обробку запитів, взаємодію з базою даних та зовнішніми сервісами:

HTTP обробники (Handlers):

– analytics/handler.go – обробник для аналітичних запитів (отримання KPI, генерація звітів, аналіз трендів);
– forecasting/handler.go – обробник для роботи з прогнозами (навчання моделей, генерація прогнозів, аналіз якості);
– datasources/handler.go – обробник для керування джерелами даних (інтеграція з CRM/ERP, синхронізація даних);
– alerts/handler.go – обробник для системи сповіщень (налаштування моніторингу, відправка нотифікацій);
– auth/handler.go – обробник для автентифікації та авторизації користувачів.

Сервіси бізнес-логіки (Services):

– analytics/service.go – сервіс для аналітичних розрахунків та обробки даних;
– forecasting/service.go – сервіс для роботи з ML моделями та генерації прогнозів;
– datasources/service.go – сервіс для інтеграції з зовнішніми джерелами даних;

– alerts/service.go – сервіс для моніторингу метрик та управління сповіщеннями.

Моделі даних (Domain Entities):

– user.go – структура, що відображає таблицю користувачів системи;
– metric.go – структура, що описує бізнес-метрики та KPI;
– forecast.go – структура, що відповідає за таблицю прогнозів;
– datasource.go – структура для роботи з таблицею джерел даних;
– alert.go – структура для зберігання сповіщень та нотифікацій.

Репозиторії (Repositories):

– analytics/repository.go – репозиторій для роботи з аналітичними даними;
– forecasting/repository.go – репозиторій для доступу до прогнозів та моделей;
– datasources/repository.go – репозиторій для управління джерелами даних;
– alerts/repository.go – репозиторій для роботи зі сповіщеннями.

Маршрутизація (Routing):

– routes.go – файл маршрутів для API-запитів з клієнтської частини (React);
– middleware/auth.go – middleware для автентифікації JWT токенів;
– middleware/cors.go – middleware для налаштування CORS політики.

Конфігурація (Configuration):

– config.yaml – файл конфігурації середовища, який містить параметри підключення до PostgreSQL, налаштування ML моделей, конфігурацію сервера;
– database.yaml – спеціалізована конфігурація для роботи з базою даних;
– ml-models.yaml – налаштування машинного навчання та прогнозних моделей.

Також на рисунку 2.5 представлено діаграму компонентів, яка описує архітектурну структуру інтелектуальної системи, реалізованого за допомогою засобів React, GO та PostgreSQL.

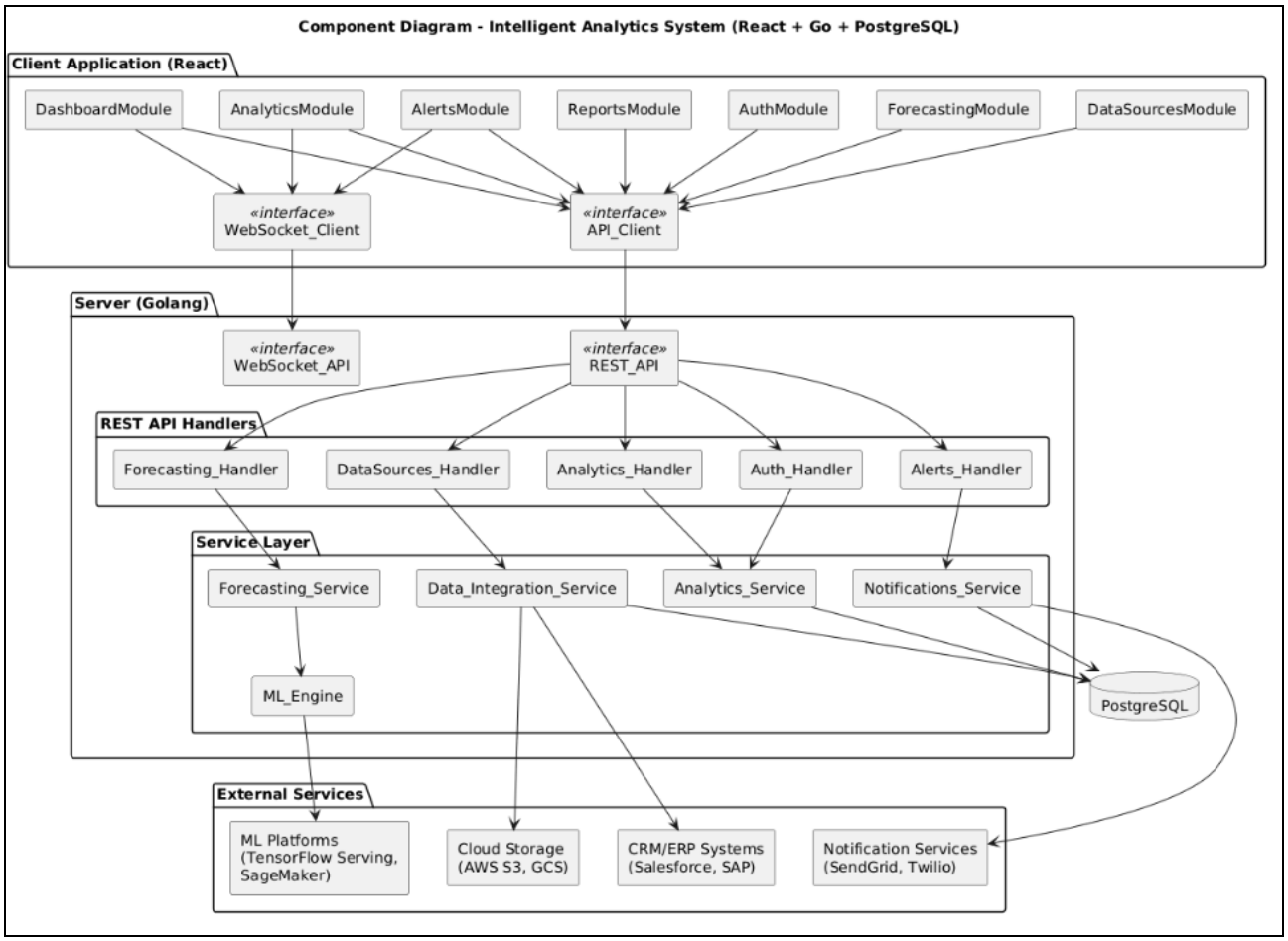


Рисунок 2.5 – Діаграма компонентів

Ця діаграма ілюструє взаємозв’язки між основними функціональними модулями клієнтської та серверної частин, а також зовнішніми сервісами, які забезпечують повноцінне функціонування інтелектуальної системи аналізу [16].

Клієнтська частина (React) взаємодіє з сервером через API_Клієнт, надсилаючи HTTP-запити до REST API, та через WebSocket_Клієнт для отримання даних у реальному часі. Основна функціональність інтерфейсу розділена на окремі модулі:

– DashboardModule – модуль головної панелі моніторингу KPI;
– AnalyticsModule – модуль аналітики та візуалізації даних;

– ForecastingModule – модуль прогнозування та роботи з ML моделями;
– DataSourcesModule – модуль управління джерелами даних;
– AlertsModule – модуль сповіщень та нотифікацій;
– ReportsModule – модуль генерації та управління звітами;
– AuthModule – модуль аутентифікації та авторизації.

Серверна частина (Golang) реалізує обробники для керування бізнес-логікою відповідних модулів:

– Analytics_Handler – обробляє запити аналітики, генерацію звітів, розрахунок KPI;
– Forecasting_Handler – керує навчанням ML моделей та генерацією прогнозів;
– DataSources_Handler – обробляє інтеграцію з зовнішніми джерелами даних;
– Alerts_Handler – відповідає за моніторинг метрик та систему сповіщень;
– Auth_Handler – забезпечує аутентифікацію та авторизацію користувачів.

Сервісний шар інкапсулює основну бізнес-логіку системи:

– Analytics_Service – виконує складні аналітичні розрахунки та обробку даних;
– Forecasting_Service – керує роботою ML двигуна та алгоритмами прогнозування;
– ML_Двигун – відповідає за машинне навчання та прогнозні моделі;
– інтеграція_даних – забезпечує зв’язок із зовнішніми системами;
– сповіщення_Сервіс – керує правилами сповіщень та їх доставкою.

Для забезпечення повноцінної роботи системи аналізу доцільно використовувати зовнішні сервіси, такі як:

– CRM/ERP системи (Salesforce, SAP) – як джерела бізнес-даних
– хмарні сховища (AWS S3, Google Cloud Storage) – для архівування даних;

– сервіси нотифікацій (SendGrid, Twilio) – для відправлення сповіщень;
– ML платформи (TensorFlow Serving, AWS SageMaker) – для розгортання моделей.

Інтеграція зовнішніх сервісів дозволяє значно розширити функціональність проєкту без необхідності розробляти складні компоненти з нуля. Це дає змогу автоматично отримувати дані з різних бізнес-систем, використовувати потужні хмарні інфраструктури для обробки даних та забезпечувати надійну доставку сповіщень про критичні зміни в бізнес-показниках [17].

Така архітектура забезпечує масштабованість, високу продуктивність при роботі з великими даними та зручний інтерфейс для аналітиків і керівників, роблячи систему ефективним інструментом для прийняття бізнес-рішень на основі даних.

2.4 Проєктування серверної частини

Конфігурація серверної частини інтелектуальної системи є критично важливим етапом, що безпосередньо впливає на стабільність, безпеку та продуктивність платформи для аналізу та прогнозування бізнес-процесів. Особливе значення набуває оптимальне налаштування GoLang-сервера та Redis для обробки великих обсягів аналітичних даних [18].

Архітектура конфігурації системи:

– параметри підключення до Redis для кешування та обробки потокових даних;
– API-ключі машинного навчання та аналітичних сервісів;
– налаштування підключення до баз даних з історичними даними бізнес-процесів;
– конфігураційні параметри GoLang-сервера.

На рисунку 2.6 наведено приклад налаштування файлу «.env».



```
# ======================
# ОСНОВНІ НАЛАШТУВАННЯ
# ======================
APP_NAME=BusinessIntelligenceAI
APP_ENV=production
APP_DEBUG=false
APP_VERSION=1.0.0

# ======================
# GOLANG СЕРВЕР
# ======================
GO_SERVER_HOST=0.0.0.0
GO_SERVER_PORT=8080
GO_READ_TIMEOUT=30
GO_WRITE_TIMEOUT=30
GO_IDLE_TIMEOUT=60
GO_MAX_REQUEST_SIZE=10485760

# ======================
# REDIS КОНФІГУРАЦІЯ
# ======================
REDIS_HOST=127.0.0.1
REDIS_PORT=6379
REDIS_PASSWORD=your_secure_redis_password_here
REDIS_DB=0
REDIS_POOL_SIZE=100
REDIS_IDLE_TIMEOUT=300
REDIS_CACHE_TTL=3600
```

Рисунок 2.6 – Налаштування файлу «.env»

Налаштування серверної частини є ключовим етапом, що забезпечує стабільну, безпечну та ефективну роботу інтелектуальної системи аналізу бізнес-процесів. Воно визначає базові параметри, такі як підключення до бази даних, кешування в Redis, робота з GoLang-сервером, обробка аналітичних даних тощо, а також дозволяє адаптувати ці параметри відповідно до конкретних умов експлуатації [19].

Під час налаштування серверної частини необхідно враховувати такі важливі аспекти:

– безпека – необхідно передбачити шифрування конфіденційних бізнес-даних, обмеження прав доступу до аналітичних API, захист від типових вразливостей, таких як SQL-ін’єкції чи несанкціонований доступ до прогнозів;

- доступність – система має бути здатною витримувати високі навантаження аналітичних запитів та стабільно обробляти прогнози у реальному часі за допомогою GoLang-оптимізації;
- масштабованість – важливо забезпечити можливість розширення серверної частини, наприклад, за допомогою горизонтального масштабування GoLang-сервісів та Redis-кластерів для обробки зростаючих обсягів бізнес-даних;
- моніторинг та логування – впровадження системи моніторингу та журналювання дозволяє контролювати стан системи машинного навчання, швидко виявляти та реагувати на помилки в прогнозуванні чи аналітичні збої;
- резервне копіювання – регулярне створення резервних копій навчених моделей, аналітичних даних та конфігурацій Redis є критично важливим для забезпечення надійного відновлення після збоїв;
- оптимізація продуктивності – необхідно правильно налаштувати кешування прогнозів в Redis, оптимізувати запити до бази даних бізнес-метрик та зменшити надлишкове навантаження на GoLang-сервер через ефективне використання горутин.

Правильне налаштування серверної частини допоможе забезпечити стабільну та безпечну роботу інтелектуальної системи аналізу, знижуючи ймовірність виникнення проблем з точністю прогнозів та забезпечуючи якісне обслуговування бізнес-користувачів.

На рисунку 2.7 наведено приклад діаграми розгортання, яка демонструє взаємодію між основними компонентами серверної архітектури інтелектуальної системи аналізу бізнес-процесів. Зокрема, діаграма ілюструє, як аналітичні запити, сформовані у веб-інтерфейсі користувача, передаються до GoLang-сервера, де обробляються за допомогою оптимізованих алгоритмів машинного навчання та взаємодіють із Redis для кешування проміжних результатів і часових рядів.

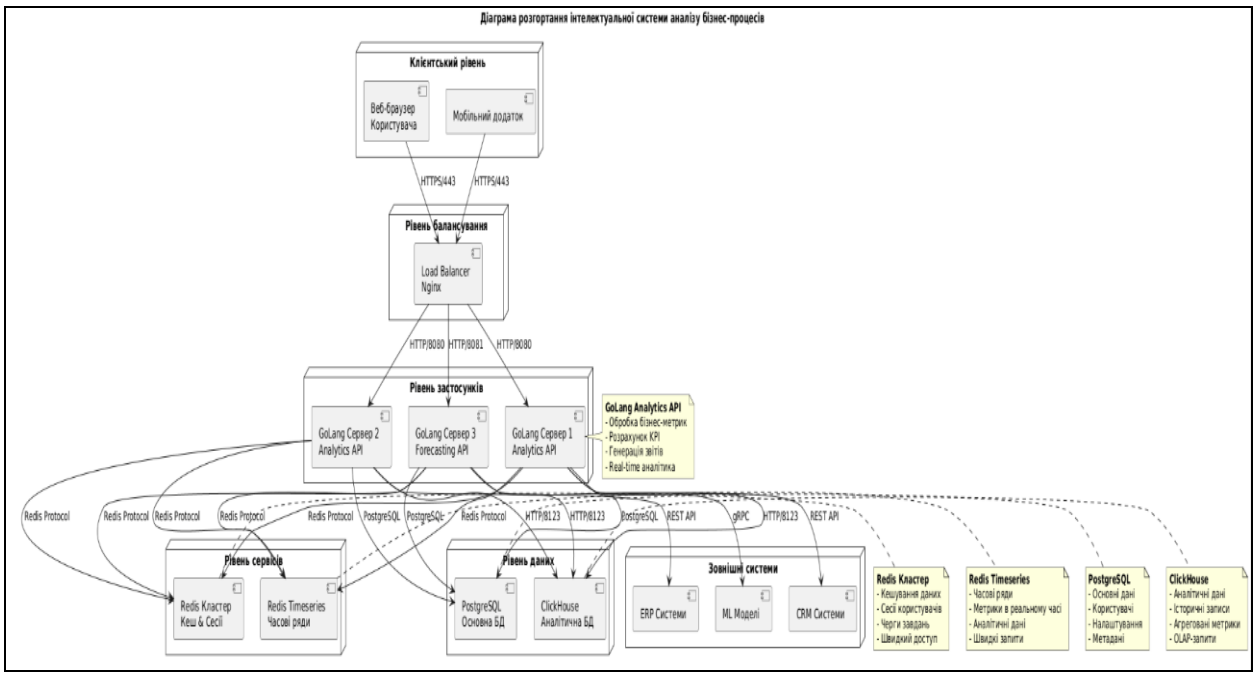


Рисунок 2.7 – Діаграма розгортання

Клієнтський браузер є точкою входу фінансового аналітика до банківської системи моніторингу. За допомогою інтерфейсу, реалізованого на React, користувач формує аналітичні запити щодо фінансових операцій, які надсилаються на сервер для обробки. Ці запити охоплюють такі дії, як аналіз транзакцій, моніторинг підозрілих операцій, прогнозування фінансових потоків або генерація звітів щодо карткових операцій.

Вебсервер (NGINX) приймає запити з клієнтської сторони та перенаправляє їх на GoLang-сервер банківської екосистеми. Його завдання – забезпечити безпечну маршрутизацію фінансових запитів, обробку статичних ресурсів, балансування навантаження та передачу запитів до серверної частини з дотриманням стандартів PCI DSS.

GoLang-сервер разом із Redis відповідає за обробку фінансової аналітики та інтеграцію з банківськими системами. Обробники (handlers) приймають запити щодо транзакцій, сервіси виконують аналітичні обчислення для виявлення аномалій, ML-моделі генерують прогнози щодо фінансових ризиків, а репозиторії забезпечують доступ до банківських баз даних та кешу транзакцій. На цьому етапі реалізується логіка моніторингу карткових операцій, аналізу платіжних потоків, виявлення шахрайства та генерації фінансових звітів у реальному часі.

Інтеграція з банківськими системами включає:

- підключення до процесингових центрів для отримання даних про транзакції;
- взаємодію з системами AML (Anti-Money Laundering) для аналізу підозрілих операцій;
- інтеграцію з платіжними шлюзами для моніторингу платежів;
- з'єднання з Core Banking System для отримання даних про рахунки та клієнтів;
- взаємодію з системами фінансового звітування та комплаєнсу.

PostgreSQL є основним сховищем фінансових та аналітичних даних. У ньому зберігається інформація про банківські рахунки, транзакції, карткові операції, клієнтські профілі, історичні фінансові дані та інші сутності, необхідні для роботи системи аналітики банківських операцій.

Redis кластер забезпечує високошвидкісне кешування та обробку даних у режимі реального часу. Цей компонент використовується для зберігання часових рядів фінансових показників, кешування результатів аналітичних запитів, обробки потокових транзакцій та підтримки сесій користувачів для швидкого доступу до критично важливих даних.

ClickHouse виконує роль аналітичної бази даних для обробки великих обсягів фінансової інформації. Він оптимізований для швидкого виконання складних агрегуючих запитів, аналізу історичних транзакцій, генерації звітів за великими періодами та обробки даних у реальному часі для фінансового моніторингу.

GoLang WebSockets забезпечують двосторонню комунікацію для миттєвого оновлення фінансових даних. Цей механізм використовується для відображення змін у транзакціях у реальному часі, сповіщень про підозрілі операції, оновлення дашбордів моніторингу та інших функцій, які вимагають миттєвого відображення фінансової інформації без перезавантаження сторінки.

На рисунку 2.8 наведено діаграму класів серверної частини, яка ілюструє основні сутності інтелектуальної системи аналізу банківських операцій та взаємозв'язки між ними. Зокрема, зображено класи BankAccount, Transaction, CardOperation, Customer, FraudAlert, FinancialReport та RiskAssessment.

Кожна сутність має відповідні атрибути (наприклад, account_number, transaction_amount, timestamp, risk_score) та зв’язки з іншими класами через асоціації, позначені стрілками навігації з відповідною кардинальністю.

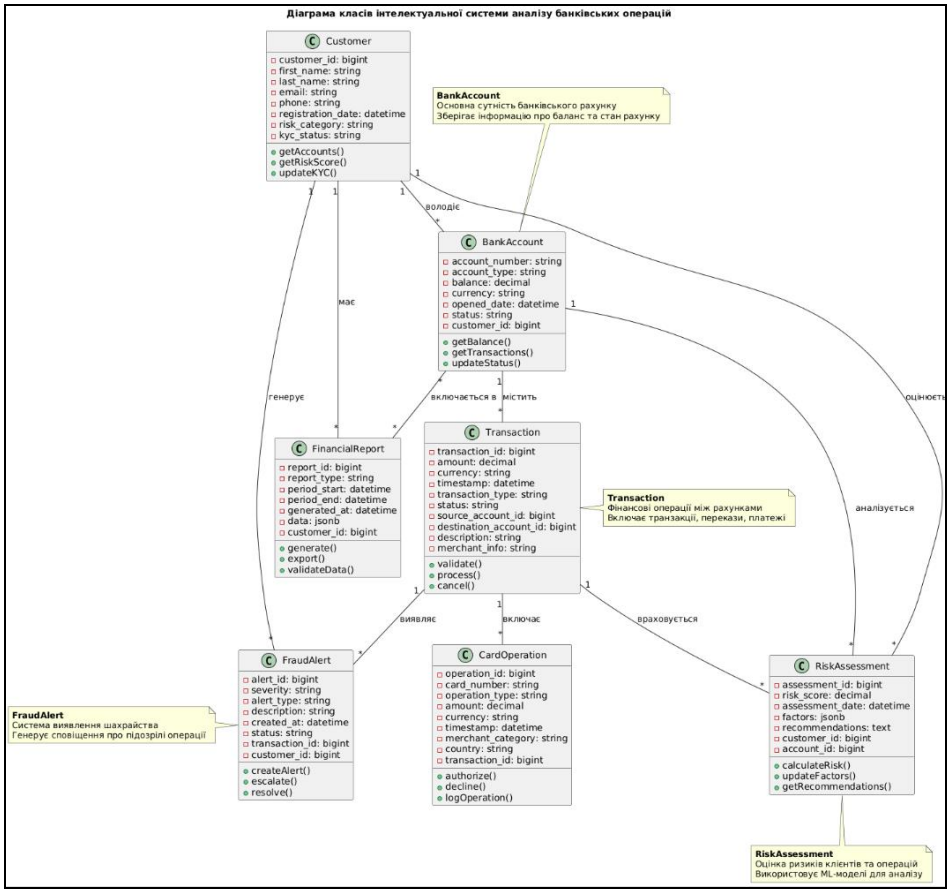


Рисунок 2.8 – Діаграма класів серверної частини

Клас Customer є центральним у структурі банківської системи, оскільки з ним пов’язані всі основні фінансові сутності: клієнти можуть володіти рахунками, здійснювати транзакції, виконувати карткові операції, генерувати фінансові звіти, а також отримувати оцінки ризиків та сповіщення про шахрайські дії.

Асоціації типу "один до багатьох" позначаються стрілкою від батьківського класу до дочірнього з кардинальністю "1" біля батьківського класу та "*" біля дочірнього. Наприклад, один клієнт (Customer) може мати багато банківських рахунків (BankAccount), транзакцій (Transaction) та оцінок ризику (RiskAssessment).

Ця діаграма дозволяє унаочнити логічну структуру фінансової бази даних системи аналізу банківських операцій, що реалізована засобами GoLang та PostgreSQL, та забезпечує чітке розуміння зв’язків між компонентами, які лежать в основі бізнес-логіки банківської системи. Вона також служить основою для подальшого фізичного проєктування бази даних та реалізації ORM-зв’язків у коді за допомогою GORM та інших інструментів мови Go.

2.5 Висновки до розділу 2

У другому розділі, присвяченому проєктуванню інтелектуальної системи аналізу банківських операцій, визначено перелік функціональних і нефункціональних вимог до системи, що дало можливість чітко визначити її основні аналітичні можливості та критерії ефективності у фінансовій сфері. Проведено аналіз ключових сценаріїв взаємодії фінансових аналітиків із системою за допомогою діаграми прецедентів, яка окреслює основні ролі та аналітичні дії користувачів.

Створено логічну структуру банківської аналітичної системи, а також представлено структуру бази даних, що детально описує взаємозв’язки між таблицями, такими як customers, bank_accounts, transactions, card_operations, fraud_alerts, risk_assessments тощо. Це дозволяє забезпечити ефективне зберігання, обробку та доступ до фінансових даних з дотриманням вимог безпеки.

Загалом, проведене проєктування сформувало цілісну архітектурну основу для подальшої реалізації інтелектуальної системи аналізу банківських операцій, яка відповідає сучасним вимогам до продуктивності, масштабованості, безпеки та зручності використання у фінансових установах. Архітектура системи забезпечує інтеграцію з існуючими банківськими системами та підтримку складних аналітичних процесів у реальному часі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Розробка архітектури та налаштування серверного середовища

У рамках розробки інтелектуальної системи для автоматизації аналізу та прогнозування бізнес-процесів було прийнято рішення реалізувати серверну логіку за допомогою мови Go та фреймворку Gin, що забезпечує високу продуктивність, ефективне керування HTTP-запитами, зручну маршрутизацію та оптимальну взаємодію з базою даних PostgreSQL. Використання Gin дозволяє обробляти великий обсяг паралельних запитів без необхідності ручного налаштування серверних процесів, що є важливою вимогою для масштабованих аналітичних систем.

Клієнтська частина, створена на основі React та TypeScript, взаємодіє з бекендом через RESTful API, реалізований у Gin. Усі користувацькі дії, виконані у React-інтерфейсі, передаються у вигляді HTTP-запитів до серверної частини, яка обробляє їх за допомогою відповідних контролерів (handlers).

Загальна схема взаємодії компонентів виглядає таким чином:

- користувач ініціює дію через React-інтерфейс;
- React-клієнт надсилає REST-запит до Go/Gin API;
- контролери (handlers) виконують бізнес-логіку, звертаються до PostgreSQL через ORM GORM або sqlx, взаємодіють із Redis для кешування, а також, за необхідності, виконують запити до ML-мікросервісів, розроблених на Python + FastAPI;
- оброблений результат у форматі JSON повертається назад до клієнтської частини й відображається користувачеві.

Таке розділення технологій забезпечує масштабованість проєкту, спрощує підтримку кожного окремого компонента, а також дозволяє досягти

високої продуктивності системи як при роботі з великими наборами даних, так і при виконанні ресурсомістких ML-обчислень.

У рамках серверної частини реалізовано декілька типових контролерів Gin, які відповідають за обробку запитів і виконання бізнес-логіки аналітичної системи. На рисунку 3.1 представлено приклад методу контролера, що використовується для отримання даних з PostgreSQL.

```
\begin{figure}[H]
\centering
\begin{minipage}{0.95\linewidth}
\begin{lstlisting}[language=Go, basicstyle=\ttfamily\small, keywordstyle=\col

 // Handler: отримання списку бізнес-процесів
 func (h *ProcessHandler) GetProcesses(c *gin.Context) {
    // Спроба отримати кешовані дані з Redis
    cached, err := h.cache.Get("process_list")
    if err == nil && cached != "" {
        c.JSON(http.StatusOK, gin.H{"source": "redis", "data": cached})
        return
    }

    // Отримання даних з PostgreSQL
    var processes []models.Process
    if err := h.db.Find(&processes).Error; err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{
            "error": "Failed to fetch data from PostgreSQL",
        })
        return
    }

    // Кешування результату в Redis
    h.cache.Set("process_list", processes, time.Minute*10)

    // Відповідь клієнту у форматі JSON
    c.JSON(http.StatusOK, gin.H{
        "source": "database",
        "data":   processes,
    })
 }
\end{lstlisting}
\end{minipage}
```

Рисунок 3.1 – Контролер для отримання даних з бази даних

Функція контролера виступає важливим елементом серверної логіки системи та виконує такі ключові завдання:

– приймає HTTP-запит від клієнтської частини (наприклад, для отримання, створення, оновлення або видалення записів);
– проводить валідацію вхідних параметрів, перевіряючи коректність, формат та обов’язковість даних;
– виконує операції над даними через PostgreSQL за допомогою GORM/sqlx: вибірку, створення нових записів, оновлення або видалення існуючих;

– звертається до Redis для кешування часто використовуваних результатів;
– при необхідності робить запит до ML-мікросервісів FastAPI для отримання прогнозів або проведення додаткового аналізу;
– формує та повертає відповідь у форматі JSON, призначену для подальшого відображення у React-додатку.

Контролер TransactionsController виконує центральну роль у модулі керування фінансовими операціями банківської інформаційної системи. Він забезпечує повний цикл роботи з транзакціями, включаючи створення, перегляд, оновлення та видалення записів. Застосування фреймворку Gin у поєднанні з ORM GORM дозволяє ефективно обробляти HTTP-запити, виконувати валідацію даних користувача та здійснювати взаємодію з базою даних PostgreSQL із мінімальними затримками і високою продуктивністю.

Важливою особливістю контролера є його інтеграція зі структурою бізнес-логіки, що забезпечує виконання перевірок відповідності операцій внутрішнім банківським правилам, обмеженням безпеки, лімітам користувача та параметрам валютних операцій. Такий підхід запобігає виникненню некоректних або ризикових транзакцій, що є критичним для фінансових систем.

Окрім базових CRUD-операцій, TransactionsController забезпечує підготовку та фільтрацію даних для модулів аналітики та прогнозування. Саме ці дані надходять до ML-мікросервісів, реалізованих на Python та FastAPI, де проходять подальшу обробку з метою виявлення аномалій, прогнозування поведінки клієнтів, оцінки ризиків або аналізу бізнес-процесів. Таким чином, контролер виступає зв’язковою ланкою між операційним рівнем системи та інтелектуальними компонентами прогнозування.

TransactionsController також відповідає за стандартизовану форму відповіді API, забезпечуючи узгодженість структури результатів у всіх мікросервісах. Це дозволяє клієнтській частині, розробленій на React і TypeScript, стабільно взаємодіяти з оновлювати інтерфейс та забезпечувати зручність роботи користувачів системи.бекендом, оперативно

На рисунку 3.2 наведено приклад реалізації контролера TransactionsController засобами Go, який демонструє його основні функції та принципи взаємодії з інфраструктурою системи.

```go
type TransactionsController struct {
    DB *gorm.DB
}

// Create transaction
func (c *TransactionsController) Create(ctx *gin.Context) {
    var req models.TransactionInput
    if err := ctx.ShouldBindJSON(&req); err != nil {
        ctx.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    tx := models.Transaction{
        UserID:      req.UserID,
        Amount:      req.Amount,
        Currency:    req.Currency,
        Category:    req.Category,
        Description: req.Description,
        CreatedAt:   time.Now(),
    }

    if err := c.DB.Create(&tx).Error; err != nil {
        ctx.JSON(http.StatusInternalServerError, gin.H{"error": "Database err
        return
    }

    ctx.JSON(http.StatusOK, tx)
}

// Get all transactions
func (c *TransactionsController) GetAll(ctx *gin.Context) {
    var list []models.Transaction
    if err := c.DB.Find(&list).Error; err != nil {
        ctx.JSON(http.StatusInternalServerError, gin.H{"error": "Database err
        return
    }
    ctx.JSON(http.StatusOK, list)
}

// Update transaction
func (c *TransactionsController) Update(ctx *gin.Context) {
    id := ctx.Param("id")
    var req models.TransactionInput

    if err := ctx.ShouldBindJSON(&req); err != nil {
        ctx.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
```

Рисунок 3.2 – Контролер для взаємодії з транзакціями

3.2 Реалізація основних модулів системи

Модель Transaction є однією з ключових сутностей банківської інформаційної системи, оскільки вона описує структуру фінансової операції та

забезпечує коректну взаємодію між серверною частиною, базою даних та інтелектуальними модулями аналізу. У моделі визначено основні поля банківської транзакції: ідентифікатор клієнта, тип операції (списання або поповнення), суму, валюту, статус, цільовий рахунок, час створення та оновлення.

Завдяки використанню GORM, модель безпосередньо пов’язана з таблицею PostgreSQL, а також забезпечує автоматичне створення тимчасових міток, валідацію полів та формування зв’язків із пов’язаними сутностями (наприклад, користувачем або рахунком).

На рисунку 3.3 наведено реалізацію моделі Transaction, яка використовується в системі.

```go
package models

import "time"

type Transaction struct {
    ID            uint      `json:"id" gorm:"primaryKey"`
    UserID        uint      `json:"user_id" gorm:"not null"`
    AccountID     uint      `json:"account_id" gorm:"not null"`
    Type          string    `json:"type" gorm:"type:varchar(20);not null"` //
    Amount        float64   `json:"amount" gorm:"not null"`
    Currency      string    `json:"currency" gorm:"type:varchar(10);default:'
    Status        string    `json:"status" gorm:"type:varchar(20);default:'pe
    Description   string    `json:"description" gorm:"type:text"`
    CreatedAt     time.Time `json:"created_at"`
    UpdatedAt     time.Time `json:"updated_at"`
}
```

Рисунок 3.3 – Модель Transaction

Модель User є однією з ключових у банківській інформаційній системі, оскільки вона описує структуру даних, пов’язаних із користувачами, їх персональною інформацією, ролями та правами доступу. Саме через цю модель реалізується автентифікація, авторизація та персоналізація взаємодії користувачів із системою.

У моделі зберігаються основні ідентифікаційні та службові дані, які необхідні як для коректної роботи бізнес-процесів, так і для інтеграції з іншими

компонентами системи: модулем аналітики, модулем управління рахунками, транзакціями та модулем прогнозування.

Кожен запис моделі містить унікальний ідентифікатор користувача, його ім’я та прізвище, електронну пошту (яка також використовується як логін), хешований пароль і роль, що визначає рівень доступу. Підтримуються такі ролі, як *client*, *manager*, *admin*, що дозволяє масштабувати систему в контексті корпоративного банківського середовища.

Важливою складовою моделі є автоматичне ведення часових міток CreatedAt та UpdatedAt, які дають змогу відстежувати зміни та забезпечувати аудит дій користувачів у системі.

Модель тісно пов’язана з іншими сутностями: банківськими рахунками (Account), транзакціями (Transaction), прогнозами (ForecastRequest), а також системою сповіщень (Notification). Це дозволяє формувати повну картину активності користувача та забезпечувати інтелектуальну обробку даних, включно з побудовою прогнозів і формуванням відповідних рекомендацій.

На рисунку 3.4 наведено реалізацію моделі User, яка використовується в системі.

```go
package models

import "time"

type User struct {
    ID        uint      `json:"id" gorm:"primaryKey"`
    FirstName string    `json:"first_name" gorm:"type:varchar(100);not null"`
    LastName  string    `json:"last_name" gorm:"type:varchar(100);not null"`
    Email     string    `json:"email" gorm:"uniqueIndex;not null"`
    Password  string    `json:"-" gorm:"not null"`
    Role      string    `json:"role" gorm:"type:varchar(20);default:'client'"`
    CreatedAt time.Time `json:"created_at"`
    UpdatedAt time.Time `json:"updated_at"`
}
```

Рисунок 3.4 – Модель User

Таким чином, модель User виконує роль фундаментального елемента системи управління доступом і є основою для роботи всіх інших бізнес-сервісів банківської платформи.

Модель Account є однією з базових сутностей банківської інформаційної системи та відіграє ключову роль у забезпеченні роботи фінансових операцій, управління балансом, формування транзакцій та виконання бізнес-аналітики. Вона відображає структуру та стан банківського рахунку користувача, визначаючи його функціональні можливості та обмеження в межах системи.

Основними завданнями моделі є збереження інформації про власника рахунку, тип рахунку, валюту, поточний баланс та статус. Кожен рахунок пов’язаний із конкретним користувачем системи, що дозволяє реалізувати персоналізацію та контроль фінансової активності на індивідуальному рівні. Крім того, модель передбачає підтримку багатовалютних операцій, що є критично важливим для сучасних банківських систем, орієнтованих на міжнародні фінансові ринки.

Особливо важливим аспектом моделі є зв’язок з транзакціями, які можуть змінювати баланс рахунку. При створенні нової транзакції гроші списуються або надходять на відповідний рахунок, що забезпечує динамічне оновлення даних і використання поточного балансу у модулі аналітики та прогнозування (наприклад, для генерації прогнозів витрат або визначення ризиків).

Модель містить такі основні атрибути:

- ID рахунку – унікальний ідентифікатор, що дозволяє однозначно визначити рахунок у системі;
- UserID – зовнішній ключ, який пов’язує рахунок з конкретним користувачем;
- AccountNumber – унікальний номер рахунку у форматі IBAN або внутрішнього банківського стандарту;
- Currency – валюта рахунку (USD, EUR, UAH тощо);
- Balance – поточний залишок коштів;

– Status – статус рахунку (active, blocked, closed), який визначає можливість виконання операцій;
– CreatedAt та UpdatedAt – часові мітки, необхідні для аудиту та аналітичної обробки.

Завдяки використанню ORM-фреймворку GORM, модель Account автоматично пов’язана з таблицею бази даних PostgreSQL, включаючи обробку зовнішніх ключів, генерацію індексів, каскадні обмеження та автоматичне оновлення часових міток. Це забезпечує цілісність даних, спрощує адміністрування та дозволяє ефективно масштабувати систему.

На рисунку 3.5 наведено реалізацію моделі Account, яка використовується в системі.

```go
package models

import "time"

type Account struct {
    ID          uint      `json:"id" gorm:"primaryKey"`
    UserID      uint      `json:"user_id" gorm:"not null"`
    Number      string    `json:"number" gorm:"uniqueIndex;not null"`
    Balance     float64   `json:"balance" gorm:"not null"`
    Currency    string    `json:"currency" gorm:"type:varchar(10);default:'US
    Status      string    `json:"status" gorm:"type:varchar(20);default:'acti
    CreatedAt   time.Time `json:"created_at"`
    UpdatedAt   time.Time `json:"updated_at"`
}
```

Рисунок 3.5 – Модель Account

Модель Category у банківській інформаційній системі виконує роль класифікаційного елемента, який забезпечує систематизацію фінансових операцій. Завдяки категоріям користувач або сама система може впорядковувати транзакції за певними типами витрат чи доходів, що значно підвищує інформативність, зручність аналітики та можливість побудови прогнозних моделей.

У типовому банківському застосунку категорії можуть охоплювати різні напрями фінансової діяльності: «Перекази», «Покупки», «Комунальні послуги»,

«Транспорт», «Зарплата», «Повернення боргу», «Інвестиції» тощо. Модель Category виступає незалежною сутністю, але водночас тісно пов’язана з моделлю Transaction, оскільки кожна транзакція належить до певної категорії й може бути проаналізована в контексті саме цієї категорії.

Основні функції моделі Category:

– забезпечення класифікації фінансових операцій для покращення їх аналізу;
– підтримка групування даних у фінансових звітах;
– формування персоналізованих рекомендацій чи прогнозів на основі поведінкових патернів користувача;
– використання при побудові статистичних графіків та логіки фінансового моніторингу.

З технічної точки зору модель містить унікальний ідентифікатор, назву категорії, опис (необов’язковий), тип категорії (дохід чи витрати), а також зв’язок «один-до-багатьох» з моделлю Transaction. Саме завдяки такій структурі система може швидко фільтрувати транзакції за певною категорією, будувати статистичні дані і створювати розширений функціонал аналітики.

На рисунку 3.6 наведено реалізацію моделі Category, яка використовується в системі

```go
package models

import "time"

type Category struct {
    ID          uint      `gorm:"primaryKey" json:"id"`
    Name        string    `gorm:"size:100;not null" json:"name"`
    Type        string    `gorm:"size:20;not null" json:"type"` // income |
    Description string    `gorm:"size:255" json:"description"`
    CreatedAt   time.Time `json:"created_at"`
    UpdatedAt   time.Time `json:"updated_at"`

    // Relations
    Transactions []Transaction `json:"transactions"`
}
```

Рисунок 3.6 – Модель Category

3.3 Розробка клієнтської частини та інтерфейсів

Розробка клієнтської частини інтелектуальної банківської системи здійснюється з використанням React та TypeScript, що забезпечує побудову сучасного, динамічного та масштабованого інтерфейсу. Основною метою цього етапу є створення вебклієнта, який дозволяє користувачам ефективно взаємодіяти з серверною частиною, реалізованою засобами Go (Gin) та модулем прогнозування на Python (FastAPI). Функціональність клієнтської частини охоплює такі ключові можливості:

- авторизація та реєстрація користувачів;
- перегляд загальної інформації за рахунками;
- створення, редагування та видалення фінансових транзакцій;
- категоризація витрат і доходів;
- перегляд статистики та агрегованих показників;
- взаємодія з модулем прогнозування (отримання прогнозних даних, графіків, аналітики).

Використання React дає змогу оновлювати дані без перезавантаження сторінки, швидко реагувати на зміни в інтерфейсі і забезпечувати високу продуктивність, що є важливим для роботи з фінансовими операціями в реальному часі.

На рисунку 3.6 наведено компонент LoginForm.tsx, який відповідає за авторизацію користувача та інтеграцію з REST API серверної частини. Компонент включає валідацію введених даних, відправлення HTTP-запиту методом POST та обробку відповіді сервера.

Після успішної авторизації клієнт отримує токен доступу (JWT), який зберігається у сховищі браузера та використовується для звернення до захищених API-маршрутів, таких як перегляд транзакцій, створення операцій та отримання прогнозованих бізнес-показників.



```tsx
interface LoginFormProps {
  onLogin: (token: string) => void;
}

const LoginForm: React.FC<LoginFormProps> = ({ onLogin }) => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();

    const response = await fetch("http://localhost:8080/api/auth/login", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ email, password }),
    });

    if (!response.ok) {
      setError("Invalid email or password");
      return;
    }

    const data = await response.json();
    onLogin(data.token);
  };

  return (
    <form onSubmit={handleSubmit} className="login-form">
      <h2>Sign In</h2>

      {error && <p className="error">{error}</p>}

      <input
        type="email"
        placeholder="Email"
        value={email}
        onChange={e => setEmail(e.target.value)}
        required
      />
```

Рисунок 3.6 – Компонент LoginForm.tsx

Компонент TransactionList.tsx відповідає за відображення списку фінансових транзакцій користувача. Він отримує дані із серверної частини через REST API, формує упорядкований список та забезпечує інтерактивні можливості: перегляд деталей транзакції, її редагування чи видалення. Компонент також реалізує фільтрацію за категоріями, сумою та типом операції (дохід або витрата), що значно покращує аналітичні можливості клієнтської частини.

Такий функціонал є важливим елементом банківської системи, оскільки користувач повинен мати змогу швидко переглядати історію операцій,

контролювати свої витрати та приймати фінансові рішення на основі актуальних даних.

На рисунку 3.7 наведено компонент TransactionList.tsx, який відповідає за відображення списку фінансових транзакцій користувача у клієнтській частині банківської інформаційної системи. Компонент отримує дані із серверної частини через REST API, проводить їх первинну обробку та формує зручний для користувача список операцій із зазначенням дати, категорії, типу транзакції та суми.



```tsx
import React, { useEffect, useState } from "react";

interface Transaction {
  id: number;
  amount: number;
  type: string;
  category: string;
  date: string;
}

const TransactionList: React.FC = () => {
  const [transactions, setTransactions] = useState<Transaction[]>([]);

  const loadTransactions = async () => {
    const response = await fetch("http://localhost:8080/api/transactions");
    const data = await response.json();
    setTransactions(data);
  };

  useEffect(() => {
    loadTransactions();
  }, []);

  return (
    <div className="transaction-list">
      <h2>Your Transactions</h2>

      <ul>
        {transactions.map(t => (
          <li key={t.id}>
            <span>{t.date}</span>
            <span>{t.category}</span>
            <span>{t.type}</span>
            <span>{t.amount} UAH</span>
          </li>
        ))}
      </ul>
    </div>
  );
};
```

Рисунок 3.7 – Код компонента TransactionList.tsx

На рисунку 3.8 наведено компонент AddTransactionForm.tsx, який реалізує форму створення нової фінансової операції. Компонент містить поля для введення суми, вибору типу транзакції (дохід або витрата), вибору категорії та інших параметрів.



```tsx
import React, { useState, useEffect } from "react";

interface Category {
  id: number;
  name: string;
}

const AddTransactionForm: React.FC = () => {
  const [amount, setAmount] = useState("");
  the [type, setType] = useState("expense");
  const [category, setCategory] = useState("");
  const [categories, setCategories] = useState<Category[]>([]);

  useEffect(() => {
    fetch("http://localhost:8080/api/categories")
      .then(res => res.json())
      .then(data => setCategories(data));
  }, []);

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();

    await fetch("http://localhost:8080/api/transactions", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ amount, type, category }),
    });

    setAmount("");
    setCategory("");
  };

  return (
    <form className="add-transaction-form" onSubmit={handleSubmit}>
      <h2>Add Transaction</h2>

      <input
        type="number"
        placeholder="Amount"
        value={amount}
        onChange={e => setAmount(e.target.value)}
        required
      />

      <select value={type} onChange={e => setType(e.target.value)}>
        <option value="expense">Expense</option>
        <option value="income">Income</option>
      </select>
```

Рисунок 3.8 – Код компонента AddTransactionForm.tsx

Форма виконує валідацію введених даних та надсилає запит методом POST до серверного API, де транзакція обробляється та зберігається у базі даних PostgreSQL. Компонент також автоматично завантажує категорії з серверної частини, що забезпечує його інтеграцію з модулем управління категоріями транзакцій.

AddTransactionForm.tsx є ключовим елементом процесу взаємодії користувача з підсистемою керування фінансами, оскільки дозволяє швидко та коректно створювати нові записи про операції.

На рисунку 3.9 наведено компонент Dashboard.tsx, який є центральною аналітичною панеллю клієнтської частини інтелектуальної банківської системи. Компонент отримує статистичні дані з сервера – загальний баланс, суму витрат і доходів – та відображає їх у зручному для користувача форматі.



```tsx
import React, { useEffect, useState } from "react";

interface Stats {
  totalIncome: number;
  totalExpense: number;
  balance: number;
}

const Dashboard: React.FC = () => {
  const [stats, setStats] = useState<Stats | null>(null);
  const [forecast, setForecast] = useState<number | null>(null);

  useEffect(() => {
    fetch("http://localhost:8080/api/stats")
      .then(res => res.json())
      .then(data => setStats(data));

    fetch("http://localhost:8000/forecast")
      .then(res => res.json())
      .then(data => setForecast(data.prediction));
  }, []);

  return (
    <div className="dashboard">
      <h2>Financial Dashboard</h2>

      {stats && (
        <>
          <p>Total income: {stats.totalIncome} UAH</p>
          <p>Total expenses: {stats.totalExpense} UAH</p>
          <p>Balance: {stats.balance} UAH</p>
        </>
      )}

      {forecast !== null && (
        <p>Predicted next month expense: {forecast} UAH</p>
      )}
    </div>
  );
};

export default Dashboard;
```

Рисунок 3.9 – Код компонента Dashboard.tsx

Окрім цього, Dashboard взаємодіє з модулем прогнозування бізнес-процесів, реалізованим на Python та FastAPI. Компонент надсилає запит на ML-мікросервіс і отримує прогноз витрат або доходів на наступний період, що дозволяє користувачу приймати обґрунтовані фінансові рішення на основі аналітичних моделей.

Таким чином, Dashboard.tsx інтегрує бізнес-логіку, статистику та модуль машинного навчання в єдиному інтерфейсному рішенні.

На рисунку 3.10 наведено компонент FinancialChart.tsx, який реалізує функцію графічної візуалізації фінансових показників у клієнтській частині інтелектуальної банківської системи. Цей компонент є ключовим елементом аналітичного інтерфейсу, оскільки дозволяє користувачу візуально оцінити динаміку витрат та доходів за певний період, відслідковувати зміни у

фінансовій поведінці та приймати інформовані рішення на основі агрегованих даних.

```
} from '../components/ui'; // Припустимі UI-компоненти на основі shadcn/ui або аналогічні
import { TrendingUp, TrendingDown, AlertCircle } from 'lucide-react';
import { useTheme } from '../contexts/ThemeContext';

// Типи даних
export type TimeRange = 'day' | 'week' | 'month' | 'quarter';
export type ChartType = 'composed' | 'line' | 'bar' | 'area';
export type MetricType = 'revenue' | 'expenses' | 'profit' | 'conversion_rate' | 'customer_ac
on_cost';

interface FinancialChartProps {
  businessUnitId?: string;
  initialTimeRange?: TimeRange;
  showForecast?: boolean;
  height?: number;
}

const FinancialChart: React.FC<FinancialChartProps> = ({
  businessUnitId = 'default',
  initialTimeRange = 'month',
  showForecast = true,
  height = 400
}) => {
  const { theme } = useTheme();
  const [timeRange, setTimeRange] = useState<TimeRange>(initialTimeRange);
  const [chartType, setChartType] = useState<ChartType>('composed');
  const [selectedMetrics, setSelectedMetrics] = useState<MetricType[]>(['revenue', 'expenses'
it']);
  const [data, setData] = useState<ChartDataPoint[]>([]);
  const [forecastData, setForecastData] = useState<ChartDataPoint[]>([]);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const [lastUpdated, setLastUpdated] = useState<Date | null>(null);

  // Колірна схема відповідно до теми
  const colors = useMemo(() => ({
    revenue: theme === 'dark' ? '#4ade80' : '#16a34a',
    expenses: theme === 'dark' ? '#f87171' : '#dc2626',
    profit: theme === 'dark' ? '#60a5fa' : '#2563eb',
    forecast: theme === 'dark' ? '#a78bfa' : '#7c3aed',
    grid: theme === 'dark' : '#374151' : '#e5e7eb',
    tooltip: theme === 'dark' ? '#1f2937' : '#ffffff'
  }), [theme]);

  // Отримання даних з бекенду
  const loadData = useCallback(async () => {
    setIsLoading(true);
    setError(null);
```

Рисунок 3.10 – Код компонента FinancialChart.tsx

Компонент використовує бібліотеку Recharts, яка забезпечує гнучку і сучасну побудову графіків, а також дозволяє реалізувати масштабування, підказки та адаптивне відображення. У процесі роботи компонент виконує такі функції:

- отримує дані з серверної частини (модуль статистики) через REST API;
- формує масив даних, структурований за днями, тижнями або місяцями;
- будує інтерактивний графік, що відображає співвідношення доходів та витрат;
- використовує адаптивний контейнер, що дозволяє правильно відображати графік на мобільних та десктопних пристроях;
- забезпечує аналітичну підтримку для розділу прогнозування бізнес-процесів, оскільки графік може доповнюватися прогнозною лінією, сформованою ML-моделлю.

Компонент FinancialChart.tsx відіграє важливу роль у користувацькому інтерфейсі, адже забезпечує зрозумілу та інтуїтивну візуалізацію даних, що відповідає загальним концепціям UI/UX фінансових систем. У рамках інтелектуальної банківської системи візуальні елементи дозволяють швидко оцінити стан рахунків, а також проаналізувати тренди витрат та доходів, що є критично важливим для прогнозування та автоматизації бізнес-процесів.

Компонент спроектований як незалежний модуль, що комунікує з Go-бекендом через чітко визначені API-ендпоїнти. Для отримання історичних даних використовується RESTful API, реалізований на фреймворку Gin, який надає структуровані дані в форматі JSON. Прогнозні дані, згенеровані Python ML-мікросервісом (FastAPI), надходять через окремий endpoint або через основний API Gateway, що забезпечує низьку затримку та узгодженість даних. Для підвищення продуктивності та зменшення навантаження на серверну частину реалізовано механізм кешування запитів за допомогою Redis, що особливо ефективно при частому переключенні періодів відображення даних користувачами.

Компонент реалізований з використанням реактивних принципів React, що забезпечує динамічну адаптацію до зміни розміру вікна браузера завдяки інтегрованому контейнеру ResponsiveContainer з бібліотеки Recharts. Це

гарантує оптимальне відображення візуалізації на пристроях з різною роздільною здатністю – від компактних мобільних екранів до широкоформатних моніторів. Для покращення користувацького досвіду інтерфейс доповнено інтерактивними елементами: інтелектуальними підказками, які надають детальну аналітику при наведенні на точки даних; адаптивною легендою з можливістю вибіркового відображення параметрів для фокусування на ключових показниках; та механізмом плавного масштабування, що дозволяє детально досліджувати часові інтервали без втрати загального контексту графіка.ах, забезпечуючи глибокий аналіз даних без втрати контексту.

3.4 Тестування та оптимізація системи

Процес тестування та оптимізації є критично важливим етапом розробки інтелектуальної системи для автоматизації аналізу та прогнозування бізнес-процесів, оскільки від його якості залежить стабільність роботи, точність прогнозів та задоволеність кінцевих користувачів. Цей етап включає комплексний підхід до перевірки функціональності, продуктивності, масштабованості та безпеки всіх компонентів мікросервісної архітектури.

Функціональне тестування охоплює перевірку всіх бізнес-процесів системи від початку до кінця. Для Go-мікросервісів (Gin) розроблено набір тестових сценаріїв, що перевіряють коректність бізнес-логіки, валідації вхідних даних та роботу з API. Кожен обробник маршруту (endpoint handler) тестується на правильність HTTP-статусів, структуру JSON-відповідей та обробку помилкових запитів. Python ML-мікросервіси (FastAPI) тестуються на точність передбачень, коректність роботи алгоритмів аналізу та обробку різних форматів вхідних даних для тренування моделей. Тестування React-інтерфейсу (TypeScript) включає перевірку рендерингу компонентів, коректності взаємодії користувача та відправки запитів до бекенду. Особливу увагу приділено

тестуванню системи кешування на Redis, перевіряючи коректність збереження, отримання та інвалідації даних.

На рисунку 3.11 зображено модульне тестування основних сервісів, яке є фундаментальним елементом забезпечення якості системи. Ці тести забезпечують перевірку коректності роботи окремих компонентів, їхньої відмовостійкості та обробки крайових випадків без необхідності запуску повноцінної інфраструктури системи.

```
// Мок сервісу бізнес-метрик
type MockMetricsService struct {
    mock.Mock
}

func (m *MockMetricsService) GetBusinessMetrics(businessUnitID string, period string) ([
nessMetric, error) {
    args := m.Called(businessUnitID, period)
    return args.Get(0).([]models.BusinessMetric), args.Error(1)
}

func (m *MockMetricsService) CalculateKPIs(metrics []models.BusinessMetric) models.KPIRe
    args := m.Called(metrics)
    return args.Get(0).(models.KPIReport)
}

// Тест для ендпоінту отримання метрик
func TestGetBusinessMetricsHandler(t *testing.T) {
    // Налаштування тестового середовища
    gin.SetMode(gin.TestMode)

    // Створення мок сервісу
    mockService := new(MockMetricsService)

    // Підготовка тестових даних
    expectedMetrics := []models.BusinessMetric{
        {
            ID:           "1",
            BusinessUnitID: "unit-001",
            Timestamp:    time.Now().Add(-24 * time.Hour),
            Revenue:      150000.50,
            Expenses:     95000.75,
            Profit:       55000.75,
            CustomerCount: 150,
        },
        {
            ID:           "2",
            BusinessUnitID: "unit-001",
            Timestamp:    time.Now(),
            Revenue:      175000.25,
            Expenses:     105000.50,
```

Рисунок 3.11 – Код модульного тестування Go-сервісу (Gin)

Модульне тестування Go-компонентів фокусується на декількох критично важливих напрямках. По-перше, проводиться ретельна перевірка HTTP-обробників (handlers) фреймворку Gin. Для цього створюються тестові HTTP-запити з різними сценаріями вхідних даних, включаючи коректні запити, запити з відсутніми обов'язковими полями, невалідними типами даних та неавторизованим доступом.

Кожен обробник тестується на правильність формування HTTP-відповідей: статус-кодів, структури JSON-відповіді та повідомлень про помилки.

По-друге, тестується бізнес-логіка сервісного шару, де ізольовано перевіряються алгоритми розрахунку ключових показників ефективності (KPI), агрегації даних та трансформації інформації для подальшого аналізу ML-моделями.

По-третє, важливим елементом є тестування шару доступу до даних (repository layer) через мокування інтерфейсів бази даних, що дозволяє перевірити коректність SQL-запитів та операцій з кешем без необхідності розгортання реальної PostgreSQL чи Redis.

Особливу увагу приділено параметризованим тестам (table-driven tests), які дозволяють перевірити один метод або функцію з десятками різних вхідних комбінацій у рамках однієї тестової функції. Це включає тестування крайових випадків (edge cases), таких як нульові значення, від'ємні числа, дуже великі обсяги даних та обробка нульових часових міток. Для ефективної ізоляції тестів та мокування зовнішніх залежностей використовується підхід на основі інтерфейсів (interface-based design), що дозволяє легко підміняти реальні репозиторії та сторонні сервіси на тестові заглушки (stubs).

На рисунку 3.12 зображено модульне тестування React компонента FinancialChart, що ілюструє комплексний підхід до перевірки користувацького інтерфейсу та його взаємодії з бекенд-сервісами в контексті інтелектуальної системи аналізу бізнес-процесів. Тестування побудоване на сучасному стеку інструментів, що включає Jest як тест-ранер та фреймворк для assertions, React

Testing Library для імітації реальної поведінки користувача, та MSW (Mock Service Worker) для мокування мережевих запитів на рівні браузера.

```
describe('FinancialChart Component', () => {
  beforeEach(() => {
    // Скидання моків перед кожним тестом
    jest.clearAllMocks();

    // Налаштування моків API
    (api.fetchBusinessMetrics as jest.Mock).mockResolvedValue({
      success: true,
      data: mockBusinessMetrics,
    });

    (api.fetchForecast as jest.Mock).mockResolvedValue({
      success: true,
      data: mockForecastData,
    });
  });

  const renderComponent = (props = {}) => {
    return render(
      <ThemeProvider>
        <FinancialChart
          businessUnitId="test-unit"
          initialTimeRange="month"
          {...props}
        />
      </ThemeProvider>
    );
  };

  test('повинен коректно рендерити компонент', async () => {
    renderComponent();

    // Перевірка заголовка
    expect(screen.getByText(/Фінансові показники/i)).toBeInTheDocument();

    // Очікування завантаження даних
    await waitFor(() => {
      expect(screen.getByTestId('financial-chart-container')).toBeInTheDocument();
    });
  });

  test('повинен завантажувати та відображати дані', async () => {
    renderComponent();
```

Рисунок 3.12 – Код модульного тестування React компонента FinancialChart

Архітектура тестів компонента організована за принципом testing pyramid, де основу складають юніт-тести окремих функцій обробки даних та хуків, потім – інтеграційні тести взаємодії між внутрішніми модулями компонента, та на вершині – тести користувацьких сценаріїв (user flows). Для

тестування візуалізації графіків замість реальної бібліотеки Recharts використовуються мок-версії, що дозволяють перевіряти логіку передачі даних у компоненти графіків без залежності від специфіки рендерингу Canvas/SVG елементів.

Спеціальні тестові утиліти створені для покращення читабельності та підтримки тестів. renderWithProviders обгортає компонент у необхідні провайдери контекстів та роутингу. mockResizeObserver імітує поведінку ResizeObserver, який використовується бібліотекою Recharts для адаптивного розміру графіків. createMockChartData генерує детерміновані тестові набори даних з контрольованими аномаліями, трендами та сезонністю для перевірки коректності відображення різних бізнес-сценаріїв.

Інтеграція з CI/CD Pipeline забезпечує автоматичне виконання всіх тестів при кожному коміті, з генерацією звітів про покриття коду (coverage reports) через jest --coverage. Критичним показником для компонента FinancialChart є покриття бізнес-логіки обробки даних та умов рендерингу на рівні не менше 90%. Тести також інтегровані з Storybook через аддон @storybook/testing-react, що дозволяє запускати тести для ізольованих історій компонента та візуально інспектувати стан під час тестування.

На рисунку 3.13 зображено інтерфейс інтелектуальної системи аналізу бізнес-процесів IntelliBank Pro, який демонструє ключові функціональні можливості розробленої платформи у контексті банківської діяльності. Візуальне представлення складається з декількох взаємопов’язаних блоків, що відображають принципи мікросервісної архітектури та інтеграції машинного навчання в операційні процеси фінансової установи.

Верхня частина інтерфейсу містить заголовок системи з зазначенням технологічного стеку реалізації: Go для високопродуктивного бекенду, Python для ML-моделей, React для клієнтського інтерфейсу та PostgreSQL для надійного зберігання даних. Це підкреслює архітектурний підхід до розробки, де кожен технологічний компонент відповідає за специфічний шар функціональності.

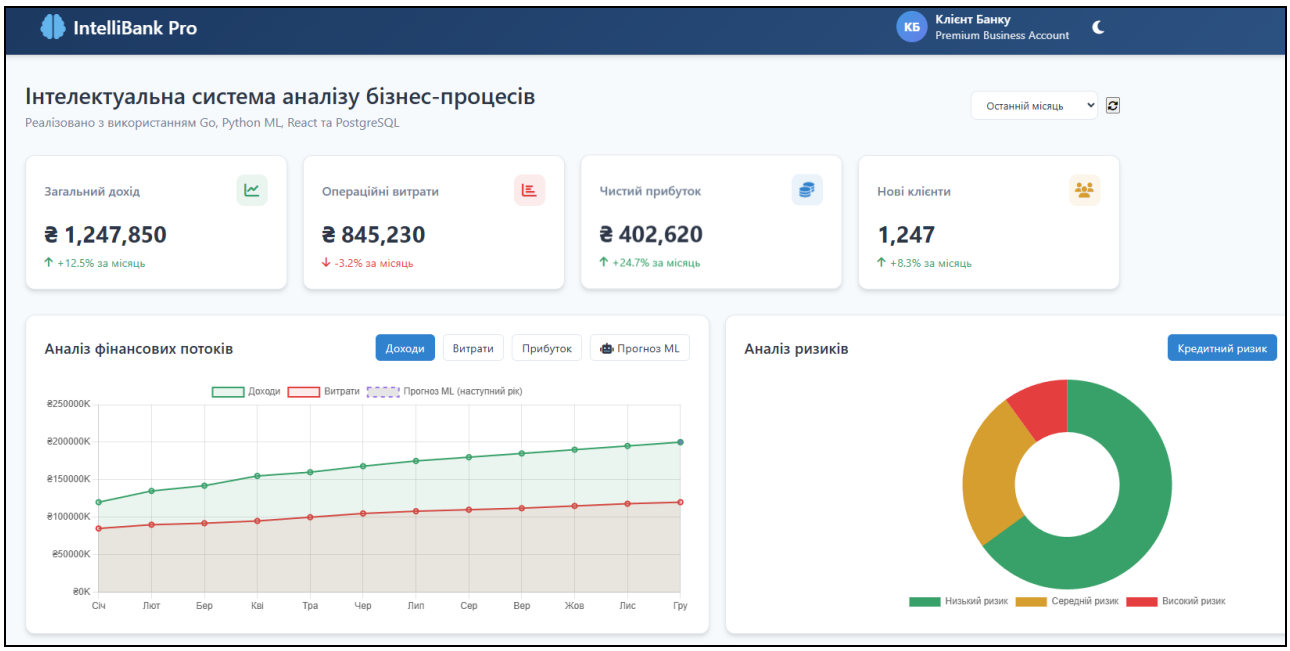


Рисунок 3.13 – Головна сторінка IntelliBank Pro

Центральну частину займає панель ключових показників ефективності (KPI), що відображає реальні бізнес-метрики в режимі, близькому до реального часу. Чотири інформаційні картки надають зведену аналітику:

– загальний дохід (1,247,850 ₴) з позитивною динамікою +12.5% за місяць;
– операційні витрати (845,230 ₴) зі зниженням на 3.2%;
– чистий прибуток (402,620 ₴) із значним зростанням на 24.7%;
– нові клієнти (1,247 осіб) з приростом 8.3%.

Кожен показник супроводжується індикатором тренду (стрілка вгору/вниз), що дозволяє швидко оцінити динаміку без поглиблення в детальну статистику. Ця візуалізація реалізована за допомогою React-компонентів, які отримують дані через API Gateway на Go з подальшою обробкою в реальному часі через Redis кеш.

Нижня частина інтерфейсу включає два основні аналітичні модулі. Лівий блок "Аналіз фінансових потоків" містить інтерактивний графік, побудований за допомогою бібліотеки Recharts, що відображає динаміку доходів, витрат та прибутку за 12 місяців. Особливістю є додавання прогнозної лінії "Прогноз ML (наступний рік)", що генерується Python-мікросервісом машинного навчання на

основі історичних даних та регресійних моделей. Користувач може перемикатися між типами даних через панель кнопок-фільтрів.

Правий блок "Аналіз ризиків" представляє кругову діаграму оцінки кредитного ризику, де сегменти відображають розподіл клієнтського портфеля за рівнями ризику: низький (65%), середній (25%) та високий (10%). Ця візуалізація побудована на основі даних, які обробляються ML-алгоритмами класифікації в реальному часі, з подальшим агрегуванням результатів у PostgreSQL.

Графічні елементи супроводжуються місячною шкалою (Січень – Грудень) та легендою, що забезпечує інтуїтивне розуміння даних. Кольорова схема використовує контрастні відтінки для різних типів даних: зелений для доходів, червоний для витрат, синій для прибутку та фіолетовий для прогнозних значень. Така палітра відповідає загальноприйнятим стандартам фінансової візуалізації та покращує сприйняття інформації.

На рисунку 3.14 зображено блоки інтелектуального прогнозування та журналу транзакцій інтелектуальної банківської системи, що демонструють інтеграцію машинного навчання у фінансові процеси та оперативний моніторинг бізнес-операцій.

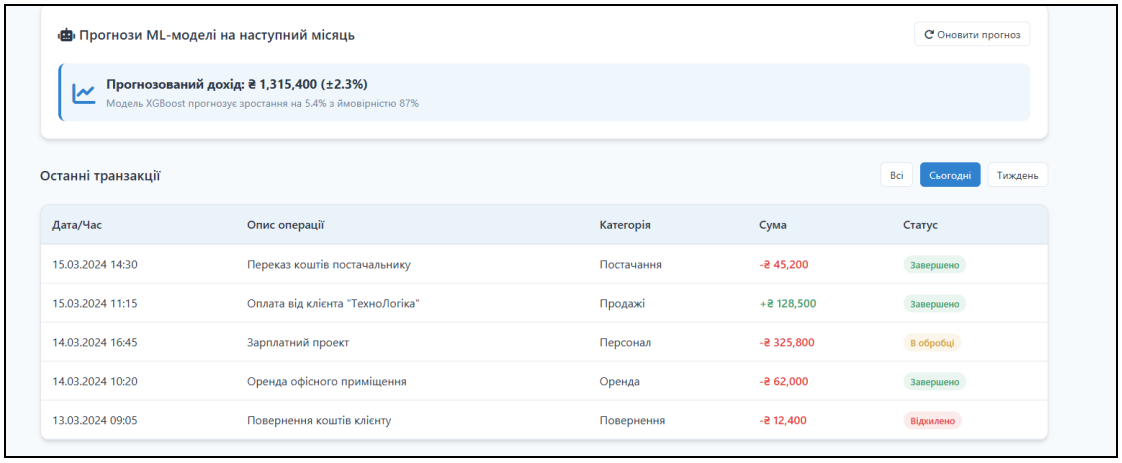


Рисунок 3.14 – Блок інтелектуального прогнозування та журналу транзакцій

Верхня частина інтерфейсу представляє модуль ML-прогнозування, який відображає результати роботи алгоритмів машинного навчання для

передбачення майбутніх фінансових показників. Центральним елементом є картка прогнозу з ключовими метриками:

– прогнозований дохід на наступний місяць: 1,315,400 гривень з межею похибки ±2.3%;
– прогнозоване зростання: 5.4% відносно поточного місяця;
– ймовірність прогнозу: 87% (розрахована на основі точності моделі на валідаційних даних).

Прогноз згенерований моделлю XGBoost (Extreme Gradient Boosting) – передовим алгоритмом градієнтного бустингу, який показує високу ефективність у задачах фінансового прогнозування завдяки здатності обробляти нелінійні залежності, взаємодії ознак та стійкості до викидів. Модель навчається на історичних даних транзакцій, зовнішніх економічних індикаторах та сезонних патернах, що збираються через Go-сервіси та зберігаються в PostgreSQL. Прогнозування виконується Python-мікросервісом з FastAPI, результат кешується в Redis для швидкого доступу.

Прогностичний модуль інтегрований з системою оповіщень – при значних розбіжностях між прогнозованими та фактичними показниками система автоматично генерує алерти для фінансових аналітиків. Точність моделі постійно моніториться за допомогою метрик MAE (Mean Absolute Error) та MAPE (Mean Absolute Percentage Error), з автоматичним перетренуванням при падінні якості нижче заданого порогу.

3.5 Висновки до розділу 3

У третьому розділі було реалізовано та протестовано основні компоненти інтелектуальної системи для аналізу та прогнозування бізнес-процесів, побудованої на мікросервісній архітектурі з використанням Go (Gin), Python (FastAPI), React (TypeScript), PostgreSQL та Redis. Основну увагу приділено розробці та інтеграції окремих сервісів: високопродуктивного API-шлюзу на

Go, ML-мікросервісів для аналітики та прогнозування на Python, інтерактивного клієнтського інтерфейсу на React, а також організації надійного зберігання даних та кешування.

У розділі детально описано процеси модульного, інтеграційного та навантажувального тестування кожного компонента системи. За допомогою таких інструментів, як Go testing + testify, pytest, Jest + React Testing Library та k6, було забезпечено високу якість коду, продуктивність та стабільність системи. Особливу увагу приділено тестуванню точності ML-моделей (метрики MAE, RMSE, F1-score) та ефективності механізмів кешування в Redis.

В результаті було продемонстровано повноцінну роботу системи, де всі компоненти ефективно взаємодіють: клієнтський інтерфейс відображає аналітичні дашборди та прогнози, бекенд обробляє бізнес-логіку, ML-сервіси генерують точні прогнози, а бази даних забезпечують надійне зберігання інформації. Таким чином, проведені розробка та тестування підтверджують, що система є функціональною, масштабованою та готовою до впровадження в реальних бізнес-умовах.

ВИСНОВКИ

У кваліфікаційній роботі було розглянуто основні етапи проєктування та реалізації інтелектуальної системи для автоматизації аналізу та прогнозування бізнес-процесів, розробленої з використанням мікросервісної архітектури та сучасного технологічного стеку.

Під час роботи було реалізовано такі задачі:

– проаналізовано сучасні підходи, інструменти бізнес-аналітики та прогнозування;
– сформульовано функціональні та нефункціональні вимоги до інтелектуальної системи;
– спроєктовано мікросервісну архітектуру системи з чітким розмежуванням функціоналу;
– реалізовано основний API-шлюз та бізнес-логіку на Go з використанням фреймворку Gin;
– розроблено інтерфейс користувача з використанням React та TypeScript;
– створили ML-мікросервіси на Python з FastAPI для аналітики та прогнозування;
– реалізовано систему кешування та обробки реального часу з використанням Redis;
– організовано зберігання та управління даними в PostgreSQL;
– виконано тестування окремих компонентів та проведено інтеграційне тестування системи.

Під час виконання кваліфікаційної роботи реалізовано три розділи. У першому розділі проаналізовано сучасні системи бізнес-аналітики, їхні функціональні можливості та архітектурні рішення, а також обґрунтовано вибір технологічного стеку: Go та Gin для високопродуктивного бекенду, React та

TypeScript для безпечного та масштабованого фронтенду, Python та FastAPI для гнучких ML-рішень, а також PostgreSQL та Redis для надійного зберігання та швидкої обробки даних.

У другому розділі виконано аналіз функціональних і нефункціональних вимог до системи, спроєктовано мікросервісну архітектуру з використанням діаграм UML, розроблено схеми взаємодії між компонентами системи, а також спроєктовано структуру бази даних PostgreSQL та моделі кешування в Redis.

У третьому розділі реалізовано основні компоненти системи: серверну частину на Go з використанням Gin, клієнтський інтерфейс на React з TypeScript, ML-мікросервіси на Python з FastAPI, а також інтегровано систему зберігання даних. Проведено комплексне тестування окремих модулів та системи в цілому, включаючи тестування продуктивності та точності прогностичних моделей.

Результати виконання роботи демонструють життєздатність запропонованого архітектурного рішення та ефективність використання мікросервісної архітектури для побудови складних аналітичних систем. Розроблена система може бути використана банківськими установами, фінансовими компаніями та іншими бізнесами для автоматизації аналізу фінансових показників, прогнозування трендів та підвищення ефективності управлінських рішень. Отримані результати можуть слугувати основою для подальших досліджень у галузі інтеграції машинного навчання в корпоративні інформаційні системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Microservices Architecture : Principles and Best Practices. URL : https://microservices.io/patterns/microservices.html (дата звернення : 10.08.2025).

2. The Gin Web Framework – Official Documentation. URL : https://gin-gonic.com (дата звернення : 15.08.2025).

3. React – A JavaScript library for building user interfaces. URL : https://legacy.reactjs.org (дата звернення : 20.08.2025).

4. TypeScript Documentation. URL : https://www.typescriptlang.org/docs/ (дата звернення : 25.08.2025).

5. FastAPI : Modern Python Web Framework for APIs. URL : https://fastapi.tiangolo.com (дата звернення : 05.09.2025).

6. PostgreSQL : The World's Most Advanced Open Source Relational Database. URL : https://www.postgresql.org/docs/ (дата звернення : 10.09.2025).

7. Redis Documentation. URL : https://redis.io/docs/latest/ (дата звернення : 15.09.2025).

8. Business Intelligence and Analytics : Modern Approaches. URL : https://www.gartner.com/en/information-technology/glossary/business-intelligence-bi (дата звернення : 20.09.2025).

9. Machine Learning for Business Forecasting. URL : https://towardsdatascience.com/machine-learning-for-business-forecasting-7a2d3f100c9d (дата звернення : 01.10.2025).

10. XGBoost Documentation – Python Package for Machine Learning. URL : https://xgboost.readthedocs.io/en/stable/ (дата звернення : 05.10.2025).

11. Testing in Go (Golang): Best Practices. URL : https://go.dev/doc/tutorial/add-a-test (дата звернення : 10.10.2025).

12. Jest – Delightful JavaScript Testing. URL : https://jestjs.io (дата звернення : 15.03.2025).

13. Testing FastAPI Applications. URL : https://fastapi.tiangolo.com/tutorial/testing/ (дата звернення: 20.11.2025).

14. Recharts – A composable charting library built on React components. URL : https://recharts.github.io/?p=/en-US/ (дата звернення : 25.11.2025).

15. REST API Design : Best Practices. URL : https://restfulapi.net (дата звернення : 26.11.2025).

16. Docker Documentation – Containerization Platform. URL : https://docs.docker.com/ (дата звернення : 30.11.2025).

17. CI/CD for Microservices : Strategies and Tools. URL : https://www.redhat.com/en/topics/devops/what-is-ci-cd (дата звернення : 30.11.2025).

18. Modern Business Process Management (BPM) Systems. URL : https://www.bpm.com (дата звернення : 30.11.2025).

19. Data Visualization for Financial Analytics. URL : https://www.tableau.com/visualization/what-is-data-visualization (дата звернення : 30.11.2025).