МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія проєктування рекомендаційного модуля для системи електронної комерції»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Ободяк В.К.**

Студента групи ІН.м-92 — **Кіхнєй Д.І.**

СУМИ 2020

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *Інформатика*

Затверджую:

Зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТОВІ

Кіхнєю Даниілу Ігоровичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія проєктування рекомендаційного модуля для системи електронної комерції*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____

_____

_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Інформаційний огляд. 2) Постановка задачі. 3) Вибір методів вирішення. 4) Проєктування рекомендаційного модуля. 5) Розробка веб-додатку.*

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

_____

_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної магістерської роботи	Термін виконання проекту (роботи)	Примітка
1.			
2.			
3.			
4.			

Студент – дипломник _____

(підпис)

Керівник проекту _____

(підпис)

РЕФЕРАТ

Записка: 73 стор., 19 рис., 1 додаток, 20 джерел.

Об’єкт дослідження — процес проєктування рекомендаційного модуля для системи електронної комерції.

Мета роботи — розробити технологію рекомендаційного модуля для системи електронної комерції.

Методи дослідження — в процесі дослідження були використані такі технології як HTML, CSS, Bootstrap, JavaScript, Vue.js, PHP, Laravel, Open Server, Postman, Visual Studio Code.

Результати — розроблений та стилізований макет додатку, розроблена база даних, розроблений рекомендаційний модуль, розроблений веб-додаток, протестована система.

HTML, CSS, BOOTSTRAP, JAVASCRIPT, PHP, LARAVEL,
POSTMAN, VUE.JS, OPEN SERVER

ЗМІСТ

ВСТУП

Робота присвячена розробці інтернет-магазину та інформаційної технології проєктування рекомендаційного модуля для системи електронної комерції.

Інтернет - це глобально пов’язана мережева система, що полегшує всесвітнє спілкування та доступ до ресурсів даних завдяки великій колекції приватних, державних, ділових, наукових та державних мереж.

Інтернет виник від уряду США, який почав будувати комп’ютерну мережу в 1960-х рр., Відому як ARPANET. У 1985 р. Національний науковий фонд США (NSF) замовив розробку першої версії університетської мережі, яку назвали NSFNET.

Система була замінена новими мережами, яка використовувалася комерційними постачальниками послуг Інтернету в 1995 році. Інтернет був широко представлений громадськості приблизно в цей час. На зараз Інтернет являється громадською, кооперативною і самодостатньою установою, доступною для мільйонів людей у світі.

Найчастіше використовуваною частиною Інтернету є Всесвітня павутина, яка скорочено називається "WWW" або "World Wide Web", на якій зберігаються мільйони веб-додатків. На більшості веб-додатках деякі слова та речення можуть бути відображені іншого кольору, також часто цей текст може бути підкресленим. Коли користувач Всесвітньої павутини обирає це слово або речення, відбувається перехід на веб-додаток, що має відношення до цього слова чи речення. Також часто бувають кнопки або зображення, на які прикріплене якесь посилання, по яким також можна перейти до веб-додатку або до його частини [1].

Електронна комерція - це купівля-продаж товару чи послуг через Інтернет, а також передача грошей та даних про покупця. Інтернет-купівля - це дія, в якій користувач придбає товари чи послуги через Інтернет. Цей процес проходить наступним чином: покупець заходить на веб-ресурс, потім

вибирає товар та спосіб його доставки. Покупець має можливість оплатити товар шляхом оплати кредитною карткою або готівкою. Також, електронна комерція дуже часто використовується не тільки для продажі фізичних товарів в Всесвітній павутині, вона також може описувати яку-небуть комерційну операцію, що проходить через Інтернет. Електронний бізнес стосується всіх аспектів ведення бізнесу в Інтернеті, а електронна комерція стосується конкретно операцій з товарами та послугами.

Історія електронної комерції почалася з першого онлайн-продажу: 11 серпня 1994 року чоловік виклав в своєму онлайн магазині NetMarket компакт-диск групи Sting та продав другові. Це був перший приклад того, як споживач купує товар у бізнесу через всесвітню павутину - або “електронну комерцію” [2].

З тих самих пір електронна комерція дуже сильно змінилася, а саме, пошук став більш легким для користувача, а отже придбання продуктів через інтернет-магазит стало доступніше.

Товарні рекомендації - це додаткові блоки на сайті інтернет-магазину, в яких демонструються товари, рекомендовані відвідувачеві магазином за певним алгоритмом. За даними опитувань і тестів, 45% споживачів з більшою ймовірністю зроблять покупки на сайті, якщо там є персональні рекомендації, а 56% покупців онлайн-магазинів з більш високою ймовірністю повернуться на сайт з товарними рекомендаціями.

Спираючись на дані факти, можна сміливо зробити висновок про те, що покупці люблять отримувати товарні рекомендації, засновані на їх інтересах та потребах, тому що це скорочує час на пошуки потрібної пропозиції в інтернет-магазині і полегшує процес покупки за рахунок релевантних і персоналізованих пропозицій. Але дати такі рекомендації - це не просто показати кілька схожих позицій. Важливо формувати пул товарів розумно, ґрунтуючись на зібраних про користувача даних і з огляду на максимальну кількість факторів, таких як ціна, популярність, бренд, аналоги і інше, щоб підвищити ймовірність покупки [2].

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Види рекомендаційних систем

Існує багато видів рекомендаційних систем, але основними з них є: Content-based та Item-based.

1) Основані на даних про товар (Content-based):

- користувачі отримують в рекомендаціях ті товари, або схожі, які раніше були куплені;

- рекомендації в основному отримує користувач по типу товару та іншими схожими ознаками;

- підбір товарів залежить від предметної області, отже корисність даного підходу може бути обмежена.

2) Колаборативної фільтрації (Item-based):

- підбір рекомендацій відбувається за допомогою оцінок користувача, про товар та інших користувачів;

- якісний підбір рекомендацій, потребує значну кількість набіру даних про товар та користувача;

- даний метод підбору, частіше за все, буде давати кращий результат;

- є свої проблеми (такі, як холодний старт, а саме відсутність рекомендацій, якщо користувач нічого не оцінював, або не купував).

Перші рекомендаційні системи у Всесвітній павутині з’явилися давно, а саме 20 років тому. Але дійсно помітний розвиток систем відбувся у 2009-2015 роках, на змаганнях Netflix Prize, ще тоді, коли компанія Netflix давала у прокати касети та диски з фільмами та серіалами. Для цієї компанії було дуже важливо покращити підбір рекомендацій, так як, чим краще працює система, тим більше користувачів буде користуватися даною платформою, а отже прибуток буде зростати ще скоріше. Згодом, компанія запустила змагання, подія відбулась у 2006 році, суть якого заключалося у тому, що учасникам потрібно було якомога краще передбачити оцінку, яку користувач поставить фільму або серіалу. Якість передбачення вимірювалося за

допомогою середнього-квадратичного відхилення. Компанія виклала у відкритий доступ роботи, які користувачі мали можливість оцінити за п’ятибальою шкалою. На той момент, у компанії Netflix вже був створений алгоритм, який дозволяв передбачити оцінку 0.95 за метрикою середнього-квадратичного відхилення, але перед учасниками стояло завдання покращити даний алгоритм, хоча б на 10%, що дорівнювало б 0.86. Учасникам, які б перемогли змагання, компанія обіцяла приз в один мільйон доларів. Змагання зайняло близько трьох років. У перші два роки, систему вдалося покращити приблизно на 8%, але потім, темп результативної розробки сповільнився. Наприкінці дві команди розробників прислали свої результати, з дуже незначною різницею у часі, у кожної з них були задовільні результати, а саме поріг в 10%, але команда у якої показник був гірший залишилася ні з чим. Тож, це змагання значною мірою прискорило розвиток розробки рекомендаційних систем [3].

1.2 Поняття веб-додатку

Веб-додаток - це програма, яка доступна з будь-якого браузера та яку можна використовувати через Інтернет, або через локальну мережу. За допомогою браузера будь-який користувач може отримати доступ до отримання інформації з веб-додатку та всіх функціональних можливостей. Наприклад: офіційні веб-сайти, сайти-візитки, інтернет-магазини та інші. На офіційному веб-сайті, такому як банк, надають інформацію про клієнтів та їхні рахунки, також дозволяють здійснювати різні операції в Інтернеті. Тобто веб-додаток - це програма на базі клієнт-сервері, яка працює на сервері і доступ до якої здійснюється на декількох клієнтських комп'ютерах.

Більшість веб-додатків використовують серверні мови програмування, тобто якогось функціоналу на сервері, а також для обробки і зберігання інформації, але веб-додатки можуть бути створені з використанням лише мови гіпертекстової розмітки та стилів. Клієнтські скрипти написані на мові програмування JavaScript і відображення інформації на гіпертекстовій мові HTML. Це дає можливість користувачам взаємодіяти з компанією та

використовувати онлайнові форми для купівлі товару або, наприклад, залишити відгук про товар.

Веб-додатки, а саме клієнтська частина, зазвичай програмується мовою, що підтримується браузером, наприклад, JavaScript. Каркас додатку створюється за допомогою HTML, а стилізують з CSS, в основному використовуючи бібліотеки, препроцесори та фреймворки, для більш швидкої та якісної розробки. Деякі програми, такі, як інтернет-магазин, офіційні веб-сайти, додатки де потрібно часто змінювати інформацію є динамічними, вимагаючи обробки на стороні сервера, а інші повністю являються статичними, так як, не мають обробки на стороні серверу. Веб-додатки, які є динамічними частіше за все мають базу даних, яка може бути як реляційною так і нереляційною, в якій зберігається значна частина контенту з веб-ресурсу, інформація про користувачів, та інше. Але існують динамічні веб-додатки, які використовують серверні технології, такі, як PHP, Node.js, .NET та інші, але без бази даних. Такі додатки частіше за все використовують браузер в якості бази даних, а саме в cookie, в сессиях, localStorage та sessionStorage. Додатки, які використовують бази даних: програми електронної комерції, інтернет-банкінг, соціальні мережі, інтерактивні ігри, онлайн-навчання, онлайн-бронювання, онлайн-опитування, інтернет-форуми, системи управління контентом, блоги тощо [4].

1.3 Технології, які використовуються для створення веб-додатків

Вибір відповідного стеку технологій є особливо складним для малого бізнесу та стартапів, оскільки вони, як правило, мають обмежений бюджет, а отже, їм потрібен стек технологій, який забезпечує найбільший виграш для того, щоб їхні проєкти почали працювати. Правильний стек технологій значною мірою є запорукою успіху проєкту, тоді як неправильний вибір технологій розробки веб-додатків може бути причиною невдачі.

У веб-розробці існує два типи кодування: клієнтська (верстка, стилізація, логіка анімацій та іншого) та серверна (те, як додаток буде працювати, що віддавати на клієнт з бази даних та інше). Клієнтська частина

також називається інтерфейсом. Клієнтська частина, тобто інтерфейсна веб-розробка включає все, що користувачі бачать на своїх екранах. Основні компоненти стекового інтерфейсу:

- HTML;
- CSS (Bootstrap);
- JavaScript (Vue, React, Angular).

Програмування на стороні сервера включає написання на мові програмування логіки роботи веб-додатку з базою даних. Серверна сторона невидима для користувачів, але вона постачає клієнтську сторону. Але існує проблема з вибором серверних технологій, так як від цього залежить наскільки швидко буде працювати додаток, наскільки він буде безпечним, та інше. Список найпопулярніших мов програмування та фреймворків, які використовують у написання серверної логіки додатку:

- Java (Struts, Spring);
- Python (Django, Flask);
- Scala (Finch, Play);
- Node.js – JavaScript, на якому також кодують серверну частину веб-додатку;
- PHP (Laravel, Yii, Symfony);
- Ruby (Grape, Nancy).

Потім, при розробці серверної частини йде вибір місця, для зберігання даних. Переважно це бази даних реляційні та нереляційні, але ще є можливість використовувати локальні місця для збереження інформації – у браузері. Але останій спосіб збереження інформації, підійде не для всіх задач. Список найпопулярніших баз даних:

- MySQL (реляційна);
- PostgreSQL (реляційна);
- MongoDB (нереляційна).

Також, інколи, є необхідність в використанні системи кешування. Це дозволить зменшити навантаження на базу даних і обробляти велики обсяги

трафіку. Memcached та Redis є найбільш розповсюдженими системами кешування.

Одним із останніх кроків, є вибір серверу для обробки запитів комп'ютерів розташованих по всій Всесвітній павутині. Список серверів:

- Apache;
- Nginx.

Щоб розробити веб-додаток, потрібно вибрати сервер, базу даних, мову програмування, фреймворк та інструменти для написання клієнтської частини, які розробник буде використовувати. Ці технології веб-розробки залежать одна від одної і називаються стеком технологій [4, 5].

1.4 Постановка задачі

Аналіз інформаційних ресурсів показав, що для розробки інформаційної технології проєктування рекомендаційного модуля для системи електронної комерції потрібно вирішити такі задачі, як:

1) вибрати методи рішення:
- вибрати алгоритм підбору рекомендацій;
- вибрати технології для розробки системи;
- вибрати структуру патерну;
- вибрати локальний сервер.

2) виконати програмну реалізацію:
- зверстати та стилізувати макет додатку
- спроєктувати базу даних;
- реалізувати алгоритм для системи електронної комерції;
- розробити веб-додаток;
- протестувати систему.

Метою проєкту є проєктування рекомендаційного модуля для системи електронної комерції та створення веб-додатку, а саме інтернет-магазину.

Одним із найяскравіших екземплярів електронної комерції є інтернет-магазин. Під час проєктування веб-додатку та модулю, потрібно продумати

весь можливий функціонал для клієнтів, тобто користувачів та адміністратора, який буде додавати, редагувати та видаляти інформацію.

Також веб-додаток буде мати два сценарії, такі, як адміністратора та користувача.

Можливості адміністратора:

- додати новий товар;
- змінити ціну товару;
- редагувати товар та інформацію про нього;
- видалити товар.

Можливості користувача:

- реєстрація у веб-додатку;
- збереження товару до кошику;
- обирати спосіб оплати товару;
- купити товар.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Вибір алгоритму

Даний алгоритм працює наступним чином: коли користувач взаємодіє з товаром, алгоритм отримує навчальні дані, такі як: “людина подивилася на товар з ідентифікатором 1 і 2, потім додала до кошику товар з ідентифікатором 3“. Алгоритм потім це все зчитує і буде завжди рекомендувати товари близькі номеру 3 після переглядів товарів типу 1 і 2.

Цей підхід добре працює, коли і товарів і покупців існує велика кількість, в цьому випадку для кожного товару є історія переглядів і покупок, і алгоритму вистачає даних для підбору рекомендацій [3].

2.2 Вибір технологій для розробки веб-додатку

У веб-додатках використовують різні стеки технологій, вибір залежить від цілі задачі та методу її рішення.

Середовище розробки

У світі веб-розробки існує велика кількість різних текстових редакторів. При виборі редактору, потрібно переглянути усі можливі варіанти, а потім вже обрати, так як, від вибору технології, залежить на скільки швидко та правильно розробник буде писати свій код. Найкращим редактором коду повинен бути той, який підвищує продуктивність розробника. Він повинен бути потужним, але простим у використанні. Більше того, він повинен мати підтримку тої мови програмування, на якій розробник збирається розробляти проєкт.

Список найкращих текстових редакторів для розробки веб-додатку:

- Visual Studio Code – найпопулярніший для розробника, який працює з операційною системою від Microsoft Windows;

- Atom – є зв’язок з системою управління версій GitHub;

- Sublime Text Editor – також один з найкращих виборів, так як, потребую малу кількість оперативної пам’яті та не сильно навантажує процесор комп’ютера;

- Notepad++ - потребує найменшу кількість ресурсів комп’ютера;

- Brackets Text Editor – багатоплатформовий текстовий редактор.

Під час розробки було обрано у якості тектового редактору – Visual Studio Code, тому що, це програмне забезпечення має доволі сильну підтримку, з боку розробників цього ПО, а отже має підтримку великої кількості різних мов програмування та бібліотек, які полегшують написання коду.

Visual Studio Code, або скорочено VSC – це швидкий, легкий і потужний редактор коду. Корпорація Microsoft розробила VSC як крос-платформний редактор коду для написання веб-додатків. Редактор вийшов 29 квітня 2015 року і компанія, яка представила дане ПО, була Microsoft на конференції 2015 року, яка проходила в Сан-Франциско. Кілька місяців потому, 18 листопада 2015 року, VSC був випущений під ліцензією MIT, а вихідний код був доступний на GitHub. 14 квітня 2016 року VSC був випущений в Інтернеті.

На рис. 2.1 наведений відкритий редактор коду, в якому яскраво підсвічується код, також встановлений плагін для зміни файлів та папок, для кращого розуміння про знаходження файлів.

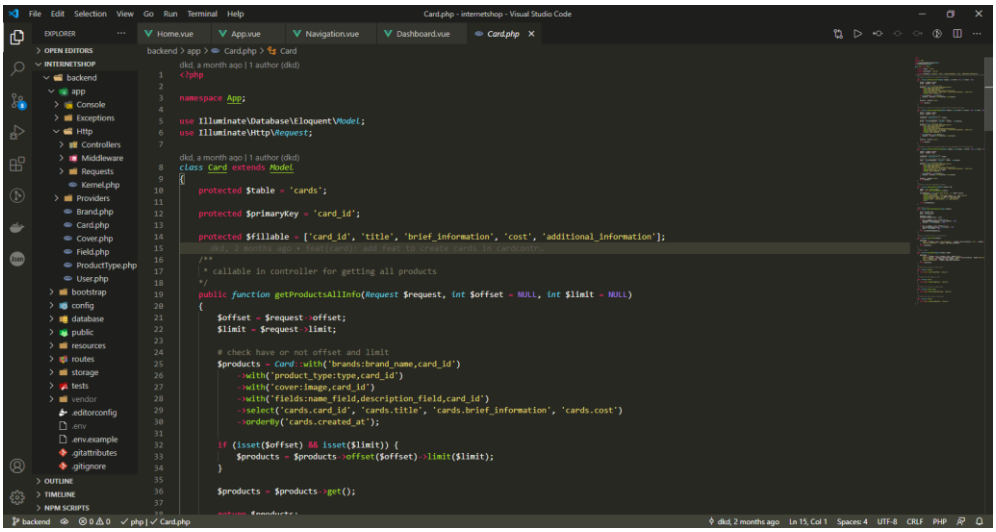

Рисунок 2.1 – Відкритий проєкт у редакторі коду Visual Studio Code

Цей редактор коду, розроблений для Windows, Linux та macOS. Крім того, VSC також поєднує в собі простоту редактора та потужні інструменти розробника, такі як налагодження, вбудований контроль Git, підсвічування синтаксису та багато іншого. Він безкоштовний і з відкритим кодом, а також базується на фреймворці Electron, який використовується для розгортання програм Node.js [6].

Часто Visual Studio Code плутають з Visual Studio 2015, який є абсолютно іншим продуктом. Різниці цих редакторів коду:

- VSC - це редактор вихідного коду, тоді як Visual Studio - повноцінна IDE;
- VSC - це багатоплатформове ПО, яке працює на Windows, Linux та macOS, тоді як Visual Studio працює лише на Windows та macOS;
- VSC швидкий і легкий, а Visual Studio 2015 не такий швидкий, але вміщує в собі більш сильну середовище розробки;
- VSC в основному використовується розробниками клієнтської частини додатку, а Visual Studio - розробниками будь-якого типу;
- VSC базується на файлах і папках, а Visual Studio 2015 - на проєктах;
- обидва вони також різняться, коли йдеться про підтримку мов програмування. Оскільки VSC може використовуватися для написання багатьох мов, таких як PHP, Python, HTML5, JavaScript та багатьох інших, тоді як Visual Studio в основному використовується для .NET, але підтримує також деякі інші мови [6].

Мова гіпертекстової розмітки (HTML)

HTML - мова гіпертекстової розмітки, є каркасом більшості веб-сайтів у Всесвітній павутині. HTML відповідає за відображення контенту у веб-додатках та їх структуру. Технології, які також використовують при розробці клієнтської частини CSS – відповідає за стилізацію веб-сторінок, JavaScript – який відповідає за функціонал та логіку додатку на клієнтській сторогні.

Гіпертекстової розмітки, тому що, має зв’язкі між посиланнями. Посилання – основний спосіб пересування по просторах Всесвітньої

павутини. Мова гіпертекстової розмітки працює наступним чином: файл, який містить HTML-код розпочинається з тегу <!DOCTYPE html>, що дає можливість веб-браузеру зрозуміти, що файл містить гіпертектову розмітку, потім йде тег <head>, в якому йде підкючення стилів та скриптів. Також в середені даного тегу є тег <title>, який містить в собі заголовок файлу, потім тег закривається</title>. Існують тегі які потрібно закривати, для того щоб уникнути некоректної розмітки, але є і ті, які не потрібно закривати, наприклад <img>, даний тег мість в собі сображення. Після закриття тегу </head>, йде наступний обов'язковий тег <body>, в якому описується саме тіло сторінки (блоки, таблиці, посилання, зображення та інше). Приклади тегів всередені тіла сторінки: <p> - параграф, <img> - зображення, - блок, <a> - посилання та інші. В кінці сторінки йдуть закриваючі теги </body> та </html>. На рис. 2.2 наведений приклад використання тегів та контенту між ними.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      <p>текст</p>
10 </body>
11 </html>
```

Рисунок 2.2 – Початкова структура HTML

Назва елементу всередині тегу не враховує регістр. Тобто, можна писати великими, малими літерами або сумішшю. Наприклад: , <Div>, <DIV>, або іншим способом [7, 8, 9].

Каскадні таблиці стилів (CSS)

CSS – це каскадні таблиці стилів, є простою мовою стилізації, дизайну, призначена для розробки веб-сторінок більш презентабельними. CSS розроблений для того, щоб сторінка виглядала більш презентабельно, після

верстки. За допомогою CSS можна: змінювати колір блоків і таблиць, можна змінювати шрифт тексту, його розмір та колір, також можна сторювати різного типу анімації і багато іншого. Частіше за все CSS використовується разом з HTML та XHTML. Завдяки HTML, також, можна за допомогою атрибутів стилізувати сторінку, але це дуже застаріла практика, на даному етапі веб-розвитку. Приклади переваги стилізації сторінки за допомогою CSS над HTML:

- якщо стилізувати за допомогою CSS - це може сильно зекономити час, так як є можливість, один раз створити клас, та потім його використовувати повторно;

- веб-сторінки будуть завантажуватися набагато швидше, ніж якщо стилізувати через HTML-атрибути, так як, непотрібно писати зайвий код, досить лише додати клас, а це означає, що завантаження буде швидше;

- легко додавати нові стилі, треба лише додати нове значення і при повторному використанні класу, це значення додасться до всіх тегів з цим класом;

- сумісність з різними пристроями. За допомогою CSS можна стилізувати веб-додаток одночасно для декількох пристроїв, наприклад, телефон та комп'ютер і сторінка буде відображатися коректно.

На рис. 2.3 наведений приклад використання CSS з селектором slider-preHeader-left-img та властивостями max-width та іншими.

```css
.slider-preHeader-left-img{
    max-width: 700px;
    max-height: 700px;
    display: flex;
    justify-content: center;

}

.slider-preHeader-left-img img{
    width: auto;
    height: auto;
    max-width: 100%;
    max-height: 100%;
    margin:0 auto;
    box-shadow: 0.968px 2.881px 18.52px 2px rgba(0, 0, 0, 0.1);
}
```

Рисунок 2.3 – Приклад використання CSS

CSS-стилі складаються з селектору та властивісю зі значенням. В одній властивості може бути одне і більше значень. В CSS всі правила починаються з селектора, який вказує на те, яке ім’я буде використано в HTML-документі при додаванні до тегу класу. У блоці, який відповідає за значення властивості встановлюють: розмій, грані, колір, розмір шрифту та інше [10].

Мова програмування – JavaScript

JavaScript являється одною з найпопулярніших скриптових мов програмування у світі.

JavaScript - це надзвичайно популярна інтерпретована скриптова мова, яка на початку 2019 року стала мовою програмування, яку найбільш часто вивчають розробники. JavaScript - це відкритий стандарт, який не контролюється жодним постачальником, з численними реалізаціями та простим у засвоєнні синтаксисом, що робить його популярним серед початківців та ветеранів-розробників.

JavaScript використовується майже з перших днів створення Всесвітньої павутини. Мова вперше була розповсюджена як спосіб додавати візуальний функціонал на стороні клієнта до веб-сторінок і сьогодні широко використовується для цієї мети. Практично будь-що інтерактивне чи анімоване на веб-сторінці сьогодні відображається за допомогою JavaScript, включаючи різні інтернет-реклами та метрики. Але JavaScript може працювати не тільки у браузері. Завдяки фреймворкам розробки, таким як Node.js, JavaScript тепер використовується для написання коду практично для будь-якої ніші, яку розробник може придумати, від клієнтської частини до серверної.

Як випливає з назви, JavaScript є скриптовою мовою. Традиційні мови, такі як C ++, компілюються до того, як вони потрапляють у виконувану двійкову форму, а компілятор перевіряє наявність помилок у всій програмі до завершення процесу, а скриптові мови, навпаки, виконуються по одному рядку за допомогою іншої програми, яка називається інтерпретатором.

Також JavaScript має великий вибір різних бібліотек та фреймворків, які допоможуть у розробці, наприклад: jQuery, Vue.js, React.js, Angular.js та інші.

Також зростання популярності Node.js за останні кілька років, дало можливість розробникам створювати серверну частину додатку, що є дуже зручно, так як, непотрібно поглиблено вивчати інши мови програмування [11].

CSS Фреймворк - Bootstrap

CSS – фреймворки потрібні для того, щоб спростити розробку, а саме розробку клієнтської частини, таку як верстку, стилізацію сторінок.

Список плюсів CSS – фреймворків:

- кросбраузерність, тобто набір компонентів, який буде працювати майже на всіх браузерах;
- можливість стилізувати та розробляти веб-сторінки, навіть якщо у розробника клієнтської частини не надто багато досвіду;
- суттєве збільшення швидкості у розробці додатку.

Мінусів:

- неможливість відійти від стилю розробки, який задає фреймворк чи бібліотека;
- наявність зайвого коду.

Завдяки дуже швидкого розвитку веб-розробки існує багато CSS фреймворків та бібліотек, наприклад: Bootstrap, Foundation, Pure, UI kit, Milligram, Skeleton, Tailwind CSS, Dead Simple Grid та інші.

При розробці клієнтської частини веб-додатку було обрано саме Bootstrap, так як постійно розширюється, а отже фреймворк завжди росте, має дуже детальну документацію та має багото прихильників, а отже завжди можна знайти відповідь на якесь питання.

Bootstrap - це безкоштовний CSS фреймворк з відкритим кодом, спрямований на адаптивну, первинну веб-розробку для мобільних пристроїв.

Він містить шаблони дизайну на основі CSS та у випадку необхідності JavaScript для форм, кнопок, навігації та інших компонентів інтерфейсу.

З моменту появи в 2011 році Bootstrap миттєво завоював визнання веб-дизайнерів та розробників за те, наскільки зручний у використанні і як легко з ним працювати. Також, Boostrap має сумісність з браузером, тобто є можливість швидко, повторно використовувати готові компоненти, а ще фрейморк має вбудовану підтримку jQuery. Bootstrap можна використовувати з IDE або редактором на вибір розробника та може бути використаним з мовами на стороні сервера, починаючи від ASP.NET і закінчуючи PHP або навіть Ruby [12, 13].

Серверна мова програмування

Постійно зростаючий перелік мов програмування може ускладнити програмістам та розробникам у виборі мови, яка найбільше підходить для проєкту. Існує дуже велика кількість мов програмування, на яких можна написати серверну частину додатку, такі як: PHP, Perl, C#, Python, Java, Node.js, Ruby та інші. За великим рахунком для багатьох проєктів не важливо яку мову обрати, але є і виключення, наприклад, якщо проєкт дуже великий, з декількома базами даних, то доречно буде обрати Java, так як мова має добру підтримку ООП, статичну типізацію та інше. А наприклад, PHP підійде для середніх проєктів, тому що, на цій мові програмування можна доволі швидко писати код, також мова має динамічну типізацію, а отже помилок при інтерпретації буде значно менше.

При написанні серверної частини проєкту було обрано скриптову мову програмування PHP, тому що, на ній можна дуже швидко писати серверну частину, мова має велику кількість різних фреймворків та бібліотек, що дає можливість не писати рутинний код, в неї доволі велика підтримка від співтовариства, що також дає можливість більш швидко знаходити потрібну інформацію.

PHP - це мова програмування та сценаріїв для створення динамічних інтерактивних веб-сайтів. Таке програмне забезпечення, як WordPress

написане з використанням PHP як мови сценаріїв. Як і WordPress, PHP також має відкритий код. PHP - це мова програмування на стороні сервера. Коли користувач виконує запит на веб-сторінку, яка містить PHP-код, код обробляється модулем PHP, встановленим на цьому веб-сервері. Потім PHP генерує відповідь на веб-сторінку і за допомогою HTML відображає на екран користувача браузеру [14].

Вибір фреймворку для розробки серверної частини

Розробка програмного забезпечення з нуля вимагає багато рутинної роботи, як створення архітекнури проєкту, написання стандартних класів та у багатьох випадках необхідно буде створити методи, які вже були написані раніше, що є не дуже ефективно. Саме тому, фреймворки створені для того, щоб прибрати рутинні проблеми розробки програмного забезпечення.

У скриптовій мові програмування PHP дуже багато різних фреймворків, наприклад: Laravel, CodeIgniter, Symfony, Zend, Phalcon, CakePHP, Yii, FuelPHP та інші.

Серед з усіх PHP – фреймворків був вибір з Laravel, Yii та Symfony, так як, це найпопулярніші фреймворки, а отже, у них є своя велика аудиторія та багато інформації у всесвітній павутині.

Функціональність фреймворків збільшується за допомогою різних розширень або бібліотек. За цим критерієм лідером являється Laravel. У каталозі Packalyst можна знайти близько 9000 пакетів для Laravel. Yii2 і Symfony можуть похвалитися 2800 і 2830 розширеннями відповідно. Також у Laravel є за-замовчуванням підтримка юніт-тестів, що дає можливість швидко і якісно тестувати функціонал усієї серверної частини.

У ході пошуку інформації про всі найвідоміші PHP – фреймворки був обраний саме Laravel.

Laravel – це PHP фреймворк, це готова структура, яка містить в собі набір компонентів, таких як, міграції, моделі, контролери, роути та інших, для більш швидкої та правильної розробки серверної частини додатку.

Список функціоналу, який робить цей фреймворк одним з найкращих:

- можливість скористатися готовим API для реєстрації;
- блейд-шаблони для взаємодії серверної та клієнтської частини;
- великий архів бібліотеки;
- підтримка багатьох різних типів файлів;
- безпечність у експлуатації готового проєкту;
- дуже зручний інтерфейс міграцій до бази даних.

Також веб-розробка на Laravel має широку екосистему, яка має можливість на швидкий хостинг, а ще систему розгортання [15].

Фреймворк для розробки клієнтської частини

Під час написання логіки клієнтської частини додатку було обрано мову програмування – JavaScript, а отже вибір з усіх створених фреймворків на цю мову був дуже великий. Найпопулярніші JS-фреймворки: Vue, React, Angular.

При розробці клієнтської частини веб-додатку був обраний фреймворк Vue.js тому, що містить в собі багато зручних бібліотек та незважаючи на те, що він новий, має велику швидкість росту, що пришвидшує написання функціоналу. Vue.js був створений нещодавно, але вже використовується у всьому світі. Даний JS-фреймворк має велику кількість різних додаткових інструментів для побудови якісної, безпечної клієнтської частини веб-додатку.

Історія Vue.js починається в 2013 році, коли Еван Ю працював у Google, створюючи безліч прототипів прямо в браузері. Для цього Еван використав зручні практики з інших фреймворків, з якими працював, і офіційно випустив Vue.js у 2014 році.

Vue.js - це прогресивний фреймворк для JavaScript, який використовується для побудови веб-інтерфейсів та односторінкових додатків. Не тільки для веб-інтерфейсів, Vue.js також використовується як для розробки настільних додатків, так і для мобільних додатків з фреймворком Electron. Розширення HTML і базові можливості JS, швидко зробили Vue

улюбленим інтерфейсним інструментом, про що свідчать прийняття таких гігантів, як Adobe, Behance, Alibaba, Gitlab та Xiaomi [19].

Вибір Системи Управління Базами Даних

Вибір системи управління базами даних був дуже важливим, так як від цього залежить наскільки швидкі будуть запроси до бази даних і наскільки швидко буде приходити відповідь.

Найпопулярнішими СУБД є: Oracle, MySQL, Microsoft SQL Server, PostgreSQL, MongoDB, DB2, Microsoft Access , Redis.

Обраною була саме MySQL тому, що вона є кросплатформовою, є можливість працювати з безкоштовнім сервером, а потім перейти на комерційну версію. Також ця система управління базами даних використовує стандартну формулу SQL. Утиліти для проєктування таблиць мають інтуїтивно зрозумілий інтерфейс, швидкість запитів та відповіді дуже велика, також має дуже сильні інструменти для роботи з безпекою додатку.

MySQL - це одна з найпопулярніших систем управління базами даних, створена в 1995 році. Дана система управління реляційними базами даних з відкритим кодом. MySQL - це безкоштовна система баз даних. Однак є також кілька платних видань, за допомогою яких користувач, даної СУБД, може використовувати розширений функціонал. MySQL простий у використанні порівняно з іншим програмними забезпеченням для баз даних, таким як Microsoft SQL Server, Oracle тощо. Вона може використовуватися з будь-якою мовою програмування, але в основному використовується з PHP. MySQL може працювати на декількох платформах, таких як Linux, Windows, Unix. Також є можливість встановити її на локальній системі або навіть на сервері. Дана база даних гнучка, масштабована, швидка та надійна [16].

2.3 Вибір структури патерну

Шаблон проєктування - це оптимізоване багаторазове рішення проблем у програмуванні, з якими розробники стикаються щодня. Шаблон проєктування - це не клас і не бібліотека, яку розробники програмного забезпечення можуть просто підключити до системи, це набагато більше. Це

шаблон, який повинен вирішувати схожі проблеми, тож це спосіб вирішення однакових задач при проєктуванні програмного забезпечення.

При розробці проєкту було принято рішення використовувати патерн MVC.

MVC розшифровується як Model-View-Controller. Це архітектура або шаблон проєктування програмного забезпечення, що полегшує створення великих додатків. Він не належить до конкретної мови програмування або фреймворку, але це концепція, яку розробник може використовувати при створенні будь-якого виду додатків чи програмного забезпечення на будь-якій мові програмування. Наприклад, якщо розробник розробляє додаток на PHP, він має можливість використовувати такі фреймворки, як Laravel або Codeigniter, які використовують архітектуру MVC, щоб допомогти йому швидко і просто розробляти програми.

На рис. 2.4 наведена структура MVC.

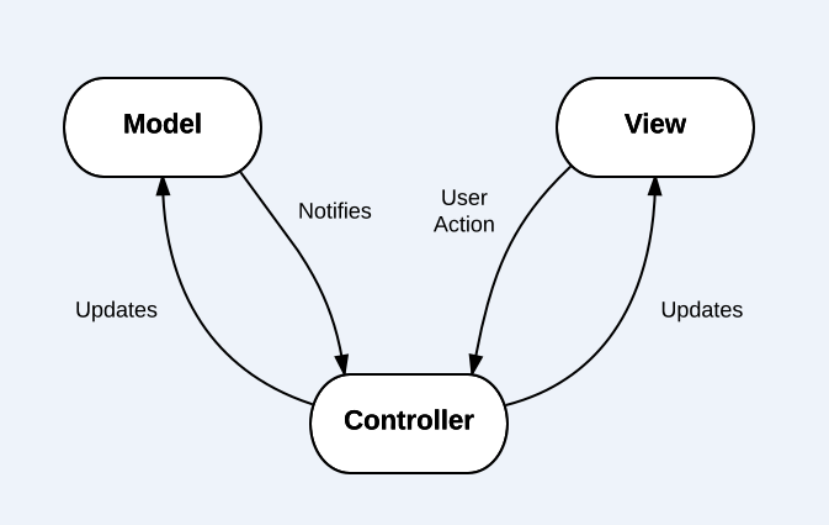

Рисунок 2.4 – Зв’язки між моделлю, контролером та видом [17]

Дана структура має три різних частини – це модель, вид та контролер. Модель працює з базою даних і утворює інтерфейс між користувачем через контролер та видом, тобто HTML – сторінкою. В контролері знаходиться логіка додатку, тобто отримання з бази даних інформації та відправлення її

на клієнтську частину і навпаки, з виду до контролеру, а контролер змінює інформацію у базі даних через модель [17].

2.4 Вибір та установка локального серверу

Перед тим, як розробляти серверну частину веб-додатку, треба обрати та встановити локальний сервер. Локальний сервер – це додаток, який дозволяє розробникам програмного забезпечення розробляти додатки на своєму комп'ютері і при цьому не залежати від доступу до інтернету.

Потреба в такому додатку виникає при розробці динамічних сайтів, тобто тих, в яких використовуються серверні мови програмування, такі як PHP, Node.js, Perl. При розробці і тестуванні статичних додатків, потреби в локальному сервері немає.

При розробці веб-додатку з серед усіх локальних серверів, було обрано саме Open Server, так як в нього дуже зручний інтерфейс, багато додатків для зручної веб-розробки, зручна консоль, можливість користуватися інтерфесом phpMyAdmin, можливість користуватися базою даних [18].

На рис. 2.4 наведена консоль користувача для керування, створення та інсталяції додатків.

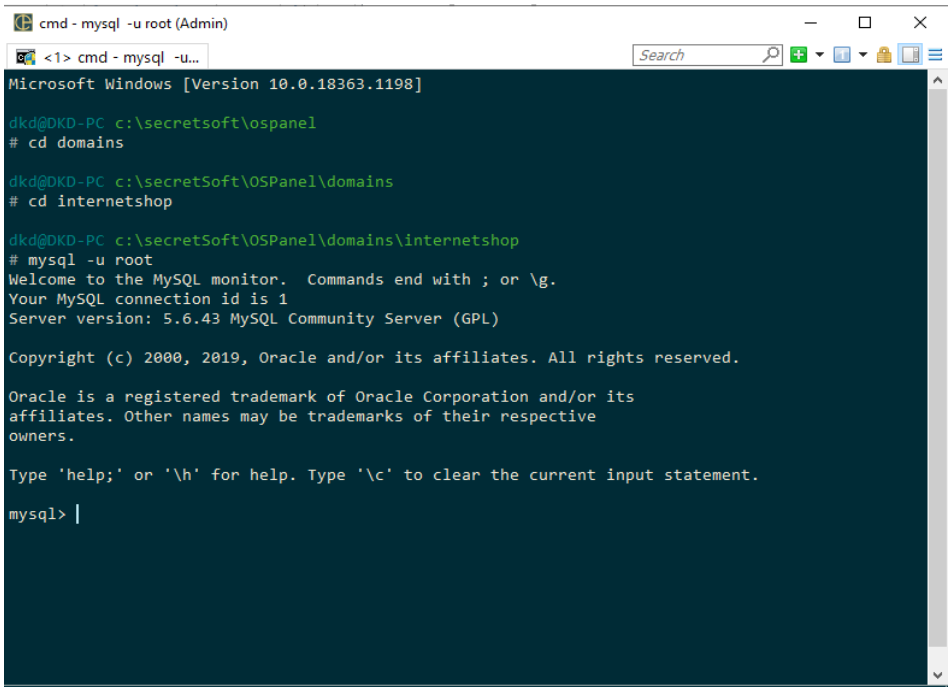

Рисунок 2.4 – Консоль користування з OpenServer

На рис. 2.5 наведені налаштування додатком та його функціонал.

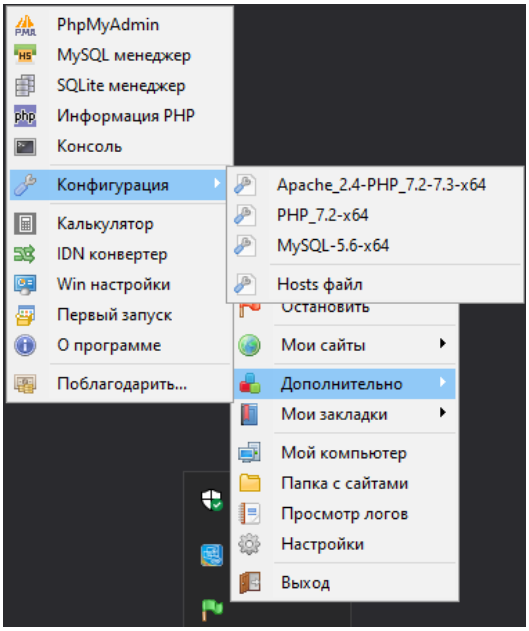

Рисунок 2.5 – Функціонал додатку

Вибраний додаток містить у собі такій корисний функціонал, як локальний сервер, інтерпретатор мови програмування, СУБД, що дозволить розгорнути серверну частину додатку на робочому комп’ютері, писати код на серверній мові програмування та користуватися базою даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проєктування бази даних

При розробці роботи було вирішено використовувати реляційну базу даних, а саме MySQL.

	Таблица	Действие	Строки	Тип	Сравнение	Размер	Фрагментировано
☐	brands		11	InnoDB	utf8mb4_unicode_ci	32 КиБ	-
☐	cards		11	InnoDB	utf8mb4_unicode_ci	16 КиБ	-
☐	cover		11	InnoDB	utf8mb4_unicode_ci	32 КиБ	-
☐	failed_jobs		0	InnoDB	utf8mb4_unicode_ci	16 КиБ	-
☐	fields		12	InnoDB	utf8mb4_unicode_ci	32 КиБ	-
☐	migrations		13	InnoDB	utf8mb4_unicode_ci	16 КиБ	-
☐	oauth_access_tokens		24	InnoDB	utf8mb4_unicode_ci	32 КиБ	-
☐	oauth_auth_codes		0	InnoDB	utf8mb4_unicode_ci	16 КиБ	-
☐	oauth_clients		2	InnoDB	utf8mb4_unicode_ci	32 КиБ	-
☐	oauth_personal_access_clients		1	InnoDB	utf8mb4_unicode_ci	16 КиБ	-
☐	oauth_refresh_tokens		0	InnoDB	utf8mb4_unicode_ci	16 КиБ	-
☐	password_resets		0	InnoDB	utf8mb4_unicode_ci	16 КиБ	-
☐	product_type		11	InnoDB	utf8mb4_unicode_ci	32 КиБ	-
☐	users		3	InnoDB	utf8mb4_unicode_ci	32 КиБ	-

Рисунок 3.1 – Структура бази даних

Таблиця brands має наступні поля:

- brand_id – унікальний ідентифікатор;
- brand_name – назва бренду;
- card_id – товар, з яким у таблиць brands та cards є зв’язки.

Таблиця cards має наступні поля:

- card_id – унікальний ідентифікатор;
- title – назва товару;
- brief_information – стисла інформація про товар;
- cost – ціна товару;
- additional_information – додаткова інформація.

Таблиця cover має наступні поля:

- cover_id – унікальний ідентифікатор;
- image – посилання в бд на зображення, яке зберігається на сервері;
- card_id – зв’язок з товарами.

Таблиця fields має наступні поля (таблиця створена для забезпечення адаптивності полів, незалежно від типу товару):

- field_id – унікальний ідентифікатор;
- name_field – назва поля;
- description_field – опис поля;
- card_id – зв’язок з товарами.

Таблиця product_type має наступні поля:

- product_type_id – унікальний ідентифікатор;
- type – тип товару;
- card_id – зв’язок з товаром.

На рис. 3.2 наведена erd – діаграма. Побудова діаграми для того, щоб забезпечити більш швидкі та вірні запити до бази даних.

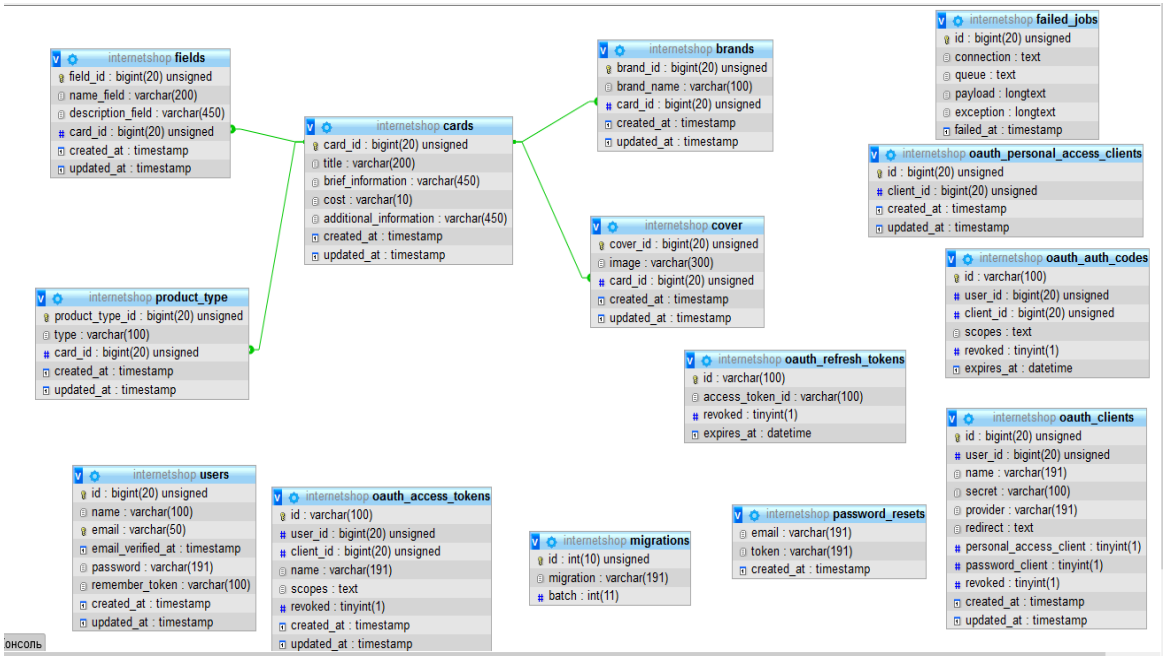

Рисунок 3.2 – Проєктування erd - діаграми

3.2 Реалізація алгоритму для системи електронної комерції

У даній роботі було розроблено 2 алгоритми для підбору рекомендацій товарів у веб-додатку. Перший алгоритм розроблений для користувачів, які будуть використовувати веб-додаток у якості простого інтернет-магазину, тобто зайдуть до додатку та одразу виберуть товар до кошику. В такому

випадку, наступні рекомендації товарів, будуть видані в залежності від ціни та типу товару.

Реалізація алгоритму підбору по-ціні:

```
public function getArrayRecommendCosts(object $product)
    {
        # get cost from model
        $cost = $product->cost;
        $cost = intval($cost);

        $copyCost = $cost;
        $arrayWithRecommendations[] = $cost;

        for ($i = 0, $count = 2; $i < $count; $i++) {
            $arrayWithRecommendations[] = intval($copyCost) - 1000;
            $copyCost = intval($copyCost) - 1000;
        }

        for ($i = 0, $count = 2; $i < $count; $i++) {
            $arrayWithRecommendations[] = intval($cost) + 1000;
            $cost = intval($cost) + 1000;
        }
        return $arrayWithRecommendations;
    }
```

Цей метод дозволяє підбирати товар схожий за ціною, а це сам метод для запиту для підбору рекомендації:

```
public function getRecommendation(Request $request, $id)
    {
        $model = Card::findOrFail($id);
        $cost = $this->getArrayRecommendCosts($model);

        $recommendations = Card::where('cards.card_id', '!=',
$model->card_id)
            ->whereIn('cost', $cost)
            ->leftJoin('product_type',    'product_type.card_id',    '=',
'cards.card_id')
            ->where('product_type.type', '=',
$model->product_type->type)
            ->leftJoin('brands', 'brands.card_id', '=', 'cards.card_id')
            ->leftJoin('cover', 'cover.card_id', '=', 'cards.card_id')
            ->get();
```

```
            return $recommendations;
        }
```

Метод отримує запит з клієнтської сторони, а потім підбирає рекомендації за заданою ціною та типом товару.

Наступний алгоритм створений для того, щоб підбирати рекомендації після додання користувачем товару до кошику. Наприклад користувач оглянув товар з id = 1, потім товар з id = 2 і після цього додав до кошику товар з id = 3. Вся ця інформація про додання товару до кошику і перегляд попередніх трьох товарів буде зберігатися у браузері, а саме в localStorage. Після повторного перегляду користувачем товарів типу id = 1 та id = 2, рекомендації користувачу буде автоматично змінені, на рекомендації близькі типу id = 3. Цей цикл підбору рекомендацій буде продовжуватись до 10 разів, потім перша сутність цього процесу буде видалена. Реалізація збереження даних на клієнтській стороні:

```
function getCart (){
  return JSON.parse(localStorage.getItem('cart'));
}
function setCart (o){
  localStorage.setItem('cart', JSON.stringify(o));
  return false;
}
function addToCart(e){
  this.disabled = true;
  var cartData = getCart (),
      parentBox = this.parentNode
      card_id = this.getAttribute('data-id'), // id товару
      title = parentBox.querySelector('.title').innerHTML,
      cost = parentBox.querySelector('.cost ').innerHTML;
  if(cartData.hasOwnProperty(card_id)){  //  якщо  товар  вже  існує,  то
додаємо ще 1 товар
    cartData[card_id][2] += 1;
  } else { // якщо товару в кошику немає, то додаємо
    cartData[card_id] = [title, cost, 1];
  }
  if(!setCart (cartData)){ // оновлюємо LocalStorage
    this.disabled = false;
  }
 return false; }
```

3.3 Розробка веб-додатку

Для розробки серверної частини веб-додатку було використано сучасний, швидкий PHP – фреймворк Laravel. Клієнтська частина була виконана за допомогою Laravel – шаблонізатора та нативного JavaScript.

На рис. 3.3 наведено початкову сторінку веб-додатку з можливістю авторизації/реєстрації користувача, додавання до кошику товарів та іншими опціями.

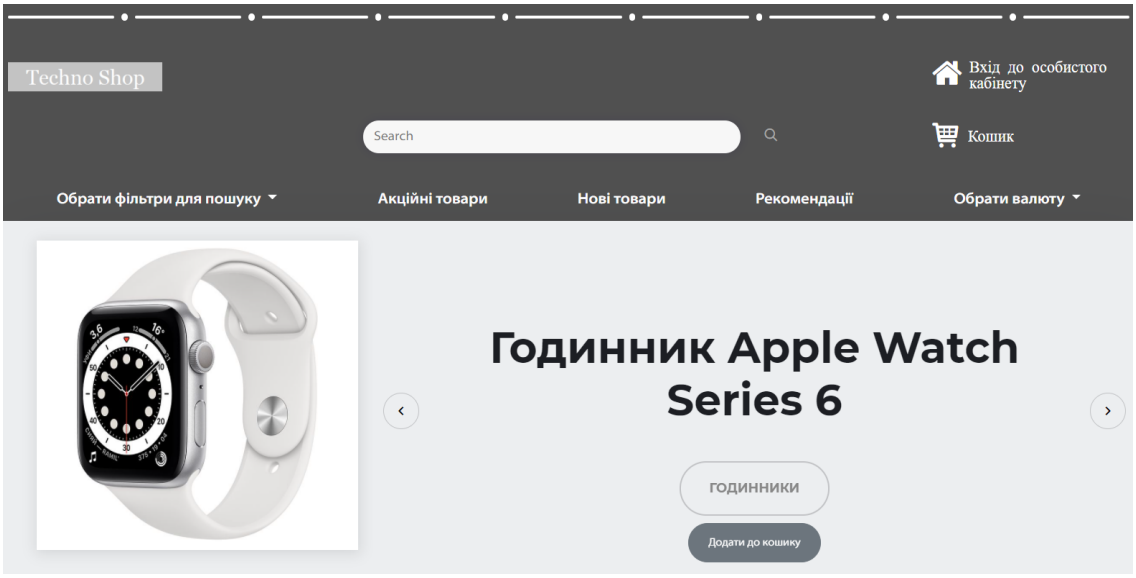

Рисунок 3.3 – Початкова сторінка

На рис. 3.4 наведена опція фільтрації пошуку товарів у веб-додатку.

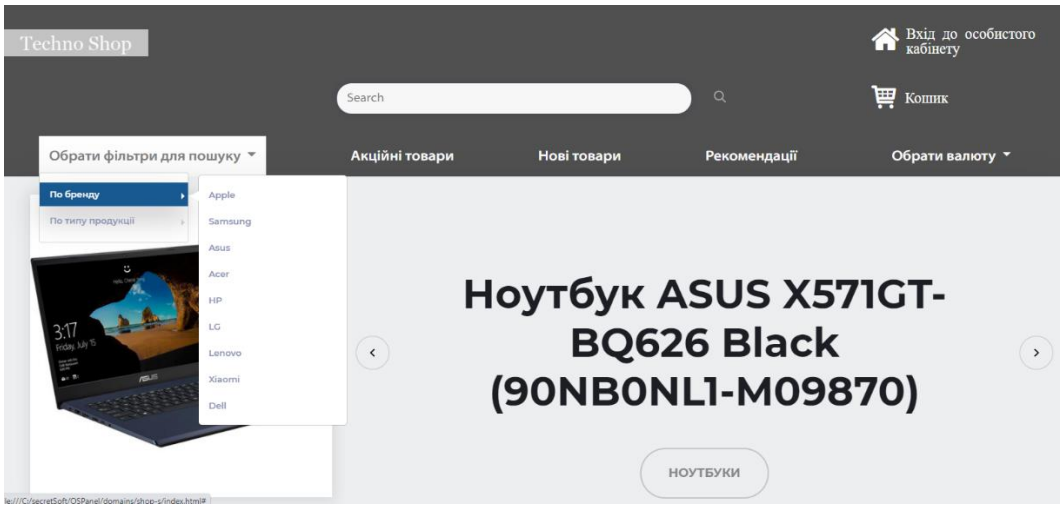

Рисунок 3.4 – Вибір фільтрації товарів для більш швидкого пошуку

На наступному рисунку наведені рекомендації товарів після перегляду двух товарів, а саме товару типу (смартфон та ноутбук), додання до кошику третього товару (годинник) та після перегляду двух товарів схожих типів (смартфону та ноутбуку), модуль рекомендації вже пропонує переглянути товари схожі на третій (годинник від компанії доданої до кошику раніше).

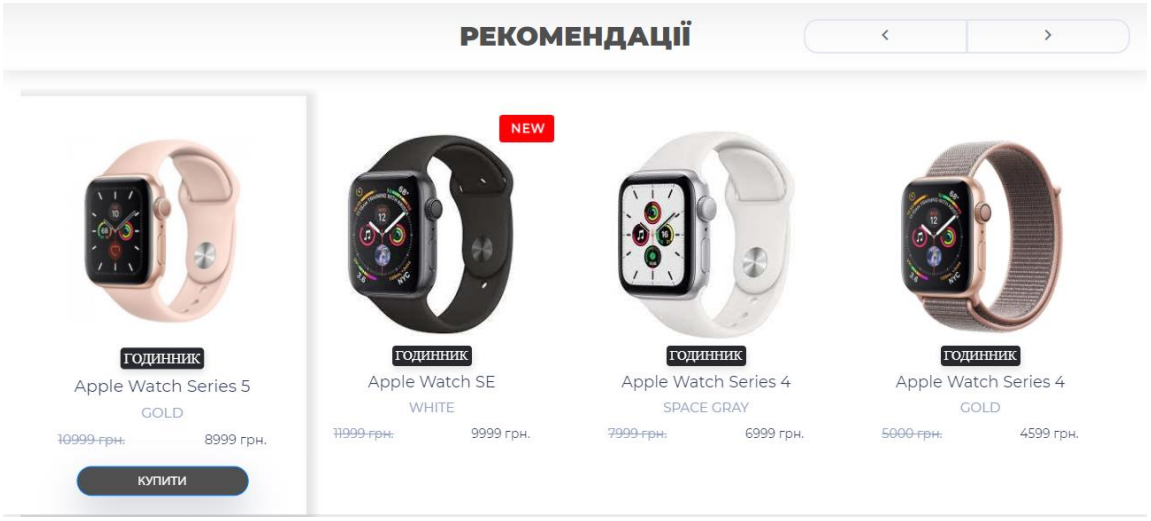

Рисунок 3.5 – Рекомендації товарів після додання до кошику

На рис. 3.6 наведені рекомендації товарів для користувача, який щойно зайшов до веб-додатку.

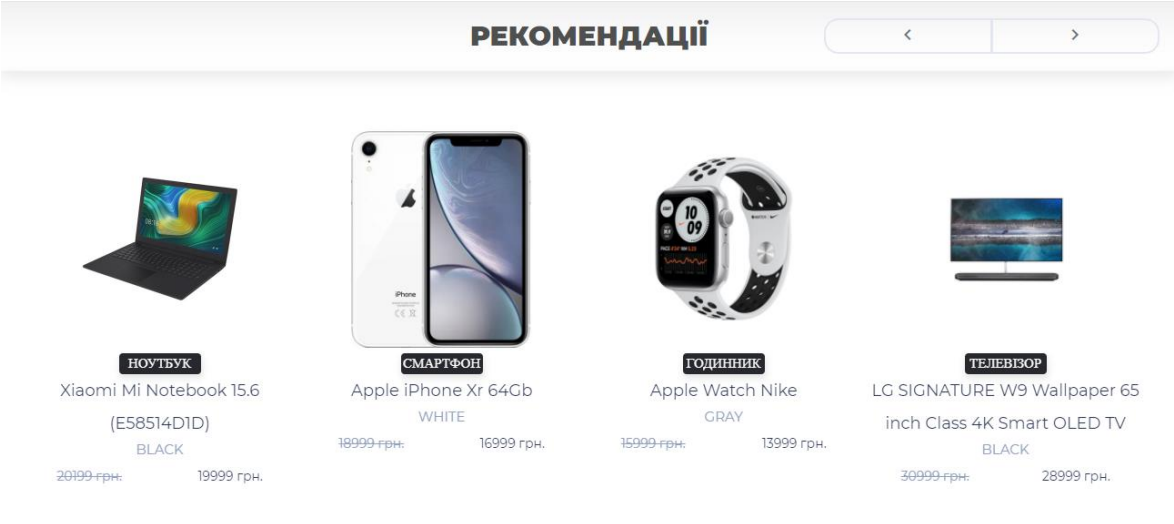

Рисунок 3.6 – Рекомендації для початкового користувача

Також у веб-додатку було реалізовано функціонал для щойно доданих товарів до магазину, товари зі знижкою та спеціальні рекомендації.

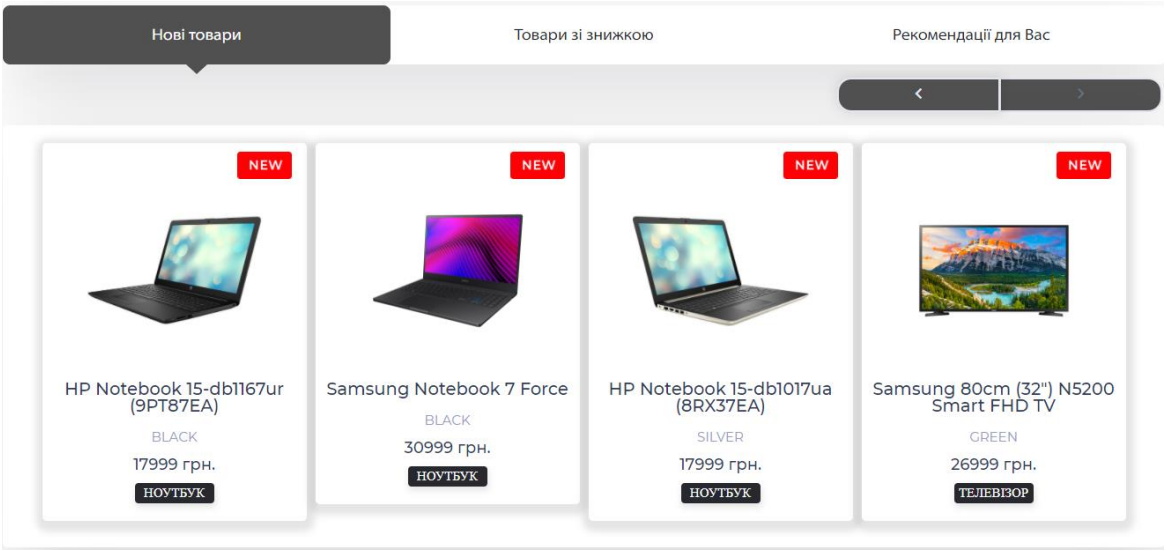

Рисунок 3.7 – Сторінка з відображенням нових товарів

В нижній частині додатку розташований футер з відображенням компаній, які співпрацюють з даним інтернет-магазином та додатковою інформацією для користувачів.

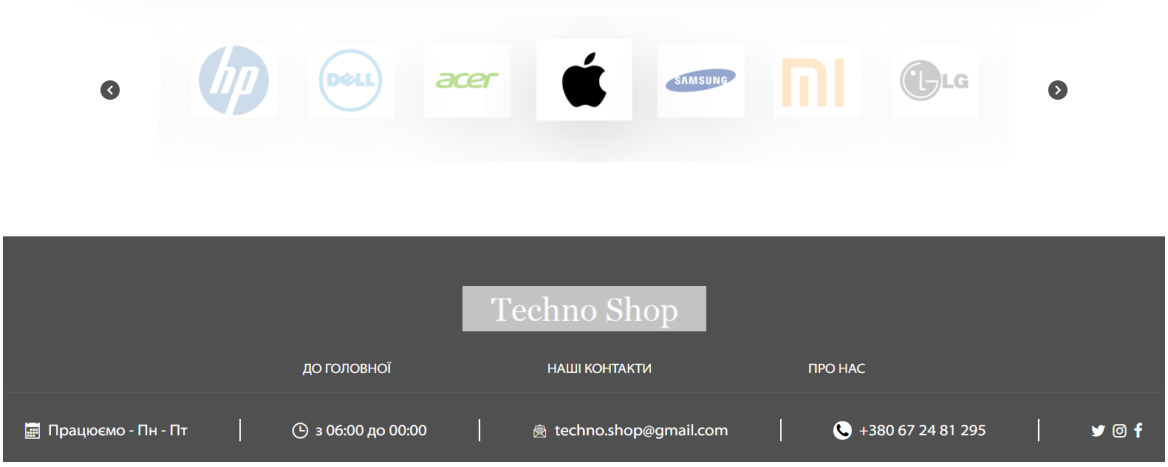

Рисунок 3.8 – Нижня частина веб-додатку

Для авторизації та реєстрації користувачів було використано Laravel – бібліотеку Passport, яка надавала можливість більш безпечно та ефективно виконати даний функціонал. При реєстрації користувачів до бази даних формувався хеш-пароль, замість звичайного паролю, що є безпечніше ніж

звичайне збереження паролю користувача у базі даних. При авторизації у кожного користувача створюється своя сесія, а бібліотека Passport генерує унікальний токен (ключ), який забезпечує більш захищене знаходження та використання веб-додатку.

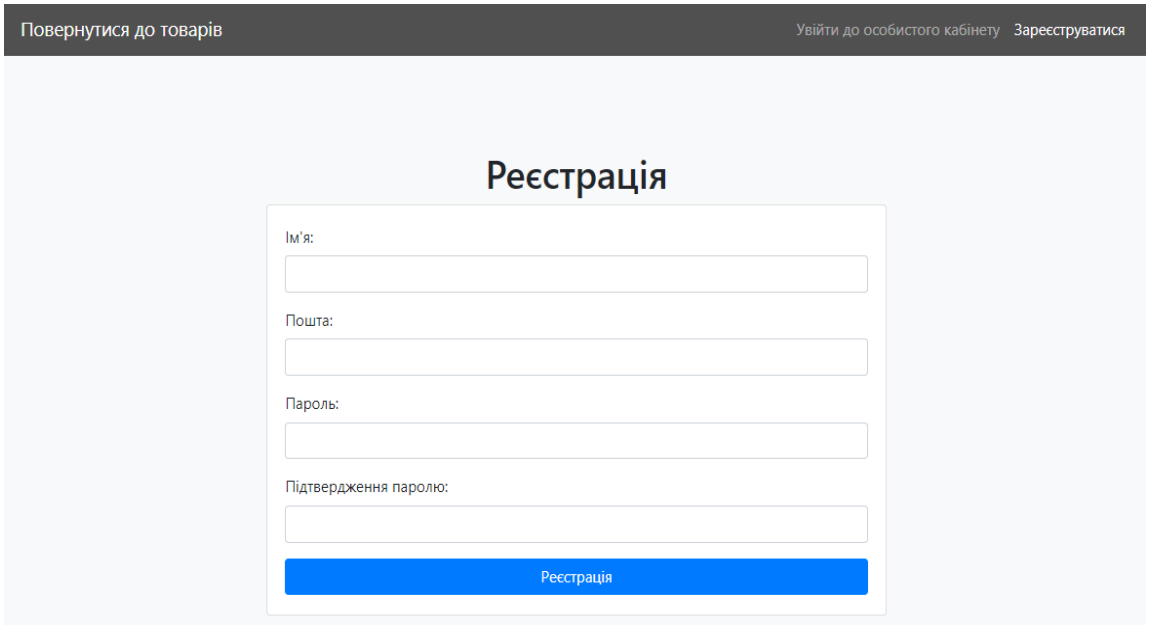

Рисунок 3.9 – Реєстрація користувачів

На рис 3.10 наведений кошик, в який користувач може збирати товари, видаляти їх, розраховувати ціну та згодом придбати їх.

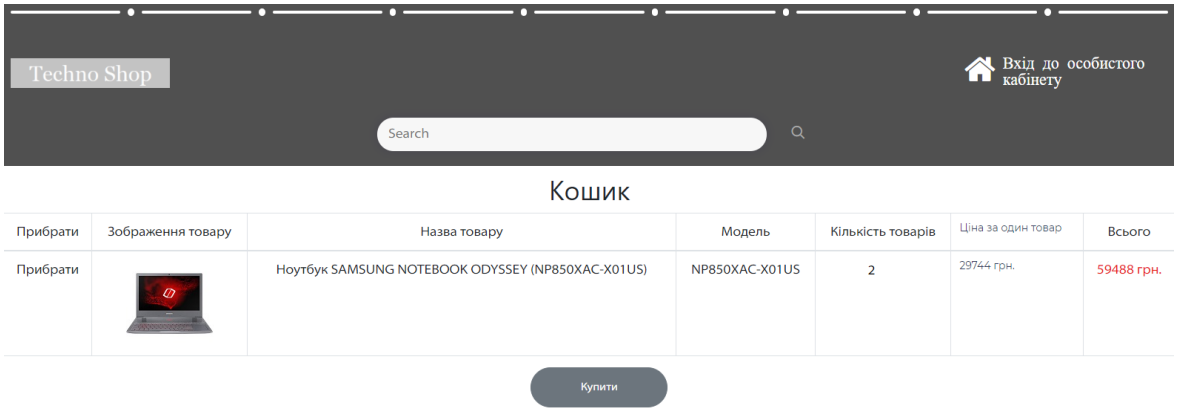

Рисунок 3.10 – Кошик користувача

Для додавання та редагування товарів у веб-додатку було розроблено адміністративну панель. Для більш швидкого пошуку товару було додано можливість пошуку через пошукову систему, яка шукає товари по їх назві в базі даних.

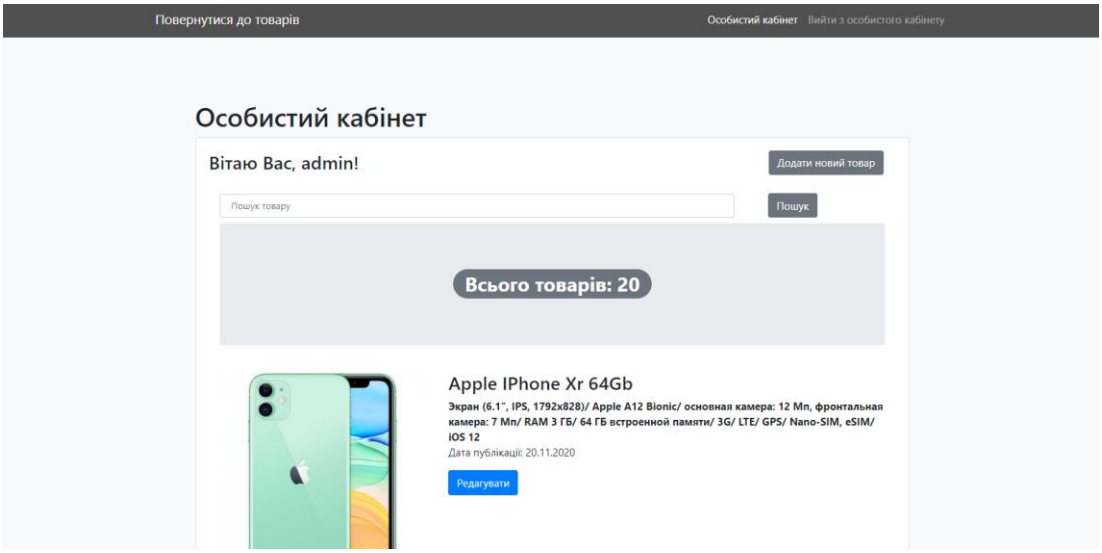

Рисунок 3.11 – Адміністративна панель

Для збереження інформації про переглянуті користувачем сторінки веб-додатку та збереження товарів до кошику було обрано рішення використати веб сховище – localStorage. На відміну від sessionStorage, у localStorage час збереження товарів в кошику залежить від того, коли сам користувач не вирішить очистити кеш в браузері.

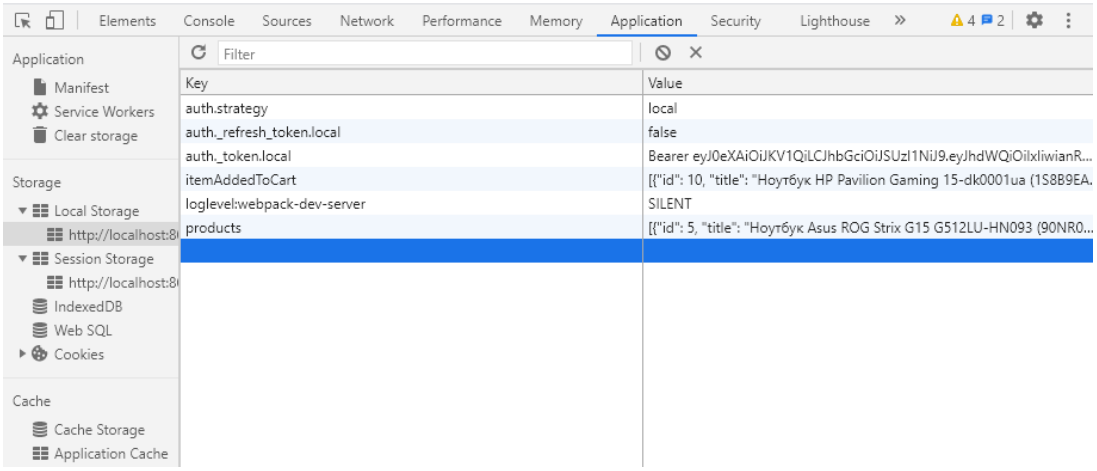

Рисунок 3.12 – Збереження товарів у веб сховище

3.4 Тестування веб-додатку

При розробці серверної частини додатку, а саме для тестування та вирішення різних проблем з роботою з API, був обраний додаток Postman.

Postman – додаток для розробки API, який допомагає створювати, тестувати та модифікувати API. Цей додаток може створювати різні типи HTTP-запитів, наприклад: GET, POST, PUT, PATCH, DELETE та інші, зберігаючи середовища для подальшого використання.

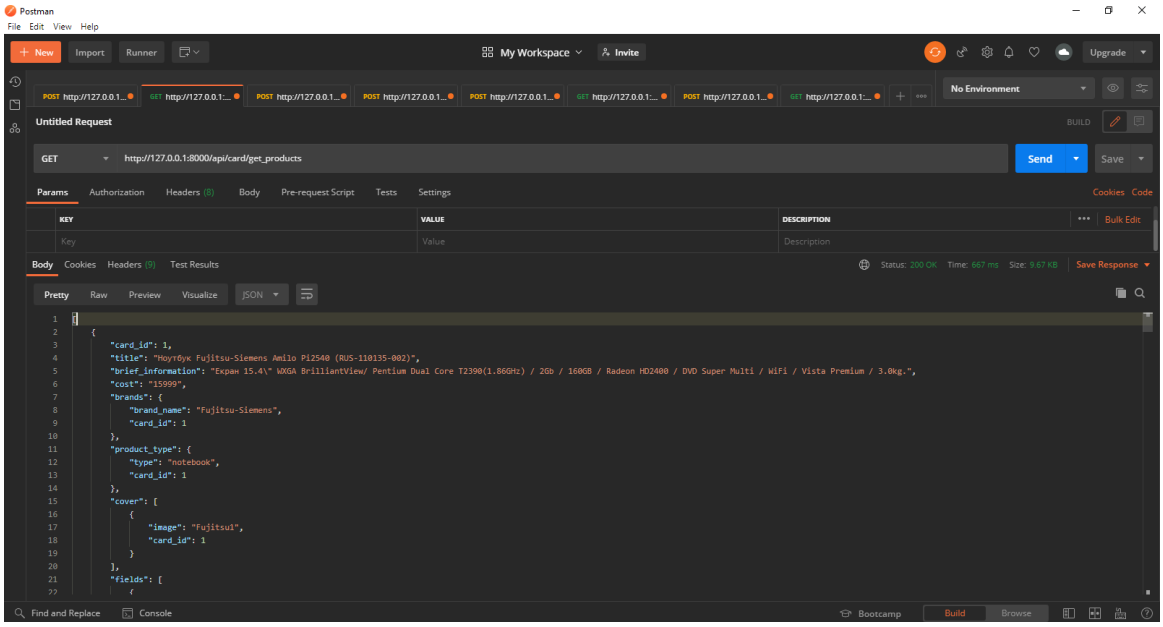

Рисунок 3.13 – Додаток для тестування

На рисунку можна побачити, який саме йде запит (GET), по якій адресі йде запит, та що веб-розробник отримує (який тип даних, які поля та іншу інформацію).

За допомогою даного додатку тестування показало, що розроблений рекомендаційний модуль та веб-додаток працюють коректно.

ВИСНОВКИ

При виконанні роботи було проведено такі дослідження:

- проведено аналітичний огляд інформаційної системи;
- розроблена база даних, за допомогою СУБД MySQL, яка має 14 таблиць;
- розроблений модуль підбору рекомендацій, за допомогою мов програмування PHP та JavaScript;
- розроблений веб-додаток, у текстовому редакторі коду – Visual Studio Code, за допомогою таких технологій, як: HTML, CSS, Bootstrap, JavaScript, Vue.js, PHP, Laravel;
- протестована система, інформаційна система була протестована за домопогою додатку Postman. Результати тестування показали, що даний рекомендаційний модуль та веб-додаток працюють коректно.

Під час розробки роботи були проаналізовані різні технології, рекомендаційні системи, мови програмування, бібліотеки та фреймворки. Серед усіх існуючих технологій, було обрано найбільш підходящі технології для обраного завдання, а саме: мови - PHP, JavaScript, CSS, HTML, програмні забезпечення – OpenServer, Postrman, VSC, фреймворки – Laravel, Vue.js, Bootstrap та СУБД – MySQL.

Отже, мета проєкту була досягнута – були досліджені технології та рекомендаційні системи, а також був створений та оптимізований веб-додаток з використанням алгоритму для формування рекомендацій товарів, який має зручний користувальницький інтерфейс, сучасний дизайн та багатофункціональну адміністративну панель для управління контентом додатку.

СПИСОК ЛІТЕРАТУРИ

1. What is Internet? How the Internet works. – https://searchwindevelopment.techtarget.com/definition/Internet

2. Product recommendations on the site – https://esputnik.com/blog/tovarnye-rekomendacii-na-sajte-instrument-povysheniya-konversii.

3. Грокаем алгоритми - Адітья Бхаргава, 2017. – 290с.

4. Web application – free encyclopedia. – https://www.maxcdn.com/one/visual-glossary/web-application/

5. Web Application Development. – https://www.comentum.com/guide-to-web-application-development.html

6. Visual Studio Code. – https://blog.eduonix.com/software-development/visual-studio-code-popular/

7. A Software Engineer Learns HTML5, JavaScript and jQuery: A guide to standards-based web applications by Dane Cameron, 2013. – 257 pages.

8. Responsive Web Design with HTML5 and CSS: Develop future-proof responsive websites using the latest HTML5 and CSS techniques, by Ben Frain, 2020. – 410.

9. HTML: HyperText Markup Language. – https://developer.mozilla.org/en-US/docs/Web/HTML

10. Таблиці каскадних стилів CSS. – http://htmlbook.ru/css

11. What is JavaScript? The full stack programming language. – https://www.infoworld.com/article/3441178/what-is-javascript-the-full-stack-programming-language.html

12. What is Bootstrap? Framework best practices. – https://wpamelia.com/what-is-bootstrap/

13. 15 Best CSS Frameworks for Developers in 2020. – https://www.mockplus.com/blog/post/css-framework

14. Introduction to Back End Programming Language. – https://www.educba.com/back-end-programming-languages/

15. Why Laravel is the best PHP Framework to use in 2020? – https://www.freecodecamp.org/news/why-laravel-is-the-best-php-framework-to-use-in-2020/

16. What is MySQL? Definition, Features, Explantation. – https://blog.templatetoaster.com/what-is-mysql/

17. What is MVC? Advantages and Disadvantages of MVC. – https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/

18. Open Server – a portable server platform and software environment. – https://alternativeto.net/software/open-server/

19. The good and the bad of Vue.js Framework programming. – https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/

ДОДАТОК

Модель з брендами товарів та методом, який зв’язує бренди з товарами:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Brand extends Model
{
    protected $table = 'brands';

    protected $primaryKey = 'brand_id';

    protected $fillable = ['brand_id', 'brand_name', 'card_id'];

    /**
     * relation one to one with Products
    */
    public function cards()
    {
        return $this->belongsTo('App\Card', 'card_id');
    }
}
```

Модель з товарами, методами підбору рекомендацій та зв’язками:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Http\Request;

class Card extends Model
```

```
{
    protected $table = 'cards';


    protected $primaryKey = 'card_id';


    protected $fillable = ['card_id', 'title', 'brief_information', 'cost',
'additional_information'];


    /**
     * callable in controller for getting all products
    */
    public function getProductsAllInfo(Request $request, int $offset = NULL,
int $limit = NULL)
    {
        $offset = $request->offset;
        $limit = $request->limit;


        # check have or not offset and limit
        $products = Card::with('brands:brand_name,card_id')
            ->with('product_type:type,card_id')
            ->with('cover:image,card_id')
            ->with('fields:name_field,description_field,card_id')
            ->select('cards.card_id',                              'cards.title',
'cards.brief_information', 'cards.cost')
            ->orderBy('cards.created_at');


        if (isset($offset) && isset($limit)) {
            $products = $products->offset($offset)->limit($limit);
        }


        $products = $products->get();


        return $products;
    }


    /**
     * callable in controller for getting products after filtered by brand
name
```

```
    */
    public function getProductsByBrandName(Request $request, string $brand,
int $offset = NULL, int $limit = NULL)
    {
        $offset = $request->offset;
        $limit = $request->limit;

        # check have or not comma in query
        $commaCheck = preg_match_all('/,/', $brand);

        # check one or more elements after choice brands
        $brand = !empty($commaCheck) ? explode(',', $brand) : array($brand);

        # check have or not offset and limit
        $products = Card::with('brands:brand_name,card_id')
            ->with('product_type:type,card_id')
            ->with('cover:image,card_id')
            ->with('fields:name_field,description_field,card_id')
            ->select('cards.card_id',                              'cards.title',
'cards.brief_information', 'cards.cost')
            ->whereIn('brands.brand_name', $brand)
            ->orderBy('cards.created_at');

        if (isset($offset) && isset($limit)) {
            $products = $products->offset($offset)->limit($limit);
        }

        $products = $products->get();

        return $products;
    }

    /**
     * callable in controller and getting products by type (smartphone or TV
or anything else)
    */
    public function getProductsByProductType(Request $request, string $type,
int $offset = NULL, int $limit = NULL)
```

```
    {
        $offset = $request->offset;
        $limit = $request->limit;

        # check have or not comma in query
        $commaCheck = preg_match_all('/,/', $type);

        # check one or more elements after choice brands
        $type = !empty($commaCheck) ? explode(',', $type) : array($type);

        # check have or not offset and limit
        $products = Card::with('brands:brand_name,card_id')
            ->with('product_type:type,card_id')
            ->with('cover:image,card_id')
            ->with('fields:name_field,description_field,card_id')
            ->select('cards.card_id',                              'cards.title',
'cards.brief_information', 'cards.cost')
            ->whereIn('product_type.type', $type)
            ->orderBy('cards.created_at');

        if (isset($offset) && isset($limit)) {
            $products = $products->offset($offset)->limit($limit);
        }

        $products = $products->get();
        return $products;
    }

    /**
     * method for getting recommendation products
    */
    public function getRecommendation(Request $request, $id)
    {
        $model = Card::findOrFail($id);
        $cost = $this->getArrayRecommendCosts($model);
```

```
        $recommendations = Card::where('cards.card_id', '!=', $model->card_id)
            ->whereIn('cost', $cost)
            ->leftJoin('product_type', 'product_type.card_id', '=', 'cards.card_id')
            ->where('product_type.type', '=', $model->product_type->type)
            ->leftJoin('brands', 'brands.card_id', '=', 'cards.card_id')
            ->leftJoin('cover', 'cover.card_id', '=', 'cards.card_id')
            ->get();

        return $recommendations;
    }


    # get costs +- 2k
    public function getArrayRecommendCosts(object $product)
    {
        # get cost from model
        $cost = $product->cost;
        $cost = intval($cost);

        $copyCost = $cost;
        $arrayWithRecommendations[] = $cost;

        for ($i = 0, $count = 2; $i < $count; $i++) {
            $arrayWithRecommendations[] = intval($copyCost) - 1000;
            $copyCost = intval($copyCost) - 1000;
        }

        for ($i = 0, $count = 2; $i < $count; $i++) {
            $arrayWithRecommendations[] = intval($cost) + 1000;
            $cost = intval($cost) + 1000;
        }
        return $arrayWithRecommendations;
    }

    /**
```

```
     * data for create new object in CardController
    */
    public function dataToCreate($request)
    {
        $dataToSave = [
            'title' => $request->title, 'brief_information' => $request->brief_information, 'cost' => $request->cost,
            'additional_information' => $request->additional_information
        ];
        return $dataToSave;
    }

    /**
     * data to update
    */
    public function dataToUpdate(Request $request, $model)
    {
        $dataToSave = [
            'title' => $request->title ? $request->title : $model['title'],
            'brief_information' => $request->brief_information ? $request->brief_information : $model['brief_information'],
            'cost' => $request->cost ? $request->cost : $model['cost'],
            'additional_information' => $request->additional_information
        ];
        return $dataToSave;
    }

    /**
     * relation has one brand in table brands
    */
    public function brands()
    {
        return $this->hasOne('App\Brand', 'card_id');
    }

    /**
     * relation has many images in cover
```

```
    */
    public function cover()
    {
        return $this->hasMany('App\Cover', 'card_id');
    }

    /**
     * relation has one type in product_types
    */
    public function product_type()
    {
        return $this->hasOne('App\ProductType', 'card_id');
    }

    /**
     * relation has many fields in field table
    */
    public function fields()
    {
        return $this->hasMany('App\Field', 'card_id');
    }
}
```

Модель для збереження картинок з методом зв’язку з товарами:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Cover extends Model
{
    protected $table = 'cover';

    protected $primaryKey = 'cover_id';
```

```
    protected $fillable = ['cover_id', 'image', 'card_id'];

    public static function getImage($model) {

        $image = $model->cover;
        $image = $image[0]['image'];
        $imageWithoutPath = str_replace('storage/', '', $image);

        return $imageWithoutPath;
    }

    // relation this table has many images for Products
    public function cards() {
        return $this->belongsToMany('App\Card', 'card_id');
    }
}
```

Адаптивна модель для необмеженого та адаптивного додання полів, зі зв’язком до товарів:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Field extends Model
{
    protected $table = 'fields';

    protected $primaryKey = 'field_id';

    protected $fillable = ['field_id', 'name_field', 'description_field',
'card_id'];

    /**
```

```
     * relation one to one with Products
    */
    public function cards() {
        return $this->belongsToMany('App\Card', 'card_id');
    }
}
```

Модель для збереження типів товарів та зв’язку з ними:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class ProductType extends Model
{
    protected $table = 'product_type';

    protected $primaryKey = 'product_type_id';

    protected $fillable = ['product_type_id', 'type', 'card_id'];

    /**
     * relation for Product. This table have product_types (smartphone, TV,
notebooks and other)
     */
    public function cards()
    {
        return $this->belongsTo('App\Card', 'card_id');
    }
}
```

Модель для реєстрації користувачів:

```
<?php

namespace App;
```

```
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Passport\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];
```

```
}
```

Контролер для сортування товарів по-бренду:

```
<?php

namespace App\Http\Controllers;

use App\Brand;
use Illuminate\Http\Request;

class BrandController extends Controller
{
    /**
     * get brands
    */
    public function getBrands()
    {
        $brands = Brand::select('brands.brand_name')->get();

        return response()->json($brands, 200);
    }
}
```

Контролер для додання нових товарів, редагування, видалення, та показу:

```
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Storage;
use App\Card;
use App\Cover;
use App\Brand;
use App\Field;
```

```
use App\ProductType;
use App\Http\Requests\CardRequest;

class CardController extends Controller
{
    /**
     * create one card
    */
    public function create(CardRequest $request)
    {
        $model = new Card();

        $card = Card::create($model->dataToCreate($request));

        // check has img in the request or not
        $image = null;
        if ($request->hasFile('image')) {
            $image = $request->file('image')->store('uploads', 'public');
            $image = 'storage/' . $image;
        }
        $fields = json_decode($request->fields);

        foreach ($fields as $key => $val) {
            Field::create([
                'name_field' => $fields[$key]->name_field,
                'description_field' => $fields[$key]->description_field,
                'card_id' => $card->card_id
            ]);
        }

        Brand::create(['brand_name' => $request->brand_name, 'card_id' => $card->card_id]);
        ProductType::create(['type' => $request->type, 'card_id' => $card->card_id]);
        Cover::create(['image' => $image, 'card_id' => $card->card_id]);

        return response()->json($card, 201);
```

```
    }


    /**
     * get all products
    */
    public function getProducts(Request $request, int $offset = NULL, int $limit = NULL)
    {
        # TODO for admin this methods
        $model = new Card();


        $model = $model->getProductsAllInfo($request, $offset, $limit);


        return response()->json($model, 200);
    }


    /**
     * get products by brand
    */
    public function getByBrandName(Request $request, string $brand, int $offset = NULL, int $limit = NULL)
    {
        $model = new Card();


        $model = $model->getProductsByBrandName($request, $brand, $offset, $limit);


        return response()->json($model, 200);
    }


    /**
     * get products by product_type
    */
    public function getByProductType(Request $request, string $type, int $offset = NULL, int $limit = NULL)
    {
        $model = new Card();
```

```
        $model = $model->getProductsByProductType($request, $type, $offset, $limit);

        return response()->json($model, 200);
    }

    /**
     * update products
    */
    public function updateCard(Request $request, int $id)
    {
        $model = Card::find($id);
        $main_image = Cover::getImage($model);
        $brand_name = $model->brands;
        $type = $model->product_type;
        $fields = $model->fields;

        $dataToSave = $model->dataToUpdate($request, $model);

        if ($request->hasFile('image')) {
            if (Storage::disk('public')->exists($main_image) )  {
                Storage::disk('public')->delete($main_image);
            }
            $pathImg = $request->file('image')->store('uploads', 'public');
            $main_image = 'storage/' . $pathImg;
        }

        // update Cards
        $model->update($dataToSave);
        // update Brands
        $model->brands->update(['brand_name'    =>    $request->brand_name    ? $request->brand_name : $brand_name->brand_name]);
        // update Product_type
        $model->product_type->update(['type'  =>  $request->type  ?  $request->type : $type->type]);
        // update Cover [0] - because oneToMany
        $model->cover[0]->update(['image' => $main_image]);
```

```
        // update Field
        $decode_fields = json_decode($request->fields);
        foreach ($decode_fields as $key => $value) {
            $model->fields[$key]->update([
                'name_field' => $value->name_field ? $value->name_field : $fields[$key]->name_field,
                'description_field' => $value->description_field ? $value->description_field : $fields[$key]->description_field
            ]);
        }

        return response()->json($model, 201);
    }

    /**
     * delete products
    */
    public function deleteCard(int $id)
    {
        $model = Card::find($id);

        if (empty($model)) return response()->json(null, 202);

        $main_image = Cover::getImage($model);

        if (Storage::disk('public')->exists($main_image)) {
            Storage::disk('public')->delete($main_image);
        }

        $model->delete();

        return response()->json('deleted', 200);
    }

    /**
     * get products recommendations
    */
```

```
    public function getRecommendations(Request $request, $id)
    {
        $model = new Card();

        $model = $model->getRecommendation($request, $id);

        return response()->json($model, 200);
    }
}
```

Контролер для видачі інформації у виді JSON до клієнтської частини:

```
<?php

namespace App\Http\Controllers;

use App\Brand;
use Illuminate\Http\Request;
use App\Card;
use App\ProductType;

/**
 * Get this class to client
*/
class HomeController extends Controller
{
    /**
     * get products
    */
    public function getProducts(Request $request, int $offset = NULL, int $limit = NULL)
    {
        $model = new Card();
        $model = $model->getProductsAllInfo($request, $offset, $limit);

        return response()->json($model, 200);
    }
```

```
    /**
     * get products by brand name
    */
    public function getByBrandName(Request $request, string $brand, int $offset = NULL, int $limit = NULL)
    {
        $model = new Card();
        $model = $model->getProductsByBrandName($request, $brand, $offset, $limit);

        return response()->json($model, 200);
    }

    /**
     * get products by product type
    */
    public function getByProductType(Request $request, string $type, int $offset = NULL, int $limit = NULL)
    {
        $model = new Card();
        $model = $model->getProductsByProductType($request, $type, $offset, $limit);

        return response()->json($model, 200);
    }

    /**
     * get brands
    */
    public function getBrands()
    {
        $model = Brand::select('brands.brand_name')->get();

        return response()->json($model, 200);
    }

    /**
```

```
     * get product types
    */
    public function getProductTypes()
    {
        $model = ProductType::select('product_type.type')->get();

        return response()->json($model, 200);
    }
}
```

Контролер для сортування по типу товару:

```
<?php

namespace App\Http\Controllers;

use App\ProductType;
use Illuminate\Http\Request;

class ProductTypeController extends Controller
{
    /**
     * get types from table product_type
    */
    public function getProductTypes()
    {
        $product_types = ProductType::select('product_type.type')->get();

        return response()->json($product_types, 200);
    }
}
```

Контролер для реєстрації користувача, зміни інформації про нього:

```
<?php

namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
use App\User;
use Illuminate\Validation\ValidationException;
use Illuminate\Support\Facades\Hash;
use App\Http\Requests\UserRequest;

class SignController extends Controller
{
    /**
     * Create user
     *
     * @param  [string] name
     * @param  [string] email
     * @param  [string] password
     * @param  [string] password_confirmation
     * @return [string] message
     */
    public function register(UserRequest $request)
    {
        $request->validate([
            'name' => ['required', 'max:100'],
            'email' => ['required', 'max:50', 'email', 'unique:users'],
            'password' => ['required', 'min:8', 'max:10', 'confirmed']
        ]);

        User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password)
        ]);

        return response()->json([
            'message' => 'Новый пользователь успешно зарегестрирован.'
        ], 201);
    }
```

```
    /**
     * Login user and create token
     *
     * @param  [string] email
     * @param  [string] password
     * @param  [boolean] remember_me
     * @return [string] access_token
     * @return [string] token_type
     * @return [string] expires_at
     */
    public function login(Request $request)
    {
        $request->validate([
            'email' => ['required', 'email'],
            'password' => ['required']
        ]);
        $user = User::where('email', $request->email)->first();

        if (!$user || !Hash::check($request->password, $user->password)) {
            throw ValidationException::withMessages([
                'email' => ['Электронная почта или пароль введены не верно.']
            ]);
        }

        return response()->json($user->createToken('Auth Token')->accessToken, 200);
    }

    /**
     * Logout user (Revoke the token)
     *
     * @return [string] message
     */
    public function logout(Request $request)
    {
        $request->user()->token()->revoke();
```

```
    return response()->json([
        'message' => 'Выход с учётной записи выполнен удачно.'
    ]);
}

/**
 * Get the authenticated User
 *
 * @return [json] user object
 */
public function getUser(Request $request)
{
    return response()->json($request->user());
}

/**
 * Update user info
*/
public function update(Request $request)
{
    $user = $request->user();

    $request->validate([
        'name' => ['max:100'],
        'email' => ['max:50', 'email', 'unique:users'],
        'password' => ['min:8', 'max:10', 'confirmed']
    ]);

    if ( !empty($request->password) ) {
        $password = Hash::make($request->password);
    } else {
        $password = $user->password;
    }

    $dataToSave = [
```

```
            'name' => $request->name ? $request->name : $request->user()->name,
            'email' => $request->email ? $request->email : $request->user()->email,
            'password' => $password
        ];

        $user->update($dataToSave);

        return response()->json([
            'message' => 'Ваши данные обновлены.'
        ], 201);

    }
}
```

Класс, який дозволяє доступ даних тільки для адміністратора:

```
<?php

namespace App\Http\Middleware;

use Closure;

class CheckUser
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if ($request->user()->email != 'admin@gmail.com') {
            return response()->json ('welcome, admin', 200);
```

```
        }
        return $next($request);
    }
}
```

Класс для валідації товарів:

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;
use App\User;

class CardRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        $admin = User::where('email', 'admin@gmail.com')->first();
        if ($admin)
            return true;
        return false;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
```

```
            'title' => ['required', 'max:200'],
            'brand_name' => ['required', 'max:100'],
            'type' => ['required', 'max:100'],
            'image'     =>     ['required',     'mimes:jpeg,png,jpg,gif,svg',
'max:4000'],
            'brief_information' => ['required', 'max:450'],
            'cost' => ['required', 'max:10'],
            'additional_information' => ['max:450']
        ];
    }

    # Messages for response
    public function messages()
    {
        return [
            'title.required' => 'Необходимо указать название.',
            'brief_information.required' => 'Необходимо указать описание.',
            'cost.required' => 'Необходимо указать цену товара.',
            'brand_name.required' => 'Необходимо указать название бренда.',
            'type.required'  =>  'Необходимо  выбрать  тип  товара  (телефон,
телевизор, компьютер или ноутбук).',
            'image.required' => 'Необходимо выбрать изображение.',
            'name_field.required' => 'Необходимо заполнить название поля.',
            'description_field.required' => 'Необходимо дать описание полю.',

            'title.max' => 'Поле должно быть не более 200 символов.',
            'brief_information.max'  =>  'Поле  должно  быть  не  более  450
символов.',
            'brand_name.max' => 'Поле должно быть не более 100 символов.',
            'type.max' => 'Поле должно быть не более 100 символов.',
            'image.max' => 'Изображени должно быть размером менее 4МБ.',
            'cost.max' => 'Поле должно быть не более 10 символов.',
            'image.mimes' => 'Изображение должно быть формата - jpeg, png,
jpg, gif или svg.',
            'additional_information.max'  =>  'Поле  должно  быть  не  более  450
символов.'
        ];
    }
```

```
}
```

Класс для валідації користувачів:

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class UserRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'name' => ['required', 'max:100'],
            'email' => ['required', 'max:50', 'email', 'unique:users'],
            'password' => ['required', 'max:10', 'min:8', 'confirmed']
        ];
    }

    # Messages for response
```

```
    public function messages()
    {
        return [
            'name.required' => 'Необходимо указать Ваше имя.',
            'email.required' => 'Необходимо указать Вашу электронную почту.',
            'password.required' => 'Необходимо указать Ваш пароль.',

            'name.max' => 'Имя должно составлять не более 100 символов.',
            'email.max' => 'Электронная почта должна иметь не более 100 символов.',
            'password.max' => 'Пароль должно иметь не более 10 символов.',

            'email.email' => 'Электронная почта должена быть настоящей.',
            'email.unique' => 'Пользователь с такой электронной почной уже зарегистрирован.',

            'password.min' => 'Пароль должен составлять не мение 8 символов.',
            'password.confirmed' => 'Неправильное подтверждение пароля.'
        ];
    }
}
```

Класс міграцій для створення нової таблиці у базі даних:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateCardsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
```

```
    public function up()
    {
        Schema::create('cards', function (Blueprint $table) {
            $table->bigIncrements('card_id');
            $table->string('title', 200);
            $table->string('brief_information', 450);
            $table->string('cost', 10);
            $table->string('additional_information', 450)->nullable();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('cards');
    }
}
```

Класс для зв’язку контроллера з видом, а саме роути які дають посилання та зв’язок клієнтської частини з серверною:

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\BrandController;
use App\Http\Controllers\ProductTypeController;
use App\Http\Controllers\CardController;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\SignController;

/*
|--------------------------------------------------------------------------
```

```
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

Route::get('/', [HomeController::class, 'getProducts']);

# Default frontend
Route::get('/get_products', [HomeController::class, 'getProducts']);
Route::get('/get_brands', [HomeController::class, 'getBrands']);
Route::get('/get_product-types', [HomeController::class, 'getProductTypes']);
Route::get('/get_brand/{brand}', [HomeController::class, 'getByBrandName']);
Route::get('/get_type/{type}', [HomeController::class, 'getByProductType']);

# Registration and logging
Route::prefix('auth')->group(function(){
    Route::post('/register', [SignController::class, 'register']);
    Route::post('/login', [SignController::class, 'login']);
});

# User
Route::group(['prefix' => 'user', 'middleware' => 'auth:api'], function() {
    Route::post('/logout', [SignController::class, 'logout']);
    Route::get('/get_user', [SignController::class, 'getUser']);

    Route::post('/update', [SignController::class, 'update']);
    Route::get('/get_recommendations/{id}',            [CardController::class,
'getRecommendations']);
});

# Admin
Route::group(['prefix' => 'admin', 'middleware' => ['auth:api', 'admin']],
function() {
```

```
    Route::prefix('card')->group(function(){

        # get info about user

        // Route::get('/get_user', [SignController::class, 'user']);

        # for admin info

        Route::post('/', [CardController::class, 'create']);

        Route::get('/get_products', [CardController::class, 'getProducts']);

        Route::get('/get_brands', [BrandController::class, 'getBrands']);

        Route::get('/get_product-types',        [ProductTypeController::class,
'getProductTypes']);

        Route::get('/get_brand/{brand}',              [CardController::class,
'getByBrandName']);

        Route::get('/get_type/{type}',                [CardController::class,
'getByProductType']);

        Route::post('/update_card/{id}',              [CardController::class,
'updateCard']);

        Route::delete('/delete_card/{id}',            [CardController::class,
'deleteCard']);

    });

});
```