

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

«Аналіз та візуалізація великих обсягів даних для прийняття бізнес-рішень»

виконав студент групи ІПЗ-21-5
КОРЕНІВСЬКИЙ Данііл Павлович

Керівник роботи:
ГРОМСЬКИЙ Олександр Олександрович

Рецензент:
САВІЦЬКИЙ Роман Святославович

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КОРЕНІВСЬКИЙ Данііл Павлович**

Керівник роботи: **ГРОМСЬКИЙ Олександр Олександрович**

Тема роботи: **«Аналіз та візуалізація великих обсягів даних для прийняття бізнес-рішень»,**

затверджена Наказом закладу вищої освіти від **«23» лютого 2025 р., №74/с**

Вихідні дані для роботи: **методи та засоби розробки веб-додатків, робота з проблематикою аналізу даних продажу, використання для аналізу Python.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Громський О.О.	27.02.2025р.	27.02.2025р.
2	Громський О.О.	27.02.2025р.	27.02.2025р.
3	Громський О.О.	27.02.2025р.	27.02.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота включає розробку веб-додатку для аналітики продажів та пояснювальну записку. Пояснювальна записка містить 85 сторінки, 23 ілюстрації та 19 таблиць. Метою роботи є створення системи на базі NestJS, Python та React, яка дозволяє візуалізувати ключові бізнес-показники, аналізувати поведінку клієнтів, тренди продажів, структуру попиту тощо. Рішення покликане полегшити процес прийняття бізнес-рішень на основі даних. У роботі розглянуто основні завдання розробки системи, проведено огляд аналогічних рішень та обґрунтовано вибір технологічного стеку. Реалізовано архітектуру з чітким розділенням обов'язків: NestJS відповідає за REST API, авторизацію та обробку запитів, тоді як Python-сервіс виконує складні обчислення для аналітичної обробки даних. Інтерфейс реалізовано з використанням React + Tailwind, що забезпечує високу зручність і адаптивність.

Додаток дозволяє імпортувати дані, проводити аналіз (RFM, ABC, аналіз кошику, сезонність, вплив знижок тощо) та виводити результати у вигляді інтерактивних графіків. Проведено розгортання системи за допомогою Docker та Traefik, реалізовано систему авторизації з розподілом прав доступу.

КЛЮЧОВІ СЛОВА: АНАЛІТИКА ДАНИХ, ВЕБ-ДОДАТОК, NESTJS, PYTHON, REACT, ВІЗУАЛІЗАЦІЯ, DOCKER, TRAEFIK, БІЗНЕС-АНАЛІЗ, ПРОДАЖІ

					ІПЗ.КР.Б-121-24-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка веб додатку для аналізу та візуалізації великих обсягів даних для прийняття бізнес-рішень	Літ.	Арк.	Аркушів
Розроб.		Коренівський Д.П.						
Керівник		Громський О.О.					3	85
						Житомирська політехніка група ІПЗ-21-5		
Н. контр.		Гичина О.П.						
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

This explanatory note contains 85 pages, 23 illustrations, and 19 tables. The aim of the project is to develop a system based on NestJS, Python, and React that enables visualization of key business indicators, analysis of customer behavior, sales trends, demand structure, and more. The solution is designed to facilitate data-driven business decision-making. The paper examines the main development tasks of the system, provides a review of existing analogous solutions, and justifies the choice of the technology stack. The system architecture is implemented with a clear separation of responsibilities: NestJS handles the REST API, authorization, and request processing, while the Python service performs complex computations for data analytics. The interface is developed using React and Tailwind, ensuring high usability and responsiveness. The application allows data import and supports various types of analysis (RFM, ABC, basket analysis, seasonality, discount impact, etc.), with results displayed as interactive charts. The deployment was carried out using Docker and Traefik, and an authorization system with role-based access control has been implemented. KEYWORDS: DATA ANALYTICS, WEB APPLICATION, NESTJS, PYTHON, REACT, VISUALIZATION, DOCKER, TRAEFIK, BUSINESS ANALYSIS, SALES

ЗМІСТ

ABSTRACT.....	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	9
1.1 Еволюція підходів до прийняття управлінських рішень на основі даних	9
1.2 Поняття великих обсягів даних: характеристики, джерела та виклики обробки	11
1.3 Методи аналізу бізнес-даних: RFM, ABC, кластеризація, сезонний аналіз..	13
Висновки до першого розділу.	16
РОЗДІЛ 2. КОНЦЕПЦІЯ, АРХІТЕКТУРА ТА ІНФОРМАЦІЙНЕ ПРОЄКТУВАННЯ ВЕБ-СИСТЕМИ.....	18
2.1 Функціональні вимоги до системи прийняття бізнес-рішень.....	18
2.2 Рольова модель користувачів і моделі доступу до даних	26
2.3 Архітектура системи та вибір технологічного стеку.....	29
2.4. Впровадження захищеної інфраструктури: Docker, HTTPS, сертифікація ..	37
Висновки до другого розділу.....	37
РОЗДІЛ 3. РЕАЛІЗАЦІЯ, АНАЛІЗ РЕЗУЛЬТАТІВ І ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОЇ СИСТЕМИ.	42
3.1 Процес розгортання програмного додатку	42
3.2 Функціональна структура інтерфейсу користувача	44
3.3 Інтеграція модулів обробки даних та побудови графіків.....	48
3.4. Аналітичний сценарій RFM, ABC аналіз товарних категорій	50
Висновки до третього розділу	67
ВИСНОВКИ	69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	71
ДОДАТКИ	74

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface – прикладний програмний інтерфейс.

BI – Business Intelligence – бізнес-аналітика.

CSV – Comma Separated Values – це формат файлу (значення, розділені комами).

JWT – JSON Web Token – це відкритий стандарт для передачі даних у форматі JSON, який часто використовується для аутентифікації та авторизації в веб-додатках.

ORM – Object-Relational Mapping – технологія для взаємодії з базою даних за допомогою об'єктів мови програмування.

REST – Representational State Transfer – це архітектурний стиль для побудови веб-сервісів, який використовує HTTP-протокол для передачі даних між клієнтом та сервером.

UI – User Interface – користувацький інтерфейс. Це частина програмного забезпечення або пристрою, з якою користувач взаємодіє, наприклад, візуальні елементи (кнопки, іконки, меню) та їх розташування на екрані.

UX – User Experience – досвід користувача під час взаємодії із системою.

SQL – Structured Query Language – мова запитів до реляційних баз даних.

DTO – Data Transfer Object – об'єкт для передачі структурованих даних між шарами застосунку.

SSL – Secure Sockets Layer – криптографічні протоколи безпечної передачі даних.

SPA – Single Page Application – це тип веб-додатку, де вся взаємодія та відображення даних відбувається на одній сторінці, без перезавантаження.

B2B – Business to Business – тип продажу від бізнесу до іншого бізнесу

B2C – Business to Customer – тип продажу від бізнесу до кінцевого споживача

ВСТУП

Актуальність теми полягає у сучасному бізнес-середовищі, що характеризується високою конкуренцією та стрімкими змінами, прийняття ефективних управлінських рішень значною мірою залежить від вчасного та точного аналізу великих обсягів даних. Традиційні способи обробки звітності більше не задовольняють потреби бізнесу, що потребує оперативної візуалізації та аналітики, прогнозувати попит.

Підвищення якості прийняття рішень можливе завдяки впровадженню систем бізнес-аналітики, які дозволяють: виявляти тренди в продажах, аналізувати поведінку клієнтів, оцінювати ефективність маркетингових кампаній,

Метою розробки випускної кваліфікаційної роботи бакалавра є розробка веб-додатку для аналізу бізнес-даних, що дозволяє приймати рішення на основі візуалізації ключових показників ефективності.

1. Проаналізувати існуючі рішення, зробити додаток зручнішим для користувача.
2. Визначити вимоги щодо функціоналу додатку.
3. Визначити засоби розробки та проектування проєкту.
4. Розробити основний функціонал веб-додатку.
5. Протестувати додаток, визначити його ефективність.

Об'єктом дослідження є веб-додаток, який забезпечує збір, обробку та інтерактивну візуалізацію операційних і маркетингових даних підприємства. Така система є інструментом підтримки управлінських рішень, що дозволяє в режимі реального часу аналізувати ключові показники ефективності, виявляти тренди та формувати стратегії розвитку. Система поєднує сучасні веб-технології, засоби візуалізації та бізнес-аналітики для забезпечення гнучкої і прозорої роботи з даними.

Предметом дослідження є функціональність веб-додатку для аналізу та візуалізації великих обсягів бізнес даних для підтримки прийняття управлінських рішень, його структура, методи аналізу даних, засоби

забезпечення безпеки даних аналізу та користувачів. Дослідження також включає аналіз переваг та недоліків використання веб-додатку у порівнянні з традиційними методами аналізу даних.

Методами аналізу є сучасні підходи до обробки бізнес-даних, що дозволяють виявляти закономірності, сегментувати клієнтів і прогнозувати попит. Зокрема, використовується АВС-аналіз для класифікації товарів за їхнім внеском у загальний обсяг продажів, що допомагає визначити найважливіші позиції. RFM-аналіз оцінює клієнтів за давністю, частотою та сумою покупок, дозволяючи виокремити цінні сегменти для маркетингу. Кластеризація даних допомагає групувати схожих клієнтів або транзакції для кращого розуміння їх поведінки. Також застосовується сезонний аналіз, який дозволяє враховувати циклічні коливання попиту та планувати діяльність з урахуванням змін у часі. Сукупність цих методів формує основу для ефективного аналізу й ухвалення обґрунтованих бізнес-рішень.

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Еволюція підходів до прийняття управлінських рішень на основі даних

У процесі розвитку бізнесу підходи до прийняття управлінських рішень зазнали значної еволюції. Початково підприємці та менеджери спиралися на інтуїцію, особистий досвід або емпіричні спостереження, часто без доступу до кількісного обґрунтування. З часом, із розширенням масштабів бізнесу та ускладненням ринкових відносин, такий підхід став недостатнім. Виникла потреба у використанні більш точних та об'єктивних інструментів – звітності, фінансового аналізу, прогнозування.

У 1960–1970-х роках із появою перших інформаційних систем почався перехід до автоматизації управління, що стало основою для появи систем підтримки прийняття рішень (Decision Support Systems, DSS). Ці системи дозволяли накопичувати історичні дані, проводити базовий аналіз і будувати прості сценарії [26].

Наступним етапом стало виникнення концепції Business Intelligence (BI), яка об'єднала в собі інструменти для збору, зберігання, обробки та візуалізації інформації. BI-системи активно впроваджувалися у великих компаніях із початку 2000-х років і дозволили менеджерам не лише бачити статистику в режимі реального часу, а й виявляти приховані залежності між показниками, здійснювати моделювання та прогнозування [27].

Сьогодні розвиток технологій, зокрема хмарних обчислень, великомасштабних сховищ та алгоритмів машинного навчання, привів до нового етапу – аналітики на основі великих обсягів даних (Big Data). Поняття Big Data включає не лише величезні за обсягом інформаційні масиви, але й такі характеристики як швидкість надходження даних, різноманітність структур (таблиці, текст, мультимедіа, логи), а також необхідність їхньої цілісності та достовірності [23].

Водночас виникає нова проблема: класичні ВІ-системи орієнтовані на підготовлені дані з чітко визначеною структурою. Проте в сучасному середовищі дані часто надходять у неструктурованому або частково структурованому вигляді – наприклад, CSV-файли з помилками форматування, великі лог-файли або масиви інформації з відкритих джерел. Це створює додаткові виклики щодо очищення, трансформації та аналізу даних до їх візуалізації.

Сучасні веб-технології дозволяють створювати рішення, які об'єднують у собі функції збору, обробки та інтерактивної візуалізації. Веб-додатки, орієнтовані на бізнес-аналітику, можуть бути доступними з будь-якого пристрою, не потребують складного встановлення й дозволяють забезпечити швидкий зворотний зв'язок між користувачем і даними. Це відкриває нові можливості для малого та середнього бізнесу, який раніше був обмежений через високу вартість ліцензійних ВІ-продуктів.

Крім того, важливою перевагою сучасного аналітичного програмного забезпечення є інтерактивність. Замість статичних звітів користувачі можуть змінювати параметри фільтрації, періоди аналізу, порівнювати сегменти даних у реальному часі, що значно підвищує ефективність прийняття рішень.

На сьогодні аналітика даних не є прерогативою виключно великих корпорацій. Завдяки відкритим технологіям, open-source бібліотекам і широкій спільноті розробників створення індивідуалізованих аналітичних інструментів стало доступним навіть для невеликих команд. Саме в цьому контексті постає завдання створення веб-додатку, який дозволяє завантажувати великі обсяги даних, автоматично їх обробляти, застосовувати популярні методи бізнес-аналізу (RFM, ABC, кластеризацію, аналіз кошика, сезонність тощо), і все це – у зручному, гнучкому інтерфейсі.

Розробка такого інструменту є не лише технічним викликом, але й соціально-економічно важливим завданням, що підвищує прозорість і обґрунтованість управлінських рішень, сприяє конкурентоспроможності компаній та адаптації до сучасного ринку.

1.2 Поняття великих обсягів даних: характеристики, джерела та виклики обробки

У сучасному цифровому середовищі поняття великих обсягів даних (Big Data) охоплює значні, динамічні та різноманітні інформаційні масиви, обробка яких виходить за межі можливостей традиційних підходів до зберігання й аналізу [26, 27]. З розвитком інформаційних технологій, мобільного зв'язку, соціальних платформ та Інтернету речей обсяг даних, що генеруються користувачами, пристроями та системами в режимі реального часу, зростає експоненційно. Це зумовлює необхідність впровадження нових методів управління, зберігання, аналізу та інтерпретації цієї інформації [23, 37], включаючи автоматизовані засоби попередньої обробки, інструменти машинного навчання та інтелектуальної аналітики.

Big Data зазвичай характеризується чотирма основними аспектами: великим обсягом (volume), високою швидкістю надходження (velocity), різноманітністю структур (variety) і високою цінністю (value). Ці характеристики об'єднуються в концепцію "4V" [33], яка є загальновизнаним підходом до опису природи великих даних. До них також часто додають правдивість (veracity), що стосується достовірності джерел, і змінність (variability), яка описує непостійність структури даних у часі [27, 34]. Таке розширене бачення підкреслює складність роботи з інформацією, яка надходить з різних джерел у різні моменти часу та в різних форматах. Дані можуть бути представлені у вигляді структурованих таблиць, напівструктурованих форматів, таких як JSON або XML, або повністю неструктурованих джерел – наприклад, текстових документів, відеофайлів, зображень, лог-файлів систем або результатів взаємодій у цифровому середовищі, зокрема повідомлень у чатах, кліків користувачів та відгуків на платформах [28].

Джерелами великих даних виступають інформаційні системи підприємств, платформи електронної комерції, соціальні мережі, мобільні застосунки, сенсори Інтернету речей, а також державні відкриті реєстри [32, 39]. Усе частіше до них долучаються системи моніторингу поведінки

споживачів, дані від GPS-пристроїв, транзакційні записи та медіаконтент. Ці джерела продукують величезну кількість інформації, яку потрібно не лише зберігати, але й ефективно обробляти та аналізувати в умовах обмежених ресурсів та високих вимог до швидкодії. Зібрана інформація може бути фрагментованою, повторюваною або містити помилки, тому одним із ключових викликів у роботі з великими даними є забезпечення їхньої цілісності, достовірності та актуальності. Обробка таких даних включає своєчасне очищення від шумів, усунення дублікатів, нормалізацію структур, приведення до уніфікованого формату, придатного для подальшої аналітики, а також контроль якості та моніторинг змін у потоках даних [24, 40].

Традиційні інструменти не здатні забезпечити необхідну продуктивність при роботі з такими обсягами, тому для обробки великих даних застосовуються високопродуктивні розподілені обчислювальні системи, серед яких можна відзначити Apache Hadoop — як платформу для зберігання і пакетної обробки даних, та Apache Spark — як інструмент для швидкої обробки потоків даних в оперативній пам'яті. До цього додаються колоночні сховища, як-от ClickHouse або Google BigQuery, які оптимізовані для виконання аналітичних запитів у режимі реального часу [33, 37]. Використання таких систем дає змогу значно зменшити час на виконання обчислень та забезпечити масштабованість навіть при роботі з петабайтами інформації. Для роботи з персональними даними надзвичайно важливим є дотримання вимог до безпеки, захисту персональної інформації та конфіденційності. З цією метою використовуються сучасні методи шифрування як у процесі зберігання, так і під час передавання даних, багаторівневий контроль доступу на основі ролей, журналювання доступу, а також дотримання міжнародних стандартів, таких як GDPR, HIPAA або ISO/IEC 27001 [40].

Ще однією важливою проблемою є складність інтерпретації результатів аналізу. Навіть найточніші обчислення можуть виявитися безрезультатними, якщо не представлені у зрозумілій формі для прийняття рішень. Тому в сучасних аналітичних системах особлива увага приділяється візуалізації даних

як інструменту перетворення складних числових масивів у доступні для розуміння графіки, діаграми та інші візуальні форми. Сучасні платформи бізнес-аналітики, такі як Power BI, Tableau або Qlik, підтримують інтерактивні інформаційні панелі, дашборди, автоматичну генерацію звітів та інтеграцію з потоковими джерелами даних у режимі реального часу [22, 26].

Таким чином, великі дані перетворюються з суто технічного явища на стратегічний ресурс для підприємств і організацій різних галузей. Їх ефективне використання дозволяє виявляти приховані закономірності у поведінці споживачів, підвищувати точність прогнозування попиту, здійснювати більш гнучке управління поставаннями, оптимізувати маркетингові кампанії та значно покращити якість прийняття управлінських рішень. У перспективі, грамотне використання Big Data сприятиме підвищенню конкурентоспроможності організацій, інноваційності рішень та створенню нових моделей цінності в цифровій економіці [25, 26, 27].

1.3 Методи аналізу бізнес-даних: RFM, ABC, кластеризація, сезонний аналіз

У сучасному бізнес-середовищі аналітика даних відіграє ключову роль у підтримці управлінських рішень. Для досягнення більшої ефективності в роботі з великими масивами інформації застосовуються спеціалізовані методи, що дають змогу виявляти закономірності, сегментувати клієнтів, прогнозувати поведінку споживачів і оптимізувати внутрішні процеси. Серед таких інструментів можна виокремити RFM-аналіз, ABC-аналіз, кластеризацію, аналіз кошика, сезонний аналіз, оцінку впливу знижок, когортний аналіз тощо.

RFM-аналіз базується на трьох ключових показниках: давність останньої покупки (Recency), частота покупок (Frequency) та загальна сума витрат (Monetary). Цей підхід дозволяє визначити найцінніших клієнтів компанії та здійснювати ефективну сегментацію аудиторії. На практиці його використовують для виявлення лояльних споживачів, прогнозування повторних продажів та налаштування маркетингових кампаній [26].

АВС-аналіз класифікує товари або клієнтів відповідно до їхнього внеску у загальний дохід підприємства. Він ґрунтується на принципі Парето, що передбачає поділ на три категорії: група А – найважливіші з погляду прибутку, група В – середньої значущості, група С – з найменшим впливом. Такий метод дає змогу зосередити ресурси на ключових напрямках і раціоналізувати управління запасами, логістику або продажі [25].

Метод кластеризації дає змогу групувати клієнтів або транзакції за подібними характеристиками. Наприклад, алгоритм k-середніх використовується для розподілу за поведінковими або демографічними ознаками. Це полегшує створення персоналізованих пропозицій і покращує стратегічну взаємодію з різними групами користувачів [27].

Сезонний аналіз дозволяє виявити циклічні коливання в попиті протягом року. Виявлені тренди використовуються для планування маркетингових кампаній, управління запасами, прогнозування обсягів продажів і коригування бюджету відповідно до очікуваних змін [23].

Додатково у системі реалізовано аналіз середнього чека по регіонах, що дає змогу порівнювати купівельну спроможність у різних географічних сегментах, а також оцінку впливу знижок – інструмент, який демонструє зміну поведінки клієнтів залежно від наданих знижок. Аналіз кошика дозволяє виявити супутні товари й структуру покупок, що сприяє формуванню релевантних пропозицій [24].

Інтеграція зазначених методів у єдину аналітичну платформу дозволяє створити глибоку, багаторівневу модель аналізу, яка охоплює різні аспекти взаємодії з клієнтами, товарними категоріями й часовими факторами. Такий підхід сприяє ухваленню обґрунтованих управлінських рішень і підвищенню конкурентоспроможності бізнесу.

Зважаючи на важливість вищезгаданих методів, було проведено порівняльний аналіз існуючих рішень для аналітики даних, щоб оцінити їх функціональні можливості у контексті реалізації подібної системи.

Таблиця 1.1

Назва	Redash	Chartbrew	Grafana
Сучасний інтерфейс	-	+	-
Рольова авторизація	-	-	+
Підтримка великих обсягів даних	-	-	+
Інтеграція с API	+	+	-

Проведений аналіз показує, що жодна з розглянутих систем не забезпечує повного набору критично важливих функцій для реалізації сучасної, гнучкої та масштабованої платформи бізнес-аналітики. Таким чином, наявні інструменти лише частково задовольняють потреби користувача, що обґрунтовує доцільність створення власного веб-додатку, який поєднає сучасний інтерфейс, повну авторизацію, інтеграцію з джерелами даних та підтримку великих обсягів інформації.

У сучасних інформаційних системах ефективно збирання, обробка та візуалізація даних є ключовими етапами для побудови аналітичних рішень. Для реалізації кожного з цих етапів використовується низка інструментів і технологій, які забезпечують гнучкість, масштабованість і високу продуктивність.

На етапі збирання даних застосовуються як ручне завантаження файлів (наприклад, у форматі CSV), так і автоматизовані джерела – API, бази даних або хмарні сервіси. Зокрема, для прийому файлів у веб-застосунках використовують такі технології, як Multer на Node.js, що дозволяє обробляти багаточастинні форми [1], або FormData на боці клієнта [12]. Для валідації даних при збиранні застосовуються бібліотеки Yup [20], Zod або class-validator [8], які допомагають перевірити відповідність введених даних вимогам системи.

Встановлено, що традиційні ВІ-системи не враховують особливості роботи з неструктурованими даними, зокрема з CSV-файлами, які часто використовуються у малому та середньому бізнесі.

Проаналізовано характеристики великих обсягів даних (Volume, Velocity, Variety, Value) та основні виклики їх обробки, включаючи необхідність забезпечення цілісності, нормалізації, захисту конфіденційної інформації та доступу в реальному часі. Уточнено джерела формування великих даних: комерційні платформи, IoT-пристрої, відкриті реєстри тощо.

Окрема увага приділена методам аналізу: RFM, ABC, кластеризації, аналізу сезонності та кошика, які дозволяють глибше розуміти поведінку споживачів і формувати персоналізовані бізнес-стратегії. Було доведено, що жоден із наявних інструментів (Redash, Chartbrew, Grafana) не поєднує у собі всі критично необхідні функції: гнучкість візуалізації, рольову авторизацію, обробку великих обсягів даних і повноцінну API-інтеграцію.

Також розглянуто сучасні технології, які можуть бути використані для створення власного веб-додатку: системи обробки файлів, бібліотеки для візуалізації, фреймворки для бек- та фронт-енд розробки, а також засоби безпеки й розгортання. Це доводить доцільність створення індивідуального веб-рішення, яке враховує зазначені вимоги та усуває недоліки наявних систем.

Таким чином, сформульовано технічне й функціональне обґрунтування для розробки веб-додатку, що дозволяє проводити інтерактивний аналіз CSV-даних за допомогою вбудованих методів бізнес-аналітики та забезпечує гнучке керування й масштабованість.

РОЗДІЛ 2. КОНЦЕПЦІЯ, АРХІТЕКТУРА ТА ІНФОРМАЦІЙНЕ ПРОЄКТУВАННЯ ВЕБ-СИСТЕМИ

2.1 Функціональні вимоги до системи прийняття бізнес-рішень

У процесі реалізації дипломного проєкту, метою якого є створення веб-застосунку для аналізу та візуалізації великих обсягів бізнес-даних, було обрано сучасний стек технологій, що поєднує клієнтську частину на React, серверну логіку на NestJS із інтеграцією Python для аналітики, а також PostgreSQL як основну систему управління базами даних. Для забезпечення ефективності розробки та логічного поділу відповідальностей система реалізована за принципами клієнт-серверної архітектури.

На клієнтському рівні користувач має можливість завантажити CSV-файл із даними, що проходить валідацію за допомогою React Hook Form у поєднанні з Yup. Після цього інформація передається на сервер для подальшої обробки. Серверна частина, розроблена на базі NestJS, здійснює прийом і передає дані до окремого Python-сервісу, що реалізує аналітичну обробку. Після аналізу оброблені результати повертаються у форматі JSON, що дозволяє легко візуалізувати їх за допомогою бібліотеки Recharts.

Користувацький інтерфейс побудований із використанням TailwindCSS для гнучкої стилізації, а керування станом у застосунку реалізоване через Zustand. Це дає змогу ефективно управляти автентифікацією, збереженням сесій та загальним станом сторінок. У результаті створено платформу, яка забезпечує не лише зручний користувацький досвід, але й високу масштабованість та підтримку великого обсягу даних.

На основі функціональних і нефункціональних вимог було визначено основні сценарії взаємодії між системою та користувачами. Для ілюстрації цих сценаріїв розроблено діаграму варіантів використання, наведену на рисунку 2.1.

Діаграма демонструє три типи користувачів: неавторизованого відвідувача, авторизованого користувача та адміністратора. Перший може пройти авторизацію, щоб отримати доступ до повного функціоналу. Після входу користувач має змогу завантажувати CSV-файли, переглядати результати

бізнес-аналізу, які представлені у вигляді інтерактивних графіків і таблиць, та взаємодіяти з такими модулями, як RFM-аналіз, ABC-аналіз, аналіз повторних покупок, вплив знижок, сезонність продажів, сегментація клієнтів та інші.

Адміністратор системи володіє додатковими повноваженнями, що включають модерацію користувачів, створення реєстраційних посилань для нових користувачів, керування даними та видалення непотрібних записів. Це дозволяє ефективно підтримувати цілісність і актуальність даних у системі.

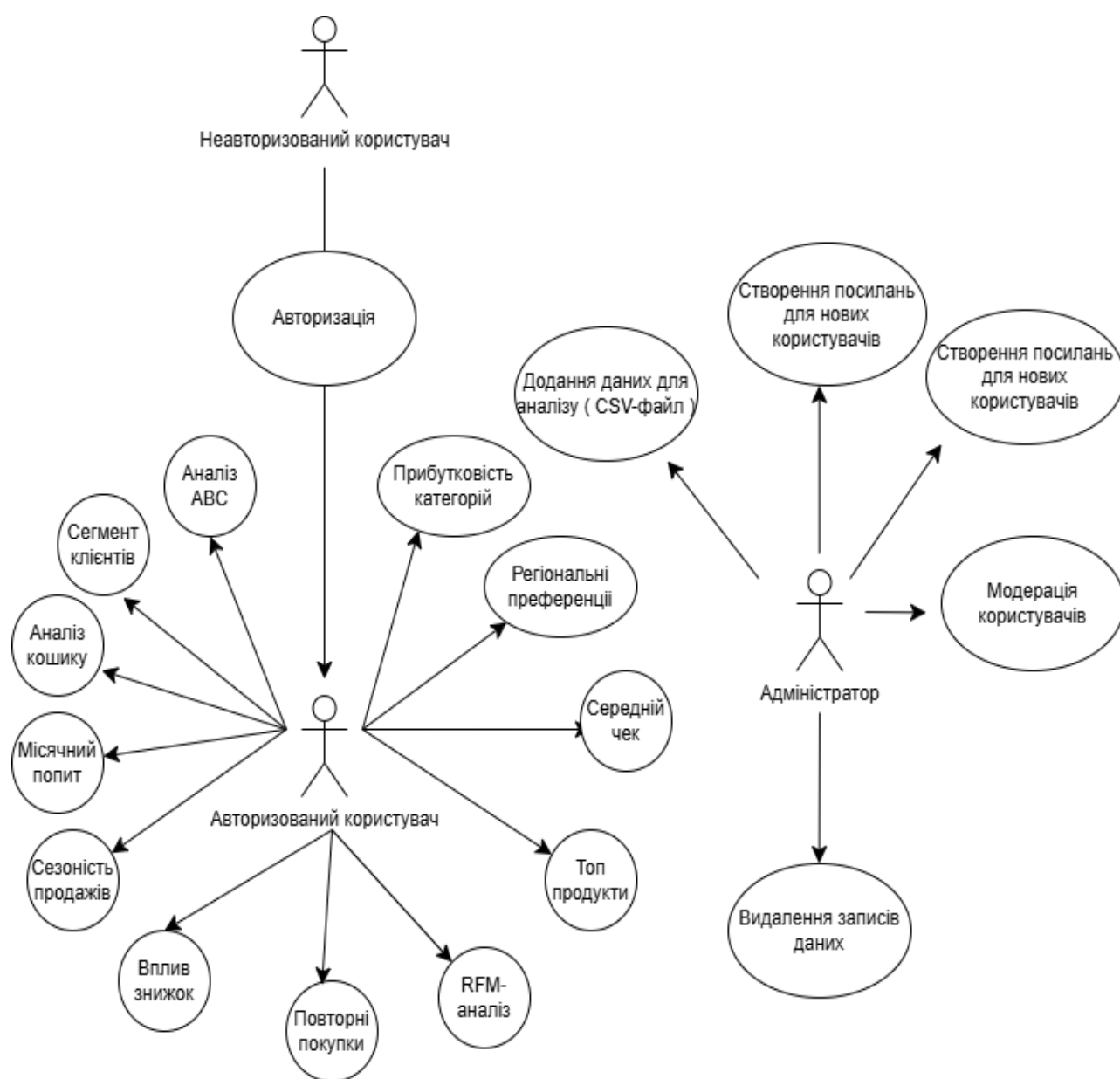


Рис. 2.1. Діаграма можливостей взаємодії з додатком для різних ролей користувачів.

Таблиця 2.1

Опис прецеденту «Авторизація»

Назва варіанту використання	Авторизація
Основні актори	Неавторизований користувач
Короткий опис	Користувач проходить процес авторизації, вводячи логін і пароль, після чого отримує доступ до функцій системи аналізу даних.

Неавторизований користувач відкриває веб-додаток і вводить свої облікові дані (логін і пароль) у відповідну форму. Після успішної перевірки даних система надає доступ до персоналізованого інтерфейсу та функціоналу, що передбачений для авторизованих користувачів. У разі помилкових даних користувач отримує повідомлення про помилку. Мета – забезпечити захищений доступ до аналітичних функцій системи.

Таблиця 2.2

Опис прецеденту «Додавання даних (по одному запису)»

Назва варіанту використання	Додати дані (по 1)
Основні актори	Адміністратор
Короткий опис	Користувач заповнює форму з деталями замовлення вручну для подальшого аналізу.

Адміністратор має можливість вручну додати окремий запис до бази даних через відповідну форму. Він вводить параметри замовлення: дату, суму, категорію товару, регіон та інші деталі. Після збереження ці дані стають доступними для подальшого аналізу. Такий варіант корисний для виправлень або невеликих доповнень до існуючих даних.

Таблиця 2.3

Опис прецеденту «Додавання даних через CSV»

Назва варіанту використання	Додати дані CSV файл
Основні актори	Адміністратор

Продовження таблиці 2.3

Короткий опис	Користувач завантажує CSV-файл із великим обсягом замовлень для масового імпорту даних у систему.
---------------	---

Адміністратор завантажує CSV-файл, що містить велику кількість записів про замовлення або транзакції. Система обробляє файл, виконує базову валідацію (чи правильна структура, формат дат, числові значення), після чого імпортує дані до бази. Це значно пришвидшує масове завантаження даних та зменшує ймовірність помилок.

Таблиця 2.4

Опис прецеденту «Аналіз АВС»

Назва варіанту використання	Аналіз АВС
Основні актори	Авторизований користувач
Короткий опис	Користувач переглядає АВС-аналіз товарів, який дозволяє класифікувати продукцію за рівнем впливу на загальний дохід.

Авторизований користувач обирає відповідну вкладку або розділ «ABC-аналіз». Система обробляє всі доступні замовлення й класифікує товари за обсягом їхнього внеску в загальний дохід. Результат подається у вигляді діаграми або таблиці з поділом на групи А, В, С, що допомагає сфокусуватися на найбільш прибуткових позиціях.

Таблиця 2.5

Опис прецеденту «Сегмент клієнтів»

Назва варіанту використання	Сегмент клієнтів
Основні актори	Авторизований користувач
Короткий опис	Користувач переглядає розподіл клієнтів за типами (B2B, B2C), що допомагає краще розуміти структуру продажів.

Користувач переглядає розподіл клієнтів за типом: B2B або B2C. Система здійснює групування за цим критерієм на основі історії покупок або реєстраційної інформації. Результат – візуалізація, яка демонструє співвідношення сегментів та дозволяє зрозуміти, який тип клієнтів переважає.

Таблиця 2.6

Опис прецеденту «Середній чек»

Назва варіанту використання	Середній чек
Основні актори	Авторизований користувач
Короткий опис	Користувач отримує візуалізацію середнього чека по регіонах для виявлення фінансової ефективності.

Користувач аналізує середній розмір покупки у різних регіонах. Система агрегує дані по кожному регіону та обчислює середній чек. Результати представлені у графіку або таблиці, що дозволяє визначити регіони з найбільшою купівельною спроможністю.

Таблиця 2.7

Опис прецеденту «Місячний попит»

Назва варіанту використання	Місячний попит
Основні актори	Авторизований користувач
Короткий опис	Користувач аналізує кількість продажів у розрізі місяців для виявлення сезонних змін.

Користувач отримує дані про кількість замовлень у розрізі місяців. Це дозволяє виявити зміни попиту протягом року. Система будує лінійний графік або гістограму, що відображає коливання кількості продажів.

Таблиця 2.8

Опис прецеденту «Сезонність продажів»

Назва варіанту використання	Сезонність продажів
Основні актори	Авторизований користувач
Короткий опис	Користувач переглядає графіки доходів по місяцях для виявлення трендів у сезонності.

На основі щомісячної виручки користувач аналізує сезонні тренди. Система обробляє дані за кілька років або кварталів та виводить закономірності у доходах, що дозволяє передбачити пікові й низькі періоди.

Таблиця 2.9

Опис прецеденту «Регіональні переваги»

Назва варіанту використання	Регіональні переваги
Основні актори	Авторизований користувач
Короткий опис	Користувач аналізує, які товари або категорії користуються попитом у різних регіонах.

Користувач переглядає інформацію про те, які категорії товарів найчастіше купують у різних регіонах. Це допомагає оптимізувати маркетингові кампанії або запаси. Система формує порівняльну таблицю або мапу.

Таблиця 2.10

Опис прецеденту «Аналіз кошику»

Назва варіанту використання	Аналіз кошику
Основні актори	Авторизований користувач
Короткий опис	Користувач бачить рекомендації по спільно купованим товарам на основі асоціативного аналізу.

Користувач бачить, які товари найчастіше купують разом. Система проводить асоціативний аналіз та пропонує пари чи групи товарів, які можуть бути рекомендовані разом. Це корисно для upsell і cross-sell стратегій.

Таблиця 2.11

Опис прецеденту «Вплив знижок»

Назва варіанту використання	Вплив знижок
Основні актори	Авторизований користувач
Короткий опис	Користувач порівнює ефективність продажів зі знижками та без них.

Користувач переглядає, як знижки впливають на обсяг продажів. Порівнюються періоди зі знижками та без них. Система показує зміну в динаміці виручки або кількості замовлень, що допомагає оцінити ефективність акцій.

Таблиця 2.12

Опис прецеденту «Повторні покупки»

Назва варіанту використання	Повторні покупки
Основні актори	Авторизований користувач
Короткий опис	Користувач переглядає, скільки клієнтів здійснюють повторні покупки, що допомагає оцінити лояльність..

Користувач аналізує, яка частка клієнтів здійснила повторні покупки, і з якою частотою. Це дозволяє оцінити рівень лояльності клієнтів та прогнозувати поведінку. Результат подається у вигляді графіка або сегментації.

Таблиця 2.13

Опис прецеденту «RFM-аналіз»

Назва варіанту використання	RFM-аналіз
-----------------------------	------------

Продовження таблиці 2.13

Основні актори	Авторизований користувач
Короткий опис	Користувач аналізує клієнтів за трьома критеріями: давністю останньої покупки, частотою та сумою витрат.

Система класифікує клієнтів за трьома критеріями: Recency (давність останньої покупки), Frequency (частота покупок) і Monetary (сума витрат). Користувач бачить, які клієнти найбільш активні та прибуткові, що дозволяє формувати цільові маркетингові стратегії.

Таблиця 2.14

Опис прецеденту «Прибутковість категорій»

Назва варіанту використання	Прибутковість категорій
Основні актори	Авторизований користувач
Короткий опис	Користувач переглядає доходи по категоріях товарів, що допомагає визначити найефективніші напрямки.

Користувач переглядає, скільки доходу принесли різні категорії товарів. Система групує дані за категоріями й обчислює суму продажів по кожній. Це дозволяє визначити найуспішніші напрями асортименту.

Таблиця 2.15

Опис прецеденту «Топ продукти»

Назва варіанту використання	«Топ продукти»
Основні актори	Авторизований користувач
Короткий опис	Користувач отримує інформацію про найбільш прибуткові товари за обраний період.

Користувач отримує список найбільш продаваних товарів за обраний період. Ця функція дозволяє швидко побачити ключові джерела прибутку та фокусувати увагу на підтримці їх доступності й просування.

2.2 Рольова модель користувачів і моделі доступу до даних

У веб-додатку реалізовано три основні класи, кожен з яких виконує окрему роль у функціональності системи.

Клас **User** відповідає за управління користувачами. Він забезпечує можливість знаходити користувача за ідентифікатором або електронною поштою, перевіряти наявність користувача з певним email, змінювати пароль, отримувати повний список усіх користувачів, а також активувати або деактивувати доступ до веб-додатку.

Клас **Analyze** реалізує логіку взаємодії з аналітичним сервісом, написаним на Python. Зокрема, він дозволяє надсилати GET-запити для отримання результатів аналізу, додавати нові записи про продажі до бази даних, видаляти записи за ID, отримувати список усіх наявних записів з можливістю пагінації та обмеження кількості, а також імпортувати дані з CSV-файлів у нормалізованому вигляді до таблиці продажів.

Клас **Authorization** відповідає за процеси автентифікації та реєстрації користувачів. Він реалізує функціональність входу в систему з перевіркою пароля та активності акаунта, надсилання електронного листа із запрошенням для реєстрації, створення нового користувача, а також зміну пароля.

Таким чином, кожен з цих класів відіграє ключову роль у загальній архітектурі веб-застосунку, забезпечуючи як керування користувачами, так і обробку аналітичних даних та захист доступу до системи.

У системі було реалізовано чіткий підхід до управління структурованими даними за допомогою серверної частини на базі NestJS та реляційної бази даних PostgreSQL. Розуміння логіки взаємодії між різними сутностями бази даних є критично важливим для забезпечення стабільної та масштабованої архітектури додатку.

NestJS відповідає за обробку запитів користувачів, створення REST API та взаємодію з базою даних через ORM TypeORM. Кожна логічна сутність у додатку реалізується у вигляді окремого модуля, який включає в себе контролер, сервіс та модель (entity), що відображає структуру відповідної таблиці в базі даних.

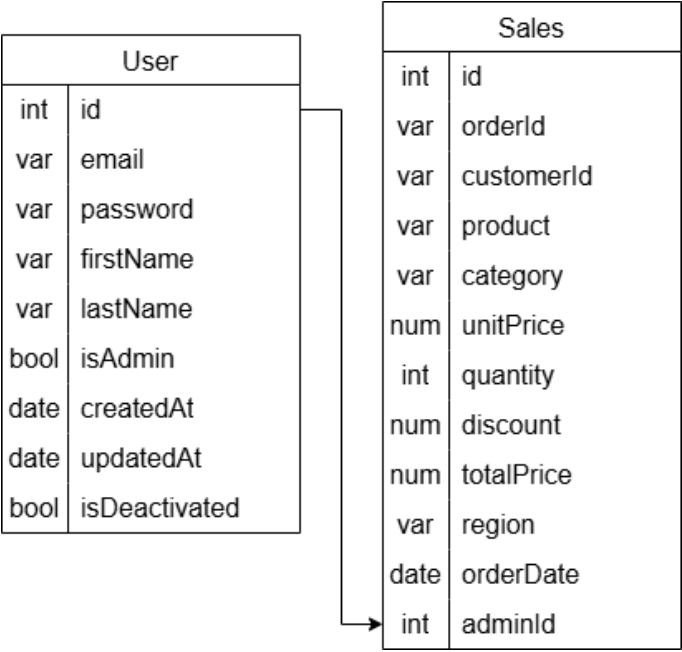


Рис. 2.2 Діаграма, що описує структуру таблиць та зв'язки між ними.

Таблиця 2.16

Опис таблиці User

Поле	Тип	Призначення
id	int	Унікальний ідентифікатор
email	var	Електронна пошта
password	var	Пароль
firstName	var	Ім'я
lastName	var	Прізвище
isAdmin	bool	Чи є адміном
createdAt	date	Дата створення
updatedAt	date	Дата оновлення

isDeactivated	bool	Чи є деактивованим
---------------	------	--------------------

Таблиця 2.17

Опис таблиці Sales

Поле	Тип	Призначення
id	int	Унікальний ідентифікатор
orderId	var	Ідентифікатор замовлення
customerId	var	Ідентифікатор покупця
product	var	Назва продукту
category	var	Категорія
unitPrice	numeric	Ціна продукту за одиницю
quantity	int	Кількість продукту
discount	numeric	Знижка
totalPrice	Numeric	Загальна ціна продукту
region	var	Регіон
orderDate	date	Дата замовлення
adminId	int	Ідентифікатор адміна, який додав запис

Між У межах системи реалізовано чітко структуровану рольову модель доступу, що визначає рівень прав і можливостей користувачів залежно від їхньої ролі. Такий підхід забезпечує як безпеку даних, так і контроль за діями всередині платформи, дозволяючи розмежовувати доступ до функцій і ресурсів відповідно до службових обов'язків або рівня відповідальності.

Основними ролями в системі виступають адміністратор та звичайний користувач. Адміністратор має повний контроль над функціоналом платформи: доступ до панелі керування, можливість запрошення нових користувачів,

управління правами доступу, перегляд і модифікація даних, контроль над результатами аналітики та можливість деактивації облікових записів. Користувач зі своєї сторони має обмежений доступ: він може лише завантажувати дані, переглядати результати аналізу й взаємодіяти з інтерфейсом у межах встановлених дозволів.

Контроль доступу до даних реалізується шляхом перевірки авторизаційного токена, виданого під час входу в систему. Кожен запит до захищених ресурсів супроводжується перевіркою ролі користувача, після чого система вирішує, чи надавати доступ до відповідного контролера API або функції в інтерфейсі.

Рольова модель також передбачає можливість подальшого розширення – наприклад, додавання ролей аналітика, менеджера або гостя – що дозволяє масштабувати систему відповідно до потреб організації. Такий підхід підвищує гнучкість, сприяє зменшенню ризиків, пов'язаних із неконтрольованим доступом до критичних даних, і відповідає загальноприйнятим принципам безпеки (principle of least privilege).

2.3 Архітектура системи та вибір технологічного стеку

У межах реалізації веб-додатку для аналізу та візуалізації великих обсягів даних було застосовано сучасний технологічний стек, який поєднує потужні інструменти для обробки, агрегації та наочного представлення даних. Такий підхід забезпечує гнучкість, масштабованість і високу продуктивність системи. На клієнтській частині використовується бібліотека React, яка дозволяє створювати динамічний і зручний інтерфейс користувача. Для візуалізації даних застосовується Recharts – легка і функціональна бібліотека для побудови графіків, що підтримує інтерактивність та адаптивність до різних типів пристроїв. Реалізовано компоненти для дашбордів, гістограм, лінійних і кругових діаграм, які дозволяють користувачам ефективно взаємодіяти з аналітичною інформацією. На серверній частині використовується NestJS – прогресивний фреймворк на базі Node.js, який завдяки своїй модульній архітектурі забезпечує зручну побудову масштабованих API. Основна логіка

бек-енду реалізує маршрути для отримання даних, авторизацію, обробку запитів користувачів та підготовку аналітичної інформації.

Окремі обчислювально-складні операції, зокрема статистичний аналіз, кластеризація або прогнозування, делегуються до сервісу на Python, що дає змогу використовувати потужні бібліотеки (наприклад, Pandas, NumPy, SciPy або scikit-learn). Це дозволяє поєднувати ефективність TypeScript-орієнтованої архітектури NestJS з аналітичними можливостями Python. Система реалізована з урахуванням захисту даних, обмеження доступу, кешування, пагінації та оптимізації запитів до бази даних. Завдяки цьому вона здатна працювати з великими наборами інформації без втрати швидкодії. Таке поєднання технологій забезпечує зручність використання, точність аналітики та високий рівень наочності, що є критично важливим для прийняття обґрунтованих бізнес-рішень на основі великих обсягів даних.

У цьому проекті клієнтський додаток взаємодіє виключно з серверним додатком на NestJS. Коли клієнтська частина надсилає запит на отримання або додання даних, серверний додаток обробляє цей запит та відправляє дані на аналіз в Python отримання необхідної інформації. Після того як Python повертає результати аналізу, NestJS форматує дані у зручний JSON формат та повертає його клієнту.

Аналітична частина програмного застосунку реалізована у вигляді окремого сервісу, створеного з використанням мови програмування Python та фреймворку FastAPI, що забезпечує обробку запитів через REST-інтерфейс. Такий підхід дозволяє ізолювати обчислювальні процеси, пов'язані з аналізом даних, від основної логіки веб-додатку, що позитивно впливає на масштабованість і підтримуваність системи. Обробка аналітичної інформації базується на використанні бібліотек pandas, scikit-learn, а також інструментів SQLAlchemy для з'єднання з базою даних PostgreSQL. Основною функцією аналітичного модуля є обробка CSV-даних або інформації, отриманої з бази, з подальшим виконанням статистичних обчислень, сегментації, класифікації та

побудови показників ефективності. Ці завдання є типовими у сфері бізнес-аналітики, що підтверджується джерелами.

Після отримання даних за допомогою SQL-запиту інформація завантажується у внутрішній кешований датафрейм, що дозволяє уникнути дублювання запитів до бази при повторному аналізі. Первинна обробка передбачає стандартизацію назв колонок, перетворення форматів дат і створення додаткових полів, необхідних для часової агрегації. Надалі користувач через API викликає конкретні аналітичні ендпоінти, кожен з яких реалізує один з методів аналізу. Наприклад, модуль ABC-аналізу базується на класифікації елементів за кумулятивною часткою у виручці (метод Парето), що дозволяє виявити критично важливі категорії товарів. Цей метод активно використовується у логістиці та маркетинговій аналітиці. Інші модулі забезпечують обрахунок RFM-метрик, аналіз повторних покупок, сезонності продажів, регіональних уподобань клієнтів, а також оцінку середнього чека та впливу знижок.

Всі обчислення виконуються на основі агрегованих даних, підготовлених із використанням бібліотеки `pandas`, яка є стандартним інструментом у галузі аналітики даних завдяки високій швидкодії та підтримці табличних операцій. Отримані результати повертаються у форматі JSON і передаються назад до NestJS через HTTP-запити, після чого надсилаються до клієнтської частини для візуалізації. Такий підхід дозволяє не лише зменшити навантаження на основний сервер, а й забезпечує гнучкість при впровадженні нових алгоритмів, не змінюючи логіку взаємодії клієнта з API. Таким чином, застосування FastAPI у зв'язці з Python-аналітикою є ефективним інженерним рішенням для побудови модульних систем аналізу даних у веб-застосунках.

Перевагами інтеграції Python та NestJS є розділення відповідальностей. Використання NestJS як основного серверного фреймворку дозволяє зосередитися на розробці основної логіки додатку, реалізувати REST API, авторизацію та керування користувачами, водночас Python відповідає виключно

за аналітичну обробку даних. Такий підхід забезпечує чітке розмежування обов’язків, що підвищує масштабованість і підтримуваність системи.

Для організації взаємодії між окремими частинами системи було спроектовано архітектурну схему, зображену на рисунку 2.2. Вона відображає ключові компоненти веб-застосунку та шляхи обміну даними між ними.

На рівні клієнта реалізовано односторінковий додаток, побудований за допомогою React. Усі запити передаються через API, створене на основі NestJS, що виконує роль посередника між інтерфейсом, базою даних і аналітичним ядром. Для зберігання та отримання інформації використовується PostgreSQL, яка забезпечує надійне управління даними. Аналітичний модуль на Python, у свою чергу, обробляє вхідні файли, виконує складну бізнес-логіку та повертає результати у зручному форматі JSON.

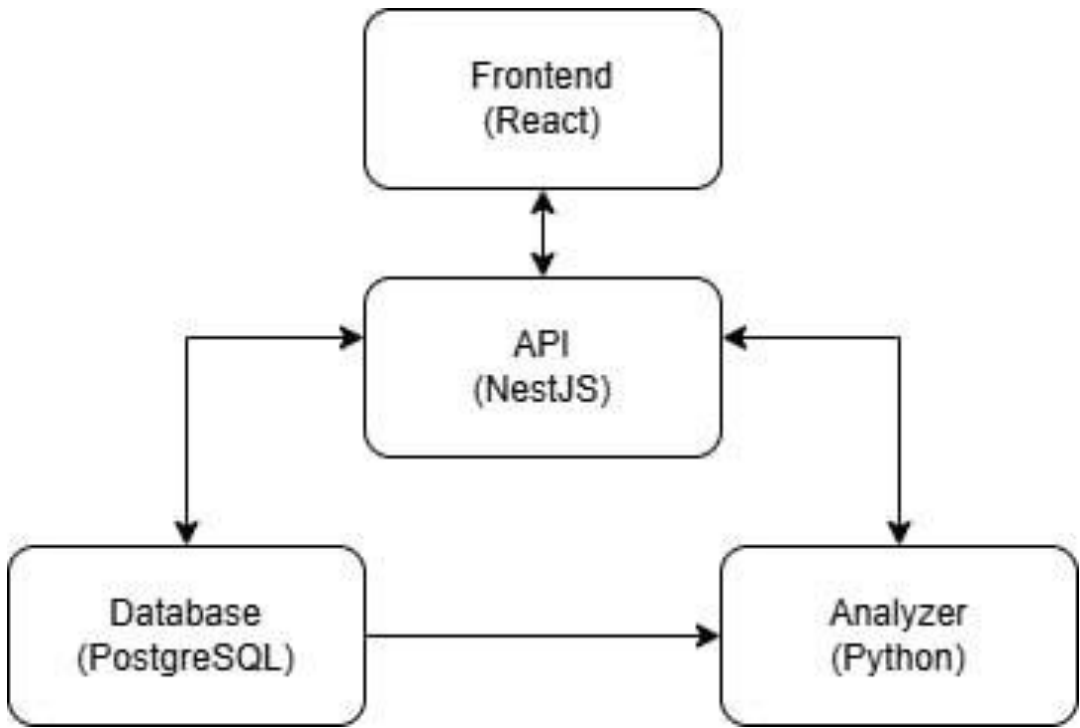


Рис. 2.3. Схема архітектури програмної системи. Ця схема наочно демонструє, як сервіси веб-додатку взаємодіють між собою.

Великою перевагою для бізнес-аналітики є використання комбінації NestJS + Python вона ідеально підходить для аналітичних додатків тому що, Nest JS забезпечує швидкий API, захист і рольову модель доступу, Python виконує всі обчислення, а інтерфейс (React) отримує готові результати, які можна легко відобразити в графічному вигляді.

Інтеграція NestJS з Python дозволяє у майбутньому легко підключити нові модулі, розширити сервіс Python в окремий кластер, якщо потрібно, наприклад, для GPU обчислень.

Python є визнаним стандартом у сфері **data science** та **аналітики**. Його використання як окремого модуля в архітектурі веб-додатку забезпечує високу точність, гнучкість і продуктивність при роботі з великими обсягами даних. Python має широкий спектр бібліотек для статистичного аналізу, машинного навчання та візуалізації (pandas, NumPy, scikit-learn, matplotlib тощо), що робить його універсальним інструментом для бізнес-аналітики.

NestJS, у свою чергу, є сучасним серверним фреймворком на основі Node.js з підтримкою **Dependency Injection**, що забезпечує модульність і високу структурованість коду. Він дозволяє будувати стабільні REST або GraphQL API, які легко масштабуються та інтегруються з іншими модулями, зокрема з аналітичними сервісами на Python.

Для підтвердження доцільності використання Python як основного аналітичного модуля було проведено порівняльний аналіз із двома іншими поширеними інструментами: **R** та **Power BI**. Результати наведено у таблиці 2.1.

Таблиця 2.18

Критерій	Python	R	Power Bi
Підтримка ML / AI	+	+	-
Розширюваність (модулі, API)	+	-	-
Масштабованість	+	-	-
Гнучкість інтеграції(з бек-ендом)	+	-	-
Підходить для веб-додатків	+	-	-

Python демонструє перевагу за більшістю ключових критеріїв, зокрема в гнучкості, масштабованості, підтримці AI та можливості обробки великих обсягів даних. Це робить його найбільш придатним для використання у складі сучасної веб-системи бізнес-аналітики.

Серверна частина на NestJS, легко інтегрується з аналітичним модулем Python через HTTP-запити, що дозволяє обробляти великі обсяги даних на стороні Python, не перевантажуючи основний бек-енд. Також у майбутньому інтеграцію можна легко масштабувати за допомогою асинхронних черг (RabbitMQ), якщо потрібно буде підвищити продуктивність системи.

NestJS дозволяє централізовано реалізовувати автентифікацію та авторизацію, обмежуючи доступ до аналітичного функціоналу лише для авторизованих користувачів. Таким чином, обробка бізнес-даних залишається під надійним контролем.

Авторизація в систему відбувається на бек-енді за допомогою JSON Web Token (JWT). JWT є стандартом для створення токенів, які використовуються для передачі інформації про користувача між клієнтом та сервером без необхідності зберігання цієї інформації на сервері:

```
async signIn(email: string, password: string) {  
  const user = await this.userService.findByEmail(email);  
  
  if (!user) {  
    throw new NotFoundException();  
  }  
  if (!user.verifyPassword(password)) {  
    throw new UnauthorizedException();  
  }  
  if (user.isDeactivated) {  
    throw new ForbiddenException();  
  }  
  const payload = { userId: user.id, isAdmin: user.isAdmin };  
  return {  
    access_token: await this.jwtService.signAsync(payload),  
  };  
}
```

Реєстрація користувача:

- Користувач вводить свої реєстраційні дані (пароль, ім'я та інше);
- Після успішної реєстрації серверна частина генерує JWT, який містить зашифровану інформацію про користувача;

- Згенерований токен передається клієнту та зберігається на стороні користувача.

Для підтвердження електронної пошти після реєстрації користувача система надсилає на вказану адресу лист із унікальним посиланням. Це посилання генерується серверною частиною застосунку та містить зашифровані дані, необхідні для верифікації. Після переходу за посиланням автоматично надсилається запит на сервер, де відбувається розшифрування та перевірка інформації. У разі успішної верифікації статус користувача оновлюється, що означає завершення процесу реєстрації. Надсилання листа виконується за допомогою зовнішнього поштового сервісу.

Використання JWT та сервісу Mailjet у веб-додатку забезпечує низку важливих переваг, які підвищують як безпеку, так і зручність роботи з системою. По-перше, з точки зору безпеки, JWT дозволяє передавати зашифровану інформацію про користувача у вигляді токена, що усуває необхідність зберігати такі дані на сервері. Це мінімізує ризики витоку особистої інформації. Додатково, інтеграція із сервісом Mailjet дозволяє реалізувати перевірку електронної пошти, що гарантує достовірність зазначених контактних даних користувача.

З технічного боку, використання токенів значно спрощує реалізацію механізмів автентифікації, а підключення зовнішнього поштового сервісу, такого як Mailjet, оптимізує процес підтвердження реєстрації. Це сприяє підвищенню ефективності роботи системи авторизації, її продуктивності та стабільності.

З погляду користувача, процес реєстрації та підтвердження електронної пошти є простим і зрозумілим, не потребує зайвих дій і забезпечує комфортну взаємодію з веб-додатком, що є важливим чинником у забезпеченні позитивного користувацького досвіду.

Запропонована схема забезпечує надійний та захищений процес авторизації, що дозволяє надавати доступ до системи виключно тим користувачам, які підтвердили свою електронну адресу.

Механізм відновлення пароля також реалізовано із використанням поштового сервісу Mailjet. У випадку втрати пароля користувач має можливість ініціювати запит на його скидання, після чого на вказану електронну адресу автоматично надсилається лист із посиланням для створення нового пароля.

Процес скидання пароля відбувається наступним чином. Користувач ініціює запит, ввівши свою електронну адресу у відповідну форму. Після цього серверна частина формує унікальне посилання, що містить зашифровані дані, необхідні для ідентифікації користувача. Це посилання надсилається електронною поштою за допомогою сервісу Mailjet. Користувач переходить за наданим URL, що відкриває інтерфейс для введення нового пароля. Після підтвердження, сервер оновлює облікові дані користувача та завершує процедуру відновлення доступу.

У процесі створення веб-застосунку для аналізу бізнес-даних одним із ключових етапів стало проєктування структури бази даних, яка забезпечує надійне зберігання, обробку та швидкий доступ до інформації. Зважаючи на специфіку задач, було обрано реляційну модель даних з реалізацією на базі PostgreSQL – СУБД, що поєднує високу продуктивність, масштабованість і підтримку складних SQL-запитів.

Архітектура бази даних передбачає наявність кількох основних сутностей, зокрема користувачів, записів про продажі, інформації про товари, клієнтів і операцій. Структура реалізована за принципами нормалізації, що дозволяє уникати дублювання даних і підтримувати логічну узгодженість між таблицями. Особливу увагу приділено підтримці історичних даних, оскільки аналітика базується на часових вибірках і агрегатах.

Проєктування моделей у системі виконано за допомогою TypeORM – ORM-фреймворку для TypeScript, який інтегрується з NestJS та дозволяє працювати з базою даних у вигляді об'єктів. Це забезпечує зручну взаємодію з сутностями, автоматичне створення зв'язків і гнучку масштабовану архітектуру серверної логіки.

Критично важливою складовою є модуль авторизації, побудований на основі JWT-токенів у NestJS. Він реалізує розмежування прав доступу, контроль сесій, блокування користувачів і підтримку ролей, що забезпечує безпеку даних відповідно до сучасних стандартів.

Для імпорту великих обсягів даних реалізовано обробку CSV-файлів, де виконується автоматична валідація полів через class-validator, а також очищення, перетворення форматів і агрегація даних. Це значно підвищує якість підготовки інформації до подальшого аналізу в Python-модулі, який використовує бібліотеки pandas та NumPy.

На фронтенді використовуються сучасні інструменти – ReactJS, TailwindCSS, Zustand – для створення швидкого, адаптивного та зручного інтерфейсу. Візуалізація реалізується через Recharts і Chart.js, що дозволяє будувати інтерактивні графіки ключових бізнес-показників.

Взаємодія між фронтендом і сервером здійснюється через REST API або GraphQL. З боку інфраструктури використовується Docker Compose для локальної розробки, а Traefik – для балансування навантаження та керування HTTPS-сертифікатами.

Таким чином, структура бази даних і обробка даних у даному проєкті відповідають вимогам надійності, розширюваності та ефективності роботи з великими обсягами інформації. Вони дозволяють забезпечити високоякісну реалізацію бізнес-логіки та повноцінну підтримку аналітичних процесів, заснованих на сучасних технологіях обробки даних.

2.4. Впровадження захищеної інфраструктури: Docker, HTTPS, сертифікація

У межах розробки веб-застосунку для аналізу та візуалізації бізнес-даних важливою складовою є забезпечення захищеного середовища для його розгортання та експлуатації. Захищена інфраструктура дозволяє не лише гарантувати цілісність і конфіденційність інформації, а й підвищити загальну надійність системи. Уразливості в архітектурі можуть стати причиною витоку даних, зниження довіри з боку користувачів та порушення регуляторних норм.

Саме тому в межах даного проєкту особлива увага приділялася створенню безпечної, стабільної й масштабованої інфраструктури.

Для досягнення цієї мети у проєкті впроваджено контейнеризацію з використанням технології Docker, налаштовано передачу даних через захищені протоколи HTTPS, передбачено процес сертифікації для підтвердження автентичності сервера, а також впроваджено зворотний проксі Traefik для керування вхідним трафіком. Така комбінація технологій дозволяє ефективно розмежовувати рівні доступу, централізовано керувати маршрутами запитів і забезпечити високий рівень безпеки на всіх етапах взаємодії клієнта з системою.

Контейнеризація за допомогою Docker дозволяє ізолювати окремі компоненти застосунку – клієнтську частину, серверну логіку та аналітичний модуль – у незалежні середовища, які легко масштабуються та переносяться. Завдяки цьому значно спрощується процес розгортання системи на будь-яких середовищах (локальних, хмарних або гібридних), адже програмне забезпечення працює в межах уніфікованих середовищ. Крім того, використання контейнерів мінімізує вплив помилок одного сервісу на роботу інших, оскільки ізоляція обмежує поширення збоїв. Такий підхід також дозволяє оновлювати окремі частини системи без повного зупинення всього застосунку, що особливо важливо для безперервної обробки даних.

Для захисту переданих даних між клієнтом і сервером використовується протокол HTTPS, який забезпечується шляхом генерації SSL-сертифікатів. Впровадження шифрування дозволяє запобігти перехопленню конфіденційної інформації на етапі її передачі через відкриті мережі. Це особливо критично у випадках, коли обробляються дані про фінансові операції або персональна інформація клієнтів. Сертифікати Let's Encrypt отримуються і оновлюються автоматично через зв'язку з Traefik, що знижує ризик людських помилок під час їх генерації й усуває потребу в ручному втручанні адміністратора.

Реалізація захищеної інфраструктури є ключовим аспектом на шляху до створення надійного та безпечного сервісу для обробки бізнес-даних. Захист від

зовнішніх загроз, таких як атаки типу Man-in-the-Middle, спроби несанкціонованого доступу або ін'єкцій, дозволяє забезпечити стабільність та довготривалу роботу платформи. Крім цього, наявність сертифікованого каналу зв'язку позитивно впливає на репутацію сервісу в очах користувачів, які все частіше звертають увагу на наявність HTTPS і відповідних сертифікатів безпеки.

Під час розгортання веб-додатку було використано зв'язку Traefik, Docker та Docker Compose, що дозволило забезпечити ефективне управління мікросервісною архітектурою. Traefik, як сучасний зворотний проксі-сервер, автоматично виявляє нові сервіси у Docker-середовищі та налаштовує відповідні правила маршрутизації без потреби у ручному оновленні конфігурацій. Це особливо важливо в динамічних середовищах, де сервіси часто оновлюються або масштабуються. Крім маршрутизації, Traefik також виконує роль балансувальника навантаження, що дозволяє рівномірно розподіляти запити та уникати перевантаження окремих компонентів.

Однією з ключових переваг використання Traefik є автоматичне отримання та оновлення SSL-сертифікатів через Let's Encrypt, що дозволяє підтримувати безпечне з'єднання без додаткових налаштувань. Завдяки інтеграції з Docker, Traefik миттєво реагує на зміни конфігурацій сервісів, і, відповідно, забезпечує безперервну доступність веб-застосунку. Така гнучкість дозволяє масштабувати систему горизонтально, додаючи нові екземпляри сервісів без зупинки всієї платформи, що позитивно впливає на SLA (Service Level Agreement) і стабільність роботи в пікові періоди.

На рівні серверного середовища на базі Ubuntu було встановлено Docker і Docker Compose, після чого сформовано конфігурацію сервісів у файлі docker-compose.yml. У цьому файлі чітко визначено всі залежності між мікросервісами, параметри середовища, відкриті порти, обмеження доступу, а також правила маршрутизації через Traefik. DNS-записи, створені для відповідних піддоменів, дозволяють точно спрямовувати користувацькі запити до відповідних контейнерів. Автоматизація оновлення SSL-сертифікатів

забезпечує постійний захист каналу передачі даних, не вимагаючи додаткових дій з боку адміністратора.

Інтеграція зазначених технологій дала змогу побудувати гнучке, надійне та захищене середовище для функціонування веб-застосунку з високим рівнем автоматизації. Це суттєво зменшило витрати на ручне адміністрування, знизило ризики, пов'язані з помилками конфігурацій, та підвищило загальний рівень довіри до системи з боку бізнес-користувачів. У майбутньому планується подальше зміцнення захисного контуру шляхом впровадження багатофакторної автентифікації, системи контролю доступу на основі ролей (RBAC), а також аудитів безпеки й автоматичного логування подій, що дозволить перейти до повноцінної Zero Trust моделі.

Висновки до другого розділу

У другому розділі було сформовано цілісну концепцію, архітектуру та інформаційне проєктування веб-системи для бізнес-аналітики. У процесі роботи визначено функціональні та нефункціональні вимоги до системи, описано ключові сценарії взаємодії користувачів із платформою, а також обґрунтовано вибір технологічного стеку. Основою реалізації стала клієнт-серверна архітектура, що поєднує React на фронтенді, NestJS на бекенді та окремий аналітичний модуль на Python із використанням FastAPI. Такий підхід дозволив досягти високої гнучкості, масштабованості й чіткого розмежування обов'язків між компонентами системи.

Було розроблено структуру бази даних на основі PostgreSQL, спроектовано рольову модель доступу, яка включає адміністраторів і звичайних користувачів із відповідним розмежуванням прав. Для зберігання та обробки інформації реалізовано класи для управління користувачами, авторизацією, а також модуль, що відповідає за аналітику, інтегрований через HTTP-запити з серверною частиною.

Особливу увагу приділено аналітичній частині системи: Python-модуль відповідає за обробку CSV-файлів, виконання ABC-аналізу, RFM-сегментації, аналіз повторних покупок, сезонності, регіональних вподобань тощо. Завдяки

використанню бібліотек Pandas, NumPy і Scikit-learn, досягнуто високої точності та продуктивності обчислень. Отримані результати передаються в JSON-форматі для подальшої візуалізації на клієнтській стороні, де використовуються Recharts та інші бібліотеки.

Система забезпечує захищений обмін даними завдяки впровадженню JWT-авторизації, верифікації електронної пошти через Mailjet, а також використанню HTTPS-протоколу. Інфраструктура побудована на основі Docker і Traefik, що дозволяє легко масштабувати застосунок, автоматично керувати сертифікатами безпеки й розгортати оновлення.

Порівняльний аналіз із альтернативними інструментами, такими як R і Power BI, підтвердив доцільність вибору Python як основного аналітичного модуля завдяки його гнучкості, масштабованості, підтримці машинного навчання й зручній інтеграції з бек-ендом. Таким чином, обґрунтовано ефективність запропонованої архітектури, що забезпечує стабільну та безпечну роботу з великими обсягами бізнес-даних і дозволяє приймати обґрунтовані рішення на основі аналітики.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ, АНАЛІЗ РЕЗУЛЬТАТІВ І ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОЇ СИСТЕМИ.

3.1 Процес розгортання програмного додатку

Розглянемо процес розгортання веб-застосунку з використанням технологій контейнеризації Docker та оркестрації через Docker Compose. Такий підхід забезпечує простоту управління залежностями, ізоляцію сервісів, повторюваність середовища та зручне масштабування системи. Усі компоненти програми описано в єдиному конфігураційному файлі `docker-compose.yml` версії 3.5, який об'єднує п'ять основних сервісів: `backend`, `data-db`, `analyzer`, `frontend` і `reverse-proxy`.

Сервіс `backend` відповідає за REST API, реалізовані на базі NestJS. Контейнер створюється з локального каталогу `./backend`, публікує порт 3000 і містить змінні середовища для підключення до бази даних і зовнішніх API (наприклад, Mailjet). Крім того, конфігурація включає мітки для Traefik, які використовуються для автоматичної маршрутизації та підключення HTTPS-сертифікатів. Сервіс `data-db` побудований на основі офіційного образу PostgreSQL (версія 14-alpine) і зберігає структуру бази даних, включаючи ім'я БД, користувача і пароль, задані через змінні середовища. Збереження даних забезпечується завдяки тому, що том бази даних винесений за межі контейнера.

Сервіс `analyzer` реалізований як окремий Python-додаток на базі FastAPI, Pandas та допоміжних бібліотек для обробки та аналізу даних. Він також містить змінні середовища для доступу до бази даних і відповідає за виконання обчислень, таких як RFM-аналіз, оцінка середнього чека, побудова асоціативних правил тощо. Після обробки даних результати передаються у форматі JSON до `backend`. Сервіс `frontend` є односторінковим застосунком на React для побудови графіків. Він збирається з каталогу `./frontend`, публікує порт 3000 і взаємодіє з `backend` через REST API. Для налаштування середовища додано томи та змінні, а маршрутизація також керується через Traefik.

Сервіс reverse-proxy реалізовано за допомогою Traefik (версія 2.4), який виступає точкою входу для всіх запитів. У конфігурації задано порти 80, 443 і 8080, що відповідають HTTP, HTTPS та адміністративному інтерфейсу Traefik відповідно. Проксі-сервер автоматично зчитує мітки інших контейнерів, визначає маршрути та динамічно отримує SSL-сертифікати через ACME (Let's Encrypt), зберігаючи їх у локальному томі.

Docker Compose автоматично створює віртуальну мережу, в межах якої контейнери взаємодіють за внутрішніми іменами сервісів, що спрощує налаштування. Завдяки політиці restart: unless-stopped сервіси автоматично перезапускаються після збою чи перезавантаження хосту. Розгортання системи відбувається за єдиною командою `docker compose up -d --build`, яка збирає образи (якщо вони ще не створені) та запускає всі сервіси у фоновому режимі. У разі змін в одному з компонентів немає потреби перезапускати весь стек – достатньо перестворити лише відповідний контейнер.

Такий підхід до розгортання має низку переваг: він забезпечує модульність архітектури, дозволяє швидко оновлювати окремі частини застосунку, спрощує CI/CD-процеси та гарантує однакову роботу додатку на будь-якій машині, де встановлено Docker. У підсумку, Docker Compose відіграє ключову роль у структурі застосунку, забезпечуючи керованість, стабільність і масштабованість усієї системи.

Описана архітектура має важливу перевагу – універсальність розгортання. Завдяки використанню Docker Compose, один і той самий набір конфігурацій може бути застосований як для локального середовища розробки, так і для продакшн-сервера. Локально це дозволяє швидко запускати усі необхідні сервіси без потреби в окремому налаштуванні бази даних, бекенду чи аналітики, що значно пришвидшує розробку та тестування. У продакшн-оточенні така структура дає змогу автоматизувати розгортання, легко переносити застосунок на інші сервери або хмарні платформи та забезпечує стабільну роботу системи завдяки чітко визначеним залежностям і мережевим зв'язкам між компонентами.

3.2 Функціональна структура інтерфейсу користувача

Веб-інтерфейс програмного продукту побудований за принципами зручності, доступності та логічної послідовності взаємодії. Для роботи з додатком користувачу достатньо мати пристрій із сучасним веб-браузером та стабільне підключення до Інтернету. Навігація реалізована на основі SPA (Single Page Application), що забезпечує швидке перемикання між сторінками без повного перезавантаження.

На рисунку 3.1 представлено структурну діаграму інтерфейсу користувача, яка демонструє основні маршрути переходів та взаємодію з ключовими розділами веб-додатку. Початкова точка взаємодії – головна сторінка або авторизація. Після входу користувач отримує доступ до особистого кабінету, в якому є функції редагування особистих даних та зміни пароля. Основний блок складається з доступу до різних методів аналітики, реалізованих у вигляді окремих інтерактивних графіків.

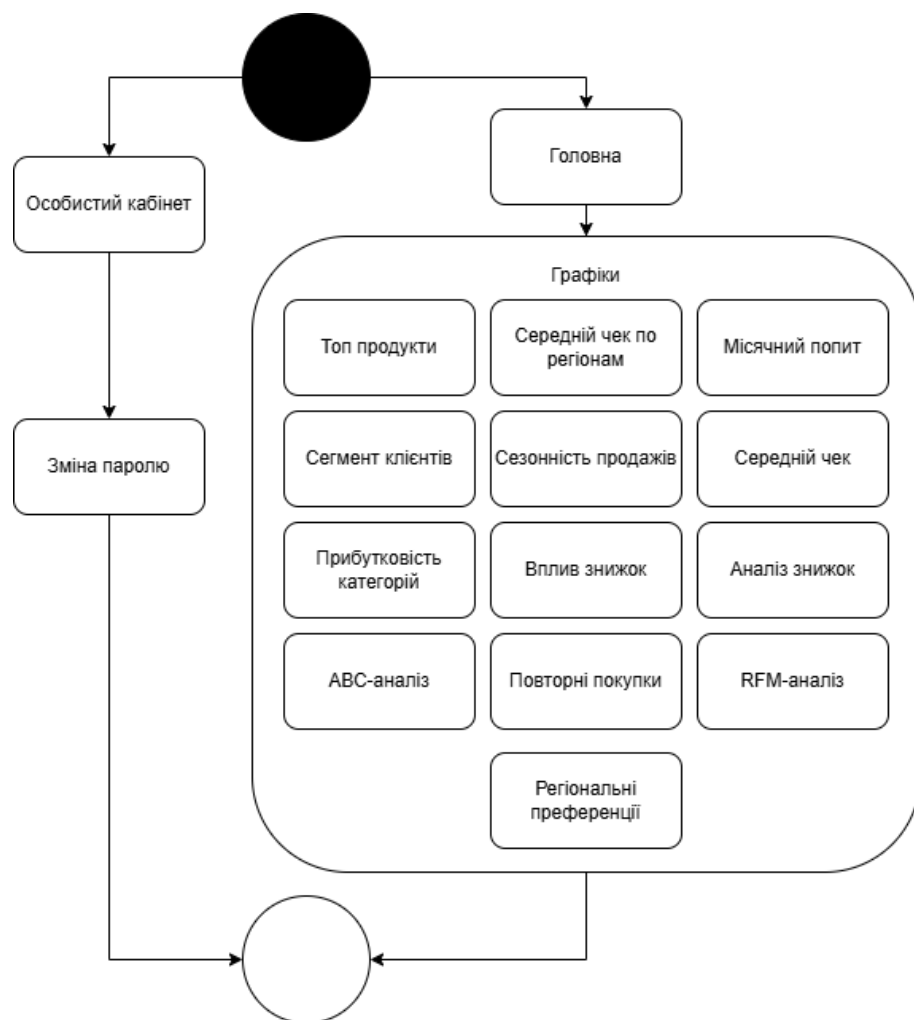


Рис. 3.1. Діаграма інтерфейсу користувача. Ця діаграма демонструє можливості взаємодії з додатком для звичайного користувача.

Загалом структура інтерфейсу та його логіка побудови відповідає принципам гнучкого UX-дизайну, що дозволяє користувачу швидко зорієнтуватися в можливостях системи й ефективно використовувати її функціонал для прийняття управлінських рішень.

Рис. 3.2 Форма для авторизації користувача.

На рис. 3.2 зображено стандартну форму входу до системи, яка дозволяє авторизованим користувачам отримати доступ до особистого кабінету та функціоналу веб-додатку. Користувач вводить свою електронну адресу та пароль, після чого натискає кнопку «Увійти». Якщо облікові дані коректні, система надає доступ відповідно до ролі користувача (адміністратор або користувач).

Рис. 3.3 Форма для запиту відновлення пароля користувача.

На рис. 3.3 зображено форму для відновлення пароля користувача. У разі, якщо користувач забув свій пароль, він може ввести адресу електронної пошти, прив'язану до облікового запису, та натиснути кнопку «Надіслати інструкції». Після цього на вказану пошту надходить лист з посиланням для відновлення доступу до системи. Такий механізм підвищує зручність користування та гарантує безпечний доступ до облікового запису. Кнопка «Повернутись до входу» дозволяє повернутися на попередню сторінку авторизації.

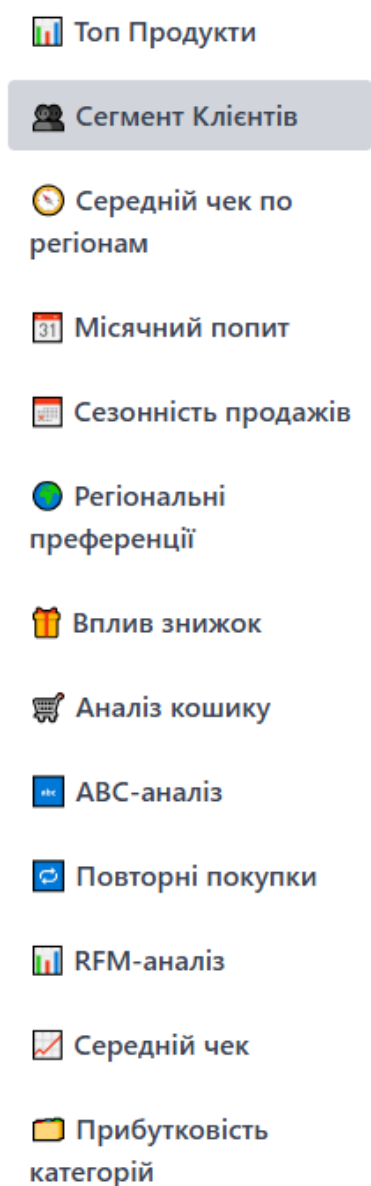


Рис. 3.4 Меню аналітики продажів. Панель для вибору графіків з аналізом даних продажів.

На рис. 3.4 представлено бокове меню аналітики продажів, яке слугує навігаційною панеллю для вибору різних інструментів візуалізації даних.

Кожен пункт відповідає окремому виду аналітичного звіту, серед яких: топові товари, клієнтські сегменти, середній чек по регіонах, місячний попит, сезонність продажів, регіональні преференції, вплив знижок, аналіз кошику, ABC-аналіз, повторні покупки, RFM-аналіз, середній чек загалом та прибутковість категорій. Інтерфейс реалізований з використанням іконок та зрозумілих підписів, що підвищує зручність у користуванні системою для аналітиків і менеджерів. Активний розділ підсвічується, що дозволяє користувачу орієнтуватися у виборі.

3.3 Інтеграція модулів обробки даних та побудови графіків

У межах реалізації веб-застосунку було здійснено інтеграцію серверного аналітичного модуля з інтерфейсом візуалізації, що забезпечує безперервний обмін даними між усіма компонентами системи. Обробка даних відбувається в окремому мікросервісі, реалізованому на Python, який виконує розрахунки для таких аналітичних методів, як RFM-аналіз, ABC-аналіз, оцінка середнього чека, аналіз впливу знижок, сегментація клієнтів тощо. Після завершення обчислень отримані результати повертаються у форматі JSON через HTTP-запити до серверного додатку на NestJS.

NestJS виступає проміжною ланкою між аналітичним ядром і клієнтською частиною. Він отримує запити від користувача, надсилає їх до Python-сервісу, який, у свою чергу, здійснює розрахунки з використанням бібліотек **Pandas** та **mlxtend**, і повертає оброблені результати. Після цього NestJS трансформує отримані дані у формат, зручний для відображення на клієнтському інтерфейсі, та надсилає їх у відповідь. Така архітектура дозволяє ізолювати аналітичну логіку від бізнес-рівня, що забезпечує гнучкість у підтримці, легкість у масштабуванні кожного з модулів та покращену безпеку даних.

Pandas – це швидкий, потужний, гнучкий і зручний у використанні інструмент з відкритим кодом для аналізу та обробки даних, створений на основі мови програмування Python. Він забезпечує ефективну роботу з табличними структурами даних у форматі DataFrame і Series, підтримує операції фільтрації, агрегації, групування, трансформації, очищення даних, а

також має засоби для обробки часових рядів, дат, текстових полів і роботи з відсутніми значеннями. Pandas дозволяє легко імпортувати та експортувати інформацію у різних форматах, зокрема CSV, Excel, SQL, JSON, що робить його надзвичайно зручним у роботі з вхідними даними для аналітики. Бібліотека також має глибоку інтеграцію з іншими науковими та візуалізаційними інструментами, зокрема NumPy, Matplotlib, Seaborn та Scikit-learn [22].

mlxtend (розширення машинного навчання) – це спеціалізована бібліотека Python, орієнтована на задачі практичного аналізу даних і прототипування моделей. Вона містить реалізації алгоритмів машинного навчання, серед яких особливо важливими є алгоритми асоціативних правил (Apriori, FP-growth), що активно застосовуються для виявлення прихованих зв'язків між товарами у корзинному аналізі. Також mlxtend включає модулі для кластеризації, класифікації, регресії, побудови обробних конвеєрів (pipeline), комбінування моделей (stacking, voting) та зручну візуалізацію результатів. Ці інструменти дозволяють швидко розробляти та перевіряти аналітичні гіпотези в рамках веб-застосунку [23].

На клієнтському рівні, побудованому з використанням фреймворку **React**, реалізовано інтерактивні компоненти візуалізації даних за допомогою бібліотеки **Recharts**. Recharts — це сучасна, легка у використанні бібліотека побудови графіків та діаграм, створена на основі D3.js і призначена спеціально для React-екосистеми. Вона підтримує декларативний підхід до побудови графіків, що дозволяє легко інтегрувати візуалізацію у компонентну структуру застосунку. Серед типів графіків, які підтримуються Recharts, — лінійні, стовпчикові, кругові діаграми, графіки розсіяння та комбіновані графіки. Бібліотека надає широкі можливості налаштування: інтерактивні підказки (tooltip), легенди, анімації, навігацію, динамічне оновлення даних, гнучке налаштування осей та масштабування. Завдяки цьому вона ефективно використовується для візуального представлення складних аналітичних результатів у зрозумілому вигляді. Компоненти приймають дані у форматі

JSON, що забезпечує повну сумісність із відповідями NestJS API, і дозволяють користувачам взаємодіяти з графіками — обирати категорії або регіони, переглядати значення по наведенню курсора, фільтрувати дані та порівнювати показники [24].

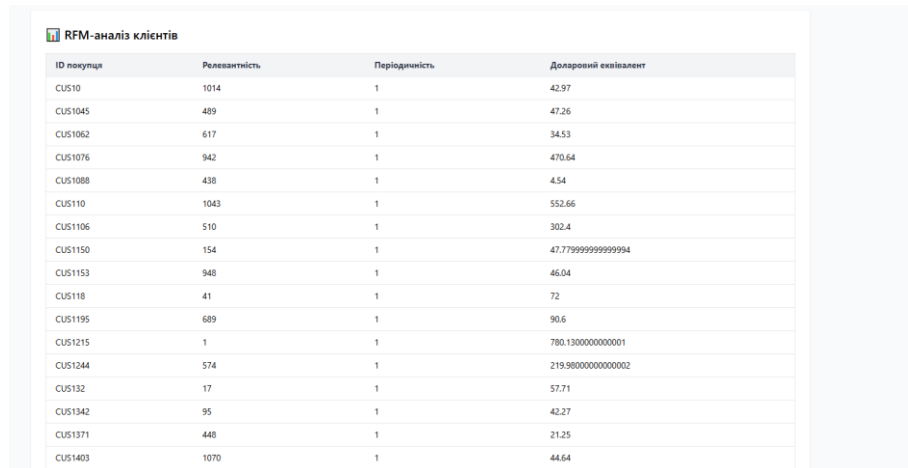
Таке чітке розділення обов’язків між модулями — аналітичним, серверним і клієнтським — дозволяє досягти високої швидкодії системи, адаптивності до змін у бізнес-логіці, можливості масштабування окремих частин застосунку та ефективної підтримки складних користувацьких сценаріїв. Уся взаємодія між компонентами реалізована відповідно до принципів REST-архітектури, що забезпечує стандартизовану, надійну і зручну для розширення комунікацію. Така архітектура є придатною як для локального використання, так і для масштабування у хмарних середовищах або на серверних інфраструктурах.

3.4 Демонстрація аналітичних сценаріїв: RFM, ABC, аналіз товарних категорій

Після успішної авторизації користувач отримує доступ до набору вбудованих методів аналітики, кожен із яких дозволяє по-різному інтерпретувати комерційну інформацію, виявляти закономірності у продажах і здійснювати сегментацію клієнтів. Застосунок підтримує інтерактивний вибір необхідного типу аналізу, зокрема RFM-аналіз (рис. 3.5) , ABC-аналіз(рис. 3.9), побудову графіків прибутковості категорій(рис. 3.10), повторних покупок(рис. 3.14), регіональних переваг, динаміки середнього чеку(рис. 3.15), топ-продуктів(рис. 3.6) тощо. Кожен метод базується на математичних та статистичних алгоритмах, реалізованих у серверній частині системи. Після вибору методу відповідний запит надсилається до аналітичного модуля, який здійснює обробку даних і повертає результат у форматі JSON.

Ці результати відображаються у вигляді діаграм або таблиць у клієнтському інтерфейсі. Такий підхід дозволяє користувачу зосередитися на прийнятті рішень, не заглиблюючись у технічні деталі аналізу. Крім того, усі

методи виконуються асинхронно, що забезпечує плавність взаємодії з інтерфейсом навіть при роботі з великими обсягами даних.



ID покупця	Релевантність	Періодичність	Доларовий еквівалент
CUS10	1014	1	42.97
CUS1045	489	1	47.26
CUS1062	617	1	34.53
CUS1076	942	1	470.64
CUS1088	438	1	4.54
CUS110	1043	1	552.66
CUS1106	510	1	302.4
CUS1150	154	1	47.779999999999994
CUS1153	948	1	46.04
CUS118	41	1	72
CUS1195	689	1	90.6
CUS1215	1	1	780.1300000000001
CUS1244	574	1	219.98000000000002
CUS132	17	1	57.71
CUS1342	95	1	42.27
CUS1371	448	1	21.25
CUS1403	1070	1	44.64

Рис. 3.5 Таблиця, що відображає вихідні дані RFM-аналізу. Відображає релевантність, періодичність та доларовий еквівалент кожного замовника.

На рис. 3.5 представлено таблицю, що відображає результати RFM-аналізу клієнтів. У таблиці наведено три ключові метрики: Релевантність (Recency) – кількість днів, що пройшли з моменту останньої покупки; Періодичність (Frequency) – кількість покупок, здійснених клієнтом за певний період; та Доларовий еквівалент (Monetary) – сумарна вартість усіх покупок клієнта. Таке представлення даних дозволяє ідентифікувати цінних клієнтів, які купують часто, нещодавно та на значні суми. Наприклад, клієнти з низьким значенням Recency, високими Frequency і Monetary є найбільш лояльними та прибутковими. Ці дані є основою для подальшої сегментації клієнтської бази та формування персоналізованих маркетингових стратегій.

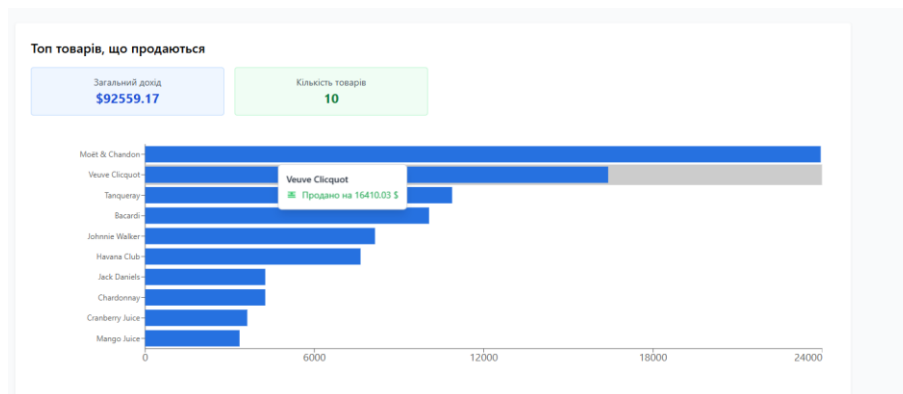


Рис. 3.6 Горизонтальна діаграма з відображенням топ 10-ти товарів, що продаються.

На рис. 3.6 зображено горизонтальну стовпчикову діаграму, яка демонструє топ-10 товарів за обсягом продажів. Графік ілюструє не лише кількість проданих одиниць кожного продукту, а й їхній вплив на загальний дохід компанії, який становить \$92559.17. Кількість урахованих товарів – 10. Кожен стовпчик відображає конкретний товар, а його довжина візуалізує обсяг продажів у доларах. Наприклад, Moët & Chandon та Veuve Clicquot є лідерами за прибутком, що свідчить про їхню популярність серед покупців. Така візуалізація дозволяє швидко і наочно оцінити асортимент з точки зору фінансової ефективності й зробити висновки щодо потенційних лідерів продажів. Отримана інформація може бути використана для планування закупівель, маркетингових акцій або оптимізації складу.

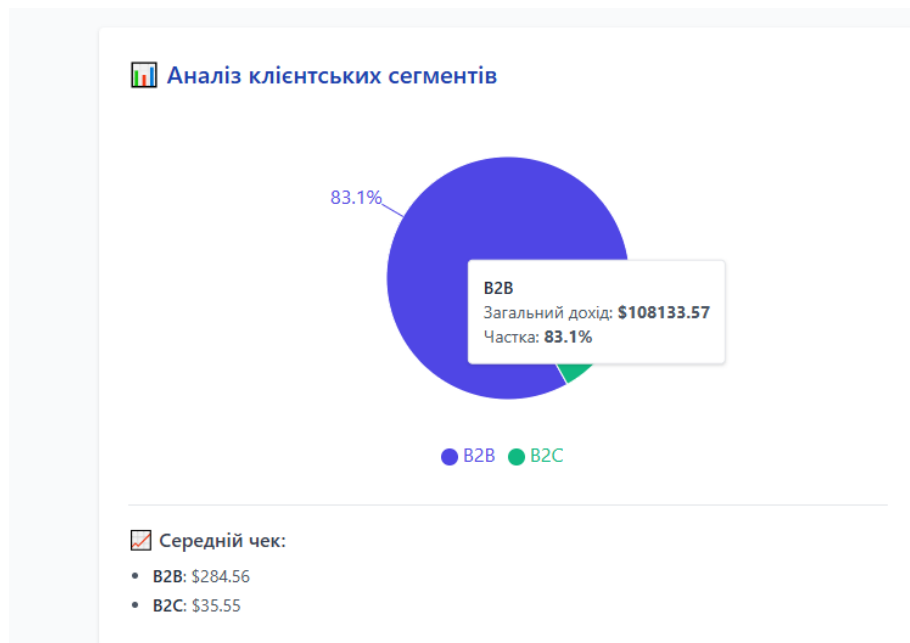


Рис. 3.7 Кругова діаграма, що відображає аналіз клієнтських сегментів з даними про загальний дохід та частку клієнтів різних типів, середній чек кожного сегменту.

Рис. 3.7 ілюструє результати аналізу клієнтських сегментів у вигляді кругової діаграми, яка демонструє розподіл доходу між двома основними типами клієнтів: B2B (business-to-business) та B2C (business-to-consumer). Діаграма дозволяє візуально оцінити частку кожного сегмента у загальному доході компанії. Згідно з аналізом, сегмент B2B є домінуючим – він формує 83.1% загального доходу, що становить \$108,133.57, у той час як частка B2C є значно меншою. Це свідчить про те, що бізнесові клієнти приносять основний прибуток і, ймовірно, здійснюють замовлення з вищою середньою вартістю. Нижче наведено дані про середній чек у кожному сегменті: для B2B він становить \$284.56, тоді як для B2C – лише \$35.55. Такий контраст підтверджує, що клієнти бізнес-сегменту мають значно вищу купівельну спроможність, що може впливати на стратегію ціноутворення, маркетингу та обслуговування. Отримана інформація є ключовою для подальшої сегментації клієнтів, налаштування таргетованих кампаній і прийняття управлінських рішень, орієнтованих на найприбутковіші сегменти ринку.

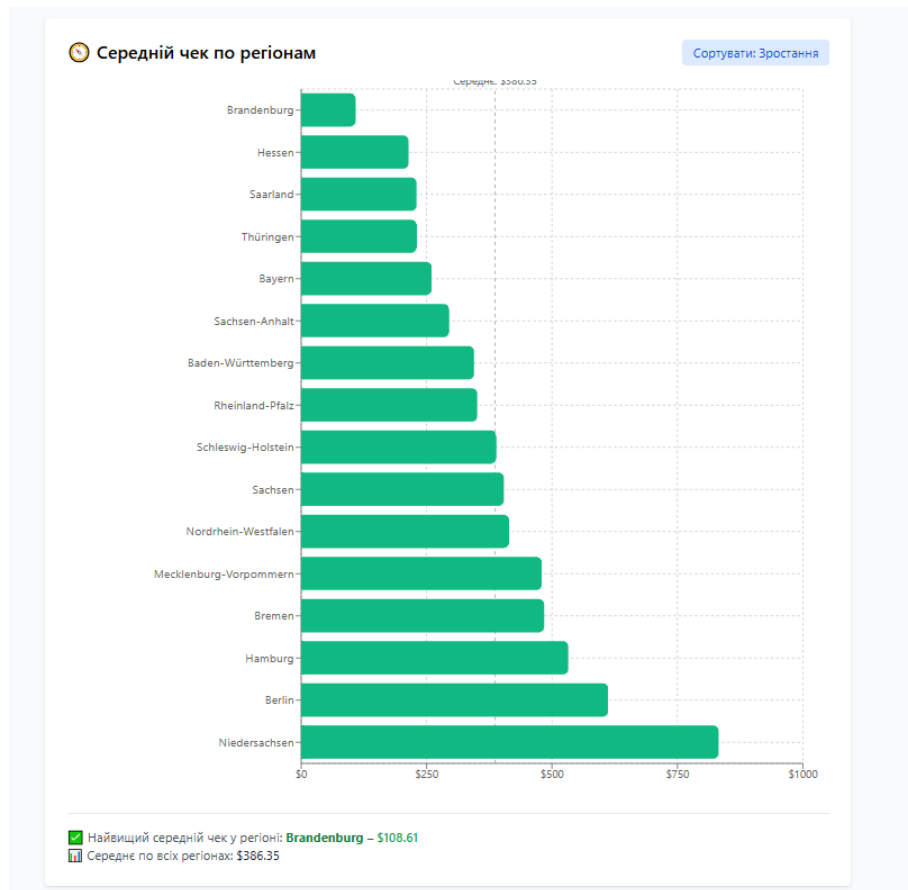


Рис. 3.8 Горизонтальна діаграма, що відображає аналіз середнього чеку по регіонам серед замовників.

Рис. 3.8 демонструє результати аналізу середнього чеку по регіонах у вигляді горизонтальної стовпчикової діаграми. Діаграма дозволяє візуально порівняти середню суму замовлення серед клієнтів з різних регіонів, що є важливим показником купівельної активності та фінансової ефективності кожного регіону. Найнижчий середній чек зафіксовано в регіоні Brandenburg – \$108.61, тоді як найвищий – у Niedersachsen, де він перевищує \$190. Середнє значення по всіх регіонах становить \$136.35. Ці дані дозволяють зробити висновки про регіональні відмінності у споживанні, що можуть бути зумовлені соціально-економічними факторами, рівнем доходів або локальними вподобаннями клієнтів. Отримані результати доцільно використовувати для оптимізації цінової політики, географічного таргетингу в маркетингових кампаніях та розподілу ресурсів між регіонами.

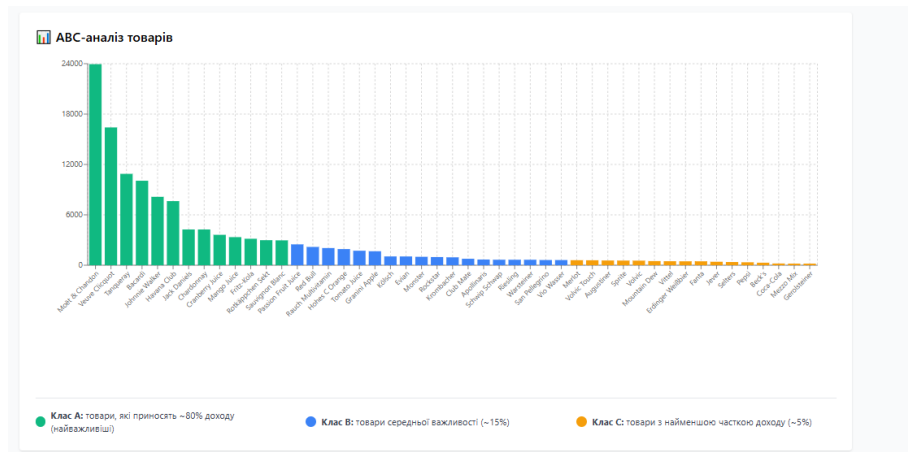


Рис. 3.9 Вертикальна діаграма, що відображає дані ABC-аналізу з описанням класів аналізу.

Рис. 3.9 демонструє результати ABC-аналізу товарів у вигляді вертикальної гістограми, де кожен стовпчик відповідає окремому продукту, а його висота відображає загальний дохід від продажу цього товару. Товари поділено на три категорії за їхнім внеском у загальну виручку: клас А включає найбільш прибуткові позиції, які формують приблизно 80% доходу при мінімальній кількості найменувань; клас В охоплює товари з помірною фінансовою значущістю, що приносять ще близько 15% доходу; до класу С належать продукти з найменшою часткою у виручці, які становлять велику кількість найменувань, але мають низьку фінансову ефективність. Такий аналіз дозволяє акцентувати увагу на ключових продуктах і приймати обґрунтовані рішення щодо оптимізації асортименту та управління запасами.

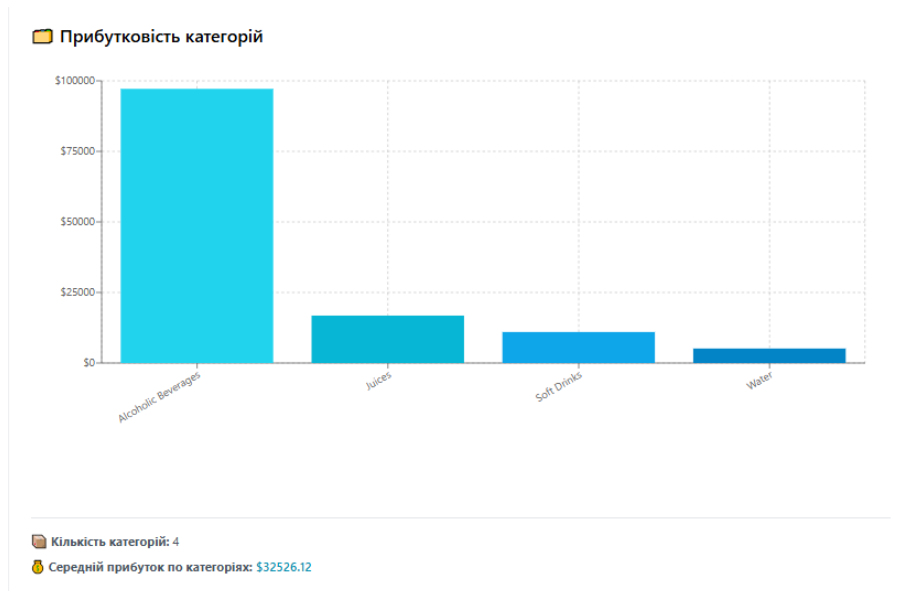


Рис. 3.10 Стовпчикова діаграма, що відображає аналіз прибутковості категорій, також відображає кількість категорій та середній прибуток серед категорій.

Рис. 3.10 ілюструє прибутковість різних категорій товарів за допомогою стовпчикової діаграми. На графіку чітко видно, що категорія "Alcoholic Beverages" домінує за рівнем доходу, значно перевищуючи інші категорії, такі як "Juices", "Milk Drinks" та "Water". Такий розподіл дозволяє швидко виявити найприбутковіші напрямки у товарному асортименті та орієнтувати маркетингові чи логістичні зусилля саме на ці сегменти. Нижче під діаграмою також зазначено загальну кількість категорій – чотири, а також середній прибуток на одну категорію, що становить \$23 526.12. Це забезпечує додатковий аналітичний контекст для оцінки ефективності кожної групи товарів.

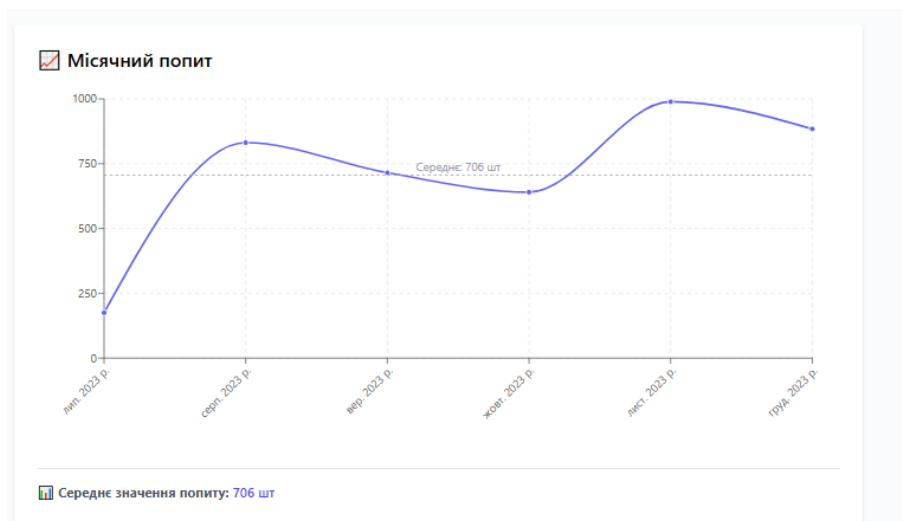


Рис. 3.11 Діаграма, що відображає місячний попит на товари по місяцям, також відображає середнє значення попиту по місяцям.

На рис. 3.11 представлено лінійну діаграму, яка демонструє динаміку місячного попиту на товари протягом року. Графік дозволяє візуально відстежити коливання кількості замовлень у різні періоди, виявити пікові місяці з найвищим попитом, а також спадні тренди. Особливу увагу привертає відмітка середнього значення, що становить 706 одиниць, яка проходить горизонтальною лінією через графік і дозволяє зіставити окремі місяці з усередненим попитом. Такий тип візуалізації є зручним для виявлення сезонності, що може бути корисним при плануванні закупівель, розподілі ресурсів або формуванні маркетингових стратегій.

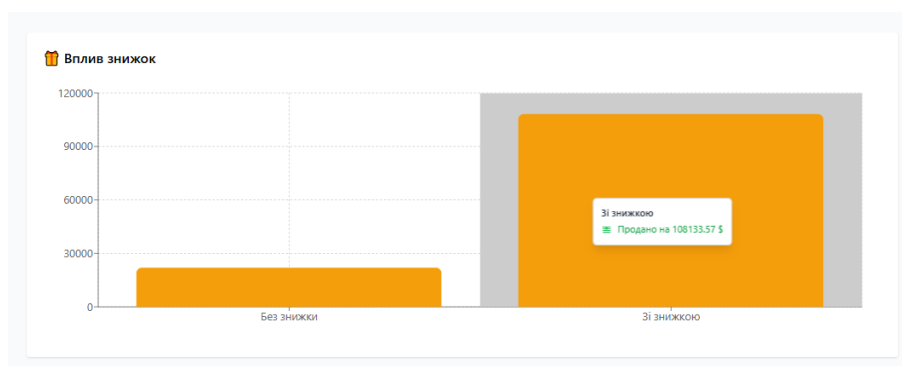


Рис. 3.12 Стовпчикова діаграма, що відображає співвідношення виручених коштів за товари зі знижкою та без.

На рис. 3.12 зображено стовпчикову діаграму, яка ілюструє вплив знижок на виручку. Діаграма порівнює загальні обсяги продажів товарів, реалізованих

зі знижкою та без неї. Видно, що виручка від продажів зі знижками суттєво перевищує виручку без них, що свідчить про ефективність застосування знижкової політики для стимулювання попиту. Такий аналіз дозволяє підприємству оцінити доцільність маркетингових заходів і знижкових кампаній, а також оптимізувати ціноутворення з урахуванням реальної поведінки споживачів.



Рис. 3.13 Діаграма, що відображає аналіз тренду середнього чека з середнім значенням періодів, що аналізуються.

На рис. 3.13 представлено лінійну діаграму, яка ілюструє зміну середнього чека за різні періоди часу. Графік дає змогу відстежити коливання середнього обсягу витрат клієнтів упродовж аналізованого періоду, а також виявити пікові значення та спади. У підписі зазначено середнє значення чека за весь період (\$391.81), що дає змогу оцінити загальну тенденцію. Такий аналіз допомагає виявляти сезонні або маркетингові впливи на поведінку покупців і коригувати стратегії продажів.

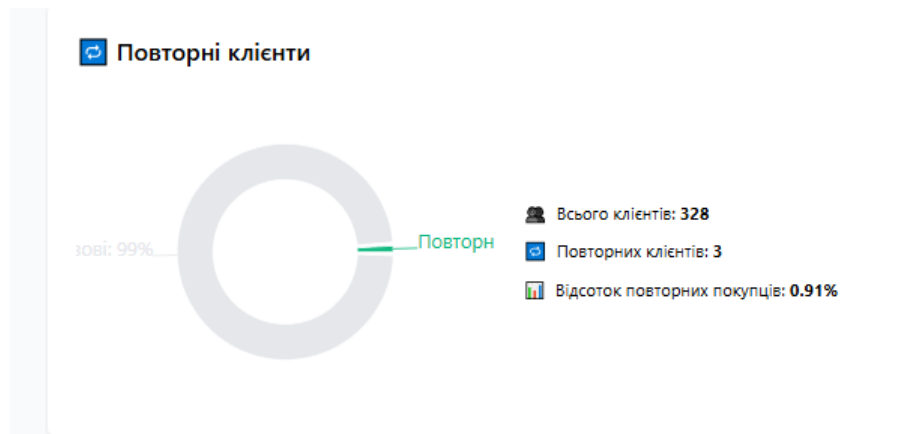


Рис. 3.14 Кругова діаграма, що відображає аналіз співвідношення клієнтів по повторному замовленню та їх кількість.

На рис. 3.14 зображено кругову діаграму, яка демонструє частку повторних клієнтів серед загальної кількості покупців. Візуалізація показує, що з 328 клієнтів лише 3 здійснили повторну покупку, що становить 0.91% від загального обсягу. Такий результат свідчить про низький рівень лояльності клієнтів або про відсутність належної програми утримання. Цей показник є важливим для подальшого аналізу клієнтської поведінки та розробки заходів з підвищення повторних продажів.

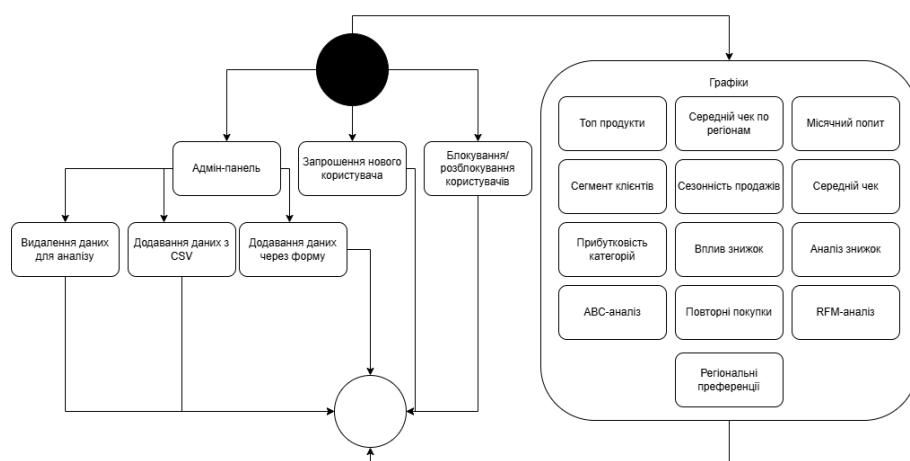


Рис. 3.15 Діаграма інтерфейсу адміністратора. Ця діаграма демонструє можливості взаємодії з додатком для адміністратора.

У рамках реалізованого веб-додатку особливу увагу приділено функціоналу адміністратора, оскільки саме він відповідає за контроль цілісності даних та управління користувачами. Після авторизації в системі адміністратор отримує доступ до адміністративної панелі, з якої можливо ініціювати процеси керування даними для аналітики. Зокрема, підтримується можливість додавання даних у систему у двох форматах – через завантаження CSV-файлу або шляхом ручного введення інформації через інтерактивну форму. Крім того, адміністратор має змогу переглядати та видаляти застарілі або некоректні записи, що дозволяє підтримувати актуальність аналітичної бази.

Окремим напрямом адміністративної діяльності є керування доступом користувачів до системи. Через адмін-панель реалізовано функції надсилання запрошень новим користувачам, а також можливість блокування або розблокування вже зареєстрованих акаунтів. Це дозволяє забезпечити високий рівень безпеки системи та ефективно розподіляти ролі між учасниками. Усі дії адміністратора відображаються в інтерфейсі у вигляді простих послідовних кроків, що демонструється на відповідній діаграмі(рис. 3.15).

Розглянемо одну з ключових можливостей адміністратора, додавання нових користувачів до системи. Для цього адміністратор має можливість вказати електронну адресу нового користувача (рис. 3.16), після чого система автоматично генерує запрошення та надсилає його через зовнішній поштовий сервіс (у нашому випадку – Mailjet). Це запрошення містить унікальне посилання (рис. 3.17), яке дозволяє майбутньому користувачеві створити особистий обліковий запис і увійти до системи (рис. 3.18).



 **Запросити користувача**

Email

Скасувати

Запросити

Рис. 3.16 Модальне вікно адміністратора для запрошення нового користувача.

На рис. 3.16 представлено модальне вікно, призначене для адміністратора системи, яке використовується з метою запрошення нового користувача до платформи. Вікно містить одне поле введення електронної пошти, кнопки «Скасувати» та «Запросити». Після введення адреси електронної пошти та натискання кнопки «Запросити» система надсилає листа-запрошення, що містить посилання для реєстрації облікового запису.

 Підтвердження запрошення

Ім'я

Прізвище

Пароль

Підтвердити

Рис. 3.17. Сторінка для завершення реєстрації користувача.

На рис. 3.17 зображено сторінку підтвердження запрошення, яка є завершальним етапом реєстрації нового користувача в системі. Після переходу за посиланням, що надійшло на електронну пошту у вигляді запрошення, користувач потрапляє на цю форму. Тут необхідно ввести ім'я, прізвище та створити пароль для свого облікового запису. Після заповнення форми й натискання кнопки «Підтвердити» дані зберігаються в системі, і користувач отримує доступ до особистого кабінету та функцій платформи згідно з наданими правами доступу.

Окрему увагу в адміністративному функціоналі приділено можливості керування активністю зареєстрованих користувачів. Інтерфейс адміністратора надає доступ до таблиці всіх створених акаунтів, у якій відображено основну інформацію: ім'я, електронну пошту, роль у системі, дату створення облікового запису та поточний статус. За потреби адміністратор має змогу деактивувати користувача, натиснувши відповідну кнопку. Ця операція миттєво змінює стан облікового запису, унеможливаючи доступ до захищених розділів системи.

Деактивація не видаляє акаунт остаточно, що дозволяє зберігати дані користувача для подальшого аудиту або повторної активації. Якщо обліковий запис має бути відновлений, адміністратор так само може активувати його знову через той самий інтерфейс. Такий підхід дає змогу гнучко керувати правами доступу без втрати історії дій та пов'язаних аналітичних результатів.

Адмін-панель

Користувачі

Дані

Ім'я	Email	Роль	Створено	Статус
Михайло Лебіга	lebiga@example.com	Користувач	24.06.2025, 23:51	Деактивувати
Петро Петренко	petrenko@example.com	Користувач	24.06.2025, 23:51	Активувати
Іван Іваненко	ivanenko@example.com	Користувач	24.06.2025, 23:51	Деактивувати

Рис. 3.19. Панель для керування користувачами з відображенням даних користувачів та статусу активності.

На рисунку 3.19 показано приклад реалізації інтерфейсу деактивації, де для кожного користувача передбачено окрему дію керування статусом, що

забезпечує точкове та контрольоване адміністрування. Це особливо важливо для середовищ із підвищеними вимогами до безпеки та конфіденційності, де навіть тимчасове обмеження доступу може мати критичне значення для збереження даних.

Важливою складовою функціональності веб-додатку є механізм завантаження вхідних даних, необхідних для проведення аналітики. Інтерфейс завантаження реалізовано у вигляді зрозумілого візуального блоку, де користувач може обрати зручний спосіб додавання інформації – шляхом завантаження CSV-файлу або ручного введення через інтерактивну форму.

При виборі режиму «Завантажити CSV» користувач натискає відповідну кнопку, після чого система відкриває доступ до локальної файлової системи для вибору необхідного документа. Після підтвердження вибору та натискання кнопки «Завантажити файл» файл передається на сервер, де проходить валідацію щодо структури та коректності полів. На цьому етапі застосовується перевірка на відповідність формату, наприклад, наявність обов'язкових стовпців, типів даних та правильності дати. Цей етап реалізовано на стороні клієнта за допомогою бібліотек React Hook Form та Yup, а також на сервері для додаткової перевірки безпеки.

У разі, якщо дані пройшли перевірку, вони зберігаються в базі даних для подальшої обробки аналітичним ядром. У випадку помилки система повідомляє користувача про характер проблеми, що дозволяє оперативно її виправити.

Такий підхід дозволяє значно зменшити бар'єр для роботи з аналітичним інструментом, забезпечити зручність для нетехнічних користувачів та підвищити загальну ефективність обробки вхідних даних. На рисунку 3.20 представлено фрагмент інтерфейсу завантаження даних, де відображено всі елементи, необхідні для ініціації та підтвердження цієї дії.

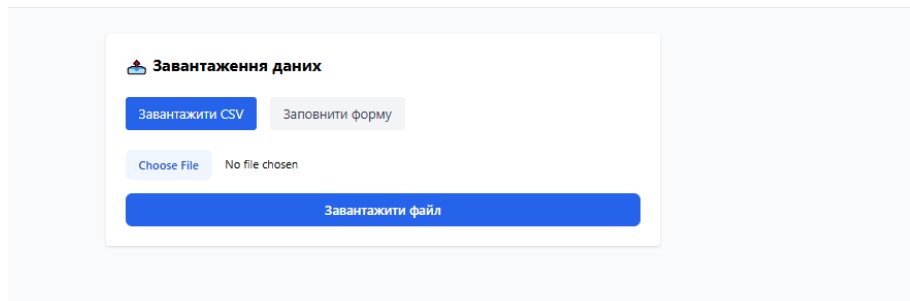


Рис. 3.20. Панель для завантаження даних у форматі CSV, потрібного формату(Додаток Б).

Крім можливості імпорту масиву даних у форматі CSV, система також надає функціонал для додавання окремих записів вручну. Такий підхід особливо зручний у випадках, коли користувачу необхідно оперативно додати одиничний продаж або оновити інформацію, не редагуючи основний файл із даними.

Для цього передбачена інтерактивна вебформа, яка включає всі ключові поля для заповнення – ідентифікатор замовлення, ідентифікатор клієнта, тип клієнта, найменування товару, категорію, ціну за одиницю, кількість, розмір знижки, загальну суму, регіон продажу та дату здійснення транзакції. Завдяки використанню сучасного підходу до обробки форм за допомогою бібліотек React Hook Form та Yup, забезпечено зручність взаємодії, а також миттєву перевірку коректності введених даних.

Після заповнення форми користувач натискає кнопку «Надіслати форму», після чого система обробляє запит та додає запис до бази даних. Вся інформація передається через захищене API-з'єднання та зберігається в таблиці sales, що використовується як джерело для подальшого аналітичного опрацювання.

Цей механізм дозволяє працювати з даними навіть за відсутності підготовленого CSV-файлу, що значно підвищує гнучкість системи в умовах реального бізнес-середовища. Такий підхід сприяє підвищенню точності введення, оскільки користувач може перевірити кожне поле до підтвердження. На рисунку 3.21 зображено інтерфейс ручного введення даних у системі.

Завантаження даних

Завантажити CSV

Заповнити форму

Завантаження даних

Order ID

Customer ID

Тип клієнта

Товар

Категорія

Ціна за одиницю

0

Кількість

0

Знижка

0

Загальна сума

0

Регіон

Дата

mm/dd/yyyy

Надіслати форму

Рис. 3.21. Форма для додавання окремих даних вручну

На рис. 3.21 представлено інтерфейс форми ручного додавання даних у систему, що використовується в модулі завантаження даних. Форма дозволяє адміністраторам або аналітикам ввести окремий запис замовлення безпосередньо через веб-інтерфейс. Вона містить поля для введення ідентифікаторів замовлення (Order ID) та клієнта (Customer ID), типу клієнта (наприклад, B2B або B2C), назви товару, категорії, ціни за одиницю, кількості, знижки, загальної суми, регіону та дати продажу. Усі числові поля мають початкове значення 0, а поле дати підтримує вибір через інтерактивний календар. Над формою користувач має змогу обрати між ручним заповненням або масовим завантаженням через CSV-файл, що робить систему гнучкою у використанні. Після заповнення всієї інформації, користувач натискає кнопку «Надіслати форму» для збереження даних у базу. Такий підхід дозволяє легко

вносити виправлення або додаткові дані без необхідності обробки окремих файлів.

Для перевірки працездатності та надійності веб-застосунку було проведено комплексне тестування, яке охоплює функціональний, навантажувальний, UX- та інтерфейсний рівні перевірки.

Функціональне тестування проводилося з метою переконатися, що всі ключові можливості системи реалізовані відповідно до вимог проєктної специфікації. Було протестовано механізми реєстрації та авторизації користувачів, імпорту CSV-файлів з комерційними даними, запуск аналітичних алгоритмів (зокрема, RFM-аналізу, ABC-аналізу, розрахунку середнього чека, оцінки ефективності знижок), а також побудову візуалізацій. У процесі тестування враховано як позитивні, так і негативні сценарії використання: наприклад, спроба авторизації з хибними обліковими даними, завантаження некоректно відформатованого CSV-файлу чи виконання запиту без автентифікаційного токена. Усі критичні шляхи проходження логіки були успішно перевірені, що засвідчило стабільність серверної частини та правильність взаємодії з аналітичним модулем.

Навантажувальне тестування дозволило оцінити поведінку системи в умовах обробки великих обсягів вхідних даних. Було змодельовано сценарії, у яких відбувається одночасний імпорту файлів з десятками тисяч записів та паралельне виконання аналітичних запитів. Система показала задовільну продуктивність без критичних втрат у швидкодії чи споживанні ресурсів. Час відповіді API залишався в межах допустимих значень (менше 700 мс у більшості випадків), а генерація візуалізацій на стороні клієнта відбувалася без відчутних затримок.

UX-тестування було орієнтоване на оцінку зручності та інтуїтивності інтерфейсу для кінцевих користувачів. Було залучено групу тестувальників, яким запропоновано виконати типові сценарії використання: імпорту даних, запуск аналізу, перегляд результатів, фільтрація та взаємодія з графіками. Результати показали, що користувачі легко орієнтуються у системі, швидко

знаходять необхідні функції та не потребують додаткової інструкції для базових дій. Було також відзначено логічну структуру навігації та передбачувану поведінку інтерфейсних компонентів.

Особливу увагу було приділено тестуванню **UI-компонентів** на предмет коректного рендерингу, адаптивності та інтерактивності. Візуальні блоки, зокрема таблиці, графіки та діаграми, перевірялися у браузерях Chrome, Firefox і Safari, а також на різних розширеннях екранів. Тестування охоплювало перевірку правильності масштабування графіків, доступності функцій наведення та фільтрації, точності підказок (tooltip) і реакції на взаємодію з елементами керування (випадаючі списки, перемикачі категорій тощо). Компоненти, побудовані на базі Recharts, продемонстрували стабільну поведінку та точне відображення аналітичних даних.

Загалом результати тестування підтвердили готовність системи до використання в умовах реального бізнес-середовища, її стабільність, масштабованість і зручність для кінцевого користувача. Досягнута якість функціональності та інтерфейсу свідчить про ефективну реалізацію як технічної, так і UX-частини веб-додатку.

Висновки до третього розділу

У третьому розділі було реалізовано прикладне впровадження архітектурних рішень, описаних у попередніх частинах проєкту. Було створено функціональну версію веб-застосунку, що дозволяє здійснювати повний цикл обробки бізнес-даних – від завантаження CSV-файлів до побудови аналітичних візуалізацій. Упроваджено гнучкий механізм взаємодії з користувачем: розмежовано доступ до функціоналу залежно від ролі, реалізовано адміністративну панель та забезпечено безпечну автентифікацію й управління обліковими записами.

Здійснено інтеграцію між клієнтською частиною, сервером на NestJS та аналітичним Python-модулем через API, що забезпечує ефективну передачу й обробку даних. Окрему увагу приділено інтерфейсу користувача, який розроблено з урахуванням принципів зручності, доступності та логічної

структури навігації. Усі компоненти взаємодіють у єдиній екосистемі, що дозволяє користувачам швидко отримувати цінну бізнес-аналітику через інтерактивні графіки та діаграми.

У результаті реалізовано стабільний, функціональний та масштабований веб-застосунок, готовий до подальшого тестування, впровадження і реального використання у сфері підтримки бізнес-рішень.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було спроектовано та реалізовано веб-застосунок для аналізу та візуалізації великих обсягів бізнес-даних, що призначений для підтримки прийняття обґрунтованих управлінських рішень. Проведено детальний аналіз предметної області, визначено актуальні методи аналітики, а також здійснено огляд сучасних інструментів і технологій, що дозволило сформулювати чіткі функціональні та нефункціональні вимоги до майбутньої системи.

У ході реалізації проєкту розроблено масштабовану архітектуру клієнт-серверного типу, що поєднує фронт-енд на базі React, бек-енд на NestJS та окремий аналітичний модуль, створений за допомогою Python і FastAPI. Для зберігання структурованих даних використано реляційну базу даних PostgreSQL, а доступ до інформації реалізовано через REST API з автентифікацією на основі JWT.

Особливу увагу приділено реалізації модулів аналітики, що включають ABC-аналіз, RFM-сегментацію, аналіз повторних покупок, впливу знижок, сезонності продажів, регіональних вподобань клієнтів та інших ключових бізнес-показників. Завдяки використанню бібліотек pandas, NumPy та scikit-learn вдалося досягти високої точності обчислень і гнучкості у впровадженні нових алгоритмів.

Також було розроблено зручний користувацький інтерфейс з можливістю завантаження даних як через CSV-файл, так і вручну через форму. Інтерфейс адміністратора дозволяє керувати користувачами, активувати чи деактивувати облікові записи, а також переглядати та редагувати інформацію в системі без необхідності доступу до коду.

У розділі, присвяченому інтерфейсу, представлено повну логіку взаємодії користувача з додатком, включаючи етапи авторизації, реєстрації, зміни пароля, навігації між розділами системи та використання аналітичних панелей.

Інтерфейс протестовано вручну, перевірено всі ключові сценарії роботи, що підтвердило надійність і стабільність функціонування веб-застосунку.

У результаті реалізовано повноцінну інформаційно-аналітичну систему, яка об'єднує зручний інтерфейс, ефективний механізм обробки великих даних та модульну архітектуру, придатну для подальшого розширення. Отримане рішення відповідає сучасним вимогам до аналітичних платформ і демонструє приклад ефективного застосування web-технологій у галузі бізнес-аналітики.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. NestJS [Електронний ресурс] - Режим доступу до ресурсу: <https://nestjs.com>
2. NestJS first steps [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.nestjs.com/first-steps>
3. NestJS modules [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.nestjs.com/modules>
4. NestJS middleware [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.nestjs.com/middleware>
5. TypeScript [Електронний ресурс] - Режим доступу до ресурсу: <https://www.typescriptlang.org/>
6. Docker compose [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.docker.com/compose/>
7. Traefik [Електронний ресурс] - Режим доступу до ресурсу: <https://doc.traefik.io/traefik/>
8. Class validator [Електронний ресурс] - Режим доступу до ресурсу: <https://github.com/typestack/class-validator>
9. Typeorm [Електронний ресурс] - Режим доступу до ресурсу: <https://orkhan.gitbook.io/typeorm/docs/find-options>
10. Ubuntu docs [Електронний ресурс] - Режим доступу до ресурсу: <https://ubuntu.com>
11. Mailjet Docs [Електронний ресурс] - Режим доступу до ресурсу: <https://dev.mailjet.com>
12. ReactJS [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev>
13. Zustand [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.pmnd.rs/zustand>
14. React Hook Form [Електронний ресурс] – Режим доступу до ресурсу: <https://react-hook-form.com>
15. Tailwind CSS [Електронний ресурс] – Режим доступу до ресурсу: <https://tailwindcss.com>

16. React Hot Toast [Електронний ресурс] – Режим доступу до ресурсу: <https://react-hot-toast.com>
17. Recharts [Електронний ресурс] – Режим доступу до ресурсу: <https://recharts.org>
18. Vite [Електронний ресурс] – Режим доступу до ресурсу: <https://vitejs.dev>
19. Axios [Електронний ресурс] – Режим доступу до ресурсу: <https://axios-http.com>
20. Yup [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/jquense/yup>
21. PostgreSQL - [Електронний ресурс] - Режим доступу до ресурсу: <https://www.postgresql.org/>
22. Pandas - [Електронний ресурс] - Режим доступу до ресурсу: <https://pandas.pydata.org/docs/>
23. mlxtend - [Електронний ресурс] - Режим доступу до ресурсу: <https://rasbt.github.io/mlxtend/>
24. mlxtend - [Електронний ресурс] - Режим доступу до ресурсу: <https://recharts.org/en-US/>
25. Banks A., Porcello E. Learning React. 3rd ed. – O'Reilly Media, 2022. – 464 p.
26. McKinney W. Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter. 2nd ed. – O'Reilly Media, 2018. – 550 p.
27. Wickham H., Grolemund G. R for Data Science: Import, Tidy, Transform, Visualize, and Model Data. – O'Reilly Media, 2016. – 522 p.
28. Monczka R., Handfield R., Giunipero L., Patterson J. Purchasing and Supply Chain Management. 7th ed. – Cengage Learning, 2020. – 880 p.
29. Camm J., Cochran J., Fry M. та ін. Business Analytics. 4th ed. – Cengage Learning, 2020. – 672 p.
30. Moroney L. Artificial Intelligence and Machine Learning for Business. – O'Reilly Media, 2020. – 320 p.
31. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. – O'Reilly Media, 2018. – 304 p.

32. Shaw Z.A. Learn Python 3 the Hard Way. – Addison-Wesley, 2017. – 320 p.
33. Nixon R. Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5. 5th ed. – O'Reilly Media, 2018. – 832 p.
34. Freeman E., Robson E. Head First Design Patterns. 2nd ed. – O'Reilly Media, 2021. – 694 p.
35. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation. – University of California, 2000.
36. Sadalage P. J., Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. – Addison-Wesley, 2012. – 192 p.
37. Fowler M. Patterns of Enterprise Application Architecture. – Addison-Wesley, 2003. – 560 p.
38. Nixon R. Learning GraphQL. – O'Reilly Media, 2019. – 336 p.
39. Burns B., Grant B., Oppenheimer D. та ін. Kubernetes: Up and Running. 3rd ed. – O'Reilly Media, 2022. – 390 p.
40. George S., Dinu G. Learning Docker. – Packt Publishing, 2020. – 360 p.
41. Holovaty A., Kaplan-Moss J. The Definitive Guide to Django: Web Development Done Right. 2nd ed. – Apress, 2009. – 447 p.
42. Deitel P., Deitel H. Internet & World Wide Web: How to Program. 5th ed. – Pearson, 2011. – 1024 p.
43. Spinellis D. Code Quality: The Open Source Perspective. – Addison-Wesley, 2006. – 368 p.

ДОДАТКИ

Лістинг файлу authorization.service.ts

```
import {
  Injectable,
  UnauthorizedException,
  Logger,
  ConflictException,
  InternalServerErrorException,
  NotFoundException,
  ForbiddenException,
} from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { UserService } from '../user/user.service';
import { SignUpDTO } from './authorization.dto';
import { ConfigService } from '@nestjs/config';
import { MailService } from '../mailer-service/mailService';

@Injectable()
export class AuthorizationService {
  private readonly logger = new Logger(AuthorizationService.name);
  constructor(
    private userService: UserService,
    private jwtService: JwtService,
    private configService: ConfigService,
    private readonly mailServiceProvider: MailService,
  ) {}

  async signIn(email: string, password: string) {
    const user = await this.userService.findByEmail(email);

    if (!user) {
      throw new NotFoundException();
    }
    if (!user.verifyPassword(password)) {
      throw new UnauthorizedException();
    }
    if (user.isDeactivated) {
      throw new ForbiddenException();
    }
    const payload = { userId: user.id, isAdmin: user.isAdmin };
    return {
      access_token: await this.jwtService.signAsync(payload),
    };
  }

  async sendInvitationEmail(email: string) {
    const token = this.encodeToken({ email }, 3);
    const appConfig = this.configService.get('app');
```

```

const url = `${appConfig.url}/confirm-invitation?token=${token}`;

const mailResponse = await this.mailServiceProvider.sendInvitationEmail(
  email,
  url,
);

if (!mailResponse) {
  this.logger.error("Mail Service wasn't able to sent mail");
  throw new InternalServerErrorException();
}

return true;
}

async registerUser(userData: SignUpDTO) {
  try {
    const decodedToken = this.decodeToken<{ email: string }>(userData.token);
    return this.userService.createUser({
      email: decodedToken.email,
      password: userData.password,
      firstName: userData.firstName,
      lastName: userData.lastName,
    });
  } catch (e) {
    this.logger.error(e);
    throw new ConflictException();
  }
}

async sendPasswordRecoveryEmail(email: string) {
  const token = this.encodeToken(email);

  const appConfig = this.configService.get('app');
  const url = `${appConfig.url}/${appConfig.changePasswordRoute}?token=${token}`;

  const mailResponse = await this.mailServiceProvider.sendInvitationEmail(
    email,
    url,
  );

  if (!mailResponse) {
    this.logger.error("Mail Service wasn't able to sent mail");
    throw new InternalServerErrorException();
  }

  return true;
}

```

```

async changePassword(token: string, newPassword: string) {

    try {
        const email = this.decodeToken<string>(token);
        const user = await this.userService.findByEmail(email);
        await this.userService.changePassword(user.id, newPassword);
        return true;
    } catch (e) {
        this.logger.error(e);
        throw new ConflictException();
    }
}

encodeToken<T>(data: T, expiresIn = 1 /* days */) {
    return this.jwtService.sign({
        data,
        iat: Math.floor(Date.now() / 1000),
        exp: Math.floor(Date.now() / 1000) + 60 * 60 * 24 * expiresIn,
    });
}

decodeToken<T>(token: string) {
    return this.jwtService.verify(token).data as T;
}
}

```

Лістинг файлу main.py

```

from fastapi import FastAPI
from analysis import (
    top_products, seasonality, segments, region, discounts,
    profitability, repeat_customers, rfm, trends, basket, abc, cancellations
)

app = FastAPI()

@app.get("/top-products")
def get_top_products():
    return top_products.analyze()

@app.get("/seasonality")
def get_seasonality():
    return seasonality.analyze()

@app.get("/segment-analysis")
def get_segment_analysis():
    return segments.analyze()

@app.get("/region-preferences")
def get_region_preferences():
    return region.analyze()

@app.get("/discount-impact")
def get_discount_analysis():
    return discounts.analyze()

```

```

@app.get("/category-profitability")
def get_category_profitability():
    return profitability.analyze()

@app.get("/region-average-check")
def get_region_avg_check():
    return region.avg_check()

@app.get("/repeat-customers")
def get_repeat_customers():
    return repeat_customers.analyze()

@app.get("/rfm-analysis")
def get_rfm_analysis():
    return rfm.analyze()

@app.get("/monthly-demand-trend")
def get_monthly_demand_trend():
    return trends.monthly_demand()

@app.get("/average-check-trend")
def get_avg_check_trend():
    return trends.avg_check()

@app.get("/basket-analysis")
def get_basket_analysis():
    return basket.analyze()

@app.get("/abc-analysis")
def get_abc_analysis():
    return abc.analyze()

```

Лістинг файлу authorization.controller.ts

```

import {
    Body,
    Controller,
    Get,
    Post,
    Query,
    ConflictException,
    NotFoundException,
    UseGuards,
} from '@nestjs/common';
import { ApiBearerAuth, ApiOkResponse, ApiTags } from '@nestjs/swagger';
import { AuthorizationService } from '../authorization.service';
import {
    SignUpDTO,
    SignInDTO,
    PasswordRecoveryRequestDTO,
    ChangePasswordDTO,
    SendInvitationDTO,
} from '../authorization.dto';
import { UserService } from '../user/user.service';
import { AdminGuard } from '../admin.guard';

@Controller('auth')
@ApiTags('Authorization')
export class AuthorizationController {
    constructor(
        private readonly authService: AuthorizationService,
        private readonly userService: UserService,
    ) {}

```

```

@UseGuards(AdminGuard)
@ApiBearerAuth()
@ApiOkResponse()
@Post('sign-up')
async inviteUser(@Body() sendInvitation: SendInvitationDTO) {
  const userExists = await this.userService.userExists(sendInvitation.email);
  if (userExists) {
    throw new ConflictException('Email already exists, please log in');
  }

  return this.authService.sendInvitationEmail(sendInvitation.email);
}

@ApiOkResponse()
@Get('confirm-sign-up')
async confirmSignUp(@Query() signUpData: SignUpDTO) {
  const createdUser = await this.authService.registerUser(signUpData);

  return createdUser;
}

@ApiOkResponse()
@Post('sign-in')
signIn(@Body() { email, password }: SignInDTO) {
  return this.authService.signIn(email, password);
}

@ApiOkResponse()
@Post('request-password-recovery')
async requestPasswordRecovery(@Body() { email }: PasswordRecoveryRequestDTO) {
  const user = await this.userService.findByEmail(email);
  if (!user) {
    throw new NotFoundException('User not found');
  }
  return this.authService.sendPasswordRecoveryEmail(email);
}

@ApiOkResponse()
@Post('change-password')
changePassword(@Body() { token, password }: ChangePasswordDTO) {
  return this.authService.changePassword(token, password);
}
}

```

Лістинг файлу user.controller.ts

```

import {
  Body,
  Controller,
  Get,
  Param,
  Post,
  Request,
  UseGuards,
} from '@nestjs/common';
import { UserService } from '../user.service';
import {
  ApiBearerAuth,
  ApiOkResponse,
  ApiParam,
  ApiTags,
} from '@nestjs/swagger';
import { AuthGuard } from '../authorization/authorization.guard';
import { ContactUsDTO, MeDTO, UpdateUserDTO } from '../user.dto';

```

```

import { AdminGuard } from '../authorization/admin.guard';

@Controller('user')
@ApiTags('User')
export class UserController {
  constructor(private userService: UserService) {}

  @UseGuards(AuthGuard)
  @ApiBearerAuth()
  @ApiOperation({ type: MeDTO })
  @Get('me')
  async me(@Request() { user: { userId } }): Promise<MeDTO> {
    const { firstName, lastName, email, createdAt, isAdmin, isDeactivated } =
      await this.userService.findById(userId);

    return {
      firstName,
      email,
      isAdmin,
      createdAt,
      lastName,
      isDeactivated,
    };
  }

  @UseGuards(AuthGuard)
  @ApiBearerAuth()
  @ApiOperation({ type: Boolean })
  @Post('update-user')
  async updateUser(
    @Request() { user: { userId } },
    @Body() userData: UpdateUserDTO,
  ) {
    await this.userService.updateUser({ id: userId, ...userData });
    return true;
  }

  @UseGuards(AdminGuard)
  @ApiBearerAuth()
  @ApiOperation({ type: Boolean })
  @Post('patch-user')
  async patchUser(@Body() userData: UpdateUserDTO) {
    await this.userService.updateUser(userData);
    return true;
  }

  @UseGuards(AdminGuard)
  @ApiBearerAuth()
  @ApiOperation({ type: Boolean })
  @ApiParam({ name: 'id', required: true, type: 'number' })
  @Post('deactivate-user/:id')
  async deactivateUser(@Param('id') id: number) {
    await this.userService.deactivateUser(id);
    return true;
  }

  @UseGuards(AdminGuard)
  @ApiBearerAuth()
  @ApiOperation({ type: Boolean })
  @ApiParam({ name: 'id', required: true, type: 'number' })
  @Post('activate-user/:id')
  async activateUser(@Param('id') id: number) {
    await this.userService.activateUser(id);
    return true;
  }
}

```

```

    }

    @UseGuards(AdminGuard)
    @ApiBearerAuth()
    @ApiResponse({ type: [MeDTO] })
    @Get('')
    async getUsers() {
        return this.userService.getAllUsers();
    }
}

```

Лістинг файлу analyze.service.ts

```

import { HttpException, Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Sales } from '../entities/sales.entity';
import { Repository } from 'typeorm';
import { CreateSaleDto } from '../dtos/create-sale.dto';
import * as csv from 'csv-parser';
import { Readable } from 'stream';
import axios from 'axios';

@Injectable()
export class AnalyzeService {
    constructor(
        @InjectRepository(Sales)
        private salesRepo: Repository<Sales>,
    ) {}

    private readonly pythonApiBaseUrl = 'http://analyzer:8000';

    async getFromPythonApi(endpoint: string) {
        try {
            const response = await axios.get(`${this.pythonApiBaseUrl}/${endpoint}`);
            return response.data;
        } catch (error) {
            throw new HttpException(
                error.response?.data || 'Error communicating with Python API',
                error.response?.status || 500,
            );
        }
    }

    async addOne(dto: CreateSaleDto) {
        const sale = this.salesRepo.create(dto);
        return await this.salesRepo.save(sale);
    }

    async deleteOne(id: number) {
        return await this.salesRepo.delete(id);
    }

    getAll(limit: number, page: number) {
        return this.salesRepo.find({ take: limit, skip: (page - 1) * limit });
    }

    async importCSV(csvContent: string) {
        const stream = Readable.from([csvContent]);
        const results: CreateSaleDto[] = [];

        return new Promise((resolve, reject) => {
            stream
                .pipe(csv())
                .on('data', (data) => results.push(this.normalize(data)))
                .on('end', async () => {

```

```

    await this.salesRepo.save(results);
    resolve({ count: results.length });
  })
  .on('error', reject);
});
}

normalize(data: any): CreateSaleDto {
  return {
    orderId: data.Order_ID,
    customerId: data.Customer_ID,
    customerType: data.Customer_Type,
    product: data.Product,
    category: data.Category,
    unitPrice: parseFloat(data.Unit_Price),
    quantity: parseInt(data.Quantity),
    discount: parseFloat(data.Discount),
    totalPrice: parseFloat(data.Total_Price),
    region: data.Region,
    orderDate: new Date(data.Order_Date).toISOString(),
  };
}
}

```

Лістинг файлу analyze.controller.ts

```

import {
  Body,
  Controller, Delete,
  Get,
  Param,
  Post,
  Query,
  UploadedFile,
  UseGuards,
  UseInterceptors,
} from '@nestjs/common';
import { FileInterceptor } from '@nestjs/platform-express';
import { CreateSaleDto } from '../dtos/create-sale.dto';
import { AnalyzeService } from '../analyze.service';
import { ApiBearerAuth, ApiBody, ApiConsumes, ApiQuery } from '@nestjs/swagger';
import { UploadCsvDto } from '../dtos/upload-csv.dto';
import { AuthGuard } from '../authorization/authorization.guard';
import { AdminGuard } from '../authorization/admin.guard';

@Controller('analyze')
export class AnalyzeController {
  constructor(private readonly analyzeService: AnalyzeService) {}

  @UseGuards(AdminGuard)
  @ApiBearerAuth()
  @Post()
  addOne(@Body() dto: CreateSaleDto) {
    return this.analyzeService.addOne(dto);
  }

  @UseGuards(AdminGuard)
  @ApiBearerAuth()
  @Post('upload')
  @ApiConsumes('multipart/form-data')
  @ApiBody({ type: () => UploadCsvDto })
  @UseInterceptors(FileInterceptor('file'))
  async uploadCSV(@UploadedFile() file: Express.Multer.File) {
    return this.analyzeService.importCSV(file.buffer.toString());
  }
}

```

```

@UseGuards(AuthGuard)
@ApiBearerAuth()
@Get('/:endpoint')
async proxyToPython(@Param('endpoint') endpoint: string) {
return this.analyzeService.getFromPythonApi(endpoint);
}

@UseGuards(AdminGuard)
@ApiBearerAuth()
@ApiQuery({ name: 'page', type: 'number', required: false, example: 1 })
@ApiQuery({ name: 'limit', type: 'number', required: false, example: 50 })
@Get()
async getAll(@Query() query: { page: number; limit: number }) {
return this.analyzeService.getAll(query.limit, query.page);
}

@ApiBearerAuth()
@UseGuards(AdminGuard)
@Delete('/:id')
async deleteSale(@Param('id') id: number) {
const deletionResult = await this.analyzeService.deleteOne(id);
return deletionResult.affected > 0;
}
}

```

Додаток Б

Лістинг файлу data.csv(скорочено)

```

Order_ID,Customer_ID,Customer_Type,Product,Category,Unit_Price,Quantity,Discount
,Totals_Price,Region,Order_Date
ORD1,CUS1496,B2B,Vio Wasser,Water,1.66,53,0.1,79.18,Baden-WTjrttemberg,2023-08-
23
ORD1,CUS1496,B2B,Evian,Water,1.56,90,0.1,126.36,Baden-WTjrttemberg,2023-08-23
ORD1,CUS1496,B2B,Sprite,Soft Drinks,1.17,73,0.05,81.14,Baden-WTjrttemberg,2023-
08-23
ORD1,CUS1496,B2B,Rauch Multivitamin,Juices,3.22,59,0.1,170.98,Baden-
WTjrttemberg,2023-08-23
ORD1,CUS1496,B2B,Gerolsteiner,Water,0.87,35,0.1,27.4,Baden-WTjrttemberg,2023-08-
23
ORD2,CUS2847,B2C,Sauvignon Blanc,Alcoholic Beverages,9.09,2,0.0,18.18,Schleswig-
Holstein,2023-03-16
ORD3,CUS1806,B2B,Tomato Juice,Juices,2.14,44,0.1,84.74,Hamburg,2022-11-20
ORD3,CUS1806,B2B,Vittel,Water,0.43,13,0.05,5.31,Hamburg,2022-11-20
ORD3,CUS1806,B2B,San Pellegrino,Water,1.21,92,0.1,100.19,Hamburg,2022-11-20
ORD3,CUS1806,B2B,Evian,Water,1.38,3,0.05,3.93,Hamburg,2022-11-20
ORD4,CUS2275,B2C,Mountain Dew,Soft Drinks,0.99,11,0.0,10.89,Bayern,2023-11-05
ORD5,CUS7919,B2B,Hohes C Orange,Juices,1.52,8,0.05,11.55,Bayern,2023-08-05
ORD5,CUS7919,B2B,Red Bull,Soft Drinks,2.13,16,0.05,32.38,Bayern,2023-08-05
ORD5,CUS7919,B2B,Chardonnay,Alcoholic Beverages,6.45,3,0.05,18.38,Bayern,2023-
08-05
ORD5,CUS7919,B2B,Tanqueray,Alcoholic
Beverages,33.36,43,0.15,1219.31,Bayern,2023-08-05
ORD6,CUS533,B2B,RotkГppchen Sekt,Alcoholic
Beverages,5.89,44,0.05,246.2,Saarland,2023-02-18
ORD6,CUS533,B2B,Mango Juice,Juices,2.5,10,0.05,23.75,Saarland,2023-02-18
ORD6,CUS533,B2B,Apollinaris,Water,0.82,13,0.05,10.13,Saarland,2023-02-18
ORD6,CUS533,B2B,Mountain Dew,Soft Drinks,1.87,89,0.15,141.47,Saarland,2023-02-18
ORD6,CUS533,B2B,Riesling,Alcoholic Beverages,11.48,21,0.05,229.03,Saarland,2023-
02-18
ORD7,CUS9366,B2C,Vio Wasser,Water,1.61,13,0.0,20.93,Baden-WTjrttemberg,2021-01-
12

```

ORD7,CUS9366,B2C,Rauch Multivitamin, Juices,2.31,7,0.0,16.17,Baden-Württemberg,2021-01-12
 ORD8,CUS6944,B2C,Granini Apple, Juices,1.64,2,0.0,3.28,Schleswig-Holstein,2022-12-03
 ORD8,CUS6944,B2C,Fanta,Soft Drinks,1.19,15,0.0,17.85,Schleswig-Holstein,2022-12-03
 ORD9,CUS6010,B2C,Hohes C Orange, Juices,1.8,2,0.0,3.6,Thüringen,2022-06-23

Додаток В

POST /analyze

```
{
  "orderId": "ORD4",
  "customerId": " CUS12275",
  "customerType": "B2C",
  "product": "ЖИВЧИК",
  "category": "Soft Drinks",
  "unitPrice": 0,
  "quantity": 4,
  "discount": 0,
  "totalPrice": 25,
  "region": "Zhytomyr",
  "orderDate": "2023-11-05"
}
```

POST /auth/sign-in

```
{
  "email": "test@example.com",
  "password": "123123"
}
```

POST /auth/request-password-recovery

```
{
  "email": "test@example.com"
}
```

POST /auth/change-password

```
{
  "token": "YOUR_TOKEN_HERE",
  "password": "321321"
}
```

Додаток Г

```
import { MigrationInterface, QueryRunner } from "typeorm";

export class UserTable1750357536227 implements MigrationInterface {
  name = 'UserTable1750357536227'

  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.query(`CREATE TABLE "users" ("id" SERIAL NOT NULL, "email"
character varying NOT NULL, "firstName" character varying NOT NULL, "lastName"
character varying, "isAdmin" boolean NOT NULL DEFAULT false, "isDeactivated"
boolean NOT NULL DEFAULT false, "password" character varying NOT NULL,
"created_at" TIMESTAMP NOT NULL DEFAULT now(), "updated_at" TIMESTAMP NOT NULL
```

```

DEFAULT now(), CONSTRAINT "UQ_97672ac88f789774dd47f7c8be3" UNIQUE ("email"),
CONSTRAINT "PK_a3ffb1c0c8416b9fc6f907b7433" PRIMARY KEY ("id"))`);
    await queryRunner.query('INSERT INTO "users" ("email", "firstName", "lastName",
"isAdmin", "isDeactivated", "password", "created_at", "updated_at") VALUES
(\\'user@example.com\\', \\'John\\', \\'Doe\\', true, false,
\\'$2b$10$3lhF8NL03lK9s5UxDcXR2OxvJ9dhmQtOinjXsQ3cLEc3efDwSHdO\\', now(),
now())`')
    }

    public async down(queryRunner: QueryRunner): Promise<void> {
        await queryRunner.query(`DROP TABLE "users"`);
    }
}

```