МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Аналітичний сервіс для вибору і купівлі товарів**

Виконав: студент 4 курсу, групи ПЗС-121-9
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
К. В. Береговой
(ініціали та прізвище)
Керівник доцент А.І. Безверхий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»
П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____перший (бакалаврський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т. В. Критська
"_01_" __березня_____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Береговому Кірілу Вікторовичу
(прізвище, ім'я, по батькові)
1.Тема роботи __Аналітичний сервіс для вибору і купівлі товарів___

керівник роботи _____Безверхий Анатолій Ігорович, доцент_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _____

2. Строк подання студентом кваліфікаційної роботи_____15.05.2023_____
3. Вихідні дані бакалаврської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми порівняння цін та пропозицій з різних джерел;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- розробка висновків та пропозицій.

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

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	15.05-16.05.23	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	16.05-17.05.23	виконано
3	Аналіз існуючих аналогів	17.05-18.05.23	виконано
4	Розробка структури бази даних для зберігання даних	18.05-19.05.23	виконано
5	Узгодження подальших дій з науковим керівником	19.05-20.05.23	виконано
6	Налаштування стартових CRUD ендпоінтів для роботи з таблицями бази даних	20.05-21.05.23	виконано
7	Розробка ефективного алгоритму зі скачування даних	22.05-25.05.23	виконано
8	Розробка API для роботи фронтенду	25.05-26.05.23	виконано
9	Розбиття бекенду на окремі пакети	26.05-27.05.23	виконано
10	Аналіз існуючого бекенду та узгодження подальшої розробки фронтенду з науковим керівником	27.05-28.05.23	виконано
11	Створення дизайну майбутнього веб застосунку	28.05-31.06.23	виконано
12	Розробка стартового проекту та налагодження взаємодії з даними через API	02.06-03.06.23	виконано
13	Реалізація функціональності фронтенду та верстка всіх необхідних сторінок	03.06-07.06.23	виконано
14	Остаточна перевірка працездатності всіх складових	07.06-08.06.23	виконано
15	Створення презентації	08.06-09.06.23	виконано
16	Оформлення звіту	09.06-14.06.23	виконано

Студент _____ К.В. Береговой
(підпис) (прізвище та ініціали)

Керівник роботи _____ А.І. Безверхий
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено
Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 76,

Рисунків — 25,

Джерел — 18.

Береговой К. В. Аналітичний сервіс для вибору і купівлі товарів: кваліфікаційна робота бакалавра спеціальності 121 «Програмне забезпечення систем» / наук. керівник А. І. Безверхий. Запоріжжя : ЗНУ, 2023. 76 с.

У сучасних умовах, для забезпечення ефективного вибору та оптимального придбання товарів, сервіси, що здійснюють порівняння цін, виявляються цінним ресурсом. Автоматизація процесу порівняння цін стає ключовим чинником у покращенні його ефективності.

Мета проекту — здійснення порівняння цін у різних магазинах та подальше надання користувачам інформації про доступні товари з метою допомогти визначити найбільш вигідне місце для їх придбання. Основним завданням проекту є автоматизація процесу порівняння цін та покращення досвіду покупців.

Проведено аналіз існуючих успішних систем порівняння цін. На основі цього аналізу розроблена інформаційна система для порівняння цін та рекомендації найвигіднішого магазину. Реалізований веб-додаток дозволяє користувачам вибирати певні товари та отримувати рекомендації про найдешевші магазини для їх покупки. Мета системи — підвищити ефективність покупок та допомогти користувачам економити гроші.

Ключові слова: *порівняння цін, рекомендація магазину, товари, ефективність покупок, економія, веб-додаток, інформаційна система*

ABSTRACT

Pages — 76,

Drawings — 25,

Sources — 18.

Berehovoy K. V. Analytical service for the selection and purchase of goods: qualification thesis of the bachelor of specialty 121 "System software" / Science. manager A. I. Bezverkhy. Zaporizhzhia: ZNU, 2023. 76 p.

In today's conditions, to ensure effective selection and optimal purchase of goods, price comparison services are a valuable resource. Automation of the price comparison process becomes a key factor in improving its efficiency.

The goal of the project is to compare prices in different stores and further provide users with information about available products in order to help determine the most profitable place to purchase them. The main objective of the project is to automate the price comparison process and improve the customer experience.

An analysis of existing successful price comparison systems was carried out. Based on this analysis, an information system was developed for comparing prices and recommending the most profitable store. The implemented web application allows users to select certain products and get recommendations on the cheapest stores to buy them. The goal of the system is to improve shopping efficiency and help users save money.

Keywords: *price comparison, store recommendation, products, shopping efficiency, savings, web application, information system*

ЗМІСТ

ВСТУП

Актуальність теми

Порівняння цін та рекомендації магазину актуальні у сучасному торговому середовищі. Споживачі шукають зручні способи порівняти ціни та вибрати оптимальний варіант покупки. Розробка сервісів порівняння цін допомагає споживачам економити час та гроші, а також знаходити найкращі пропозиції та угоди. Це актуально як покупців, так бізнесів, які прагнуть залучити клієнтів своїми конкурентними цінами.

Технологічний прогрес та доступ до Інтернету значно спрощують процес порівняння цін. Користувачі можуть швидко та зручно отримувати інформацію про ціни на різні товари та послуги, а також порівнювати пропозиції різних магазинів чи постачальників. Це дозволяє їм приймати усвідомлені рішення та здійснювати покупки, ґрунтуючись на найкращому співвідношенні ціни та якості.

Сервіси порівняння цін є також корисними інструментами для бізнесів. Вони дозволяють компаніям відстежувати конкуренцію та аналізувати ринкові тенденції. Це допомагає оптимізувати стратегії ціноутворення, адаптуватися до потреб ринку та залучати більше клієнтів.

Застосування інформаційних технологій дозволяє зменшити витрати на обробку, передачу та зберігання інформації, а також пропонує постійно товари та послуги, список яких постійно оновлюється.

В сучасному світі обробка інформації відіграє важливу роль в багатьох галузях. Для полегшення цього процесу створюються інформаційні системи, які об'єднують бази даних, інформаційні технології та технічні засоби. Їх функції включають надійне зберігання інформації, виконання специфічних перетворень та обчислень, а також надання зручного інтерфейсу користувачам.

Інформаційні системи використовуються в різних галузях, таких як банківська справа, бухгалтерія, авіація, залізниця, податкова служба,

статистика, готельний бізнес та багато інших. Ці системи зазвичай опрацьовують великі обсяги складної структурованої інформації.

Мета дослідження

Розробка інтелектуальної інформаційної системи порівняння та аналізу торгівельних пропозицій в мережах супермаркетів.

Завдання дослідження

Розглянути особливості та існуючі рішення аналітичних порівняльних сервісів. Проаналізувати існуючі підходи та технології до розробки складних інформаційних систем та вибрати відповідну для реалізації інтелектуальної інформаційної системи порівняння товарів.

Об’єкт дослідження

Об’єктом дослідження є процес розробки інтелектуальної інформаційної системи порівняння товарів.

Предмет дослідження

Предметом дослідження є технології створення власного застосунку для задоволення потреб клієнтів супермаркетів та бізнесів які хочуть відстежувати пропозиції своїх конкурентів або планують заощаджувати на купівлях.

Методи дослідження

Теоретичні — аналіз і порівняння технологій створення інформаційних систем; види класифікації та принципів проєктування інформаційних систем; порівняльний аналіз для вибору програмного забезпечення.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у тому щоб володіти інформацією про товари та зберігати її у себе з метою

подальшого використання. Результати дослідження можуть бути використані для створення аналітичної інформаційної системи з порівняння цінових пропозицій різних торгівельних мереж, з метою заощадження коштів або визначення перспективності цінової політики в порівнянні з конкурентами.

Глосарій

Система порівняння цін на товари — Інформаційна система, розроблена для збирання, зберігання та надання інформації про ціни на товари з різних магазинів з метою полегшити процес порівняння цін для користувачів.

Товар — Фізичний продукт, доступний для придбання чи споживання, та підлягає порівнянню цін у межах системи.

Магазин — Торгова організація чи платформа, де пропонуються товари для продажу.

Ціна — Грошова вартість товару чи послуги, вказана продавцем.

Інформаційна система — Сукупність програмного забезпечення, апаратних засобів та даних, призначених для збирання, зберігання, обробки та надання інформації.

Інтерфейс користувача — Графічний або текстовий інтерфейс, що надає можливість взаємодії користувача з інформаційною системою.

База даних — Структуроване сховище даних, що використовується для зберігання та організації інформації про товари, магазини та ціни.

Бекенд — частина системи, яка працює в позаду сцени і забезпечує логіку, обробку даних та взаємодію з базою даних. Він відповідає за обробку запитів користувача, збереження та витягування даних, виконання бізнес-логіки та інші операції, необхідні для правильної роботи системи.

Фронтент — частина системи, з якою користувач прямо взаємодіє. Це веб-сторінки, інтерфейси, графічні елементи, кнопки, форми та інші елементи, які відображаються на екрані користувача. Фронтенд відповідає за представлення даних та взаємодію з користувачем.

Ефективність інформаційної системи — Здатність системи порівняння цін на товари надавати точну, актуальну та корисну інформацію для користувачів з метою оптимізації покупок та економії ресурсів.

Споживач — Особа чи організація, які здійснюють купівлю товарів чи послуг.

Оптимізація — Процес підвищення ефективності та результативності шляхом покращення чи оптимізації певних параметрів чи процесів.

Бізнес-процеси — Серія взаємопов'язаних дій та операцій, що виконуються в рамках організації або підприємства, спрямованих на досягнення певних бізнес-цілей.

Інтерфейс — це сукупність засобів, методів та правил, призначених для взаємодії елементів системи (або цілих систем) між собою.

Мова програмування — це штучна мова, створена для передачі команд машинам, зокрема комп'ютерам.

Прив'язка даних — процес приєднання даних з одного об'єкту до іншого об'єкту. Вона забезпечує зручний шлях передачі даних між різними рівнями застосунку.

Веб-застосунок — програмне забезпечення, яке працює через веб-браузер, забезпечуючи доступ до функцій та сервісів без необхідності установки на комп'ютер.

Користувач — особа, яка використовує веб-застосунок для виконання певних завдань або отримання певних послуг.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення аналітичного сервісу

Аналітика - це процес збору, обробки, аналізу та інтерпретації даних з метою отримання інформації, прийняття рішень та виявлення патернів або тенденцій у різних галузях.

Аналітичний сервіс є інформаційною системою, спрямованою на проведення аналізу та порівняння товарів з метою визначення найвигіднішої ціни та магазину для покупок цілої корзини товарів. Він надає користувачам можливість швидко та зручно знайти оптимальний варіант для покупки товарів з урахуванням цінової політики різних магазинів та доступних пропозицій.

Аналітичний сервіс базується на зборі та обробці інформації про товари з різних джерел (див. Рис.1). Важливим етапом реалізації такого сервісу є збір та обробка даних про товари з різних магазинів, онлайн-платформ та інших джерел інформації. Для цього можуть використовуватись механізми сканування та парсингу даних.

Після отримання та обробки даних, інформація про товари зберігається у власній базі даних, що дозволяє ефективно здійснювати подальший аналіз і порівняння пропозицій. Застосування бази даних дозволяє зберігати значну кількість даних про товари, що забезпечує швидкий доступ до необхідної інформації під час проведення аналітики.

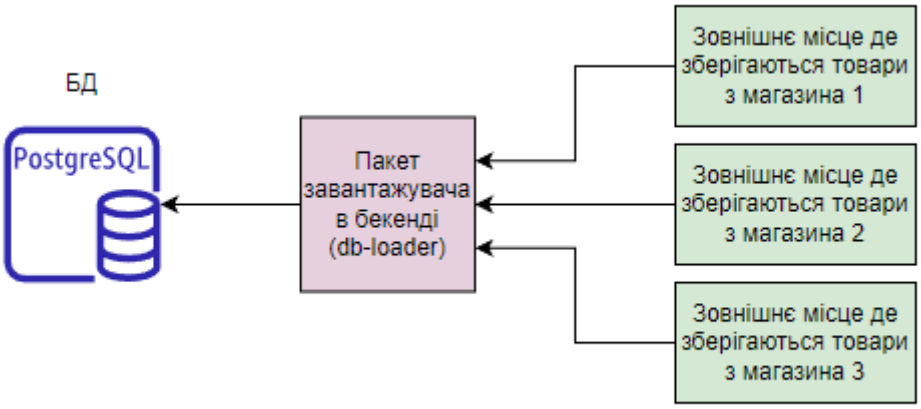

Рис. *1 Принцип роботи бекенда*

Для взаємодії з аналітичним сервісом, користувачам доступний фронтендовий інтерфейс (див. Рис.2), який надає зручний спосіб отримати інформацію про товари та здійснювати аналітику.

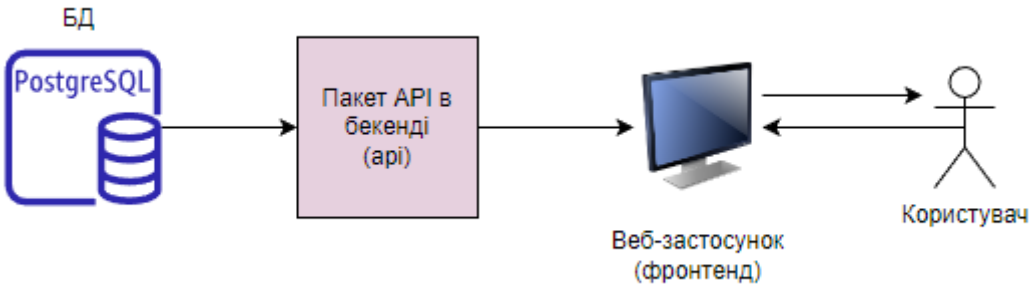

Рис. *2 Принцип роботи фронтенда*

Аналітичний сервіс є потужним інструментом, що дозволяє користувачам здійснювати порівняння товарів і знаходити найвигідніші пропозиції. Він допомагає визначати найкращі ціни на товари і найбільш вигідні магазини для покупки.

1.2 **Огляд літературних джерел**

В ході аналізу предметної області проведено огляд існуючих літературних джерел [1-3], що відносяться до теми роботи. У процесі огляду літературних джерел було визначено основні підходи, методи та технології, що застосовуються у сфері прямого порівняння, зокрема, цін та пропозицій.

У книзі "Designing Data-Intensive Applications" [1] за авторством Мартіна Клеппманна розглядається широкий спектр систем зберігання даних. Автор пропонує докладне дослідження різних моделей та підходів до зберігання даних, з метою допомогти читачам вибрати найбільш підходящу систему для своїх потреб. Одним із ключових аспектів, розглянутих у книзі, є реляційні бази даних (РБД). Клеппманн визначає принципи та особливості РБД, включаючи мову SQL, схему даних, транзакції та оптимізацію запитів. Він також обговорює методи реплікації та шардування даних, що дозволяють масштабувати РБД. Саме досвід з РБД найголовніший. Була розглянута така важлива тема як надійність. Безпека є важливою складовою тому її було розглянуто детальніше. Одним із важливих питань безпеки є вибір стратегії резервного копіювання. Клеппманн описує кілька стратегій, які можна використовувати в залежності від вимог та характеристик системи. Ось деякі з них:

- Повне копіювання: При цій стратегії всі дані повністю копіюються в резервне сховище. Це дозволяє відновити всі дані повністю, але може вимагати значних ресурсів для створення та зберігання повних копій даних.
- Інкрементне копіювання (incremental backup): При цій стратегії створюються копії лише змінених чи доданих даних з останнього повного чи інкрементного копіювання. Це скорочує час та ресурси, необхідні для створення резервної копії, але для відновлення даних потрібно послідовне застосування всіх інкрементальних копій.

- Диференціальне копіювання: При цій стратегії створюються копії лише змінених даних з моменту останнього повного копіювання. На відміну від інкрементного копіювання, кожна диференційна копія містить усі зміни з моменту повного копіювання, що спрощує відновлення даних.

Важливим аспектом резервного копіювання є періодичність створення копій даних. Клеппманн рекомендує визначити оптимальну частоту створення резервних копій, враховуючи бізнес-вимоги, обсяг даних, можливі втрати даних та ресурсні обмеження. Також завдяки автору книги "Designing Data-Intensive Applications" [1] Мартіном Клеппманном була проведена робота з різними аспекти продуктивності та оптимізації SQL-запитів. Він пояснює, як використання індексів, правильне проектування таблиць та оптимізація запитів може підвищити продуктивність роботи з базами даних. Клеппманн також обговорює проблеми, пов'язані з продуктивністю, такі як повільні запити, блокування та конфлікти при паралельному доступі до даних. В результаті цей досвід допоможе при проектуванні БД ті запитів до неї так як даних буде багато та робота з ними може витрачати значний час.

Наступним корисним джерелом для проекту була книга "Data Analytics Made Accessible"[2] за авторством Аніла Махешварі. За його книгою було вирішено обрати структуровані дані в базі даних. Ці дані зазвичай зберігаються в таблицях з певними стовпцями та типами даних, а їх структура та схема можуть бути визначені заздалегідь.

Деякі корективи було введено завдяки Big Data: Principles and Best Practices of Scalable Realtime Data Systems[3] Натана Марзом та Джеймса Уорена. В книзі розповідаються основні принципи та кращі практики роботи з великими даними в реальному часі. Автори пропонують практичні посібники, приклади використання та поради щодо вибору та застосування відповідних інструментів та методів для ефективної роботи з даними в контексті великих даних. Але їх практичні поради, наприклад, розподілення баз даних не можуть бути реалізовані через відсутність потреби та коштів на хостинг додаткових БД.

Теоретичні концепції з практичними прикладами книги Web Application Architecture: Principles, Protocols, and Practices [4] авторів Леона Шклара та Річа Розена були ціннии для розуміння принципів та кращих практик створення надійних та масштабованих веб-додатків. Вона докладно розглядає різні архітектурні шаблони та стилі, які широко використовуються у веб-розробці, такі як модель-представлення-контролер (MVC) та архітектурний стиль REST. Автори обговорюють переваги та міркування кожного шаблону, допомагаючи читачам зрозуміти, як вибрати відповідну архітектуру для своїх додатків. У книзі також розглядаються важливі теми, пов'язані з розробкою веб-застосунків, включаючи технології фронтенду, мови програмування на стороні сервера та варіанти зберігання даних. Вона пропонує розуміння клієнтських технологій, таких як HTML, CSS та JavaScript, а також серверних технологій, таких як PHP, Java та .NET. Крім того, автори досліджують важливі протоколи та стандарти, які використовуються у веб-додатках, такі як HTTP, HTTPS та WebSocket. Вони обговорюють, як ці протоколи забезпечують комунікацію між клієнтами та серверами, та надають рекомендації щодо їх ефективного використання. Загалом книга "Web Application Architecture: Principles, Protocols, and Practices" надає всебічний огляд архітектури веб-додатків, що робить її корисною для аналітичного сервісу. Вона поєднує теоретичні концепції з практичними прикладами, що робить її цінним ресурсом для розуміння принципів та кращих практик створення надійних та масштабованих веб-додатків.

Завдяки "Vue.js in Action" [5] Еріка Ханчетта та Бенджаміна Ліствона було отримано розуміння процесу розробки на фреймворку Vue.js. Автори докладно розглядають концепцію компонентів у Vue.js і показують, як створювати та використовувати компоненти для побудови модульної архітектури програми. Вони пояснюють, як працювати з директивами Vue.js для додавання інтерактивності та реактивності до програм. Крім того, книга охоплює теми, такі як маршрутизація, керування станом за допомогою Vuex, анімації та робота з API. Автори пропонують безліч практичних прикладів та

вправ, які допоможуть читачам краще зрозуміти та застосувати вивчені концепції. Вони також розглядають найкращі практики та поради щодо розробки на Vue.js, засновані на своєму досвіді роботи з фреймворком. Книга "Vue.js in Action" будується таким чином, що допомагає розробникам з великим досвідом роботи розширити свої знання та навички у роботі з Vue.js. Вона надає практичну інформацію яка допоможе розробникам створювати масштабовані та продуктивні веб-застосунки з використанням Vue.js.

Щодо актуальності теми, Уільям Паундстоун в книзі Priceless: The Myth of Fair Value [6] доводить читачеві, що справедливої вартості не існує. Це психологічна робота про концепцію ціноутворення та про те як споживачі сприймають вартість товарів на ринку. Він стверджує, що ідея справедливої вартості - це міф, і сприйняття споживачем вартості часто піддається впливу зовнішніх факторів, таких як соціальні підказки, маркетингові стратегії та когнітивні упередження. Але, маркетингові хитрощі та психологічні спонукання ніщо, коли перед тобою статична картинка та порівняння «лоб в лоб» товарів з різних мереж. Мережі більше не зможуть маніпулювати вибором ставлячи необхідний товар на необхідне місце або не зможуть давити на конкурента більш красивою вивіскою. Подібні аналітичні сервіси вигідні для кінцевих споживачів та для тих мереж котрі хочуть конкурувати ціною зі своїми конкурентами.

Аналіз літературних джерел також розкриває, що порівняння цін на товари є актуальною та важливою проблемою, оскільки споживачі все більше зацікавлені у знаходженні найкращих цінових пропозицій. Порівняння цін дозволяє споживачам приймати обґрунтовані рішення при покупці товарів, економити гроші та час.

Недоліки існуючих методів порівняння цін, такі як складність процесу, недостатня точність та обмежені можливості автоматизації, свідчать про необхідність розробки нової інтелектуальної інформаційної системи, яка враховуватиме сучасні методології та забезпечить зручний та ефективний процес порівняння цін на товари.

Ця робота може мати важливе значення для роздрібної торгівлі, оскільки може сприяти підвищенню ефективності ринку, забезпеченню конкурентоспроможності підприємств і поліпшенню вибору споживачів.

1.3 Аналіз програмних продуктів-аналогів

Проблематика вибору товарів досить популярна та затребувана, тому на даний час існує досить багато крупних та не досить конкурентів, які існують на ринку доволі довго та вже встигли довести свої продукти до гарного стану. Розглянемо деякі з них.

Сервіс PriceGrabber (див. Рис.3) є популярним онлайн-сервісом, який дозволяє користувачам порівнювати ціни на товари в різних інтернет-магазинах. Він надає детальну інформацію про товари, їхні вартості, рейтинги та відгуки користувачів.

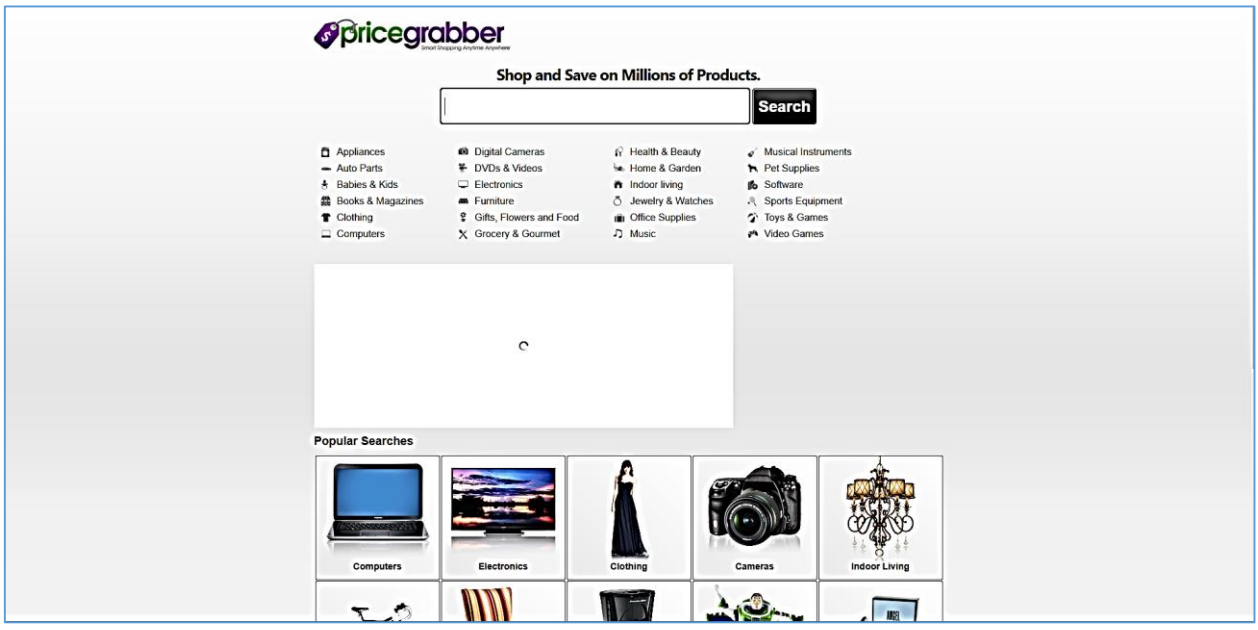

Рис. *3 Головна сторінка PriceGrabber*

Google Shopping сервіс Google для пошуку продуктів по онлайн магазинах та порівняння цін між різними пропозиціями. З самого початку

сервіс надавав лише список цін, затверджених продавцями і монетизувався через AdWords, як і інші сервіси Google. Однак з 2012 року перейшли на модель покупок місць, де продавці мають платити за те, щоб їхні списки були на сайті (див. Рис.4).

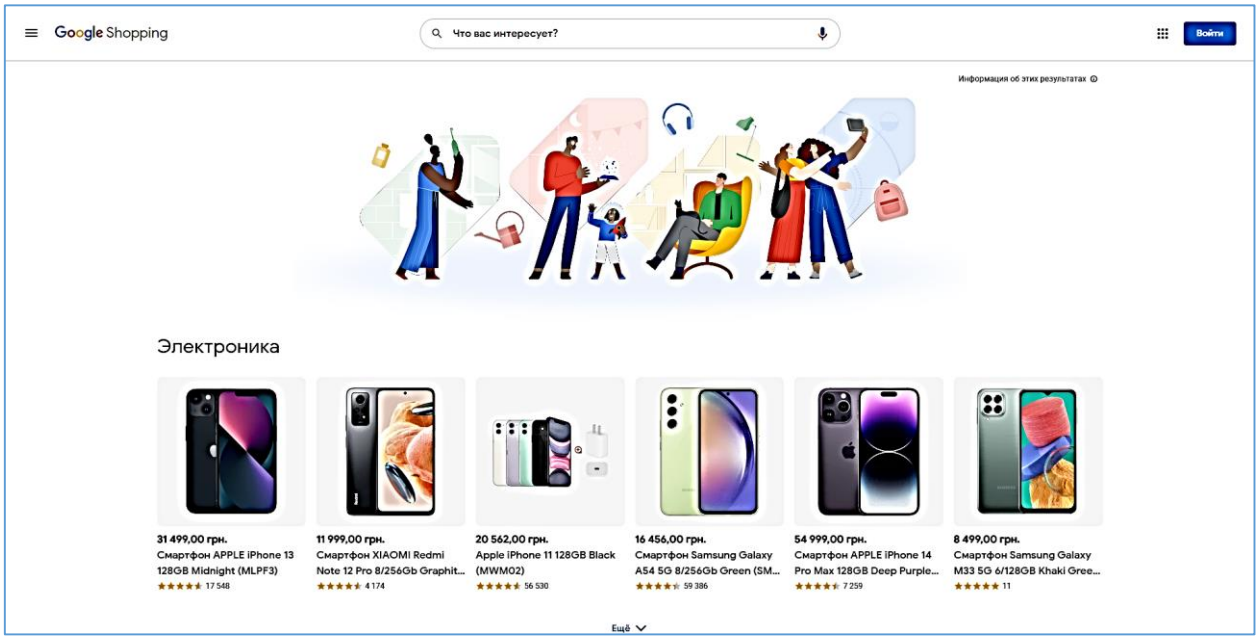

Рис. *4 Головна сторінка Google Shopping*

PriceRunner - це онлайн-платформа для порівняння цін та пошуку товарів, яка розпочала свою діяльність у Швеції і тепер є однією з провідних платформ такого роду в Європі. Вона надає користувачам інформацію щодо цін на різні товари в різних магазинах, а також пропонує відгуки та рейтинги товарів від інших користувачів (див.Рис.5).

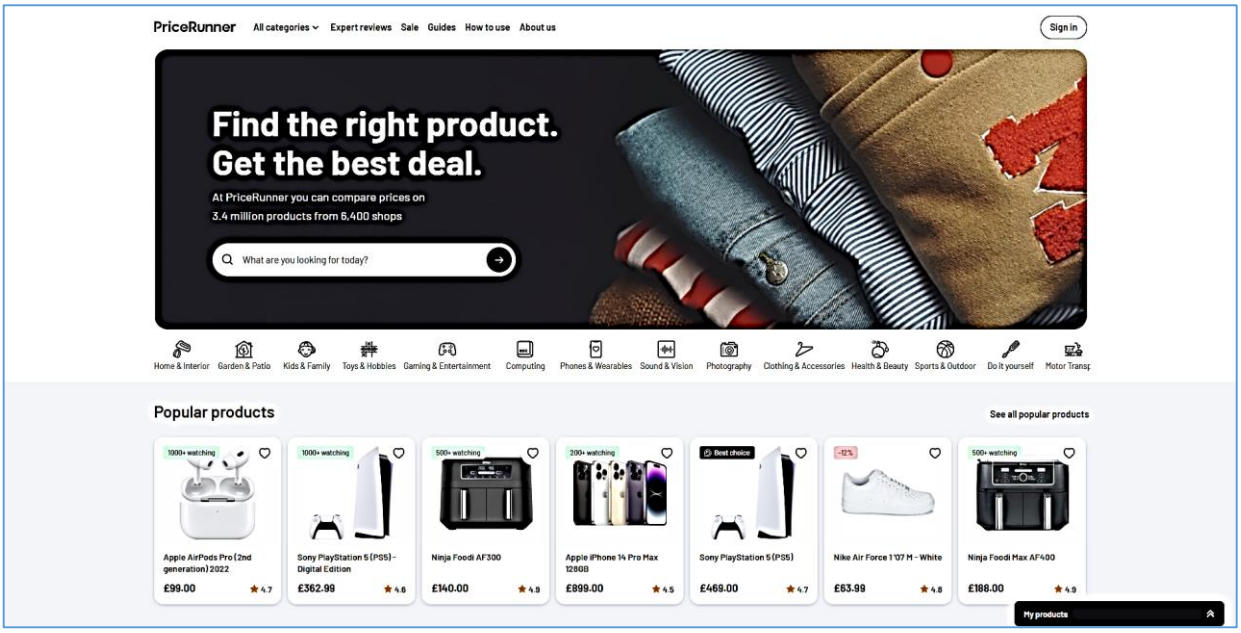

Рис. *5 Головна сторінка сайту PriceRunner*

PriceCheck - це веб-сайт, що пропонує рішення для порівняння цін на товари в Південній Африці. Крім порівняння цін, PriceCheck також допомагає користувачам знайти відгуки та рейтинги товарів. Ви можете переглядати думки та досвід інших покупців, що дозволяє прийняти більш інформоване рішення при покупці. Окрім того, PriceCheck пропонує можливість підписатися на сповіщення про знижки. Ви зможете отримувати повідомлення, коли ціна на певний товар знизиться, що дозволить вам бути в курсі найкращих пропозицій (див. Рис.6).

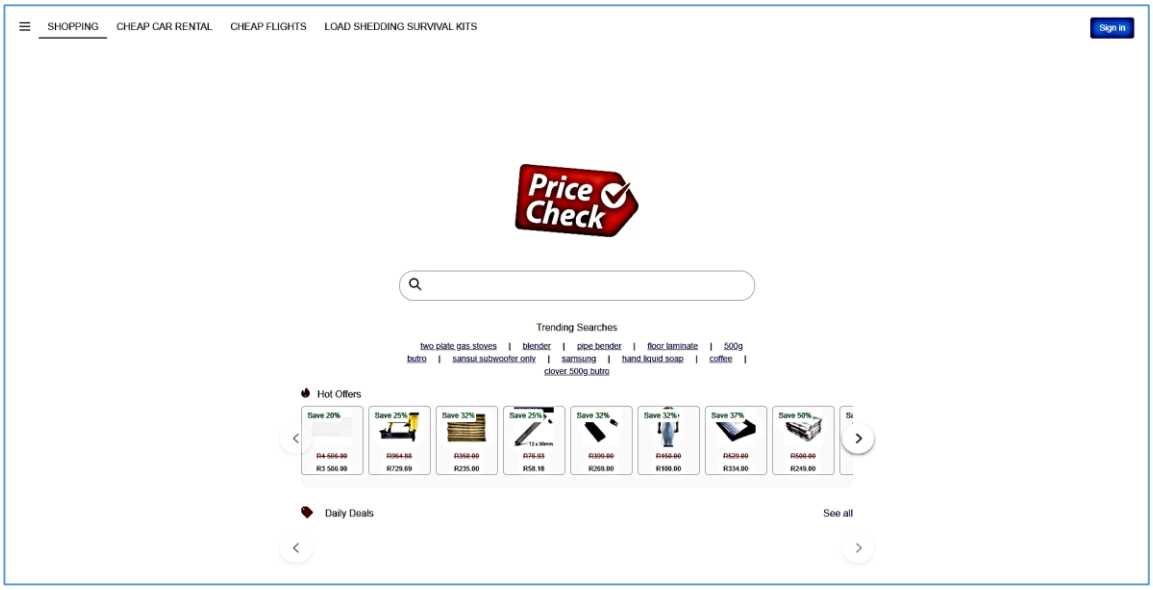

Рис. *6 Головна сторінка сайту PriceCheck*

CamelCamelCamel є веб-сайтом, який допомагає користувачам порівнювати ціни на товари на платформі Amazon. Сайт надає графіки цін, які візуалізують історію зміни цін на певні товари. Це дає змогу отримати уявлення про тенденції ціноутворення та визначити оптимальний момент для покупки. Загалом, CamelCamelCamel допомагає споживачам знайти найкращі пропозиції та зробити розумні покупки на платформі Amazon, шляхом порівняння цін та відстеження їх змін (див. Рис.7).

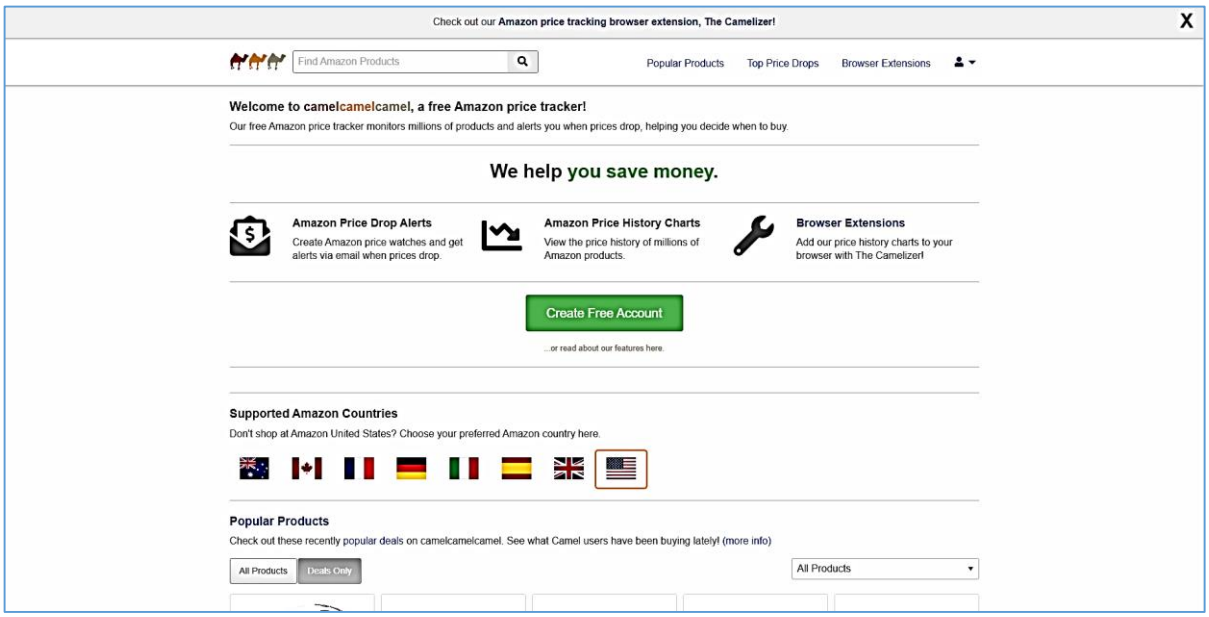

Рис. *7 Головна сторінка сайту CamelCamelCamel*

Аналітична ситсема має ряд функціональних можливостей, серед яких слід виділити:

- **Пошук товарів.** Користувачі зможуть шукати конкретні товари за назвою та категорією;
- **Фільтрація та сортування.** Система може надавати можливість фільтрувати та сортувати результати пошуку за різними критеріями, такими як ціна, рейтинг, категорія тощо;
- **Заміна товара.** Якщо не існує товару в обраному магазині можна швидко замінити на аналог;
- **Порівняння цін.** Система збирає ціни на товари з різних джерел інтернет-магазинів та представляти їх у зручному форматі для порівняння;
- **Порівняння купи товарів.** система аналізує наявність в окремих магазинах обраних товарів та обирає найвигідніший магазин з цією вібиркою товарів;
- **Зберігання інформації.** всі дані про товари зберігаються у власній базі даних.

Слід звернути увагу що всі ці конкуренти є конкурентами лише за ідеєю порівняння цін. Всі вони або працюють з конкретними площадками, або на території окремого регіону, або і те і інше. Також ці конкуренти працюють з непродуктовими товарами. Серед них немає тих, які допомогли б користувачеві під час вибору та планування походу до супермаркету за їжею та дрібними товарами.

Інтерфейс програмного застосунку наведено на рисунку 8.

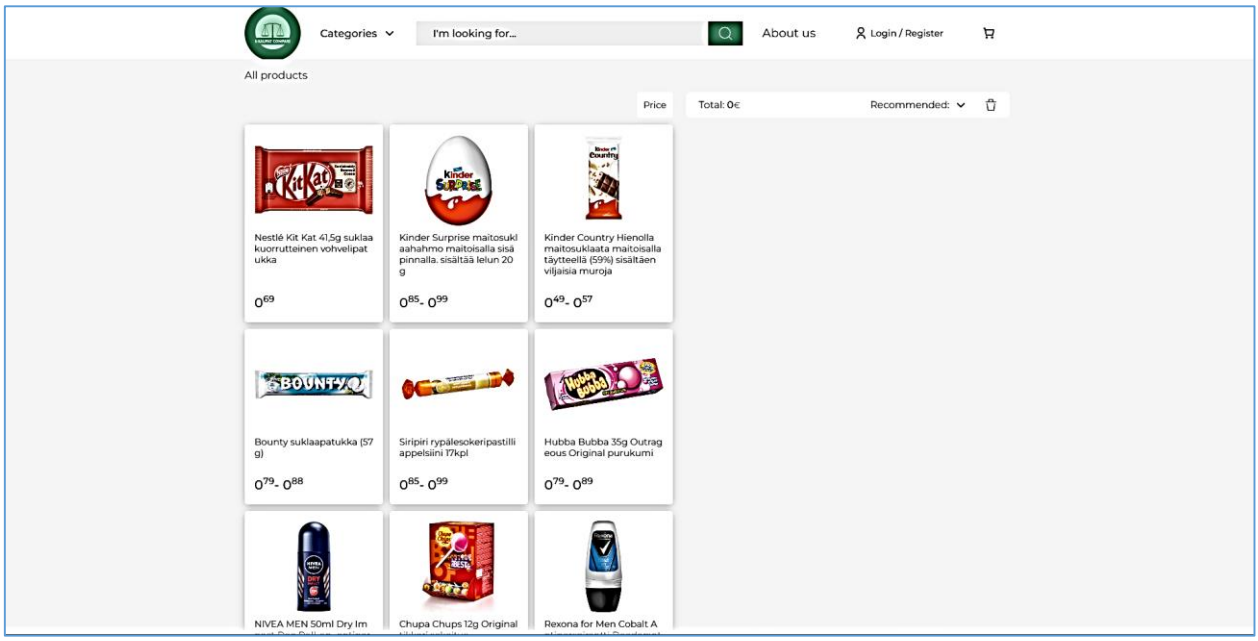

Рис. *8 Додаток аналітичного сервісу порівняння товарів*

Аналіз показав, що існують різні програмні продукти, призначені для порівняння цін на товари. Деякі з них пропонують широкий спектр функціональних можливостей, таких як пошук товарів, порівняння цін, відображення характеристик, відгуків користувачів та інші. Інші можуть бути спрямовані на певні ринкові сегменти або мати обмежені функціональні можливості. Отже, на основі проведеного аналізу можна зробити висновок, що розробка інформаційної системи порівняння цін на товари є актуальною та перспективною задачею, що може принести значну користь споживачам і сприяти оптимізації процесу вибору товарів на ринку.

1.4 **Постановка завдання**

Мета кваліфікаційної роботи — це створення аналітичного сервісу для вибору і купівлі товарів. З використанням комп'ютерних технологій можна ефективно аналізувати актуальні пропозиції та давати можливість користувачам вибирати місце покупки яке буде підходити безпосередньо під їх потреби.

Для досягнення поставленої мети потрібно вирішити наступні завдання:

1. Пошук можливості отримувати актуальні дані про торгівельні пропозиції
2. Створення бази даних яка буде в собі містити таблиці для зберігання актуальніх даних
    1. Обрати СУБД
    2. Підібрати легкіщі типи даних для стовбців
    3. Створити таблиці
    4. З’єднати таблиці по ключах
3. Розробити наймога легкий та оптимізований алгоритм по завантаженню даних до бази даних
    1. Розробити начальний алгоритм завантажання
    2. Зменьшити кількість запитів до бази даних
    3. Під час роботи алгоритму частину даних зберігати локально
4. Розробити API завдяки якому буде можливість розробляти застосунки навколо отриманних даних
    1. Розробити основні ендпоінти для отримання конкретних даних з БД
5. Створити дизайн інтерфейсу застосунку, який буде надавати можлівість аналізувати ціни
6. Розробити застосунок
    1. Зверстати основні сторінки
    2. Зв’язати застосунок з бекендом
    3. Реалізувати логіку порявнянь наданої інвормації

Поставлене завдання включає розробку надійного та зручного сервісу, який допоможе користувачам зробити інформоване рішення при покупках і покращить їхній досвід вибору товарів. Майбутній аналітичний сервіс повинен буде відкривати нові можливості для покупців, дозволяючи їм зробити обґрунтований вибір та буде забезпечувати швидкий та зручний доступ до актуальної інформації про товари.

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Аналіз сучасних систем управління базами даних

Збереження даних це дуже важлива частина роботі застосунку, тому правильний вибір СУБД є дуже важливим етапом проектування роботи системи. Це програмне забезпечення, яке дозволяє зберігати, керувати і забезпечувати доступ до даних у структурованому форматі. СУБД дозволяють ефективно організовувати, зберігати, оновлювати і отримувати інформацію з бази даних. СУБД можуть бути реляційними, об'єктно-орієнтованими, ієрархічними, мережевими та іншими, в залежності від того, яку модель даних вони підтримують і як організовують дані. Деякі популярні СУБД включають MySQL, Oracle Database, Microsoft SQL Server, PostgreSQL та MongoDB. СУБД дозволяють ефективно організовувати, зберігати, оновлювати і отримувати інформацію з бази даних.

Визначення критеріїв вибору СУБД:

- **Функціональність.** Треба щоб обрана СУБД мала необхідні функції для зберігання та обробки даних, які потрібні для подальшого порівняння цін на товари.
- **Продуктивність.** Окремо треба звернути увагу на швидкодію операцій з даними, обсяги даних, які вона може обробляти, та можливості оптимізації запитів. Важливо, щоб система була ефективною та забезпечувала високу продуктивність в контексті обробки великих обсягів даних.
- **Масштабованість.** СУБД повинна буде здатна працювати зі зростаючим обсягом даних та виконувати навантаження в майбутньому. Зростаюча кількість даних не повинна шкодити працездатності.
- **Надійність.** Важливо, щоб СУБД була стабільною та мала мінімальну ймовірність виникнення проблем.

- **Економність.** даних багато як і подальших запитів на ці дані, тому потрібно щоб СУБД не витрачала багато фінансових ресурсів. Ціна ліцензії також грає певну роль.
- **Спільнота.** важливо щоб по СУБД було достатньо інформації та тем з певними проблемами, які можуть заважати під час розробки.

Огляд сучасних СУБД:

- **MySQL.** є однією з найпопулярніших відкритих реляційних СУБД. Вона відома своєю швидкістю та надійністю, а також підтримкою різних операційних систем. MySQL надає широкий спектр функцій для роботи з базами даних і має велику спільноту користувачів.
- **PostgreSQL.** є потужною реляційною СУБД з великою кількістю розширень та можливостей. Вона підтримує розподілені операції, транзакції ACID і повнотекстовий пошук. PostgreSQL також надає можливості геопросторового розширення та підтримує реплікацію для високої доступності.
- **MongoDB.** є документоорієнтованою NoSQL СУБД, яка забезпечує гнучкість та швидкодію при роботі з невстановленою схемою даних. Вона дозволяє зберігати дані у вигляді документів JSON і має вбудовану підтримку реплікації та горизонтального масштабування.
- **Microsoft SQL Server.** Microsoft SQL Server є комерційною реляційною СУБД, розробленою Microsoft. Вона має широкий спектр функцій, таких як розширена підтримка транзакцій, аналітичні можливості та інтеграція з інструментами розробки Microsoft. SQL Server також надає можливості реплікації та кластеризації для забезпечення високої доступності.
- **Oracle Database.** є однією з найпотужніших комерційних реляційних СУБД. Вона відома своїми високими продуктивністю, масштабованістю та надійністю. Oracle також надає широкі можливості для розподіленої обробки даних та аналітики.

Після проведеного аналізу було обрано PostgreSQL [7] як оптимальну СУБД для збереження даних товарів, з урахуванням його функціональності, надійності, масштабованості та підтримки спільноти.

Загальний огляд використання СУБД у подібних проектах вказує на те, що використання Систем управління базами даних (СУБД) є невід'ємною складовою реалізації інформаційних систем у різних сферах діяльності. Особливо в сфері обробки та аналізу великих обсягів даних, використання СУБД стає критичним для забезпечення ефективності та швидкодії проекту.

В результаті проведеного аналізу було визначено, що СУБД PostgreSQL відповідає всім встановленим критеріям та є найбільш підходящою для реалізації інформаційної системи порівняння цін на товари. На основі цього вибору буде проведено подальшу реалізацію та налагодження системи.

2.2 Аналіз сучасних технологій для створення серверної частини

В моєму випадку бекенд відповідає за логічну та технічну частину системи, обробку даних та взаємодію з базою даних. Тому він повинен включати наступні функціональні можливості:

- **Збір даних.** Бекенд повинен мати здатність збирати дані з різних магазинів. Після збору, дані повинні бути оброблені та підготовлені для подальшого використання в порівнянні цін.
- **Управління доступом до бази даних.** Бекенд повинен забезпечувати ефективне управління базою даних, де зберігаються інформація про товари, ціни та інші додаткові атрибути. Він має забезпечувати швидкий доступ до даних та можливість оновлення та редагування інформації.
- **Забезпечення цілісності даних.** Бекенд має мати механізми для захисту цілісності даних. Дані повинні розподілятися серед необхідних таблиць бази даних з коректними ключами зав’язків.

Завдяки формулюванню цих функціональних вимог було виявлено, що вони не є вимогливими та немає необхідності використовувати якусь

конкретну технологію. Це «розв'язало» руки та дозволило обирати технологію з досить великого переліку.

Мій вибір пав на Node JS [8] з наявності декількох доволі значних переваг:

- **Швидкодія.** Node.js використовує подієву та неблокуючу модель вводу/виводу, що дозволяє обробляти багато запитів одночасно без блокування виконання інших операцій. Це забезпечує високу швидкодію та продуктивність системи.
- **Єдина мова програмування.** Використання JavaScript як мови програмування для фронтенду та бекенду дозволяє уникнути перекладу коду та спрощує розробку та обслуговування системи. Це зменшує час, затрачений на навчання та розробку, і сприяє збереженню ресурсів.
- **Асинхронність.** Асинхронність є ключовою особливістю Node.js. Це дозволяє ефективно обробляти багато запитів одночасно, використовуючи мало ресурсів. Бекенд, побудований на Node.js, може легко масштабуватись та впоратись з великими навантаженнями.
- **Спільнота та підтримка.** Node.js має велику активну спільноту розробників, яка постійно вносить внески до розвитку фреймворку та розробляє нові інструменти. Це забезпечує наявність багатьох ресурсів, документації та прикладів, які допомагають у вирішенні проблем та розробці високоякісних додатків.

Основну свою роль зіграла єдина мова програмування та асинхронність. Єдина мова програмування TypeScript дозволила мені витрачати менший час на адаптацію при паралельній роботі з фронтендом та бекендом та взагалі зробила процес розробки більш комфортним. Асинхронність, в свою чергу, зіграла значну роль тому що вона дозволила водночас робити купу різних запитів. Наприклад, користувач зможе отримувати результати своїх невеликих запитів під час роботи з інтерфейсом, наприклад, запити на лист товарів під час того, що паралельно з цим база даних може заповнюватись новими даними. Завантаження даних може займати купу часу (до 40 хвилин) та

критично важливо щоб бекенд міг під час завантаження працювати ще з іншими запитами.

Також дуже важливим було наявність широкого вибору модулів та бібліотек. Так як бекенд буде завантажувати данні, форматувати та робити зв’язки, віддавати дані через API і т.д. тому було дость важливо щоб всі ці операції підтримувались різними модулями. Наприклад, завдяки модулю pg йде зв’язок між бекендом та базою даних PostgreSQL.

З огляду на вищезазначені переваги, можна висунути висновок, що Node.js відповідає всім необхідним вимогам для створення серверної частини. Його продуктивність, масштабованість, широкий вибір модулів та активна спільнота розробників роблять його ідеальним вибором для розробки сучасних та потужних серверних додатків.

Під час вибору фреймворку мій вибір пав на Express [9]. Загалом, Express є потужним інструментом для створення серверної частини додатків. Його простота використання, гнучкість та розширюваність роблять його популярним вибором серед розробників програмного забезпечення, допомагаючи їм швидко та ефективно розробляти веб-додатки. Доволі важливим для мене параметром була швидкість в розробці, що цей фреймворк розробникам і надає. Також важливим параметром була швидкодія, яка досягається шляхом використання неблокуючій моделі вводу/виводу та простому обробнику запитів.

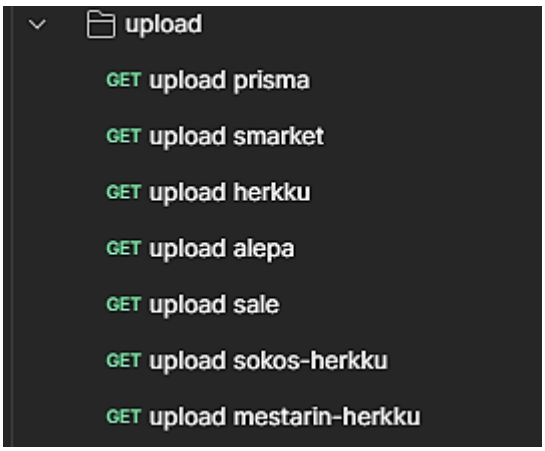

Рис. *9 Лист запитів на завантаження даних*

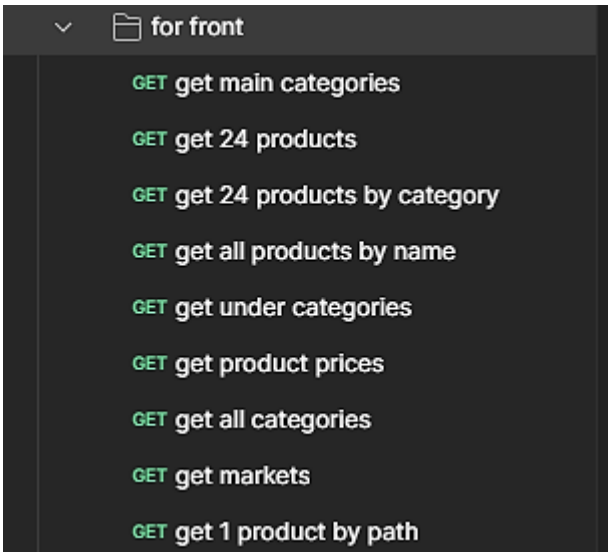

Рис. *10 Лист API запитів*

В сервісі досить багато запитів як на скачуванні та зберіганні даних(див. Рис. 9)., так і на віддачу існуючих даних через API (див. Рис. 10).

В результаті проведеного аналізу було визначено, що Node JS в пов’язці з Express відповідає всім встановленим критеріям та є найбільш підходящою для реалізації інформаційної системи порівняння цін на товари.

2.3 **Аналіз сучасних технологій для створення клієнтської частини**

В моєму випадку фронтенд відповідає не тільки за взаємодію з користувачем, а й за аналітику даних. Логіка саме фронтенду дозволяє

створювати діапазони цін товарів, обирати найвигіднішу пропозицію для товару та проводити порівняння корзин з набором товарів з подальшим обиранням найвигіднішого магазину для купівлі саме цього набору товарів.

Для розробки фронтенду було вирішено використати відповідний фреймворк.

Основні складові для вибору фреймворку:

- **Швидкість розробки.** Фреймворк повинен бути зручним та простим в розробці задля найменшого часу розробки.
- **Швидкість в роботі.** Не повинно бути гальмування під час роботи задля найкомфортнішого користувацького досвіду.
- **Продуктивність.** Добре було б мати можливість використовувати серверний рендеринг та статичне генерування, що поліпшувало б швидкість завантаження сторінок та покращувало взаємодію з користувачем.
- **Підтримка.** Спільноти користувачів фреймворку повинно бути такою щоб не гаяти час на помилки які можна було б виправити завдяки формам, та баги які можуть бути в нових версіях швидко би виправлялись.

Під ці критерії вибору попадають такі фреймворки та бібліотеки як: Nuxt, React, Angular. Ці фреймворки [14] досить схожі та обрати кращого досить складно, але у Nuxt є дуже значна перевага — екосистема Vue [11].

Екосистема Vue.js є одним із факторів успіху фреймворку, забезпечуючи широкий спектр інструментів та ресурсів для розробки, налагодження, тестування та розширення додатків на Vue.js, зокрема на Nuxt. З основних представників екосистеми можна виділити:

- **Vue Router.** Надає маршрутизацію на основі компонентів для програм Vue.js. Він дозволяє визначати маршрути та навігацію всередині програми, керувати історією переходів та підтримувати різні режими маршрутизації, такі як хешовані URL або історичні API.

- **Pinia.** [13] Нова офіційна бібліотека для керування станом, розроблена для Vue 3. Вона надає простий та ефективний спосіб керування станом програми, заснований на концепції глобального сховища.
- **Vue Test Utils.** Зручний набір інструментів та API для тестування компонентів Vue.js. Він дозволяє створювати та монтувати компоненти, емулювати дії користувача, перевіряти відображення та поведінку компонентів, а також писати автоматичні тести для перевірки функціональності додатків на Vue.js.

Це найчастіше використовувані складові екосистеми. Також є Vue CLI, який дозволяє створити готову конфігурацію проекту, Vue Devtools для налагодження та профілювання додатків у браузері, Vuex для створення глобального сховища, і так далі. Завдяки цій обширній екосистемі можна створювати та конфігурувати багатосторінкові додатки, використовувати глобальні сховища, налагоджувати програми в браузері та тестувати компоненти не використовуючи сторонні бібліотеки від інших розробників.

В результаті аналізу було визначено, що використання Nuxt 3 [10] найкраще відповідає вимогам системи і надає широкі можливості для розробки функціонального та продуктивного фронтенду.

2.4 **Огляд інших технологій**

Під «іншими» маються на увазі ті технології які напряму не допомогли написати код програмного сервісу, але допомогли при проектуванні та плануванні майбутніх робіт.

Diagrams.net (раніше відомий як draw.io) [12] - це веб-додаток, який дозволяє створювати діаграми, схеми, прототипи та інші візуальні представлення. Цей інструмент був незамінним у процесі розробки програмного забезпечення, оскільки допомагав втілювати свої ідеї та концепції у візуальну форму. Завдяки цьому сервісу були побудовані наглядні

діаграми взаємодії сутностей, було сплановано архітектуру бази даних і так далі. Цей інструмент допоміг покращити якість роботи, зрозуміти складні структури завдяки наглядному зображенню взаємозв’язків між компонентами системи, та покращити спілкування з науковим керівником.

Figma [15] - це засіб для дизайну і прототипування програмного забезпечення. Завдяки своїм потужним функціям та інтуїтивному інтерфейсу, Figma також надає безліч інструментів для створення високоякісних дизайнів і прототипів. З його допомогою можна створювати розкладки екранів, анімацію, взаємодію та інші важливі елементи користувацького інтерфейсу. Головна перевага те що Figma надає розширені функціональність для дизайну, включаючи редактор векторних графічних зображень, можливість створення компонентів, стилів та бібліотек елементів дизайну. Це дозволяє розробникам швидко створювати стильні та консистентні макети. І все це доступно безкоштовно та навіть у веб версії. Цим програмним рішенням було успішно розроблено інтерфейс для прогнозованого веб-додатку, що вказує на ефективне використання зазначеного інструменту для досягнення поставленої мети.

3 РОЗРОБКА АНАЛІТИЧНОГО СЕРВІСУ ВИБОРУ ТОВАРІВ

3.1 Опис предметної області

Предметна область, в рамках якої розглядається цей проект, відноситься до сфери електронної комерції і спрямована на створення аналітичного сервісу для вибору і купівлі товарів. У сучасному світі, де онлайн-шопінг стає все більш популярним, розробка ефективних інструментів для аналізу пропозицій товарів та надання користувачам можливості здійснювати свідомий вибір місця покупки стає дедалі важливішою задачею.

Предметна область охоплює такі аспекти, як пошук, порівняння цін, вибір місця покупки та огляди товарів. Крім того, вона включає аналіз актуальних пропозицій на ринку, збір та обробку даних про товари та їх характеристики.

Основною метою даного проекту є створення аналітичного сервісу, який допоможе користувачам ефективно аналізувати наявні пропозиції товарів, порівнювати ціни та інші характеристики, а також забезпечити їм можливість здійснювати свідомий вибір місця покупки, що найкраще відповідає їхнім потребам та вимогам. З ростом електронної комерції та зростанням кількості доступних товарів і магазинів, важливо мати систему, яка забезпечить користувачам зручний і надійний спосіб отримання інформації про товари та можливостей порівняння їх характеристик та цін. Таким чином, створення аналітичного сервісу, який поєднує в собі функціонал пошуку, порівняння та відслідковування цін, стає необхідним для сприяння покупцям у прийнятті обґрунтованих рішень.

3.2 Вимоги до системи

Розробка будь-якої програмної системи передбачає чіткі вимоги, які визначають функціональні та нефункціональні характеристики, що мають бути задоволені. Вимоги до системи є необхідними, оскільки вони визначають

характеристики та функціональність, які система повинна мати для відповіді на потреби користувачів та бізнес-вимоги. Вони служать основою для розробки та проектування системи, забезпечуючи виконання вимог, встановлення критеріїв успішності та забезпечення задоволення потреб усіх зацікавлених сторін.

З загальних вимог до створення аналітичного сервісу для вибору і купівлі товарі можна виділити такі функціональні та нефункціональні вимоги:

Функціональні вимоги:

- Забезпечення можливості пошуку користувачем товарів за різними параметрами, включаючи категорію та назву.
- Реалізація порівняння цін товарів, з метою виявити найдешевшу пропозицію.
- Можливість сортувати список товарів за спадаючою або спадною ціною.
- Реалізація порівняння корзини товарів в якої знаходиться купа товарів з подальшим виведенням ціни.
- Можливість продивлятися основну інформацію по кожному товару.
- Змога користувача вручну обирати мережу супермаркетів , що цікавить його.
- Можливість користувача швидко замінити товар в корзині, якщо його немає в наявності в якомусь магазині.
- Можливість завантаження повних даних про товари з різних магазинів до БД.
- Реалізація створення стрічки товарів як без фільтрів категорій або пошуку так і з ними.

Нефункціональні вимоги:

- Висока продуктивність та швидкодія системи для швидкого відгуку на запити користувачів.
- Забезпечення стабільності та надійності системи, що дозволить уникнути відмов та збоїв у роботі.
- Забезпечення безпеки даних та захисту конфіденційної інформації користувачів.
- Масштабованість системи, щоб забезпечити підтримку зростаючої кількості користувачів та товарів.
- Зручний та інтуїтивно зрозумілий інтерфейс користувача для забезпечення легкої навігації та зручного використання системи.
- Відокремлені один від одного на сервері сервіси по завантаженню даних та API для клієнтської частини.

Вище приведені основні вимоги до системи визначають необхідні характеристики та функціональність, які має мати програмний продукт. Ці вимоги служать основою для подальшого проектування та розробки системи. Врахування вимог до системи допомагає забезпечити задоволення потреб користувачів та бізнесу, покращити якість програмного продукту та забезпечити його успішне впровадження.

3.3 **Архітектура системи**

Система аналітичного сервісу для вибору і купівлі товарів реалізована за архітектурою клієнт-сервер. Клієнт-серверна архітектура - це модель, у якій клієнт (користувач або програма) взаємодіє з сервером (центральний компонент). Клієнт надсилає запити, а сервер обробляє їх і надсилає відповіді. Це дозволяє розділити функціональність та ефективно спілкуватися між компонентами системи.

Основні складові системи:

- База даних (БД)
- Серверна частина (бекенд)

- Клієнтська частина (фронтенд)

3.3.1 Огляд бази даних

База даних зберігає в собі всю інформацію про товари та їх додаткові складові. Схема за якою зберігається вся інформація про товари (див. Рис. 11). Завдяки існуючий купі унікальних товарів в таблиці unique_products потім буде йти порівняння цін між всіма торговельними мережами, в яких ця позиція мається в наявності.

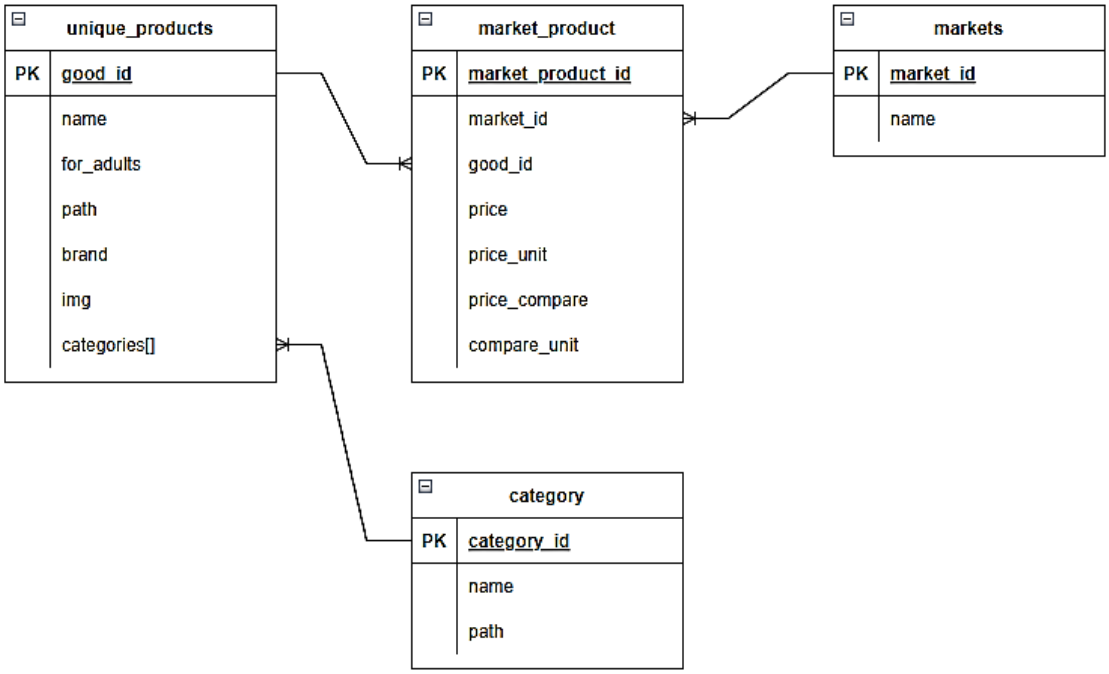

Рис. *11 Схема бази даних для зберігання інформації по товари*

Усього мається 4 таблиці: markets, category, unique_products, market_product.

- **markets.** Таблиця з інформацією про магазини, товари яких маються в базі даних. Стовбці:
  - **market_id.** Унікальний ідентифікатор магазина та первинний ключ таблиці.

- **name.** Назва магазину.

- **category.** Таблиця з переліком категорій, які маються у товарах бази даних. Стовбці:
  - **category_id.** Унікальний ідентифікатор категорії та первинний ключ таблиці.
  - **name.** Назва категорії.
  - **path.** Унікальний шлях до цієї категорії, який можна використати, наприклад, для пошуку.
- **unique_products.** Таблиця з переліком усіх унікальних товарів та інформацією про них. Стовбці:
  - **good_id.** Унікальний ідентифікатор товару та його первинний ключ таблиці.
  - **name.** Ім’я товару.
  - **for_adults.** Поле яке відповідає за те чи можна продавати цей товар неповнолітнім.
  - **path.** Унікальний шлях до цього товару, який можна використати, наприклад, для пошуку.
  - **brand.** Назва бренду виробника або дистриб’ютора.
  - **img.** Шлях по якому можна знайти jpg зображення до цього товару.
  - **categories.** Масив з первинними ключами категорій таблиці category.
- **market_product.** Таблиця яка поєднує таблиці markets та unique_products. Стовбці:
  - **market_product_id.** Унікальний ідентифікатор зв’язку магазина та товара та його первинний ключ таблиці.
  - **market_id.** Унікальний ідентифікатор магазина, з яким зв’язана ця запис.

- **good_id.** Унікальний ідентифікатор товару, з яким зв’язана ця запис.
- **price.** Ціна цього товару конкретно в цьому магазині.
- **price_unit.** Одиниця, яку отримає покупець за ціну, наприклад, штука.
- **price_compare.** Ціна цього товару конкретно в цьому магазині в переводі до стандартизованої одиниці.
- **compare_unit.** Сама стандартизована одиниця порівняння, наприклад кілограм, літр і т.д.

3.3.2 Огляд серверної частини

Серверна частина є прикладом здебільшого монолітної архітектури. Це підхід до розробки програмного додатку, при якому всі його компоненти, функціональність та шари логіки об'єднані в одній кодовій базі та виконуються як єдине ціле. У монолітній архітектурі всі модулі та компоненти програми зазвичай компілюються разом і розгортаються на одній платформі чи сервері. Звернення до функцій та сервісів відбувається всередині моноліту у вигляді виклику функцій чи методів. Однак в архітектурі були певні кроки до переходу на мікросервісну архітектуру. У мікросервісної архітектурі кожен сервіс відповідає виконання конкретної завдання чи функції. Кожен сервіс може мати власну базу даних, свій код і команду розробників. Комунікація між сервісами зазвичай здійснюється через мережеві протоколи, такі як HTTP або повідомлення. Було це реалізовано шляхом розбиття моноліту на окремі пакети (див. Рис. 12).

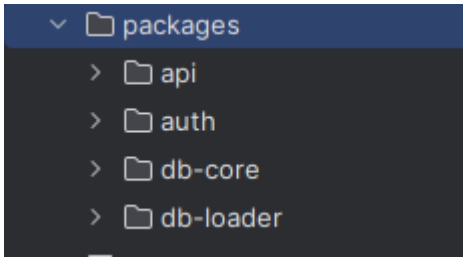

Рис. *12 Пакети серверної частини*

Кожен пакет відповідає за свою частину функціональності:

- **api.** Відповідає за “віддачу” даних з бази даних на фронтенд або на будь-які інші сервери.
- **auth.** Пакет який працює з авторизацією і всім іншим що пов’язано зі створенням особистого кабінету.
- **db-core.** Основний пакет по роботі з базою даних. Має в собі CRUD операції.
- **db-loader.** Відповідає за завантаження товарів з купи магазинів за вибором.

Всі ці пакети запускаються з кореню проекту та працюють на різних портах. Саме пакетна структура дозволяє зробити гібрид моноліта та мікросервіса. З однієї сторони, все працює на одній машині та спілкуються між собою не через HTTP запити, а з іншої це все різні відокремленні один від одного проекти зі своїми власними версіями. Це могло би дозволити щоб над кожним окремим пакетом працював свій розробник не заважаючи роботі своїх колег. Дозволило окремо по кожному пакету моніторити зміни та не переживати що зміни на одному пакеті приведут до збоїв в іншому. Ці пакети відокремленні один від одного, що оберігає розробника від таких неприємностей. Для реалізації повноцінного мікросервіса слід було окремі пакети хостити на різних серверах та зробити спілкування між пакетами через HTTP запити, однак в поточному стані це б дало більше негативних наслідків, аніж позитивних.

3.3.3 Огляд клієнтської частини

При розробці клієнтської частини було реалізовано MVVM (Model-View-ViewModel) архітектуру. Це шаблон проектування, який використовується для поділу відповідальності у додатку та організації коду. Він забезпечує чіткий поділ між моделлю даних, поданням інтерфейсу

користувача і прошарком, званої ViewModel, яка пов'язує модель і подання. ViewModel реалізується за допомогою нативних можливостей фреймворка Vue. Змінюючи моделі завдяки двосторонньому зв’язуванню водночас зі зміною v-model перерисовуються й інтерфейси та computed обчислення. Взагалі користуючись фреймворком таким як Nuxt треба розуміти що вони диктують свою власну архітектуру та по більшій частині файлові рішення. Використовуються компоненти як основні будівельні блоки. Компоненти представляють окремі частини інтерфейсу користувача і можуть бути повторно використані в різних частинах програми. Поясніть, як ви організували компоненти у вашому додатку та як вони взаємодіють між собою. Компоненти спілкуються один з одним не тільки напряму, але й використовуючи глобальне сховище. З основних є всього п’ять сховищ глобальних даних:

- **cart.** Сховище в якому зберігаються всі товари які потрапили у кошик та описуються функції, котрі використовуються для отримання або редагування даних сховища.
- **categories.** Сховище, в якому зберігаються завантаженні з API категорії та обрана користувачем категорія. Також містяться функції по отриманню або зміненню інформацій.
- **goods tape.** Сховище яке відповідає за стрічку товарів. Відповідає також за завантаження даних товарів з API до сховища.
- **markets.** Сховище з магазинами. Містить перелік існуючих магазинів та обраний користувачем магазин
- **sorting.** Сховище яке працює з різними сортуваннями, котрі користувач може обрати під час роботи з клієнтською частиною.

Всі компоненти котрі використовуються для відображення даних працюють саме з цими глобальними сховищами. Вони як відрисовують інформацію, так і змінюють її. Таким чином головна інформація не знаходиться безпосередньо в компонентах та основна логіка відділена від представлення. Це відокремлення допомагає при розробці та підтримці коду.

3.4 **Розробка бази даних**

Під час розробки бази даних було обрано PostgreSQL як СУБД. Відповідно, мова для БД це SQL.

Лістинг 1 — *лістинг коду створення таблиці магазинів*

```
create table  markets(
    market_id smallserial primary key,
    name text
);
```

Лістинг 2 — *лістинг коду створення таблиці унікальних товарів*

```
create table unique_products(
  good_id serial primary key,
  name text,
  for_adults boolean,
  path text,
  brand text,
  img text,
  categories integer[]
);
```

Лістинг 3 — *лістинг коду створення таблиці категорій*

```
create table category(
  category_id serial primary key,
  name text,
  path text
);
```

Лістинг 4 — *лістинг коду створення таблиці зв'язку товару та магазин*

```
create table market_product(
  market_product_id serial primary key,
  market_id smallint,
  good_id integer,
  price real,
```

```
  price_unit text,
  price_compare float,
  compare_unit text,
  foreign key (market_id) references markets (market_id),
  foreign key (good_id) references unique_products
(good_id)
);
```

У лістингах кодів 1 — 4 зазначений код для створення таблиць для зберігання повної інформації про магазини, товари та їх складові.

В результаті було створено базу даних PostgreSQL з таблицями магазинів, унікальних продуктів, категорій, зв’язків продуктів та магазинів. В подальшому це дозволе серверній частині зберігати інформацію.

3.5 Програмна реалізація серверної частини

Як нам відомо, ми обрали для розробки мову типізованого JavaScript — TypeScript. В якості платформи для розробки було обрано Node JS в парі з серверним фреймворком Express. Структура проекту це моноліт, який в ході розробки через громіздкість та незручність було вирішено розбити на дрібні самодостатні пакети. Цього було досягнуто завдяки Lerna. Lerna - це інструмент для управління багатопакетними репозиторіями JavaScript. Він допомагає керувати проектами, що складаються з декількох пакетів, та полегшує розробку і публікацію пакетів. Саме завдяки цьому зараз бекенд складається з основних пакетів: api, db-core, db-loader.

3.5.4 Розробка пакету db-core

Пакет db-core є фундаментальним пакетом завдяки якому йшла первісна настройка взаємодії сервера за базою даних та через який зараз може йти ручне корегування будь якої з таблиць в БД. Він є окремим Express проектом та працює на своєму порті.

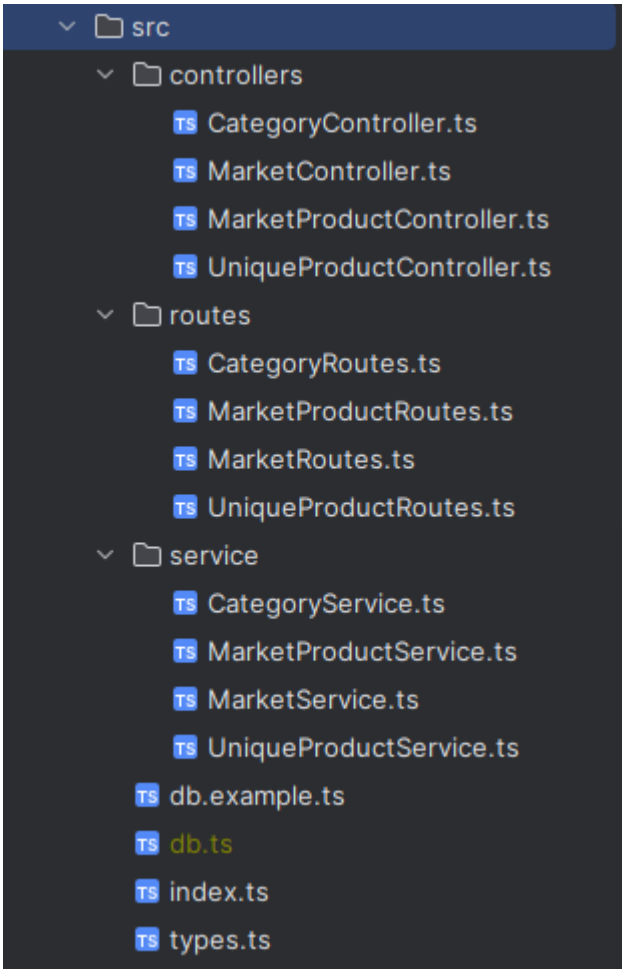

Рис. *13 Файлова структура db-core*

На рис.13 представлена файлова структура пакету. З цього рисунку видно, що код пакету логічно розбитий по трьом секціям: роути, контролери, сервіси. Для кожної окремої таблиці в базі даних є свій окремий роут, контролер та сервіс, наприклад, для таблиці category існує CategoryRoutes, CategoryController, CategoryService.

В головному файлі index розміщена логіка по створенню сервера на конкретному порті та додавання ендпоінтів, які буде прослуховувати сервер.

Лістинг 5 — *лістинг коду index файлу*

```
const express = require('express')
const marketsRouter = require('./routes/MarketRoutes')
const uniqueProductsRouter =
require('./routes/UniqueProductRoutes')
const categoryRouter = require('./routes/CategoryRoutes')
const marketProductRouter =
```

```
require('./routes/MarketProductRoutes')

const PORT = process.env.PORT || 1400

const app = express()

app.use(express.json())
app.use('/market', marketsRouter)
app.use('/unique', uniqueProductsRouter)
app.use('/category', categoryRouter)
app.use('/market-product', marketProductRouter)

app.listen(PORT, () => console.log(`Server db-core starts
on port ${PORT}`))
```

На лістингу 5 знаходиться код index файлу, який імпортує всі роути з папки routes та додає їх на прослуховування сервером, після чого запускає север на порті 1400. Структура index файлу однакова во всіх пакетах.

Лістинг 6 — *лістинг коду router файлу*

```
const Router = require('express')
const router = new Router()
const categoryController =
require('../controllers/CategoryController')
router.post('/', categoryController.create)
router.get('/', categoryController.getAll)
router.get('/:id', categoryController.getOne)
router.get('/:name', categoryController.getOneByName)
router.put('/', categoryController.update)
router.delete('/:id', categoryController.delete)

module.exports = router
```

На лістингу 6 зображено як приклад код файлу CategoryRoutes. Код файлу визначає тип запиту та шлях роуту, на який буде визиватися певний метод з класа контролера. Структура роутів однакова і для інших файлів.

Лістинг 7 — *лістинг прикладу коду controller файлу*

```
const CategoryService =
require('../service/CategoryService')
import {ICRUDController} from "../types";
```

```
interface ICategoryController extends ICRUDController{
  getOneByName(req: any, res: any): Promise<void>
}

class CategoryController implements ICategoryController{
  async create(req:{body:{name:string, path: string}},
res:any){
    try {
      const {status, response} = await
CategoryService.create(req.body)
      res.status(status).json(response)
    } catch (e) {
      res.status(500).json(e)
    }
  }
 }

module.exports = new CategoryController()
```

На лістингу 7 зображено як приклад частина коду файлу CategoryController. Код файлу визначає методи контролеру, які приймають запит та відповідь, та визивають певний метод з service файлу. У разі помилку поверне статус помилки та опис. Така структура не тільки для create запиту, а й для інших також. Структура контроллерів однакова і для інших файлів.

Лістинг 8 — *лістинг прикладу коду service файлу*

```
const db = require("../db");
import {IStatusResponse, ICRUDService} from "../types";

interface ICategoryService extends ICRUDService{
  getOneByName(name: string): Promise<IStatusResponse>
}
class CategoryService implements ICategoryService{
  async create(body:{name:string, path:string}):
Promise<IStatusResponse>{
    if (!body) return {status: 400, response: 'No category
specified'}
    if (!body.name || !body.path)
      return {status: 400, response: 'Required input
fields: name, path'}

    //checking if this store is already in our database
    let duplicate = false
```

```
    const all = (await this.getAll()).response
    for (const item of all){
      if (item.name === body.name && item.path ===
body.path) {
        duplicate = true
      }
    }
    if (duplicate) return {status: 404, response: 'This
category is already in db'}

    //adding a new category
    const createNew = await db.query(`insert into category
(name, path) values ($1, $2) returning * `, [body.name,
body.path])
    return {status: 200, response: createNew.rows[0]}
  }
}

module.exports = new CategoryService()
```

На лістингу 8 зображено як приклад частина коду файлу CategoryService. Код файлу представляє create, в якому спочатку перевіряються поля котрі повинні бути в запиті, потім перевіряється наявність такого запису в таблиці і якщо ні то створюється новий запис. В разі виявлення якоїсь з помилок буде повернено певний статус код помилки та опис проблеми. Структура сервісів однакова і для інших файлів.

Таким чином розроблено пакет який представляє собою комплексний набір функцій, які дозволяють точково редагувати або створювати рядки в будь-якій таблиці, що існує в нашій базі даних. У складі пакету повністю реалізовані операції CRUD, що означає можливість створення (Create), читання (Read), оновлення (Update) та видалення (Delete) даних в таблицях бази даних. Окрім базових CRUD операцій, пакет також містить додаткові корисні запити, що розширюють можливості користувачів. Наприклад, він надає можливість здійснювати пошук записів за ім'ям, що дозволяє швидко та зручно знаходити потрібні дані у великій таблиці. Це особливо корисно в ситуаціях, коли потрібно швидко знайти конкретний запис серед багатьох інших.

Всі ці функціональні можливості пакету спрямовані на полегшення та прискорення роботи з базою даних. Вони дозволяють користувачам здійснювати потрібні операції з даними безпосередньо, без необхідності використовувати складні запити або програмувати власні функції для кожної операції. Таким чином, пакет надає зручні та потужні інструменти для ефективного управління даними у вже створеній базі даних.

3.5.5 Розробка пакету db-loader

Пакет db-loader представляє з себе завантажувач повних даних про товари з різних магазинів. Саме він наповнює базу даних даними, котрі потім можуть бути використані для будь-яких цілей на розсуд власника. Цей пакет також є окремим Express проектом та працює на своєму порті.

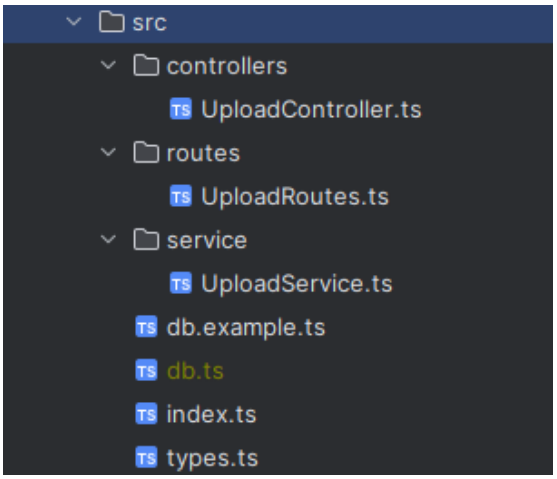

Рис. *14 Файлова структура db-loader*

На рис.14 представлена файлова структура пакету. Це стандартна структура з index файлом в якому запускається сервер на окремому порті та файлами роутів, контроллерів та сервісів. Головний файл index нічим не відрізняється від db-core, як і роути з контролерами, але має іншу бізнес-логіку яка розташована в service.

Під час використання цього пакету користувач має можливості, які візуально зображено на use-case діаграмі (див. Рис. 15). Діаграма в себе

включає єдиного актора Користувача та 7 дій які направлені на завантаження кожного магазину окремо. Таким чином визиваючи будь-який запит буде проходити завантаження даних обраного магазина до наявної бази даних.

Рис. *15 Use-case діаграма db-loader*

Алгоритм завантаження даних можна розбити на такі складові:

- **Наповнення локального масиву.** Під час виконання відправляються запити на сервер з даними та під час отримання результатів дані пушаться до локального масиву.
- **Додавання категорій.** Алгоритм проходиться по масиву та аналізує наявність цієї категорії в базі даних.
- **Додавання унікальних товарів.** Алгоритм проходиться по масиву та аналізує наявність цього унікального товару в базі даних.
- **Додавання зв'язку магазин-продукт.** Алгоритм проходиться по масиву та створює рядки у відповідній таблиці в базі даних.

Лістинг 9 — *лістинг наповнення локального масиву*

```
const market_id = await this.getMarketId(market)
let i: number = 0
const step: number = 2500
let arr: any[] = []
let total: number = 0
while(true){
  i += step
  const res = await this.uploadQuery(step, i - step,
market)
  total += res.data.data.store.products.items.length
  console.log(res.data.data.store.products.items.length + "
total: " + total)
  if (res.data.data.store.products.items.length === 0) {
    break
  }
  arr.push(...res.data.data.store.products.items)
}
```

Наповнення локального масиву записаному на лістингу 8 проходить шляхом циклічного визову uploadQuery, який являється лише шаблоном запиту з динамічними полями кроку, відступу та магазину, з якого потрібно взяти інформацію. Кожного разу, коли запит проходить вдало та повертає якісь дані то цикл продовжується, але переривається одразу якщо даних вже немає.

Після цього наступає наступний етап — сортування цієї купи даних та завантаження по різних таблицях починаючи з додавання категорій.

Лістинг 10 — *циклічний запис категорій по таблицях шляхом визову методу updateCategory*

```
private async insertAllCategories(arr:any[])
  console.log('starts insert categories')
  for (const item of arr){
    if (!item.hierarchyPath) continue
    for (const category of item.hierarchyPath)
      await this.updateCategory(category.name,
category.slug)
  }
}
```

Лістинг 11 — *код методу updateCategory*

```
private async updateCategory(name:string, path:string){
if (this.localCategories.find(item => item.name === name &&
item.path === path)){
  return this.localCategories.find(item => item.name ===
name && item.path === path)?.category_id
}
const query = await
CategoryService.getOneByNameAndPath(name, path)

if (query.response?.category_id) {
  this.localCategories.push(query.response)
  return this.localCategories.find(item => item.name ===
name && item.path === path)?.category_id
}
const body = {
  name: name,
  path: path
}
//create new category

const createCategory = (await
CategoryService.create(body)).response

this.localCategories.push(createCategory)
return this.localCategories.find(item => item.name === name
&& item.path === path)?.category_id
}
```

На лістингах 10 — 11 зображено алгоритм додавання категорій до бази даних. Метод insertAllCategories циклічно проходить по всіх елементах в масиві завантажених товарів та на кожен товар викликає метод updateCategory. Цей метод робить перевірку чи вже є в локальному масиві така категорія, так як якщо вже вона є то це означає що вона в минулому була додана до бази даних та немає сенсу зайвий раз робити запити до бази даних. Після цієї перевірки метод робить запит до бази даних та шукає чи є в БД така категорія. Якщо категорія в базі даних знайшлась, то вона пушиться до локального масиву категорій та виконання методу приривається. Якщо ж ця категорія є унікальною, то робиться запит до бд на створення категорії та вона записується до локального списку з метою подальшого використання без

додаткових запитів до БД. Наступним кроком йде додавання унікальних продуктів.

Лістинг 12 — *код методу заповнення таблиці унікальних товарів з циклічним викликом методу updateUniqueProducts*

```
private async insertAllUniqueProducts(arr:any[]) {
  console.log('starts insert unique')
  for (const item of arr){
    const getArrCategories = async () => {
      let categories:any[] = []
      if(item.hierarchyPath !== null && item.hierarchyPath
!== undefined){
        for (let i = 0; i < item.hierarchyPath.length;
i++){
          if (item.hierarchyPath[i].name)
          categories.push(this.localCategories.find(local
=> local.name === item.hierarchyPath[i].name && local.path
=== item.hierarchyPath[i].slug)?.category_id)
        }
      }else{
        return []
      }

      return categories
    }

    const product = {
      name: item.name,
      for_adults: item.isAgeLimitedByAlcohol,
      path: item.slug,
      brand: item.brandName,
      img: item.ean,
      categories: await getArrCategories()
    }

    await this.updateUniqueProducts(product)
  }
}
```

Лістинг 13 — *код методу updateUniqueProducts*

```
private async updateUniqueProducts(product: {name: string,
for_adults: boolean, path: string, brand: string, img:
string, categories: any[]}){
  if (this.localUniqueProducts.find(item => item.name ===
product.name && item.path === product.path)) {
```

```
    return this.localUniqueProducts.find(item => item.name
=== product.name && item.path === product.path)?.good_id
  }
  const query = await
UniqueProductService.getOneByNameAndPath(product.name,
product.path)
  if (query.response?.good_id) {
    this.localUniqueProducts.push(query.response)
    return this.localUniqueProducts.find(item => item.name
=== product.name && item.path === product.path)?.good_id
  }

  //create new unique product
  const createUniqueProduct = (await
UniqueProductService.create(product)).response
  this.localUniqueProducts.push(createUniqueProduct)
  return this.localUniqueProducts.find(item => item.name
=== product.name && item.path === product.path)?.good_id
}
```

На лістингах 12 — 13 реалізований алгоритм додавання унікальних товарів до бази даних. Метод insertAllUniqueProducts циклічно проходить по всіх елементах в масиві завантажених товарів та на кожен товар викликає метод updateUniqueProducts передаючи в нього масив первинних ключів категорій, які належать цьому продукту. Метод updateUniqueProducts робить перевірку чи вже є в локальному масиві такий унікальний продукт, так як якщо вже він є то це означає що він в минулому був додан до бази даних та немає сенсу зайвий раз робити запити до бази даних. Після цієї перевірки метод робить запит до бази даних та шукає чи є в БД такий унікальний продукт. Якщо унікальний продукт в базі даних знайшовся, то він пушиться до локального масиву унікальних продуктів та виконання методу приривається. Якщо ж він є унікальним, то робиться запит до бд на створення унікального продукту та він записується до локального списку з метою подальшого використання без додаткових запитів до БД. Далі йде фінальний крок — формування та заповнення таблиці зв’язків продуктів та магазинів.

Лістинг 14 — *код методу заповнення таблиці зв'язків товару та магазину з циклічним викликом методу updateMarketProduct*

```
private async insertAllMarketProduct(arr:any[],
market_id:number){
  console.log('starts insert market-products')
  for (const item of arr){
    const good_id = this.localUniqueProducts.find(local =>
local.name === item.name && local.path ===
item.slug)?.good_id
    const body:{market_id: number, good_id: number |
undefined, price: number, price_unit: string,
price_compare: number, compare_unit: string} = {
      market_id: market_id,
      good_id: good_id,
      price: parseFloat(item.price.toFixed(2)),
      price_unit: item.priceUnit,
      price_compare: item.comparisonPrice,
      compare_unit: item.comparisonUnit
    }
    await this.updateMarketProduct(body)
  }
}
```

Лістинг 15 — *код методу updateMarketProduct*

```
private async updateMarketProduct(body:{market_id: number,
good_id: number | undefined, price: number, price_unit:
string, price_compare: number, compare_unit: string}){

  const query = await
MarketProductService.searchByProductAndMarket(body)
  if (query?.market_product_id) return

  return (await MarketProductService.create(body)).response
}
```

На лістингах 14 — 15 останній крок завантаження — додавання зв'язків між магазином та товаром. Зроблено це з метою на кожен зв'язок виставляти окремі дані по цінам на товар в конкретному магазині. Метод insertAllMarketProduct циклічно проходить по всіх елементах в масиві завантажених товарів та на кожен товар викликає метод updateMarketProduct. Цей метод робить запити до бази даних на створення рядку в таблиці market_product.

Результатом виконання цього алгоритму є заповнення всіх таблиць даними про товари, які були нещодавно завантажені. Алгоритм працює порівняно швидко, та якщо мати на увазі що дані які було завантажено є актуальними, то можна зробити висновок що алгоритм ефективно забезпечує оновлення таблиць даними про товари в режимі реального часу, забезпечуючи користувачам актуальну інформацію про наявність товарів, їх характеристики та ціни. Це сприяє покращенню точності та швидкості обробки запитів користувачів, що дозволяє забезпечити високу якість обслуговування та задоволення потреб клієнтів.

3.5.6 Розробка пакету API

Після наповнення бази даних актуальними файлами, необхідно мати можливість надавати їх для використання користувачами. Для цього розроблений API надає шляхи доступу до даних БД. API забезпечує стандартизований спосіб комунікації між серверною частиною та клієнтськими додатками або системами.

Структура пакета api (Див. Рис. 16) така сама як і в інших пакетах: головний файл index який запускає сервер на конкретному порті та роути з контролерами.

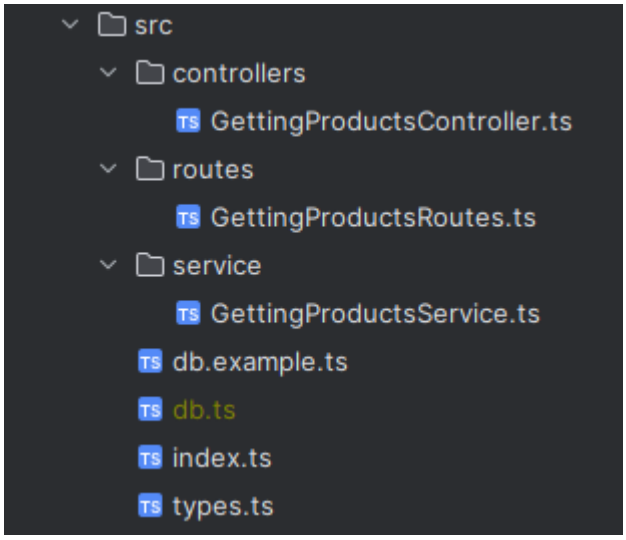

Рис. *16 Файлова структура пакета api*

Під час використання цього пакету користувач має можливості, які візуально зображено на use-case діаграмі (див. Рис. 17). Діаграма в себе включає єдиного актора Користувача та дії по ендпоінтах які дозволяють використовувати інформацію з баз даних:

- **get24.** Отримання 24 товарів. Корисно для формування стрічки рандомних товарів так як 24 карточки товарів націло будуть ділитися як на 3 так і на 4 стовбці.
- **get24withCategory.** Таке саме отримання товарів для стрічки але з конкретною категорією.
- **get1ByPath.** Отримання товару по його шляху. Корисно якщо треба реалізувати на клієнті окрему сторінку товару.
- **getProductsByName.** Пошук товару. Повертає масив товарів які підходять по назві.
- **getMainCategories.** Отримання основних категорій. Це ті які не є підкатегорією жодної з інших категорій.
- **getUnderCategories.** Отримання дочірніх категорій конкретної іншої категорії.
- **getAllCategories.** Отримання переліку всіх наявних категорій в базі даних.
- **getProductPrices/:good_id.** Отримання ціни на конкретний продукт по його ідентифікатору.
- **getSameProducts.** Отримання схожих продуктів. Корисно коли треба обрати альтернативу для конкретного товару.
- **getMarkets.** Отримання всіх наявних магазинів.

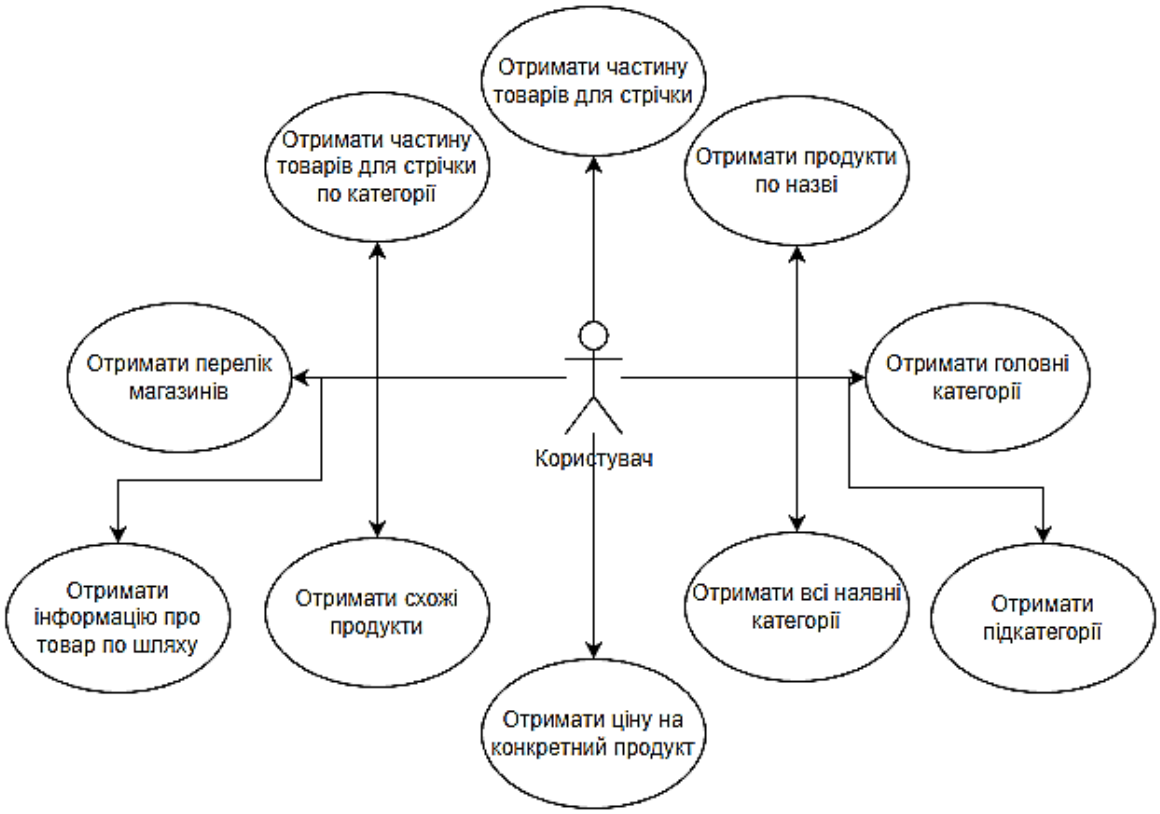

Рис. *17 Use-case діаграма пакета api*

Так як структура контролерів і роутів є єдиною, слід розглянути лише реалізацію сервіс файлу:

Лістинг 16 — *приклад коду запиту на сервер*

```
class GettingProductsService implements
IGettingProductsService{

  async get1ByPath(path:string): Promise<IStatusResponse>{

    const query = await db.query(
      'select * from unique_products ' +
      'where path = $1', [path]
    )

    return {status: 200, response: await
this.addFullCategories(await this.addPrices(query))}
```

```
  }
}
```

В лістингу 16 зображено приклад одного з методів api класу. В цьому прикладі реалізовано отримання повної інформації на конкретний продукт. Робиться це завдяки SQL запиту та декільком додатковим функціям. Завдяки запиту ми отримаємо інформацію з бази даних про цей товар, але необхідно додатково завантажити масив цін на продукт з різних супермаркетів та завантажити повну інформацію про категорії. Хоча в PostgreSQL й доступна можливість користуватися масивами ініціалізуючи такий тип в стовбці, але неможливо задати як елементом цього масиву первинний ключ. З метою не додавати ще одну таблицю зв'язків, так як запитів дуже багато і треба було зменшити навантаження на систему, було прийняте рішення записувати первинні ключі до масиву незважаючи на те, що система не буде вважати ці дані ключами та не буде гарантувати цілісність.

Лістинг 17 — *метод додавання цін на товар*

```
private addPrices = async (query: any) => {
  const arr: { rows: number[] } = { rows: [] };
  for (let i = 0; i < query.rows.length; i++) {
    const arrOfPrices = async () => {
      const productPrices = await
this.getProductPrices(query.rows[i].good_id);
      return { ...query.rows[i], prices:
productPrices.response };
    };
    arr.rows.push(await arrOfPrices());
  }
  return arr;
};
```

В лістингу 17 розписано метод який використовується майже в усіх запитах. Завдяки цьому методу до товарів додається повна інформація про їх ціни та магазини, в яких встановлені такі ціни в зручному правильному форматі. Реалізується це шляхом перебору всіх елементів масиву та виклику для кожної ітерації методу getProductPrices.

Лістинг 18 — *метод getProductPrices*

```
async getProductPrices(good_id:number):
Promise<IStatusResponse>{
  const query = await db.query(
    'select * from market_product ' +
    'join markets on market_product.market_id =
markets.market_id ' +
    'where good_id = $1;', [good_id]
  )

  return {status: 200, response: query.rows}
}
```

В лістингу 18 зображено той самий метод з лістингу 17, котрий отримуючи унікальний ідентифікатор товару завдяки запросу до бази даних шукає всі ціни на товар та додає завдяки джоіну інформацію й про супермаркет, в якому така ціна представлена.

Лістинг 19 — *метод додавання повної інформації про категорії addFullCategories*

```
private addFullCategories = async (query: any) => {
  const arrOutput: any[] = [];
  for (let i = 0; i < query.rows?.length; i++) {
    const getArrCategories = async () => {
      const arrCategories: string[] = [];
      for (let j = 0; j < query.rows[i].categories?.length;
j++) {
        const category = await
CategoryService.getOne(query.rows[i].categories[j]);
        if (category.response === 'There is no category
with this id') {
          console.log('error with category:');
          console.log(query.rows[i]);
        }
        arrCategories.push(category.response);
      }
      return arrCategories;
    };
    const output = { ...query.rows[i], categories: await
getArrCategories() };
    arrOutput.push(output);
```

```
  }
  return arrOutput;
};
```

На лістингу 19 представлений останній важливий метод, який слід було реалізувати. Він дозволяє обходити слабкість PostgreSQL пов’язаною з неможливістю додавати до масиву первинні ключи. Цей алгоритм проходиться по масиву категорій в товарі та відправляє запит на пошук її. По завершенню заповнює масив ті віддає його через ретурн. Завдяки цьому на виході отримаємо з масива ідентифікаторів масив з повною інформацією про категорії, які належать до товару.

В результаті було розроблено пакет api завдяки якому можна за допомогою HTTP запитів отримувати різноманітну інформацію про дані з БД. Завдяки розробці пакету API, можливо забезпечити стабільну та безпечну комунікацію між клієнтськими додатками та базою даних. Це дозволяє уникнути прямого доступу до бази даних і контролювати взаємодію з даними.

3.6 **Розробка дизайну застосунку**

При розробці дизайну майбутнього застосунку було приділено велику увагу ергономіці, користувацькому досвіду та візуальній привабливості. Головною метою було створення інтуїтивно зрозумілого та легкого використання інтерфейсу для кінцевого користувача.

Перш за все було обрано основні кольори для майбутнього застосунку та в цих кольорах було намальовано лого для аналітичного сервіса (Див. Рис. 18). Спокійні зелені кольори та округлі форми повинні привернути до себе майбутніх клієнтів.

Рис. *18 Логотип аналітичного сервісу*

Для забезпечення зручності використання та навігації була створена логічна та інтуїтивно зрозуміла структура меню та розділів. Основна навігація була розміщена зручно та доступно для користувачів, з можливістю швидко переходити між різними розділами та функціональними можливостями застосунку. Дизайн інтерфейсу був створений з урахуванням сучасних трендів та дизайнерських стандартів, з використанням привабливих кольорів, візуально згуртованої тематики та чіткого розташування елементів на екрані. Було забезпечено зручну читабельність тексту, добре підібрані шрифти та іконки для полегшення сприйняття інформації. Окрім візуального аспекту, також було приділено увагу респонсивному дизайну, щоб забезпечити належне відображення застосунку на різних пристроях та розмірах екранів. Це дозволило користувачам комфортно використовувати застосунок на різних пристроях, включаючи комп'ютери, планшети та мобільні телефони.

Головна сторінка (Див. Рис. 19) представляє собою перелік знайденних товарів за запитом зліва та корзиною з обраних товарів користувачем справа, в межах якої будуть проводитись порівняння цін на обрані товари. Зверху повинен бути хедер, який включає в собі перелік категорій, пошук та посилання на інші корисні сторінки. Також на головній сторінці є сортування та хлібні крошки.

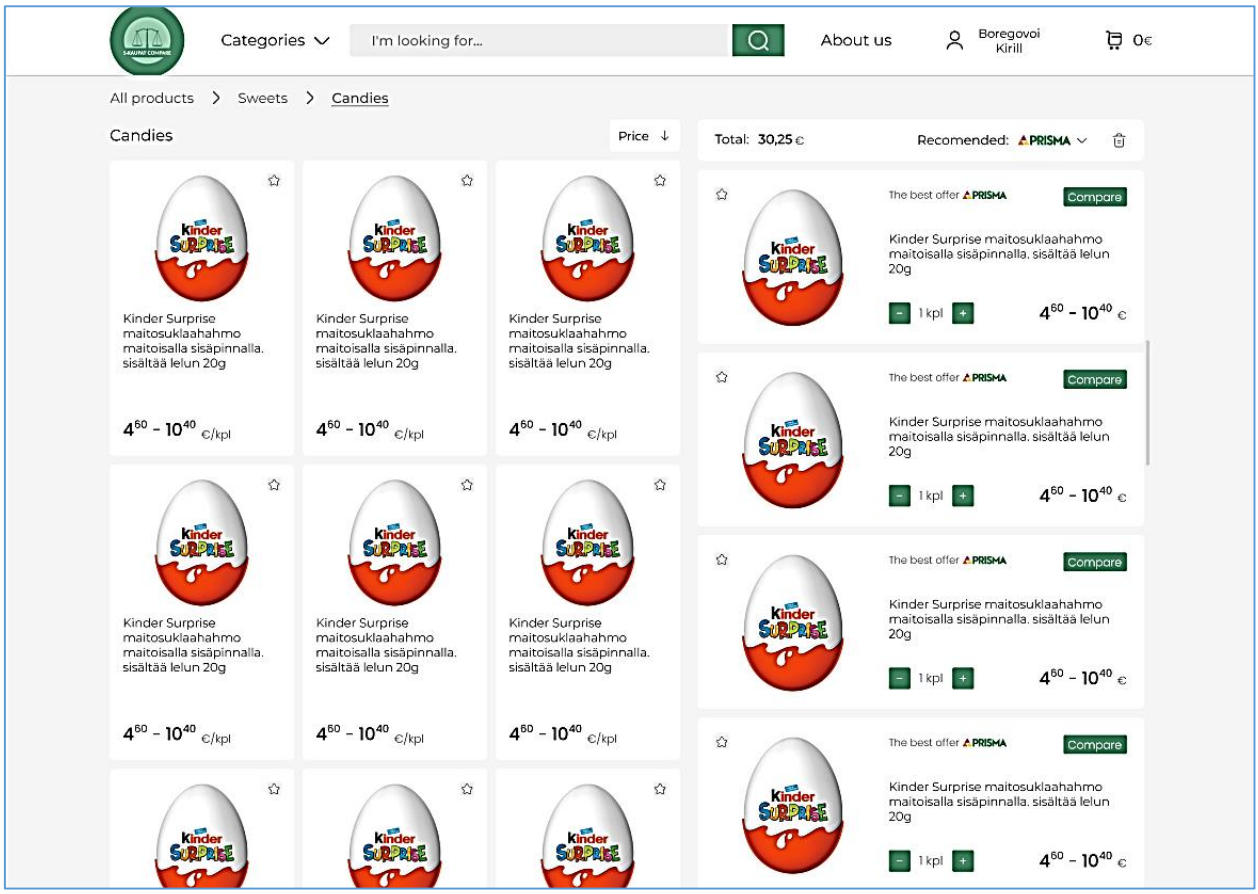

Рис. *19 Головна сторінка сервісу*

Також було спроектовано дизайн для окремої сторінки товару (Див. Рис. 20). На ній міститься вся інформація про товар: зображення, назва, діапазон цін, ціни в різних мережах супермаркетів і так далі. Також додатково знизу сторінки знаходиться перелік зі схожими товарами, які можуть конкурувати з цим товаром за місце в кошику клієнта.

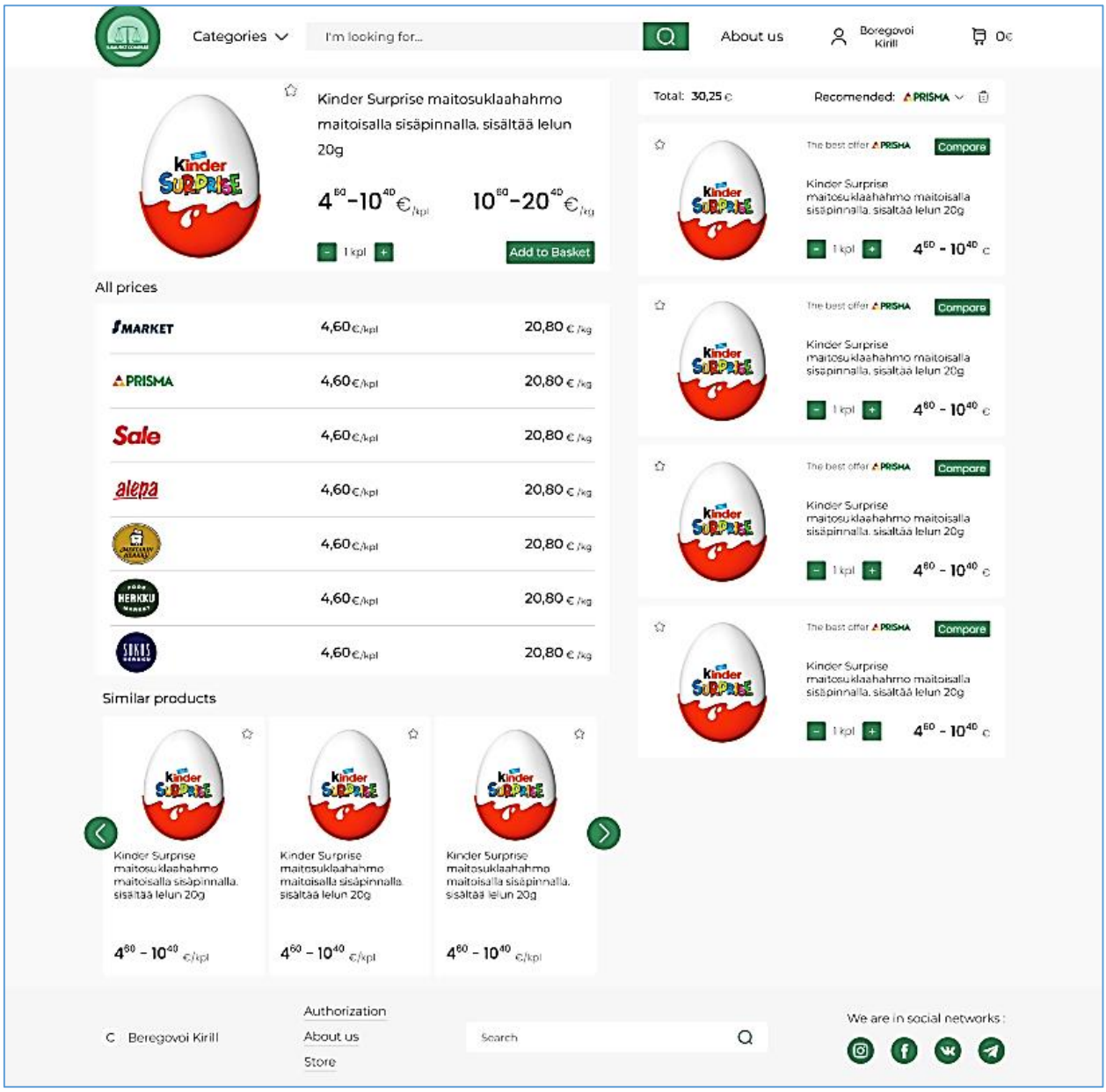

Рис. *20 Внутрішня сторінка товару сервісу*

У процесі розробки дизайну майбутнього застосунку було приділено значну увагу створенню інтуїтивно зрозумілого та привабливого інтерфейсу для користувачів. Застосування принципів ергономіки та користувацького досвіду, спільно з використанням сучасних дизайнерських рішень, допомогло створити застосунок зі зручним та привабливим інтерфейсом. Результатом є інтуїтивно зрозумілий і простий у використанні застосунок, який забезпечує задоволення потреб та очікувань користувачів.

3.7 **Програмна реалізація застосунку**

Клієнтська частина представляє собою застосунок написаний за допомогою типізованої мови TypeScript з фреймворком Nuxt. Це є розширений за можливостями фреймворк Vue, тому в нього такой застосовується компонентний підхід, аналогічна файлова структура (Див. Рис. 21 - 23) та взагалі екосистема. Програмна реалізація застосунку на Nuxt дозволяє створювати веб-додатки з використанням серверного рендерингу, що поліпшує швидкодію та відкликаність додатків. Завдяки розподіленню завантаження між клієнтом та сервером, Nuxt забезпечує швидкий ініціальний рендеринг сторінок та можливість оновлювати вміст без перезавантаження сторінки.

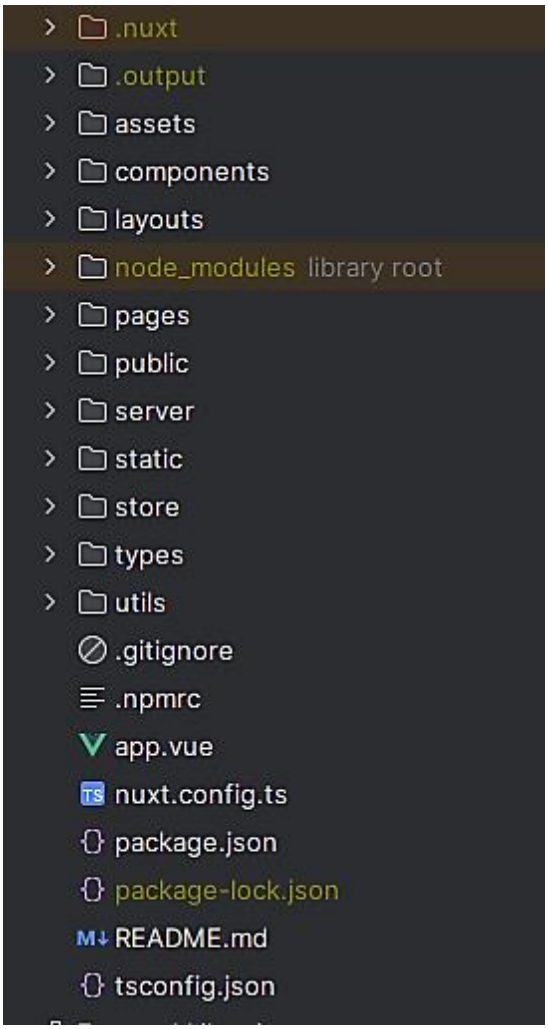

Рис. *21 Загальна файлова система*

Файлова система в Nuxt має певну строгу структуру, яка допомагає організувати різні частини додатку. Основна ідея цієї структури полягає в тому, що окремі файли розташовуються в певних директоріях згідно з їх функціональним призначенням. Компоненти (Див. Рис. 22) є файлами розширення .vue та ніяк не змінюються.

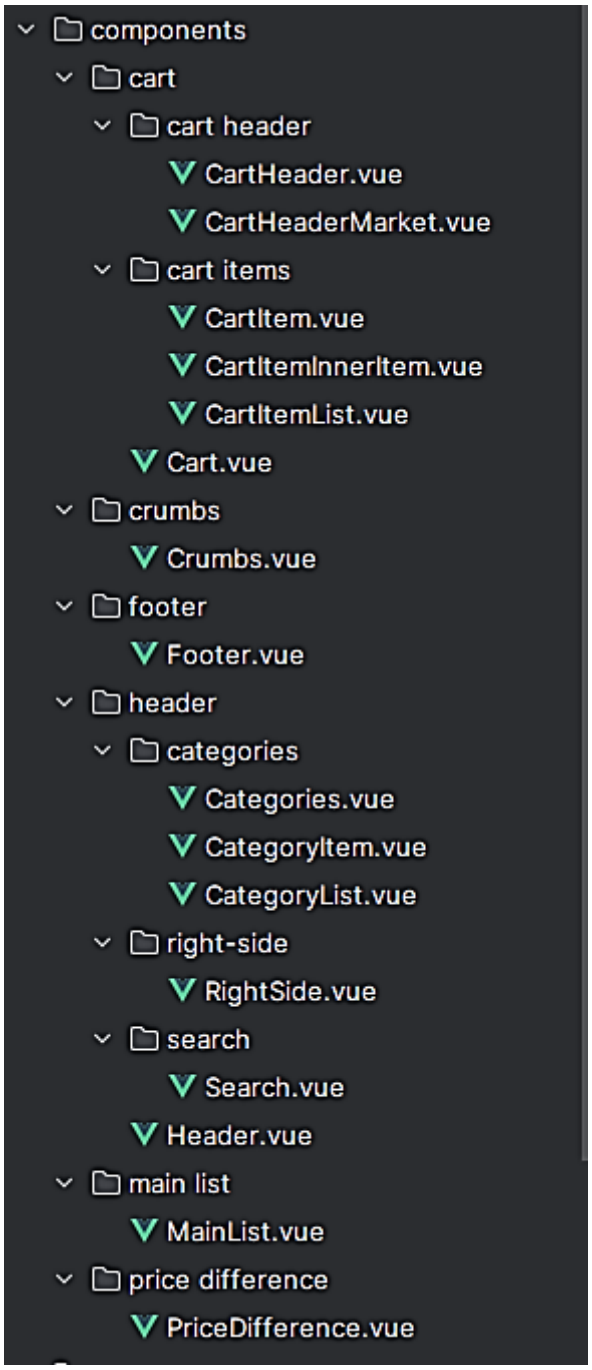

Рис. *22 Файлова система компонентів*

Посторінкову пагінацію (Див. Рис. 20) було реалізовано через кореневу директиву pages в якій index є головною сторінкою. Щоб мати змогу по конкретному адресу отримати параметр через URL треба було зробити структуру як зображено на рисунку 23. За адресою /product/:path можна буде отримати те саме значення path яке й передавалося в шлях.

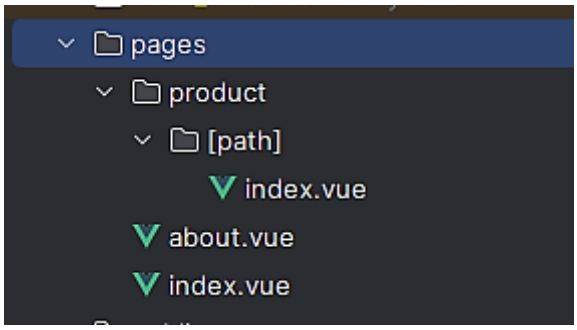

Рис. *23 Файлова система сторінок*

Лістинг 20 — *метод product сторінки з використання параметру*

```
onMounted(async () => {
  const productPath = route.params.path;
  await
fetch(`http://localhost:1500/get/get1ByPath?path=${productP
ath}`)
    .then(response => response.json())
    .then(responseData => {
      product.value = responseData;
    })
    .catch(error => {
      console.error(error);
    });
})
```

На лістингу 20 зображено використання завантаження даних про товар на сторінці за адресою /product/:path. Це стало можливим завдяки тому що було реалізовано структуру за якою можна відловлювати параметри в URL запиті.

При розробці клієнтського застосунку глобальні сховища грали важливу роль, а для їх реалізації я використовував бібліотеку Pinia. Pinia є становим

управлінцем, який надає простий та потужний спосіб організації глобальних сховищ у Nuxt додатку.

Лістинг 21 — *базові методи глобального сховища cart*

```
export const useCartStore = defineStore('cart', () => {
  const cart = reactive<Map<number, object>>(new Map());

  function deleteCartItem(itemId: number):void{
    cart.delete(itemId)
  }
  function getCartItem(itemId: number): any | undefined{
    return cart.get(itemId)
  }
  function setCartItem(itemId: number, product:
object):void{
    cart.set(itemId, product)
  }
  function getAllCartItems(): object[] | undefined{
    return [...cart.entries()]
  }
}
```

В лістингу 21 я створив глобальне сховище cart та додав до нього базові методи по додаванню, отриманню та видаленню елементів. Завдяки такому підходу логіка відокремлена, а не знаходиться всередині компонентів, тому у випадку чого можна буде змінювати інтерфейс на новий або додавати нові можливості без риску зламати логіку.

Для аналітичного сервісу є важливим порівняння цін та вибір найдешевшого магазину як для окремого товару так і для купи товарів в корзині. Тому було реалізовано методи по порівнянню цін та їх рахуванню.

Лістинг 22 — *метод findCheapestMarket сховища cart*

```
function findCheapestMarket(): {name: string, price:
number} {
  const cartPrices: Map<string, number> = new Map<string,
number>();

  const commonMarkets = findCommonMarkets();
  if (commonMarkets.length === 0) return;
```

```
  // Calculate prices
  const cartProducts = Array.from(cart.values());
  for (let i = 0; i < cartProducts.length; i++){
    cartProducts[i].product.prices.forEach((market: {name:
string, price: number}) => {
      cartPrices.set(market.name,
((cartPrices.get(market.name) || 0) + market.price || 0) *
cartProducts[i].metadata.count);
    });
  }

  // Cut common markets
  let commonMarketPrices = new Map<string, number>();
  commonMarkets.forEach(market => {
    commonMarketPrices.set(market,
cartPrices.get(market)!);
  });

  // Find cheapest prise
  const cheapestMarket =
getSmallestKey(commonMarketPrices);
  return { name: cheapestMarket!, price:
cartPrices.get(cheapestMarket!).toFixed(2) }
}
```

У лістингу 22 зображено метод findCheapestMarket, який дозволяє аналізуючи купу обраних товарів в кошику знаходити супермаркет, в якому купити всі ці продукти буде дешевше за всіх інших. Алгоритм починається з ініціалізації cartPrices, в який буде записуватись пари магазин ціна. Потім отримуємо перелік магазинів, я яких є в наявності всі товари з корзини з допомогою функції findCommonMarkets (Див. Лістинг 23). Потім йде циклічне підсумовування цін всих на обрані товари в кожному з маркетів, після чого визивається getSmallestKey (Див. Лістинг 24), котрий з переліку підсумувань знаходить найвигіднішу пропозицію та повертає її.

Лістинг 23 — *метод findCommonMarkets сховища cart*

```
function findCommonMarkets() {
  let initialArrayInitiated = false;
  let commonMarkets = new Array<string>();
```

```
  const cartProducts = Array.from(cart.values());

  for (let i = 0; i < cartProducts.length; i++) {
    const currentProductMarkets =
cartProducts[i].product.prices.map(market => market.name);

    if (!initialArrayInitiated) {
      commonMarkets = currentProductMarkets;
      initialArrayInitiated = true;
      continue;
    }

    commonMarkets = commonMarkets.filter(e =>
currentProductMarkets.includes(e));
  }

  return commonMarkets;
}
```

Лістинг 23 представляє функцію який формує перелік магазинів, які мають в наявності всі товари, котрі користувач поклав до своєї корзини. Алгоритм циклічно проходить по всім даним з корзини на кожній ітерації фільтруючи початковий масив наявних магазинів залишаючи лише елементи, які є у продукта на даній ітерації. На кожній ітерації циклу масив зменшується, містячи лише загальні ринки між товаром у поточній ітерації та попередніми ітераціями. По завершенню функція повертає масив тих магазинів, які залишилися після перевірки.

Лістинг 24 — *метод getSmallestKey сховища cart*

```
function getSmallestKey(map: Map<string, number>): string |
undefined {
  let smallestKey: string | undefined;
  let smallestValue: number | undefined;

  for (const [key, value] of map.entries()) {
    if (smallestValue === undefined || value <
smallestValue) {
      smallestKey = key;
      smallestValue = value;
    }
  }
```

```
  return smallestKey;
}
```

Функція з лістинга 24 призначена для отримання ключа з найменшим значенням в списку. В ньому запускається цикл, в якому на кожній ітерації знаходиться ключ з найменшим значенням у списку шляхом порівняння значення магазина на цієї ітерації із найменшим значенням, що було встановлено раніше.

В процесі розробки клієнтського застосунку було використано глобальні сховища для управління станом додатку. Ці глобальні сховища були розроблені з урахуванням кращих практик та загальних принципів архітектури. Кожне глобальне сховище мало схожу структуру, включаючи розділення на стани, мутації, дії та геттери. Це розділення дозволяло зберігати стан окремо від компонентів та бізнес-логіки, а також надавало зручний спосіб взаємодії з даними. При розробці інших глобальних сховищ було дотримано принципу єдності структури та способу взаємодії з даними. Це дозволяло легко розширювати функціональність додатку, оскільки розробники мали загальне розуміння процесу роботи з глобальними сховищами. Таким чином, завдяки стандартизованій структурі глобальних сховищ та уважному проектуванню методів, було досягнуто зручності, чистоти коду та ефективності управління станом додатку.

Лістинг 25 — *код компоненту Crumbs*

```
<template>
  <div class="crumbs">
    <span
      @click="useCategoriesStore().selectedCategory =
undefined"
    >
      All products
    </span>
    <div
      class="crumbs-path"
      v-for="category in
useCategoriesStore().crumbsSelectedCategory"
```

```
      @click="crumbClickHandler(category)"
    >
      <Icon name="iconamoon:arrow-up-2-light" size="20"
color="#777777" :rotate="1"/>
      <span>{{category?.name}}</span>
    </div>
  </div>
</template>

<script lang="ts" setup>
import { useCategoriesStore } from "~/store/categories";
import {ICategory} from "~/types/category";

function crumbClickHandler(category: ICategory):void{
  useCategoriesStore().selectedCategory = category
}
</script>
```

В лістингу 25 представлене типове використання графічних компонентів інтерфейсу. Інформація та логіка знаходиться в глобальному сховищі useCategoriesStore, з якого компонент бере інформацію та за необхідністю змінює її. Завдяки двосторонньому зв’язуванню все що потрібно зробити це змінити реактивну змінну. Після цього все що було з нею пов’язано перерендериться автоматично. Саме тому компоненти виглядають начебто пусто.

В результаті програмної реалізації клієнтського застосунку було досягнуто високого рівня структурованості, модульності та повторного використання коду. Глобальні сховища забезпечили відокремлення бізнес-логіки від інтерфейсу, що сприяло підтримці чистоти та розширюваності кодової бази. При розробці застосунку було використано мову програмування TypeScript, що дозволило забезпечити типізацію та покращити роботу з кодом завдяки перевірці типів на етапі компіляції. Це знизило кількість помилок та сприяло підвищенню стабільності та надійності застосунку. Використання передових технологій та кращих практик розробки допомогло забезпечити якість та ефективність програмного рішення.

3.8 **Практичне використання додатку**

Коли остаточно розроблена функціональність головної сторінки, де реалізована аналітика цін в різних магазинах, можна приступити до практичного використання. Завдяки прикладу використання (Див. Рис. 24) можна зрозуміти, яким чином працює додаток.

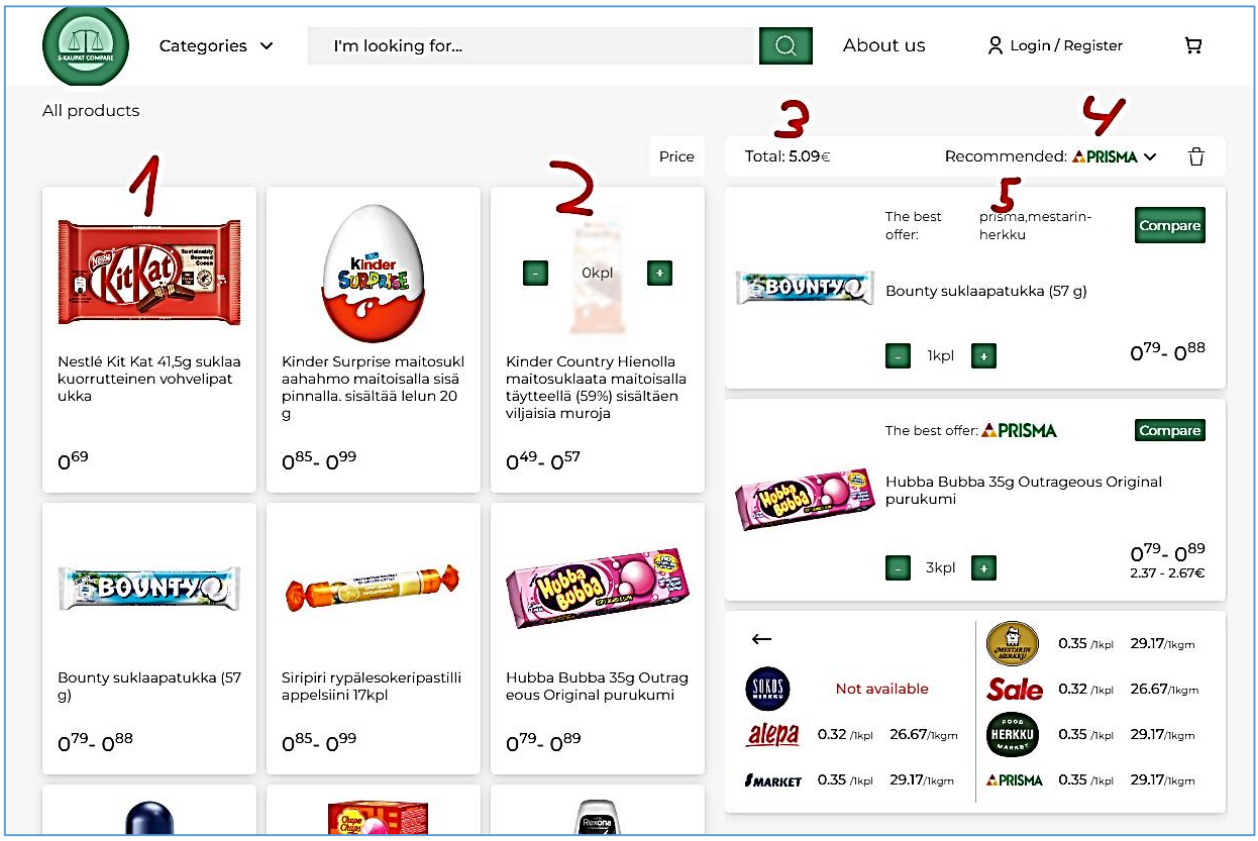

Рис. *24 Приклад використання застосунку*

На прикладі використання (Див. Рис. 24) є такі помітки:

1. Карточка товару у списку.
2. Карточка товару при наведенні.
3. Ціна за обрану купу товарів в обраному супермаркеті.
4. Обраний супермаркет. За замовчуванням динамічно обирається найдешевший з усіх можливих.
5. Карточка товару в корзині.

Щоб додати товар до кошика слід навести курсор на товар зі списку (Див. Рис. 24 1) після чого на карточці відобразиться додаткова функціональність (Див. Рис. 24 2) з можливістю додавати або віднімати товари з корзини. Після додавання першого елементу до корзини одразу ж проходить порівняння цін у всіх мережах та найвигідніша мережа обирається за замовчуванням (Див. Рис. 24 4), а ціна з цієї мережи поміщається в хедер корзини (Див. Рис. 24 3). Також є й порівняння окремо по кожному товару. В карточці товару в корзині (Див. Рис. 24 5) проходить порівняння та відображається логотип магазину з найнижчою ціною, або перелік магазинів якщо найнижча ціна однакова в кількох магазинах. Також для того щоб порівняти всі існуючи ціни на конкретний товар можна натиснути на зелену кнопку порівняння в самій карточці товару в корзині. Тоді інтерфейс картки зміниться на той який можна побачити останнім у переліку в корзині (Див. Рис. 24 5)

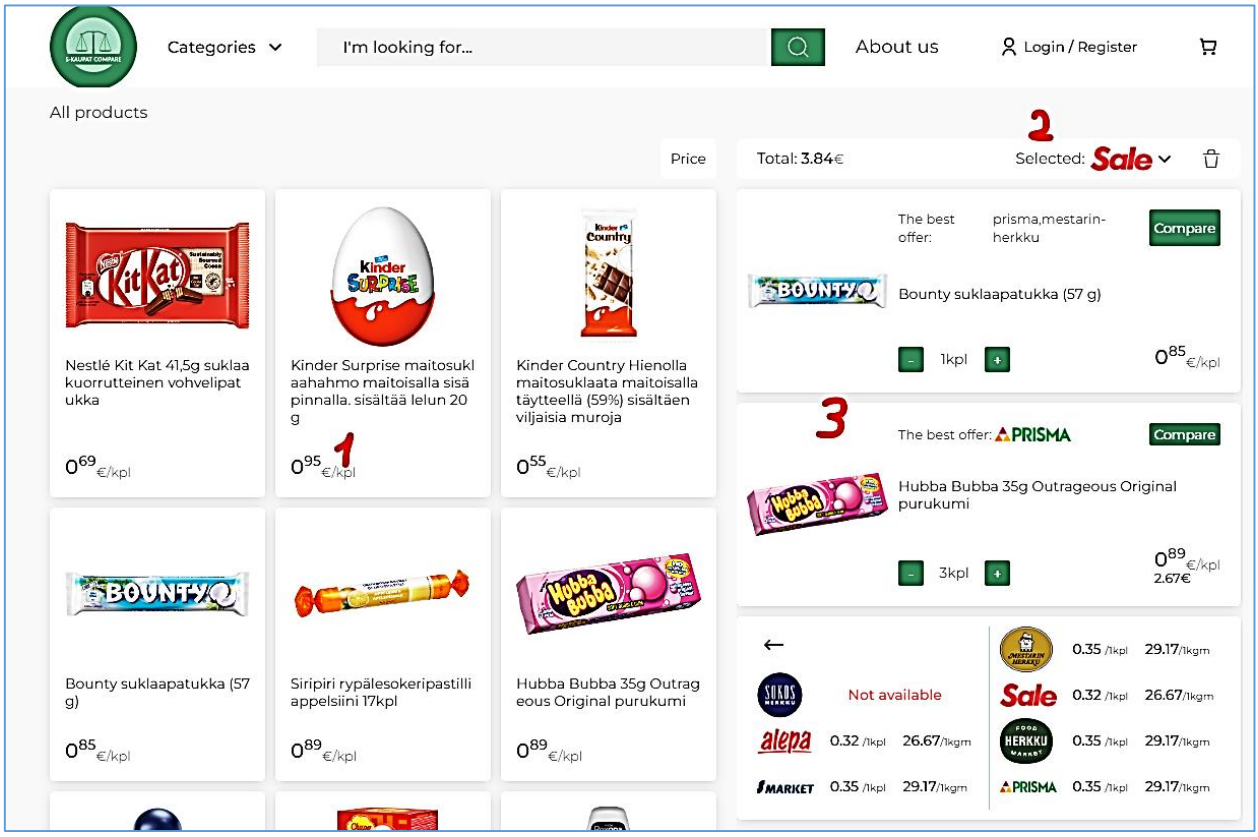

Рис. *25 Використання застосунку з ручним вибором магазина*

Замість автоматичного порівняння можна вручну обрати цікавий для користувача магазин (Див. Рис. 25). Ручний вибір проходить шляхом натискання на відповідну кнопку (Див. Рис. 25 2) та вибір необхідного магазина зі списку, що випадає. Після встановлення магазина вручну застосунок більше не буде виводити діапазони цін в переліку товарів (Див. Рис. 25 1), або в корзині товарів (Див. Рис. 25 3). Замість цього, будуть підставлятися ціни саме з того магазину, який обрав користувач.

Досліджене практичне використання розробленого додатку і його взаємодію з користувачами. Застосування додатку дозволить користувачам зробити обґрунтовані вибори та знайти найбільш вигідні пропозиції на ринку. Продемонстровано головну функціональність застосунку — порівняння цін та пропозицій.

ВИСНОВКИ

1. У процесі виконання кваліфікаційної роботи бакалавра була проведена робота з літературними джерелами, які корелюються з темою роботи, а саме: великі дані, аналітика, оптимізація, ціноутворення.
2. Проведено аналіз існуючих технологій серверної та клієнтської частини та порівняні за багатьма критеріями.
3. Спроектований дизайн з використанням сучасних підходів користувацького інтерфейсу та користувацького досвіду.
4. Реалізована серверна частина, яка може завантажувати дані та повертати їх у вигляді роботи API, завдяки чому, дані можна використовувати не лише конкретно в цьому аналітичному сервісі.
5. Розроблений аналітичний сервіс, який надає користувачам зручні інструменти для вибору та купівлі товарів. Сервіс дозволяє збирати та аналізувати інформацію про товари з різних мереж супермаркетів, надає рекомендації та допомагає здійснити покупку з максимальною ефективністю.
6. Результати експериментального використання сервісу показали його ефективність та корисність для користувачів. Застосування аналітичних методів та алгоритмів дозволило отримати точні рекомендації та покращити процес вибору та купівлі товарів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Martin Kleppmann Designing Data-Intensive Applications : O'Reilly Media, 16 березня 2017 р. С. 616.
2. Anil Maheshwari Data Analytics Made Accessible : Apress 1 травня 2014 р. С. 448.
3. Nathan Marz and James Warren Big Data: Principles and Best Practices of Scalable Realtime Data Systems : Manning Publications 10 травня 2015 р. С. 328.
4. Leon Shklar and Rich Rosen Web Application Architecture: Principles, Protocols, and Practices : Wiley 27 квітня 2009 р. С. 448.
5. Erik Hanchett and Benjamin Listwon Vue.js in Action : Manning Publications 12 березня 2018 р. С 304.
6. William Poundstone Priceless: The Myth of Fair Value (and How to Take Advantage of It) : Hill and Wang 1 січня 2010 р. С. 336.
7. PostgreSQL Documentation 15: PostgreSQL 15.3 Documentation : веб-сайт. URL: https://www.postgresql.org/docs/current/index.html (дата звернення: 10.06.2023).
8. Index | Node.js v19.9.0 Documentation : веб-сайт. URL: https://nodejs.org/docs/latest-v19.x/api/ (дата звернення: 10.06.2023).
9. Express - Node.js web application framework : веб-сайт. URL: https://expressjs.com/ (дата звернення: 10.06.2023).
10. Documentation · Nuxt : веб-сайт. URL: https://nuxt.com/docs (дата звернення: 10.06.2023).
11. Introduction | Vue.js : веб-сайт. URL: https://vuejs.org/guide/introduction.html (дата звернення: 10.06.2023).
12. Diagrams : веб-сайт. URL: https://app.diagrams.net/ (дата звернення: 10.06.2023).
13. Introduction | Pinia : веб-сайт. URL: https://pinia.vuejs.org/introduction.html (дата звернення: 10.06.2023).

14. Angular vs React vs Vue: Core Differences | BrowserStack : веб-сайт. URL: https://www.browserstack.com/guide/angular-vs-react-vs-vue (дата звернення: 10.06.2023).
15. Figma: the collaborative interface design tool. : веб-сайт. URL: https://www.figma.com/ (дата звернення: 10.06.2023).
16. Lau Tiam Kok Hands-on Nuxt.js Web Development : Packt Publishing, 14 квітня 2020 р. С. 698.
17. Solomon Eseme Architecting Vue.js 3 Enterprise-Ready Web Applications : Packt Publishing, 14 квітня 2023р. С. 272.
18. Andrew Mead Learning Node.js Development : Packt Publishing, 31 січня 2018 р. С. 658.
19. Береговой К. В., магістрант, Безверхий А. І., доцент — науковий керівник. Аналітичний сервіс для вибору і купівлі товарів. *Молода наука-2023* : зб. наук. праць студентів, аспірантів і молодих вчених. Запоріжжя : ЗНУ, 2023. Т. 5. С. 75–76.