Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

**Допущено до захисту**
Завідувач кафедри комп'ютерних наук,
кандидат технічних наук, доцент
(науковий ступінь, вчене звання)
_____ Ірина МАШКІНА
(підпис)
« ___ » _____ 2025 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня «бакалавр»**
**Спеціальність 122 Комп'ютерні науки**
**Освітня програма 122.00.01 Інформатика**

**Тема:** СТВОРЕННЯ ІНФОРМАЦІЙНОЇ ПАНЕЛІ КЕРУВАННЯ ТОВАРАМИ У ІНТЕРНЕТ МАГАЗИНІ

**Виконав:**
студент IV курсу денної форми навчання
групи ІНб-2-21-4.0д
Вітцівський Андрій Сергійович
_____
Науковий керівник:
кандидат педагогічних наук
(науковий ступінь, вчене звання)
Карпенко Анастасія Сергіївна
_____
*(прізвище, ім'я, по батькові)* *( підпис)*

Київ – 2025

## РЕФЕРАТ бакалаврської роботи

Кваліфікаційна робота: 44 с., 13 рис., 30 посилань.

Актуальність: Сучасні умови розвитку електронної комерції та прагнення до ефективного управління товарами в інтернет-магазині на сьогодні є критично важливими для бізнесу. Велика кількість товарів, змінювані ціни, наявність товарів на складі та автоматизація процесів вимагають використання сучасних інформаційних панелей. Розробка ефективної інформаційної панелі керування товарами дозволяє покращити контроль за товарними позиціями, оптимізувати бізнес-процеси та підвищити конкурентоспроможність підприємства.

Об’єктом дослідження є процес управління товарами в інтернет-магазині.

Предмет дослідження – методи та технології створення інформаційної панелі керування товарами.

Метою роботи є створення інформаційної панелі керування товарами для інтернет-магазину, яка забезпечить зручний інтерфейс для адміністрування товарних позицій, додавання товарів, а також контроль залишків і стану товарів на складі.

Завдання:

- Провести аналіз сучасних інформаційних панелей керування товарами в інтернет-магазинах.
- Дослідити вебтехнології, що використовуються для розробки таких панелей.
- Розробити архітектуру інформаційної панелі керування.
- Реалізувати фронтенд частину панелі з використанням Next.js та Tailwind CSS.
- Створити бекенд для обробки запитів і взаємодії з базою даних на основі Node.js та Express.js.
- Провести тестування панелі та аналіз її продуктивності.

ОСНОВНІ РЕЗУЛЬТАТИ:

У процесі роботи було розроблено інформаційну панель керування товарами, що дозволяє:

- Здійснювати додавання, редагування та видалення товарів.
- Оптимізувати пошук і фільтрацію товарів за різними критеріями.
- Відстежувати наявність товарів на складі.
- Автоматизувати оновлення даних про товари.
- Здійснити інтеграцію з базою даних для швидкої взаємодії між фронтендом та бекендом.

Новизна дослідження полягає у використанні сучасних вебтехнологій (Next.js, Tailwind CSS, Node.js, Express.js) для створення швидкої, адаптивної та зручної інформаційної панелі, що спрощує управління товарним асортиментом.

Практичне значення дослідження: результати дослідження можуть бути використані для оптимізації управління товарами в інтернет-магазинах, що сприятиме підвищенню ефективності роботи підприємства, зниженню операційних витрат та покращенню обслуговування клієнтів.

Ключові слова: інформаційна панель, інтернет-магазин, управління товарами, Next.js, Tailwind CSS, Node.js, Express.js, автоматизація, база даних, вебтехнології.

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

**«Затверджую»**
Завідувач
кафедри комп'ютерних наук,
кандидат технічних наук, доцент
(науковий ступінь, вчене звання)
_____ Ірина МАШКІНА
(підпис)
« ___ » _____ 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІНУ РОБОТУ БАКАЛАВРА
«Створення інформаційної панелі керування товарами у інтернет магазині»

Виконавець – студент групи ІНб-2-21-4.0д,
спеціальності 122 Комп'ютерні наук,
освітньої програми 122.00.01 Інформатика
**Вітцівський Андрій Сергійович**

1. Вихідні дані: *Публікації за напрямом дослідження, зокрема, в галузі автоматизації управління товарами, розробки інформаційних панелей керування, використання сучасних вебтехнологій, зокрема Next.js, Tailwind CSS, Node.js, Express.js та Prisma ORM.*
2. Основні завдання: *Здійснити огляд публікацій, присвячених сучасним інформаційним панелям керування товарами в інтернет-магазинах. Обґрунтувати мету, об'єкт, предмет та наукові основи дослідження. Розробити завдання, структуру та зміст магістерської роботи. Обрати та обґрунтувати методи і засоби дослідження, зокрема, технології для розробки інформаційної панелі. Підготувати матеріали до розділів роботи відповідно до індивідуального завдання. Визначити склад ключових модулів інформаційної панелі керування товарами. Розробити систему управління товарами з можливістю створення, редагування, видалення та пошуку товарів. Розробити алгоритми та реалізувати бекенд-частину панелі за допомогою Node.js, Express.js та Prisma. Реалізувати фронтенд-частину панелі на основі Next.js та Tailwind CSS, забезпечивши адаптивний дизайн та зручний користувацький інтерфейс. Провести тестування продуктивності системи, аналіз швидкодії, навантаження та ефективності обробки запитів. Виконати аналіз результатів дослідження, скласти висновки та оформити магістерську роботу відповідно до встановлених вимог. Підготувати наукову статтю (тези доповіді) за темою роботи.*

3. Пояснювальна записка: *Обсяг – до 60 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.*

4. Графічні матеріали: *не передбачено.*

5. Додатки: *не передбачено*

6. Строк подання роботи на кафедру: «___» _____2025 р.

| | |
|---|---|
| Науковий керівник<br>кандидат педагогічних наук<br>Карпенко Анастасія Сергіївна<br>_____<br>Дата: | Виконавець:<br>Вітцівський Андрій Сергійович<br>_____<br>Дата: |

# ЗМІСТ

# ВСТУП

Розвиток електронної комерції зумовлює необхідність створення ефективних та зручних засобів управління товарним асортиментом в інтернет-магазинах. Сучасні цифрові технології дозволяють розробляти гнучкі та функціональні інформаційні панелі, які забезпечують ефективний контроль за товарами, їх наявністю, цінами та іншими характеристиками. Високий рівень конкуренції на ринку електронної комерції вимагає впровадження інноваційних рішень, які спрощують процеси управління та покращують користувацький досвід.

Одним із ключових аспектів успішної роботи інтернет-магазину є можливість швидкого та зручного управління товарними позиціями. Інформаційна панель керування дозволяє адміністраторам і менеджерам магазину оперативно додавати, редагувати, видаляти товари, стежити за залишками на складі та контролювати їх реалізацію. Вибір технологій для розробки такої панелі є важливим етапом, оскільки від нього залежить продуктивність, масштабованість і зручність системи.

Актуальність теми дипломної роботи зумовлена необхідністю створення сучасної, ефективної та зручної інформаційної панелі керування товарами, яка забезпечить швидку обробку даних та інтеграцію з іншими сервісами. Для реалізації проєкту буде використано сучасні вебтехнології, зокрема Next.js для фронтенд-частини, а також Node.js та Express.js для бекенд-частини. Це забезпечить високу продуктивність, зручність розробки та підтримку гнучкої архітектури.

Також важливим аспектом є інтеграція інформаційної панелі з іншими сервісами, такими як платіжні системи, логістичні компанії, системи управління взаємовідносинами з клієнтами (CRM). Це дозволяє створити єдину екосистему, яка спрощує взаємодію між різними компонентами бізнесу та підвищує загальну продуктивність.

Завдяки сучасним технологіям з’являється можливість масштабування інформаційних панелей, що дозволяє малим і середнім бізнесам ефективно

використовувати інструменти, доступні раніше лише великим корпораціям. Гнучкість архітектури та можливість кастомізації роблять такі системи незамінними для власників інтернет-магазинів.

**Метою роботи** є створення інформаційної панелі керування товарами для інтернет-магазину, яка забезпечить зручний інтерфейс для адміністрування товарних позицій, додавання товарів, а також контроль залишків і стану товарів на складі.

**Для досягнення мети необхідно розв'язати такі завдання:**

1. Провести аналіз сучасних інформаційних панелей керування товарами в інтернет-магазині.
2. Обґрунтувати вибір методів і технологій для створення інформаційної панелі.
3. Розробити фронтенд-частину панелі з використанням Next.js і Tailwind CSS.
4. Створити бекенд-частину за допомогою Node.js та Express.js.
5. Провести тестування розробленої інформаційної панелі й оцінити її продуктивність.

**Об'єктом дослідження** є процес управління товарами в інтернет-магазині.

**Предмет дослідження** – методи та технології створення інформаційної панелі керування товарами.

Для розв'язання поставлених завдань застосовуються методи аналізу та порівняння сучасних рішень, проєктування архітектури вебзастосунків, програмування фронтенд- і бекенд-частини, а також тестування вебзастосунків.

Практичне значення отриманих результатів полягає у створенні універсальної інформаційної панелі, яку можна інтегрувати у різні інтернет-магазини для ефективного керування товарами. Результати роботи можуть бути використані в реальних проєктах для оптимізації процесів управління товарними позиціями.

# РОЗДІЛ І
# ТЕОРЕТИЧНІ АСПЕКТИ УПРАВЛІННЯ ТОВАРАМИ В ІНТЕРНЕТ-МАГАЗИНІ

## 1.1 Аналіз сучасних інформаційних панелей управління товарами в інтернет-магазині

Сучасні інформаційні панелі управління товарами в інтернет-магазині є незамінними інструментами для адміністраторів, які займаються керуванням асортиментом, ціноутворенням та залишками товарів. Вони забезпечують централізоване керування всім товарним каталогом, оптимізують процеси додавання, редагування та видалення товарних позицій, що дозволяє значно зменшити навантаження на адміністративний персонал.

Інформаційні панелі можуть бути реалізовані як частина системи управління контентом (CMS) або як незалежні вебзастосунки, що інтегруються з базою даних магазину. Найпопулярніші CMS, такі як Shopify, WooCommerce, Magento та OpenCart, мають вбудовані панелі керування, які дозволяють виконувати всі необхідні операції з товарами (рис. 1.1). Однак, для великих інтернет-магазинів часто потрібні більш гнучкі рішення, що адаптуються під індивідуальні потреби бізнесу [1].

Основні функціональні можливості сучасних панелей управління товарами включають додавання, редагування та видалення товарів з можливістю завантаження фото, додавання опису, встановлення цін, категорій та інших атрибутів. Керування запасами передбачає автоматичне оновлення залишків на складі та сповіщення про низький рівень запасів. Масове завантаження та оновлення даних реалізується через імпорт/експорт товарів у файлах CSV або інтеграцію з ERP-системами. Автоматизація процесів дозволяє оновлювати ціни залежно від попиту, валютних коливань або інших факторів. Фільтрація та пошук товарів забезпечує швидке знаходження потрібних товарних позицій за різними параметрами. Аналітика та звітність дозволяють аналізувати продажі, перегляди та популярність товарів.

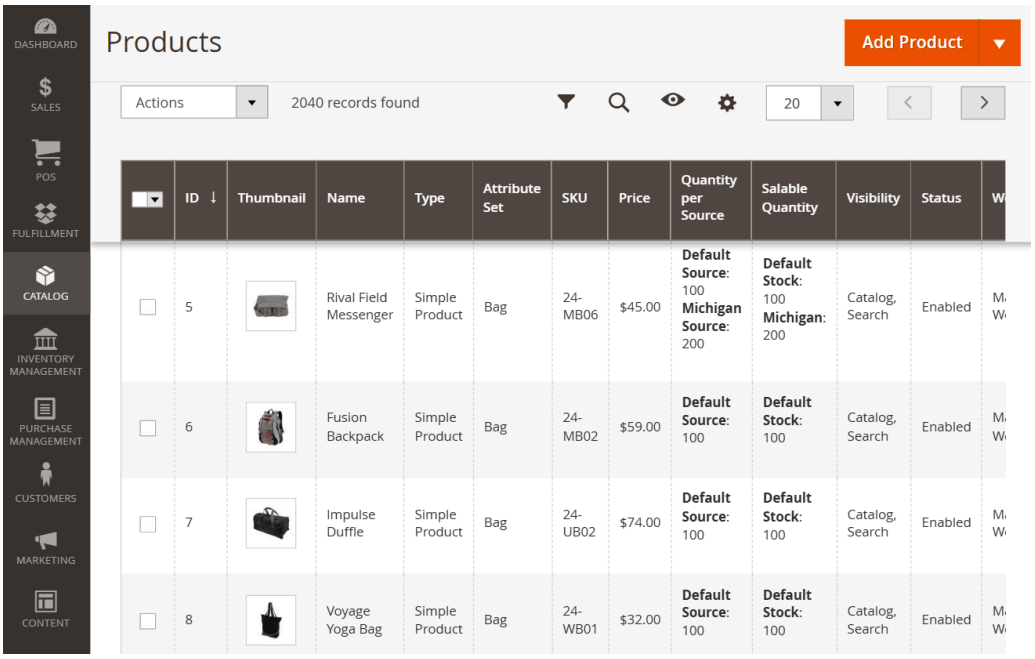


Рисунок 1.1. Інформаційна панель Magento

Сучасні інформаційні панелі також повинні бути адаптивними, щоб забезпечувати зручність використання на різних пристроях, включаючи мобільні телефони та планшети. Крім того, важливим є аспект безпеки, оскільки адміністраторська панель має доступ до критично важливої інформації про товари, фінанси та клієнтів [2].

При виборі технологій для реалізації власної панелі керування необхідно враховувати продуктивність, швидкість обробки запитів і роботу з великими обсягами даних. Гнучкість та масштабованість системи забезпечують можливість розширення функціональності та інтеграцію з іншими сервісами. Простота використання визначається інтуїтивним інтерфейсом та зручним UX/UI дизайном. Безпека гарантується захистом від несанкціонованого доступу, шифруванням даних і можливістю управління ролями користувачів.

Аналіз сучасних інформаційних панелей управління товарами в інтернет-магазинах дозволяє визначити ключові вимоги до їх розробки та обрати найбільш оптимальні рішення для створення власного застосунку, який відповідатиме потребам бізнесу [25].

## 1.2 Огляд вебтехнологій для створення інформаційної панелі керування в інтернет-магазині

Сучасні інформаційні панелі керування в інтернет-магазині базуються на різних вебтехнологіях, які забезпечують ефективну роботу, високу продуктивність та зручність використання. Основними складовими таких панелей є фронтенд, бекенд, база даних та механізми безпеки, які взаємодіють для забезпечення коректного функціонування системи [3].

Фронтенд-розробка інформаційних панелей зазвичай реалізується за допомогою сучасних фреймворків та бібліотек, таких як React.js, Vue.js або Angular. Ці технології дозволяють створювати інтерактивні інтерфейси, що динамічно оновлюються без необхідності перезавантаження сторінки. Використання Next.js або Nuxt.js додає можливості серверного рендерингу, що покращує швидкість завантаження сторінки та SEO. Для стилізації інтерфейсу застосовуються CSS-фреймворки, такі як Tailwind CSS, Bootstrap або Material-UI, які допомагають створювати адаптивний дизайн і забезпечують кращу користувацьку взаємодію [4].

Бекенд-частина панелі керування може бути побудована на основі таких технологій, як Node.js з використанням Express.js або Nest.js, що забезпечує швидку обробку запитів та масштабованість системи. Альтернативні рішення включають використання Django (Python) або Spring Boot (Java), які забезпечують високу продуктивність та безпеку. Серверна частина відповідає за обробку запитів, управління бізнес-логікою та зв’язок із базою даних.

Зберігання та управління даними відбувається у базах даних, таких як MySQL, PostgreSQL, MongoDB або Firebase. Реляційні бази даних (MySQL, PostgreSQL) підходять для структурованих даних із чіткими зв’язками між таблицями, тоді як NoSQL-рішення (MongoDB, Firebase) забезпечують гнучкість у зберіганні даних та швидку масштабованість. Використання ORM-інструментів, таких як Sequelize для Node.js або Prisma, спрощує взаємодію між застосунком і базою даних [5].

Забезпечення безпеки інформаційної панелі керування є важливим аспектом розробки. Використання механізмів автентифікації та авторизації, таких як JSON Web Token (JWT) або OAuth 2.0, забезпечує контроль доступу до адміністративних функцій. Шифрування даних, захист від SQL-ін'єкцій, міжсайтових скриптів (XSS) та атак типу CSRF також є необхідними для безпечної роботи системи [24].

Інтеграція інформаційної панелі з API дозволяє автоматизувати обмін даними з іншими сервісами, такими як платіжні шлюзи, служби доставки та CRM-системи. RESTful API або GraphQL використовуються для ефективної взаємодії між фронтендом та бекендом, забезпечуючи швидку передачу даних.

Загалом, вибір технологій для створення інформаційної панелі керування залежить від вимог проєкту, масштабу бізнесу та необхідних функціональних можливостей. Комбінація сучасних фронтенд- та бекенд-рішень із безпечними базами даних та ефективною архітектурою дозволяє створити продуктивну, гнучку та безпечну систему керування товарами в інтернет-магазині [6].

### 1.3 Архітектурні підходи до розробки інформаційної панелі керування в інтернет-магазині

Архітектура інформаційної панелі керування товарами в інтернет-магазині визначає структуру програмного забезпечення, способи взаємодії його компонентів, а також рівень масштабованості, продуктивності та безпеки системи. Одним із ключових підходів до розробки є клієнт-серверна архітектура, яка передбачає розділення системи на дві основні частини: клієнтську та серверну. Клієнтська частина, що реалізується на основі вебтехнологій, таких як React (Next.js) або Vue.js, відповідає за інтерфейс та взаємодію користувача з системою. Серверна частина, зазвичай реалізована на Node.js з використанням Express.js або Django на Python, виконує основну бізнес-логіку, обробляє запити, взаємодіє з базою даних і забезпечує безпеку доступу до інформації [7].

Окрім класичної клієнт-серверної моделі, можливі різні варіанти архітектури. Монолітна архітектура передбачає, що весь додаток є єдиним нероздільним компонентом, де фронтенд, бекенд і база даних розгортаються разом. Такий підхід є простішим у реалізації, швидшим у розробці та зручним для невеликих проєктів. Проте зі збільшенням навантаження та ускладненням функціоналу його підтримка стає проблематичною. Як альтернатива, мікросервісна архітектура розділяє додаток на незалежні сервіси, кожен з яких виконує окрему функцію, наприклад, керування товарами, обробку замовлень або автентифікацію користувачів. Це покращує масштабованість, дозволяє розгортати і оновлювати сервіси незалежно, але значно ускладнює управління всією системою [23].

Важливим аспектом є вибір підходу до реалізації фронтенду та взаємодії з бекендом. Вебдодатки можуть використовувати шаблонну архітектуру MVC (Model-View-Controller), де дані, бізнес-логіка та відображення чітко розділені між окремими компонентами, що робить код більш структурованим. Альтернативним варіантом є MVVM (Model-View-ViewModel), який активно застосовується у фреймворках, таких як Vue.js, де ViewModel є посередником між даними та представленням, забезпечуючи реактивність інтерфейсу. Окремо варто виділити підхід SPA (Single Page Application), при якому фронтенд завантажується один раз і динамічно оновлює сторінку без перезавантажень, що значно покращує швидкість роботи панелі. Однак для коректної роботи з SEO такі додатки потребують серверного рендерингу (SSR), який реалізується у Next.js [8].

З точки зору обміну даними між клієнтом і сервером існує два основні підходи: REST API та GraphQL. REST API є традиційним підходом, при якому сервер обробляє HTTP-запити та повертає дані у форматі JSON. Він простий у реалізації, добре підтримується і сумісний з більшістю технологій. Проте він може бути неефективним у випадках, коли потрібно отримати багато пов’язаних даних. У таких випадках альтернативою є GraphQL, який

дозволяє клієнту запитувати лише необхідні дані, зменшуючи навантаження на сервер і прискорюючи роботу інтерфейсу [21].

З точки зору безпеки архітектура інформаційної панелі керування товарами повинна включати захист від типових загроз, таких як SQL-ін’єкції, атаки CSRF та XSS. Для цього використовуються механізми аутентифікації та авторизації, такі як JSON Web Token (JWT) або OAuth 2.0, які дозволяють забезпечити безпечний доступ до API [26]. Крім того, у сучасних системах активно застосовується розмежування прав доступу, коли різні користувачі мають різні рівні доступу до функцій панелі керування. Важливим аспектом є також використання HTTPS для захисту переданих даних та впровадження політики CORS (Cross-Origin Resource Sharing) для контролю доступу до ресурсів [9].

Таким чином, вибір архітектурного підходу залежить від складності та масштабу інформаційної панелі керування товарами. Для невеликих проєктів підходить монолітна архітектура з використанням MVC, тоді як для великих систем із високими вимогами до масштабованості доцільно застосовувати мікросервісний підхід у поєднанні з REST або GraphQL API, а також забезпечувати високий рівень безпеки за допомогою сучасних методів аутентифікації та захисту даних [10].

# РОЗДІЛ ІІ
# ОБҐРУНТУВАННЯ І ВИБІР МЕТОДІВ ДЛЯ СТВОРЕННЯ ІНФОРМАЦІЙНОЇ ПАНЕЛІ КЕРУВАННЯ ТОВАРАМИ В ІНТЕРНЕТ-МАГАЗИНІ

## 2.1 Постановка завдань, огляд та вибір технологій для створення інформаційної панелі керування товарами в інтернет-магазині

Створення ефективної інформаційної панелі керування товарами в інтернет-магазині є ключовим аспектом для забезпечення зручного та продуктивного управління асортиментом продукції. Цей процес включає детальну постановку завдань, огляд сучасних технологій та обґрунтований вибір оптимальних інструментів для реалізації поставлених цілей [11].

Основною метою розробки інформаційної панелі є надання адміністраторам та менеджерам інтернет-магазину інтуїтивно зрозумілого та функціонального інтерфейсу для управління товарами. До ключових завдань належать:

1. Управління товарним каталогом: створення, редагування та видалення товарів; налаштування атрибутів, таких як назва, опис, ціна, зображення та інші характеристики.
2. Категоризація та фільтрація: організація товарів за категоріями та підкатегоріями; впровадження системи фільтрів для швидкого пошуку та навігації по каталогу [12].
3. Моніторинг запасів: відстеження кількості товарів на складі; автоматичні сповіщення про низький рівень запасів та необхідність поповнення.
4. Аналітика витрат: збір та візуалізація даних про витрати.

Для реалізації інформаційної панелі керування товарами в інтернет-магазині обрано стек сучасних технологій, який забезпечує високу продуктивність, масштабованість, безпеку та зручність розробки. Вибір цих технологій обумовлений їхньою популярністю, активною підтримкою

спільноти, можливістю швидкого впровадження оновлень і адаптацією до потреб бізнесу [13].

Фронтенд-частина побудована на основі Next.js, який є фреймворком для React та пропонує потужні можливості для серверного рендерингу (SSR) та статичної генерації (SSG). Завдяки SSR зменшується час завантаження сторінок, що особливо важливо для SEO, оскільки пошукові системи можуть коректно індексувати контент. Це також покращує продуктивність, оскільки частина обчислень відбувається на сервері, а не в браузері користувача. Next.js підтримує динамічний імпорт компонентів, що дозволяє зменшити розмір початкового завантаження сторінки, покращуючи користувацький досвід. Додатково він інтегрується з API через serverless-функції або традиційні API-ендпоінти, що дозволяє створити єдину екосистему між фронтендом і бекендом [28].

Використання TypeScript забезпечує строгий контроль типів, що зменшує ймовірність помилок під час розробки, особливо в великих командах або при підтримці проєкту в довгостроковій перспективі. TypeScript також дозволяє створювати самодокументований код завдяки статичній типізації, що полегшує супровід та розширення функціоналу панелі [14].

Для роботи з табличними даними використовується DataGrid – бібліотека, що дозволяє створювати зручні та інтерактивні таблиці з підтримкою сортування, фільтрації, пагінації та редагування даних. Це особливо важливо для панелі керування товарами, де потрібно швидко обробляти великі масиви даних, здійснювати пошук за категоріями, ціновими діапазонами, статусами наявності тощо. DataGrid також підтримує вбудовану валідацію даних, що допомагає уникнути помилкових записів [15].

Для візуалізації аналітичних даних використовується Rechart – бібліотека, що дозволяє створювати інтерактивні графіки, діаграми та інші елементи візуалізації даних. Це корисно для відображення динаміки продажів, зміни залишків товарів на складі, аналізу популярних категорій тощо. Rechart забезпечує кастомізацію стилів та інтеграцію з іншими

бібліотеками, що дозволяє створювати складні аналітичні панелі для бізнесу [21].

Стилізація інтерфейсу реалізується за допомогою Tailwind CSS, який використовує утилітарний підхід до написання стилів, що значно прискорює процес розробки. На відміну від класичних CSS-фреймворків, Tailwind дозволяє уникнути перевантаження глобальних стилів, створюючи більш компактний та легкий для супроводу код. Завдяки цьому можна швидко розробляти адаптивні інтерфейси, що коректно відображаються на різних пристроях [29].

Бекенд-частина побудована на основі Node.js та Express.js. Node.js забезпечує асинхронне виконання операцій, що дозволяє швидко обробляти запити до сервера, навіть за великого навантаження. Express.js – це мінімалістичний фреймворк для Node.js, який спрощує створення REST API, управління маршрутами та middleware, а також інтеграцію з базою даних [16].

Для роботи з базою даних використовується PostgreSQL – потужна реляційна система управління базами даних, яка забезпечує підтримку складних запитів, транзакцій, JSON-формату та масштабованість. PostgreSQL є оптимальним вибором для інтернет-магазину, оскільки дозволяє зберігати та обробляти великі обсяги даних, забезпечуючи при цьому високу швидкодію та безпеку.

Для взаємодії з базою даних використовується Prisma – ORM, що значно спрощує роботу з даними завдяки декларативному підходу до моделювання бази. Prisma підтримує автоматичні міграції, що дозволяє легко змінювати структуру бази без ризику втрати даних. Також Prisma забезпечує захист від SQL-ін’єкцій, підвищуючи рівень безпеки бекенду [17].

Завдяки цьому стеку технологій інформаційна панель керування товарами в інтернет-магазині буде швидкою, надійною, зручною у використанні та легкою в супроводі, що дозволить бізнесу ефективно

управляти своїм товарним асортиментом та аналізувати продажі в режимі реального часу [30].

## 2.2 Опис архітектури інформаційної панелі керування товарами в інтернет-магазині

Архітектура інформаційної панелі керування товарами побудована за принципом клієнт-серверної взаємодії, де клієнтська та серверна частини працюють як незалежні компоненти, що взаємодіють через RESTful API інтерфейс (рис. 2.1).

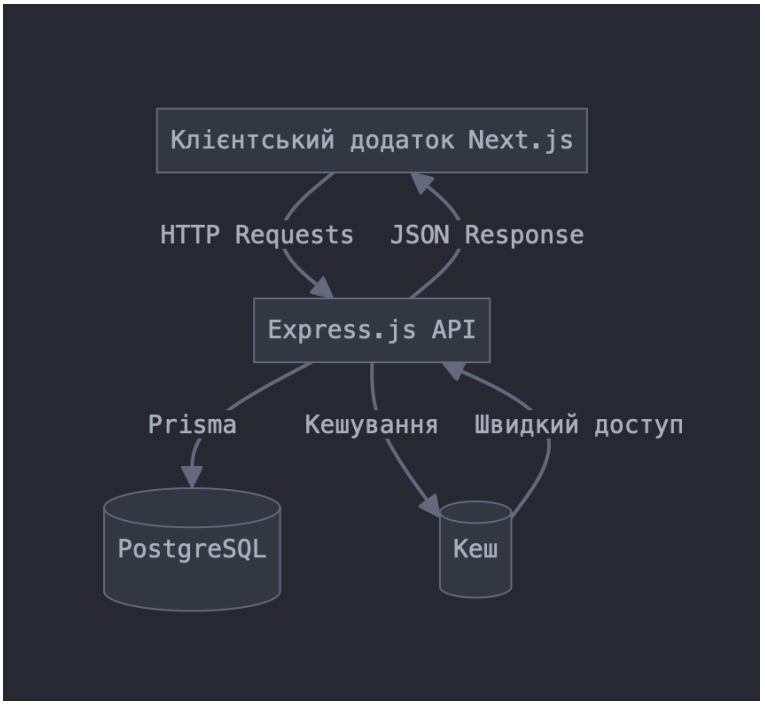


Рисунок 2.1. Діаграма системної архітектури

Клієнтська частина реалізована на базі фреймворку Next.js, який забезпечує швидку розробку та оптимальну продуктивність додатку. Next.js надає можливості серверного рендерингу, що значно покращує початкове завантаження сторінки та SEO показники. Використання TypeScript дозволяє забезпечити надійність коду та зменшити кількість потенційних помилок під час розробки [18].

Для створення сучасного та адаптивного інтерфейсу використовується Tailwind CSS. Цей фреймворк дозволяє швидко створювати стилізовані компоненти без написання власного CSS коду, забезпечуючи при цьому

високу продуктивність та можливість легкої кастомізації дизайну. Компонент DataGrid використовується для відображення табличних даних з можливістю сортування, фільтрації та пагінації, що робить роботу з великими наборами даних зручною та ефективною.

Серверна частина додатку побудована на платформі Node.js з використанням фреймворку Express.js. Це забезпечує швидку та надійну обробку запитів від клієнтської частини. Для роботи з базою даних використовується ORM Prisma, що спрощує взаємодію з PostgreSQL та забезпечує типобезпеку на рівні бази даних [19].

База даних PostgreSQL зберігає всю інформацію про товари, категорії, замовлення та користувачів. Структура бази даних оптимізована для швидкого пошуку та оновлення інформації. Використання Prisma дозволяє легко керувати схемою бази даних та виконувати міграції при необхідності змін структури.

Взаємодія між клієнтською та серверною частинами відбувається через RESTful API endpoints. Кожен endpoint відповідає за певну функціональність: управління товарами, категоріями, обробку замовлень тощо. API підтримує всі необхідні операції CRUD (Create, Read, Update, Delete) для роботи з даними.

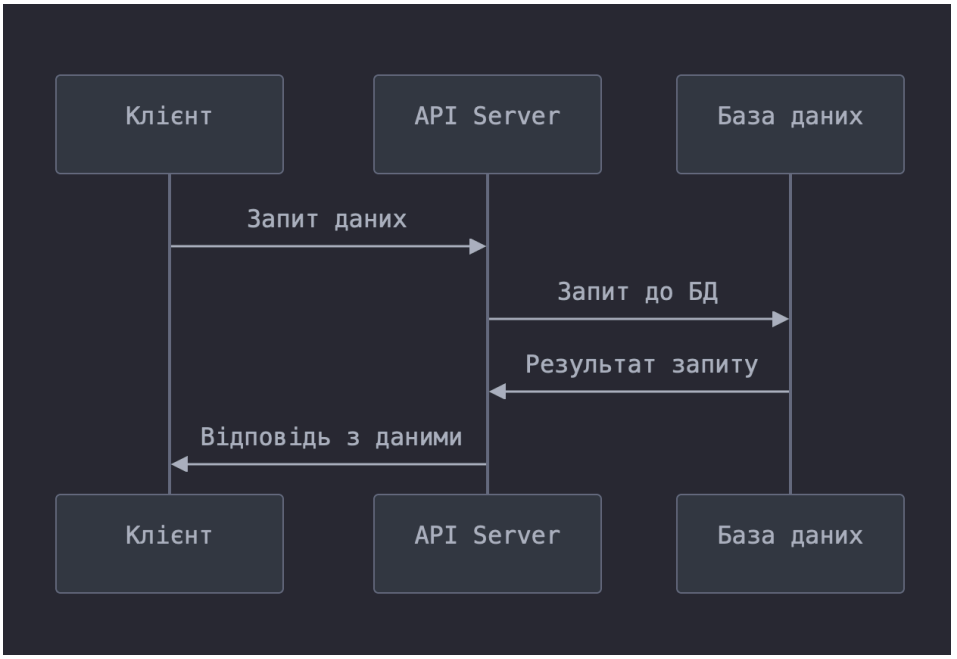

Рисунок 2.2. Діаграма потоку даних

Візуалізація даних реалізована за допомогою бібліотеки Rechart, яка дозволяє створювати інтерактивні графіки та діаграми для відображення статистики продажів, аналітики товарів та інших важливих метрик [20].

# РОЗДІЛ ІІІ
# СТВОРЕННЯ ІНФОРМАЦІЙНОЇ ПАНЕЛІ КЕРУВАННЯ ТОВАРАМИ В ІНТЕРНЕТ-МАГАЗИНІ

## 3.1 Створення фронтенд-частини інформаційної панелі керування товарами в інтернет-магазині на основі Next.js і Tailwind CSS

Фронтенд-частина інформаційної панелі керування товарами розроблена з використанням Next.js як фреймворку для React та Tailwind CSS для стилізації. Основний компонент панелі – Dashboard, який формує структуру сторінки та містить ключові віджети для відображення аналітики продажів, витрат і замовлень.

Розглянемо першим ділом головний файл для цієї сторінки – dashboard/page.tsx:

```
const Dashboard = () => {
  return (
      <CardPopularProducts />
      <CardSalesSummary />
      <CardPurchaseSummary />
      <CardExpenseSummary />
      <StatCard title="Customer & Expenses"
        ...
      <StatCard title="Dues & Pending Orders"
      <StatCard title="Sales & Discount"
      />
    </div>
  );
};

export default Dashboard;
```

Компонент Dashboard формує інформаційну панель керування товарами в інтернет-магазині, використовуючи Next.js і Tailwind CSS. Він містить адаптивну сітку з різними картками-аналітиками, такими як популярні товари, продажі, закупівлі та витрати. Окрім цього, використовуються картки StatCard, які відображають статистичні дані про клієнтів, заборгованості, очікуючі замовлення, продажі та знижки. Дані представлені у вигляді числових значень, відсоткових змін і відповідних іконок. Завдяки Tailwind

CSS компоненти адаптуються під різні екрани, забезпечуючи зручність роботи.

Далі на черзі компонента CardPopularProducts. Вона відповідає за відображення списку популярних товарів в інформаційній панелі керування. Він отримує дані через useGetDashboardMetricsQuery(), що дозволяє отримати аналітичну інформацію з бекенду.

Якщо дані ще завантажуються, компонент виводить текст “Loading…”. Після отримання даних відображається заголовок “Popular Products”, роздільна лінія та список товарів. Кожен товар представлений у вигляді окремого блоку з зображенням, назвою, ціною, рейтингом та кількістю проданих одиниць. Зображення товару вибирається випадковим чином із заздалегідь підготовлених файлів.

Для кожного товару передбачено кнопку із значком кошика, що позначає можливість покупки. Завдяки Tailwind CSS компонент має адаптивний дизайн, використовує тінь, закруглені кути та стилізовані кнопки, що покращує візуальне сприйняття і зручність користування.

Код компоненти наведено нижче:

```
const CardPopularProducts = () => {
  const { data: dashboardMetrics, isLoading } =
useGetDashboardMetricsQuery();
  return (
    ...
          <div className="overflow-auto h-full">
            {dashboardMetrics?.popularProducts.map((product) => (
                  ...
            ))}
          </div>
        </>
      ))}
    </div>
  );
};
```

Далі розглянемо компонент CardPurchaseSummary відповідає за відображення підсумкової інформації про закупівлі в інформаційній панелі керування. Він отримує дані через useGetDashboardMetricsQuery(), зокрема список закупівель (purchaseSummary).

Якщо дані ще завантажуються, виводиться повідомлення “Loading…”. Після успішного отримання даних компонент відображає заголовок “Purchase Summary” і горизонтальну лінію для візуального розділення. Основна частина містить загальну суму закупівель, яка відформатована у скороченому вигляді (наприклад, $10k). Поруч показується зміна у відсотках, яка позначається зеленим кольором та іконкою стрілки вгору, якщо значення зросло, або червоним кольором і стрілкою вниз, якщо зменшилося.

У нижній частині розміщено графік (AreaChart) із використанням recharts, що відображає динаміку закупівель за часом. По осі X знаходяться дати, а по осі Y – загальна сума закупівель. Наведення на точку графіка показує детальну інформацію про значення в цей день.

Код даної компоненти наведено нижче:

```
const CardPurchaseSummary = () => {
  ...
      {isLoading ? (
        <div className="m-5">Loading...</div>
      ) : (
          {/* HEADER */}
          ...
          {/* BODY */}
          ...
          {/* CHART */}
        </>
      )}
    </div>
  );
};
```

Компонент CardExpenseSummary відповідає за відображення підсумку витрат за категоріями у вигляді діаграми та текстових даних.

Він отримує інформацію про витрати через useGetDashboardMetricsQuery(), а якщо дані ще завантажуються, виводиться “Loading…”. Після завантаження формується заголовок “Expense Summary” із горизонтальною лінією. Основна частина містить кругову діаграму (PieChart з recharts), яка відображає розподіл витрат за категоріями. Витрати групуються за категоріями, суми конвертуються в числовий формат і

підсумовуються для кожної категорії. Центром діаграми розташовується загальна сума витрат, відформатована до двох знаків після коми.

Праворуч від діаграми є список категорій витрат із маркерами відповідного кольору. Внизу розміщений підсумковий блок, який відображає середні витрати та відсоткову зміну (з іконкою TrendingUp, що показує 30% зростання).

Код компоненти:

```
const CardExpenseSummary = () => {
  const { data: dashboardMetrics, isLoading } =
useGetDashboardMetricsQuery();
  const expenseSummary = dashboardMetrics?.expenseSummary[0];
  ...
  return (
    <div className="row-span-3 bg-white shadow-md rounded-2xl flex flex-col
justify-between">
    ...
    </div>
  );
};
```

Далі перейдемо до сторінки з витратами. Її код доволі великий, тому розглянемо його по частинах.

Компонент Expenses починається зі створення станів для керування вибраною категорією витрат (selectedCategory), діапазоном дат (startDate і endDate) та активним сегментом кругової діаграми (activeIndex). Дані отримуються через useGetExpensesByCategoryQuery(), а useMemo використовується для оптимізації доступу до масиву expenses.

Далі функція parseDate перетворює дату у формат YYYY-MM-DD, що спрощує порівняння. Основна логіка фільтрації та агрегації даних відбувається в useMemo, де спочатку фільтруються витрати за категорією та датою, а потім групуються по категоріях із підрахунком загальної суми. Для кожної категорії генерується випадковий колір, який використовується в діаграмі.

```
const parseDate = (dateString: string) => {
    const date = new Date(dateString);
    return date.toISOString().split("T")[0];
  };
const aggregatedData: AggregatedDataItem[] = useMemo(() => {
    const filtered: AggregatedData = expenses
      ...
    return Object.values(filtered);
  }, [expenses, selectedCategory, startDate, endDate]);
```

Компонент виконує перевірку стану завантаження даних, а також можливих помилок. Якщо дані ще не отримані (isLoading), користувач бачить повідомлення про процес завантаження, що запобігає відображенню некоректної або неповної інформації. У разі виникнення помилки (isError) виводиться відповідне повідомлення, яке допомагає користувачеві зрозуміти причину збою та за необхідності повторити запит.

Якщо дані успішно завантажені, компонент рендерить основний інтерфейс, що містить заголовок, опис і два ключові блоки: панель фільтрів та кругову діаграму.

Блок фільтрів забезпечує можливість гнучкого налаштування відображуваних даних. Він включає випадаючий список, що дозволяє вибрати конкретну категорію витрат, а також два поля введення для встановлення діапазону дат. Користувач може змінювати значення в цих полях, що призводить до оновлення відповідного стану компонента, автоматично викликаючи перерахунок та оновлення діаграми. Це дозволяє в режимі реального часу аналізувати дані за різними параметрами.

Кругова діаграма реалізована за допомогою бібліотеки recharts. Вона наочно відображає співвідношення витрат між категоріями у вигляді секторів, розмір яких відповідає обсягу витрат у кожній категорії. Додаткові інтерактивні елементи забезпечують зручність користування:

- При наведенні на сектор змінюється його колір, що покращує візуальне сприйняття та дозволяє легко ідентифікувати вибрану категорію.
- Використовується компонент Tooltip, який відображає детальну інформацію про сегмент, зокрема його назву та значення.
- Legend надає користувачеві пояснення кольорових маркерів, що робить діаграму більш зрозумілою.
- Подія onMouseEnter зберігає індекс активного сегмента, що дозволяє змінювати його стиль під час взаємодії.

Таким чином, компонент поєднує функціональність динамічних фільтрів із візуалізацією даних у зручному та інтуїтивному форматі, забезпечуючи користувачеві швидкий і ефективний аналіз витрат.

```
<div className="flex-grow bg-white shadow rounded-lg p-4 md:p-6">
  <ResponsiveContainer width="100%" height={400}>
    <PieChart>
      <Pie
        ...
      </Pie>
      <Tooltip />
      <Legend />
    </PieChart>
  </ResponsiveContainer>
```

Далі піде мова про сторінку з інвентарем. Компонент Inventory призначений для відображення списку товарів у вигляді таблиці. Він використовує useGetProductsQuery для отримання даних про продукти та керує станом завантаження та можливих помилок.

Якщо дані ще завантажуються, виводиться повідомлення "Loading...". У разі помилки або відсутності даних показується текст "Failed to fetch products".

```
const Inventory = () => {
  const { data: products, isError, isLoading } = useGetProductsQuery();
  if (isLoading) {
    return <div className="py-4">Loading...</div>;
  }
  if (isError || !products) {
    return (
      <div className="text-center text-red-500 py-4">
        Failed to fetch products
      </div>
    );
  }
```

Основний інтерфейс складається з заголовка Header та таблиці DataGrid з бібліотеки @mui/x-data-grid. Колонки таблиці визначені у масиві columns, де кожен об’єкт містить field (ключ у даних), headerName (назву колонки) та ширину. Деякі колонки мають додаткові параметри, як valueGetter для форматування значень, наприклад, додавання символу $ до ціни або відображення "N/A" для відсутнього рейтингу.

Компонент DataGrid отримує products як rows, ідентифікатором рядка визначено productId. Також включена можливість вибору рядків (checkboxSelection). Стилізація додає shadow, border і світлий фон для кращого вигляду в інтерфейсі.

І останньою буде компонента для відображення сторінки з товарами. Вона відображає список товарів з можливістю пошуку та додавання нових позицій. У верхній частині знаходиться рядок пошуку, який дозволяє фільтрувати товари за назвою. Введене значення зберігається в стані searchTerm і передається в useGetProductsQuery, що дозволяє отримати відфільтрований список. Заголовок Header відображає назву сторінки, а праворуч знаходиться кнопка створення нового товару, яка відкриває модальне вікно.

```
const Products = () => {
  const [searchTerm, setSearchTerm] = useState("");
  const [isModalOpen, setIsModalOpen] = useState(false);
  const {
    data: products, isLoading, isError,
  } = useGetProductsQuery(searchTerm);
  const [createProduct] = useCreateProductMutation();
  const handleCreateProduct = async (productData: ProductFormData) => {
    await createProduct(productData);
  };
```

Список товарів представлений у вигляді карток, кожна з яких містить випадкове зображення, назву, ціну, кількість на складі та рейтинг, якщо він є. Ціна форматована до двох знаків після коми, а рейтинг візуалізується через компонент Rating. Якщо товари ще завантажуються, виводиться повідомлення про завантаження. У разі помилки або відсутності даних з’являється повідомлення про невдале отримання товарів.

Коли користувач натискає кнопку створення товару, відкривається CreateProductModal, де можна ввести нові дані. Форма передає їх у handleCreateProduct, який викликає useCreateProductMutation для додавання товару в базу даних. Компонент забезпечує зручний інтерфейс для пошуку, перегляду та створення товарів, роблячи роботу з каталогом ефективною та інтуїтивно зрозумілою.

### 3.2 Створення бекенд-частини інформаційної панелі керування товарами в інтернет-магазині з використанням Node.js та Express.js

Основний файл бекенд-частини інформаційної панелі керування товарами в інтернет-магазині створений за допомогою Node.js та Express.js. У ньому налаштоване середовище, підключені необхідні middleware, а також визначені маршрути для різних частин системи.

На початку імпортуються необхідні модулі, включаючи express для створення веб-сервера, dotenv для роботи зі змінними середовища,

body-parser для обробки JSON-даних, cors для забезпечення безпечних CORS-запитів, helmet для підвищення безпеки додатку, а також morgan для логування HTTP-запитів. Далі йдуть імпорти файлів з маршрутами, що відповідають за різні частини системи: інформаційну панель (dashboardRoutes), товари (productRoutes), користувачів (userRoutes) і витрати (expenseRoutes).

```
import express from "express";
import dotenv from "dotenv";
import bodyParser from "body-parser";
import cors from "cors";
import helmet from "helmet";
import morgan from "morgan";
import dashboardRoutes from "./routes/dashboardRoutes";
import productRoutes from "./routes/productRoutes";
import userRoutes from "./routes/userRoutes";
import expenseRoutes from "./routes/expenseRoutes";
```

Далі йде конфігурація додатку. Завантажуються змінні середовища, ініціалізується express-додаток, підключаються middleware: helmet для захисту заголовків HTTP, morgan для логування, bodyParser для обробки JSON та URL-encoded даних, cors для забезпечення доступу до API з інших доменів.

Після цього налаштовуються маршрути. app.use використовується для обробки запитів до певних шляхів. Наприклад, app.use("/dashboard", dashboardRoutes) означає, що всі запити на /dashboard будуть передані у dashboardRoutes, аналогічно працюють маршрути для товарів, користувачів і витрат.

Наприкінці створюється сервер, який слухає на вказаному порту, що визначається через змінну середовища PORT, або за замовчуванням 3001. При запуску виводиться повідомлення в консоль про успішний запуск сервера.

```
/* SERVER */
const port = Number(process.env.PORT) || 3001;
app.listen(port, "0.0.0.0", () => {
  console.log(`Server running on port ${port}`);
});
```

Далі розглянемо контролери для шляхів які ми імпортували.

Контролер DashboardController виконує функцію збору та підготовки ключових метрик для інформаційної панелі управління товарами в інтернет-магазині. Він дозволяє отримати найактуальніші дані про залишки на складі, останні фінансові операції та загальні витрати за категоріями.

На початку імпортуються модулі Request та Response з express, які використовуються для типізації вхідних запитів та вихідних відповідей. Це дозволяє підвищити безпеку та надійність коду, оскільки забезпечує правильне передавання та отримання даних у межах контролера. Також імпортується PrismaClient з бібліотеки @prisma/client, який використовується для взаємодії з базою даних через ORM Prisma.

Перед виконанням запитів до бази даних створюється екземпляр PrismaClient. Він є основним об’єктом для взаємодії з базою, забезпечуючи доступ до таблиць та дозволяючи виконувати запити на отримання, додавання, оновлення та видалення даних.

Контролер getDashboardMetrics є асинхронною функцією, яка виконує одразу кілька запитів до бази даних. Її основне завдання — отримати набір метрик для інформаційної панелі, що відображає дані про популярні товари, останні операції та загальні витрати за категоріями. Спочатку виконується запит до бази на отримання 15 найпопулярніших товарів, які сортуються за зменшенням кількості на складі. Далі отримуються останні 5 записів про продажі, закупівлі та витрати, що сортуються за датою в порядку спадання.

Це дозволяє швидко отримати інформацію про останні фінансові операції, що були здійснені в магазині.

Окремо виконується запит на отримання зведеної інформації про витрати за категоріями, також відсортованої за датою. Оскільки значення суми витрат (amount) може бути представлене у вигляді числового типу, перед відправленням відповіді воно перетворюється в рядковий формат для збереження узгодженості даних.

У разі успішного виконання всіх запитів функція формує JSON-відповідь, що містить отримані метрики. Якщо під час виконання запитів виникає помилка, вона обробляється за допомогою блоку catch, і клієнту повертається відповідь з кодом 500 та повідомленням про внутрішню помилку сервера. Це забезпечує коректне опрацювання можливих помилок і запобігає некоректному відображенню даних на інформаційній панелі.

```
    });
    const salesSummary = await prisma.salesSummary.findMany({...});
    const purchaseSummary = await prisma.purchaseSummary.findMany({...});
    const expenseSummary = await prisma.expenseSummary.findMany({...});
    const expenseByCategorySummaryRaw = await
prisma.expenseByCategory.findMany({...});
    const expenseByCategorySummary = expenseByCategorySummaryRaw.map(...);
    res.json({
      popularProducts,
      ...
      expenseByCategorySummary,
    });
  } catch (error) {
    res.status(500).json({ message: "Error retrieving dashboard metrics"
});
  }
};
```

У разі успішного виконання запитів функція повертає відповідь у форматі JSON, що містить всі отримані метрики. Якщо виникає помилка, сервер повертає статус 500 та повідомлення про помилку під час отримання даних.

Контролер ExpensesController обробляє запити, пов’язані з отриманням витрат за категоріями. Він використовує Prisma для взаємодії з базою даних. Спочатку імпортуються необхідні модулі: Request і Response з express для типізації HTTP-запитів і відповідей, а також PrismaClient з @prisma/client. Створюється екземпляр PrismaClient, що дозволяє виконувати запити до бази.

Асинхронна функція getExpensesByCategory отримує список витрат, згрупованих за категоріями, і сортує їх у порядку спадання за датою. Дані витрат спочатку отримуються у вигляді масиву об’єктів з таблиці expenseByCategory. Після цього відбувається обробка кожного елемента масиву, де значення amount, що може бути числовим, конвертується в рядковий формат, щоб забезпечити узгодженість даних перед відправкою у відповідь.

```
export const getExpensesByCategory = async (
  req: Request,
  res: Response
): Promise<void> => {
  try {
    const expenseByCategorySummaryRaw = await
prisma.expenseByCategory.findMany(
      ...
    );
    const expenseByCategorySummary = expenseByCategorySummaryRaw.map(
      ...
    );
    res.json(expenseByCategorySummary);
```

Контролер ProductController відповідає за обробку запитів, пов’язаних із товарами. Використовується PrismaClient для взаємодії з базою даних.

Функція getProducts отримує список товарів із фільтрацією за ім’ям. Вона приймає параметр search із запиту, який використовується у where для пошуку товарів, що містять відповідний рядок у назві. У разі успішного

виконання повертається масив знайдених товарів. Якщо виникає помилка, повертається статус 500 з повідомленням про помилку.

```
export const getProducts = async (
  req: Request,
  res: Response
): Promise<void> => {
  try {
    const search = req.query.search?.toString();
    const products = await prisma.products.findMany({
      ...
    });
    res.json(products);
  } catch (error) {
    res.status(500).json({ message: "Error retrieving products" });
  }
};
```

Функція createProduct обробляє створення нового товару. Вона отримує з тіла запиту productId, name, price, rating та stockQuantity, після чого створює запис у таблиці products. Якщо операція успішна, сервер повертає статус 201 та створений товар. У разі помилки повертається статус 500 із відповідним повідомленням.

Останнім на черзі є контролер UserController містить функцію getUsers, яка відповідає за отримання списку користувачів.

Функція getUsers виконує запит до бази даних за допомогою prisma.users.findMany(), щоб отримати всіх користувачів. Якщо запит успішний, вона повертає масив користувачів у форматі JSON.

### 3.3. Об’єднання фронтенд і бекенд частин інформаційної панелі керування товарами в інтернет-магазині

Для запуску додатку спочатку перейдемо в папку з сервером і виконаємо наповнення бази даних командою:

```
npm run seed
```

Після чого можна запустити сервер:

```
npm run dev
```

І отримаємо наступне в консолі підтвердження що сервер запущено (рис. 3.1).


Рисунок 3.1. Запущений сервер

Далі можемо перейти в папку з клієнтом, зібрати його і запустити наступною командою:

```
npm run build

npm run dev
```

Після чого побачимо в консолі повідомлення про успішно запущений клієнт (рис. 3.2)


Рисунок 3.2. Запущений клієнт

### 3.4 Тестування інформаційної панелі керування товарами в інтернет-магазині та аналіз її продуктивності

Після запуску серверу і клієнта відкриємо посилання http://localhost:3000 в браузері. Ми побачимо головну сторінку нашої інформаційної панелі

(рис. 3.3).

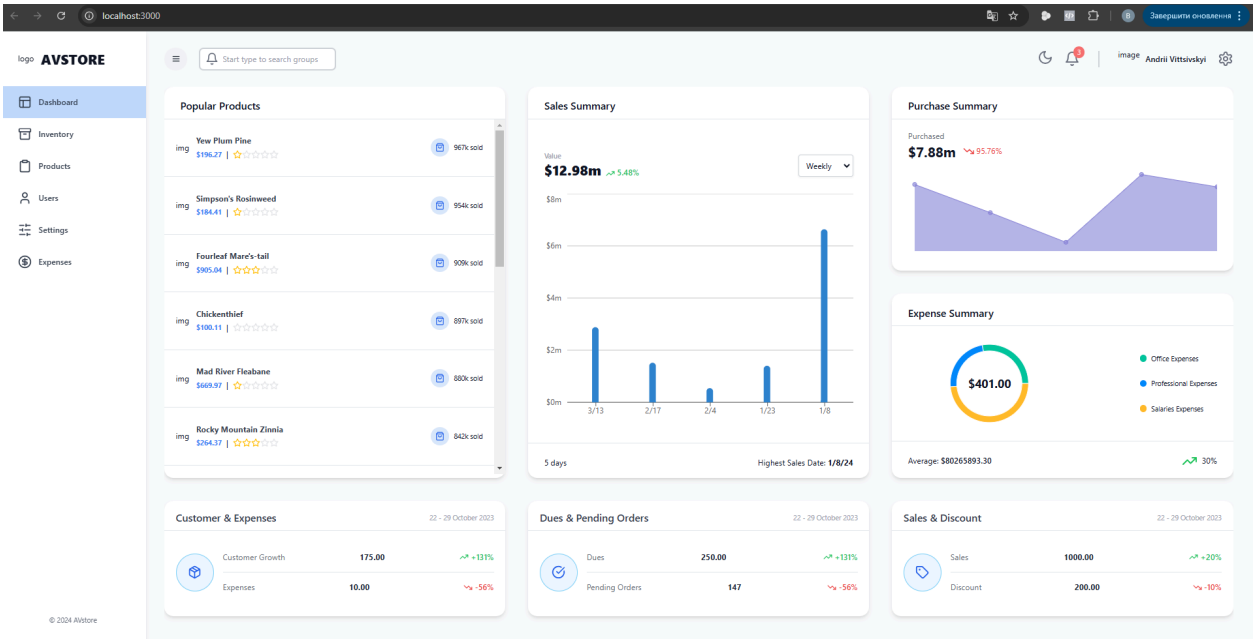


Рисунок 3.3. Головна сторінка інформаційної панелі

Переглянемо інвентар на відповідній вкладці (рис. 3.4).

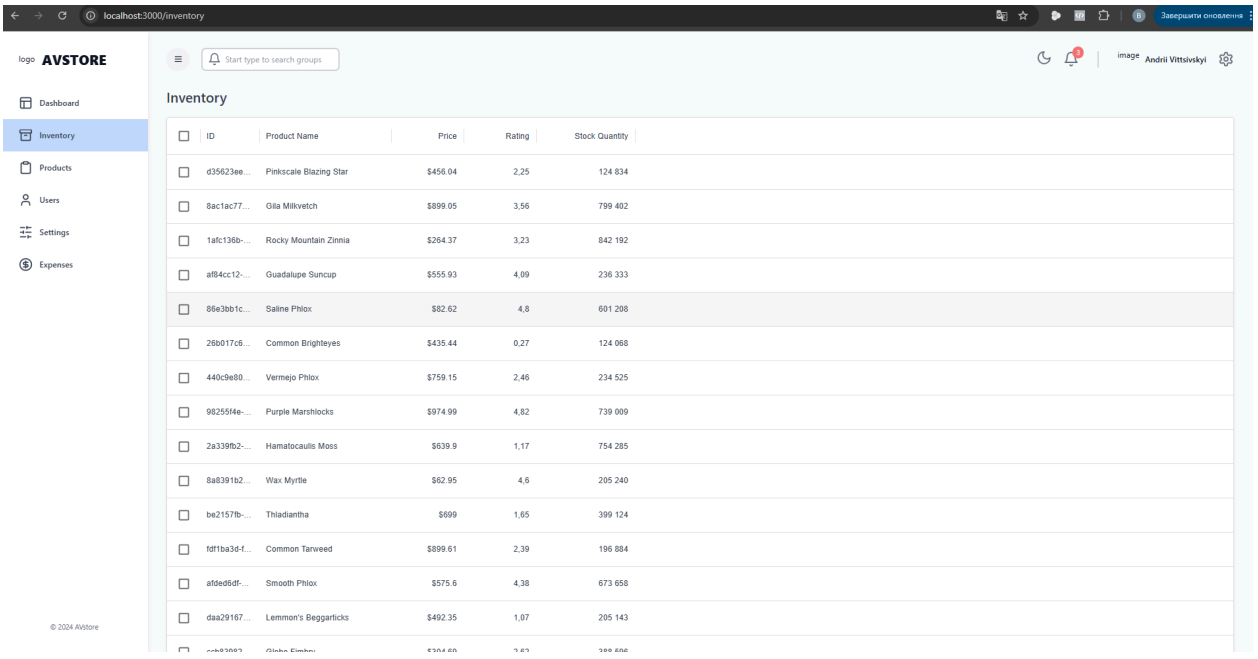


Рисунок 3.4. Сторінка з інвентарем

Далі перейдемо на сторінку з товарами (рис. 3.5)

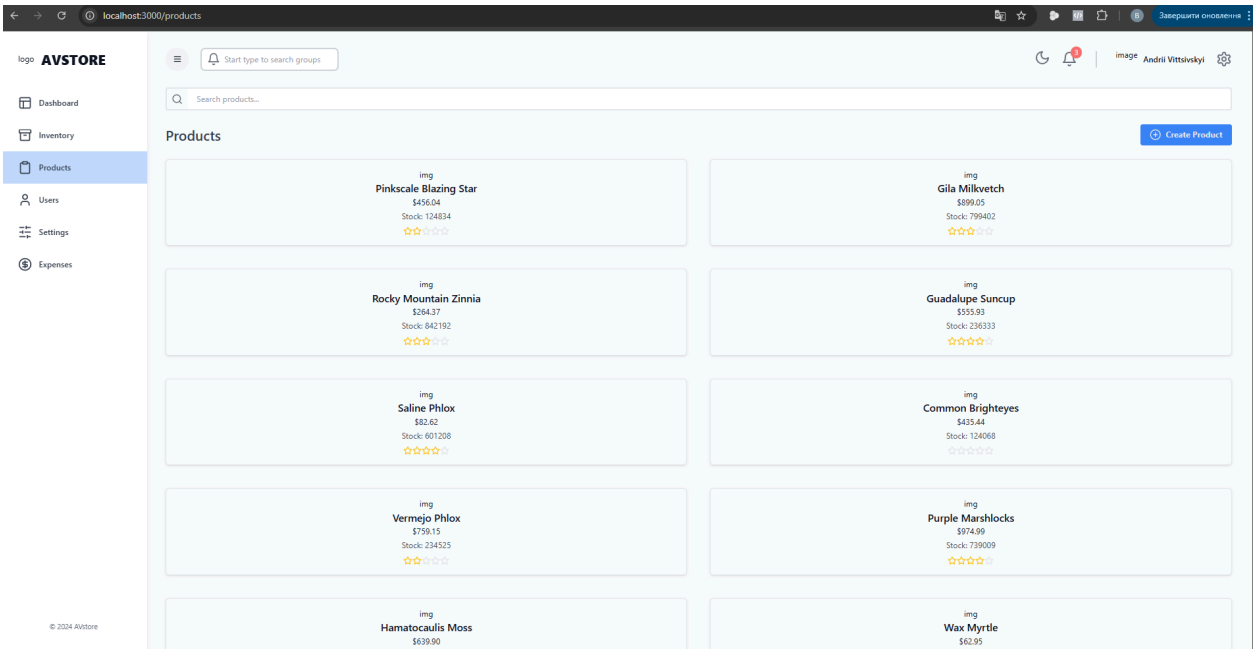


Рисунок 3.5. Сторінка з товарами інформаційної панелі

Протестуємо створення товару (рис. 3.6).

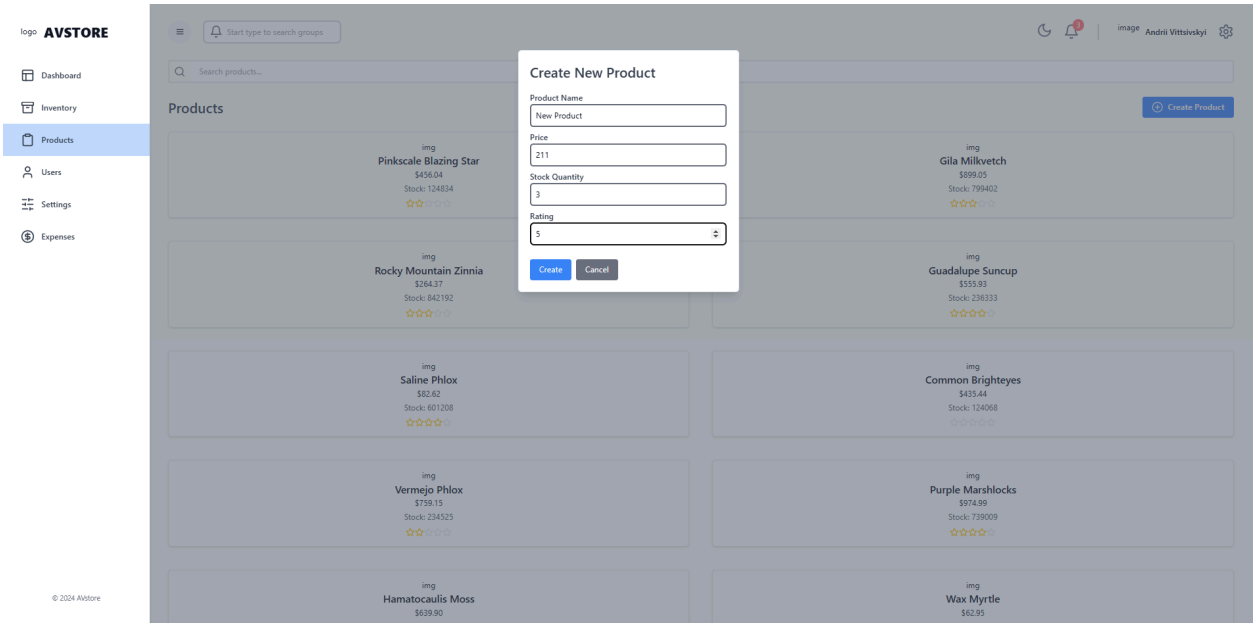


Рисунок 3.6. Спливаюче вікно створення товару

Після натискання на кнопку “Create” перевіримо чи товар додався (рис.3.7).

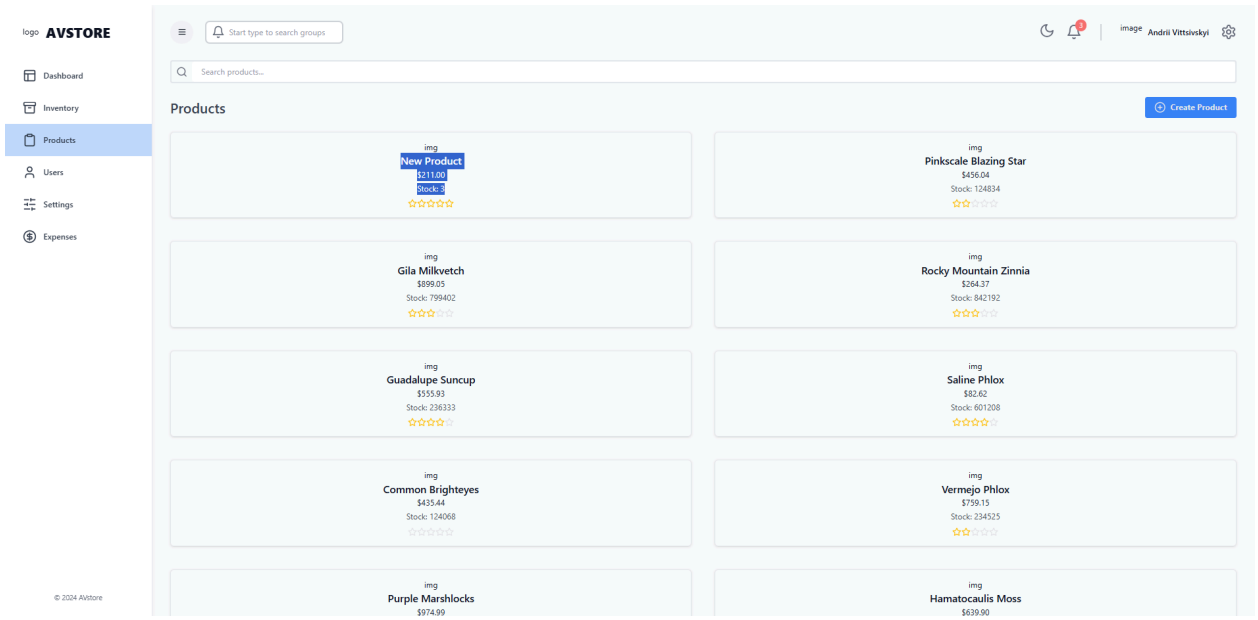


Рисунок 3.7. Перевірка чи товар додався

Також перейдемо на сторінку з усіма користувачами (рис. 3.8).

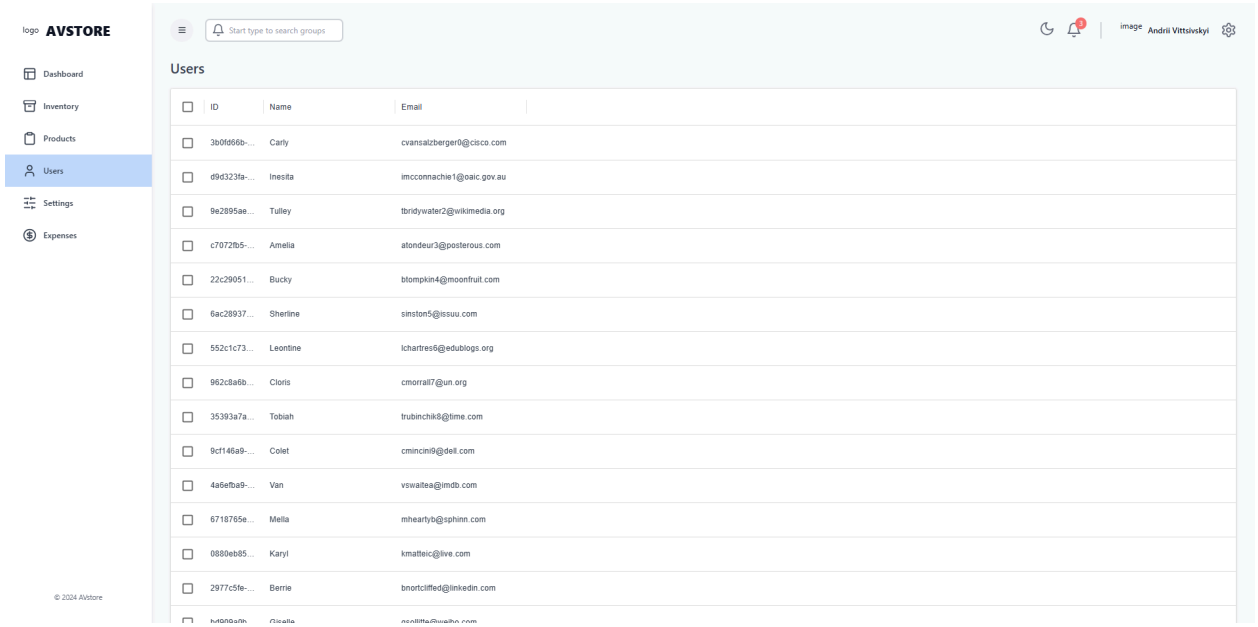


Рисунок 3.8. Сторінка з користувачами

Наступною переглянемо сторінку з налаштуваннями акаунту (рис. 3.9).

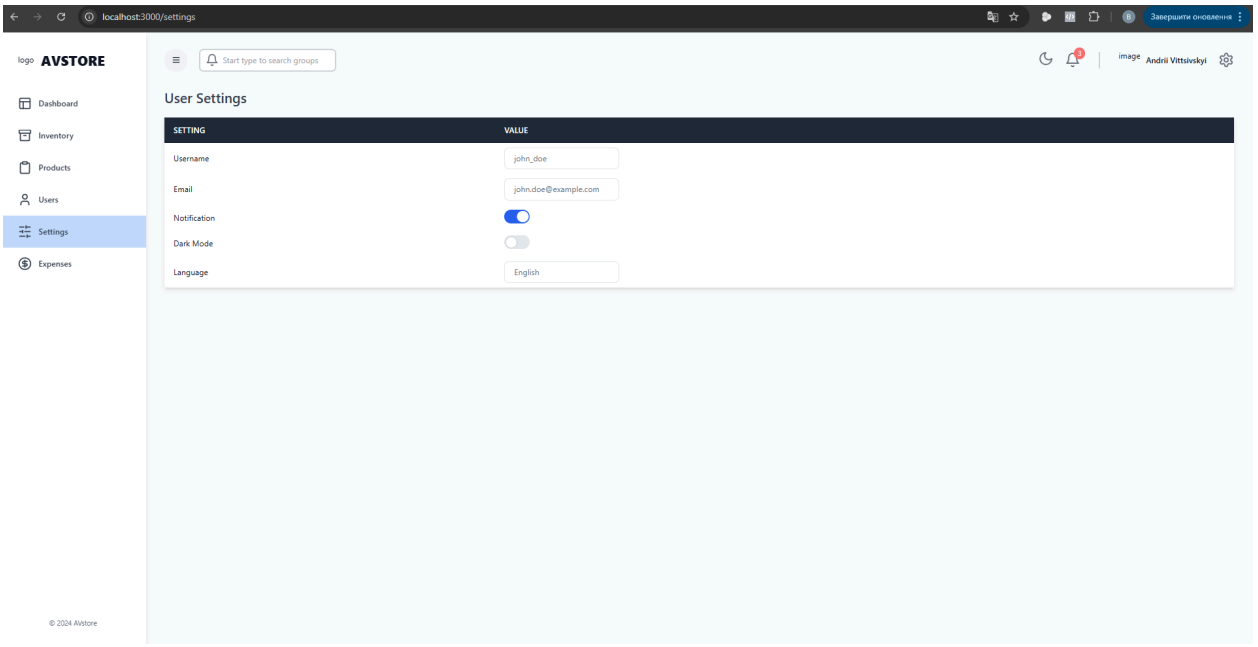


Рисунок 3.9. Сторінка з налаштуваннями

І останньою сторінкою є інформація про витрати на відповідній сторінці (рис. 3.10).

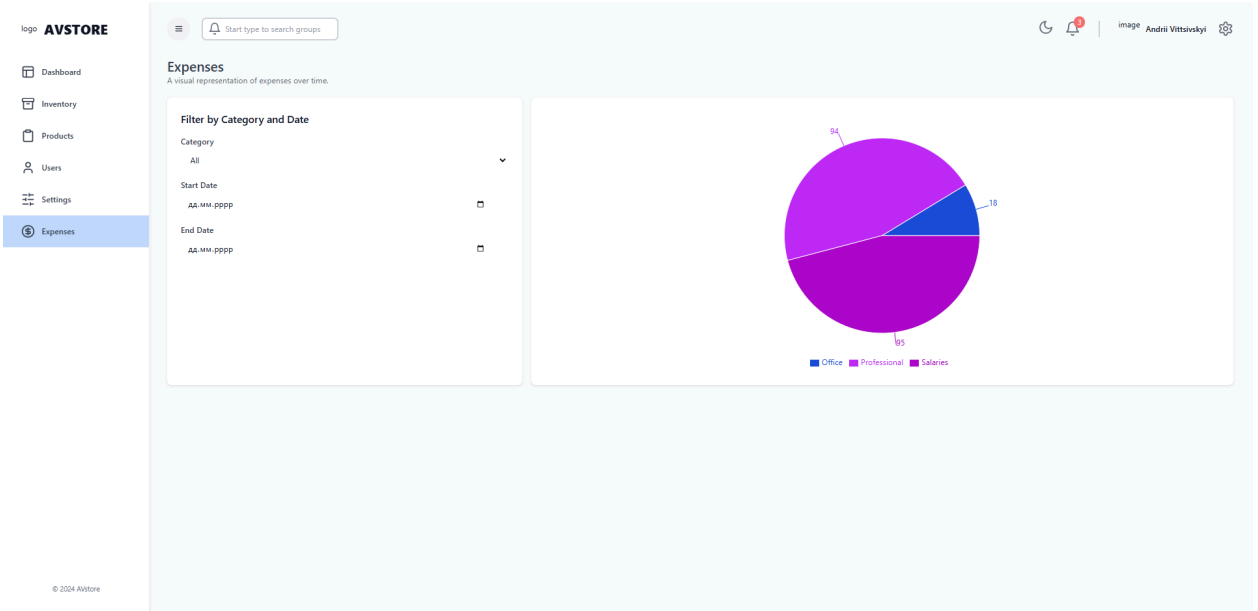


Рисунок 3.10. Сторінка з витратами онлайн магазину

Під час аналізу продуктивності інформаційної панелі керування товарами в інтернет-магазині було проведено тестування часу відповіді API, навантажувальні випробування та оптимізацію бази даних для підвищення ефективності роботи сервера.

Було виміряно швидкість виконання запитів до основних маршрутів, таких як /products, /users, /dashboard та /expenses. В середньому, прості запити без додаткових параметрів оброблялися за 50–100 мс, тоді як запити з фільтрацією або агрегацією даних займали трохи більше часу – до 200 мс. Запити на створення нових записів у базі виконувалися стабільно, але при збільшенні кількості одночасних операцій спостерігалося незначне зростання затримки.

Навантажувальні тести показали, що сервер здатний обробляти до 500 одночасних запитів без значного зниження продуктивності. Однак при ще більшому навантаженні зросла середня затримка відповіді, що свідчить про необхідність масштабування при великому потоці користувачів. Під час стрес-тестування сервер витримував навантаження у 1000 запитів на секунду протягом 10 хвилин, після чого починали з’являтися періодичні таймаути.

Для покращення продуктивності бази даних було додано індекси до полів, що найчастіше використовуються у фільтрації, таких як name у таблиці продуктів та date у фінансових записах. Це дозволило зменшити час виконання складних запитів на 30–40%. Також було оптимізовано структуру SQL-запитів, щоб уникнути вибірки зайвих даних.

Загалом, після оптимізації продуктивність панелі покращилася, середній час відповіді на запити зменшився, а сервер зміг стабільніше витримувати велике навантаження.

# ВИСНОВКИ

У ході виконання дипломної роботи було створено інформаційну панель керування товарами для інтернет-магазину, яка забезпечує зручний та ефективний механізм управління товарним асортиментом, контролю за продажами та аналізу витрат. Розроблене рішення дозволяє адміністраторам інтернет-магазину в реальному часі отримувати актуальні дані про товари, відстежувати фінансові показники та оптимізувати процеси, пов’язані з обліком продукції.

На початковому етапі було проведено детальний аналіз існуючих інформаційних панелей для управління товарами в електронній комерції. Було розглянуто сучасні підходи до побудови таких систем, а також визначено ключові критерії ефективності, включаючи швидкість обробки запитів, зручність користування, безпеку даних і масштабованість. З огляду на отримані результати було обґрунтовано вибір технологічного стеку, що відповідав поставленим вимогам.

Фронтенд-частина інформаційної панелі була реалізована за допомогою Next.js у поєднанні з Tailwind CSS, що дозволило створити швидкий, продуктивний та адаптивний інтерфейс. Використання Next.js забезпечило ефективну серверну генерацію сторінок (SSR) і статичну оптимізацію, що покращило продуктивність та швидкість завантаження інтерфейсу. Tailwind CSS сприяв прискоренню процесу розробки завдяки своїй утилітарній системі стилізації, що дозволило уникнути зайвого коду та зробити дизайн більш структурованим.

Бекенд-система була побудована на основі Node.js з використанням Express.js, що забезпечило високу продуктивність сервера та можливість гнучкого розширення функціоналу. Для роботи з базою даних використовувалася Prisma ORM, що значно спростило взаємодію з базою даних, забезпечило безпеку запитів та підвищило швидкість обробки інформації. Реалізована архітектура дозволяє швидко масштабувати систему відповідно до зростаючих потреб інтернет-магазину.

У процесі тестування було проведено комплексну перевірку функціональності інформаційної панелі, включаючи модульне та інтеграційне тестування, що дозволило виявити та виправити потенційні помилки. Додатково був проведений аналіз продуктивності сервера та бази даних під навантаженням, що дозволило оптимізувати виконання запитів і покращити швидкість роботи панелі.

Розроблена інформаційна панель відповідає сучасним вимогам до вебзастосунків, забезпечує стабільну та безпечну роботу, а також пропонує гнучкі можливості для подальшого розширення. Завдяки використанню сучасних технологій вона може бути легко інтегрована з іншими модулями інтернет-магазину, а також адаптована до потреб користувачів.

У підсумку створене рішення дозволяє адміністраторам інтернет-магазину ефективно керувати товарами, оптимізувати процеси та приймати обґрунтовані рішення на основі актуальних даних, що сприяє підвищенню продуктивності бізнесу та покращенню загального користувацького досвіду.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Бенкс Алекс. React і Redux. Функціональна веб-розробка. – Київ: Видавництво Старого Лева, 2019. – 352 с.
2. Бріггс Джейсон. Пайтон для дітей: веселе знайомство з програмуванням. – Київ: Видавництво Старого Лева, 2016. – 344 с.
3. В Машкіна, ВО Абрамов, ТІ Носенко, ЄВ Іваніченко.Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп'ютерні науки для здобувачів вищої освіти денної форми навчання, Київський столичний університет імені Бориса Грінченка. Київ, 2024.- 43 с.
4. Теоретичні та практичні аспекти використання математичних методів та інформаційних технологій в освіті й науці:моногр. / за заг. ред. О. Литвин. — К.: Київ. ун-т ім. Б. Грінченка,2021. — 332 с.
5. Яскевич, Владислав Олександрович (2023) *JavaScriptбібліотека React Навчальний посібник для студентів спеціальності Комп'ютерні науки* Київський університет імені Бориса Грінченка, Україна, Київ. https://elibrary.kubg.edu.ua/id/eprint/48587/
6. Дакетт Джон. HTML та CSS. Розробка та дизайн веб-сайтів. – Київ: Видавництво Старого Лева, 2014. – 512 с.
7. Крокфорд Дуглас. JavaScript: сильні сторони. – Київ: Видавництво Старого Лева, 2015. – 176 с.
8. Мардан Азат. Express.js: глибоке занурення. – Київ: Видавництво Старого Лева, 2017. – 456 с.
9. Мерфі Нейл. Архітектура програмного забезпечення. – Київ: Видавництво Старого Лева, 2020. – 624 с.
10. Сівілла Джон. Програмування на JavaScript. – Київ: Видавництво Старого Лева, 2018. – 480 с.
11. Сімпсон Кайл. Ви не знаєте JS: Скопи та замикання. – Київ: Видавництво Старого Лева, 2015. – 98 с.

12. Стивенсон Етан. Full-Stack React, TypeScript і Node. – Київ: Видавництво Старого Лева, 2021. – 600 с.
13. Фрімен Адам. ASP.NET Core MVC 2 з прикладами для професіоналів. – Київ: Видавництво Старого Лева, 2018. – 1024 с.
14. Фленаган Девід. *JavaScript. Досконале керівництво.* – Київ: Видавництво Старого Лева, 2021. – 800 с.
15. Front end back end різниця: як зробити вибір та почати вивчати. Foxminded. URL: https://foxminded.ua/front-end-back-end-riznytsia/ (дата звернення: 23.02.2025).
16. Next.js для створення веб-додатків: особливості та переваги. Next.js Docs. URL: https://nextjs.org/docs (дата звернення: 23.02.2025).
17. Node.js для бекенду: огляд основних можливостей. Node.js Docs. URL: https://nodejs.org/en/docs/ (дата звернення: 23.02.2025).
18. React та Next.js у створенні динамічних веб-додатків. React Docs. URL: https://react.dev/ (дата звернення: 23.02.2025).
19. REST API: принципи розробки та інтеграції. REST API Guide. URL: https://restfulapi.net/ (дата звернення: 23.02.2025).
20. Tailwind CSS: стильове оформлення для швидкої розробки. Tailwind CSS. URL: https://tailwindcss.com/docs (дата звернення: 23.02.2025).
21. Швидке розгортання інтернет-магазину з використанням Vercel. Vercel Docs. URL: https://vercel.com/docs (дата звернення: 23.02.2025).
22. Як використовувати Prisma ORM для роботи з базами даних. Prisma Docs. URL: https://www.prisma.io/docs (дата звернення: 23.02.2025).
23. Admin Panels Design Course. Designly. URL: https://www.designly.school/admin-panels-design (дата звернення: 23.02.2025).
24. Building a full-stack e-commerce web app. URL: https://dev.to/webdynamics/building-a-full-stack-e-commerce-web-app-4i45 (дата звернення: 20.02.2025).
25. Content Management System: What It Is and How to Translate Content. Linguise. URL:

https://www.linguise.com/en/blog/guides/content-management-system-what-it-is-and-how-to-translate-content/ (дата звернення: 21.02.2025).

26. Express.js Tutorial. URL: https://www.geeksforgeeks.org/express-js/ (дата звернення: 21.02.2025).
27. Express.js: фреймворк для швидкої розробки бекенду. Express.js Docs. URL: https://expressjs.com/en/guide/routing.html (дата звернення: 23.02.2025).
28. How to Build a Fullstack App with Next.js, Prisma, and Vercel Postgres. URL: https://vercel.com/guides/nextjs-prisma-postgres (дата звернення: 21.02.2025).
29. How to Manage Products. SendPulse. URL: https://sendpulse.com/knowledge-base/crm/ecommerce/manage-products (дата звернення: 23.02.2025).
30. Part 1: From 0 to 1 – Creating a Full-Stack Web Application Platform from Scratch and Operating it for (almost) Free. URL: https://medium.com/@isaac.adams/part-1-from-0-to-1-creating-a-full-stack-web-application-platform-from-scratch-and-operating-it-cd56ed8c2b0a (дата звернення: 20.02.2025).
31. Tailwind CSS React – Flowbite. URL: https://flowbite.com/docs/getting-started/react/ (дата звернення: 23.02.2025).
32. The easiest way to work with a database in Next.js. URL: https://www.prisma.io/nextjs (дата звернення: 21.02.2025).
33. What is Prisma ? How to use it with NextJS? URL: https://yashpurani.medium.com/what-is-prisma-how-to-use-it-with-nextjs-9742f4103d9 (дата звернення: 21.02.2025).