Національний університет «Полтавська політехніка імені Юрія Кондратюка»

(повне найменування вищого навчального закладу)

Навчально-науковий інститут інформаційних технологій та робототехніки

(повна назва факультету)

Кафедра комп’ютерних та інформаційних технологій і систем

(повна назва кафедри)

**Пояснювальна записка**

**до дипломного проєкту (роботи)**

магістра

(освітньо-кваліфікаційний рівень)

на тему

Інформаційна система електронної комерції на основі веб-технологій

Виконав: студент 6 курсу, групи 601-ТН спеціальності

122 Комп’ютерні науки

(шифр і назва напряму)

Бочкарь В.О

(прізвище та ініціали)

Керівник Здоренко Ю.М

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Полтава – 2023 року

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ   
« ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»**

**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ**

**КАФЕДРА КОМП’ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ**

**КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА**

**спеціальність 122«Комп’ютерні науки»  
на тему**

**«Інформаційна система електронної комерції на основі веб-технологій»**

**Студента групи 601-ТН Бочкаря Володимира Олександровича**

Керівник роботи

кандидат технічних наук,

доцент Здоренко Ю.М

Завідувач кафедри

Кандидат технічних наук,

доцент Головко Г.В.

Полтава – 2023

# РЕФЕРАТ

Кваліфікаційна робота магістра: 86 с., 21 малюнок, 2 додатки, 26 джерел.

**Об’єкт дослідження:** процес проектування, розробки інформаційної система електронної комерції на основі веб-технологій.

**Мета роботи:** розробка інформаційної системи електронної комерції для удосконалення процесу продажу товарів комерційною організацією шляхом використання програмних рішень на основі веб-технологій.

**Методи:** аналіз існуючих технологій для створення інформаційних систем та засобів візуалізації інформації з використанням веб-технологій, обґрунтування необхідності створення інформаційної системи електронної комерції, проектування та розробка баз даних, розробка архітектури клієнт-серверної взаємодії, використання алгоритмів машинного навчання.

**Ключові слова**: інформаційна система, е-комерція, веб-додаток, система рекомендацій, онлайн торгівля.

# ANNOTATION

Qualification work of master’s degree: 86 p., 21 pictures, 2 applications, 26 sources.

**Research Object:** the process of designing and developing an e-commerce information system based on web technologies.

**Objective of the work**: the development of an e-commerce information system to improve the process of selling goods by a commercial organization through the use of software solutions based on web technologies.

**Methods:** analysis of existing technologies for creating information systems and visualization tools using web technologies, justification of the need for creating an e-commerce information system, design and development of databases, development of client-server interaction architecture, use of machine learning algorithms.

**Keywords:** information system, e-commerce, web application, recommendation system, online trading.

# ЗМІСТ

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ 7](#_Toc154174207)

[ВСТУП 8](#_Toc154174208)

[РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД СУЧАСНИХ ІНФОРМАЦІНИХ СИСТЕМ Е-КОМЕРЦІЇ 10](#_Toc154174209)

[1.1 Аналіз предметної області 10](#_Toc154174210)

[1.2 Огляд сучасних трендів е-комерції та онлайн торгівлі 13](#_Toc154174211)

[1.3 Обґрунтування вибору технічних засобів для розробки клієнтської частини додатку 17](#_Toc154174212)

[1.4 Обґрунтування вибору технічних засобів для розробки серверної частини додатку 31](#_Toc154174213)

[1.5 Висновки до розділу 1 35](#_Toc154174214)

[РОЗДІЛ 2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ Е-КОМЕРЦІЇ ТА СИСТЕМИ РЕКОМЕНДАЦІЇ КОНТЕНТУ 37](#_Toc154174215)

[2.1 Проектування структури веб-додатку е-комерції 37](#_Toc154174216)

[2.2 Огляд систем рекомендації контенту 39](#_Toc154174217)

[2.3 Висновки до розділу 2 47](#_Toc154174218)

[РОЗДІЛ 3 РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ Е-КОМЕРЦІЇ 49](#_Toc154174219)

[3.1 Проектування та реалізація бази даних для платформи е-комерції 49](#_Toc154174220)

[3.2 Реалізація серверної частини веб-додатку для е-комерції 54](#_Toc154174221)

[3.3 Реалізація клієнтської частини веб-додатку для е-комерації 60](#_Toc154174222)

[3.4 Реалізація системи рекомендації товарів 62](#_Toc154174223)

[3.5 Висновки до розділу 3 64](#_Toc154174224)

[**ВИСНОВКИ 66**](#_Toc154174225)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 67**](#_Toc154174226)

[**ДОДАТОК А ВИХІДНИЙ КОД ПРОГРАМИ 70**](#_Toc154174227)

[**ДОДАТОК Б РЕЗУЛЬТАТИ АПРОБАЦІЇ 82**](#_Toc154174228)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ ІТЕРМІНІВ

**Е-комерція** – електронна комерція

**ІС** – інформаційна система

**API** – application programming interface

**БД** – база даних

**MVVM** – model view view-model

**MVC** – model view controller

**FCP** – first contentful paint

**LCP** – largest contentful paint

**CLS** – cumulative layout shift

**CDN** – content delivery network

**DOM** – document object model

**TTFB** – time to first byte

**TBT** – total blocking time

# ВСТУП

В сучасному інформаційному суспільстві електронна комерція визначає нові парадигми бізнесу, надаючи компаніям можливість динамічного розвитку та пристосування до потреб споживачів. Завдяки швидкому технологічному розвитку та все більшій доступності Інтернету, важливість створення інноваційних та ефективних інформаційних систем електронної комерції стає надзвичайно актуальною.

Метою даної дипломної роботи є вивчення, розробка та модернізація інформаційної системи електронної комерції, що базується на передових веб-технологіях. У світлі зростання конкуренції та зміни підходів до взаємодії з клієнтами, необхідність надання персоналізованих послуг виходить на передовий план.

Одним із основних напрямків дослідження є використання сучасних та провідних технологій для побудови клієнтської та серверної частини веб-додатку, що дозволяє побудувати ефективну, масштабовану та надійну систему електронної комерції. Це зумовлено потребою в уніфікованій та високопродуктивній платформі, здатній забезпечити зручний та швидкий досвід для кінцевих користувачів.

Окрім цього, робота включає в себе використання машинного навчання та нейромереж для підвищення якості та точності рекомендацій користувачам. Персоналізація стає важливим чинником не лише для задоволення потреб користувачів, але й для оптимізації стратегій продажів та розвитку власного бізнесу. В роботі розглянута розробка модулю рекомендаційної моделі, що базується на методах машинного навчання, зокрема на алгоритмах колаборативної фільтрації. Описаний модуль використовує історію взаємодії користувачів з ІС для прогнозування їхніх майбутніх виборів, що дає змогу передбачати і рекомендувати найбільш релевантний товар користувачу.

Подібний підхід до електронної комерції забезпечує переваги як для користувачів ІС — зниження інформаційного шуму, надання швидкого доступу до бажаної інформації так і для бізнесу — покращуючи ефективність продажів та стимулюючи лояльність споживачів.

Результати дослідження можуть не лише зробити вагомий внесок у сучасну електронну комерцію, але й стати важливим етапом в розвитку та вдосконаленні систем електронної торгівлі в майбутньому. Такий підхід відкриває нові перспективи для бізнесу та дослідницької спільноти, створюючи можливість для більш ефективного використання потенціалу електронної комерції в епоху цифрових інновацій. Робота враховує актуальні виклики електронної комерції та пропонує практичні рекомендації для впровадження та підтримки рекомендаційних систем в онлайн-торгівлі.

Дослідження проведені в рамках дипломної роботи були апробовані на конференції «Інновації та перспективні шляхи розвитку інформаційних технологій» 6 грудня 2023 року та лягли в основу написання тез до конференції за темою «Методи забезпечення доступності інформаційних ресурсів на основі використання мікросервісної архітектури веб-додатків».

# РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД СУЧАСНИХ ІНФОРМАЦІНИХ СИСТЕМ Е-КОМЕРЦІЇ

## 1.1 Аналіз предметної області

Е-комерція – це широкий набір інтерактивних методів ведення діяльності з надання споживачам товарів та послуг. Під е-комерцією розуміють будь-які форми ділових операцій, де сторони взаємодіють через електронні технології, а не в процесі фізичного обміну чи контакту.

ЕК як важлива галузь бізнесу, активно використовує потужність Інтернету для здійснення торгівлі товарами та послугами. Заснована на використанні веб-технологій, е-комерція розгортається в широкому спектрі форм торгівлі, включаючи B2C (Business-to-Consumer), B2B (Business-to-Business), C2C (Consumer-to-Consumer), C2B (Consumer-to-Business) та G2C (Government-to-Consumer) [1].

У контексті дипломної роботи, основна увага буде зосереджена на розробці інформаційної системи для покращення B2C та B2B взаємодії. Система буде орієнтована на оптимізацію електронного магазину, сприяючи покращення користувацького інтерфейсу та управлінню продуктами для бізнес-клієнтів.

Е-комерція перетворила спосіб, яким підприємства та споживачі взаємодіють між собою, розширюючи глобальні ринки та надаючи доступ до широкого спектру товарів та послуг. Цей сектор відзначається високою динамікою та конкретністю, що вимагає постійного удосконалення та інновацій у використанні технологій.

Важливим елементом дослідження буде вивчення та аналіз сучасних тенденцій у розвитку е-комерції, зокрема використання штучного інтелекту, аналізу даних та персоналізації в електронних магазинах. Особлива увага приділятиметься вивченню кращих практик у створенні інтерфейсів, які забезпечують найвищий рівень зручності та доступності для різних категорій користувачів. Крім того розглянуто використання мікросервісної архітектури для побудови модульних додатків, що дає можливість подальшого масштабування та розширення системи [2].

Важливо зазначити, що ЕК являє собою широку галузь, що включає в себе різноманітні складові, які допомагають підприємствам та споживачам ефективно взаємодіяти та проводити трансакції. Основні складові електронної комерції включають (Рис.1.1):

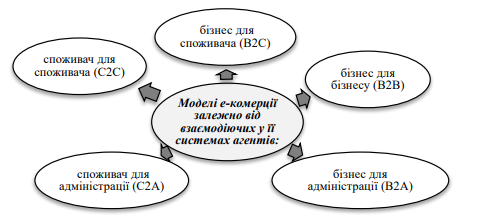


Рисунок.1.1 Моделі е-комерції

Модель «бізнес для споживача» (В2С) визначається як електронна роздрібна торгівля, що передбачає взаємодію компанії (юридичної особи) з кінцевим споживачем (фізичною особою) з метою продажу товарів, послуг або інформації. Ця модель електронної комерції сприяє освоєнню нових сегментів ринку, підвищенню рівня реагування, наданню нових послуг, зниженню витрат, підтримці бізнес-процесів у режимі онлайн, тісному партнерству та цілодобовому доступу [1]. Саме такий вид взаємодії буде детально розглянуто в дипломній роботі, ІС розроблена в рамках даної роботи являє собою програмне рішення для ефективного ведення подібного роду діяльності.

У цьому контексті, модель «бізнес для бізнесу» (В2В) представляє обмін товарами, послугами або інформацією між компаніями, не включаючи в цей процес кінцевого фізичного споживача. Використання цієї моделі сприяє підтримці ділової взаємодії між корпораціями та підприємствами.

Модель «бізнес для адміністрації» (В2А) передбачає взаємодію бізнесу з адміністрацією, охоплюючи ділову взаємодію комерційної структури з державними організаціями на різних рівнях влади, від місцевої до міжнародної. Ця модель сприяє ефективній взаємодії між бізнесом і сферою публічного управління.

Модель «споживач для адміністрації» (С2А) передбачає взаємодію державної структури зі споживачами, зокрема в соціальній та податковій сферах. Цей напрямок, хоча й менш розвинений, має високий потенціал для подальшого росту.

Модель «споживач для споживача» (С2С) визначає взаємодію між користувачами для обміну комерційною інформацією, досвідом та участю в аукціонах. Ця модель підтримує електронні торгові операції між фізичними особами та визначається як важлива складова будь-якого бізнесу, де компанії, які активно використовують інформаційно-комунікаційні технології, мають значущі конкурентні переваги [1].

У контексті дипломної роботи з інформаційної системи електронної комерції на основі веб-технологій, велика увага буде приділена оптимізації електронного магазину та поліпшенню взаємодії у сегментах B2C та B2B.

Очікується, що розроблена система буде інтегрувати ці ключові складові для створення зручного, безпечного та ефективного середовища для комерції та торгівлі.

Така ІС повинна відповідати усім вимогам і викликам які існують на сучасному ринку, а саме:

* швидкість та взаємодія, відмовостійкість;
* інтуїтивний інтерфейс взаємодії з системою;
* кросс-девайсність, адаптивність, мобільність;
* адаптивність під користувацькі вимоги (системи рекомендацій, персоналізація).

## 1.2 Огляд сучасних трендів е-комерції та онлайн торгівлі

Швидкість і висока продуктивність насамперед є критично важливими аспектами ІС е-комерції, оскільки вони безпосередньо впливають на взаємодію з користувачем, коефіцієнти конверсії та загальний успіх бізнесу електронної комерції. Нижче ми розглянемо важливість цих аспектів, основні методи їх забезпечення та їхній вплив на функціонування та успішність онлайн бізнесу.

Стрімкий розвиток мережевих технологій в XXI столітті істотно поліпшив можливості отримання швидкого доступу до інформації. Завдяки мережі Інтернет на отримання потрібної інформації середньостатистичний користувач має витратити всього декілька секунд, якщо не менше. Така наддоступність сприяє також і не готовністю користувача чекати завантаження сторінки довше певного періоду часу. Дослідження проведені Google [3], демонструють який відсоток користувачів «відсіюється» з кожною секундою очікування завантаження веб-сторінки (Рис. 1.2). З графіку видно, що більше половини респондентів не готові очікувати більше 5 секунд.

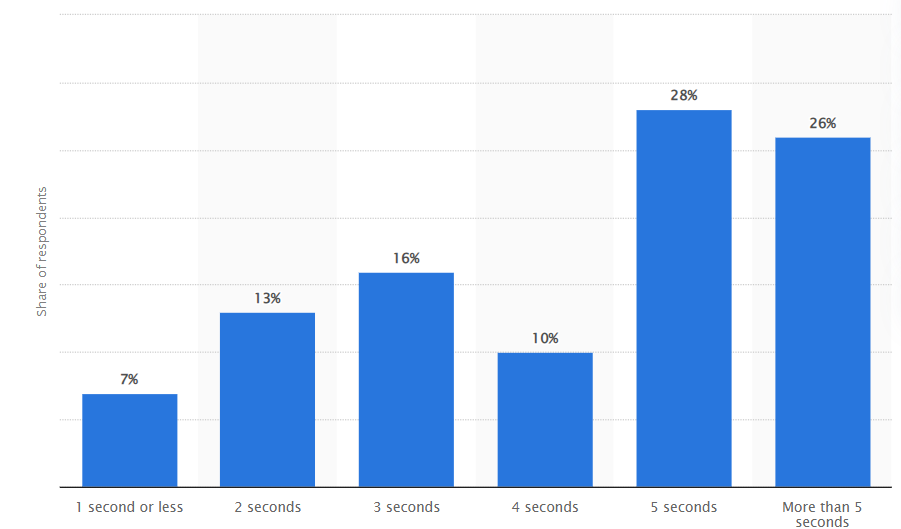


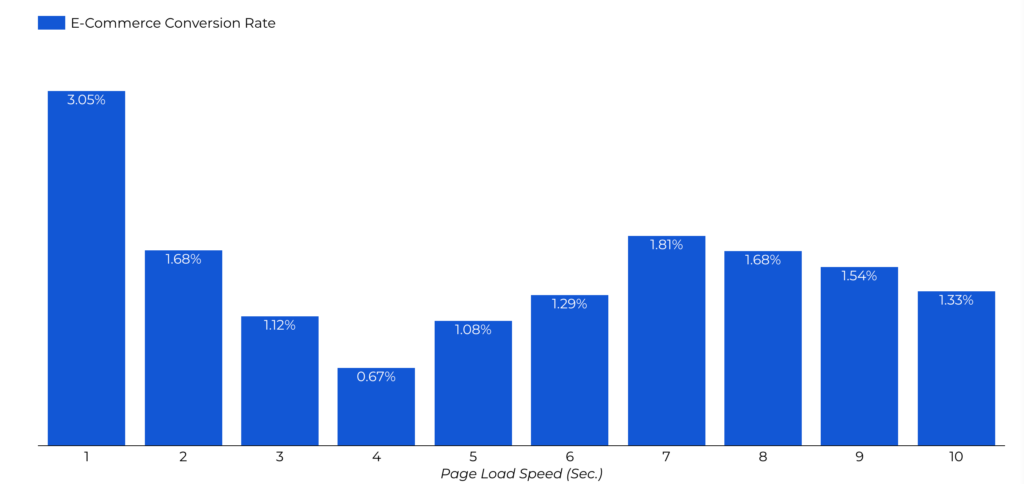
Рисунок.1.2 Відсоток користувачів які покидають веб-сторінку після кожної секунди очікування

Швидкодія є однією з найважливіших змінних, яка визначає рейтинг веб-сайту в Google та інших пошукових сервісів, його трафік і коефіцієнт конверсії. Статистика показує, що 40% відвідувачів залишать веб-сайт, якщо він завантажується довше трьох секунд.

Отже, чим довше веб-сайт завантажується, тим вищим буде його показник відмов — високий показник відмов в свою чергу вказує пошуковим системам, що читачі не вважають вміст сторінки корисним, що призводить до зниження рейтингу. Саме тому швидкість завантаження сторінки корелює з показниками конверсії онлайн-магазинів.

Потрібно зробити уточнення, що під конверсією в інтернет-маркетингу розуміють відношення числа відвідувачів сайту, які виконали на ньому певні цільові дії (придбання, реєстрацію, підписку, відвідування певної сторінки сайту, перехід по рекламному посиланню тощо), до загального числа відвідувачів сайту, виражене у відсотках.

Дослідження демонструють таку залежність конверсії від швидкості завантаження сторінки [4] (Рис 1.3).

  
Рисунок 1.3 Показники конверсії для кожної секунди очікування завантаження веб-сторінки

З графіку видно, що швидкість завантаження сторінки безпосередньо впливає на коефіцієнт конверсії. Дослідження показують, що навіть невеликі затримки часу завантаження можуть значно знизити ймовірність того, що користувач зробить покупку. Враховуючи це, можна зробити висновок, що оптимізація швидкості завантаження сторінки має вирішальне значення для забезпечення високих коефіцієнтів конверсії в ECIS.

З технічної точки зору виділяють більш широкий діапазон метрик, які відображають швидкодію веб-додатку. А саме:

First Contentful Paint (FCP) (перше змістовне відображення) є одним із ключових метрик, що оцінює швидкість завантаження веб-сторінки. Цей показник фіксує час, який потрібен для того, щоб будь-який вміст сторінки став видимим для користувача. Він визначається моментом, коли браузер відображає перший текст, зображення, відео або інший контент на екрані [5].

Однією з головних переваг оптимізації FCP є покращення сприйняття користувачем часу завантаження сторінки. Швидкий FCP відображається у тому, наскільки швидко користувач може взаємодіяти з веб-сайтом та отримувати потрібну інформацію. Це особливо важливо для перших відвідувачів, які уникатимуть очікування і отримають позитивне перше враження.

Методи оптимізації FCP включають у себе мінімізацію розміру ресурсів, використання кешування та вибіркове завантаження контенту. Важливо також враховувати оптимізацію зображень, асинхронне завантаження ресурсів та ефективне використання CDN для забезпечення швидкого доставлення контенту.

Метрика Largest Contentful Paint (LCP) є ще однією важливою метрикою, яка вимірює швидкість завантаження веб-сторінки. LCP фіксує час, необхідний для завантаження найбільшого контенту на сторінці, такого як картинки чи текстового блоку. Це важливо, оскільки саме цей елемент зазвичай має ключовий вплив на візуальне сприйняття користувача [5].

Оптимізація LCP визначається можливістю швидкої доставки та відображення найбільшого контенту (змістовного елементу веб сторінки). Це допомагає уникнути ситуацій, коли користувач довго очікує на завантаження важливого елемент сторінки, що може призвести до негативного враження, а у випадку електронної комерції втрати потенційного покупця.

Для поліпшення LCP важливо використовувати ефективні сервіси доставки контенту (CDN) для швидкого завантаження ресурсів, оптимізувати графічний контент та мінімізувати вплив JavaScript на відображення сторінки. Розсудливе використання кешування та оптимізація серверної відповіді також можуть покращити метрику Largest Contentful Paint.

Cumulative Layout Shift (CLS) вимірює стабільність візуального відображення сторінки під час її завантаження. Ця метрика визначає, наскільки суттєві зміни в макеті сторінки відбуваються під час взаємодії з нею. Величина CLS враховує площу та величину всіх зсувів макету, що виникають під час взаємодії з користувачем.

Важливість CLS полягає в тому, щоб уникнути неочікуваних та дратуючих змін у відображенні сторінки, які можуть призвести до негативного враження від користувачів. Особливо це актуально для мобільних пристроїв, де розмір екрану обмежений, і зсуви макету можуть викликати невідповідність та незручності.

Для оптимізації CLS важливо уникати динамічних елементів, які можуть змінювати розміри чи місце розташування на сторінці без попередження. Завантаження резервних розмірів для медіа-контенту та використання CSS для фіксації розмірів блоків можуть сприяти стабільності макету та зменшити вплив на CLS.

Time to First Byte (TTFB) — це метрика, яка вимірює час, який потрібний серверу для надсилання першого байту відповіді на запит користувача. Цей параметр вказує на швидкість відповіді сервера та його ефективність у відправленні даних [5].

Важливість метрики TTFB полягає в тому, що цей час визначає, наскільки швидко користувач отримає перші дані після того, як відправив запит. Чим менший TTFB, тим швидше користувач отримає відповідь від сервера, що сприяє загальній швидкості завантаження сторінки.

Оптимізація TTFB включає в себе ряд заходів. Перш за все, важливо використовувати швидкі та ефективні сервери, які можуть швидко обробляти запити. Використання кешування та компресії даних може допомогти зменшити обсяг даних, що також впливає на TTFB.

Total Blocking Time (TBT) — це метрика, яка вказує на сумарний час блокування на сторінці, коли користувач не може взаємодіяти з контентом через блокуючі завдання JavaScript. TBT враховує тривалість та кількість блокуючих задач, що впливають на інтерактивність сторінки [5].

Ця метрика важлива, оскільки користувачі можуть сприймати сторінку як повільну та неефективну, якщо вони зустрічають блокуючі завдання, що ускладнює їх взаємодію з контентом.

Зменшення Total Blocking Time досягається за допомогою оптимізації та ефективного управління завданнями JavaScript. Важливо уникати виконання довгих та ресурсномістких операцій на основному потоці виконання браузера, а також використовувати асинхронні методи та відкладати виконання завдань, які не впливають на видимий контент сторінки.

Отже, проаналізувавши зміст описаних метрик, можна зробити висновок що їх покращення є критичними для забезпечення високої конверсії в контексті е-комерції, враховуючи це розглянемо технічні методи для їх досягнення.

## 1.3 Обґрунтування вибору технічних засобів для розробки клієнтської частини додатку

Клієнтська частина (front-end) — це та частина веб-додатку, яка відповідає за відображення інформації та взаємодію з користувачем. Основною метою фронтенду є створення інтуїтивного та ефективного інтерфейсу, який дозволяє користувачеві взаємодіяти з веб-додатком. Він відповідає за виведення вмісту веб-сторінки чи додатку на екран та забезпечення користувача зручними засобами взаємодії з цим вмістом.

Для ефективної розробки великих масштабованих систем існує і активно розвивається широка інфраструктура програмних рішень, що полегшує реалізацію прикладних задач.

Програмний фреймворк, або каркас програмного забезпечення, представляє собою готовий комплекс програмних рішень, що включає в себе не лише дизайн та логіку, але й базовий функціонал системи чи підсистеми. Він може охоплювати допоміжні програми, бібліотеки коду, скрипти та інші елементи, спрямовані на спрощення процесу створення та інтеграції різних компонентів у комплексне програмне забезпечення чи програмного продукту. Зазвичай, кінцевий продукт будується на основі єдиного API [6].

Використання каркасних застосунків має ряд значущих переваг, а серед них особливо виділяється стандартизована структура програм. Фреймворки стали вкрай популярними після появи елементів інтерфейсу, які виявили тенденцію до стандартизації структури додатків. Завдяки їм виникла можливість легко створювати інструменти для автоматизованого створення графічних інтерфейсів, оскільки структура внутрішньої реалізації коду програми стала відомою наперед. Для побудови каркаса часто використовують об'єктно-орієнтоване програмування (ООП), наприклад, частини програми можуть успадковуватися від базових класів фреймворку.

В контексті веб-розробки фреймворки часто протиставляються бібліотекам підпрограм, які являють собою збірки програмних функцій для вирішення певних завдань [6]. Подібні бібліотеки різняться за тематикою та складністю реалізації, виділяють утилітарні бібліотеки підпрограм, як наприклад бібліотеки для роботи з датами і часом (date-fns), анімаціями (three.js), HTTP запитами (axios) і т. д, та більш широкі за використанням – бібліотеки для розробки взаємодії користувача з веб-сторінкою (jQuery, pug тощо).

Фреймворк є великою структурою, що надає загальну основу для розробки програм, визначаючи структуру та контролюючи її виконання. Він забезпечує стандартизацію, загальну архітектуру і прискорює розробку, натомість може бути менш гнучким та вимагати більше часу для впровадження в умовах реального продукту.

З іншого боку, бібліотека — це набір функцій чи підпрограм, які можна використовувати для конкретних завдань. З цього слідує головна її перевага, бібліотека дозволяє використовувати лише ті компоненти, які необхідні, забезпечуючи гнучкість використання та легшу інтеграцію, але може вимагати більшої відповідальності від розробника та призводити до менш структурованого коду.

Для узагальнення можна розглянути таку аналогію — фреймворк можна порівняти з промисловим верстатом, що визначає загальну структуру та інфраструктуру для виробництва, включаючи стандарти та процеси. У випадку фреймворку визначається основна архітектура програми та механізми взаємодії компонентів.

В свою чергу, бібліотеку можна розглядати як набір інструментів чи пристроїв, які можна використовувати для виконання конкретних завдань. Кожен інструмент у бібліотеці є окремим функціональним компонентом, доступним для виклику від розробника.

Можна зробити висновок, що фреймворк на відміну від бібліотеки визначає загальний контекст розробки, забезпечуючи структурованість та уніфікацію процесів. З іншого боку, бібліотека надає конкретні функції та інструменти, спрощуючи виконання окремих задач.

Дослідження проведені онлайн ресурсом з технічного профайлінгу Wappalyzer показують, що в області е-комерції використання фреймворків є переважаючим, оскільки вони надають розробникам стабільну та зручну основу для побудови веб-застосунків [7]. Фреймворки ефективно керують архітектурними аспектами, такими як безпека, масштабованість та оптимізація продуктивності. Вони також часто постачаються з готовими модулями та компонентами, що спрощує розробку та розширення функціоналу електронних комерційних платформ. Безумовно всіх цих аспектів можна досягти у випадку використання збірних бібліотек функцій, натомість такий підхід при розробці великих за розміром систем призводить до великої кількості зовнішніх залежностей, деякі бібліотеки можуть припинити оновлюватись, в той же час нові версії можуть містити критичні зміни, які порушують роботу існуючого функціоналу тощо.

Однією з основних переваг використання фреймворку в е-комерції є можливість швидкого розгортання та розвитку системи. Враховуючи високий рівень абстракції та готові рішення, розробники можуть ефективно взаємодіяти з різними компонентами, прискорюючи час від ідеї до впровадження.

Крім того, фреймворки надають стандартні практики та підходи, що допомагають уникнути багатьох типових помилок та забезпечують консистентність в розробці. Це особливо важливо в е-комерції, де системи часто великі та складні, тому однаковий та стандартизований підхід є ключовим для ефективного управління подібними ІС.

Отже, враховуючи потреби електронної комерції в стабільності, ефективності та швидкості розробки, використання фреймворку обрано оптимальним рішенням для успішного впровадження та функціонування електронних комерційних платформ.

Проаналізуємо існуючі популярні веб-фреймворки, для вибору найбільш оптимального варіанту для розробки ІС з врахування описаних вище вимог. Аналіз існуючих веб-фреймворків для використання в електронній комерції вимагає врахування декількох ключових аспектів, що визначають їхню придатність та ефективність.

Одним із ключових критеріїв є архітектура та структурна складова фреймворку. Саме в контексті ІС для е-комерції важливо, щоб фреймворк був модульним та масштабованим, спроможним легко адаптуватися до зростання обсягу даних та користувачів, що є переважаючою характеристикою сучасних онлайн системи. Не останнє місце також займає безпека фреймворку, що визначається його здатністю захищати від різних видів атак, таких як SQL-ін'єкції чи XSS. Ефективна система аутентифікації та авторизації є важливою для конфіденційності користувачів. Крім того, швидкодія та оптимізація продуктивності визначають ефективність веб-застосунків, забезпечуючи швидке обслуговування запитів та відповідей.

Спільнота розробників та якісна документація впливають на доступність та підтримку фреймворку. Активна спільнота сприяє регулярним оновленням та виправленню помилок. Сумісність фреймворку з різними технологіями розширює його можливості, а висока результативність гарантує ефективне використання ресурсів сервера та клієнта.

Враховуючи описаний вище набір вимог, відзначимо декілька ключових фреймворків, які є популярними та широко використовуються в індустрії.

React від Facebook — це один із найпоширеніших та найвпливовіших фреймворків у світі веб-розробки. Визначаючись декларативним підходом до розробки, він спрощує створення складних інтерфейсів шляхом опису потрібного стану та забезпечує ефективне відслідковування змін, автоматично оновлюючи інтерфейс.

Екосистема React включає в себе не лише базовий фреймворк, але й велику кількість сторонніх бібліотек та інструментів, що допомагають вирішувати різні задачі в розробці. Наприклад, Redux використовується для ефективного управління станом додатку, а React Router — для навігації між різними сторінками [8].

Однак, важливо відзначити, що React може вимагати значної кількості налаштувань та залежностей, що може бути надмірним для невеликих та середніх проектів. Розгортання та конфігурація проекту можуть вимагати додаткового часу та уваги. Однак, коли встановлені необхідні компоненти, React надає потужний та гнучкий інструментарій для розробки високоякісних веб-додатків.

Angular від Google — це повноцінний фреймворк, який відрізняється широким функціоналом та повністю інтегрованою системою для розробки веб-додатків. Завдяки своїй структурі та набору вбудованих інструментів, Angular часто вважається ідеальним вибором для великих та складних проектів, де потрібна висока стабільність та розширюваність [9].

Однак, використання Angular може бути більш обтяжливим порівняно з іншими фреймворками. Розробники можуть стикатися з великою кількістю обов'язкового коду та правил, яких необхідно дотримуватися. Це може призвести до збільшення часу розробки та підтримки коду. Саме тому Angular часто називають ідеальним для великих продуктових рішень, більшість сервісів компанії Google використовує Angular в своїх розробках.

Незважаючи на це, Angular забезпечує високий рівень безпеки та продуктивності. Його система модульності, двостороннє зв'язування та розширений набір інструментів роблять його потужним фреймворком для розробки великих корпоративних додатків.

Vue.js — це сучасний веб-фреймворк, який відзначається своєю легкістю та простотою використання. Створений з урахуванням поступового підходу, Vue.js надає розробникам зручний інструмент для швидкої і ефективної розробки інтерактивних інтерфейсів [10].

Фреймворк пропонує чистий та інтуїтивний синтаксис, що полегшує розуміння та обслуговування коду. Слід відзначити добре документоване API, що швидкому освоєнню розробниками різного досвіду.

Vue.js дозволяє поетапно впроваджувати його в існуючі проекти, забезпечуючи гнучкість інтеграції. Фреймворк відомий своєю ефективністю у керуванні станом додатку та легкою взаємодією з іншими бібліотеками чи компонентами.

Завдяки своїй легкості та простоті, Vue.js стає привабливим вибором для розробників, які цінують ефективність у роботі та потребують зручного інструмента для реалізації інтерактивних веб-додатків.

Враховуючи гнучкість, легкість інтеграції та підтримки в даній роботі розглянуто розробку додатку з використанням фреймворку Vue.js.

Сучасні фреймворки, зокрема Vue.js, відіграють ключову роль у покращенні продуктивності та ефективності веб-застосунків через кілька ключових аспектів, які розглянуто далі.

Vue.js сприяє розділенню великих і складних інтерфейсів на невеликі компоненти. Це полегшує розробку та управління кодом, оскільки кожен компонент може мати свій стан, методи та стилі. Такий підхід робить код більш структурованим та піддається легшому тестуванню, що важливо для підтримки та масштабування системи електронної комерції, де інтерфейс доцільно розділяти на структурні елементи, такі як картка товару, фільтри, меню вибору категорії, вітрина товарів тощо. Подібний підхід до архітектури робить можливим повторне використання елементів додатку, що знижає відсоток дублювання коду, яке суперечить принципу розробки ПЗ, відомим як «Don’t repeat yourself» (з англ. «не повторюйся).

Характерною ознакою сучасних веб-фреймворків є «реактивність». Реактивність вказує на здатність фреймворку автоматично реагувати на зміни у стані даних і оновлювати відображення без прямого втручання розробника. В Vue.js це реалізовано через концепцію «реактивних об'єктів». Якщо дані, які використовуються в компоненті, змінюються, то змінюється і відображення, пов'язане з цими даними. Це значно спрощує управління станом та взаємодію компонентів. В рамках обраної тематики, можна навести простий приклад — при додаванні товару у корзину, автоматично перераховується сумарна вартість замовлення, кількість товарів тощо.

На ряду з реактивністю сучасним рішенням для побудови веб-додатків притаманна також здатність до зворотної взаємодії, яка означає можливість обробки подій які користувач ініціює через інтерфейс та синхронізація даних. В Vue.js це реалізовано через обробники подій, методи та інші механізми. Коли користувач взаємодіє з елементами на сторінці, Vue.js реагує на ці дії, викликаючи відповідні функції або оновлюючи дані. В електронній комерції ці аспекти є ключовими, оскільки дозволяють створювати динамічні та інтерактивні веб-застосунки, які забезпечують зручний та ефективний досвід для користувачів. Реактивність дозволяє автоматизувати оновлення та відображення даних, а зворотна взаємодія — створювати різноманітні функціональні можливості, такі як додавання товарів у кошик, фільтрація товарів та багато іншого.

Vue.js володіє легкістю та швидкістю завдяки вбудованій системі віртуального DOM (Document Object Model). Він ефективно оновлює лише ті частини сторінки, які зазнали змін, замість повного перезавантаження веб-сторінки [10]. Це сприяє високій продуктивності та забезпечує високу швидкість роботи.

Врешті решт, розглянуте програмне рішення активно розвивається, а значить має велику та активну спільноту розробників, що дозволяє швидко знаходити рішення, обмінюватися досвідом та використовувати багато готових бібліотек та розширень для вдосконалення функціоналу. Наявність спільноти та постійною підтримки програмного засобу, означає можливість тривалої підтримки, оновлення, виправлення недоліків та вразливостей у його роботі в майбутньому.

Крім того, використання обраних технологій надає можливість до впровадження технологій серверного рендерингу SSR, таких як Nuxt.js, що є важливим елементом для оптимізації продуктивності. SSR дозволяє виконувати генерацію HTML-коду на сервері, зменшуючи обсяг переданих даних та час завантаження сторінки [11]. Це особливо корисно для електронної комерції, де швидкість завантаження важлива для збільшення конверсії та задоволення основної потреби користувача – швидкого доступу до бажаної інформації.

Розглянемо архітектурні особливості побудови веб-додатку такої складності. Оскільки, в якості основного фреймворку для побудови клієнтської частини додатку обрано Vue.js то в якості шаблону проектування архітектури обрано Model-View-ViewModel (MVVM).

Шаблон проектування MVVM (абревіатура від модель, вигляд, модель-вигляду) розроблений на основі іншого шаблону Presentation Model, широко використовується при побудові додатків на базі сучасних фреймворків (Vue, Angular). Шаблон ділиться на три частини:

* модель (Model) – являє собою базові дані, які продиктовані бізнес-вимогами;
* вигляд (View) – безпосередньо графічний інтерфейс програми (веб-сторінка, елементи сторінки, кнопки, форми тощо);
* модель представлення (View-Model) – являє собою абстракцію Вигляду і виконує роль обгортки Моделі для зв’язування даних. Використовується для забезпечення взаємозв’язку між моделлю та відображенням, відслідковує зміни у користувацькому вводі.

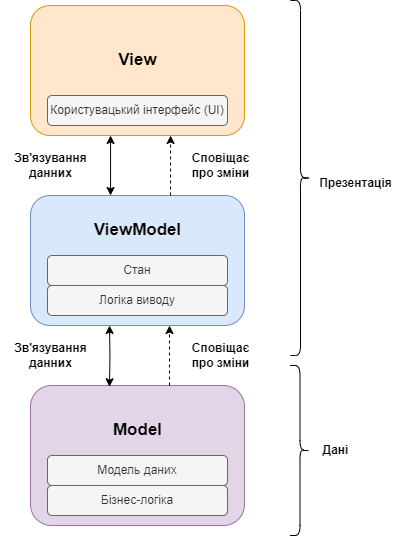


Рисунок 1.4 Схема взаємодії компонентів шаблону MVVM

Модель вигляду (View-Model), перекладається як «Модель Вигляду», або «модель представлення» функціонує як інтерфейс між Моделлю та Відображенням. З одного боку, вона втілює Модель у форматі, придатному для відображення. З іншого боку, вона включає команди, які Відображення може використовувати для впливу на Модель. Іншими словами, View-Model є посередником між логікою даних (Модель) і представленням цих даних (Відображенням). Це дозволяє відслідковувати та реагувати на зміни даних, а також забезпечує Відображенню можливість взаємодії з Моделлю через визначені команди [12].

View-Model відіграє ключову роль у здійсненні зв'язку між компонентами системи. Ця абстракція відділяє представлення від деталей роботи з даними, спрощуючи відображення і роблячи його менш залежним від внутрішньої реалізації моделі даних.

Одна з основних функцій моделі представлення – це відстеження змін у вихідних даних, які можуть бути внесені користувачем (взаємодія з інтерфейсом). Це дозволяє компоненту View-Model реагувати на зміни, визначати логіку обробки даних та повідомляти компоненту відповідальному за вивід (View) про будь-які зміни, які повинні бути відображені в інтерфейсі. View-Model є ефективним інструментом для обробки комунікації між фронтендом і бекендом. Вона дозволяє створити рівень абстракції, який спрощує розробку і підтримку програмного продукту, розділяючи відповідальності за обробку даних і їх відображення.

Причина виникнення і зростання популярності MVVM полягає в необхідності у певному відокремленні моделі та представлення для незалежного їхнього розвитку. Такий підхід може мати ряд переваг у командній розробці ПЗ. Наприклад, прикладний розробник може працювати над логікою обробки даних, тоді як дизайнер може бути залучений до розробки користувацького інтерфейсу.

Шаблон MVVM нерідко ставлять у противагу іншому розповсюдженому підходу до архітектури ПЗ відомому як MVC (Model-View-Controller). Розглянемо ключові відмінності цих шаблонів проектування.

MVC визначає три основні компоненти: Модель (Model), Вигляд (View) та Контролер (Controller). Модель представляє дані та бізнес-логіку, Вигляд відповідає за відображення даних користувачу, а Контролер виконує реакції на взаємодію користувача та зміни в моделі [13].

Основна відмінність між цими підходами полягає в ролі Моделі представлення. У MVC Контролер відповідає за обробку взаємодії, тоді як у MVVM цю роль виконує Модель представлення. MVVM дозволяє відокремити логіку відображення від бізнес-логіки, що спрощує тестування та розширення програмного продукту.

Отже, хоча обидва архітектурні підходи пропонують відокремлення компонентів для поліпшення структурованості та розширюваності, MVVM пропонує більш сучасний підхід, де Модель представлення виступає в якості посередника між Моделлю та Виглядом, дозволяючи більш ефективно вирішувати завдання, пов'язані із динамічними змінами в інтерфейсі користувача.

Розглянемо особливості використання MVVM підходу для Vue.js додатків. Для реалізації комплексних додатків на базі Vue.js розглянемо використання бібліотеки Vuex.

Vuex — це шаблон управління станом та бібліотека для додатків на Vue.js. Вона виступає як централізоване сховище даних для всіх компонентів додатку з правилами, які гарантують, що стан може бути змінений лише передбачуваним способом [14]. Подібний підхід до роботи з даними у взаємодії з фреймворком Vue досить зручно використовувати для побудови додатків за принципами MVVM розглянутими вище.

Основними складовими Vuex є:

1. Стан (State): Представляє собою централізований об'єкт, що містить стан всього додатку. Зміни в стані відбуваються тільки через визначені мутації.
2. Мутації (Mutations): Функції, які змінюють стан. Вони використовуються для виконання синхронних та передбачуваних змін у стані.
3. Дії (Actions): Асинхронні функції, які використовуються для виклику мутацій або виконання складних операцій, таких як взаємодія з сервером.
4. Геттери (Getters): Функції, які дозволяють отримувати та обчислювати значення зі стану. Вони корисні для відслідковування та обчислення деяких властивостей стану.

Vuex забезпечує односторонній потік даних, що означає, що дані рухаються в одному напрямку — від стану через мутації до компонентів. Це спрощує влагодження та управління станом додатку, особливо в разі складних та великих проектів.

Взаємодія між сутностями Vuex визначає консистентний та ефективний обмін даними у клієнтському додатку. Компонентів Користувацького Інтерфейсу, отримують доступ до стану додатка через геттер-функцій, що дозволяє їм отримувати та відображати потрібні дані, не маючи безпосереднього доступу до стану додатку, компонент отримує тільки ті дані які необхідні йому для презентації.

Коли користувач здійснює взаємодію з додатком, це викликає дію (action). Дії викликають асинхронні операції та взаємодію з зовнішніми ресурсами. Ці дії мають властивість викликати Мутації, які, в свою чергу, мають на меті змінювати стан додатка.

Мутації грають ключову роль у синхронізації та внесенні передбачуваних змін у стан. Вони виступають як проміжний шар, перетворюючи дані від дій та вносячи їх до стану додатка.

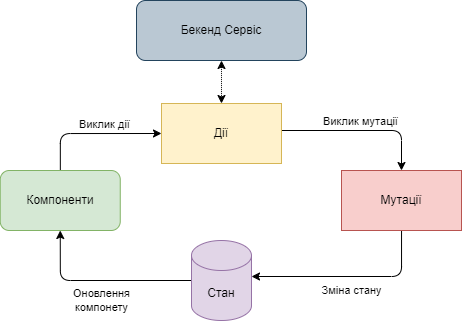


Рисунок. 1.5 Цикл роботи Vuex

В контексті взаємодії між сутностями Vuex, важливо зауважити, що Дії можуть включати в себе взаємодію із зовнішнім сервером через API. Коли користувач викликає дію, таку як завантаження даних чи оновлення вже отриманих даних, вона може призводити до асинхронних операцій із сервером. В той же час, мутації призначені для синхронних змін у стані додатка. Оскільки вони безпосередньо модифікують стан, вони не можуть бути асинхронними. Такий розділ функціональності між діями та мутаціями обумовлений прагматичним підходом до керування станом, який передбачає що змінювати внутрішній стан програми можуть лише синхронні операції [14].

Дії використовуються для виконання асинхронних завдань, таких як взаємодія із зовнішніми ресурсами, і вони можуть викликати мутації після завершення операцій. Це дозволяє додатку ефективно обробляти асинхронні запити та забезпечує чітку структуру управління станом програми.

Стан додатку, як централізоване сховище, є місцем, де зберігаються та оновлюються всі дані, це єдине джерело даних для усіх компонентів ІС. Це забезпечує консистентність та передбачуваність усього додатка.

Таким чином, цикл взаємодії розпочинається з користувацького вводу, переходить до виклику дій, активує мутації, які змінюють стан, і, у кінцевому результаті, оновлюють представлення компонентів інтерфейсу. Такий підхід забезпечує ясну структуру та керований потік даних у додатку.

Завдяки чіткому поділу на дані та операції з даними, додаток з використанням менеджеру vuex може бути використаний в рамках шаблону проектування архітектури MVVM, розглянемо особливості такого підходу.

В даному контексті, стан Vuex виступає як модель додатку. Всі дані, якими керує менеджер Vuex, представляють стан додатку, а їх зміни здійснюються через мутації та дії.

Компоненти Vue, що відображають інтерфейс користувача, виступають у ролі представлення (View). Вони взаємодіють зі станом через геттер-функції, які обчислюють необхідні дані та властивості, які відображають дані із Vuex. Слід зазначити, що компоненти отримують лише необхідні для представлення даних. Наприклад, компонент «шапки» веб-додатку е-комерції, який містить посилання на кошик та сумарну вартість обраних користувачем товарі, отримує через геттер-функцію тільки обчислену сумарну вартість товарів і не має доступу до детального списку всіх обраних користувачем товарів.

Модель представлення в MVVM відповідає за конвертацію даних між моделлю та представленням, а також за обробку логіки користувача. У випадку Vuex, цю роль виконує логіка дій та мутацій, які взаємодіють із станом додатку.

Зв'язок між моделлю, представленням та моделлю представлення реалізується через систему зв'язку та обміну повідомленнями, яку забезпечує Vuex. Зміни в стані автоматично відображаються в представленні завдяки системі реактивності Vue.

Отже, інтеграція менеджеру стану Vuex у додаток на базі фреймворку Vue.js дозволяє реалізувати ефективний підхід MVVM, забезпечуючи чітку структуру та підтримуючи реактивність для автоматичного оновлення представлення при зміні стану.

## **1.4 Обґрунтування вибору технічних засобів для розробки серверної частини додатку**

Ключову роль при розробці сучасних веб-рішень для електронної комерції, відіграє використання надійних інструментів для побудови серверної частини додатку (бекенду). Бекенд містить у собі основну бізнес-логіку продукту, у випадку систем е-комерції – збереження, обробка, передача даних про товари, відповідь на запити веб-клієнтів, обробка замовлень, внутрішня взаємодія сервісів тощо. Головні принципи якими керувались при виборі засобів розробки серверної частини в даній роботі – надійність та можливість масштабування. Оскільки обрана предметна область передбачає велику кількість активних підключень до ІС то до вимог також можна додати можливість роботи при значних навантаженнях.

У процесі вибору технічних засобів для розробки серверної частини додатку надзвичайно важливо провести всебічний огляд наявних варіантів. Розглянуто популярні платформи та мови програмування, які активно використовуються для побудови серверної

Node.js – платформа на базі мови програмування JavaScript, відома своєю асинхронною та неблокуючою моделлю вводу-виводу, ідеально відповідає сучасній потребі високого паралелізму та швидкості реагування в режимі реального часу. Завдяки цій архітектурі він особливо добре підходить для додатків із динамічним вмістом або тих, що потребують ефективної обробки одночасних операцій [15].

В якості фреймворку розглянуто фреймворк Nest.js. Інтеграція Nest.js у стек технологій обумовлена його здатністю забезпечувати структуровану модульну архітектуру в програми Node.js. Черпаючи натхнення в Angular, Nest.js представляє такі поняття, як модулі, контролери та служби, покращуючи організацію коду та зручність обслуговування. Цей структурований підхід стає особливо вигідним, оскільки складність програми масштабується, забезпечуючи основу для довгострокової стійкості.

Крім того, Nest.js вимагає використання типізованої мови програмування TypeScript. Прийняття TypeScript як основної мови для серверної розробки ще більше сприяє успіху проекту, оскільки TypeScript представляє статичну типізацію, покращуючи читабельність коду та полегшуючи раннє виявлення потенційних помилок під час розробки. Цей аспект особливо важливий для побудови зручного та надійного коду, особливо у великих проектах або середовищах спільної розробки.

Екосистема Node Package Manager (NPM) відіграє ключову роль у вибраному стеку, надаючи доступ до різноманітного сховища пакетів з відкритим кодом. Ця екосистема дозволяє розробникам використовувати наявні рішення для загальних функціональних можливостей, сприяючи модульному й ефективному процесу розробки. Доступність широкого спектру пакетів через NPM сприяє гнучкості проекту та зменшує потребу в переосмисленні рішень для стандартних вимог [16].

Підсумовуючи, стратегічний вибір Node.js і Nest.js випливає з їх спільних сильних сторін у асинхронній обробці, уніфікованій екосистемі TypeScript, структурованій архітектурі, TypeScript для вдосконаленої розробки та ефективності екосистеми NPM. Цей ретельно продуманий пакет технологій готовий задовольнити вимоги проекту, поєднуючи швидкість реагування, зручність обслуговування та ефективний процес розробки.

Розглянемо інші розповсюджені рішення.

Java – мова програмування загального призначення, є непохитним лідером у додатках корпоративного рівня та пропонує зрілу та надійну екосистему для розробки на стороні сервера.

Відома своєю портативністю, масштабованістю та великими бібліотеками, Java є надійним вибором для створення великомасштабних, критично важливих програм. Віртуальна машина Java (JVM) забезпечує незалежність від платформи, дозволяючи програмам на Java безперешкодно працювати в різних операційних системах.

Крім того, об’єктно-орієнтована парадигма Java та сильна типізація сприяють організації коду та зручності обслуговування, сприяючи створенню модульних і розширюваних систем. Наявність потужних фреймворків, таких як Spring і Jakarta EE, ще більше розширює можливості Java, надаючи комплексні рішення для таких аспектів, як впровадження залежностей, керування транзакціями та корпоративна інтеграція [17].

Однак важливо визнати, що Java, хоч і є безумовно надійним корпоративним рішенням, Java історично асоціюється з певним ступенем багатослівності та надмірності в коді, що потенційно сповільнювало цикли розробки. Процеси ініціалізації та конфігурації в проектах Java можуть бути більш складними порівняно з відносно легкими та швидкими підходами до розробки, які пропонують Node.js і Nest.js.

Мова програмування Python, завдяки своїм різноманітним фреймворкам, заслуговує ретельного огляду в контексті вибору її як основну для розробки серверного додатку.

Високорівнева мова програмування Python відома своєю зрозумілістю, універсальністю та величезною екосистемою бібліотек, що робить її популярним вибором для різних областей застосування, включаючи веб-розробку. Популярні фреймворки, такі як Django та Flask, ілюструють здатність мови сприяти швидкому розвитку, дотримуючись при цьому кращих практик в інженерії програмного забезпечення.

Django, високорівневий веб-фреймворк, пропонує широкий набір функцій «з коробки», включаючи систему ORM, аутентифікацію та адміністративний інтерфейс. Цей фреймворк є досить ефективним у сценаріях, де віддається перевага конвенціям над конфігурацією, спрощуючи процес розробки та підтримуючи консистентність коду.

З іншого боку, Flask, як мікро-фреймворк, надає розробникам більше гнучкості, дозволяючи використовувати більш модульний та настроюваний підхід. Flask підходить для проектів, які вимагають легкої та мінімалістичної основи, дозволяючи розробникам обирати та інтегрувати компоненти за потребою [18].

Підсумовуючи, підхід Python із такими фреймворками, як Django та Flask, дає численні переваги щодо читабельності, універсальності та підтримки коду. Однак потенційні недоліки, такі як питання продуктивності та швидкодії, особливо в випадку використання фреймворку Django, можуть бути досить критичними для побудови веб-додатків з зазначеними характеристиками.

Враховуючи описані вище характеристики, в якості основною платформою для побудови серверної частини додатку обрано платформу Node.js та фреймворк Nest.js. Ключовим аспектом у виборі є високий рівень ефективності Node.js у керуванні асинхронними операціями та потенційною масштабованістю такої системи. Крім того, використання Nest.js, побудованому на принципах модульності та організаційній цілісності, забезпечує розробникам універсальну платформу для створення високопродуктивних веб-додатків, що відповідають сучасним стандартам розробки.

Оскільки, е-комерція передбачає роботу з великими обсягами даних, необхідно обґрунтувати вибір технічних засобів для розробки та підтримки бази даних такої ІС.

В рамках обраної теми більш вмотивованим є використання реляційних баз даних, оскільки вони дотримуються структурованого табличного формату, реалізують чітко визначення відносини між сутностями даних. Реляційні БД застосовуються в сценаріях, де необхідна цілісність, таких як фінансові системи чи підприємницькі додатки. Порівняємо основні технології для побудови реляційних баз даних [19].

У контексті вибору систем управління базами даних (СУБД) для розробки веб-додатків, можна виділити дві передові технології MySQL та PostgreSQL кожна з яких має свої унікальні особливості та переваги.

MySQL широко використовується та виділяється простотою в розробці та підтримці. Ця СУБД широко використовується в різноманітних проектах завдяки швидкості та надійності операцій. Однак, в порівнянні з PostgreSQL, MySQL може виявити меншу підтримку для складних запитань та операцій, а також має деякі обмеження в реалізації стандартів SQL.

Натомість, СУБД PostgreSQL вирізняється своєю гнучкістю та розширюваністю. Вона надає користувачам широкий спектр функцій, які включають складні запити, операції з JSON та підтримку великої кількості типів даних. PostgreSQL володіє високою ступенем ACID-сумісності та можливістю розгортання на різних операційних системах. Її велика активна спільнота та активний розвиток дозволяють швидко виправляти помилки та вдосконалювати функціональність [20].

Вибір PostgreSQL в рамках дипломної роботи обумовлений кількома факторами. По-перше, важливо враховувати рівень підтримки для розширених функцій та стандартів SQL, що особливо корисно для проектів, які передбачають складні запити та операції з даними, що є розповсюдженою вимогою для великих ІС електронної комерції, які можуть містити велику кількість сутностей з відносинами різних типів. Окрім того, гнучкість PostgreSQL у роботі з різними типами даних, включаючи JSON, сприяє розвитку веб-додатків, які вимагають різноманітності у структурі даних. Не останнім фактором є також активна спільнота та розвиток PostgreSQL, що гарантує швидке вирішення можливих проблем та невпинне покращення функціоналу системи.

Такий об'єктивний вибір СУБД, як PostgreSQL, відповідає вимогам сучасної веб-розробки та забезпечує стабільну, гнучку та ефективну основу для проекту даного характеру

## 1.5 Висновки до розділу 1

В розділі проведено аналітичний огляд предметної області – електронної комерції, розглянуто основні складові цієї галузі, задачі та сучасні тренди. Визначено основні метрики які впливають на успішність таких систем та сприяють збільшенню показників конверсії.

В контексті е-комерції та онлайн торгівлі, розглянуто технічні засоби та інструменти для побудови надійних ІС для е-комерції, для реалізації веб-додатку обрано мову програмування JavaScript, платформу Node.js для реалізації серверу та Vue.js для реалізації клієнтської частини веб-додатку. В якості СУБД обрано PostgreSQL.

# РОЗДІЛ 2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ Е-КОМЕРЦІЇ ТА СИСТЕМИ РЕКОМЕНДАЦІЇ КОНТЕНТУ

## 2.1 Проектування структури веб-додатку е-комерції

В практичній частині дипломної роботи реалізовано ІС для е-комерції, крім того здійснена розробка та інтеграція рекомендаційної системи. Архітектура інтернет-магазину побудована із використанням трирівневої моделі. Клієнтська частина реалізована з використанням фреймворку Nuxt.js, забезпечуючи інтерфейс користувача за допомогою Vue.js. Веб-сервер Nginx використовується для обробки статичних ресурсів, балансування навантаження та забезпечення безпеки.

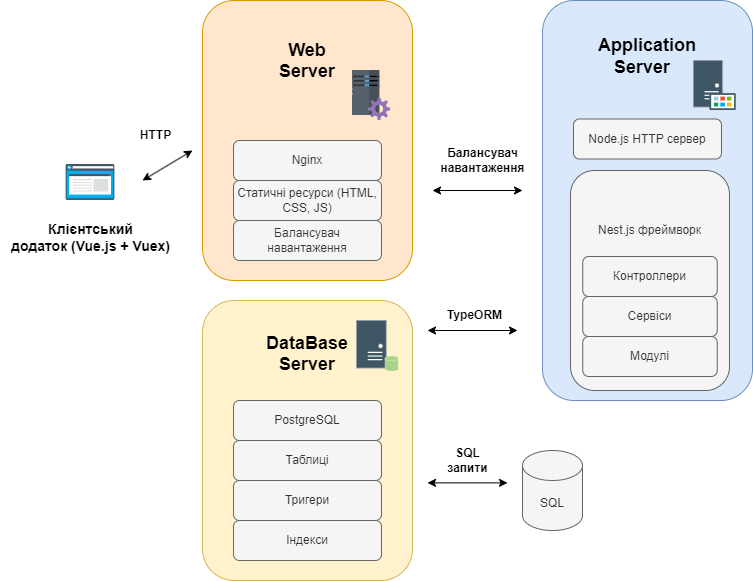
На рівні веб-сервера Nginx виконує декілька ключових функцій. Він обробляє статичні ресурси, такі як HTML, CSS та зображення, покращуючи продуктивність та знижуючи навантаження на сервер додатків. Nginx також служить як балансувач навантаження, розподіляючи вхідні запити між кількома екземплярами сервера додатків. Це підвищує стійкість до відмов, забезпечує рівномірний розподіл навантаження і покращує продуктивність системи.

Механізм балансування навантаження особливо корисний у сценаріях, де збільшується кількість користувачів або коли потрібно забезпечити високу доступність програми. Він дозволяє ефективно використовувати ресурси серверів, а також надає рівень ізоляції та безпеки, що вкрай важливе для сфери е-комерції.

Опис конфігурації Nginx включає налаштування віртуальних хостів, правил обробки запитів, а також налаштування безпеки. Додатково на цьому рівні можна розглянути використання SSL для забезпечення шифрування даних та забезпечення безпечної передачі інформації між клієнтом та сервером.

На рівні програми використовується Node.js у зв'язці з Nest.js, що забезпечує ефективну обробку запитів, валідацію даних, реалізацію бізнес-логіки та взаємодію з базою даних PostgreSQL. Поділ системи на три рівні забезпечує чітке розмежування обов'язків, що спрощує розробку, масштабування та підтримку програми.

Архітектурні рішення також включають заходи безпеки на кожному рівні, включаючи використання HTTPS, перевірку даних та обробку помилок. Для забезпечення ефективності та чуйності системи введено стратегії масштабування, а Nginx налаштований для оптимальної обробки запитів та забезпечення високого рівня продуктивності. На Рис. 2.1 зображена схема трирівневої архітектури для реалізації ІС.

  
Рисунок. 2.1 Трьохрівнева клієнт-серверна архітектура для реалізації ІС

Використання трьохрівневої клієнт-серверної архітектури дає широкі можливості для подальшої інтеграції інших сервісів, оскільки, кожен рівень може бути реалізований та оновлюватися незалежно.

## 2.2 Огляд систем рекомендації контенту

Однією з ключових ознак сучасних систем е-комерції є використання систем рекомендації контенту. Це системи, які ґрунтуються на використанні алгоритмів машинного навчання та аналізу великої кількості даних, виступають як не тільки ефективний інструмент для поліпшення клієнтського досвіду, але являються ключовим чинником збільшення конверсії та прибутковості електронних платформ [21].

Рекомендаційні системи базуються на використанні алгоритмів машинного навчання та аналізу великих даних для надання користувачам рекомендацій, які відповідають їхнім уподобанням та інтересам. Саме у сфері електронної комерції такі системи вкрай розповсюджені, їх можна розглядати як інтелектуальні алгоритмічні механізми, спрямовані на пропозицію персоналізованих рекомендацій користувачам щодо товарів та послуг.

Досвід показує що адаптивні рекомендаційні системи – важливий інструмент для вдосконалення клієнтського досвіду та збільшення конверсії. Вони базуються на аналізі покупок, переглядів, а також демографічних та соціальних даних користувачів для надання індивідуально підібраних рекомендацій, що відповідають конкретним потребам та уподобанням.

Прикладом успішного впровадження рекомендаційної системи може служити діяльність платформ електронної комерції, таких як Amazon та Netflix, Spotify. Ці компанії вдаються до різноманітних методів, включаючи аналіз історії покупок, перегляду та врахування відгуків, тощо. Це дозволяє створювати досить точні та індивідуалізовані рекомендації, спрямовані на конкретні потреби та уподобання кожного користувача.

Варто зазначити, що з точки зору бізнесу, ефективність цих систем полягає не лише в поліпшенні користувацького досвіду, а й у можливості адаптації до змінних умов ринку. Надання персоналізованих пропозицій для кожного клієнта стає ключовим фактором успіху, особливо в умовах конкурентного середовища. Крім того, використання подібних систем знижає витрати на маркетинг, проведення додаткових досліджень аудиторії, оскільки кожен користувач отримує власну персоналізовану вибірку яка забезпечую найвищі шанси здійснення конкретним користувачем покупки.

Розглянемо реалізацію адаптивних систем рекомендації на прикладі гіганту онлайн торгівлі – Amazon. Це одна з перших та найуспішніших платформ е-комерції яка активно залучає інструменти машинного навчання та штучного інтелекту для збільшення обсягу продажів. Досліджуючи успіх компанії аналітики зазначають, що понад 35% продажів компанія здійснює за рахунок вдалих рекомендацій товарів своїм користувачам. [21]. На сьогоднішній момент Amazon використовує різноманітні техніки для побудови надійних та адаптивних рекомендаційних моделей, такі як метод колаборативного фільтрування, які дозволяють формувати великі за обсягом масиви даних що дає змогу створювати високоякісні системи рекомендацій в реальному часі.

На головній сторінці користувачам виводяться персоналізовані рекомендації, враховуючи їхню попередню історію покупок та переглядів. Такий підхід допомагає створити зручний простір, де користувачі можуть швидко знайти товари, які їх можуть зацікавити (Рис. 2.2).

  
Рисунок 2.2 Приклад вітрини рекомендованих товарів на сайті Amazon

Цей процес сприяє збільшенню популярності товарів серед подібних груп користувачів, що підвищує ймовірність того, що рекомендації будуть відповідати індивідуальним уподобанням кожного користувача.

Варто зазначити що користувачу не обов’язково здійснювати реєстрацію на платформі для того щоб бачити рекомендовані товари, оскільки Amazon використовує інформацію про сесію в браузері користувача та файли cookie, то кожен користувач ІС в не залежності від факту авторизації може мати доступ до вибірки персоналізованого контенту, крім того його пошуки та взаємодії з системою впливають на рекомендації для інших користувачів зі схожими уподобаннями.

Такий підхід до обробки даних сприяє стрімкому зростанню об'ємів інформації для аналізу та адаптації системи до змінних умов і в свою чергу мінімізує ризик так званого «холодного старту», який виникає коли в системі недостатньо даних для аналізу в результаті може призвести до неперсоналізованих рекомендацій або відсутності пропозицій взагалі.

Важливим елементом системи рекомендацій є також товарно-орієнтований підхід (content-based), що дозволяє забезпечити персоналізовані рекомендації на основі характеристик конкретних товарів. Amazon використовує деталі товарів, такі як категорії, бренди, технічні характеристики та описи, для створення профілю кожного товару. Після цього система аналізує взаємодію користувачів з подібними товарами, враховуючи їхні покупки та перегляди.

Цей підхід дозволяє Amazon рекомендувати товари, які мають схожі характеристики чи призначення, що підвищує ймовірність того, що користувачі знайдуть інші продукти, які відповідають їхнім потребам. Наприклад, якщо користувач переглядає сторінку з клавіатурами то на цій же сторінці платформа рекомендує схожі товари, враховуючи бренд, цінову категорію та характеристики, на Рис. 2.3 подано приклад такої рекомендації.

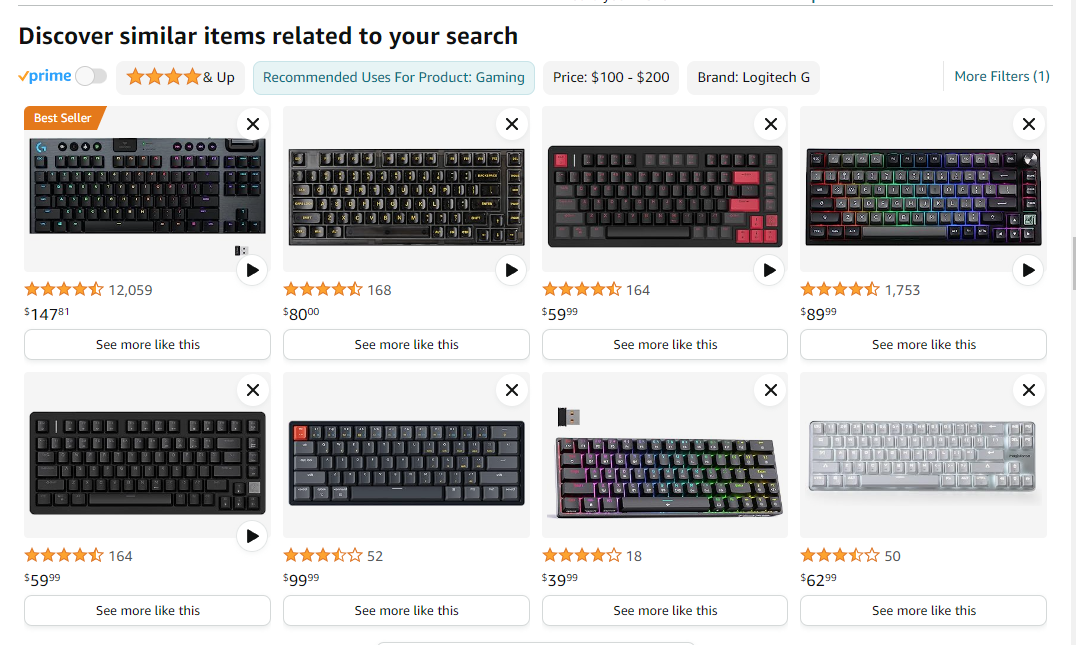
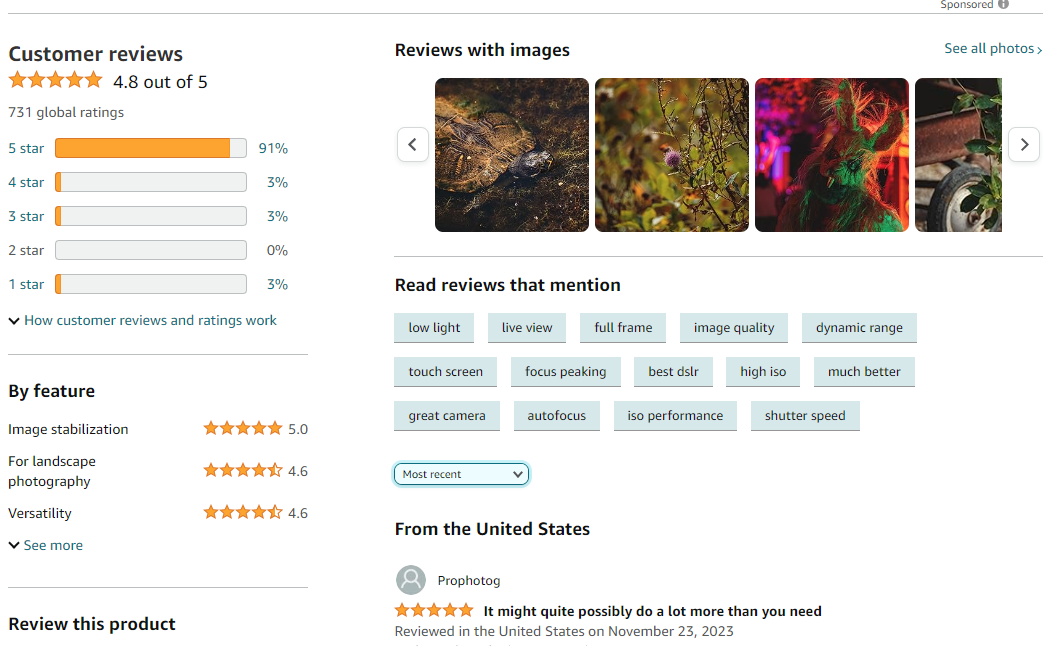


Рисунок 2.3 Вітрина рекомендованих схожих товарів

У такому товарно-орієнтованому підході ключовим елементом є використання алгоритмів, які можуть автоматично виділяти та аналізувати характеристики товарів. Окрім цього, розширення асортименту та оновлення бази даних дозволяють системі стежити за новими трендами та швидко адаптуватися до змін у виборі товарів та вимогах користувачів.

Велика роль також відводиться системам рейтингу та оцінок для кожного товару. У своїй основі система рейтингів дозволяє користувачам висловлювати свої погляди на якість та задоволення від покупки. Amazon використовує ці дані для покращення точності рекомендацій та забезпечення користувачів більш об'єктивною інформацією.

Позитивні оцінки та рецензії визначають популярність та довіру до конкретного товару. Amazon враховує не лише загальну оцінку, але й конкретні коментарі, в яких користувачі можуть детально описати свій досвід з використання товару. Ця інформація допомагає уточнити контекст рейтингу та надає додаткові відомості для алгоритмів рекомендацій.

  
Рисунок 2.4 Панель рейтингу та відгуків на платформі Amazon

Негативні відгуки також мають важливий вплив. Їхній аналіз дозволяє ідентифікувати слабкі сторони товару та вживати заходів для покращення якості. Водночас, система враховує індивідуальні вподобання користувачів, дозволяючи пропонувати товари з високими оцінками та позитивними відгуками, які відповідають конкретним вподобанням та потребам покупця. Крім того, Amazon активно використовує технології машинного навчання для аналізу звичок та взаємодій користувачів. Алгоритми враховують не лише конкретні товари, на які клієнти часто переходять чи здійснюють покупки, але й часові параметри та частоту взаємодій. Наприклад, якщо певний товар стає особливо популярним у певний час року (наприклад, сезонні товари), система може пропонувати його активніше під час цього періоду для інших користувачів зі схожими інтересами.

Однією з важливих характеристик алгоритмів Amazon є їх здатність адаптуватися до змін в споживчому попиті. Наприклад, якщо користувач раніше купував товари для новонароджених, але з часом ці інтереси змінилися на товари для дітей старшого віку, система швидко адаптується до цих змін і пропонує відповідні рекомендації.

Крім того, платформа Amazon використовує інтеграції з соціальними мережами та іншими платформами для збільшення точності рекомендацій. Якщо користувач здійснив вхід на свій обліковий запис через соціальну мережу, Amazon може аналізувати його або її соціальний граф та інтереси на основі активності у цих мережах. Це дозволяє побудувати більш глибокий контекст щодо вподобань та ставлень користувача, надаючи додаткові дані для персоналізації рекомендацій.

Важливо відзначити, що Amazon враховує не лише покупки, але й перегляди товарів. Це дозволяє враховувати можливі інтереси користувачів, які, наприклад, додали товар у кошик, але ще не здійснили покупку. Такий глибокий аналіз взаємодій користувачів дозволяє побудувати більш точний і комплексний профіль користувача для забезпечення персоналізованих та відповідних рекомендацій.

Система також враховує фактори, пов'язані з лояльністю клієнта. Клієнти, які регулярно обирають товари в певній категорії або від конкретних виробників, можуть отримувати персоналізовані рекомендації та пропозиції, спеціально адаптовані до їхніх звичок покупок. Це стимулює лояльність клієнтів та збільшує їхню задоволеність від використання платформи.

Усі ці функції рекомендаційної системи Amazon спрямовані на досягнення головної мети - забезпечення персоналізованих рекомендацій, які відповідають потребам та уподобанням користувачів, забезпечуючи високий рівень задоволеності та зручності під час взаємодії з платформою.

Основні методи рекомендаційних систем охоплюють колаборативний та контент-базований підходи, глибоке навчання та гібридні методи. Ці підходи сприяють оптимізації пропозицій та підвищенню ефективності систем електронної комерції.

Виділяють персоналізовані та не персоналізовані підходи до формування рекомендацій. В випадку не персоналізованого підходу рекомендовані товари та послуги формуються на основі загальних тенденцій, наприклад рекомендація найпопулярніших товарів за категорією, рекомендація акційних товарів тощо. Тобто такі рекомендації мають загальний характер і пропонують ідентичну вибірку товарів для всіх користувачів системи. В свою чергу персоналізовані рекомендації враховують вподобання конкретного користувача і формують унікальну пропозицію для кожного клієнта окремо.

Відокремлюють також гібридні рекомендаційні системи для яких теж характерним є використання алгоритмів машинного навчання, але для більш широкої групи користувачів, наприклад рекомендовані товари можуть відрізнятися в залежності від приналежності клієнта до певної групи – за статтю, за віком, за географічним місцезнаходженням тощо.

Метод розглянутий в межах даної роботи використовується для підбору товарів або послуг, що можуть зацікавити користувачів зі спільними інтересами – це метод колаборативного фільтрування. Цей метод базується на аналізі взаємодії між користувачами та їх спільних виборах або переглядах.

Витоки цього методу можна відслідкувати в роботах проведених в галузі систем рекомендацій у 1990-2000 роках. У ранніх варіантах колаборативного фільтрування використовувались прості алгоритми, такі як метод найближчого сусіда, але з розвитком обчислювальних та алгоритмічних можливостей метод став більш точним та ефективним. На сьогоднішній момент потужність обчислювальної техніки стрімко росте, що дає можливість впроваджувати нові більш комплексні алгоритми для формування рекомендацій.Технічно колаборативне фільтрування може бути реалізоване двома основними способами: модель на основі користувачів та модель на основі об'єктів [22].

1. Модель на основі користувачів (User Based method): Цей підхід визначає схожість між користувачами на основі їхніх дій та виборів. Наприклад, якщо користувач А та користувач Б мають схожі інтереси чи вибирають подібні товари, система передбачає, що їм подобаються подібні об'єкти.
2. Модель на основі об'єктів (Content Based method): У цьому випадку алгоритм знаходить схожість між об'єктами чи товарами. Якщо користувач взяв участь у виборі товару, система рекомендує інші товари, які були обрані або переглянуті іншими користувачами з подібними смаками [23].

В рамках даної роботи реалізовано систему рекомендації на основі користувачів. Розглянемо математичні основи та суть цього методу.

Нехай *U* – множина користувачів, *I* – множина товарів, *R* – матриця оцінок, де *Rui*​ представляє оцінку, яку користувач *u* надав товару *i*. Для обчислення схожості між двома користувачами, використовується косинусна подібність (2.1)

Де – оцінки товарів *i* від користувачів *u1* та *u2* відповідно.

*I* – множина товарів по яким існує оцінка від обох користувачів.

Після обчислення подібності між користувачами, можна використати отримане значення для прогнозу оцінки, яку новий користувач *unew* може поставити товару *inew* ​. Прогнозована оцінка обчислюється за наступною формулою (2.2):

де – прогнозована оцінка яку користувач *unew* може поставити товару *inew*

До переваг такого підходу можна віднести простоту реалізації: user-based метод відносно простий для реалізації, оскільки використовує лише подібність між користувачами.

Крім того, рекомендації в цьому методі базуються на відмінностях між схожими користувачами, що може бути більш природнім способом побудови рекомендацій на відміну від моделі на основі об’єктів, які можуть мати суттєві відмінності, що може впливати на точність прогнозування.

Натомість User-Based підхід до колаборативної фільтрації може мати декілька значних недоліків, таких як проблема холодного старту, яка виникає, коли новий користувач або товар не має достатньо історії для ефективного визначення схожості. Крім того, з ростом кількості користувачів чи товарів суттєво зростає обчислювальна складність. Вирішенням зазначених проблем може бути використання гібридних методів або додаткових евристик та введення регуляризації для контролю перенавчання та забезпечення кращої узагальненості моделі. Для ефективної роботи з великими наборами даних, можна використовувати апроксимації та методи для зменшення обчислювального навантаження [24].

Отже, метод колаборативної фільтрації на основі користувачів, враховуючи обрану предметну область, можна вважати оптимальним рішенням для побудови рекомендаційної моделі для ІС електронної комерції.

## 2.3 Висновки до розділу 2

В розділі здійснено проектування структури майбутнього веб-додатку для е-комерції, в якості архітектурної моделі обрано трьохрівневу клієнт-серверну архітектуру яка є найбільш оптимальною для реалізації поставленого завдання, оскільки забезпечує достатню децентралізацію та модульність системи, спрощує інтеграцію інших сервісів у подальшому.

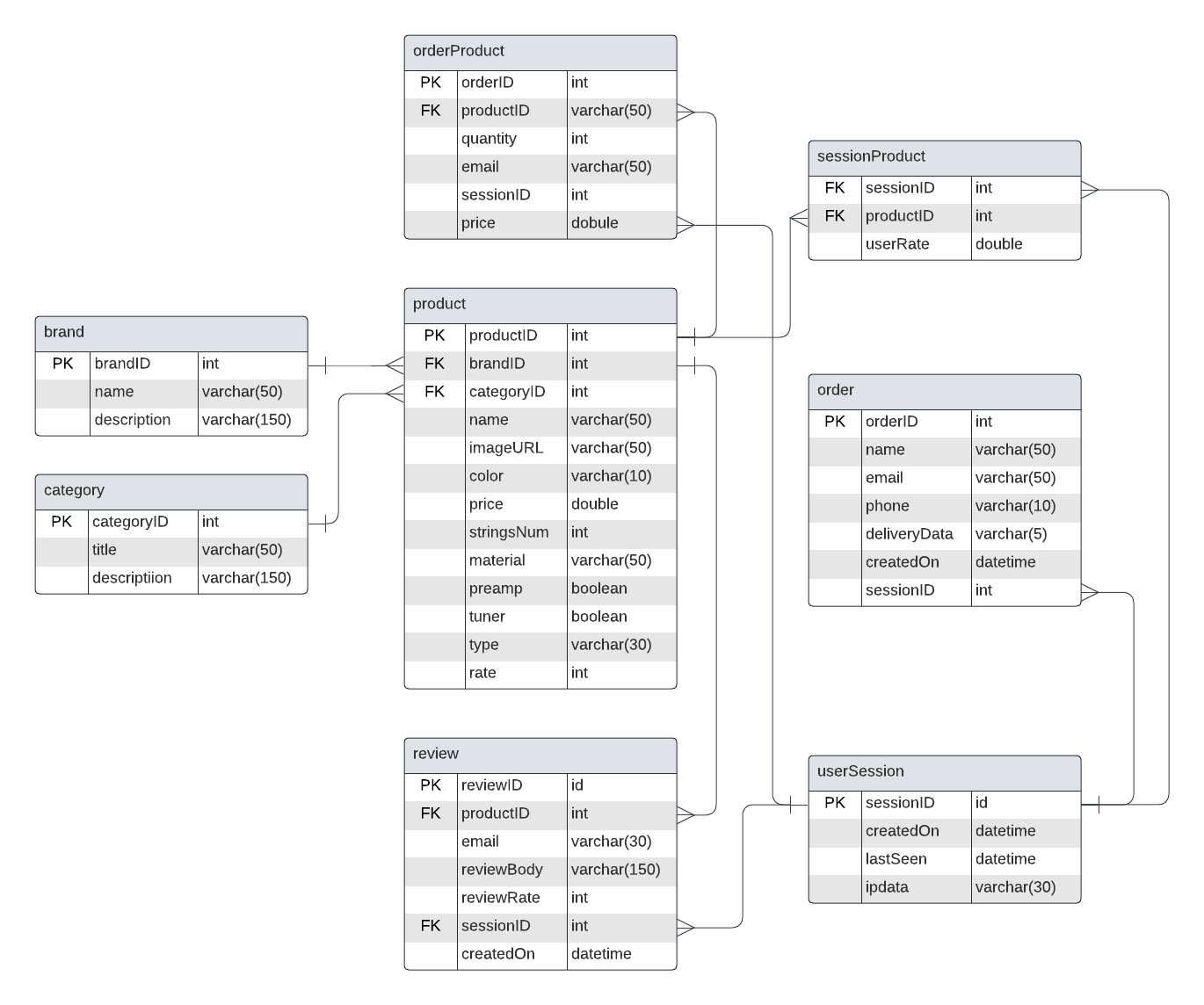
Розглянуто системи рекомендації контенту, на прикладі онлайн-магазину Amazon визначено основні методи та практики які використовуються у сучасній е-комерції.

В якості методу для формування рекомендацій обрано метод колаборативної фільтрації, та підхід на основі користувачів. Розглянуто суть цього методу, недоліки та переваги такого підходу та його математичну основу.

# РОЗДІЛ 3 РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ Е-КОМЕРЦІЇ

## 3.1 Проектування та реалізація бази даних для платформи е-комерції

Відповідно до поставленого завдання було розроблено БД для онлайн платформи з продажу музичного обладнання. На Рис. 3.1 зображено схему БД.

  
Рисунок. 3.1 Схема бази даних для онлайн магазину «music-shop»

Розглянемо схему зазначеної БД, зміст таблиць та зв’язки між ними. Центральною сутністю у розроблюваній БД є таблиця «product», яка містить інформацію про конкретний товар в системі. Дана таблиця описує сутність товару в інтернет-магазині. Таблиця «brand» має зв'язок один до багатьох з таблицею «product», також «category» має зв'язок один до багатьох з таблицею «product», оскільки багато товарів може підпадати під один бренд або категорію. Також, вона встановлює зв'язок один до багатьох з таблицею «review», оскільки один товар може мати багато відгуків, та «orderProduct» має зв'язок один до багатьох з таблицею «product», оскільки один товар може бути включений в багато замовлень. В Таблиці 3.1 подано детальну структуру наведеної таблиці.

Таблиця 3.1 – Структура таблиці «product»

|  |  |  |  |
| --- | --- | --- | --- |
| **Ім’я** | **PK** | **Тип** | **Опис** |
| productID | ✓ | INT | Унікальний ідентифікатор товару |
| brandID |  | INT | Зовнішній ключ до brand (brandID) |
| categoryID |  | INT | Зовнішній ключ до category (categoryID) |
| Name |  | VARCHAR(50) | Назва товару |
| imageURL |  | VARCHAR(50) | URL зображення товару |
| Color |  | VARCHAR(10) | Колір товару |
| Price |  | DOUBLE | Ціна товару |
| stringsNum |  | INT | Кількість струн (для музичних інструментів) |
| Material |  | VARCHAR(50) | Матеріал товару |
| Preamp |  | BOOLEAN | Наявність підсилювача (для музичних інструментів) |
| Tuner |  | BOOLEAN | Наявність тюнера (для музичних інструментів) |
| Type |  | VARCHAR(30) | Тип товару |
| Rate |  | INT | Рейтинг товару |

Таблиця «review» встановлює один до багатьох зв'язок з таблицею «product», оскільки один відгук відноситься до одного товару. Таблиця «orderProduct» має один до багатьох зв'язок з таблицею «order», оскільки один товар може брати участь у багатьох замовленнях. Детальний опис структури таблиці «review» наведено у Таблиці 3.2.

Таблиця 3.2 – Структура таблиці «review»

|  |  |  |  |
| --- | --- | --- | --- |
| **Ім’я** | **PK** | **Тип** | **Опис** |
| reviewID | ✓ | INT | Унікальний ідентифікатор відгуку |
| productID |  | INT | Зовнішній ключ до product (productID) |
| Email |  | VARCHAR(30) | Адреса електронної пошти користувача |
| reviewBody |  | VARCHAR(150) | Текст відгуку |
| reviewRate |  | INT | Рейтинг відгуку |
| sessionID |  | VARCHAR(50) | Унікальний ідентифікатор сесії користувача |
| createdOn |  | DATETIME | Дата та час створення відгуку |

Таблиця order встановлює зв'язок один до багатьох з таблицею userSession, оскільки одне замовлення може бути пов'язане із кількома сесіями користувача. Також, вона має зв'язок один до багатьох з таблицею «orderProduct», оскільки одне замовлення може включати в себе кілька товарів.

Таблиця 3.3 – Структура таблиці «order»

|  |  |  |  |
| --- | --- | --- | --- |
| **Ім’я** | **PK** | **Тип** | **Опис** |
| orderID | ✓ | INT | Унікальний ідентифікатор замовлення |
| name |  | VARCHAR(50) | Ім'я клієнта |
| email |  | VARCHAR(50) | Адреса електронної пошти клієнта |
| phone |  | VARCHAR(10) | Номер телефону клієнта |
| deliveryData |  | VARCHAR(150) | Інформація про доставку |
| createdOn |  | DATETIME | Дата та час створення замовлення |
| sessionID |  | INT | Унікальний ідентифікатор сесії користувач |

Таблиця «userSession» містить інформацію про сесії користувачів. Дані про сесію користувача потрібні для того, щоб відслідковувати дії користувача на сайті, які потім будуть використовуватись системою рекомендації для формування списку рекомендацій конкретному користувачу. Для цієї мети також створено таблицю «sessionProduct», яка поєднує користувацьку сесію та товар, який зацікавив користувача. Таблиця «userSession» має зв'язок один до багатьох з таблицею «order», оскільки один користувач може мати кілька замовлень.

Таблиця 3.4 – Структура таблиці «userSession»

|  |  |  |  |
| --- | --- | --- | --- |
| **Ім’я** | **PK** | **Тип** | **Опис** |
| sessionID | ✓ | INT | Унікальний ідентифікатор сесії користувача |
| createdOn |  | DATETIME | Дата та час створення сесії |
| lastSeen |  | DATETIME | Дата та час останнього візиту користувача |
| ipdata |  | VARCHAR(30) | Інформація про IP-адресу |

Таблиці «category» та «brand» містять дані про категорії та бренди товарів відповідно. Таблиця «brand» має один до багатьох зв'язок з таблицею «product», оскільки один бренд може представляти багато товарів у магазині. Таблиця «category» також встановлює один до багатьох зв'язок з таблицею product, оскільки одна категорія може включати в себе кілька товарів.

Таблиця 3.5 – Структура таблиці «brand»

|  |  |  |  |
| --- | --- | --- | --- |
| **Ім’я** | **PK** | **Тип** | **Опис** |
| brandID | ✓ | INT | Унікальний ідентифікатор бренду |
| name |  | VARCHAR(50) | Назва бренду |
| description |  | VARCHAR(150) | Опис бренду |

Таблиця 3.6 – Структура таблиці «category»

|  |  |  |  |
| --- | --- | --- | --- |
| **Ім’я** | **PK** | **Тип** | **Опис** |
| categoryID | ✓ | INT | Унікальний ідентифікатор категорії товарів |
| title |  | VARCHAR(50) | Назва категорії |
| description |  | VARCHAR(150) | Опис категорії |

Аналогічно, таблиця «productOrders» встановлює один до багатьох зв'язок із таблицями «product» та «order», щоб вказати, що один товар може брати участь у багатьох замовленнях, і навпаки.

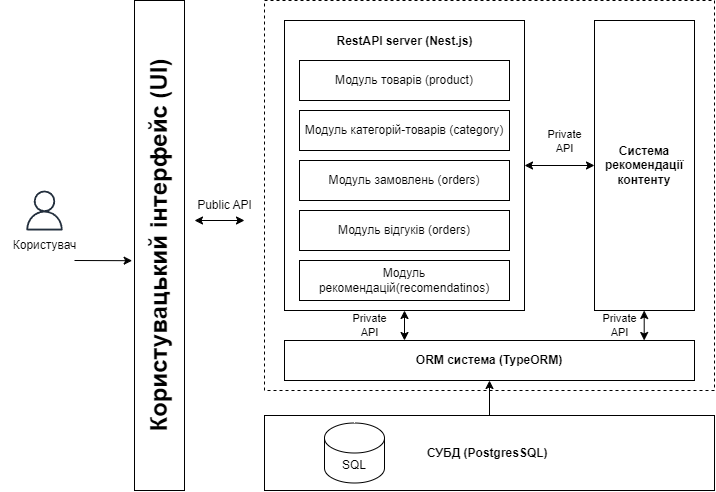
Таблиця 3.7 – Структура таблиці «productOrders»

|  |  |  |  |
| --- | --- | --- | --- |
| **Ім’я** | **PK** | **Тип** | **Опис** |
| orderID |  | INT | Зовнішній ключ до order (orderID) |
| productID |  | INT | Зовнішній ключ до product (productID) |
| quantity |  | INT | Кількість товару в замовленні |
| sessionID |  | INT | Унікальний ідентифікатор сесії користувача |
| price |  | DOUBLE | Ціна товару в замовленні |

## 

## 3.2 Реалізація серверної частини веб-додатку для е-комерції

Проаналізувавши вимоги до ІС, було прийнято рішення розробки серверної частини додатку, дотримуючись архітектурного підходу RESTfull API. Основна ідея полягає в тому, щоб розділити функціональність на окремі ресурси, якими можна керувати та маніпулювати за допомогою стандартних HTTP-запитів. Взаємодія з ресурсами відбувається за допомогою стандартних методів HTTP, таких як GET (отримання), POST (створення), PUT (оновлення) та DELETE (видалення) [25].Такий підхід є оптимальним для реалізації ІС з визначеною структурою, оскільки передбачає простоту інтеграції та спрощену структуру запитів до серверу. Масштабованість та відкритість такої системи дозволить в майбутньому інтегрувати в неї інші сервіси, наприклад рекомендаційну систему. Загальна структура інформаційної системи подана на Рис 3.2.

  
Рисунок 3.2 Структура інформаційної системи е-комерції

Інтеграція користувацького інтерфейсу та серверу відбувається через відкрите API, використовуючи структуру REST. Цей підхід дозволяє ефективно передавати дані між фронтендом і бекендом, використовуючи стандартні HTTP-методи.

Система також використовує приватне API для внутрішньої комунікації між модулями. Це дозволяє модулям системи обмінюватися даними та виконувати спільні операції, забезпечуючи при цьому високу безпеку та обмежуючи доступ до внутрішніх функцій системи.

Кожен модуль системи має централізований доступ до бази даних через систему об’єктно-реляційного відображення (ORM). Це спрощує взаємодію модулів із базою даних, оскільки вони можуть використовувати об’єктну модель даних замість прямих SQL-запитів. Такий підхід робить систему більш масштабованою, а також сприяє відокремленню бази даних від модулів, покращуючи їхню незалежність.

Для реалізації серверної частини додатку обрано платформу Node.js та бекенд-фреймворк Nest.js. За допомогою менеджеру пакетів NPM було ініційовано проект та встановлено необхідні бібліотеки.

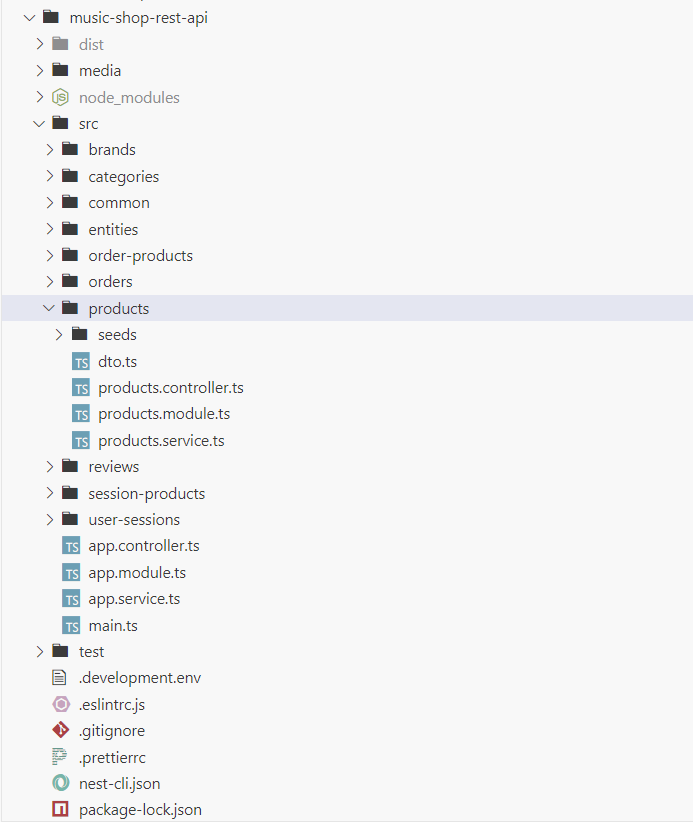
Nest.js передбачає зручне розділення коду на модулі, контролери та сервіси. Модулі: В Nest.js модулі використовуються для групування компонентів за функціональністю. Це дозволяє створювати відокремлені, самостійні блоки коду, які легко повторно використовувати в інших частинах додатку або навіть в інших проектах.

Контролери визначають обробники маршрутів і відповідають за обробку HTTP-запитів. Вони взаємодіють з клієнтом та викликають відповідні методи сервісів для обробки бізнес-логіки.

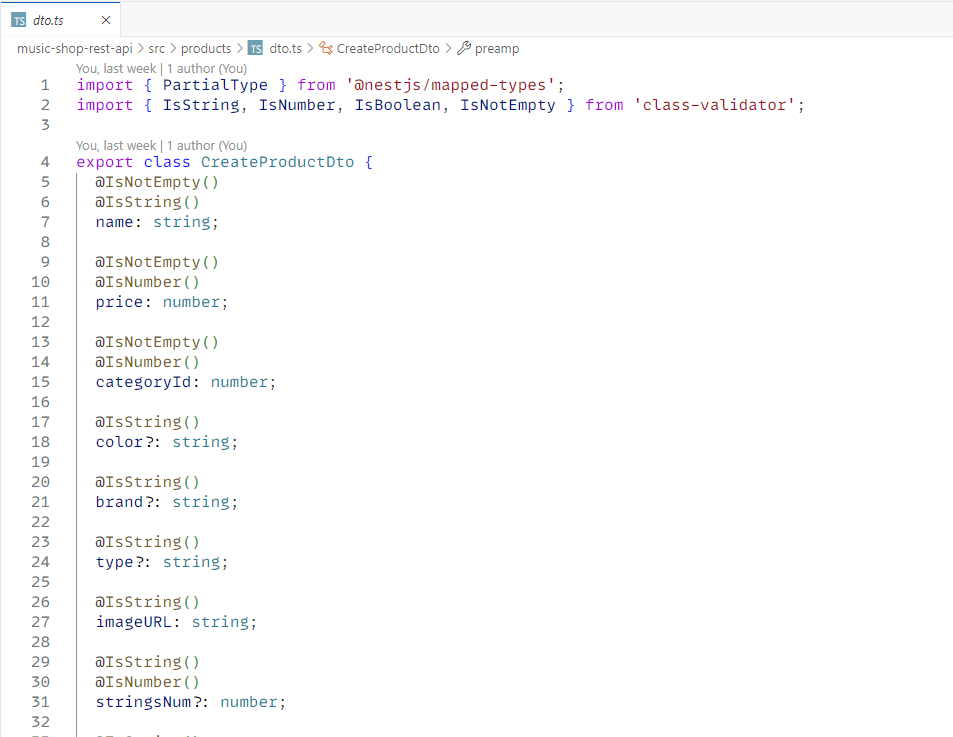
Сервіси відповідають за виконання бізнес-логіки. Вони можуть містити методи для отримання, оновлення та видалення даних, а також виконання інших операцій, необхідних для коректної роботи додатку.

Така структура дозволяє відокремити логіку додатку на модулі та компоненти, зробити код більш зрозумілим та легко підтримуваним. Nest.js також надає вбудовану систему впровадження залежностей, що полегшує управління зв'язками між різними компонентами та забезпечує їхню автоматичну ініціалізацію.

Nest також надає можливість створювати чітку і лаконічну файлову структуру проекту. Для кожної з сутностей системи, створено теку яка містить необхідні модулі, контролери, сервіси та інші допоміжні ресурси (Рис. 3.3).

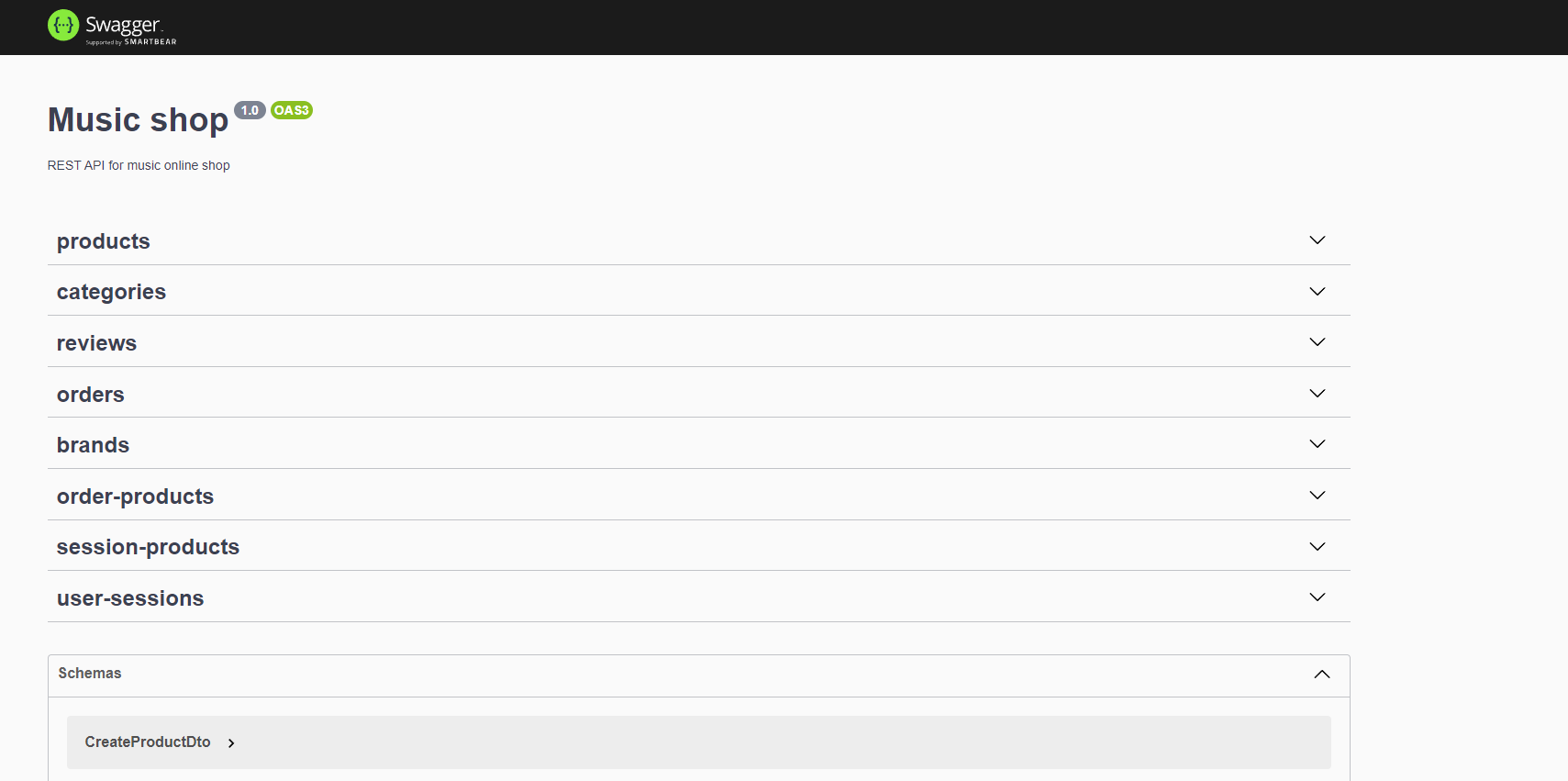
   
Рисунок. 3.3 Файлова структура серверної частини проекту

Окремої уваги також заслуговує файл dto.ts. Файли DTO (Data Transfer Objects) використовуються для передачі даних між різними частинами програми. Основною їхньою метою є упаковка та передача обмеженого набору даних, необхідного для виконання конкретної операції чи передачі інформації. У контексті Nest.js DTO використовуються для обміну даними між клієнтом та сервером, а файл dto.ts містить чітко визначену типізовану структуру даних яку отримує чи відправляє веб-сервер на запит клієнта. Він також може містити функціонал для валідації, трансформації та перетворення даних. Наприклад, файл «products/dto.ts» містить перевірки типів для об’єкту який передається клієнтом для створення нового продукту (Рис. 3.4).

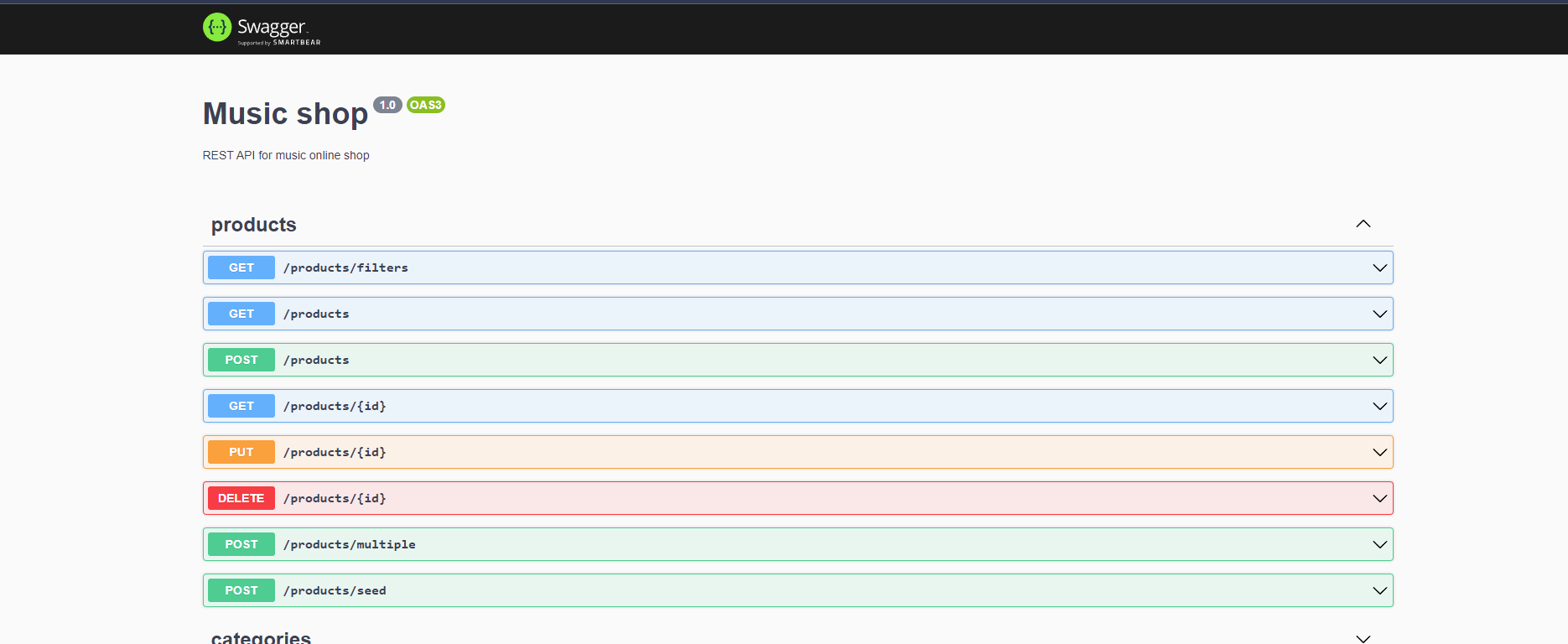
  
Рисунок. 3.4 Зміст файлу DTO для сутності product

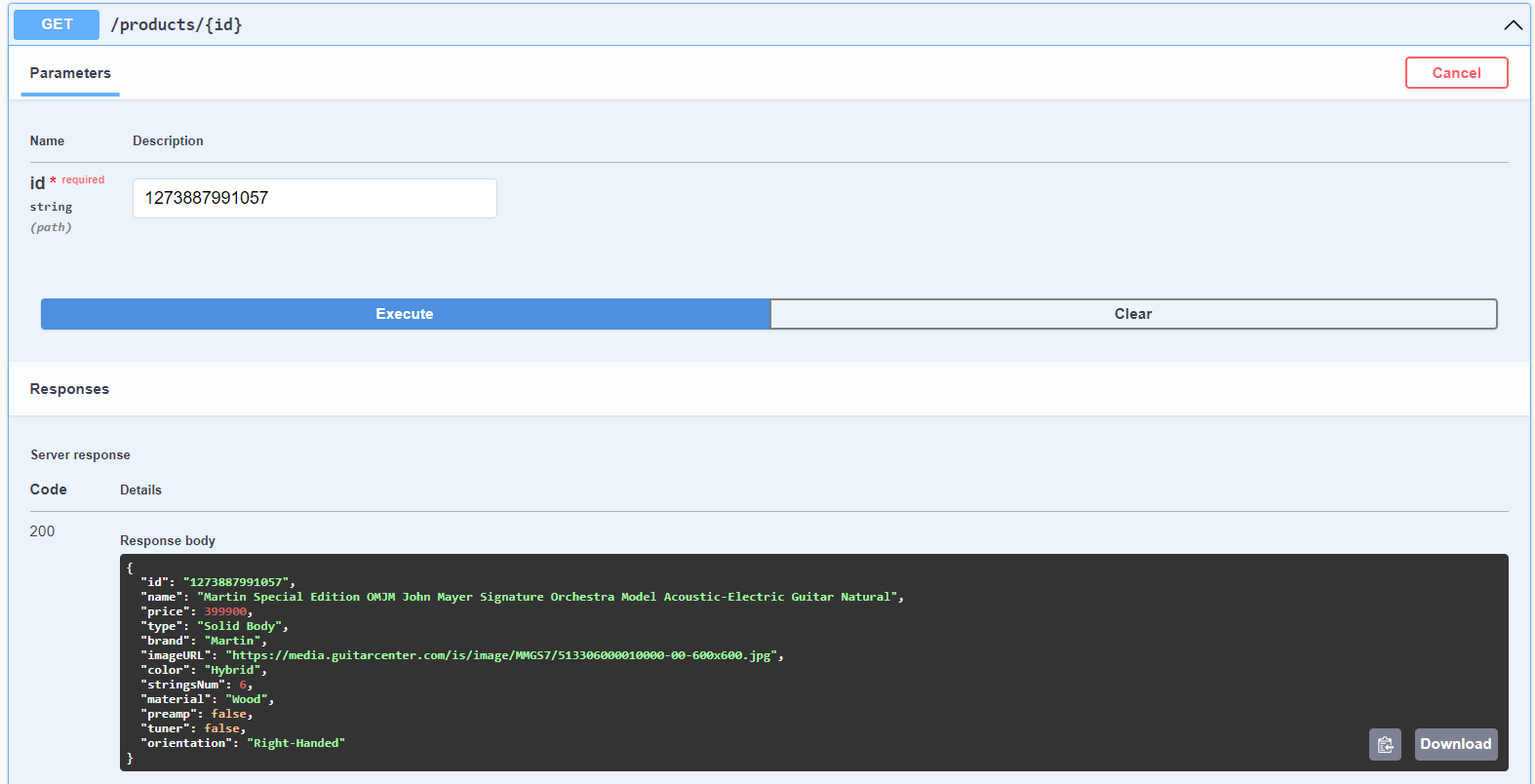
Для документації реалізованого серверного API використано інструмент Swagger, який використовується для автоматичного створення документації API на основі анотацій та метаданих в коді серверної частини Nest.js. Мета такого підходу полягає в тому, щоб спростити процес створення та підтримки документації, що важливо для подальшої підтримки ІС. Модуль автоматично аналізує код та використовує метадані, анотації та коментарі для створення документації API в форматі OpenAPI. Це значно зменшує трудомісткість написання та підтримки документації. Згенерована документація включає інтерактивний інтерфейс Swagger, який дозволяє розробникам легко взаємодіяти з API прямо з браузера. Інформація в документації завжди актуальна, оскільки вона автоматично оновлюється на основі змін у вихідному коді. Це дозволяє уникнути розбіжностей між документацією та реальною реалізацією API.

На Рис. 3.5 зображено інтерфейс задокументованого API для реалізації основних модулів ІС.

  
Рисунок 3.5 Основні модулі API

Кожен з модулів містить відповідні запити для отримання, створення, редагування та видалення даних з БД, на Рис. 3.6 зображено доступні запити для модулю product.

 Рисунок 3.6 HTTP запити для модулю product

  
Рисунок 3.7 Результат запиту на отримання товару

У Таблиці 3.8 наведено наявні запити, які відповідають моделі «CRUD», та забезпечують базове функціонування модулю товарів в системі.

Таблиця 3.8 – Список API запитів для модулю товарів «product»

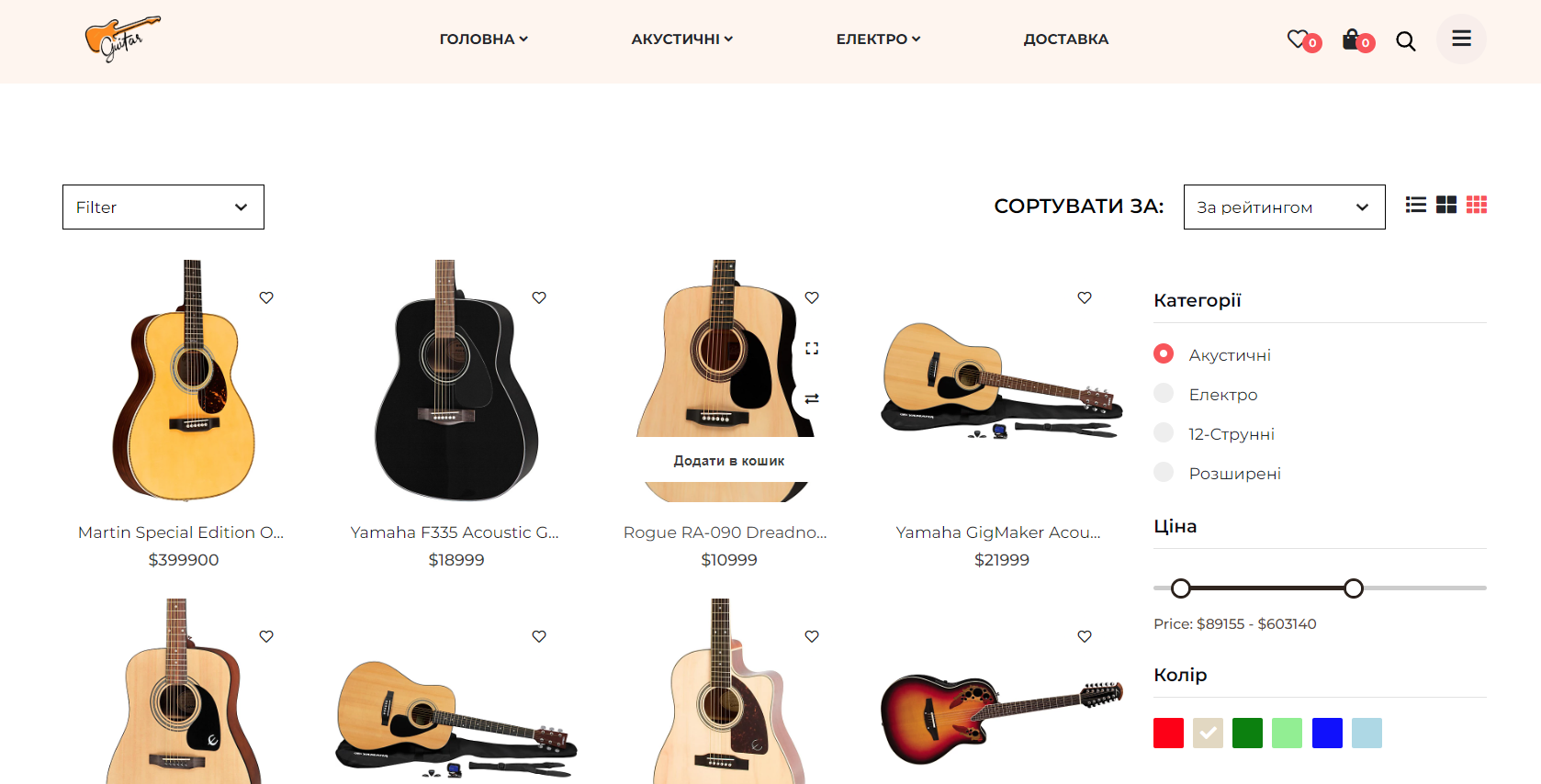
|  |  |  |
| --- | --- | --- |
| **Маршрут запиту** | **Параметри запиту** | **Опис** |
| **GET** /products/filters | – | Отримання списку наявних фільтрів для продуктів |
| **GET** /products | page – номер сторінки  page\_size – розмір сторінки  filters – фільтри для товарів | Отримання масиву продуктів за фільтрами та сторінкою |
| **GET** /products/{id} | id – ідентифікатор товару | Отримати товар за ID |
| **POST** /products | модель product | Створення нового продукту |
| **PUT** /products/{id} | id – ідентифікатор товару | Оновити продукт за ID |
| **DELETE** /products/id | id – ідентифікатор товару | Видалити товар за id |

Аналогічно реалізовані інші модулі системи, чіткий розподіл між операціями над даними надає можливість розробляти легко розширювані модулі, що сприяє високій гнучкості та масштабованості системи. Кожен модуль виконує конкретні функції, які можуть незалежно розвиватися та модифікуватися без впливу на інші частини системи.

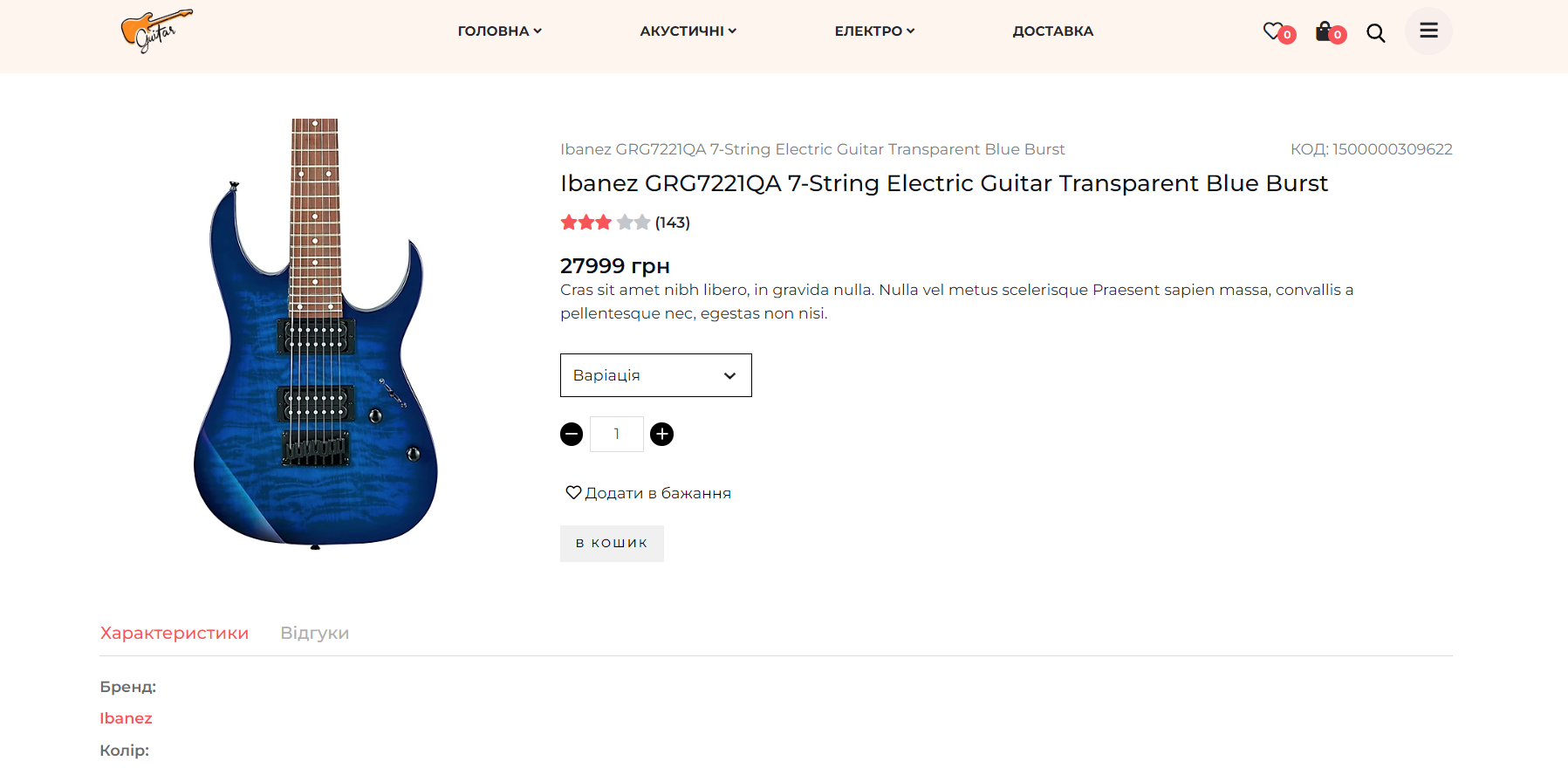
Такий підхід спрощує процес підтримки, розширення та вдосконалення системи, забезпечуючи зручну архітектуру для роботи з різними функціональними можливостями.

## 3.3 Реалізація клієнтської частини веб-додатку для е-комерації

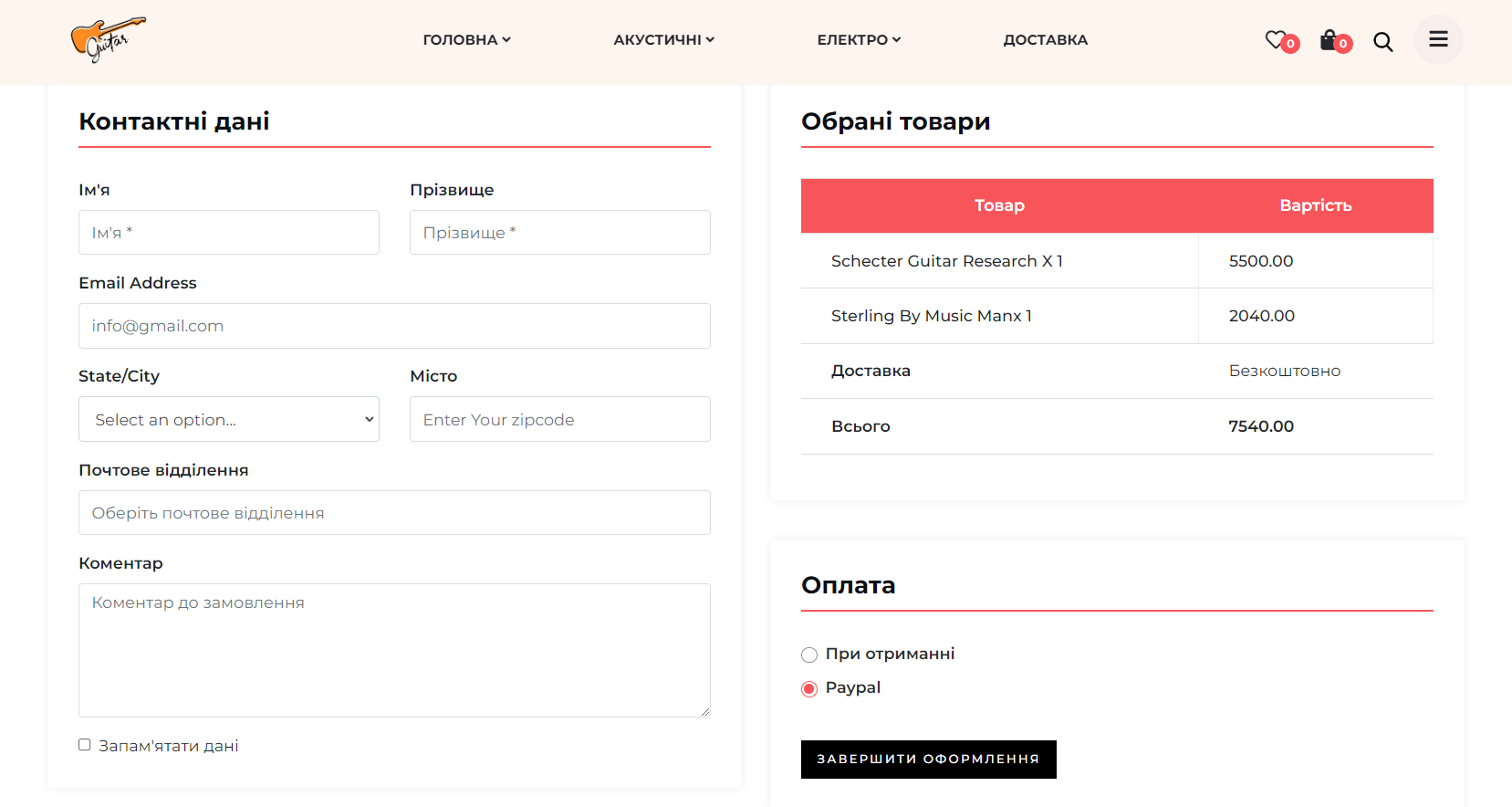
Клієнтський додаток реалізовано на основні Vue.js та Vuex Store, та має декілька ключових модулів. Основний модуль – це сітка товарів, або product-grid. На цій сторінці користувач може переглядати наявні в системі товари, сортувати, фільтрувати, додавати в кошик, тощо. На Рисунку 3.8 зображено вигляд сторінки з вітриною товарів.

  
Рисунок 3.8 – Вигляд сторінки з вітриною товарів

При переході на конкретний товар, користувач матиме змогу переглядати детальні характеристики товару, відгуки інших покупців, додати товар в замовлення тощо. На Рис. 3.9 подано вигляд сторінки товару.

  
Рисунок. 3.9 – Вигляд сторінки з конкретним товаром

Після того як користувач визначився с замовленням він може здійснити його оформлення на сторінці оформлення замовлення. На Рис. 3.10 подано вигляд сторінки оформлення, де користувач має надати необхідні контакті дані.

  
Рисунок 3.10 Сторінка оформлення замовлення

Після того як користувач ввів необхідні дані та на натиснув кнопку «завершити оформлення», на сервер буде відправлено POST запит на створення в БД нового замовлення в таблиці «orders», яке вже потім буде опрацьовано відповідним сервісом.

## 3.4 Реалізація системи рекомендації товарів

Також за поставленим завданням розроблено систему рекомендації контенту, яка використовує метод колаборативної фільтрації, аналізує історію товарів які переглядав користувач та повертає список товарів які можуть зацікавити користувача. В якості мови програмування для реалізації моделі обрано мову програмування Python та декілька бібліотек для машинного навчання.

Бібліотека pandas слугує основним інструментом для створення та обробки DataFrame, який використовується для структурування і представлення інформації про взаємодію користувачів з продуктами. Це дозволяє ефективно працювати з табличними даними та здійснювати різноманітні операції над ними [26].

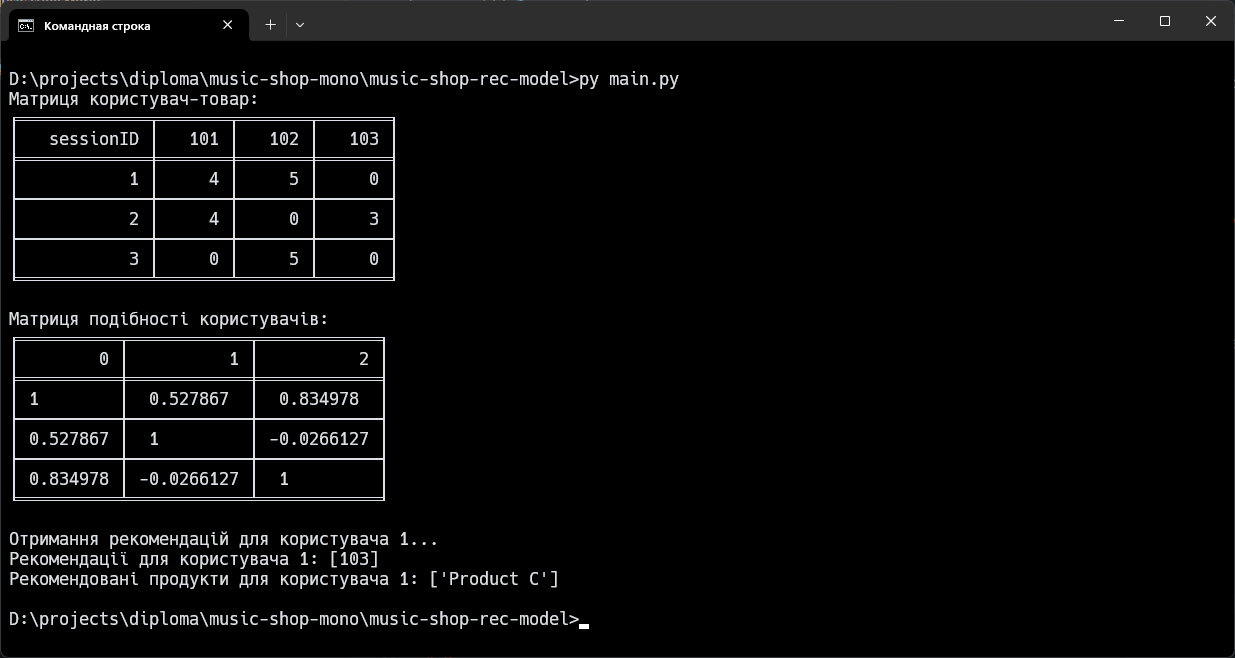
У сфері машинного навчання ключову роль відіграє бібліотека scikit-learn, яка надає необхідні інструменти для обчислення косинусної схожості між користувачами. Це дозволяє визначати ступінь подібності між їхніми уподобаннями на основі взаємодії з продуктами.

Спершу, дані про інтеграції завантажуються в DataFrame за допомогою бібліотеки pandas. Далі, на основі цих даних, побудовано матрицю користувач-товар, яка служить основою для аналізу.

Використовуючи бібліотеку scikit-learn, обчислюється схожість між користувачами на основі косинусної подібності. Це дозволяє визначити ступінь подібності між їхніми вподобаннями.

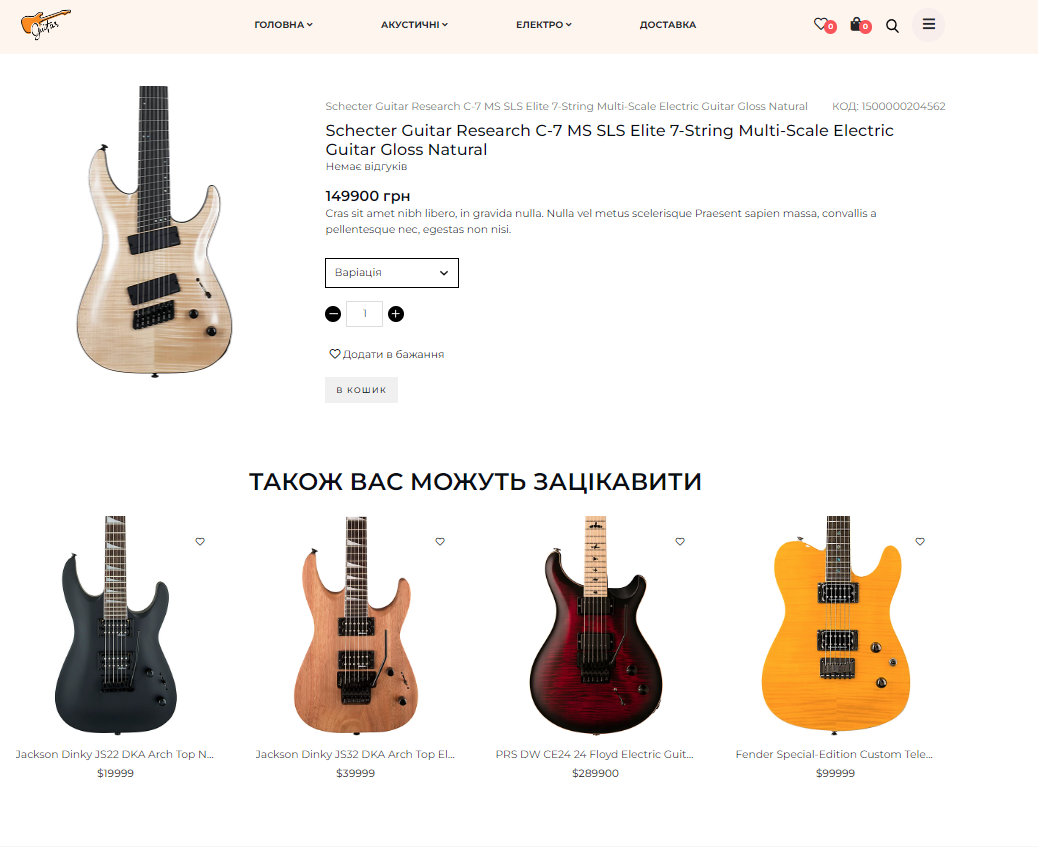
За допомогою отриманих результатів алгоритм генерує персоналізовані рекомендації для кожного користувача. Результати, такі як рекомендовані продукти, можуть бути виведені у вигляді списку для подальшого використання у веб-додатку або іншому інтерфейсі. Крім того, можуть бути виведені додаткові метрики, такі як матриці схожості та інші показники, для аналізу та візуалізації ефективності системи рекомендацій.

Результат виконання програми у консольному вікні, з виведенням матриці подібності, та проміжних обчислень подано на Рис. 3.11.

  
Рисунок 3.11 Результат роботи рекомендаційної моделі для конкретного користувача

На основі побудованої моделі розроблено додатковий веб-сервер який за запитом генерує рекомендації для конкретного користувача. Веб-клієнт здійснює запит на отримання рекомендацій до основного серверу, в свою чергу основний RestAPI сервер здійснює запит до сервісу рекомендацій та передає отримані дані на веб-клієнт, де вони можуть бути представлені користувачу в різних місцях веб-додатку.

На Рис. 3.12 зображено інтеграцію системи рекомендацій у веб-додаток інтернет магазину з продажу музичного обладнання.

  
Рисунок. 3.12 Інтеграція рекомендаційної системи в веб-додаток е-комерції

## 3.5 Висновки до розділу 3

У практичній частині дипломної роботи здійснено розробку веб-додатку е-комерції з продажу музичного обладнання. З використанням обраних інструментів було розроблено реляційну базу-даних ІС, побудовано звязки між таблицями. З використанням фреймворку Nest.js та платформи Node.js розроблено серверну частину веб-додатку, який представляє RestAPI сервер, який здійснює основні операціями над даними в БД. На базі фреймворку Vue.js побудовано веб-клієнт додатку, розроблено основні модулі та функціонал.

Реалізовано і інтегровано систему рекомендацій, яка використовує мову програмування Python та бібліотеку з машинного навчання scikit-learn. Рекомендаційна система отримує масив даних переглянутих користувачами товарів, використовуючи метод колаборативної фільтрації визначає подібність між користувачами та формує список рекомендованих товарів для конкретного користувача. Розроблений модуль інтегровано в серверну частину додатку та використано при відображені рекомендованих товарів на веб-клієнті.

# ВИСНОВКИ

Відповідно до теми дипломної роботи у першому розділі здійснено аналітичний огляд галузі електронної комерції. Розглянуто основні складові обраної предметної області, досліджено чинники які впливають на успіх платформ е-комерції. Проведено огляд існуючих засобів візуалізації інформації з використанням веб-технологій, розглянуто їх переваги та недоліки.

У другому розділі дипломної роботи здійснено проектування ІС е-комерції, побудовано трирівневу модель архітектури веб-додатку е-комерції, розглянуто основні компоненти системи онлайн торгівлі та особливості взаємодії між ними. На прикладі платформи інтернет торгівлі Amazon, здійснено аналітичний огляд систем рекомендації контенту, обґрунтовано доцільність їх використання. Розглянуто основні методи генерації рекомендацій та їх математичний апарат.

В третьому розділі детально розглянуто реалізацію веб-додатку інтернет магазину музичного обладнання, спроектовано базу даних системи, серверну частину та користувацький інтерфейс. В якості окремого сервісу реалізовано модуль рекомендації контенту на основі методу колаборативної фільтрації, розроблений модуль інтегровано до ІС е-комерції.

Реалізована ІС побудована на основі сучасних архітектурних рішень, що сприяє масштабуванню та розширенню системи у подальшому. Розроблений додаток відповідає основним вимогам галузі е-комерції, забезпечуючи швидкодію та надання персоналізованого клієнтського досвіду. Отже, мета роботи яка полягала у удосконалення процесу продажу товарів комерційною організацією шляхом використання програмних рішень на основі веб-технологій досягнута.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Краус К.М, Електронна комерція та Інтернет-торгівля: навчально методичний посібник / К.М Краус, Н.М Краус, О.В Манжура, – Київ: Аграр Медіа Груп, 2021. – 454с.
2. Здоренко Ю.М., Андрєєв А.С., Бочкарь В.О. Методи забезпечення доступності інформаційних ресурсів на основі використання мікросервісної архітектури веб-додатків // Збірник тез доповідей Міжнар. наук.-практич. конфер. «Інновації та перспективні шляхи розвитку інформаційних технологій» (б груд. 2023 р., м. Черкаси) Упоряд. : Т. О. Прокопенко, Я. В. Тарасенко. М-во освіти і науки України, Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2023. – 293 с
3. How fast should a website load in 2023. URL: <https://www.browserstack.com/guide/how-fast-should-a-website-load> (дата звернення: 05.11.2023).
4. Аналітичні дослідження залежності конверсії онлайн бізнесу від швидкості завантаження веб-ресурсів. URL: <https://www.portent.com/blog/analytics/research-site-speed-hurting-everyones-revenue.htm> (дата звернення: 05.11.2023).
5. L. Wagner. Web Performance in Action. Manning, 2017. – 376.
6. Тлумачний словник з інформатики / Г. Г. Півняк, Б. С. Бусигін, М. М. Дівізінюк та ін. — Д., Нац. гірнич. ун-т, 2010. — 600 с
7. Wappalyzer research on e-commerce. URL: <https://www.wappalyzer.com/technologies/>
8. Технічна документація бібліотеки React.js. URL: <https://react.dev/reference/react>
9. Технічна документація фреймворку Angular.js URL: <https://angular.io/guide/understanding-angular-overview>
10. Технічна документація до фреймворку Vue.js. URL: <https://react.dev/blog/2023/03/16/introducing-react-dev>
11. Технічна документація фреймворку Nuxt.js URL: <https://nuxt.com/docs/getting-started/introduction>
12. The MVVM Pattern. URL: <https://learn.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)?redirectedfrom=MSDN>
13. Patterns of Enterprise Application Architecture / Fowler M., Rice D., Foemmel M., Hieatt E., Mee R., Stafford R., Wesley A. 2002. 560 c.
14. Технічна документація до бібліотеки Vuex. URL: <https://vuex.vuejs.org/#what-is-a-state-management-pattern>
15. Node.js в дії / Кантелон М., Хартер М., Головайчук T., Райлих Н. Print2print, 2018. 432 c.
16. Технічна документаця фреймворку Nest.js. URL: <https://docs.nestjs.com>
17. Eckel B., Thinking in Java 4th Edition. Pearson. 2006 – 1150
18. Grinberg М., Flask Web Development: Developing Web Applications with Python 2nd Edition. O'Reilly, 2018 – 312
19. Silberschatz A., Database System Concepts. McGraw Hill, 2019 -1376
20. Regina O., PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database 3rd Edition. O'Reilly, 2017
21. Altintas I., McAuley J. Amazon Recommender System. URL: <https://library.ucsd.edu/dc/object/bb8503744c/_2_1.pdf>
22. Снитюк В.Е. Прогнозирование, модели, методы и алгоритмы: учебное пособие. – К.: «Маклаут», 2008. – 364 с.
23. Sammut C., Webb J. (Eds.). Encyclopedia of Machine Learning. — NY, USA : IBM T. J.Watson Research Center, 2010. — Т. 1. — С. 829-838
24. Понизовкин Д.М. Построение оптимального графа связей в системах коллаборативной фильтрации (журнал) // «Программные системы: теория и приложения». — 2011. — № 4(8). — С. 107-114
25. Richardson L. RESTful Web APIs: Services for a Changing World. O'Reilly. 2013 – 406.
26. Geron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 3rd Edition. O'Reilly. 2022. – 850.

# ДОДАТОК А ВИХІДНИЙ КОД ПРОГРАМИ

// Product.module.ts

import { Module } from '@nestjs/common';

import { TypeOrmModule } from '@nestjs/typeorm';

import { ProductsController } from './products.controller';

import { ProductsService } from './products.service';

import { Product, Category, Brand } from '../entities';

@Module({

imports: [TypeOrmModule.forFeature([Product, Category, Brand])],

controllers: [ProductsController],

providers: [ProductsService],

})

export class ProductsModule {}

// Product.controller.ts

import { Controller, Get, Post, Body, Param, Delete, Put, Query } from '@nestjs/common';

import { ProductsService } from './products.service';

import { CreateProductDto, UpdateProductDto } from './dto';

import { ApiTags } from '@nestjs/swagger';

@ApiTags('products')

@Controller('products')

export class ProductsController {

  constructor(private readonly productsService: ProductsService) {}

  @Get('filters')

  getFilters() {

    return this.productsService.getAvailableFilterValues();

  }

  @Get()

  findAll(@Query() filters: any, @Query('page') page = 1, @Query('pageSize') pageSize = 24) {

    return this.productsService.findAll(filters, page, pageSize);

  }

  @Get(':id')

  findOne(@Param('id') id: string) {

    return this.productsService.findOne(+id);

  }

  @Post()

  create(@Body() createProductDto: CreateProductDto) {

    return this.productsService.create(createProductDto);

  }

  @Post('multiple')

  createMultiple(@Body() createProductDto: CreateProductDto[]) {

    const promises = createProductDto.map(productDTO => {

      this.create(productDTO);

    });

    return Promise.all(promises);

  }

  @Post('seed')

  seedProductsData() {

    return this.productsService.seedProducts();

  }

  @Put(':id')

  update(@Param('id') id: string, @Body() updateProductDto: UpdateProductDto) {

    return this.productsService.update(+id, updateProductDto);

  }

  @Delete(':id')

  remove(@Param('id') id: string) {

    return this.productsService.remove(+id);

  }

}

// Product.service.ts

import { Injectable, NotFoundException } from '@nestjs/common';

import { InjectRepository } from '@nestjs/typeorm';

import { Repository } from 'typeorm';

import { Product, Category } from '../entities';

import { CreateProductDto, ProductFiltersDto, UpdateProductDto } from './dto';

import { getPrepearedData } from './seeds/seed.transformer';

import { toArr } from 'src/common/helpers';

@Injectable()

export class ProductsService {

constructor(

@InjectRepository(Product)

private readonly productRepository: Repository<Product>,

@InjectRepository(Category)

private readonly categoryRepository: Repository<Category>,

) {}

private readonly FilterTypes = {

Range: 'range',

Multiple: 'multiple',

Select: 'select',

} as const;

async getAvailableFilterValues(): Promise<Record<string, any[]>> {

const availableFilterValues: Record<string, any[]> = {};

const filterNames = this.productFilters

.filter(f => f.type !== this.FilterTypes.Range)

.map(f => f.field);

await Promise.all(

filterNames.map(async filterName => {

const distinctValues = await this.productRepository

.createQueryBuilder('product')

.select(`DISTINCT product.${filterName}`, filterName)

.getRawMany();

availableFilterValues[filterName] = distinctValues.map(item => item[filterName]);

}),

);

availableFilterValues.price = await Promise.all([

this.productRepository.minimum('price'),

this.productRepository.maximum('price'),

]);

return availableFilterValues;

}

async findAll(filters: ProductFiltersDto, page: number, pageSize: number): Promise<Product[]> {

const queryBuilder = this.productRepository.createQueryBuilder('product');

enum Operators {

IN = 'IN',

BETWEEN = 'BETWEEN',

EQUAL = '=',

}

const mapFilterToOperator = (filter: any) => {

switch (filter) {

case this.FilterTypes.Multiple:

return Operators.IN;

case this.FilterTypes.Range:

return Operators.BETWEEN;

case this.FilterTypes.Select:

return Operators.EQUAL;

}

};

type FilterCondition = { operator: `${Operators}`; value: string[] };

const conditions: Record<string, FilterCondition> = this.productFilters.reduce((acc, curr) => {

acc[curr.field] = {

operator: mapFilterToOperator(curr.type),

value: filters[curr.field],

};

return acc;

}, {});

Object.keys(conditions).map(key => {

if (conditions[key].value && conditions[key].operator === Operators.IN) {

conditions[key].value = toArr(conditions[key].value);

}

});

Object.entries(conditions).forEach(([key, condition]) => {

if (condition.value === void 0) {

return;

}

switch (condition.operator) {

case Operators.IN:

queryBuilder.andWhere(`product.${key} ${condition.operator} (:...${key})`, {

[key]: condition.value,

});

break;

case Operators.BETWEEN:

queryBuilder.andWhere(`product.${key} ${condition.operator} :${key}1 AND :${key}2`, {

[`${key}1`]: condition.value[0],

[`${key}2`]: condition.value[1],

});

break;

default:

queryBuilder.andWhere(`product.${key} ${condition.operator} :${key}`, {

[key]: condition.value,

});

}

});

const skip = (page - 1) \* pageSize;

queryBuilder.skip(skip).take(pageSize);

const products = await queryBuilder

.leftJoinAndMapOne('product.category', 'product.category', 'category')

.getMany();

return products;

}

async findOne(id: number): Promise<Product> {

const product = await this.productRepository.findOneBy({ id });

if (!product) {

throw new NotFoundException(`Product with ID ${id} not found`);

}

return product;

}

async create(createProductDto: CreateProductDto): Promise<Product> {

const { categoryId, ...productData } = createProductDto;

const category = await this.categoryRepository.findOneBy({

id: categoryId,

});

const product = this.productRepository.create({

...productData,

category,

});

return this.productRepository.save(product);

}

async seedProducts() {

const seedData = getPrepearedData(2);

for (const product of seedData) {

try {

await this.create(product);

console.log(`${product.id} has been saved to DB`);

} catch (err) {

console.log(err);

}

}

return 200;

}

async update(id: number, updateProductDto: UpdateProductDto): Promise<Product> {

const { categoryId, ...productData } = updateProductDto;

const category = await this.categoryRepository.findOneBy({

id: categoryId,

});

const product = await this.productRepository.preload({

id,

...productData,

category,

});

if (!product) {

throw new NotFoundException(`Product with ID ${id} not found`);

}

return this.productRepository.save(product);

}

async remove(id: number): Promise<void> {

const product = await this.productRepository.findOneBy({ id });

if (!product) {

throw new NotFoundException(`Product with ID ${id} not found`);

}

await this.productRepository.remove(product);

}

}

const state = {

productslist: [],

products: [],

activeProduct: null,

shuffleproducts: [],

wishlist: [],

compare: [],

searchProduct: [],

categories: [],

filters: []

}

// getters

const getters = {

getProductById: (state) => {

return id => state.products.find((product) => {

return product.id === +id

})

},

activeProduct: (state) => {

return state.activeProduct;

},

wishlistItems: (state) => {

return state.wishlist

},

compareItems: (state) => {

return state.compare

},

}

// mutations

const mutations = {

addToWishlist: (state, payload) => {

const product = state.products.find( item => item.id === payload.id )

const wishlistItems = state.wishlist.find( item => item.id === payload.id )

if (wishlistItems) {

} else {

state.wishlist.push({

...product

})

}

},

removeWishlistItem: ( state, payload ) => {

const index = state.wishlist.indexOf(payload)

state.wishlist.splice(index, 1)

},

addToCompare: (state, payload) => {

const product = state.products.find(item => item.id === payload.id)

const compareItems = state.compare.find(item => item.id === payload.id)

if (compareItems) {

} else {

state.compare.push({

...product

})

}

},

removeCompareItem: (state, payload) => {

const index = state.compare.indexOf(payload)

state.compare.splice(index, 1)

},

searchProduct: (state, payload) => {

payload = payload.toLowerCase()

state.searchProduct = []

if (payload.length) {

state.products.filter((product) => {

if (product.title.toLowerCase().includes(payload)) {

state.searchProduct.push(product)

}

})

}

},

getallProduct: (state, payload) => {

state.shuffleproducts = products.data

},

setProducts: (state, payload) => {

state.productslist = payload;

},

setActiveProduct: (state, payload) => {

state.activeProduct = payload;

},

setFilters: (state, payload) => {

state.filters = payload;

}

}

// actions

const actions = {

addToWishlist: (context, payload) => {

context.commit( 'addToWishlist', payload)

},

removeWishlistItem: ( context, payload ) => {

context.commit( 'removeWishlistItem', payload )

},

addToCompare: (context, payload) => {

context.commit('addToCompare', payload)

},

removeCompareItem: (context, payload) => {

context.commit('removeCompareItem', payload)

},

searchProduct: (context, payload) => {

context.commit('searchProduct', payload)

},

getallProduct: (context) => {

context.commit('getallProduct')

},

async fetchHotProducts(context) {

try {

const products = await this.$axios.$get('products', {params: { color: 'Black', pageSize: 12 }});

context.commit('setProducts', products);

} catch(err) {

console.error(err);

}

},

async fetchProductDetails(context, id) {

try {

const product = await this.$axios.$get(`products/${id}`);

context.commit('setActiveProduct', product);

} catch(err) {

console.error(err);

}

},

async fetchProductsByFilters(context, filters = {}) {

try {

const products = await this.$axios.$get('products', { params: filters });

context.commit('setProducts', products);

} catch(err) {

console.error(err);

}

},

async fetchAvailableFilters(context) {

try {

const filters = await this.$axios.$get('products/filters');

context.commit('setFilters', filters);

} catch(err) {

console.error(err);

}

}

}

export default {

namespaced: true,

state,

getters,

actions,

mutations

}

Products.vue

<template>

<div>

<section id="shop\_main\_area" class="ptb-100">

<div class="container">

<div class="row">

<div class="col-lg-6 col-md-12">

<div class="product\_filter">

<div class="customs\_selects">

<select name="product" class="customs\_sel\_box" @change="randomProduct">

<option value="Filter">Filter</option>

<option value="most\_popular">Most Popular</option>

<option value="best\_seller">Best Seller</option>

<option value="tranding">Tranding</option>

<option value="featured">Featured</option>

</select>

</div>

</div>

</div>

<div class="col-lg-6 col-md-12">

<div class="product\_shot">

<div class="product\_shot\_title">

<p>Сортувати за:</p>

</div>

<div class="customs\_selects">

<select name="product" class="customs\_sel\_box" @change="randomProduct">

<option value="popularity">За рейтингом</option>

<option value="low">За зростанням ціни</option>

<option value="high">За спаданням ціни</option>

</select>

</div>

<div class="product\_shot\_view">

<ul>

<li><nuxt-link to="/shop/shop-3"><i class="fas fa-list"></i></nuxt-link></li>

<li><nuxt-link to="/shop/shop-2"><i class="fas fa-th-large"></i></nuxt-link></li>

<li><nuxt-link to="/shop" class="active"><i class="fas fa-th"></i></nuxt-link></li>

</ul>

</div>

</div>

</div>

</div>

<div class="row">

<div class="col-lg-9">

<div class="row">

<div class="col-lg-3 col-md-4 col-sm-6 col-12" v-for="(product,index) in productslist" :key="index" v-show="setPaginate(index)">

<ProductBox1 :product="product" :index="index" @showalert="alert" @alertseconds="alert" />

</div>

<div class="col-lg-12">

<div class="product-pagination mb-0" v-if="productslist.length > this.paginate">

<nav aria-label="Page navigation">

<ul class="pagination">

<li class="page-item">

<a class="page-link" href="javascript:void(0)" @click="updatePaginate(current-1)">

<span aria-hidden="true">

<i class="fa fa-chevron-left" style="font-size:10px;" aria-hidden="true"></i>

</span>

</a>

</li>

<li class="page-item" v-for="(page\_index, index) in this.pages" :key="index" :class="{'active': page\_index == current}">

<a

class="page-link"

href="javascrip:void(0)"

@click.prevent="updatePaginate(page\_index)"

>{{ page\_index }}</a>

</li>

<li class="page-item">

<a class="page-link" href="javascript:void(0)" @click="updatePaginate(current+1)">

<span aria-hidden="true">

<i class="fa fa-chevron-right" style="font-size:10px;" aria-hidden="true"></i>

</span>

</a>

</li>

</ul>

</nav>

</div>

</div>

</div>

</div>

<ShopSidebar :available-filters="filters" :categories="categories" />

</div>

</div>

</section>

</div

</template>

<script>

import { mapState } from 'vuex'

import ProductBox1 from '~/components/product-box/ProductBox1'

import ShopSidebar from '~/components/widgets/ShopSidebar'

export default {

components: {

ProductBox1,

ShopSidebar

},

data() {

return {

title: 'Shop',

dismissCountDown: 0,

breadcrumbItems: [

{

text: 'Home',

to: '/'

},

{

text: 'Shop',

to: '/shop/shop-5'

}

],

current: 1,

paginate: 12,

paginateRange: 3,

pages: [],

paginates: '',

}

},

async fetch() {

const productID = this.$route;

try {

await this.$store.dispatch("products/fetchProductsByFilters", { pageSize: 24 });

await this.$store.dispatch("products/fetchAvailableFilters");

await this.$store.dispatch("categories/fetchCategories");

} catch (err) {

this.$router.redirect("/404");

}

},

fetchOnServer: true,

computed: {

...mapState({

shuffleproducts: state => state.products.shuffleproducts,

productslist: state => state.products.productslist,

filters: state => state.products.filters,

categories: (state) => state.categories.categories,

}),

},

mounted() {

this.getPaginate()

this.updatePaginate(1)

window.scrollTo(0, 0)

},

methods: {

alert(item) {

this.dismissCountDown = item

},

getPaginate() {

this.paginates = Math.round(this.shuffleproducts.length / this.paginate)

this.page = []

for (let i = 0; i < this.paginates; i++) {

this.pages.push(i + 1)

}

},

setPaginate(i) {

if (this.current === 1) {

return i < this.paginate

} else {

return ( i >= (this.paginate \* (this.current -1)) && i < (this.current \* this.paginate))

}

},

updatePaginate(i) {

this.current = i

let start = 0

let end = 0

if (this.current < this.paginateRange - 1) {

start = 1

end = start + this.paginateRange - 1

} else {

start = this.current - 1

end = this.current + 1

}

if (start < 1) {

start = 1

}

if (end > this.paginates) {

end = this.paginates

}

this.pages = []

for (let i = start; i <= end; i++) {

this.pages.push(i)

}

return this.pages

},

getallProduct(){

this.$store.dispatch('products/getallProduct')

}

},

head() {

return {

title: this.title,

meta: [

{

hid: 'description',

name: 'description',

content: 'Shop page'

}

]

}

}

}

</script>

// recommendation.service.py

from flask import Flask, request, jsonify

from flask\_sqlalchemy import SQLAlchemy

from sklearn.metrics.pairwise import cosine\_similarity

import pandas as pd

app = Flask(\_\_name\_\_)

db = SQLAlchemy(app)

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

field1 = db.Column(db.String(255)) # Наприклад, поле 1

field2 = db.Column(db.String(255)) # Наприклад, поле 2

class Product(db.Model):

id = db.Column(db.Integer, primary\_key=True)

product\_id = db.Column(db.Integer)

field1 = db.Column(db.String(255))

field2 = db.Column(db.String(255))

user\_query = db.session.query(User.id, User.field1, User.field2)

product\_query = db.session.query(Product.product\_id, Product.field1, Product.field2)

user\_df = pd.read\_sql(user\_query.statement, db.session.bind)

product\_df = pd.read\_sql(product\_query.statement, db.session.bind)

user\_item\_matrix = pd.pivot\_table(df, values=['field1', 'field2'], index='user\_id', columns='product\_id', fill\_value=0)

user\_similarity = cosine\_similarity(user\_item\_matrix)

@app.route('/recommendations/<int:user\_id>', methods=['GET'])

def get\_recommendations(user\_id):

try:

user\_views\_query = db.session.query(User.field1, User.field2).filter\_by(id=user\_id)

user\_views\_df = pd.read\_sql(user\_views\_query.statement, db.session.bind)

recommendations = get\_recommendations(user\_id, user\_similarity, user\_item\_matrix)

return jsonify({'user\_id': user\_id, 'user\_views': user\_views\_df.to\_dict(), 'recommendations': recommendations})

except Exception as e:

return jsonify({'error': str(e)})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

# ДОДАТОК Б РЕЗУЛЬТАТИ АПРОБАЦІЇ



