НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

До захисту допущено:

Завідувач кафедри

_____Олександр РОЛІК

«___»_____20___р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютеризовані системи управління»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
на тему: «Інформаційно-аналітична система управління магазином»

Виконав:
студент IV курсу, групи ІА-61
Черненко Максим Олексійович _____

Керівник:
ст. викладач
Моргаль Олег Михайлович _____

Рецензент:
ст. викладач
Польшакова Ольга Михайлівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент_____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-професійна програма «Комп'ютеризовані системи управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Олександр РОЛІК

«___»_____20___р.

ЗАВДАННЯ

на дипломний проєкт студенту

Черненко Максиму Олексійовичу

1. Тема проєкту «Інформаційно-аналітична система управління магазином», керівник проєкту ст. викладач Моргаль Олег Михайлович, затверджені наказом по університету від «__ »_____20 р. №_____

2. Термін подання студентом проєкту 05.06.2020р. _____

3. Вихідні дані до проєкту: обслуговування облікових записів системи, контроль роботи працівників, оптимізація та аналіз системи.

4. Зміст пояснювальної записки: вступ, 1 Опис предметної області, 2 Огляд існуючих ІАС управління магазином, 3 Опис розроблюваної ІАС, 4 Розробка веб-серверу, 5 Розробка інтерфейсу користувача, 6 Розробка механізмів та алгоритмів оптимізації та аналізу в ІАС.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): діаграма бази даних, Контекстна блок-схема алгоритму роботи системи, Схема структурна, Функціональна діаграма.

7. Дата видачі завдання 28.02.2020р. _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Огляд предметної області	13.03.2020	
2	Аналіз існуючих рішень	24.03.2020	
3	Опис розроблюваної ІАС	29.03.2020	
4	Вибір засобів розробки	16.04.2020	
5	Проектування програми та бази даних	26.04.2020	
6	Розробка веб-серверу	11.05.2020	
7	Розробка інтерфейсу користувача	23.05.2020	
8	Тестування та відлагодження	30.05.2020	
9	Оформлення звіту	05.06.2020	

Студент Максим Черненко

Керівник Олег Моргаль

АНОТАЦІЯ

Черненко М.О. Інформаційно-аналітична система управління магазином. КПІ ім. Ігоря Сікорського, Київ, 2020.

Проєкт містить 76 с. тексту, 38 рисунків, 5 таблиць, посилання на 15 літературних джерел, 4 додатки та 4 конструкторських документа.

Ключові слова: інформаційно-аналітична система, аналіз, оптимізація, облікові записи, фреймворк, прибуток, попит, виручка, JavaScript, веб-сервер, технологія.

Об’єктом розробки є інформаційно-аналітична система управління магазином.

Мета розробки – створення інформаційно-аналітичної системи, яка буде надавати додаткові можливості контролю користувачів системи, проводити аналіз роботи магазину на основі облікових записів та надавати можливості максимізації виручки і прибутку при зміні ціни на товари.

У дипломному проєкті розроблено інформаційно-аналітичну систему з сучасним інтерфейсом, простим адмініструванням та додатковими можливостями аналізу та оптимізації роботи та діяльності магазину. Для цього були розроблені і оптимізовані алгоритми роботи з сутностями системи, алгоритми аналізу роботи працівників, постачальників та продаж. Значну увагу було приділено модулю оптимізації, який має доступ до повної звітності магазину і дозволяє проводити процеси максимізації прибутку.

Отримані результати можуть бути корисними при автоматизації роботи з магазином у малому та середньому бізнесі.

SUMMARY

Chernenko M.O. Information and analytical system of store management. Igor Sikorsky KPI, Kyiv, 2020.

The project contains 76 pages. text, 38 figures, 5 tables, links to 15 literary sources, 4 annexes and 4 design documents.

Keywords: information-analytical system, analysis, optimization, accounts, framework, profit, demand, revenue, JavaScript, web-server, technology.

The object of development is the information and analytical system of store management.

The purpose of the development - creation of an information-analytical system that will provide additional opportunities to control system users, analyze the operation of the store on the basis of accounts and provide opportunities to maximize revenue and profits when changing the price of products.

The graduation project developed information-analytical system with a modern interface, simple administration and additional capabilities for analysis and optimization of the work and activities of the store. For this purpose, algorithms for working with system entities, algorithms for analyzing the work of employees, suppliers and sales were developed and optimized. Considerable attention has been paid to the optimization module, which has access to full reporting of the store and allows for profit maximization processes.

The results obtained can be useful in automating of work with the store in small and medium-sized businesses.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Приміт.
1			Документація загальна			
2						
3						
4			Заново розроблена			
5						
6	А4	ІА61.290БАК.005 ТП	Відомість технічного проекту	1		
7	А4	ІА61.290БАК.005 ПЗ	Пояснювальна записка	76		
8						
9	А3	ІА61.290БАК.005 Э1	Схема структурна	1		
10	А3	ІА61.290БАК.005 Д1	Функціональна діаграма	1		
11	А3	ІА61.290БАК.005 Д2	Контекстна блок-схема	1		
12	А3	ІА61.290БАК.005 Д3	Діаграма бази даних	1		
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата
Розроб.		Черненко М.О.		
Перевір.		Моргаль О.М.		
Реценз.				
Н. Контр.				
Затв.				

Літ.	Аркуш	Аркушів
		1

Пояснювальна записка
до дипломного проєкту
на тему: «Інформаційно-аналітична система управління магазином»

Київ – 2020 року

ЗМІСТ

СПИСОК ВИКОРИСТАНИХ У РОБОТІ СКОРОЧЕНЬ

АСПД – Автоматизовані Системи Печаті Документів,

IAC – Інформаційно-Аналітична Система,

CRUD – Create Read Update Delete,

DND – Drag and Drop,

JSX – JavaScript XML,

JWT – JSON Web Token,

MERN – MongoDB Express React Node.js,

UX – User Experience,

OLAP - Online Analytical Processing

ВСТУП

Інформаційно-аналітична система управління магазином - це комп'ютерна система, яка дозволяє отримувати інформацію, створювати її та здійснювати її обробку та аналіз. Завданням системи є ефективне зберігання, обробка та аналіз даних. Технологічна платформа системи дозволяє магазину здійснювати інтеграцію та координацію його бізнес-процесів.

Наразі існує велика кількість інформаційно-аналітичних систем для різних підприємств, організацій та магазинів, які дозволяють не просто оперувати даними системи та бізнесу, але й забезпечують додатковий функціонал, такий як: аналіз роботи працівників, постачальників, дослідження попиту на товари та ін. Нажаль, часто такі системи мають не дуже приємний інтерфейс та проблеми і складнощі з адмініструванням.

Метою даної роботи є створення інформаційно-аналітичної системи магазина, яка б дозволяла дуже зручно оперувати сутностями системи, мала зручний та приємний інтерфейс, просте адміністрування, надавала додаткові можливості контролю користувачів системи, проводила аналіз роботи магазину на основі облікових записів та надавала можливості максимізації виручки і прибутку при зміні ціни на товари.

Для досягнення поставленої мети були вирішені наступні завдання:

- огляд існуючих рішень інформаційно-аналітичних систем;
- формування вимог до розроблюваного програмного забезпечення та вибір засобів розробки;
- розробка архітектури програми;
- створення бази даних;
- створення інтерфейсу користувача програми;
- розроблення та оптимізація алгоритмів аналізу роботи магазину;

- впровадження алгоритму максимізації виручки та прибутку.

Кожна ІАС розрахована на певний тип підприємства, організації, бізнесу, адже всі вони мають свої особливості. Враховуючи меншу кількість таких систем спрямованих на роботу з магазином, їх обмежений функціонал та простоту, розроблювана програма спрямована на роботу саме з магазином з метою надати кінцевим користувачам зручну, потужну програму з розширеними можливостями по аналізу системи.

Для створення даного проекту використано наступні засоби та технології розробки програмного забезпечення: Node.js, як платформа для роботи з серверними скриптами для веб-запитів; Express.js, як технологія для створення каркасу бекенду на платформі Node.js; MongoDB, як документно-орієнтована система керування базами даних (СКБД); React, як бібліотека для створення інтерфейсу користувача.

Результуючу програму можна використовувати як типовий веб-застосунок на любій операційній системі.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Принцип роботи

ІАС управління магазином оперує первинними даними, які консолідуються в центрі обробки даних (ЦОД) в режимі реального часу. Завдяки цьому сформовані за допомогою ІАС звіти максимально оперативно відображають процеси, що протікають в магазині.

Обробка даних в ІАС виробляється на основі OLAP-технологій, що дозволяє отримувати результат у лічені секунди. Інформація в ІАС структурована у вигляді так званих OLAP-кубів, які об'єднують дані, що містяться в системі магазину. Така технологія обробки інформації забезпечує динамічну побудову багатовимірних звітів в різних розрізах, проведення аналізу великих обсягів даних в режимі реального часу, моніторинг і прогнозування ключових показників діяльності учасників системи. За допомогою ІАС власники і керівники магазину отримують чітке уявлення про те як проходять бізнес процеси та життєвий цикл магазину.

1.2 Основні функції

1.2.1 Інформаційний модуль

Інформаційний модуль призначений для перегляду довідкової інформації про стан системи з використанням табличного підходу.

1.2.2 Модуль моніторингу та аналізу

Модуль моніторингу та аналізу дозволяє проводити довільний аналіз результатів діяльності системи магазину на основі первинних даних.

1.2.3 Модуль звітності

Модуль звітності призначений для формування фінансових, звітних і аналітичних даних про статистичну та економічну облікову діяльність магазину. Надає такі можливості:

- створення довільних звітів фіксованою форми для різних періодів;
- можливість ручного і автоматичного заповнення осередків звітів, використання раніше створених в модулі звітності документів;
- передача сформованого звіту на контроль або переформування;
- автоматичне збереження даних звіту в разі втрати зв'язку з сервером з подальшим відновленням даних;
- установка контролю типів даних в осередках звітів;
- друк звіту.

1.3 Аспекти проблеми аналізу та їх реалізація в програмних продуктах

Вся проблема аналітичної підготовки прийняття рішень має такі аспекти:

- витяг з багатьох джерел різнорідних даних, представлених в різних форматах і приведення їх до єдиного формату і єдиній структурі;
- організація зберігання та надання користувачам необхідної для прийняття рішень інформації;
- власне аналіз, в тому числі оперативний і інтелектуальний, і підготовка планової або регулярної оцінки стану керованого об'єкта у вигляді паперових документів або екранних форм;
- підготовка результатів оперативного та інтелектуального аналізу для ефективного їх сприйняття споживачами та прийняття на основі адекватних рішень.

Аспект, що стосується збору та зберігання інформації з супутньою доопрацюванням, оформився в концепцію інформаційних сховищ (Data Warehouse). Ця концепція полягає в тому, що відомості про діяльність магазину, підприємства чи організації накопичуються протягом тривалого періоду часу (роки) в інформаційному сховищі за певними правилами. Накопичені дані використовуються в різних часових режимах для аналізу, як джерело даних для різного роду звітності і роботи з партнерами (Reporting) і обґрунтування управлінських рішень. У зв'язку з великим обсягом і складністю аспект проблеми власне аналізу має два напрямки - оперативний аналіз даних (інформації), широко поширена абревіатура англомовної назви - Online Analytical Processing - OLAP. Основним завданням оперативного або OLAP-аналізу є швидке (в межах секунд) вилучення необхідної аналітики для обґрунтування інформації або прийняття рішення. Інтелектуальний аналіз інформації - має також широко поширене спеціальній літературі англомовну назву Data mining. Призначений для фундаментального дослідження проблем в тій чи іншій предметної області. вимоги по часу менш жорсткі, але використовуються більш складні методики. Ставляться, як правило, завдання і отримують результати стратегічного значення. При вирішенні складних завдань в режимі Data mining доводиться використовувати досить потужні спеціальні програмні інструменти [1].

Аспекти проблеми аналізу і необхідні для їх вирішення функції знайшли вираз у відповідних програмних продуктах. Відповідно засоби автоматизації аналізу представлені в різних видах. Є комплексні інформаційно-аналітичні системи, які виконують в тій чи іншій мірі функції відповідно до розглянутими аспектами. Представлені на ринку програмних продуктів і цільові програмні системи, які виконують у збільшеному обсязі, розширеному складі і підвищеної складності функції, наприклад оперативного або інтелектуального аналізу. ІАС інформаційно підживлюють системи підтримки прийняття рішень (СППР), в літературі також застосовують абревіатуру DSS (Decision

Support System). В цілому склався ринок інструментальних засобів створення і підтримки OLAP-систем, інформаційних сховищ (DWH), СППР (DSS), інтелектуального аналізу Data mining (DM), який отримав узагальнену назву - Business intelligence (BI), якому поки не підібраний відповідний український термін . Як правило, всі інструментальні засоби, призначені для автоматизації аналітичних робіт, пристосовані для обробки багатовимірних масивів інформації; мають також можливість імпорту / експорту даних в інші операційні середовища, розвинені засоби візуального двовимірного (2D) і тривимірного (3D) подання інформації. Модулі, призначені для виконання функцій OLAP- аналізу, входять також і до складу інтегрованих інформаційних систем (ІВС) (системи, що виконують весь комплекс автоматизації робіт в інформаційному просторі економічного або якого-небудь іншого об'єкта). Найбільш розвинені ІВС виконують функції і оперативного та інтелектуального аналізу [2].

1.4 Функціональний склад

Слід зауважити, що ІАС відіграє об'єднуючу роль, консолідує розрізнені ІТ-технології в єдину інтегровану інформаційну систему управління магазином, організацією тощо, як її називають ІІСУП. АСУ ТП - автоматизовані системи управління технологічними процесами. САПР - системи автоматизованого проектування. ЕСУД - електронні системи управління документообігом. ІІСУП - інтегровані системи управління підприємством (Рисунок 1.1).

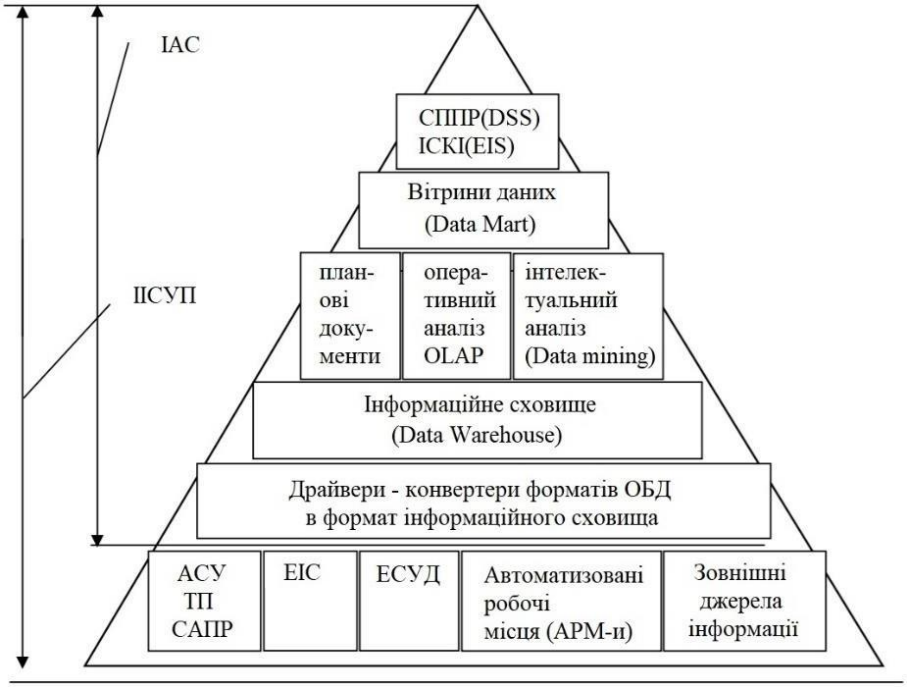

Рисунок 1.1 - Функціональний склад і місце ІАС в забезпеченні магазина ІТ-технологіями

Отже, цільові програмні продукти і ІОС (інформаційно-обчислювальні системи) досить дорогі і поки малодоступні для масового українського споживача. Виходом з цього положення є використання на практиці можливостей масових програмних інструментальних засобів як Excel, Mathcad, Stadia, Statistica та ін.

2 ОГЛЯД ІСНУЮЧИХ ІАС УПРАВЛІНЯ МАГАЗИНОМ

2.1 Огляд можливостей ІАС «Мій магазин»

Програма «Мій магазин» (Рисунок 2.1) може використовуватися для:

- магазину біля будинку;
- універсаму;
- спеціалізованого магазину;
- магазину на ринку.

Переваги програми:

- необхідність додаткового доопрацювання відсутня;
- автоматизація магазину займає 5 робочих днів;
- для роботи потрібні тільки базові знання комп'ютер

Рисунок 2.1 – Устрій програми ІАС «Мій магазин»

Робота на касі:

- каса працює безперебійно і автономно;
- готівковий і безготівковий розрахунок;
- робота з кількома юридичними особами на одній касі;
- робота зі штучним і ваговим товаром.

Бухгалтерія:

- управління ціноутворенням;
- робота з роздрібними, дрібногруповим, внутрішнім прайс-листами;
- облік витрат і доходів за різними статтями;
- докладний облік (програма зберігає дату поставки і закупівельну ціну, і не усереднює собівартість товару);
- розрахунок розміру кінцевої чистого прибутку за період.

Звітність:

- звіти про рух товарів, груп товарів і грошей;
- звіти по постачальникам, складах, секціях, чеками оперативні звіти;
- звіти за певний період;
- вивантаження даних в сторонні системи.

Робота складу:

- управління асортиментом;
- робота з постачальниками;
- передпродажна підготовка товару;
- робота з оптовими покупцями.

Інвентаризація:

- за допомогою терміналу збору даних;
- за допомогою сканера і ноутбука;
- без використання технічних засобів, шляхом перерахунку та "ручного" заповнення інвентаризаційної відомості;
- швидка переоцінка товару.

Залучення і утримання клієнтів:

- знижки за категоріями клієнтів;
- створення власних акцій;
- розмір знижки може знижуватися або бути нульовим, якщо націнка на ці товари не дозволяє давати знижки.

2.2 Огляд можливостей ІАС «GBS.Market»

Програма «GBS.Market» (Рисунок 2.2) може використовуватися для автоматизація обліку в роздрібній торгівлі для малого і середнього бізнесу.

Переваги програми «GBS.Market»:

- немає необхідності вивчати бухоблік або отримувати спеціалізовану підготовку;
- невисока вартість безстрокової ліцензії. 30 днів безкоштовного тестування;
- підтримує всі напрямки роздрібної торгівлі. Спеціальний модуль для кафе.

Облік продажів необхідний для будь-якого бізнесу, при цьому абсолютно байдуже, є у вас фізичний товар або ваша компанія займається наданням послуг. Починають все з зошитів або EXCEL. Зі збільшенням асортименту та персоналу, розширенням одного магазина або кафе до мережі, облік в зошиті стає недоцільним, а в деяких випадках навіть збитковим. Логічний висновок: потрібна автоматизація.

Можливості роботи:

- робота з асортиментом:
    - додавання товарів вручну і з Excel;
    - облік продажів за готівку, банківською картою і в борг;
    - створення складених товарів і комплектів;
    - переміщення між торговими точками;

- статистика продаж:
    - побудова звітів і діаграм за будь-який період;
    - генерація звітів по днях, місяцях або категорій;
    - сортування статистки по часу, найменуванню, вартості;
    - друк і експорт статистики в Excel;
- контроль на дому:
    - контроль необмеженої кількості ваших магазинів;
    - управління базою магазину з дому;
    - перегляд статистики в реальному часі;
    - звіти на електронну пошту з заданим інтервалом часу;
- робота з базою покупців:
    - ведення бази покупців і клієнтів;
    - дисконтна система і управління знижками;
    - накопичувальна система (бонуси / бали);
    - облік боржників;
- контроль за співробітниками:
    - статистика продажів для кожного співробітника;
    - налаштування прав доступу до даних;
    - "підписи" співробітників при виконанні дій;
- інвертизація і ревізії:
    - введення даних вручну або за допомогою сканера;
    - автоматичне коректування бази даних;
    - призупинення та скасування інвентаризації;
- друк документів:
    - цінники і етикетки будь-якого формату;
    - товарні чеки, накладні, рахунок в момент продажу;
    - накладні, акти інвертизації і списання;
    - гнучкий редактор шаблонів документів;
- підтримка обладнання:

- сканери штрих-кодів і дисконтних карти;
- принтери і етикеток;
- фіскальні реєстратори, онлайн-каси і АСПД;
- дисплеї покупців;
- торговельні ваги;

- працює без інтернету:
  - програма встановлюється на ваш комп'ютер і може працювати без підключення до інтернету. Таким чином GBS.Market можна використовувати навіть у віддалених регіонах.

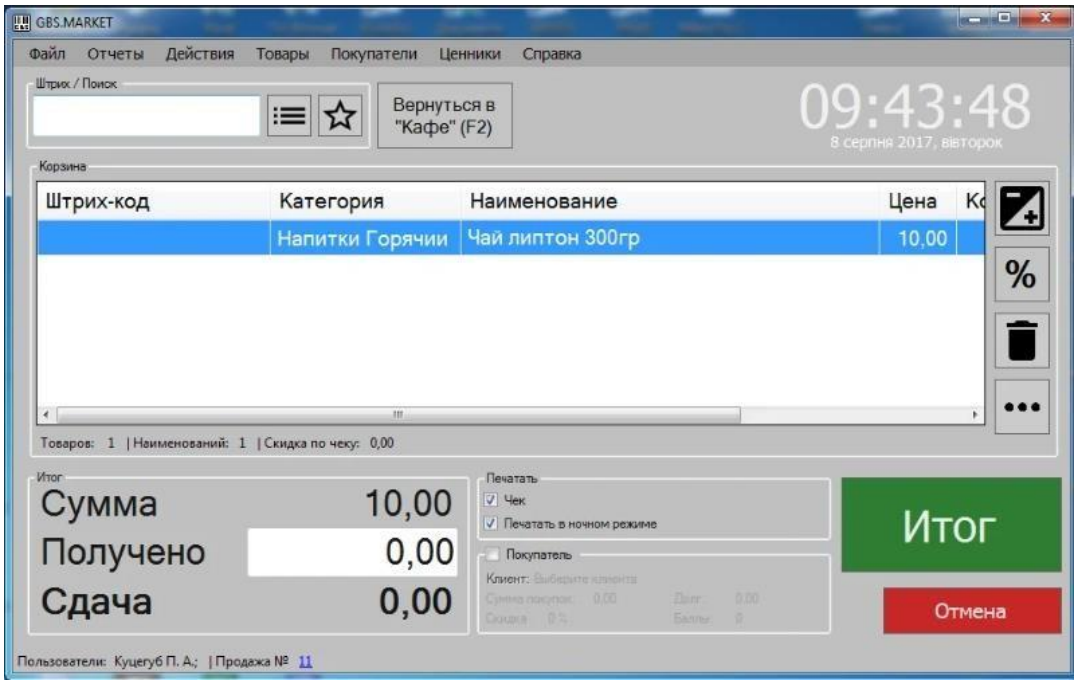

Рисунок 2.2 – Загальний вигляд ІАС «GBS.Market»

Були проаналізовані два аналоги ІАС управління магазином, а саме зазначені можливості, переваги та практичне використання програм «Мій магазин» та «GBS.Market».

3 ОПИС РОЗРОБЛЮВАНОЇ ІАС

3.1 Інтерфейс користувача

Усі розглянуті в попередньому розділі ІАС мають простий та інтуїтивно зрозумілий, навіть недосвідченому користувачеві інтерфейс, але вигляд та графічні можливості бажають кращого. Найбільш вдалим, з точки зору зручності, компактності та логічності, я вважаю інтерфейс ІАС «Мій магазин», але інтерфейс власної ІАС я намагатимусь зробити якомога більш продуманим, який буде відповідати сучасним тенденціям дизайну.

Отже, вихідний продукт буде мати:

- підтримку клавіш швидкого доступу та буде зручним для використання людям з обмеженими можливостями. Навігація матиме прямий та зворотній напрямок з любої частини програми;
- дві сторінки для входу та реєстрації у систему;
- сторінки вибору існуючого та створення нового магазину;
- сторінку меню магазина з можливістю налаштовувати її вигляд за допомогою DND;
- сторінку конфігурації облікових записів, яка матиме можливість вибору необхідної сутності за допомогою вертикальних табів. Кожна така сутність матиме приємне відображення та функціонал створення, фільтру, сортування, редагування, видалення та імпорту існуючих даних;
- сторінку життєвого циклу магазину, виконану за допомогою горизонтальних табів, буде можливість встановлювати фільтр історії за користувачем;
- сторінку аналізу роботи системи, виконану у вигляді меню панелей розширення з можливістю зміни їх розташування за допомогою техніки DND;

- сторінку оптимізації, виконану у вигляді степера, де користувач матиме зручну можливість проаналізувати ціноутворення товару та побудувати криву прибутку;
- для підтвердження та спостереження прогресу операцій над даними відображатимуться спеціальні діалогові вікна та лоадери;
- система буде відловлювати любі можливі помилки та відображатиме їх користувачеві;
- стиль та кольорова гама оформлення програми відповідатиме новим тенденціям дизайну та буде легко сприйматися користувачем.

3.2 Функціональні можливості

Крім основних операцій характерних ІАС, таких як створення, редагування, видалення, пошук та сортування розроблювана програма має підтримувати такі додаткові можливості:

- аналіз попиту на товари;
- аналіз роботи працівників;
- аналіз роботи з постачальниками;
- аналіз товару з можливостю підбирати необхідну кількість продажів при відповідній зміні ціни та побудова кривої збереження прибутку в діапазоні 25% від початкової ціни;
- наявність гнучкої системи прав. Є власник, адміністратор та користувач. Останній має обмеженні можливості, адміністратор – привілейовані, а саме: надається можливість редагування та видалення сутностей. Власник має додаткову можливість по редагуванню самого магазину, може змінити назву та видалити його.

Для всього вказаного функціоналу необхідно застосувати найкращі алгоритми, які дозволять не тільки отримати бажаний результат, але й зробити це з мінімальним затратами часу та пам’яті. Особливу увагу слід звернути на

алгоритми, швидкодія яких є критичним показником, такі як процедури пошуку в базі даних IAC.

3.3 Засоби розробки

3.3.1 Засоби розробки бекенду

Основою бекенду цієї системи є платформа Node.js. Вона створена на движку V8 (здійснює трансляцію JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованої мови в мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями введення-виведення через свій API (написаний на C ++), підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript-коду. Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти на Node.js і десктопні віконні додатки (за допомогою NW.js, AppJS або Electron для Linux, Windows і macOS) і навіть програмувати мікроконтролери (наприклад, tessel, low.js і espruino). В основі Node.js лежить подієво-орієнтоване і асинхронне (або реактивне) програмування з неблокуючим введенням / виводом [9].

В якості каркасу сервера використовується фреймворк Express, реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT. Він спроектований для створення веб-застосунків і API. Де-факто є стандартним каркасом для Node.js. Автор фреймворку, TJ Holowaychuk, описує його як створений на основі написаного на мові Ruby каркаса Sinatra, маючи на увазі, що він мінімалістичний і включає велику кількість додаткових плагінів [6]. Express може бути бекендом для програмного стека MEAN, разом з базою даних MongoDB і каркасом Vue.js, React або AngularJS для фронтенду [11]. В цій системі я буду використовувати стек MERN.

В якості бази даних використовується MongoDB - документоорієнтована система управління базами даних з відкритим вихідним кодом, яка не потребує

опису схеми таблиць. Класифікована як NoSQL, використовує JSON-подібні документи і схему бази даних. Написана на мові C ++. Використовується в веб-розробці, зокрема, в рамках JavaScript-орієнтованого стека MEAN [7].

Можливості MongoDB:

- система підтримує ad-hoc-запити: вони можуть повертати конкретні поля документів і призначені для користувача JavaScript-функції. Підтримується пошук за регулярними виразами. Також можна налаштувати запит на повернення випадкового набору результатів;
- є підтримка індексів;
- система може працювати з набором реплік, тобто містити дві або більше копії даних на різних вузлах. Кожен екземпляр набору реплік може в будь-який момент виступати в ролі основної або допоміжної репліки. Всі операції запису і читання за замовчуванням здійснюються з основною реплікою. Допоміжні репліки підтримують в актуальному стані копії даних. У разі, коли основна репліка дає збій, набір реплік проводить вибір, яка з реплік повинна стати основною. Другорядні репліки можуть додатково бути джерелом для операцій читання;
- система масштабується горизонтально, використовуючи техніку сегментування об'єктів баз даних - розподіл їх частин з різних вузлів кластера. Адміністратор вибирає ключ сегментування, який визначає, за яким критерієм дані будуть рознесені по вузлах (в залежності від значень хешу ключа сегментування). Завдяки тому, що кожен вузол кластера може приймати запити, забезпечується балансування навантаження.

База даних MongoDB підходить для наступних застосувань:

- зберігання та реєстрація подій;
- системи управління документами і контентом;
- електронна комерція;

- дані моніторингу, датчиків;
- мобільні додатки;
- сховище операційних даних веб-сторінок (наприклад, зберігання коментарів, оцінок, профілів користувачів, сеанси користувачів).

3.3.2 Засоби розробки інтерфейсу

React - JavaScript-бібліотека з відкритим вихідним кодом для розробки призначених для користувача інтерфейсів. React розробляється і підтримується Facebook, Instagram і співтовариством окремих розробників і корпорацій. React може використовуватися для розробки односторінкових і мобільних додатків. Його мета - надати високу швидкість, простоту і масштабованість. Як бібліотека для розробки призначених для користувача інтерфейсів React часто використовується з іншими бібліотеками, такими як Redux і GraphQL.

Особливості React:

- односпрямована передача даних. Властивості передаються від батьківських компонентів до дочірнім. Компоненти отримують властивості як безліч незмінних значень, тому компонент не може безпосередньо змінювати властивості, але може викликати зміни через callback функції. Такий механізм називають «властивості вниз, події наверх».
- віртуальний DOM. React використовує віртуальний DOM. React створює кеш структуру в пам'яті, що дозволяє обчислювати різницю між попереднім і поточним станами інтерфейсу для оптимального поновлення DOM браузера. Таким чином програміст може працювати зі сторінкою, вважаючи, що вона оновлюється вся, але бібліотека самостійно вирішує, які компоненти сторінки необхідно оновити.

- JSX - це розширення синтаксису JavaScript, яке дозволяє використовувати схожий на HTML синтаксис для опису структури інтерфейсу. Як правило, компоненти написані з використанням JSX, але також є можливість використання звичайного JavaScript. JSX нагадує іншу мову, створений в компанії Фейсбук для розширення PHP, XHP [4].

Redux - бібліотека для JavaScript з відкритим вихідним кодом, призначена для управління станом додатків. Найчастіше використовується в зв'язці з React або Angular для розробки клієнтської частини. Містить ряд інструментів, що дозволяють значно спростити передачу даних сховища через контекст. Вона працює за тим же принципом, що і функція reduce, один з концептів функціонального програмування. Її творці: Данило Абрамов і Ендрю Кларк надихалися функціональним мовою програмування Elm [5].

3.4 Середовище та мова розробки

JavaScript – мультипарадигмальна мова програмування. Підтримує об'єктно-орієнтований, імперативний і функціональний стилі. Є реалізацією стандарту ECMAScript. JavaScript зазвичай використовується як вбудований мова для програмного доступу до об'єктів додатків. Найбільш широке застосування знаходить в браузерах як мова сценаріїв для додання інтерактивності веб-сторінок. Основні архітектурні риси: динамічна типізація, слабка типізація, автоматичне керування пам'яттю, прототипне програмування, функції як об'єкти першого класу. На JavaScript вплинули багато мов, при розробці була мета зробити мову схожим на Java, але при цьому легким для використання непрограмістів. Мовою JavaScript не володіє будь-яка компанія або організація, що відрізняє його від ряду мов програмування, використовуваних в веб-розробці.

JavaScript є об'єктно-орієнтованою мовою, але використовуване в мові прототипування обумовлює відмінності в роботі з об'єктами в порівнянні з традиційними клас-орієнтованими мовами. Крім того, JavaScript має ряд властивостей, властивих функціональним мовам: функції як об'єкти першого класу, об'єкти як списки, каррінг, анонімні функції, замикання - що додає мові додаткову гнучкість.

Незважаючи на схожий з C синтаксис, JavaScript в порівнянні з мовою C має корінні відмінності:

- об'єкти з можливістю інтроспекції;
- функції як об'єкти першого класу;
- автоматичне приведення типів;
- автоматичне прибирання сміття;
- анонімні функції.

У мові відсутні такі корисні речі, як:

- стандартна бібліотека: зокрема, відсутній інтерфейс програмування додатків по роботі з файловою системою, управління потоками введення-виведення, базових типів для бінарних даних;
- стандартні інтерфейси до веб-серверів і баз даних;
- система управління пакетами, яка б відстежувала залежності і автоматично встановлювала їх.

Синтаксис мови JavaScript багато в чому нагадує синтаксис C і Java, семантично ж мова набагато ближче до Self, Smalltalk або навіть Ліспі.

В JavaScript:

- всі ідентифікатори чутливі до регістру;
- в назвах змінних можна використовувати букви, підкреслення, символ долара, арабські цифри;
- назви змінних не можуть починатися з цифри;

- для оформлення однорядкових коментарів використовуються //, багаторядкові внутрішні коментарі починаються з / * і закінчуються * /.

Структурно JavaScript можна представити у вигляді об'єднання трьох чітко помітних одна від одної частин:

- ядро (ECMAScript),
- об'єктна модель браузера (Browser Object Model або BOM),
- об'єктна модель документа (Document Object Model або DOM).

Якщо розглядати JavaScript в відмінних від браузера середовищах, то об'єктна модель браузера і об'єктна модель документа можуть не підтримуватися. Об'єктну модель документа іноді розглядають як окрему від JavaScript сутність, що узгоджується з визначенням DOM як незалежного від мови інтерфейсу документа. На противагу цьому ряд авторів знаходить BOM і DOM тісно взаємопов'язаними.

Галузі застосування JavaScript:

- веб-застосунки. JavaScript використовується в клієнтської частини веб-застосунків: клієнт-серверних програм, в якому клієнтом є браузер, а сервером - веб-сервер, що мають розподілену між сервером і клієнтом логіку. Обмін інформацією в веб-застосунках відбувається по мережі. Одним з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-застосунки є кросплатформеними сервісами;
- AJAX. JavaScript використовується в AJAX, популярному підході до побудови призначених для користувача інтерфейсів веб-застосунків, що полягає в «фоновому» асинхронному обміні даними браузера з веб-сервером. В результаті, при оновленні даних веб-сторінка не перезавантажується повністю і інтерфейс веб-застосунка стає швидше, ніж це відбувається при традиційному підході;

- Comet. Це широке поняття, яке описує механізм роботи веб-застосунків, що використовують постійні HTTP-з'єднання, що дозволяє веб-сервера відправляти дані браузеру без додаткового запиту з боку браузера. Для таких додатків використовуються технології, безпосередньо підтримувані браузерами. Зокрема, в них широко використовується JavaScript;
- браузерні операційні системи. JavaScript широко використовується в браузерних операційних системах. Так, наприклад, вихідний код IndraDesktop WebOS на 75% складається з JavaScript, код браузерної операційної системи IntOS - на 70%. Частка JavaScript у вихідному коді eyeOS - 5%, однак і в рамках цієї операційної системи JavaScript грає важливу роль, беручи участь в візуалізації на клієнті і будучи необхідним механізмом для клієнта і сервера.

3.4.1 Ядро

ECMAScript не є браузерні мовою і в ньому не визначаються методи введення і виведення інформації. Це, скоріше, основа для побудови скриптових мов. Специфікація ECMAScript описує типи даних, інструкції, ключові і зарезервовані слова, оператори, об'єкти, регулярні вирази, не обмежуючи авторів похідних мов в розширенні їх новими складовими.

3.4.2 Об’єктна модель браузера

Об'єктна модель браузера - браузер-специфічна частина мови, що є прошарком між ядром і об'єктною моделлю документа. Основне призначення об'єктної моделі браузера - управління вікнами браузера і забезпечення їх взаємодії. Кожне з вікон браузера представляється об'єктом window, центральним об'єктом DOM. Об'єктна модель браузера на даний момент не

стандартизована, однак специфікація знаходиться в розробці WHATWG і W3C.

Крім управління вікнами, в рамках об'єктної моделі браузера, браузерами зазвичай забезпечується підтримка наступних сутностей:

- управління кадрами;
- підтримка затримки у виконанні коду і зациклення з затримкою, системні діалоги;
- управління адресою відкритої сторінки;
- управління інформацією про браузері;
- управління інформацією про параметри монітора;
- обмежене управління історією перегляду сторінок;
- підтримка роботи з HTTP cookie.

3.4.3 Об’єктна модель документа

Об'єктна модель документа - інтерфейс програмування додатків для HTML і XML-документів. Згідно DOM, документ (наприклад, веб-сторінка) може бути представлений у вигляді дерева об'єктів, що володіють рядом властивостей, які дозволяють виробляти з ним різні маніпуляції:

- генерація і додавання вузлів,
- отримання вузлів,
- зміна вузлів,
- зміна зв'язків між вузлами,
- видалення вузлів.

Проаналізувавши аналоги були зроблені необхідні висновки на рахунок інтерфейсу та функціоналу програми. Також були обрані технології для створення результуючого продукту.

4 РОЗРОБКА ВЕБ-СЕРВЕРУ

4.1 Загальні відомості про REST API

REST (Representational State Transfer - «передача стану уявлення») - архітектурний стиль взаємодії компонентів розподіленого додатка в мережі. REST є узгоджений набір обмежень, що враховуються при проектуванні розподіленої гіпермедіа-системи [9].

REST API має на увазі під собою прості правила:

- кожен URL є ресурсом;
- при зверненні до ресурсу методом GET повертається опис цього ресурсу;
- метод POST додає новий ресурс;
- метод PUT змінює ресурс;
- метод DELETE видаляє ресурс.

Ці правила надають простий CRUD інтерфейс для інших додатків, взаємодія з яким відбувається через протокол HTTP.

Відповідність CRUD операцій і HTTP методів:

- CREATE – POST;
- READ – GET;
- UPDATE – PUT;
- DELETE – DELETE.

REST API інтерфейс дуже зручний для міжпрограмної взаємодії, наприклад мобільний застосунок може виступати в ролі клієнта, який маніпулює даними за допомогою REST (Рисунок 4.1).

Властивості Архітектури REST:

- продуктивність - взаємодія компонентів системи може бути домінуючим фактором продуктивності і ефективності мережі з точки зору користувача;
- масштабованість для забезпечення великої кількості компонентів і взаємодій компонентів;
- простота уніфікованого інтерфейсу;
- відкритість компонентів до можливих змін для задоволення мінливих потреб (навіть при працюючому додатку);
- прозорість зв’язків між компонентами системи для сервісних служб;
- переносимість компонентів системи шляхом переміщення програмного коду разом з даними;
- надійність, що виражається в стійкості до відмов на рівні системи при наявності відмов окремих компонентів, з'єднань або даних.

Переваги REST:

- відсутність додаткових внутрішніх прошарків, що означає передачу даних в тому ж вигляді, що і самі дані. Тобто дані не обертаються в XML, як це робить SOAP і XML-RPC, не використовується AMF, як це робить Flash тощо. Просто віддаються самі дані;
- кожна одиниця інформації (ресурс) однозначно визначається URL - це значить, що URL по суті є первинним ключем для одиниці даних. Причому абсолютно не має значення, в якому форматі знаходяться дані за адресою - це може бути і HTML, і jpeg, і документ Microsoft Word;
- як відбувається управління інформацією ресурсу - це цілком і повністю ґрунтується на протоколі передачі даних. Найбільш поширений протокол звичайно ж HTTP. Для HTTP дію над даними задається за допомогою методів: GET (отримати), PUT (додати, замінити), POST (додати, змінити, видалити), DELETE (видалити).

Таким чином, дії CRUD можуть виконуватися як з усіма 4-ма методами, так і тільки за допомогою GET і POST.

4.2 Реалізація RESTful node.js API у IAC управління магазином

В каркасі серверу знаходиться фреймворк Express, який дозволяє дуже зручно будувати повноцінний сервер. Як можна побачити (Рисунок 4.2) за допомогою функції use, яка працює по принципу прошарок, ми можемо дуже легко надавати нашій серверній програмі новий функціонал.

```
const startServer = port => {
  app
    .use(bodyParser.urlencoded({ extended: true }))
    .use(bodyParser.json())
    .use(corsMiddleware())
    .use(morgan('dev'))
    .use('/auth', authRouter)
    .use(checkAuthMiddleware)
    .use('/store', storeRouter)
    .use('/history', historyRouter)
    .use('/position', positionRouter)
    .use('/employee', employeeRouter)
    .use('/product', productRouter)
    .use('/sale', saleRouter)
    .use('/supplier', supplierRouter)
    .use('/waybill', waybillRouter)
    .use('/analysis', analysisRouter)
    .use('/menu-config', menuConfigRouter)
    .use('/optimization', optimizationRouter)
    .use(errorMiddleware);

  app.listen(port);

  console.log(`Server was started at http://localhost:${port}`);
};
```

Рисунок 4.2 – Загальний вигляд Express серверу

4.2.1 Схеми бази даних MongoDB

Для зв’язку додатку node.js та бази даних MongoDB я буду використовувати бібліотеку Mongoose, вона дозволяє визначати схеми зі строго-типізований даними [8]. Відразу після визначення схеми Mongoose дає можливість створити модель, засновану на певній схемі. Потім модель синхронізується з документом MongoDB за допомогою визначення схеми моделі.

Відразу після визначення схем і моделей можна користуватися різними функціями Mongoose для перевірки, збереження, видалення і запиту даних, використовуючи звичайні функції MongoDB.

Для роботи розроблюваної системи мною були визначені та створені наступні схеми:

- схеми для роботи з обліковими записами IAC:
    - схема товару. Визначаються наступні властивості: ім’я, ціна, змінна вартість, оптова вартість, кількість та відношення до схем магазина та постачальника;
    - схема посади. Визначаються наступні властивості: назва, зарплата та відношення до схеми магазина;
    - схема постачальника. Визначаються наступні властивості: назва, номер телефона, місто та відношення до схеми магазина;
    - схема накладної. Визначаються наступні властивості: Ціна, кількість та відношення до схем товару та магазина;
    - схема працівника. Визначаються наступні властивості: ім’я, прізвище, посада, номер телефона, зарплата та відношення до схеми магазина;
    - схема продажі. Визначаються наступні властивості (Рисунок 4.3).

```
const saleSchema = new Schema({
  employeeId: { type: Types.ObjectId, ref: 'Employee' },
  productId: { type: Types.ObjectId, ref: 'Product' },
  date: Date,
  count: Number,
  price: Number,
  paymentType: {
    type: String,
    enum: ['cash', 'card'],
    description: 'Can only be one of the enum values and is required',
  },
  storeId: { type: Types.ObjectId, ref: 'Store' },
});
```

Рисунок 4.3 – Вигляд типової схеми на прикладі продажі

- схеми для повноцінної роботи системи:
  - схема користувача. Визначаються наступні властивості: ім’я, пароль, пошта та відношення до схеми магазина;
  - схема магазина. Визначаються наступні властивості: ім’я, власник та адміністратори;
  - схема користувацьких налаштувань меню. Визначаються наступні властивості: тип, елементи меню та відношення до схем користувача та магазина;
  - схема історії життєвого циклу. Визначаються наступні властивості: користувач, дія та відношення до схеми магазина;

4.2.2 Маршрути Express

Маршрутизація визначає, як застосунок відповідає на клієнтський запит до конкретної адреси (URI).

Метод route є похідним від одного з методів HTTP і приєднується до примірника класу express.

Express підтримує перераховані далі методи маршрутизації, які відповідають методам HTTP: get, post, put, head, delete, options, trace, copy, lock,

mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m -search, notify, subscribe, unsubscribe, patch, search і connect.

У цій системі будуть використовуватись основні методи: get, post, put та delete.

Шляхи маршрутів, в поєднанні з методом запиту, визначають конкретні адреси (кінцеві точки), в яких можуть бути створені запити. Шляхи маршрутів можуть являти собою рядки, шаблони рядків або регулярні вирази. Для обробки запиту можна вказати кілька функцій зворотного виклику, подібних middleware. Єдиним винятком є те, що ці зворотні виклики можуть ініціювати next ('route') для обходу інших зворотних викликів маршруту. За допомогою цього механізму можна включити в маршрут попередні умови, а потім передати управління подальшим маршрутами, якщо продовжувати роботу з поточним маршрутом не потрібно. Обробники маршрутів можуть приймати форму функції, масиву функцій або їх поєднання.

У цій системі я буду визначати для кожної сутності окремий express router. Всі такі маршрутизатори будуть під'єднані до сервера за допомогою функції підключення прошарок use. Перелік:

- маршрути для роботи з обліковими записами. Усі вони підтримують CRUD операції:
    - маршрути для роботи з посадами. Надають можливість отримати, створити, редагувати та видалити посаду;
    - маршрути для роботи з товарами. Надають можливість отримати, створити, редагувати та видалити товар;
    - маршрути для роботи з працівниками. Надають можливість отримати, створити, редагувати та видалити працівника;
    - маршрути для роботи з продажами. Надають можливість отримати, створити, редагувати та видалити продаж;
    - маршрути для роботи з постачальниками. Надають можливість отримати, створити, редагувати та видалити постачальника;

- маршрути для роботи з накладними. Надають можливість отримати, створити, редагувати та видалити накладну.

- маршрути для повноцінної роботи системи:
  - маршрути для роботи з історією життєвого циклу системи. Надають можливість отримати та створити історії;
  - маршрути для роботи з магазином (Рисунок 4.4). Надають можливість отримати всі, отримати один по id, створити, редагувати та видалити магазин. Також надати та видалити права адміністратора;
  - маршрути для роботи з блоком оптимізації. Надають можливість розрахувати дані для оптимізації та побудови кривої збереження рівня прибутку;
  - маршрути для роботи з користувачем. Надають можливість зареєструватися, увійти та вийти з системи. Також отримати всіх користувачів та статус авторизації, додати та видалити користувача з магазина;
  - маршрути для роботи з користувацькими налаштуваннями меню. Надають можливість отримати, створити, редагувати та видалити налаштування;
  - маршрути для роботи з блоком аналізу. Надають можливість дослідити роботу працівників, постачальників та побудувати рейтинг продуктів за попитом.

```
const storeRouter = express.Router();

storeRouter
  .get('/', getStores)
  .get('/:id', getStore)
  .post('/', createStore)
  .put('/:id', setAdminRights)
  .put('/info/:id', updateStore)
  .delete('/:id/', deleteStore)
  .delete('/:id/:userId', deleteAdminRights);
```

Рисунок 4.4 – Вигляд типового маршрутизатора на прикладі роботи з магазином

4.2.3 Прошарки та авторизація

У даній системі для розширення функціоналу використовуються такі прошарки:

- прошарка перехвату помилок (Рисунок 4.5), яка просто видає користувачу статус помилки та звітує про це;

```
const errorHandler = (err, _, res) => {
  console.error(err.message);

  res.json(404).send(err.message);
};
```

Рисунок 4.5 – Прошарка перехвату помилок

- прошарка встановлення часу (Рисунок 4.6), яка додає до кожної новоствореної сутності час створення, а для кожної оновленої – час оновлення;

```
const setTimestamp = schema => {
  schema.add({
    createdAt: Date,
    updatedAt: Date,
  });

  schema.pre('save', function addTimestamp(next) {
    const now = Date.now();

    this.updatedAt = now;

    if (!this.createdAt) {
      this.createdAt = now;
    }

    next();
  });
};
```

Рисунок 4.6 – Прошарка встановлення часу

- прошарка перевірки наявності токену для надання користувачу доступу для захищених маршрутів у системі (Рисунок 4.7).

```
const checkAuth = (req, res, next) => {
  const authorizationHeader = req.headers.authorization;
  let result;
  if (authorizationHeader) {
    const token = authorizationHeader.split(' ')[1]; // Bearer <token>

    const options = {
      expiresIn: '1h',
    };
    try {
      result = jwt.verify(token, jwtSecret, options);
      req.decoded = result;
      next();
    } catch (err) {
      throw new Error(err);
    }
  } else {
    result = {
      error: `Authentication error. Token required.`,
      status: 401,
    };
    res.status(401).send(result);
  }
};
```

Рисунок 4.7 – Прошарка перевірки токену

Авторизація у системі виконання класичним шляхом з використанням JWT - це відкритий стандарт (RFC 7519) для створення токенів доступу, заснований на форматі JSON. Як правило, використовується для передачі даних для аутентифікації в клієнт-серверних додатках. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який в подальшому використовує даний токен для підтвердження своєї особи.

Були розроблені допоміжні функції по роботі з токеном:

- функція генерації токену;
- функція перевірки та декодування токену.

Реєстрація у систему була реалізована таким чином, що на основі переданого користувачем пароля проводиться його хешування для подальшого збереження, далі проводиться валідація даних, якщо вона не проходить – то користувач отримує відповідну помилку з повідомленням про невірно введені дані. Далі проводиться перевірка на унікальність введених користувачем ім’я та пошти, якщо унікальні – новий користувач зберігається у системі, в противному випадку – відправляється помилка.

Вхід у систему працює таким чином, що проводиться пошук користувача по імені, якщо існує, то порівнюються хешований пароль з паролем, введеним користувачем, який у свою чергу також хешується за тим самим jwt-секретом, що і при реєстрації. Якщо хоч якась із умов не виконується – то користувач отримує у відповіді помилку.

Вихід із системи дуже простий, проводиться перевірка і декодування токену. Користувач у свою чергу отримує повідомлення о статусі операції.

В результаті був побудований сучасний, швидкий та безпечний веб-сервер, який задовольняє усім потребам IAC і надає повний контроль над базою даних.

5 РОЗРОБКА ІНТЕРФЕЙСУ КОРИСТУВАЧА

5.1 Опис використовуваних технологій

5.1.1 Стилі

Програма була виконана у стилі Material Design. Для свого додатку я використовував популярний фреймворк Material UI, який надає доступ к великій кількості корисних компонентів та іконок для подальшого використання. У своїй програмі були використані такі компоненти:

- TextField;
- Button;
- Fab;
- Tooltip;
- Breadcrumbs;
- Tabs;
- Menu;
- AppBar;
- Expansion Panel;
- Circular Progress;
- Linear Progress;
- Avatar;
- Modal;
- Typography.

Всім компонентам була надана унікальна стилістика та палітра кольорів. У якості select я використовував популярне рішення react-select, який з коробки надає багато можливостей та легко налаштовується за бажанням користувача.

Для відображення табличних даних була використана бібліотека ag-grid, яка надає багатофункціональні та зручні у використанні таблиці.

Для задавання стилів я використовував модель css-in-js, для цього було використане рішення styled-components, яке надає можливість користувачу задавати стилі для компонентів прямо у .js файлі. Перевага такого рішення в дуже легкому контролі та зміні стилів в залежності від стану компонента. Також зменшується кількість файлів у проекті, адже не потрібно створювати додаткові .css/.scss файли, усі стилі зберігаються в файлі компоненту.

5.1.2 Управління станом

Вимоги до функціонала додатків постійно зростають, в результаті зростає кількість станів інтерфейсу: переходи по раутам, лоадери, асинхронна завантаження даних, нескінченна кількість елементів інтерфейсу тощо. За всім ці необхідно стежити і обробляти. В один момент можна просто перестати вловлювати зв'язок між змінами, так як контроль над тим, коли, чому і як змінився стан втрачений через складність самого стану. Ідеальний варіант, це коли інтерфейс взагалі не знає про бізнес-логікою. У цьому допомагають такі бібліотеки управління станом, Redux і Mobx найпопулярніші. Для бекендів основаних на GraphQL є Apollo. В цій IAC я буди використовувати Redux.

Використання Redux в цій системі (Рисунок 5.1).

```
const middlewares = [thunk, authHeaderMiddleware, appHistoryMiddleware];

const enhancer = composeWithDevTools(applyMiddleware(...middlewares));

const store = createStore(rootReducer, enhancer);

const persistor = persistStore(store);
```

Рисунок 5.1 – Сховище Redux

Сховище було створене при використані rootReducer та enhancer. Enhancer включає в себе надання можливості використовувати devTools під час розробки та декілька прошарок: thunk, authHeader та appHistory. Прошарка

thunk (Рисунок 5.2) необхідна для того, щоб була можливість використовувати асинхронні actions для виконання запитів до серверу. Прошарка authHeader необхідна для встановлення/очищення токену в заголовку авторизації HTTP запиту. Це необхідно для того, щоб мати можливість відправляти запити на захищений сервер.

```
const authHeaderMiddleware = ({ getState }) => next => action => {
  if (
    action.type === actions.REFRESH_USER[START] ||
    action.type === actions.REFRESH_USER[SUCCESS]
  ) {
    const state = getState();
    const { token } = state.session;

    if (!token) return;

    setAuthHeader(token);
  }
  if (
    action.type === actions.SIGN_UP[SUCCESS] ||
    action.type === actions.SIGN_IN[SUCCESS]
  ) {
    const { token } = action.data;
    setAuthHeader(token);
  }
  if (
    action.type === actions.SIGN_OUT[SUCCESS] ||
    action.type === actions.REFRESH_USER[ERROR]
  ) {
    clearAuthHeader();
  }
  next(action);
};
```

Рисунок 5.2 – Прошарка authHeader

Прошарка appHistory (Рисунок 5.3) потрібна для того щоб робити запис в історії життєвого циклу системи в тому разі коли це потрібно користувачу.

```
const appHistoryMiddleware = ({ dispatch, getState }) => next => action => {
  const { note } = action;
  if (note) {
    const state = getState();
    const user = sessionSelectors.getUser(state);
    const storeId = storeSelectors.getStore(state).id;
    const name = R.propOr('', 'name', user);

    const payload = {
      user: name,
      action: note,
      storeId,
    };

    dispatch(actions.addNote({ payload }));
  }
  next(action);
};
```

Рисунок 5.3 – Прошарка appHistory

RootReducer (Рисунок 5.4) в свою чергу задає кореневий вигляд Redux сховища. Також надане спеціальне налаштування для redux-persist, яке вказує на ті поля сховища, які потрібно зберігати та регідрувати в localStorage. У нашому випадку це token.

```
const tokenPersistConfig = {
  key: 'token',
  storage,
  whitelist: ['token'],
};

const rootReducer = combineReducers({
  [SESSION_ROOT]: persistReducer(tokenPersistConfig, sessionReducer),
  [HISTORY_ROOT]: historyReducer,
  [STORE_ROOT]: storeReducer,
  [ENTRIES_ROOT]: entriesReducer,
  [MENU_CONFIG_ROOT]: menuConfigReducer,
  [OPTIMIZATION_ROOT]: optimizationReducer,
});
```

Рисунок 5.4 – RootReducer

Для комфортної та швидкої роботи з redux були створенні такі допоміжні функції:

- функція створення типів запиту для їх використання з асинхронними actions (Рисунок 5.5);

```
export function createRequestTypes(base) {
  return [START, SUCCESS, ERROR].reduce((acc, type) => {
    acc[type] = `${base}_${type}`;
    return acc;
  }, {});
}
```

Рисунок 5.5 – Функція createRequestTypes

- функція для легкого створення асинхронного action creator (Рисунок 5.6). Для повноцінної роботи функції потрібно лише передати типи actions та api запит.

```
export function makeApiActionCreator(
  _apiCall,
  startActionType,
  successActionType,
  errorActionType,
) {
  return (params, note) => {
    return async (dispatch, getState) => {
      const state = getState();
      const { token } = state.session;
      if (startActionType === 'SESSION/REFRESH_USER_START' && !token) {
        dispatch(setIsRefreshed(true));
        return;
      }

      dispatch({ ...params, type: startActionType });
      try {
        const data = await _apiCall(params);
        dispatch({ ...params, data, type: successActionType, note });
        // eslint-disable-next-line consistent-return
        return data;
      } catch (error) {
        console.log(error); // eslint-disable-line no-console
        dispatch({ ...params, error, type: errorActionType });
        throw error;
      }
    };
  };
}
```

Рисунок 5.6 – Функція makeApiActionCreator

5.1.3 Інші технології

- axios. HTTP-клієнт на основі обіцянок для браузера і node.js;
- moment. Для роботи з часом та датою;
- ramda. Набір корисних функцій виконаних у функціональному стилі;
- react-excel-renderer. Для роботи з excel файлами;
- react-router-dom. Для надання додатку можливостей маршрутизації;
- react-confirm. Для підтвердження певної дії користувачем;
- react-dnd. Для реалізації техніки DND;
- react-loadable. Для асинхронного завантаження компонентів;
- recharts. Для створення різних графіків та діаграм.

5.2 Вигляд програми

Сторінки авторизації були оформлені в мінімалістичному стилі. Вони надають можливість користувачу дуже зручно зареєструватись (Рисунок 5.8) або увійти у систему (Рисунок 5.7).

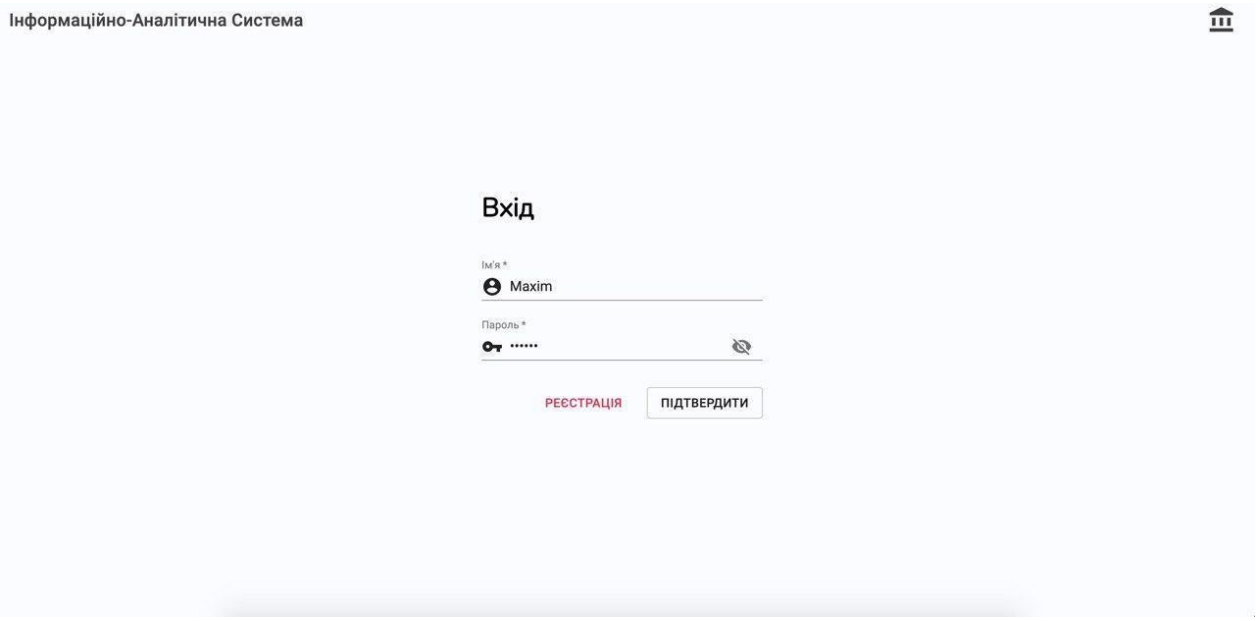

Рисунок 5.7 – Сторінка входу

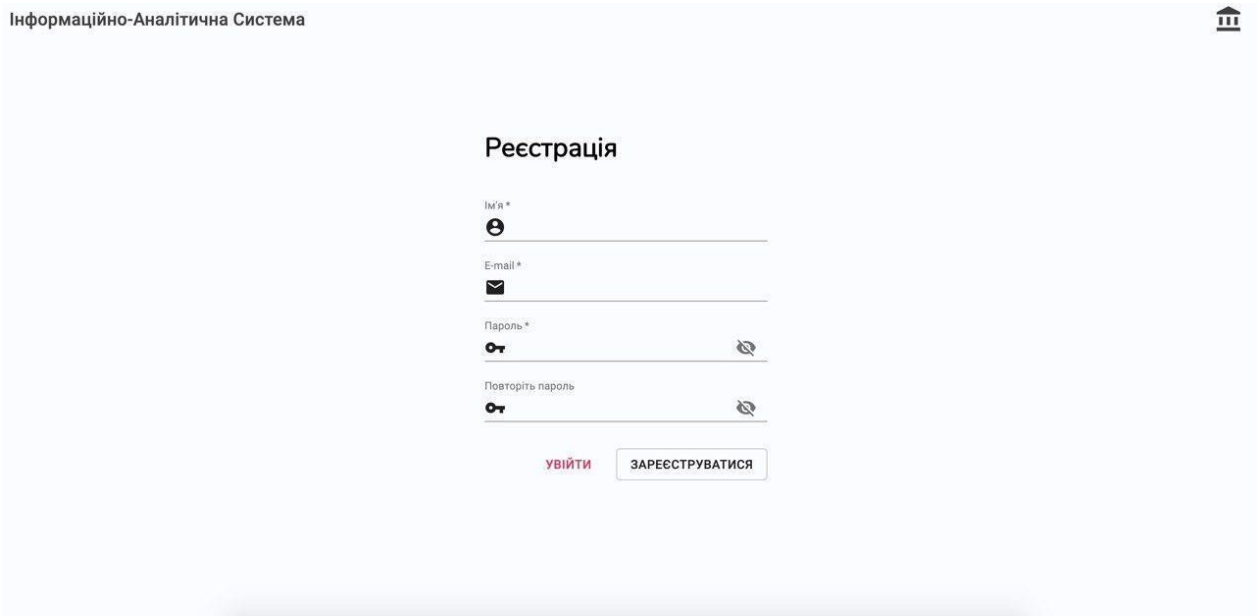

Рисунок 5.8 – Сторінка реєстрації

Сторінки вибору існуючого (Рисунок 5.9) та створення нового магазину (Рисунок 5.10) надають користувачу зручну та просту можливість оперувати базою магазинів.

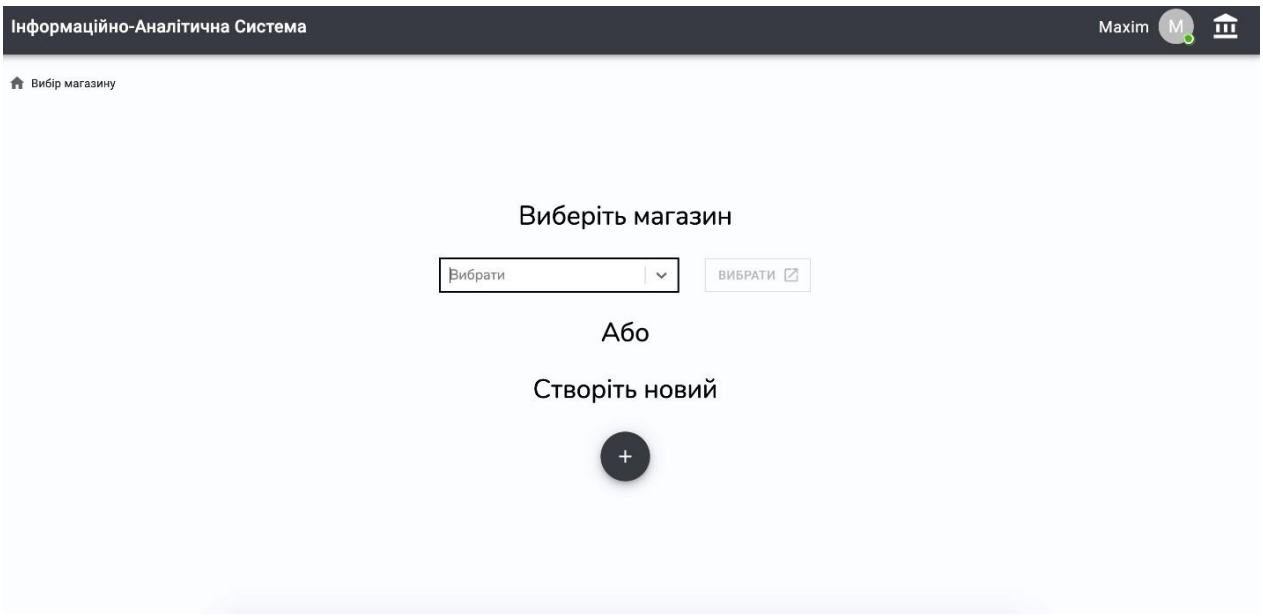

Рисунок 5.9 – Сторінка вибору магазину

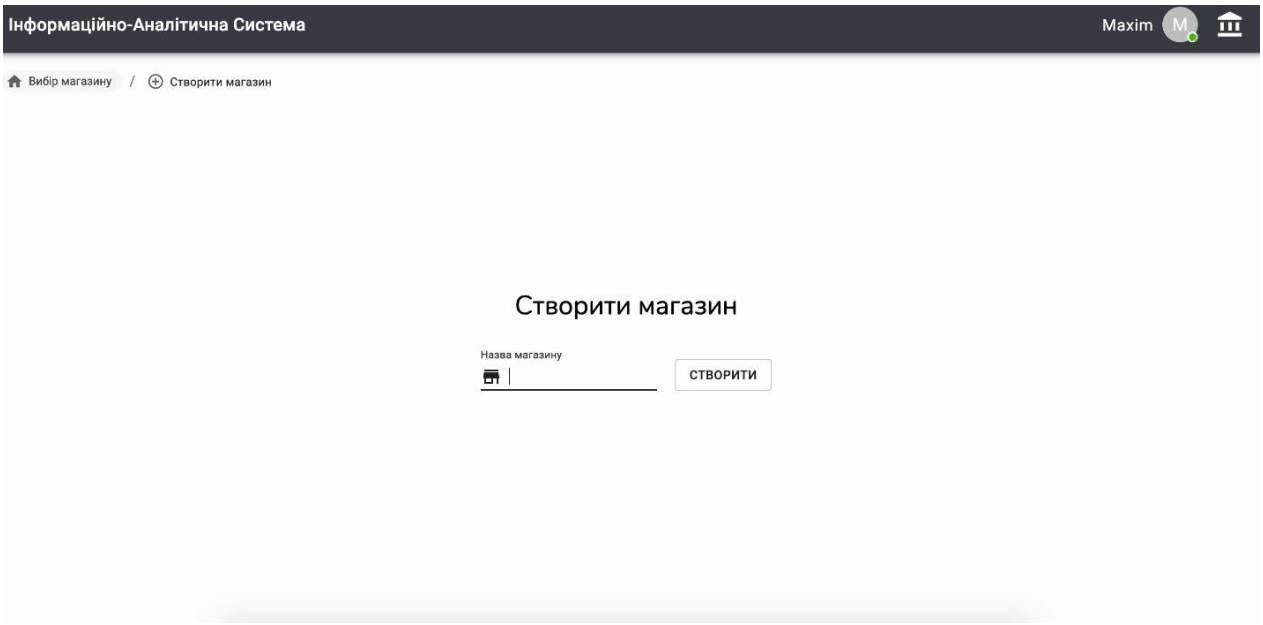

Рисунок 5.10 – Сторінка створення магазину

Головне меню магазину (Рисунок 5.11) надає користувачу можливість вибрати любий з модулів для роботи. З цього моменту, якщо користувач має необхідні права, з’являється можливість перейти в налаштування магазину. Сторінка меню також надає можливість налаштовувати вигляд за допомого техніки DND.

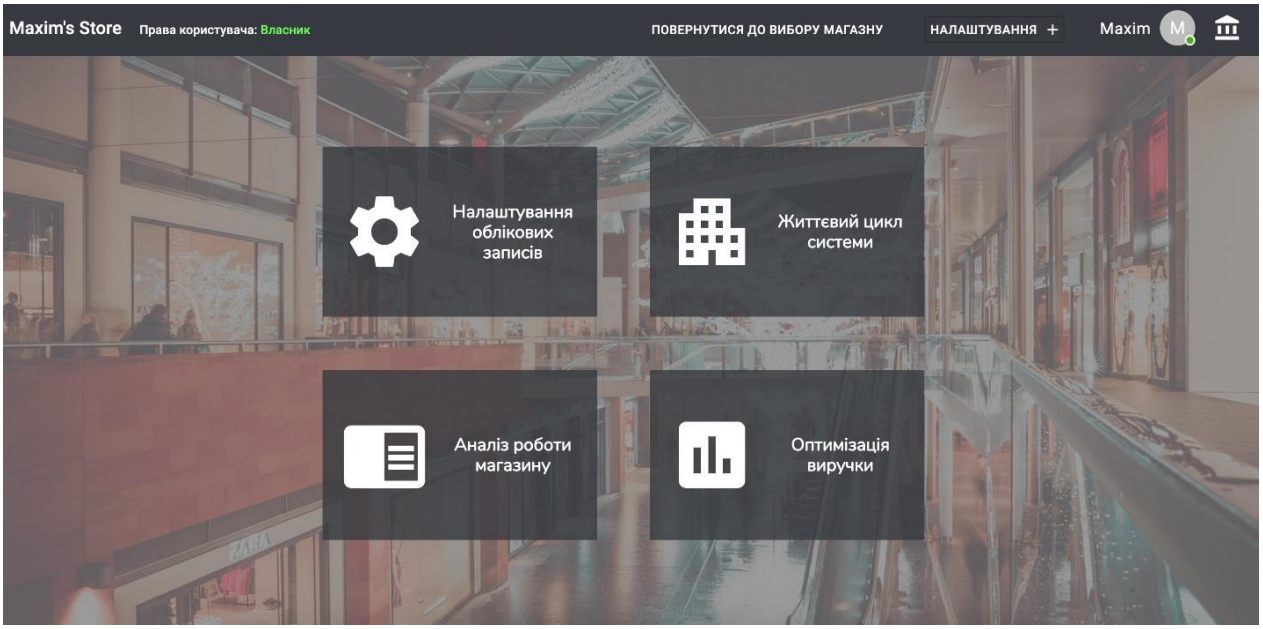

Рисунок 5.11 – Сторінка меню магазина

Модальне вікно «Налаштування», виконане з використанням табів надає такий функціонал:

- таб «Додати» (Рисунок 5.12). Надає можливість додати користувача до магазину;

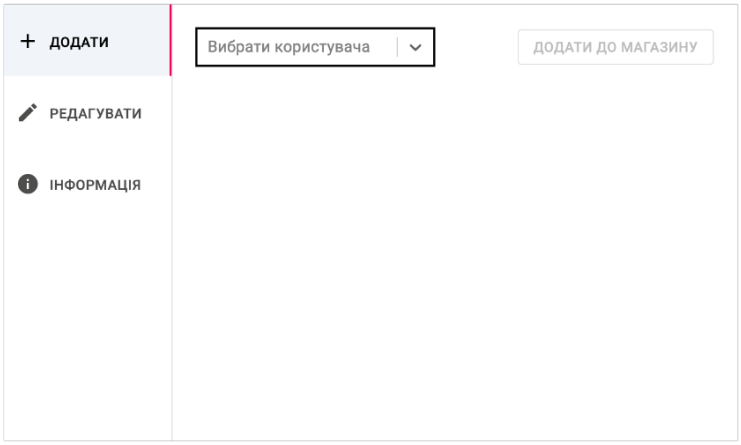

Рисунок 5.12 – Таб «Додати»

- таб «Редагувати» (Рисунок 5.13). Надає можливість оперувати правами та видаляти користувачів із магазину;

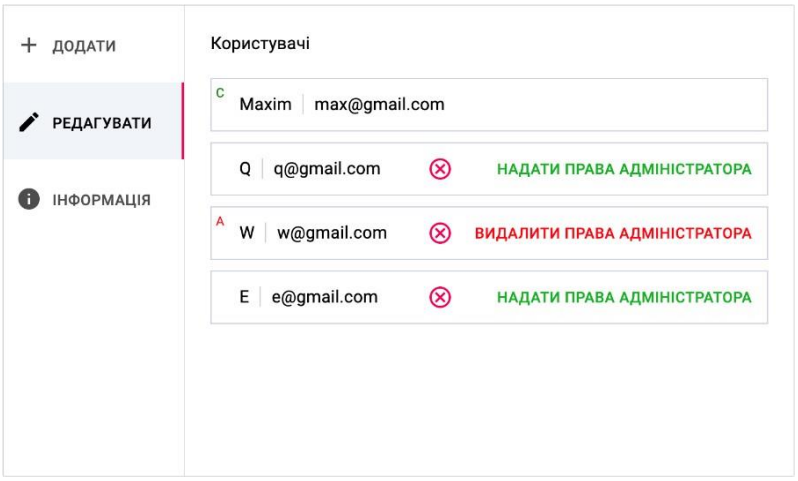

Рисунок 5.13 – Таб «Редагувати»

- таб «Інформація» (Рисунок 5.14). Доступ до нього має тільки власник магазину. Надає можливість змінити ім’я та видалити магазин.

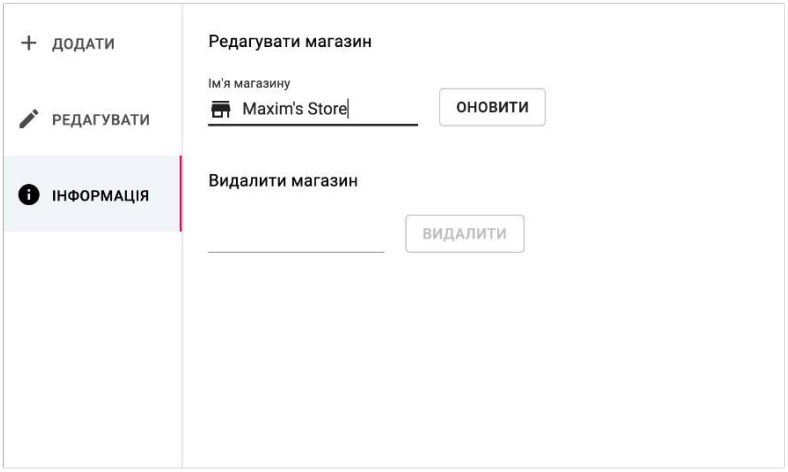

Рисунок 5.14 – Таб «Інформація»

Сторінка налаштування облікових засобів (Рисунок 5.15) надає можливість користувачу оперувати обліковими засобами. Ця сторінка реалізована у вигляді табів. Де на кожному табі можна працювати з відповідною сутністю, всього таких сутностей шість:

- працівники;
- посади;
- товари;
- продажі;
- постачальники;
- накладні.

При роботі з кожною сутністю користувач має можливості:

- переглянути перелік сутностей;
- профільтрувати сутності;
- відсортувати сутності;
- виконати імпорт сутностей;
- видалити сутність;

- редагувати сутність;
- створити нову сутність (Рисунок 5.16).

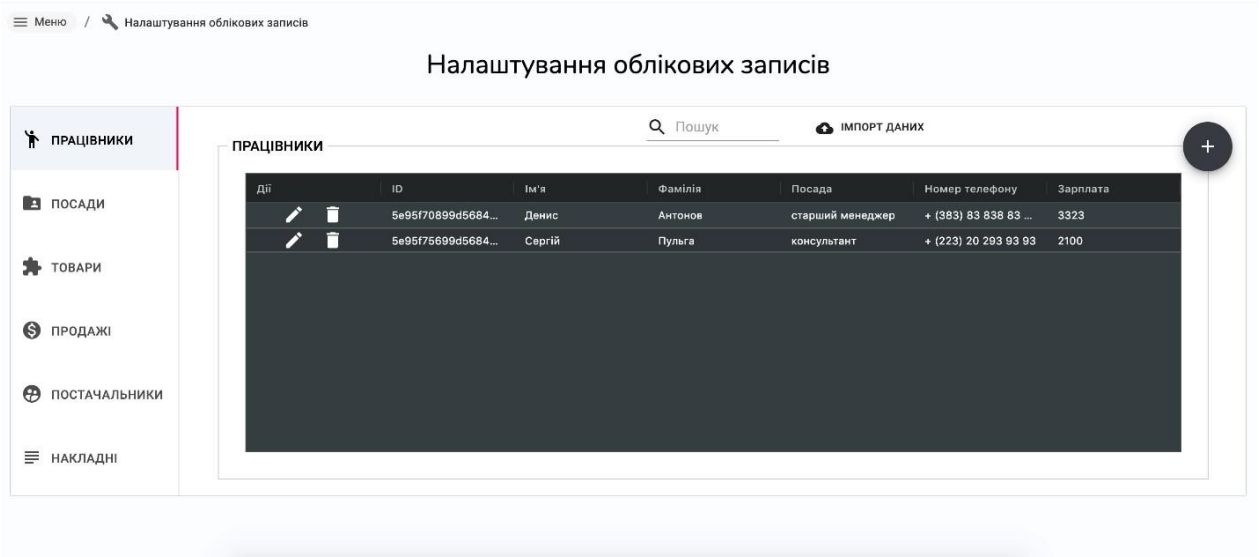

Рисунок 5.15 – Сторінка «Налаштування облікових записів»

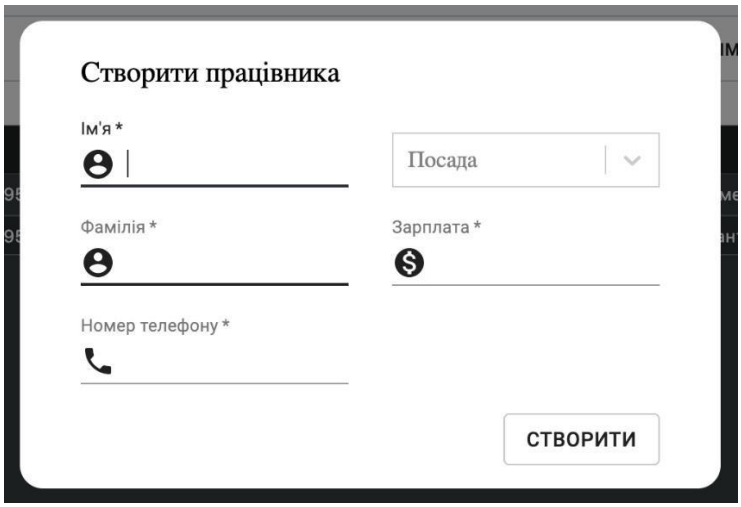

Рисунок 5.16 – Модальне вікно створення нової сутності

Сторінка життєвого циклу системи (Рисунок 5.17) виконано у вигляді табів. Перший таб надає користувачу можливість переглядати історію роботи магазина. Другий (Рисунок 5.18) надає можливість встановити фільтр історії за користувачем.

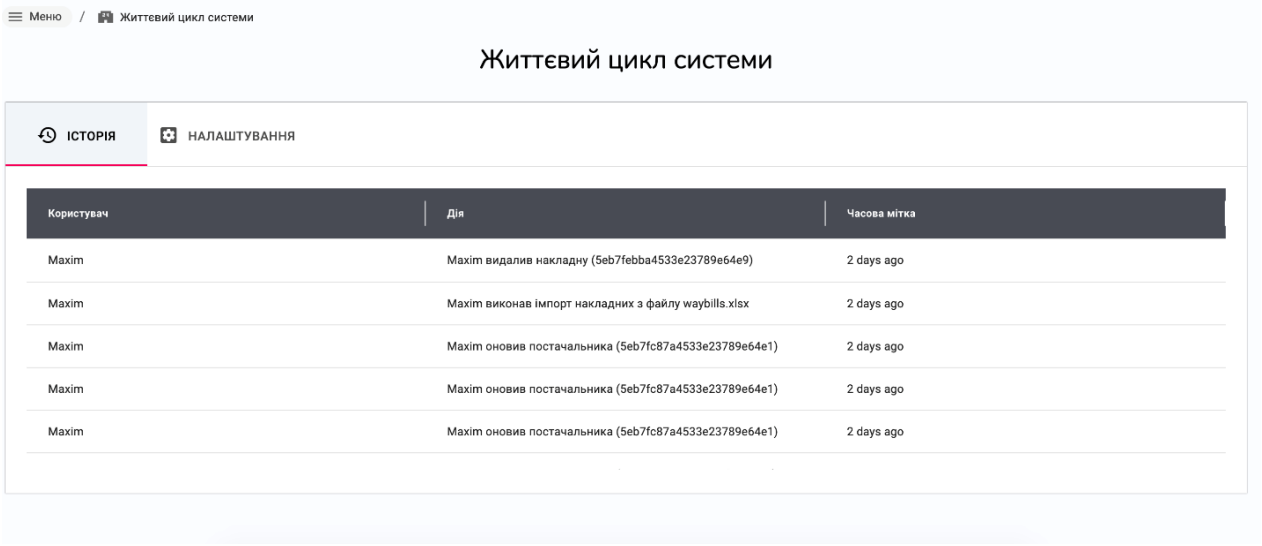

Рисунок 5.17 – Таб «Історія»

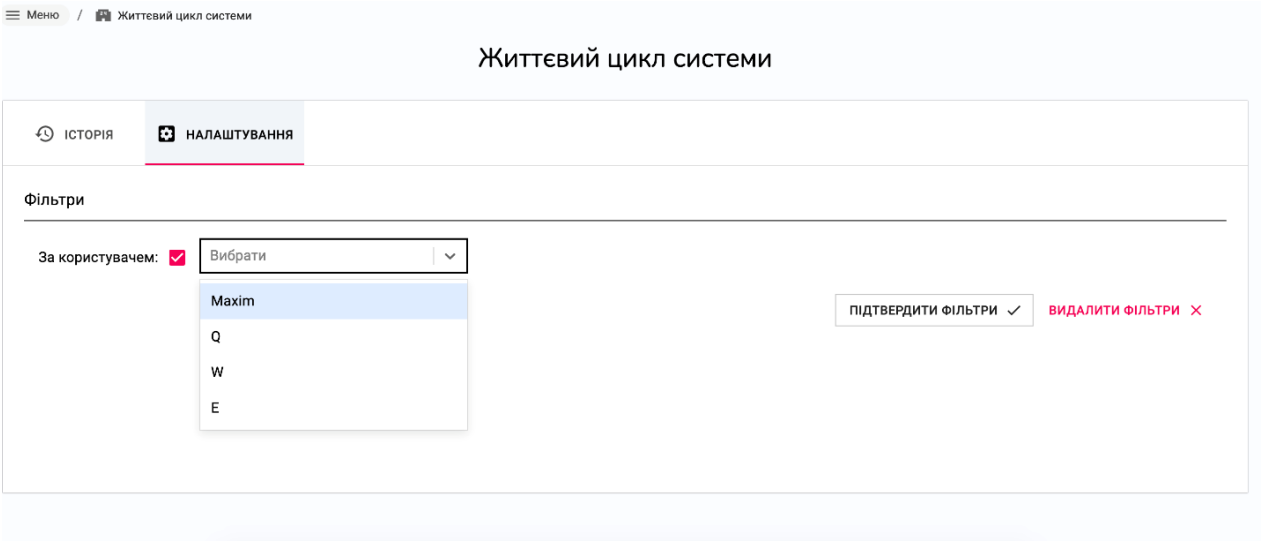

Рисунок 5.18 – Таб «Налаштування»

Сторінка аналізу роботи магазина (Рисунок 5.19) виконана у вигляді розширюваних панелей та надає користувачу можливість відстежувати попит товарів (Рисунок 5.20), відстежувати співробітництво з постачальниками (Рисунок 5.21) та відстежувати роботу працівників (Рисунок 5.22). Ця сторінка також надає можливість налаштовувати вигляд за допомого техніки DND.

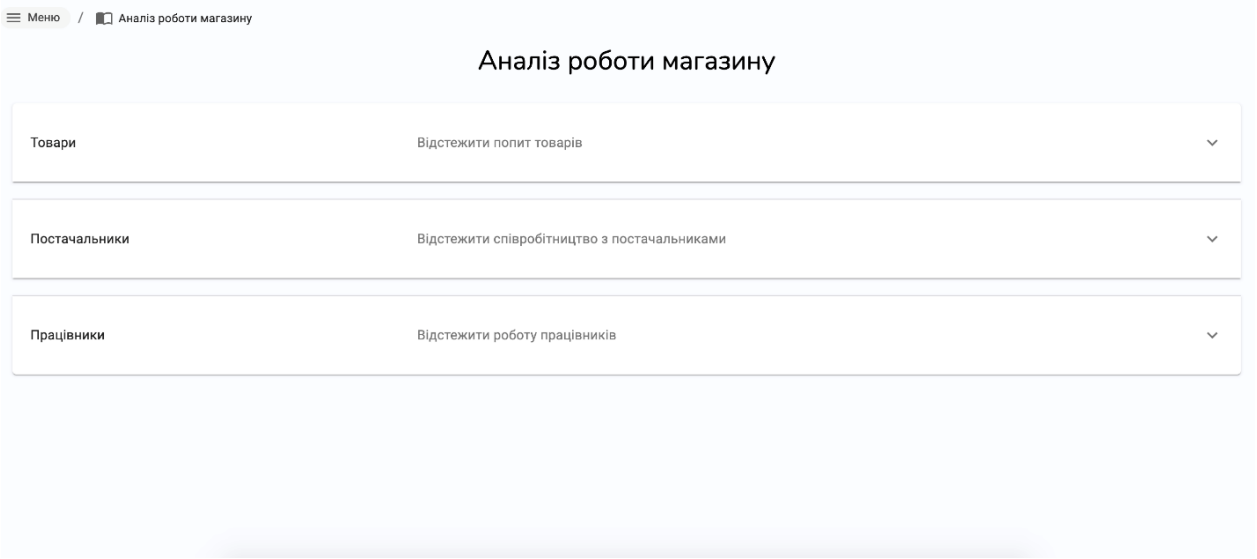

Рисунок 5.19 – Сторінка «Аналіз роботи магазину»

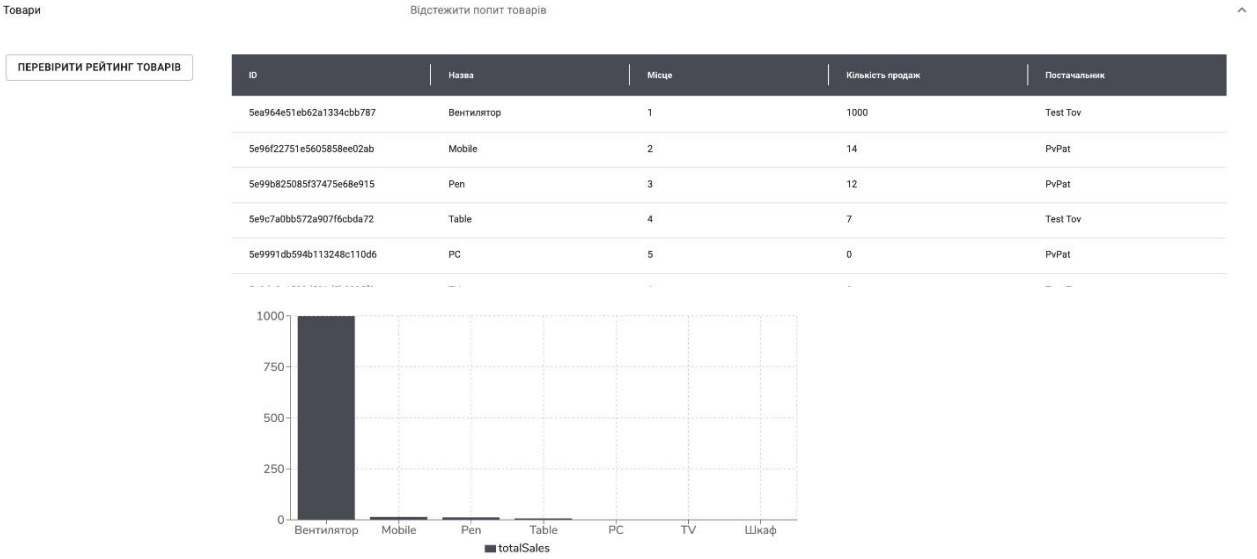

Рисунок 5.20 – Розширювана панель «Відстежити попит товарів»

Рисунок 5.21 – Розширювана панель «Відстежити співробітництво з постачальниками»

Рисунок 5.22 – Розширювана панель «Відстежити роботу працівників»

Сторінка оптимізації виручки виконана у вигляді степера.

- крок «Вибір товару» (Рисунок 5.23). Вибір продукту та задання необхідної зміни ціни та постійних витрат;
- крок «Прибуток по товару» (Рисунок 5.24). Виведення таблиці обрахованих значень для збереження рівня прибутку;
- крок «Крива збереження рівня прибутку» (Рисунок 5.25). Виведення таблиці обрахованих значень для збереження рівня прибутки в діапазоні [-25% ; 25%] та побудова кривої збереження прибутку.

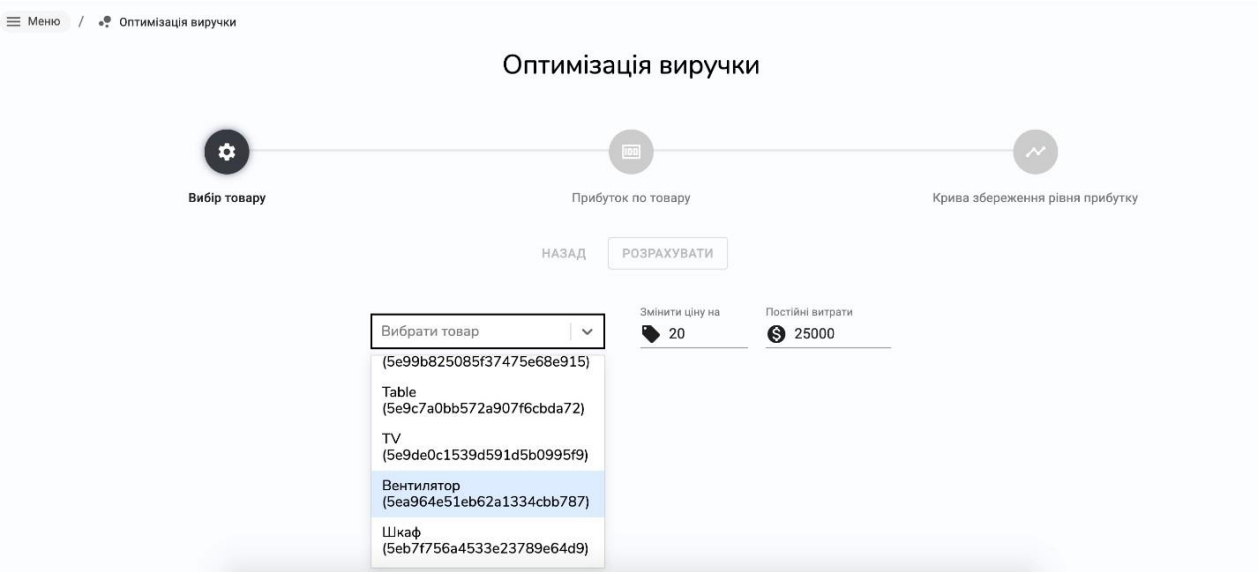

Рисунок 5.23 – Крок «Вибір товару»

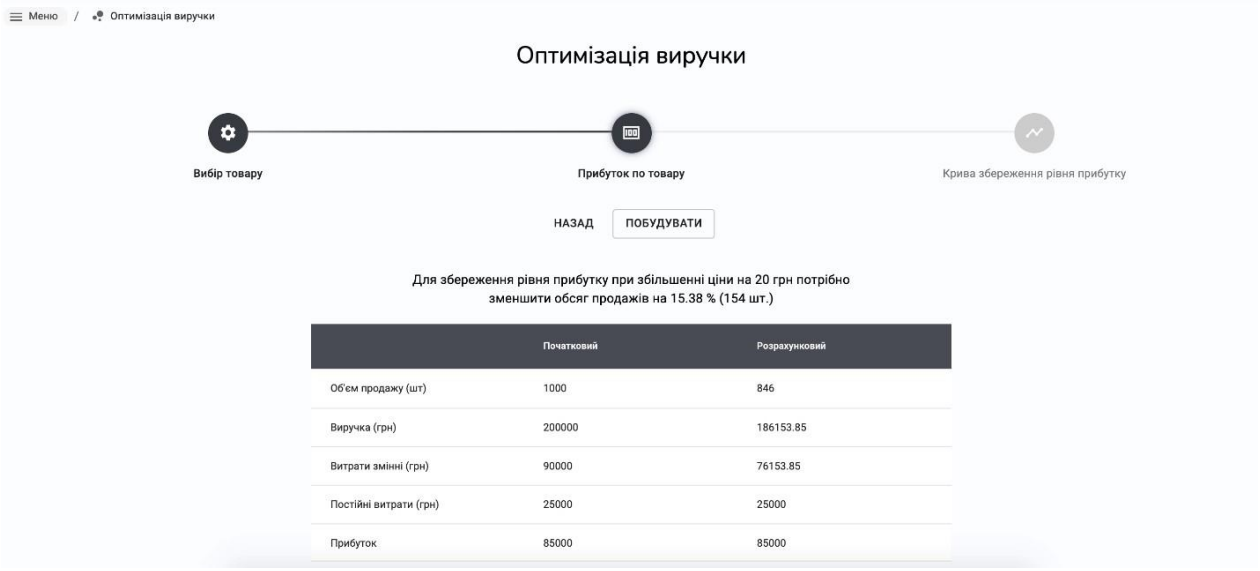

Рисунок 5.24 – Крок «Прибуток по товару»

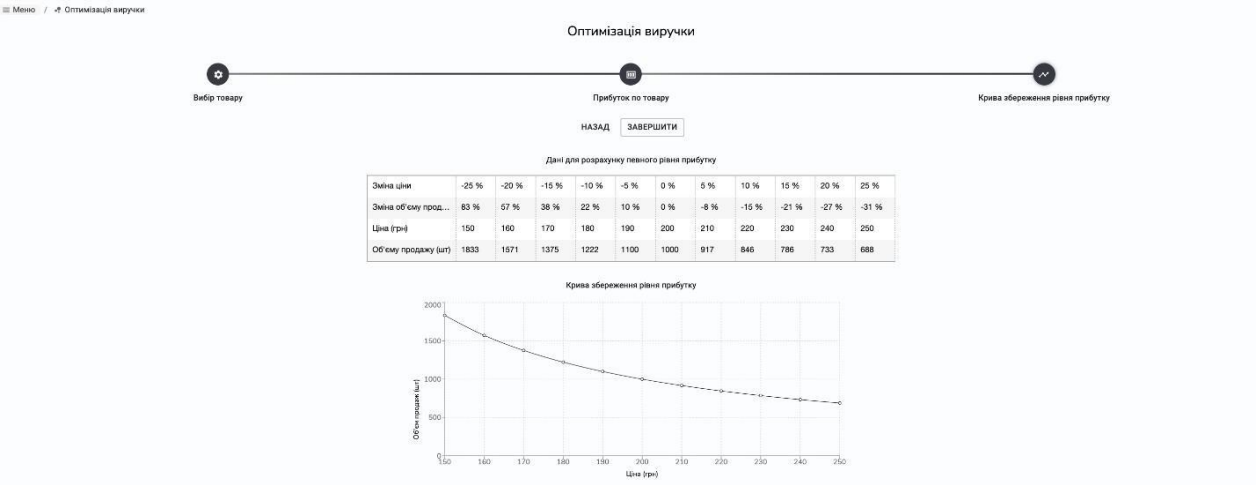

Рисунок 5.25 – Крок «Крива збереження рівня прибутку»

У разі коли користувач намагатиметься перейти в магазин до якого він не має прав доступу - буде відбуватися переправлення до сторінки заборони.

Для виходу із системи користувачу необхідно натиснути на свій аватар, після чого відкриється контекстне меню, де буде пункт «Вийти з системи».

В результаті був побудований приємний, сучасний та зручний інтерфейс, який задовольняє усім вимогам та критеріям, які були представлені до ІАС.

6 РОЗРОБКА МЕХАНІЗМІВ ТА АЛГОРИТМІВ ОПТИМІЗАЦІЇ ТА АНАЛІЗУ В ІАС

6.1 Аналіз роботи магазину

Створена ІАС має можливість проводити певний аналіз облікових записів. Із основних можливостей:

- відстежити попит товарів;
- відстежити співробітництво з постачальниками;
- відстежити роботу працівників.

6.1.1 Відстежити попит товарів

Користувач має можливість отримати рейтинг топ10 найбільш продаваних товарів. Результат виводиться в таблицю та діаграму. Є можливість побачити рейтинг, кількість продаж та постачальника кожного із товарів. Алгоритм (Рисунок 6.1) був реалізований таким чином:

- береться масив всіх товарів. Для кожного з них виконуємо наступні пункти:
    - знаходяться всі здійснені продажі;
    - знаходиться сума продажів;
    - знаходиться постачальник;
    - оформлюється об’єкт зі всіма актуальними даними;
    - додається об’єкт до масиву;
- сортується масив отриманих товарів за кількістю продаж;
- проставляється для кожного елемента відповідне місце у рейтингу;
- відправляється результуючий масив користувачу;

```
try {
  const resultList = [];
  const products = await Product.find();

  for (const product of products) {
    const productId = product._id;
    const productSales = await Sale.find({ productId });
    const totalSales = productSales.reduce(
      (acc, item) => acc + item.count,
      0,
    );

    const supplierId = product.supplierId;

    const { name: supplier } = await Supplier.findById({ _id: supplierId });

    const productInfo = {
      id: productId,
      name: product.name,
      totalSales,
      supplier,
    };
    resultList.push(productInfo);
  }
  const listOfProductsInOrderOfSale = resultList
    .sort((a, b) => b.totalSales - a.totalSales)
    .map((item, idx) => ({ ...item, order: 1 + idx }));

  sendResponse({
    listOfProductsInOrderOfSale,
  });
} catch (error) {
  sendError(error);
}
```

Рисунок 6.1 – Алгоритм зіставлення рейтингу товарів

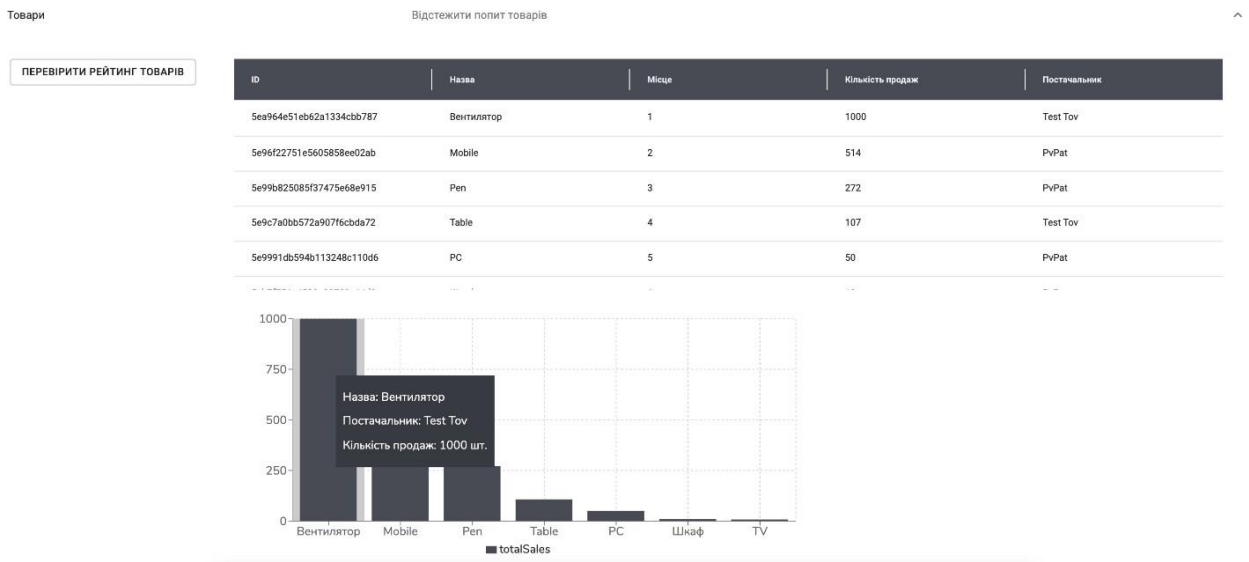

Рисунок 6.2 – Результат роботи алгоритму

6.1.2 Відстежити співробітництво з постачальниками

Користувач має можливість дослідити певну роботу з постачальниками, а саме відстежити:

- загальну вартість закуплених товарів;
- загальну вартість проданих товарів;
- коефіцієнт корисності.

Алгоритм був реалізований таким чином (Рисунок 6.3):

- знаходяться всі товари, які були закуплені в досліджуваного постачальника;
- знаходиться загальна оптова ціна всіх товарів від постачальника;
- для кожного товару повторюються наступні пункти:
    - знаходяться всі продажі конкретного товару;
    - знаходяться загальні кошти, отримані від всіх продаж даного товару;
    - сортуються ці кошти;
- підраховується коефіцієнт ефективності, який дорівнює відношенню загальних коштів, отриманих від продажів до загальної оптової ціни всіх товарів;
- відправляється результат користувачу;

```
try {
  const products = await Product.find({ supplierId });

  const purchasedProductsCost = products.reduce((acc, item) => {
    const totalProductPrice = item.count * item.wholesalePrice;
    return acc + totalProductPrice;
  }, 0);

  let saleProductsCost = 0;
  for (const product of products) {
    const sales = await Sale.find({ productId: product._id });
    if (sales.length > 0) {
      sales.forEach(sale => {
        saleProductsCost += sale.count * sale.price;
      });
    }
  }

  const supplierEfficiency = (
    saleProductsCost / purchasedProductsCost
  ).toFixed(2);

  sendResponse({
    purchasedProductsCost,
    saleProductsCost,
    supplierEfficiency,
  });
} catch (error) {
  sendError(error);
}
```

Рисунок 6.3 – Алгоритм аналізу постачальників

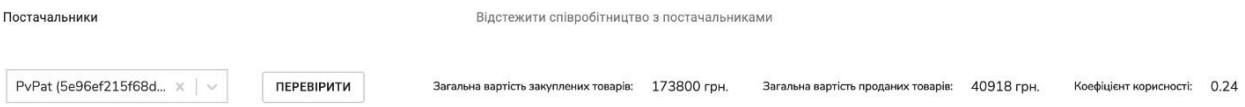

Рисунок 6.4 – Результат роботи алгоритму

6.1.3 Відстежити роботу працівників

Користувач має можливість відстежити роботу конкретного працівника, а саме:

- побачити кількість проданих товарів;
- побачити виручку;

Алгоритм був реалізований таким чином (Рисунок 6.5):

- знаходяться всі продажі даного працівника;
- підраховується загальна кількість продажів;
- підраховується загальна сума коштів, отриманих зі всіх продажів;
- для кожного проданого працівником продукту знаходиться загальна оптова ціна, яка підсумовується;
- вираховується виручка використовуючи отримані дані;
- відправляється результат користувачу.

```
try {
  const employeeSales = await Sale.find({ employeeId });

  const salesCount = employeeSales.reduce((acc, item) => acc + item.count, 0);

  const productsPrice = employeeSales.reduce(
    (acc, item) => acc + item.price,
    0,
  );
  const productIds = employeeSales.map(({ productId }) => productId);

  let productsWholesalePrice = 0;
  for (const id of productIds) {
    const product = await Product.findOne({ _id: id });
    const productWholesalePrice = product ? product.wholesalePrice : 0;
    productsWholesalePrice += productWholesalePrice;
  }
  const earning = productsPrice - productsWholesalePrice;
  sendResponse({ earning, salesCount });
} catch (error) {
  sendError(error);
}
```

Рисунок 6.5 – Алгоритм аналізу працівників

Рисунок 6.6 – Результат роботи алгоритму

6.2 Максимізація виручки та прибутку

6.2.1 Основні поняття

Як відомо, зміна ціни продукту або послуги тягне за собою зміну обсягу продажів. При цьому по кожному окремому товару ця залежність може бути різною. Для її оцінки використовується коефіцієнт еластичності попиту за ціною (E), який показує, наскільки відсотків зміниться обсяг продажів (q) при зміні ціни (p) на 1%.

$$E = (\Delta q/q)/(\Delta p/p) = \frac{\text{процентна зміна об'єму продаж}}{\text{процентна зміна ціни}}. \quad (6.1)$$

де Δ - цілковита зміна.

Залежність обсягу попиту від ціни відображає крива попиту. Нахил між будь-якими двома точками на ній і визначає еластичність попиту при даному рівні цін. Знаючи форму такої кривої, можна розрахувати ціни, при яких досягається максимум виручки і прибутку [12].

Максимальна виручка буде при такій ціні, коли відсоткова зміна обсягу продажів дорівнює процентній зміні ціни (с зворотним знаком).

Умова досягнення максимуму виручки:

$$E = 1 \text{ або } \frac{\Delta q}{q} = -\frac{\Delta p}{p}. \quad (6.2)$$

Якщо при поточній ціні еластичність менше 1, то для збільшення виручки вигідно ціну підвищувати і, навпаки, знижувати, якщо еластичність більше 1.

Виручка хоча і вважається одним з найважливіших параметрів діяльності компанії, проте більш значущим є визначення рівня цін, при яких досягається максимум прибутку.

Максимум прибутку досягається при такій ціні, коли відсоткова зміна обсягу продажів дорівнює процентній зміні ціни, помноженому на коефіцієнт p / (p - c).

Умови досягнення максимум прибутку:

$$E_{p-c} = \frac{p-c}{p} \times E = 1 \text{ або } \frac{\Delta q}{q} = -\frac{\Delta p}{p} \times \frac{p}{p-c}, \quad (6.3)$$

де з - змінні витрати на одиницю продукції;

p - ціна;

q - обсяг продажів;

E - коефіцієнт еластичності.

Якщо при поточній ціні еластичність менше p / (p - c), то для збільшення виручки вигідно ціну підвищувати і, навпаки, знижувати, якщо еластичність більше p / (p - c).

Отримані вище висновки зводяться в таблицю (Таблиця 6.1).

Таблиця 6.1 – Рекомендації зі зміни ціни для максимізації прибутку

	$E < 1$	$E = 1$	$1 < E < \frac{p}{p-c}$	$E = \frac{p}{p-c}$	$E > \frac{p}{p-c}$
Для збільшення виручки	Збільшувати	Зберегати	Зменшувати		
Для збільшення прибутку	Збільшувати			Зберегати	Зменшувати

Максимум прибутку і максимум виручки досягаються при різних значеннях ціни. А саме: максимум прибутку завжди досягається при ціні більшій, ніж ціна, за якої досягається максимум виручки [12].

Вище були описані умови для визначення оптимальної ціни для максимізації виручки і прибутку на підставі даних кривої попиту. Однак на практиці точно встановити криву попиту дуже складно.

Можна виділити кілька методів визначення цінової еластичності:

- обробка статистичної інформації про продаж товарів на різних ринках або на одному ринку, але в різні моменти часу і за різними цінами, але для застосування даного методу необхідна хороша база даних, приведена до однакових умов щодо ринкових сегментів, типів споживачів, місць продажу, які впливають на цінову еластичність;
- постановка цінових експериментів. Ціни можна змінювати протягом певного часу в кількох магазинах або призначати різні ціни на однакові товари в декількох магазинах, але істотно важливим при проведенні цінових експериментів є збереження незмінними всіх інших факторів. Подібний експеримент під силу далеко не всім компаніям, оскільки його проведення вимагає значних коштів і, крім того, як зазначалося вище, на продажу крім цін впливають і інші фактори, які не піддаються контролю;
- проведення опитування споживачів з метою з'ясування, за яких цінах вони готові купувати певні товари, але зазвичай спостерігається істотна відмінність між висловлюваннями споживачів і їх реальною поведінкою на ринку;
- побудова економіко-математичних моделей, що моделюють поведінку груп споживачів, але моделювання поведінки людини, переклад на мову формульних залежностей багатьох психологічних і соціальних факторів з виробленням конкретних кількісних рекомендацій, що цікавлять практиків, - важко вирішуване завдання. Такі моделі швидше представляють теоретичний інтерес і в практиці не використовуються [3].

На практиці важко домогтися оцінок еластичності, достатньо стабільних і надійних, для визначення на їх основі оптимальних цін. За оцінками деяких експертів, точність визначення цінової еластичності становить ± 25%. Такий значний розкид може істотно вплинути на кінцевий результат при вирішенні практичних завдань.

Тому треба поглянути на проблему з іншого боку. Забудемо про питання «Яка еластичність попиту на товар?». Поставимо інше питання: «Яка мінімальна еластичність попиту потрібно для того, щоб не зменшився рівень прибутку при зміні ціни?». Розглянемо рекомендовані для застосування методи в співвідношенні з потенційними бар'єрами в їх реалізації.

Для опису умови скористаємося наступними позначеннями:

p - ціна продажу одиниці продукції;

Δp - зміна ціни (при зниженні ціни Δp <0);

c - змінні витрати на одиницю продукції;

q - обсяг продажів в натуральному вираженні; Δq - зміна обсягу продажів.

Умова не зменшення рівня прибутку виглядає наступним чином:

$$\frac{\Delta q}{q} \geqslant -\frac{\Delta p}{p} \times \frac{p}{p - c + \Delta p}$$

або

$$\%\ \text{зм. об'єму} \geqslant -\%\ \text{зм. ціни} \times \frac{\text{ціна}}{\text{ціна} - \text{тимч. затрати} + \text{зм. ціни}}.$$

(6.4)

Тобто для збереження рівня прибутку при зміні ціни процентна зміна обсягу продажів має бути більше, ніж процентна зміна ціни (з протилежним знаком), помножене на множник $\frac{p}{p - c + \Delta p}$.

Зміна ціни може бути частиною маркетингового плану, який включає в себе і зміна витрат. Приклад за визначенням максимуму виручки і прибутку:

Відома якась функція попиту (Рисунок 6.7).

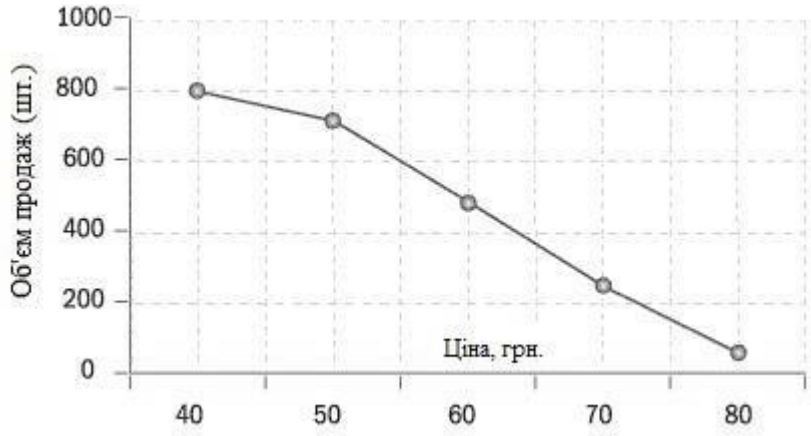

Рисунок 6.7 – Функція попиту

Змінні витрати на одиницю продукції становлять 35 гривень за шт. Загальні постійні витрати складають 5000 гривень.

Розраховуються суми виручки і прибутку для різних рівнів цін (Таблиця 6.2).

Таблиця 6.2 – Суми виручки і прибутку для різних рівнів цін

Ціна (від - до)	40–50	50–60	60–70	70–80
Еластичність попиту (Е)	0,73	1,9	4,1	10
Коефіцієнт р/(р - с)	4,5	2,75	2,17	1,88

Середня еластичність попиту в інтервалі цін:

$$E = -\frac{\text{зміна об'єму}}{\text{середній об'єм}} \Big/ \frac{\text{зміна ціни}}{\text{середня ціна}} = -\frac{(q2 - q1)}{(q2 + q1)/2} \Big/ \frac{(p2 - p1)}{(p2 + p1)/2} \quad (6.5)$$

В інтервалі цін від 40 до 50 середня еластичність попиту (0,73) менше 1 і менше коефіцієнта p / (p - c) - (4,50). Тому при збільшенні ціни в цьому діапазоні ростуть і виручка, і прибуток.

В інтервалі від 50 до 60 середня еластичність (1,90) більше 1, але менше коефіцієнта p / (p - c) - (2,75). Тому при збільшенні ціни в цьому діапазоні виручка починає знижуватися, але прибуток продовжує рости (Рисунок 6.8).

У наступних інтервалах середня еластичність більше і 1, і коефіцієнта p / (p - c). Тому і виручка, і прибуток сильно знижуються.

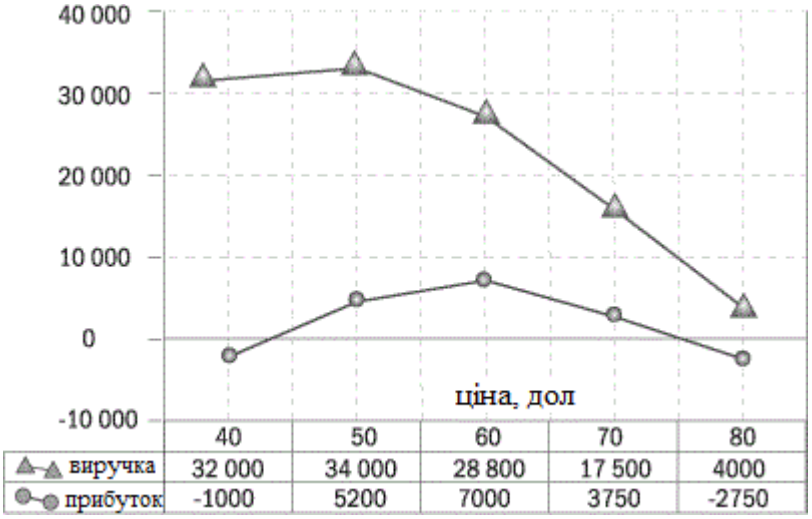

Рисунок 6.8 – Максимум прибутку і максимум виручки досягаються при різних цінах

Ціна може збільшуватися в зв'язку з поліпшеннями якості продукції. Зниження ціни може бути викликано прагненням вивести на ринок товар з більш низькими змінними витратами.

Для загального випадку, коли при зміні ціни змінюються і змінні, і постійні витрати, наведемо формулу залежності для збереження рівня прибутку:

$$\frac{\Delta q}{q} \geqslant -\frac{\Delta p - \Delta c}{(p - c) + (\Delta p - \Delta c)} + \frac{\Delta F}{((p - c) + (\Delta p - \Delta c)) \times q}. \quad (6.6)$$

де ΔF - зміна загального обсягу постійних витрат.

Крім того, деякі рішення по ціноутворенню можуть вимагати зміни і постійних витрат. Слід зауважити, що якщо немає зміни ні змінних, ні постійних витрат, то формула трансформується в первісну:

$$\frac{\Delta q}{q} \geqslant -\frac{\Delta p}{p} \times \frac{p}{p - c + \Delta p}. \quad (6.7)$$

Незважаючи на наявність загальної формули, яка може бути застосована в більшості ситуацій, на практиці часто вистачає простої формули для визначення необхідного зміни обсягу продажів і збереження рівня прибутку.

Компанія планує зниження ціни на один з продуктів на 5% (з 200 грн. за одиницю до 190 грн.). Це та інші початкові дані вносимо у таблицю (Таблиця 6.3).

Таблиця 6.3 – Початкові дані

p	Ціна	200	грн.
c	Змінні витрати (на од.)	90	грн.
.	Постійні витрати. Всього:	25 000	грн.
q	Поточний об'єм продаж	300	шт.
Δp	Хочемо змінити ціну на:	–10	грн.

Потрібно оцінити, наскільки відсотків повинні збільшитися продажу цього продукту для збереження рівня прибутку. За формулою знаходимо необхідне збільшення обсягу:

$$\frac{\Delta q}{q} \geqslant -\frac{\Delta p}{p} \times \frac{p}{p - c + \Delta p} = -(-5\%) \times \frac{200}{200 - 90 + (-5\% \times 200)} = 10\% \quad (6.8)$$

Для збереження рівня прибутку при зниженні ціни на 5% треба збільшити обсяг продажів на 10%, що в натуральному вираженні має скласти 330 шт.

Якщо за оцінками компанії після зниження ціни обсяг продажів збільшиться більш ніж на 10%, то компанії вигідно це рішення. Якщо ж збільшення буде менше 10%, то знижувати ціну не слід.

Перевіримо отримані результати прямим розрахунком прибутку по продукту (Таблиця 6.4). Як бачимо, в початковому варіанті (при обсязі продажів 300 шт.) І розрахунковому після зміни ціни (при обсязі продажів 330 шт.) Величина прибутку зберігається. Якщо обсяг продажів складе більше розрахункового (наприклад, 370 шт.), То прибуток збільшиться. Якщо ж він збільшиться недостатньо (310 шт.), Відбудеться зменшення прибутку.

Таблиця 6.4 – Розрахунок прибутку по продукту

	Початковий	Розрахунковий	Варіант 1	Варіант 2
Об'єм продаж (шт.)	300	330	370	310
Виручка (грн.)	60 000	62 700	70 300	58 900
Змінні витрати (грн.)	27 000	29 700	33 300	27 900
Постійні витрати (грн.)	25 000	25 000	25 000	25 000
Прибуток (грн.)	8000	8000	12 000	6000

За умови збереження рівня прибутку також можна розглянути діапазон зміни ціни, тобто провести аналіз беззбиткових продажів для декількох змін ціни одночасно (Таблиця 6.5).

Таблиця 6.5 – Дані для розрахунку певного рівня прибутку

Зміна ціни	–20%	–15%	–10%	–5%	0%	5%	10%	15%	20%
Зміна об'єму продаж	57%	38%	22%	10%	0%	–8%	–15%	–21%	–27%
Ціна (грн.)	160	170	180	190	200	210	220	230	240
Об'єм продаж (шт.)	1 571	1 375	1 222	1 100	1000	917	846	786	733

Який зручно представити графічно (Рисунок 6.9).

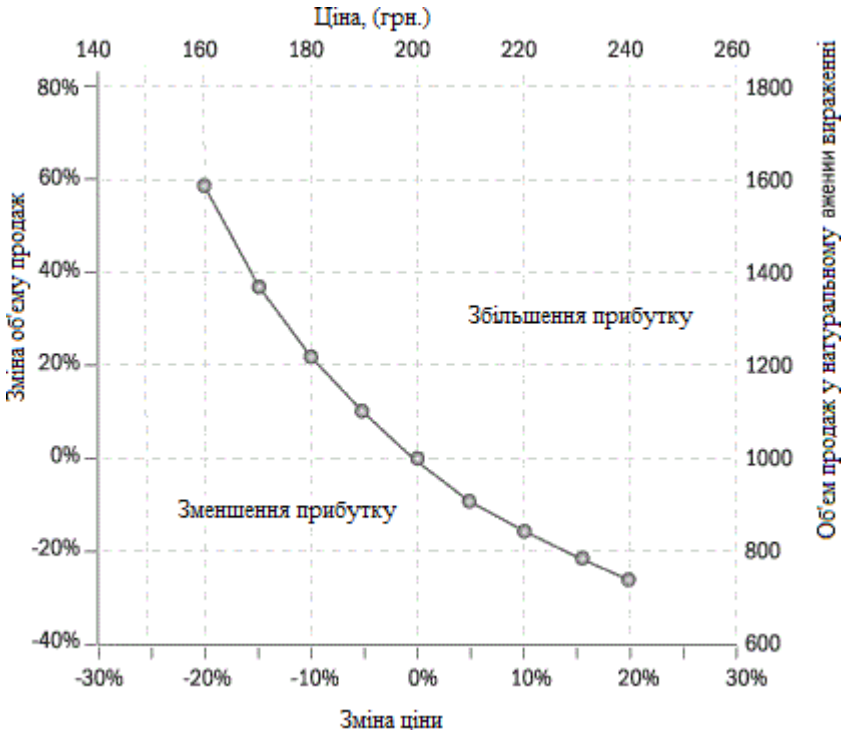

Рисунок 6.9 – Крива збереження рівня прибутку

Кожна точка на ній представляє обсяг продажів, необхідний для досягнення такої ж прибутку, яка була до зміни ціни. Крива збереження рівня прибутку - простий, але досить потужний інструмент для узагальнення та оцінки динаміки подальшої прибутку після зміни ціни. Можна розглянути взаємне розташування кривої попиту і кривої збереження прибутку. Якщо попит більш еластичний, то зниження ціни по відношенню до базового рівня збільшує прибуток (точка зміщується вище кривої збереження прибутку, що означає прибутковість), і навпаки, підвищення ціни веде до зниження прибутку (Рисунок 6.10).

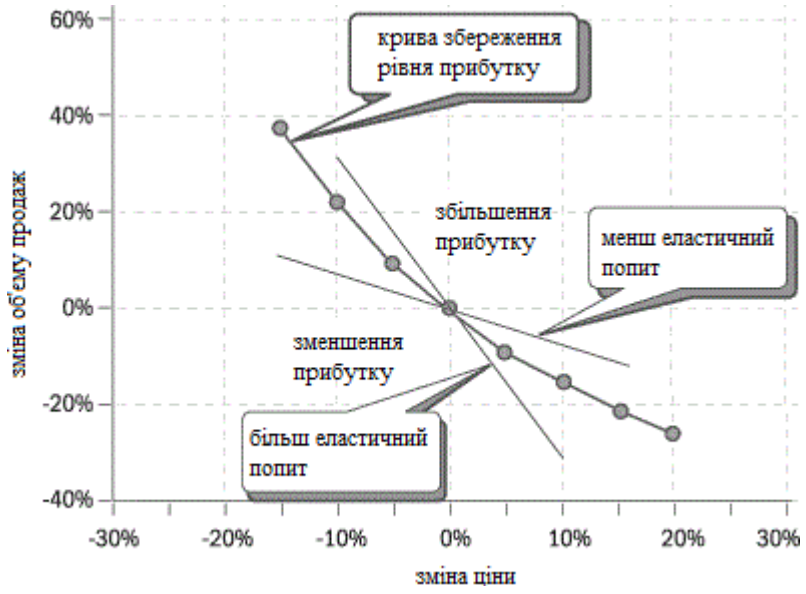

Рисунок 6.10 – Взаємне розташування кривої попиту і кривої збереження прибутку

Якщо ж попит менш еластичний, то підвищення ціни по відношенню до базового рівня збільшує прибуток (точка зміщується правіше кривої збереження прибутку, що означає прибутковість), а зниження ціни знижує прибуток.

Хоча далеко не всі менеджери знають вигляд кривої попиту на товар, але багато хто з них можуть оцінити, як змінюється обсяг продажів, що дає їм

можливість впевнено приймати рішення про зміну ціни. При цьому для побудови кривої збереження прибутку і оцінки необхідної зміни обсягу продажів використовуються тільки дані управлінського обліку про структуру витрат компанії.

6.2.2 Реалізація в IAC

В даній IAC управління магазином користувачу надається можливість змінювати ціну на товар і відстежувати динаміку зміни необхідного попиту, розраховувати дані для розрахунку певного рівня прибутку та будувати на їх основі криву збереження рівня прибутку для оцінки динаміки зміні прибутку при зміні ціни відповідного товару.

Реалізація цього процесу була розбита на 3 етапи:

- на першому етапі користувач проводить вибір товару, встановлює бажану зміну ціни та вказує постійні витрати даного товару (Рисунок 6.11);

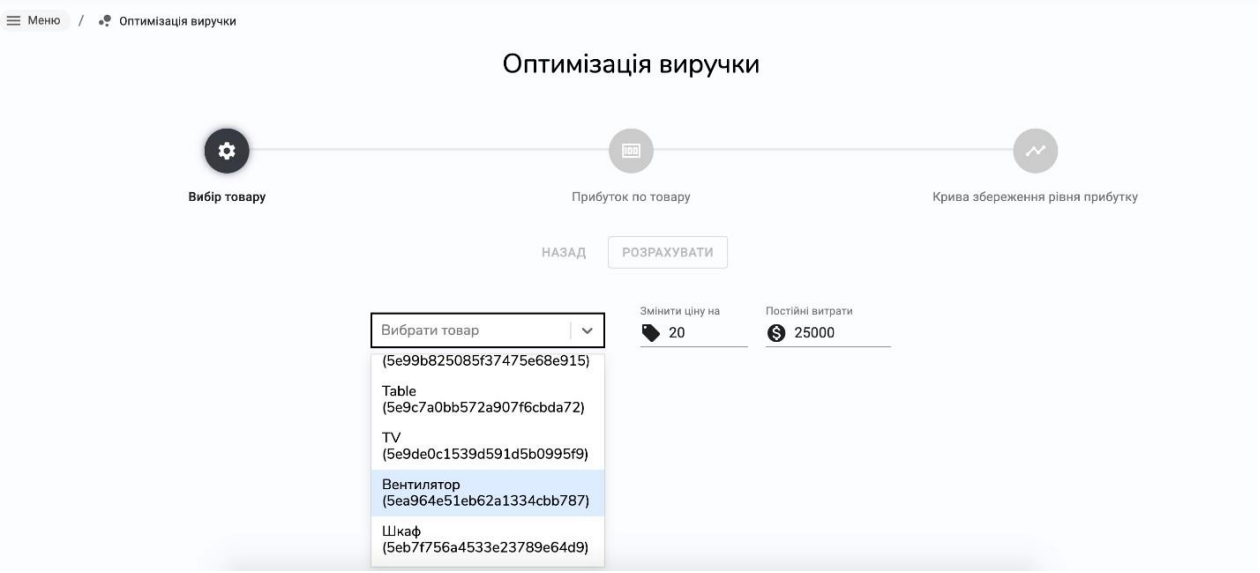

Рисунок 6.11 – Вибір товару та введення початкових даних

- на другому етапі проводяться підрахунки початкового та розрахункового стану. Вираховуються об’єм продажу, виручка, витрати змінні, витрати постійні та прибуток відповідно. Користувач має можливість побачити пораду та вираховуванні дані, представлені у табличній формі (Рисунок 6.12);

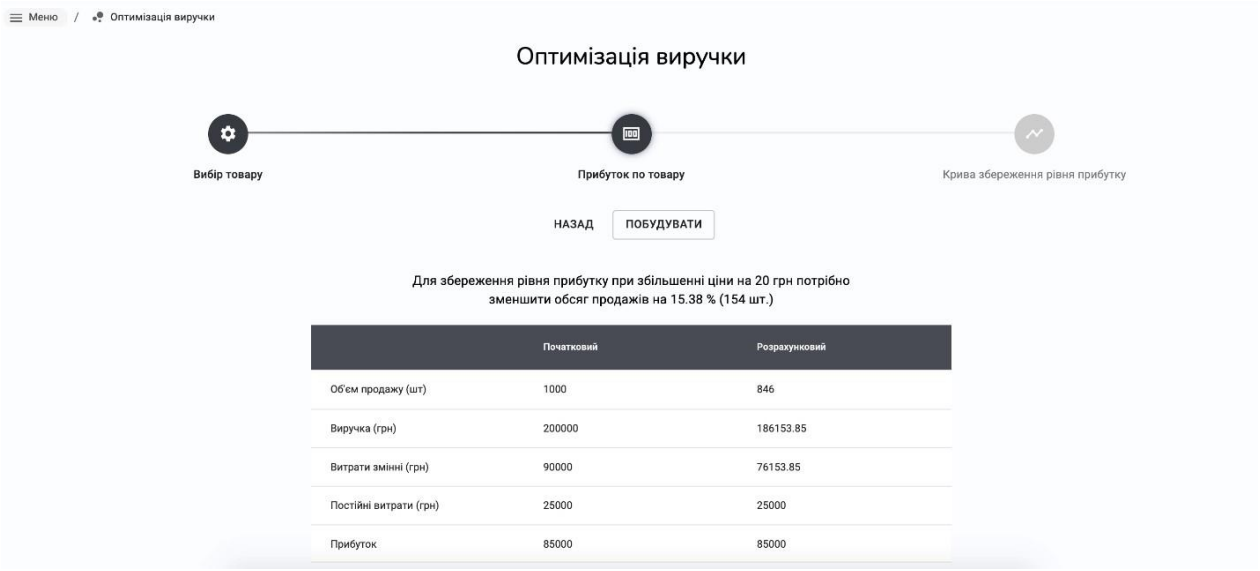

Рисунок 6.12 – Розрахунок прибутку по товару

- на третьому етапі підраховуються дані для розрахунку певного рівня прибутку та будується крива збереження рівня прибутку. Користувач має можливість побачити всі дані у табличній формі та графік кривої, зроблений за допомогою пакету recharts (Рисунок 6.13).

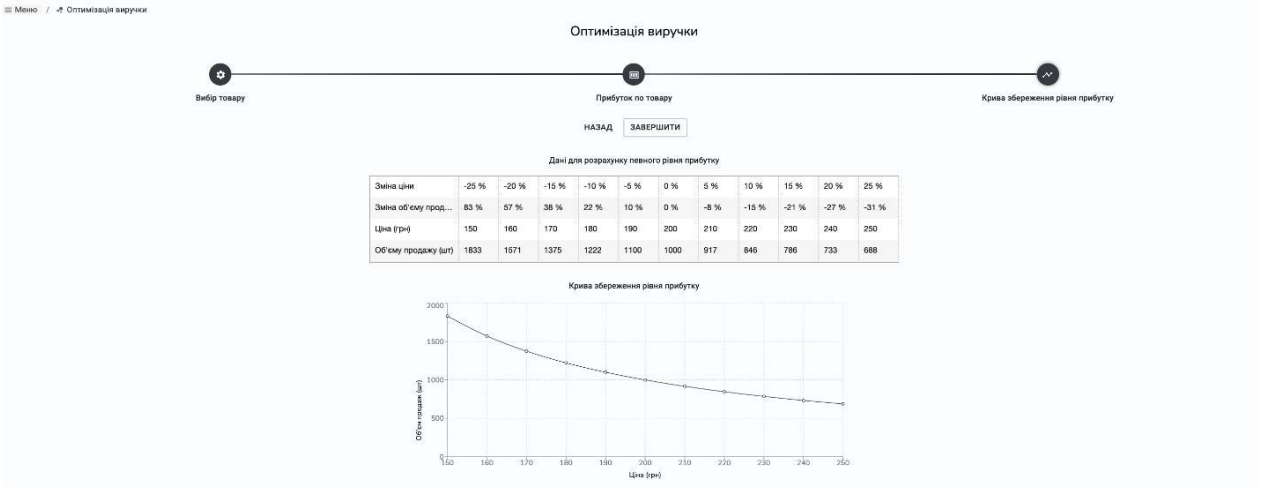

Рисунок 6.13 – Крива збереження рівня прибутку

Алгоритм роботи етапу підрахунку початкового та розрахункового стану товару (Рисунок 6.14):

- знаходяться ціна та змінні витрати обраного товару;
- знаходяться всі продажі обраного товару та підраховується їх кількість;
- вираховується початкова виручка;
- вираховуються початкові загальні змінні витрати;
- вираховується початковий прибуток;
- вираховується розрахункова виручка;
- вираховуються розрахункові загальні замінні витрати;
- вираховується розрахунковий прибуток;
- формується результуючий об’єкт, який включає в себе початкові дані, розрахункові дані, необхідну зміну обсягу продажів у відсотках та кількісно.

```
const { price, variableCosts: varCosts } = await Product.findById(
  productId,
);
const productSales = await Sale.find({ productId });
const productSalesCount = productSales.reduce(
  (acc, item) => acc + item.count,
  0,
);

const sourceRevenue = productSalesCount * price;
const sourceVariableCosts = productSalesCount * varCosts;
const sourceProfit = sourceRevenue - sourceVariableCosts - fixedCosts;

const necessaryVolume =
  (((-1 * +changePriceTo) / price) * 100 * price) /
  (price - varCosts + +changePriceTo);

const estimatedVolumeOfSales =
  productSalesCount + necessaryVolume * 0.01 * productSalesCount;
const estimatedRevenue = estimatedVolumeOfSales * (price + +changePriceTo);
const estimatedVariableCosts = estimatedVolumeOfSales * varCosts;
const estimatedProfit =
  estimatedRevenue - estimatedVariableCosts - fixedCosts;
```

Рисунок 6.14 – Алгоритм розрахунків по прибутку

Алгоритм побудови кривої збереження рівня прибутку (Рисунок 6.15):

- вибираються всі продажі обраного товару;
- підраховується їх кількість;
- створюється функція для отримання відсоткової зміни ціни;
- створюється функція для отримання необхідної зміни об’єму продаж;
- створюється функція для отримання необхідної зміни ціни товару;
- формується результуючий об’єкт з необхідними даними, використовуючи вище вказані функції для діапазону [-25% ; 25%].

```
const { price, variableCosts: varCosts } = await Product.findById(
  productId,
);
const productSales = await Sale.find({ productId });
const productSalesCount = productSales.reduce(
  (acc, item) => acc + item.count,
  0,
);

const getChangePriceTo = necessaryPriceChange =>
  price * necessaryPriceChange * 0.01;

const getVolumeChange = changePriceTo =>
  (((-1 * changePriceTo) / price) * 100 * price) /
  (price - varCosts + changePriceTo);

const getNecessaryVolumeChange = necessaryPriceChange => {
  const changePriceTo = getChangePriceTo(necessaryPriceChange);
  const necessaryVolumeChange = getVolumeChange(changePriceTo);
  return Math.round(necessaryVolumeChange);
};

const getNecessaryPrice = necessaryPriceChange => {
  const changePriceTo = getChangePriceTo(necessaryPriceChange);
  return price + changePriceTo;
};

const getNecessaryProductSalesCount = necessaryPriceChange => {
  const changePriceTo = getChangePriceTo(necessaryPriceChange);
  const necessaryVolumeChange = getVolumeChange(changePriceTo);
  return Math.round(
    productSalesCount + necessaryVolumeChange * 0.01 * productSalesCount,
  );
};
```

Рисунок 6.15 – Алгоритм побудови кривої збереження рівня прибутку

Отже, проаналізувавши всі теоретичні дані, для даної IAC був обраний додатковий функціонал по аналізу та оптимізації системи, який був успішно реалізований та впроваджений у застосунок.

ВИСНОВКИ

У даному дипломному проєкті була розроблена ІАС управління магазином у вигляді веб-застосунка. Її функціональні можливості відповідають поставленій меті, яка полягала в створенні ІАС, яка, крім базового функціоналу, має приємний інтерфейс, просте адміністрування та надає додаткові можливості аналізу та оптимізації роботи та діяльності магазину.

Після детального розгляду існуючих у даний момент рішень на ринку програмного забезпечення, були сформовані основні вимоги до розроблюваної програми. У ході наступних етапів розробки програми всі поставлені вимоги було цілковито задоволено, у результаті розроблена ІАС має наступні можливості:

- повноцінна робота з сутностями системи, а саме: імпорт, створення, редагування, видалення, пошук, сортування та їх обслуговування;
- збереження усіх дій користувачів у програмі та їх відображення у модулі життєвого циклу системи:
- аналіз попиту на товари;
- аналіз роботи працівників;
- аналіз роботи з постачальниками;
- аналіз товару з можливостю підбирати необхідну кількість продажів при відповідній зміні ціни та побудова кривої збереження прибутку в діапазоні 25% від початкової ціни.
- наявність гнучкої та зручної системи прав.

Для досягнення максимальної швидкості роботи, використовувався стек MERN, який включає у себе найсучасніші технології веб-розробки, такі як: система управління базами даних MongoDB, фреймворк Express у якості REST сервісу, бібліотека побудови користуватського інтерфейсу React та платформа Node.js у якості серверу. Вони надають необмежені можливості для

розробника зручно і швидко додавати нові програмні модулі та дозволяють легко масштабувати систему.

Були розроблені і оптимізовані алгоритми роботи з сутностями системи, які крім звичайних операцій, таких як: створення, видалення, редагування, перегляд, сортування та пошук, надають користувачу додаткові можливості по аналізу роботи працівників, постачальників та продаж. Також був доданий окремий модуль оптимізації, який має доступ до повної звітності магазину, на основі цих даних він дозволяє користувачу проводити максимізацію виручки та прибутку при зміні ціни на відповідний товар у магазині.

Був створений швидкий, зручний та приємний інтерфейс, який відповідає всім потребам користувача та найсучаснішим стандартам дизайну та UX.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Белов В.С. Інформаційно-аналітичні системи. Основи проектування та застосування [Електронний ресурс] : Режим доступу : https://ed.eenu.edu.ua/upload/storage/1933/91ecc3349a348d23b7e6c56097c9ddff.pdf – 16.02.2005 р.
2. П’янков О. В. Інформаційно-аналітичні система: призначення, роль, властивості. [Електронний ресурс] : Режим доступу : https://cyberleninka.ru/article/n/informatsionno-analiticheskaya-sistema-naznachenie-rol-svoystva/viewer – 02.10.2014 р.
3. Максимізація виручки та прибутку [Електронний ресурс] : Режим доступу : https://www.cfin.ru/finanalysis/math/price_flexibility.shtml – 02.08.2005 р.
4. Документація React [Електронний ресурс] : Режим доступу : https://reactjs.org/docs/getting-started.html – 03.02.2020 р.
5. Робота з Redux [Електронний ресурс] : Режим доступу : https://redux.js.org/introduction/getting-started – 15.02.2020 р.
6. Керівництво по Express [Електронний ресурс] : Режим доступу : https://expressjs.com/ru/guide/routing.html – 25.01.2020 р.
7. Онлайн-руководство по MongoDB [Електронний ресурс] : Режим доступу : https://metanit.com/nosql/mongodb/ – 28.03.2018 р.
8. Использование базы данных (с помощью Mongoose) [Електронний ресурс] : Режим доступу : https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs/mongoose – 18.03.2019 р.
9. Чем на самом деле является Node.js? [Електронний ресурс] : Режим доступу : https://habr.com/ru/post/420123 – 14.08.2018 р.

10. Баронов В. В. Информационные технологии и управление предприятием [Електронний ресурс] : Режим доступу : https://econ.wikireading.ru/44180 – 20.06.2018 р.
11. Тім С. Что такое стек MERN, и как с ним работать? [Електронний ресурс] : Режим доступу : https://habr.com/ru/company/piter/blog/458096 – 28.06.2019 р.
12. Практическое приложение теории: максимизация выручки производителя [Електронний ресурс] : Режим доступу : https://studopedia.info/5-72682.html – 05.01.2016 р.
13. Оформлення текстових документів у навчальному процесі. Стандарт організації (кафедри) СОУ АУТС 01-15. Для студентів кафедри автоматики та управління в технічних системах [Текст] / Уклад.: Я.Ю. Дорогий, Н.Б. Репнікова, О.І. Ролік, Л.Ю. Юрчук – К.: НТУУ «КПІ», 2015. – 27 с.
14. ДСТУ 1.5 : 2003 «Правила побудови, викладання, оформлення та вимоги до змісту нормативних документів». «Державна система стандартизації України. Загальні вимоги до побудови, викладу, оформлення та змісту стандартів» [Текст]. – На заміну ДСТУ 1.5-93 ; Чинний від 2003-07-01. – Київ : Видавництво стандартів, 2003. – 44 с.
15. Методичні вказівки до виконання дипломних робіт освітньо-кваліфікаційного рівня «бакалавр» для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри «Автоматики та управління у технічних системах» [Текст] / Уклад.: К.С. Дорошенко, А. О. Новацький, Н. Б. Репнікова, Л. Ю. Юрчук. – К.: НТУУ «КПІ», 2014. – 67 с.

ДОДАТОК А Лістинг функції checkProductsRate

(довідковий)

```
const checkProductsRate = async (_, res) => {
  const sendResponse = ({ listOfProductsInOrderOfSale }) => {
    res.status(200).json({
      status: 'success',
      listOfProductsInOrderOfSale,
    });
  };
  try {
    const resultList = [];
    const products = await Product.find();
    for (const product of products) {
      const productId = product._id;
      const productSales = await Sale.find({ productId });
      const totalSales = productSales.reduce(
        (acc, item) => acc + item.count,
        0,
      );
      const supplierId = product.supplierId;
      const { name: supplier } = await Supplier.findById({ _id: supplierId });
      const productInfo = {
        id: productId,
        name: product.name,
        totalSales,
        supplier,
      };
      resultList.push(productInfo);
    }
    const listOfProductsInOrderOfSale = resultList
      .sort((a, b) => b.totalSales - a.totalSales)
      .map((item, idx) => ({ ...item, order: 1 + idx }));
    sendResponse({
      listOfProductsInOrderOfSale,
    });
  }
```

ДОДАТОК Б Лістинг функції trackCollaborationWithSuppliers (довідковий)

```
const trackCollaborationWithSuppliers = async (req, res) => {
  const { supplierId } = req.params;
  const sendResponse = (data => {
    res.status(200).json({
      status: 'success',
      data,
    });
  };
  try {
    const products = await Product.find({ supplierId });
    const purchasedProductsCost = products.reduce((acc, item) => {
      const totalProductPrice = item.count * item.wholesalePrice;
      return acc + totalProductPrice;
    }, 0);
    let saleProductsCost = 0;
    for (const product of products) {
      const sales = await Sale.find({ productId: product._id });
      if (sales.length > 0) {
        sales.forEach(sale => {
          saleProductsCost += sale.count * sale.price;
        });
      }
    }
    const supplierEfficiency = (
      saleProductsCost / purchasedProductsCost
    ).toFixed(2);
    sendResponse({
      purchasedProductsCost,
      saleProductsCost,
      supplierEfficiency,
    });
  } catch (error) {
    sendError(error);
  }
};
```

ДОДАТОК В Лістинг функції calculateProductProfit

(довідковий)

```
const calculateProductProfit = async (req, res) => {
  const { productId, changePriceTo, fixedCosts } = req.params;
  const sendResponse = ({ result }) => {
    res.status(200).json({
      status: 'success',
      result,
    });
  };
  try {
    const { price, variableCosts: varCosts } = await Product.findById(
      productId,
    );
    const productSales = await Sale.find({ productId });
    const productSalesCount = productSales.reduce(
      (acc, item) => acc + item.count,
      0,
    );
    const sourceRevenue = productSalesCount * price;
    const sourceVariableCosts = productSalesCount * varCosts;
    const sourceProfit = sourceRevenue - sourceVariableCosts - fixedCosts;
    const necessaryVolume =
      (((-1 * +changePriceTo) / price) * 100 * price) /
      (price - varCosts + +changePriceTo);
    const estimatedVolumeOfSales =
      productSalesCount + necessaryVolume * 0.01 * productSalesCount;
    const estimatedRevenue = estimatedVolumeOfSales * (price + +changePriceTo);
    const estimatedVariableCosts = estimatedVolumeOfSales * varCosts;
    const estimatedProfit =
      estimatedRevenue - estimatedVariableCosts - fixedCosts;
    const result = {
      source: {
        volumeOfSales: productSalesCount,
        revenue: Number.isInteger(sourceRevenue)
          ? Number(sourceRevenue)
          : Number(sourceRevenue.toFixed(2)),
```

```
        variableCosts: Number.isInteger(sourceVariableCosts)
          ? Number(sourceVariableCosts)
          : Number(sourceVariableCosts.toFixed(2)),
        fixedCosts: Number.isInteger(+fixedCosts)
          ? Number(fixedCosts)
          : Number(Number(fixedCosts).toFixed(2)),
        profit: Number.isInteger(sourceProfit)
          ? Number(sourceProfit)
          : Number(sourceProfit.toFixed(2)),
      },
      estimated: {
        volumeOfSales: Math.floor(estimatedVolumeOfSales),
        revenue: Number.isInteger(estimatedRevenue)
          ? Number(estimatedRevenue)
          : Number(estimatedRevenue.toFixed(2)),
        variableCosts: Number.isInteger(estimatedVariableCosts)
          ? Number(estimatedVariableCosts)
          : Number(estimatedVariableCosts.toFixed(2)),
        fixedCosts: Number.isInteger(+fixedCosts)
          ? Number(fixedCosts)
          : Number(Number(fixedCosts).toFixed(2)),
        profit: Number.isInteger(estimatedProfit)
          ? Number(estimatedProfit)
          : Number(estimatedProfit.toFixed(2)),
      },
      necessaryVolume: Number.isInteger(necessaryVolume)
        ? Number(necessaryVolume)
        : Number(necessaryVolume.toFixed(2)),
      targetPriceChange: Number.isInteger(changePriceTo)
        ? Number(changePriceTo)
        : Number(Number(changePriceTo).toFixed(2)),
    };
    sendResponse({ result });
  } catch (error) {
    sendError(error);
  }
};
```

ДОДАТОК Г Лістинг функції buildProfitRetentionCurve

(довідковий)

```
const buildProfitRetentionCurve = async (req, res) => {
  const { productId } = req.params;
  const sendResponse = ({ result }) => {
    res.status(200).json({
      status: 'success',
      result,
    });
  };
  try {
    const { price, variableCosts: varCosts } = await Product.findById(
      productId,
    );
    const productSales = await Sale.find({ productId });
    const productSalesCount = productSales.reduce(
      (acc, item) => acc + item.count,
      0,
    );
    const getChangePriceTo = necessaryPriceChange =>
      price * necessaryPriceChange * 0.01;
    const getVolumeChange = changePriceTo =>
      (((-1 * changePriceTo) / price) * 100 * price) /
      (price - varCosts + changePriceTo);
    const getNecessaryVolumeChange = necessaryPriceChange => {
      const changePriceTo = getChangePriceTo(necessaryPriceChange);
      const necessaryVolumeChange = getVolumeChange(changePriceTo);
      return Math.round(necessaryVolumeChange);
    };
    const getNecessaryPrice = necessaryPriceChange => {
      const changePriceTo = getChangePriceTo(necessaryPriceChange);
      return price + changePriceTo;
    };
    const getNecessaryProductSalesCount = necessaryPriceChange => {
      const changePriceTo = getChangePriceTo(necessaryPriceChange);
      const necessaryVolumeChange = getVolumeChange(changePriceTo);
      return Math.round(
```

```
      productSalesCount + necessaryVolumeChange * 0.01 * productSalesCount,
    );
  };
  const result = [
    {
      title: 'Зміна ціни',
      '-25': '-25 %',
      '-20': '-20 %',
      '-15': '-15 %',
      '-10': '-10 %',
      '-5': '-5 %',
      '0': '0 %',
      '5': '5 %',
      '10': '10 %',
      '15': '15 %',
      '20': '20 %',
      '25': '25 %',
    },
    {
      title: Зміна об'єму продажу,
      '-25': ${getNecessaryVolumeChange(-25)} %,
      '-20': ${getNecessaryVolumeChange(-20)} %,
      '-15': ${getNecessaryVolumeChange(-15)} %,
      '-10': ${getNecessaryVolumeChange(-10)} %,
      '-5': ${getNecessaryVolumeChange(-5)} %,
      '0': '0 %',
      '5': ${getNecessaryVolumeChange(5)} %,
      '10': ${getNecessaryVolumeChange(10)} %,
      '15': ${getNecessaryVolumeChange(15)} %,
      '20': ${getNecessaryVolumeChange(20)} %,
      '25': ${getNecessaryVolumeChange(25)} %,
    },
    {
      title: 'Ціна (грн)',
      '-25': getNecessaryPrice(-25),
      '-20': getNecessaryPrice(-20),
      '-15': getNecessaryPrice(-15),
      '-10': getNecessaryPrice(-10),
      '-5': getNecessaryPrice(-5),
      '0': price,
```

```
        '5': getNecessaryPrice(5),
        '10': getNecessaryPrice(10),
        '15': getNecessaryPrice(15),
        '20': getNecessaryPrice(20),
        '25': getNecessaryPrice(25),
      },
      {
        title: Об'єму продажу (шт),
        '-25': getNecessaryProductSalesCount(-25),
        '-20': getNecessaryProductSalesCount(-20),
        '-15': getNecessaryProductSalesCount(-15),
        '-10': getNecessaryProductSalesCount(-10),
        '-5': getNecessaryProductSalesCount(-5),
        '0': productSalesCount,
        '5': getNecessaryProductSalesCount(5),
        '10': getNecessaryProductSalesCount(10),
        '15': getNecessaryProductSalesCount(15),
        '20': getNecessaryProductSalesCount(20),
        '25': getNecessaryProductSalesCount(25),
      },
    ];
    sendResponse({
      result,
    });
  } catch (error) {
    sendError(error);
  }
};
```