

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ВЕБРОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ З ВИКОРИСТАННЯМ
ОБ'ЄКТНО-РЕЛЯЦІЙНОГО ВІДОБРАЖЕННЯ
WEB DEVELOPMENT OF AN ONLINE STORE USING OBJECT-
RELATIONAL MAPPING

Виконав: студент(ка) 4 курсу, групи КНТ-127

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Зосенко К.В.

(прізвище та ініціали)

Керівник Зайко Т.А

(прізвище та ініціали)

Рецензент Скрупський С.Ю.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ, д.т.н,
проф.
С.О. Субботін
“ ” 2021 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Зосенка Кирила Віталійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Веброзробка інтернет-магазину з використанням об'єктно-реляційного відображення Web Development of an Online Store Using Object-Relational Mapping

керівник проєкту (роботи) Зайко Тетяна Анатоліївна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) 17 травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, завдання

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Опис програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Зайко Т.А., доцент		
Нормоконтроль	Липовець М.В., асистент		

7. Дата видачі завдання « 12» березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1.	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2.	Аналіз та дослідження предметної області	2-3 тижні	Розділи 1
3.	Опис програми	4 тиждень	Розділ 2
4.	Розробка програми	5 тиждень	Розділ 3
5.	Експлуатація, тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6.	Оформлення пояснювальної записки та документів до неї	7 тиждень	Додатки
7.	Нормоконтроль та рецензування	8 тиждень	
8.	Захист роботи	9 тиждень	

Студент(ка)

Зосенко К.В.

(підпис)

(прізвище, ініціали)

Керівник проєкту (роботи)

Зайко Т.А.

(підпис)

(прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 77 с., 46 рис., 1 табл., 3 дод., 16 джерел.

ІНТЕРНЕТ-КОМЕРЦІЯ, JWT, API, REACT.JS, JAVASCRIPT, NODE.JS, EXPRESS, ORM, POSTGRESQL, SEQUELIZE.

Об'єкт дослідження – процес реалізації вебсайтів інтернет-магазинів.

Предмет дослідження – методи розробки вебсайтів інтернет-магазинів з використанням JavaScript, Node.js та React.js.

Мета дипломної роботи – розробка інтернет-магазину електроніки з реалізацією клієнтської і серверної частини з додаванням технології ORM.

Матеріали, методи та технічні засоби: Node.js, React.js, Sequelize, Express та СКБД PostgreSQL. Особливістю розробки є використання виключно мови JavaScript та технологій ORM для реляційних баз даних.

Висновок: розроблено сайт з повноцінним Rest API інтернет-магазину електроніки. Була побудована діаграма баз даних, яка складається з 8 таблиць. Також була створена авторизація по JWT токiну.

Галузь використання – розробка інтернет-магазину, програмне забезпечення вибору товарів у додатку.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 77 pages, 46 figures, 1 tables, 3 appendixes, 16 sources.

INTERNET COMMERCE, JWT, API, REACT.JS, JAVASCRIPT, NODE.JS, EXPRESS, ORM, POSTGRESQL, SEQUELIZE.

The object of research is the process of realization of online store websites.

The subject of research – methods of developing websites of online stores with using JavaScript, Node.js and React.js.

The purpose of the thesis – the development of an online electronics store with implementation of client and server part with the addition of ORM technology.

Materials, methods and technical means: Node.js, React.js, Sequelize, Express and PostgreSQL database. A feature of the development is the use of exclusively JavaScript language and ORM technologies for relational databases.

Conclusion: Developed a site with a full-fledged Rest API online electronics store. A database diagram consisting of eight tables was constructed and JWT token for authorization.

Field of use – creating an online store, software for selecting products in the application.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	7
Вступ	8
1 Аналіз предметної області.....	10
1.1 Методи та принципи організації сучасного інтернет-магазину..	10
1.2 Огляд інтернет-магазинів.....	17
1.3 Висновки за розділом 1	24
2 Опис програми.....	28
2.1 Вибір мови програмування та технологій.....	28
2.2 Визначення структури бази даних.....	31
2.3 Визначення структури клієнтської частини.....	32
2.4 Висновки за розділом 2.....	35
3 Розробка програми.....	36
3.1. Опис розробленої серверної частини	36
3.2 Опис розробленої клієнтської частини.....	44
3.3 Висновки за розділом 3.....	46
4 Експлуатація, тестування програми та експериментальне дослідження програми.....	47
4.1 Опис застосування.....	47
4.2. Умови запуску.....	47
4.3 Робота з сайтом.....	47
Висновки	54
Перелік джерел посилання.....	56
Додаток А Технічне завдання.....	58
Додаток Б Текст програми.....	61
Додаток В Слайди презентації.....	72

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CMS	– Content management system;
IDE	– Integrated Development Environment;
JS	– JavaScript;
PWA	– Progressive Web Application;
БД	– база даних;
ЕР	– електронний ринок;
ОС	– операційна система;
ОРМ	– об'єктно-реляційне відображення;
ПК	– персональний комп'ютер;
СКБД	– система керування базою даних.

ВСТУП

Роль інформації в сучасному світі не можливо перебільшити, саме вона являється керівною силою будь-якої прогресу. У теперішній час Інтернет – це невід'ємна частина нашого життя. Понад 4,1 мільярда людей у світі користуються інтернетом. У сучасному світі понад 1,24 мільярда вебсайтів [1].

З давних часів існування людства, було зроблено багато винаходів, які мали як позитивні, так і негативні сторони. Вказане також стосується мережі Інтернету. Позитивною стороною мережі Інтернету є те, що він є самою великою бібліотекою у світі. Тут можна знайти інформацію на будь-які теми, починаючи з погоди, новин та найновітніших винаходів. Також, однією з позитивних сторін мережі Інтернету є інтернет-магазини, які є досить гарним способом для зайнятих і ділових людей, ефективно упорядкувати/організувати вільний час, вивільняючи його для соціальних стосунків та відпочинку.

Онлайн шопінг набагато вигідніший ніж звичайний, як для покупців, так і для виробників (продавців) будь-якого товару та послуг в світі.

Інтернет-магазин – це вебсторінка, що продає товари або послуги за допомогою мережі Інтернет.

Необхідно з'ясувати цілі та завдання магазину, основні вимоги до оформлення і змісту. Розробити структури і дизайн сайту інтернет-магазину, що вимагає ретельного підходу. Вони повинні бути простими у використанні і інтуїтивно зрозумілими. Чим більш продумано функції, тим впевненіше і комфортніше почуваються відвідувачі інтернет-магазину. Сайт інтернет-магазину повинен легко і швидко завантажуватися з будь-яких пристроїв, комп'ютерів, планшетів і мобільних телефонів. Також мати зручний і простий функціонал. До нього належать: пошук, сортування, фільтрування і підбір товарів або послуг. Також це може бути розсилка та публікація новин, комунікація з клієнтами та SEO-функції. Все це покращує якість обслуговування і приваблює клієнтів [2].

Тому важливість створення програмного забезпечення інтернет-

магазину, яке дозволить користувачам знаходити для придбання необхідний товар, а виробникам (продавцям) збільшити товарообіг, не можливо переоцінити. Програмне забезпечення інтернет-магазину завжди матиме позитивну динаміку розвитку в найближчий час і в майбутньому.

Метою дипломної кваліфікаційної роботи бакалавра було визначено розробку програмного забезпечення інтернет-магазину для простоти та зручності їх використання.

Для досягнення поставленої мети роботи необхідно розв'язати такі завдання:

- аналіз предметної області;
- опис програми;
- розробка програми;
- експлуатація, тестування програми та експериментальне дослідження програми.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Методи та принципи розробки сучасного інтернет-магазину

Кожна сфера сучасної комерції має свої особливості.

Сучасний бізнес характеризується постійним зростом можливостей компаній – постачальників та виробників, що призводить до зросту світової конкуренції та поліпшення якості товарів та послуг. Відбуваються та з прогресивною наростаючою будуть відбуватись, зміни способів організації та керування бізнесом, також реінжинірингу (модернізація) процесів бізнесу та впровадження систем автоматизації. Для успішного ведення бізнесу, в сучасному світі, вкрай необхідно залучати комп'ютерні системи та мережі. На теперішній час сміливо можна сказати, що нова промислова революція триває протягом останніх двадцять років [3].

Розвиток господарчих процесів на базі інтернет-технологій з комерційною формою реалізації – вебсайтами, можна визначити як електронний ринок (ЕР). Основою системи цифрової комерції – є вебсайт (інтернет-сайт), який є сукупністю технічних засобів, програмних продуктів і методів для реалізації в автоматизованому режимі технологічних процесів у певній комерційній операції [3].

Інтернет-магазин (вітрина) – це вебресурс, який виділяється за схемою надання інформації, її обсягу та категорії задач, які вирішуються. Це сайт з каталогом продукції, за допомогою якого клієнти можуть замовити необхідні йому товари. Також даний сайт надає можливість здійснити розрахунки за придбаний товар у електронному вигляді, тобто за допомогою цифрових грошей або електронних платіжних систем.

Урахування мережних взаємодій, тобто проєктування інформаційної системи є новою методологією створення вебсайтів, про якість яких свідчить прогресивне збільшення користувачів.

Основними ознаками цієї методології створення вебсайтів, є наступні принципи:

- формат презентації сайту – відмова від зайвого дизайну сайту та збільшення вмісту та функціональності сайту;

- принцип формування сайту – залучення користувачів до наповнення та перевірки вмісту (контенту) сайту.

принцип зручності (юзабіліті) – визначає зручність та корисність сайту для відвідувачів. Тобто, цей принцип дозволяє з'ясувати, для кого призначена сторінка, яку інформацію потребує аудиторія вашого сайту, як швидко ви можете знайти цю інформацію на своєму сайті, які додаткові послуги можуть бути корисними для ваших відвідувачів [3].

Вищезазначені принципи дозволяють створювати нові вебтехнології для просування товарів та послуг із наступними функціями:

- використання різних вебслужб – замість звичайного програмного забезпечення;

- використання різних нових технологій для поліпшення якості взаємодії користувача з сайтом (наприклад AJAX, RSS);

- поява сайтів нового формату [3].

Окрему увагу треба звернути на вебслужби, це програми, доступ до яких здійснюється через мережу Інтернет в онлайн режимі. Переваги використання цих служб полягають в наступному:

- вебслужба розташована на серверах компанії розробника, тому користувачам не потрібно використовувати свої власні обчислювальні ресурси;

- завжди доступна остання версія служби з повним пакетом оновлень;

- вебслужби є незалежними від будь-якої платформи (засоби для роботи з веб є в усіх операційних системах) [3].

Також не можна залишити поза увагою Систему керування вмістом (контентом) (CMS).

Система керування вмістом (контентом) від англійського Content management system, CMS. CMS – це комп'ютерна програма або система, яка використовується для забезпечення та організації спільного процесу

створення, редагування та керування текстовими та мультимедійними документами (вмістом або контентом) сайту [4].

Принципи, сформульовані для глобальної мережі Інтернет, повинні послужити основою для розвитку принципів систем інтернет-торгівлі та цифрової торгівлі, частиною яких вони є. У цьому випадку принципи, розроблені для систем на різних рівнях ієрархії, не повинні суперечити один одному, а повинні доповнювати та розвивати один одного [3].

Сьогодні при розробці цифрових торгових систем ми можемо визначити деякі принципи, яких слід дотримуватися:

- принцип глобалізації – системи цифрової економіки повинні створюватися з урахуванням не лише потреб внутрішнього ринку, а й потреб світу. Іншими словами, систему потрібно інтегрувати;

- принцип "онлайн" – створені цифрові торгові системи повинні базуватися на дотриманні постійного (тобто в режимі реального часу) режиму роботи, який сприяє необхідній зручності потенційних споживачів у мережі та забезпечує сприятливу економічну ефективність інтернет-компаній;

- принцип інтерактивності – він призначений для характеристики цифрової торгової системи, яка здатна негайно реагувати на будь-яку діяльність (запит) клієнтів мережі;

- принцип безпеки – інформація про учасників бізнесу не повинна переходити у власність третіх осіб; це одна з головних умов успіху системи електронної комерції та відсутності економічної чи фінансової шкоди одному з учасників бізнесу; безпека операцій забезпечується спеціальною платіжною системою;

- зручний принцип роботи інтерфейсу – використання мережевих технологій, що надаються цифровою торговою системою, має бути однаково простим, зручним та доступним для широкого кола потенційних користувачів [3].

Ряд загальних принципів (наприклад, вірогідності, стандартності, інтерактивності, дружності інтерфейсу) повинні увійти як складові частини в

перелік принципів, які формуються для конкретних підсистем, що забезпечують функціонування інтегрованої системи електронної комерції.

Інтернет-магазини – є одним із типів системи сучасного роздрібного продажу товарів у всесвітній мережі Інтернет.

Автоматизація торгівлі через мережу інтернет стала вигідною в зв'язку з прогресивним ростом її масштабів.

Інтернет-магазин – це система, яка охоплює усі основні бізнес-процеси торговельного підприємництва, такі як вибір товарів, оформлення замовлень, проведення взаєморозрахунків, відстеження виконання замовлень, а у випадку продажу інформаційних товарів або надання інформаційних послуг – доставка їх за допомогою мереж електронних комунікацій [3].

Переваги інтернет-магазину у тому, що покупцям можуть запропонувати персональний сервіс, персональну гнучку систему знижок, відразу виписати рахунок з урахуванням вартості доставки товару, типу платежу й страховки, податкових відрахувань.

Крім того, покупець може отримати інформацію про всі етапи опрацювання свого замовлення. Використання даної моделі в електронній комерції дозволяє істотно зменшити товарні запаси на складах, за рахунок глобальності мережі Інтернет, та отримати таким чином, значну економію на витратах у порівнянні з оффлайновими торговельними комплексами. Оскільки замовлення в інтернет-магазині обробляються автоматично, менеджер (продавець) вже не є необхідною ланкою при обслуговуванні покупця, його завдання – це загальний контроль роботи системи [3].

Інтернет-магазин включає до своїх складових наступні основні компоненти:

- інтернет-вітрину (фронт), розташовану на вебсервері, яка має у своєму складі віртуальний споживчий кошик;
- систему прийому платежів;
- систему обліку та контролю виконання замовлень;
- бэк, інформаційні системи якого інтегровані з системами фронту.

Інтернет-магазин призначений для виконання наступних завдань:

- надання онлайнової допомоги покупцям;
- реєстрація покупців;
- надання інтерфейсу доступу до бази даних товарів, які продаються (у вигляді каталогу, листа з ціною);
- робота з електронним кошиком («візком») покупця;
- оформлення замовлень відповідно до вибору методу оплати, доставки, страхування та виписки рахунку;
- резервування товарів на складі;
- проведення розрахунків (при виборі електронних методів оплати) або контроль оплати (при використанні традиційних форм розрахунків);
- формування заявок на доставку товарів покупцям та виписки супровідних документів;
- надання покупцеві засобів відстеження виконання замовлень;
- доставка товарів;
- збір та аналіз різної маркетингової інформації;
- забезпечення безпеки приватної (особистої) інформації покупців;
- автоматичний обмін інформацією з бэком компанії [3].

Вітрина інтернет-магазину розташовується на інтернет-сервері та являє собою вебсайт із активним вмістом. Через те, що інтернет-магазин повинен мати постійний зв'язок з інформаційною системою компанії, він розміщується або на корпоративному сервері в локальній мережі підприємства, або на вилученому сервері з постійно діючим каналом зв'язку.

Необхідність повної автоматизації бізнес-процесів компанії визначає високі вимоги до системи керування процесами бэка. Ця система повинна забезпечувати автоматичне виконання усіх дій, пов'язаних з продажами, складськими операціями, мати внутрішні механізми контролю позаштатних ситуацій і т.п. У загальному випадку, мінімальний набір апаратно-програмних компонентів, який необхідний для роботи інтернет-магазину, включає:

- вебсервер – розподіляє запити, які надходять з інтернету, здійснює

розмежування доступу до інформації;

- сервер додатків – керує роботою торговельної системи, зокрема бізнес-логікою інтернет-магазину;

- СКБД сервер – забезпечує зберігання та обробку даних про товари, клієнтів, рахунки й т.п. [3].

Існує дві моделі функціонування інтернет-магазинів:

- онлайнний магазин (відсутня традиційна торговельна мережа);
- сполучення оффлайнового бізнесу з онлайнним (коли інтернет-магазин створюється на основі реально діючої торговельної структури).

На сьогоднішній день друга модель має більше переваг і більш широко використовується. Сполучення традиційної торговельної мережі з мережею електронної комерції (інтернет-магазином) надає наступні переваги:

- інтернет-магазин користується перевагами доставки з існуючої мережі роздрібних магазинів, він може пропонувати варіант одержання товару в обраному магазині, на відміну від чисто онлайнного магазину для нього немає проблем при поверненні товарів;

- оффлайнві покупці можуть спочатку ознайомитися з товарними асортиментом та його характеристиками на сайті, а потім прийти у найближчий реальний магазин та здійснити покупку.

За наявності товарних запасів інтернет-магазини можна поділити на:

- працюючі за договорами з постачальниками – характеризуються відсутністю будь-яких значних власних товарних запасів;

- маючі власні склади – характеризуються наявністю товарних запасів.

Модель роботи за договорами з постачальниками базується на електронному посередництві віртуального торговельного підприємства між виробниками або дистриб'юторами товарів та роздрібними клієнтами. Більше привабливі, ніж оффлайнові конкуренти, ціни пояснюються відсутністю витрат на придбання (оренду), торговельних приміщень і складів та невисоким рівнем витрат на персонал [3].

Інший тип інтернет-магазинів – ті, хто має власний склад і товарні

запаси. Це може бути інтернет-підрозділ оффлайнової торгово-сервісної або виробничої компанії. Схема взаємодії між продавцем та покупцем у цьому випадку майже не відрізняється від схеми першої моделі. Різниця полягає лише у тому, що в даному випадку магазин оперує товарами власного складу, а не складу постачальника а, отже, менше залежить від зовнішніх факторів. Ця модель не так легко копіюється конкурентами, як попередня, оскільки вимагає капітальних витрат на створення складської системи та товарних запасів [3].

Прийнято виділяти наступні функції керування інтернет-магазином:

- керування інформацією про товари – формування та ведення повного каталогу товарів або послуг;
- керування рекламою – проведення рекламних компаній, акцій, партнерських програм і т.п.;
- керування даними про покупців – на основі профайлів (статистики дій) покупців здійснюється аналіз процесів взаємодії з ними та виробляється відповідна тактика взаємовідносин;
- керування бізнес-логікою – розрахунок податків, знижок, вартості доставки, керування складом, відвантаженням товару й доставкою замовлень і т.п.

Ймовірність здійснення відвідувачем купівлі в інтернет-магазині залежить від того, наскільки широкий спектр супутніх персональних послуг він пропонує та наскільки простий та дружній інтерфейс він має [3].

За даними досліджень було виділено 10 головних параметрів, які дозволяють оцінити якість обслуговування та функціонування інтернет-магазину, до них відносяться:

- асортименти товару;
- наявність товару на складі;
- своєчасність доставки;
- ціна;
- наявність інформації про товар або послугу;
- сервісна підтримка;

- простий механізм повернення товару;
- увага до покупця;
- навігація в інтернет-магазині;
- зовнішній вигляд (дизайн) інтернет-магазину [3].

Найголовніше в успіху інтернет-магазину, це систематичний збір та аналізу статистичної інформації про покупців, через пряму систему комунікації з ними.

1.2 Огляд інтернет-магазинів

Відповідно, для виявлення тенденцій розвитку сучасних інтернет-магазинів можна проаналізувати 10 основних параметрів, які оцінюють якість обслуговування та функціонування інтернет-магазину: асортимент товарів, наявність товарів на складі, своєчасна доставка, ціна, інформація про наявність товар або послуга, сервісна підтримка, простий механізм повернення коштів, увага покупців, навігація в інтернет-магазині, зовнішній вигляд (дизайн) – інтернет-магазину це двигун збільшення продажів товарів і послуг.

Отже, коли ми заходимо в будь-який інтернет-магазині, ми оцінюємо його інтер'єрну чистоту і обстановку, якість викладки магазину (асортимент та наявність), кольору, які нас оточують. Все це в комплексі вже з першої секунди складає враження про магазин і ми приймаємо рішення, будемо купувати щонебудь в цьому магазині або підемо в «сусідній». Також, дизайн сайту повинен бути зрозумілий для відвідувача інтернет-магазину [5].

Як правило, інтернет-магазини містять велику кількість сторінок. І якщо на сайті юзабіліті не продумано, то це загрожує користувачу плутаниною, а власнику сайту – втратою клієнтів. Сайт магазину повинен бути інтуїтивно зрозумілий і зручний у користуванні. Кожен інтернет-користувач вже звик до певного розташуванню навігації. Він знає, що розділ контактів найчастіше розташовується в правому верхньому куті, а меню сайту часто починається в лівому верхньому кутку. Тому юзабіліті дуже важливо для

сайту і, як результат, для зручності відвідувача. Йому повинно бути з першого погляду зрозуміло куди йому кликнути, щоб знайти те, що його цікавить. В інших випадках низький рівень юзабіліті прирікає сайт інтернет-магазину на провал [6].

Отже, аналізу ми будемо піддавати успішні онлайн-магазини, які містять необхідний набір таких елементів, а саме:

- функція пошуку по сайту за назвою товару, коду товару, певних запитах;
- фільтри для пошуку товарів за індивідуальними параметрами;
- фільтри для пошуку товарів за рекомендаційними параметрами
- функція онлайн зв'язку з консультантом;
- функція порівняння товару, додавання продукту в обране;
- функції оплати з прив'язкою надійних банківських систем.

Хороший сайт інтернет-магазину містить велику кількість елементів, які ведуть відвідувача від першого кліка на сайті до моменту завершення покупки, з огляду на його побажання в зручності і психологічне сприйняття інформації. Тут дуже важливо не допустити помилки, які можуть призвести до катастрофічних наслідків [7].

Розглянемо інтернет-магазини, які відповідають гарному прикладу.

Перший інтернет-магазин, який ми розглянемо – Amazon.com (рис. 1.1).

Ясність і простота для вебсторінки – як хороше освітлення для вітрини магазину: при них все здається краще. Легко використовувати сайт, який не примушує замислюватися про несуттєве. Непотрібні роздуми забирають у користувача сили, інтерес і, звичайно, час. Гармонічно простий інтернет-магазин Amazon.com з інтуїтивно зрозумілою структурою, чистим дизайном і мінімальною кількістю елементів. Обмеження навігаційних переходів дає можливість користуватися чистими але лімітованими опціями. Ідея в тому, що великий вибір збиває користувача, а при його обмеженні покупець відчує себе впевненіше (рис. 1.2).

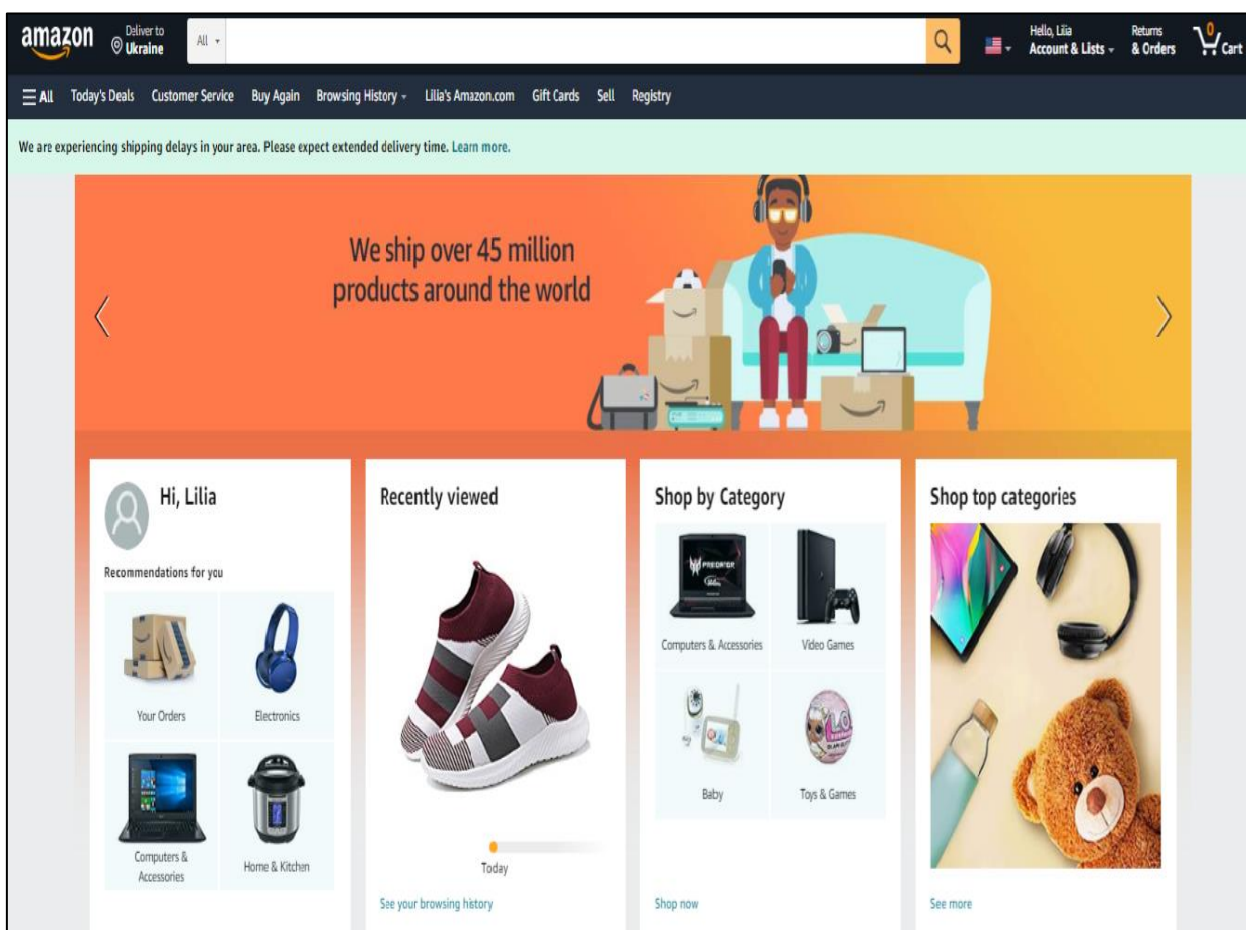


Рисунок 1.1 – Головна сторінка інтернет-магазину Amazon.com

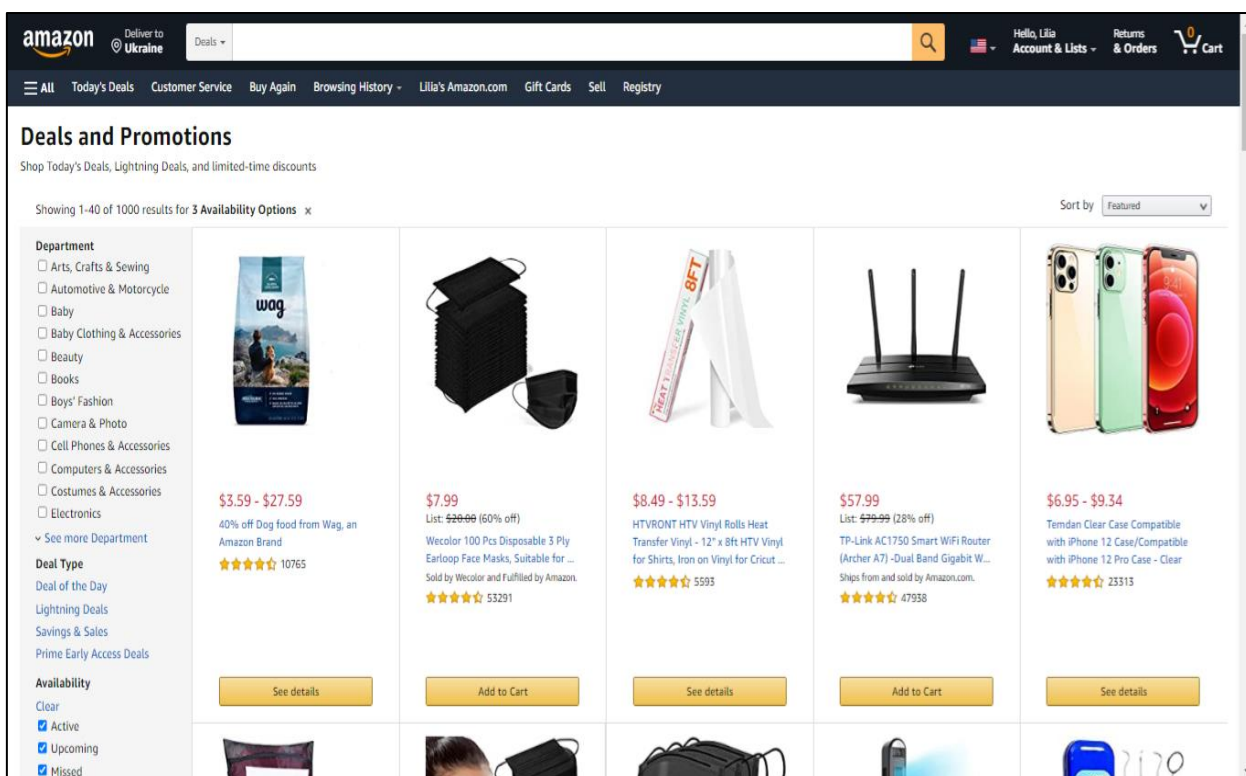


Рисунок 1.2 – Каталог товарів інтернет-магазину Amazon.com

Наступний гарний приклад інтернет комерції – eBay.com (рис. 1.3).

Онлайн майданчик (eBay.com) для проведення аукціонів і торговельний вебсайт, на якому приватні та юридичні особи здійснюють продаж та купівлю різноманітних товарів та послуг. Пошук товару доступний і без реєстрації. Простий пошук проводиться тільки за ключовими словами. Пошук можна уточнити завдяки категоріям і фільтрам, такими як: країна місцезнаходження продавця; відстань продавця від вашого дому; товари, що можна придбати миттєво та товари з швидкою доставкою. Зручні функції, як для продавців так і для покупців для того, щоб оцінити стан ринку. Оплату товару можна здійснювати різноманітними способами (Paypal, кредитка, Money Order, чек). Найпоширенішим і найбезпечнішим способом оплати є оплата за допомогою PayPal.

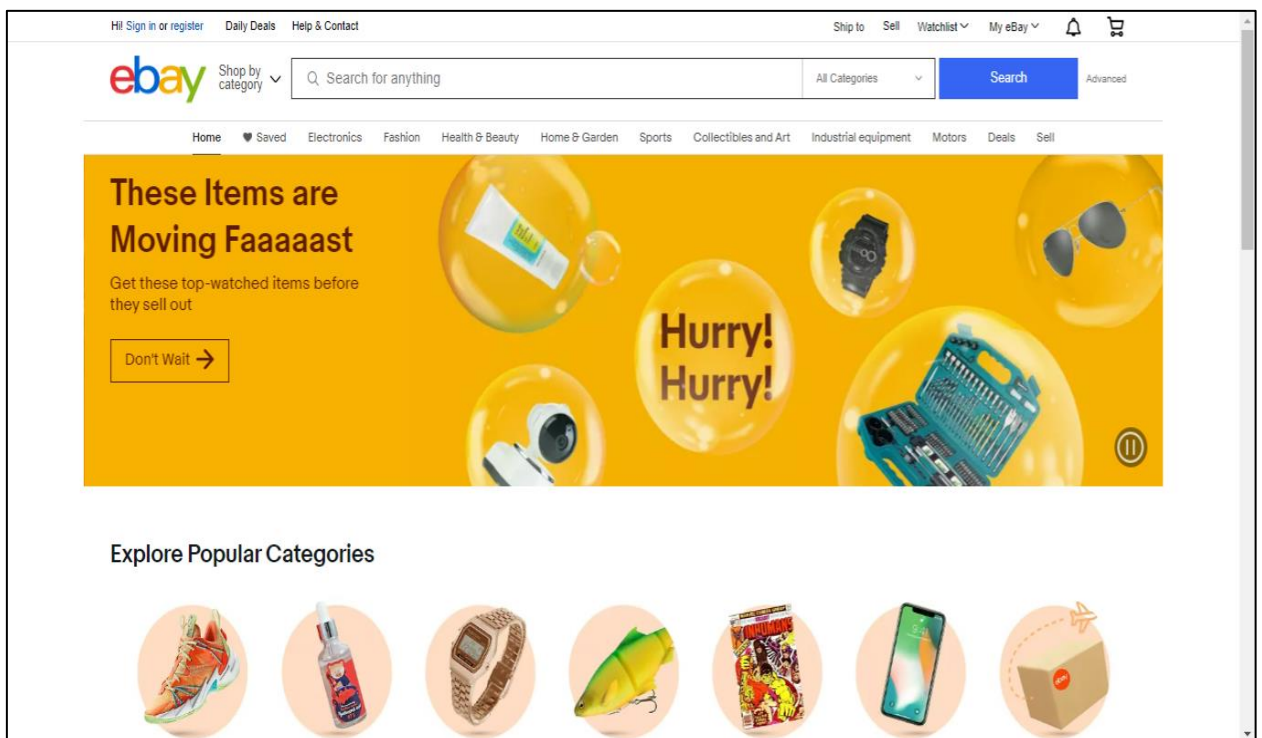


Рисунок 1.3 – Головна сторінка онлайн майданчику eBay.com

Третій, вдалий приклад українського інтернет-магазину – Rozetka.ua (рис. 1.4).

ROZETKA – український інтернет-магазин і маркетплейс, найбільший

онлайн-рітейлер в країні [8].

Звична для покупця навігація сайту інтернет-магазину – Rozetka.ua, інтуїтивно зрозуміла і зручна у користуванні (рис. 1.5). Потужні фільтри каталогів та підкаталогів, допомагають покупцю вибрати необхідну категорію товару, обрати товар за ціною та детальними функціональними властивостями. Переглянувши відгуки покупців про інтернет-магазин Rozetka.ua, в більшості клієнти наголошують, що: «...Інтернет-супермаркет Rozetka.ua це «...зручний інтерфейс...», «...популярні товари...», «...новини, статті та огляди...», «...популярні товари...», «...безкоштовна або недорога доставка...», «...оплата при отриманні...», «...Також, що мені дуже подобається це список бажань, в який можна додавати вподобані товари, моніторити ціни і чекати акції. Детальніше про акції, вони на розетці бувають дуже часто, що як раз можна відстежити маючи список бажань, і знаючи вартість товарів без акцій...». [9].

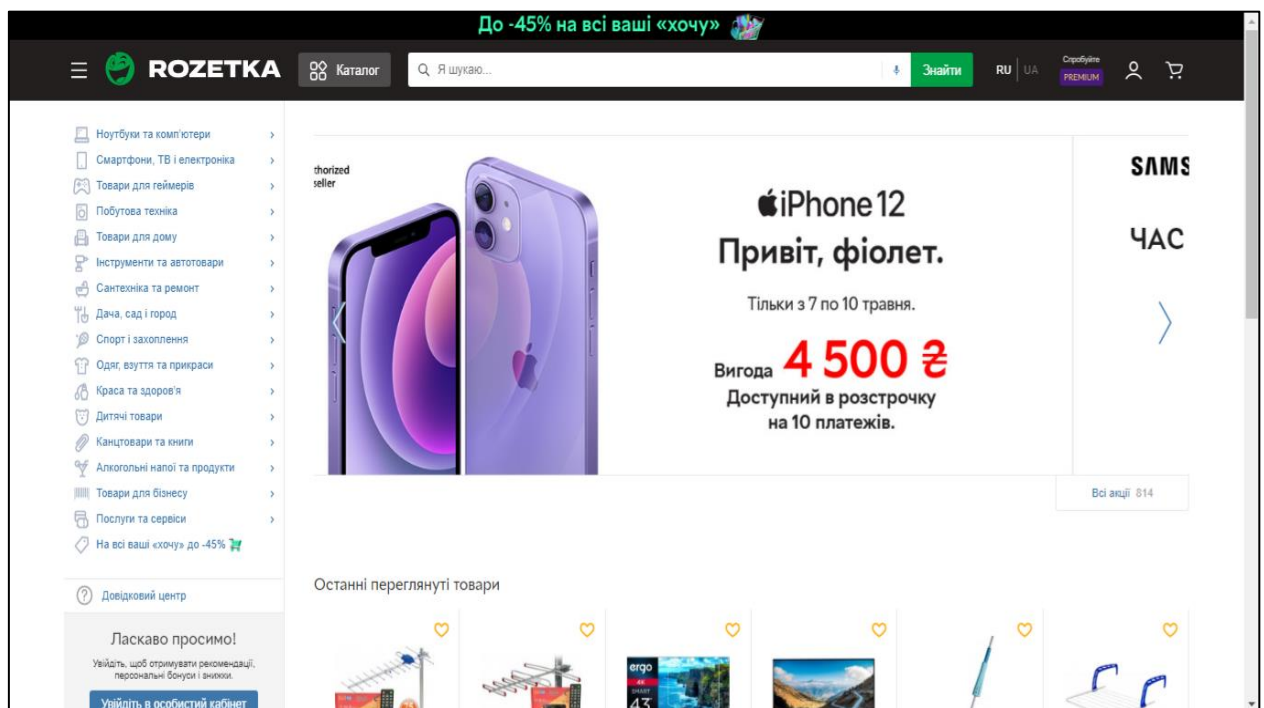


Рисунок 1.4 – Головна сторінка інтернет-магазину Rozetka.ua

Доречі, детальний фільтр якісних та технічних характеристик товару не тільки дозволяє покупцю знайти необхідний товар, а і виробникам

просувати товар з новітніми технічними якостями, користуючись природньою цікавістю покупця. Виробнику набагато дешевше за офлайнову рекламу просувати товар з новітніми технічними якостями, розміщуючи про це інформацію у фільтрах каталогів інтернет-магазинів, чому яскравим прикладом є інтернет-магазин Rozetka.ua.

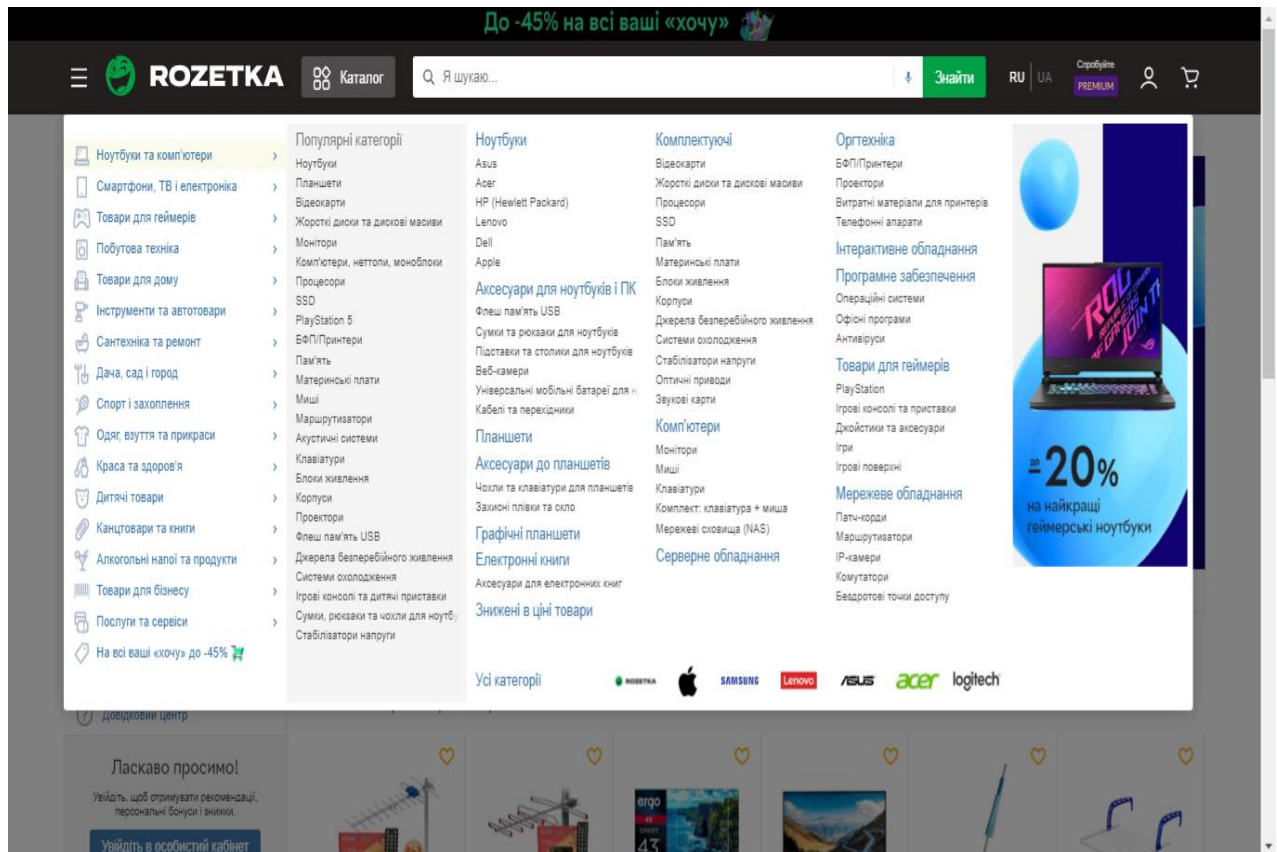


Рисунок 1.5 – Головна сторінка інтернет-магазину Rozetka.ua

Кошик у бічній панелі та можливість додати у кошик додаткові товари є гарним рішенням для зручності покупки товарів. Відвідувач сам контролює перегляд. Це означає потенційний покупець не загубиться, заощадить час і охочіше скористається пропозицією (позитивний досвід користування).

Найголовнішою перевагою інтернет-магазину Rozetka.ua – є те, що у покупців при остаточному виборі товару, є можливість ознайомитись з досвідом користування інших покупців цього товару через «коментарі» до нього.

Кожному фахівцю, що приступає до розробки інтернет-магазинів, потрібно враховувати потреби, мотиви і цільові прагнення людини. Зручні функції повинні працювати на конкретного клієнта, спрощувати її досвід взаємозв'язку з інтерфейсом пропонувати якісні рішення для посилення.

Результати порівняння зведено до табл. 1.1.

Таблиця 1.1 – Порівняння таблиця юзабіліті Інтернет - магазинів

Критерій	Amazon.com	eBay.com	Rozetka.ua
Функція пошуку по сайту за назвою товару, коду товару, певних запитах	Так	Так	Так
Фільтри для пошуку товарів за індивідуальними параметрами	Менш	Так	Так
Функція онлайн зв'язку з консультантом	Так	Зв'язок з продавцем (під сумнівом об'єктивність)	Так
Функції оплати з прив'язкою надійних банківських систем	Так	Так	Так
Функція порівняння товару, додавання продукту в обране	Так	Відсутня	Так

Завдяки систематичному збору та аналізу статистичної інформації про покупців, через зручну та пряму систему комунікації з ними на сайті інтернет-магазину, Rozetka.ua з успіхом пережила два потужних економічних кризи та вийшла в динамічний та позитивний прибуток [8].

1.3 Висновок до розділу 1

Успішність роботи інтернет-магазину залежить, зрозуміло, від дуже великої кількості різних факторів, одним з яких є зокрема його структура.

Важливо ретельно продумати майбутню структуру ще на ранніх етапах розробки інтернет-магазину, щоб користувач міг з першої ж хвилини зрозуміти, де він знаходиться, і почати подорож по сайту.

Структура інтернет-магазинів в кожному випадку буде індивідуальною. Але є деякі її елементи, які в наші дні можна назвати стандартними, без яких онлайн-магазини практично не мають шансу на комерційне використання взагалі [5].

Головна сторінка є «обличчям» проєкту інтернет-магазину. Цей елемент структури покликаний забезпечити відвідувачам зручне і приємне знайомство із запропонованими товарами. Як правило, головна сторінка інтернет-магазину є певним резюме всього його вмісту: відвідувачі тут можуть побачити і частину представлених товарів (наприклад, найновіші або найпопулярніші), і коротку інформацію про магазин, і якісь особливо цікаві акційні пропозиції або знижки, і контактні дані.

Структура майбутнього інтернет-магазину повинна бути зручною в просторовій навігації по сайту. Якщо відвідувач заплутається в каталозі товарів, він полишить сайт, не зробивши покупки.

Якщо «обличчям» інтернет-магазину називати його головну сторінку, то каталог – це, звичайно ж «серце» такого сайту, каталоги товарів потрібно проєктувати за ієрархічним принципом. Логічно і послідовно розроблена структура каталогу просто необхідна для зручної і, що важливо, результативної роботи з інтернет-магазином [5].

Створюючи каталог інтернет-магазину, надзвичайно важливо правильно визначити необхідну глибину вкладеності. Отже, є два важливі чинники при розробці каталогу, а саме: сторінка певного товару, яку потенційний покупець може отримати до 2-3 клика, і важливо, щоб кількість товарів, об'єднаних в одну категорію чи розділ, не надто великий для зручного

пошуку. Для цього потрібна продумана, універсальна навігація (фільтр), яка допомагає вибирати товари, виходячи з цих характеристик (вартість, колір, розмір, склад тощо).

Крім того, щоб полегшити покупцеві пошук бажаного товару, не вдаючись до просторової навігації, бажано здійснити швидкий пошук та розмістити його в верхньому правому кутку заголовку сайту.

Завантаження сайту інтернет-магазину з будь-яких пристроїв, комп'ютерів, планшетів і мобільних телефонів, повинно відбуватись легко і швидко. Згідно статистики, 40% людей не заходять на вебсайт, якщо він завантажується довше 3 секунд.

Сайт інтернет-магазину відкривають з різних пристроїв. Тому необхідно створювати адаптовану верстку. Користуватися сторінками такого магазину буде набагато зручніше.

Якщо структура інтернет-магазину незручна для відвідувачів і навіть не оптимізована для просування в пошуковій системі, такий бізнес втратить майже половину шансів на успіх, якщо не більше.

Не менш важливим для інтернет-магазину є сторінка кошика та замовлення клієнта. Без кошика для покупок і сторінки клієнта жоден інтернет-магазин не зможе працювати для покупців, якщо їх немає в структурі. Загалом можна сказати, що вам потрібна коротка інформація про замовлений товар (назва, кількість, ціна), пропозиція повернутися до покупки, поради щодо заповнення полів у формі замовлення. Але обсяг інформації в кошику покупця буде унікальним для кожного проєкту, залежно від товарів та послуг, що пропонуються покупцеві [5].

Крім того, ви не можете уявити вебсайт інтернет-магазину без сторінки з інформацією про магазин, оплату та доставку, гарантії, повернення та контакти.

Користувачі інтернет-магазину не зможуть здійснити покупку, якщо не отримають точної та повної інформації про те, що це за магазин, які умови оплати та доставки, які інтернет-магазин дає гарантії якості представлених

товарів і на яких умовах робить повернення. Отже, сторінки, що містять таку інформацію (або принаймні одну сторінку), повинні бути включені в структуру інтернет-магазину. Рівень вкладеності таких сайтів, як правило, найвищий – перший.

Для вигоди потенційних покупців та для оптимізації пошукової системи дуже важливо, щоб у структурі інтернет-магазину регулярно оновлювався розділ вмісту. Розділ вмісту може містити: новини, корисні поради щодо використання або вибору товарів, що пропонуються в магазині, огляд нових продуктів тощо. У навігаційній структурі інтернет-магазину такий розділ, як правило, називається за його основним змістом – «Коментарі», «Думки», «Статті та думки», «Новини».

Привабливий дизайн вебсайту робить процес покупки більш приємним. Якісний дизайн привертає увагу. Це врешті-решт призводить до придбання товару.

Зручна та проста функціональність, яка включає: пошук, сортування, фільтрування та вибір товарів чи послуг. Це також може бути надсилання та публікація новин, спілкування з клієнтами та SEO функції. Все це покращує якість обслуговування та приваблює клієнтів.

Вебсайт інтернет-магазину відкривається з різних гаджетів. Тому необхідно створити адаптований макет, ретельне планування. Користуватися сторінками такого магазину буде набагато зручніше [5].

2 ОПИС ПРОГРАМИ

2.1 Вибір мови програмування та технологій

Для реалізації проєкту було обрано мультіпарадігменну мову програмування Javascript. Ґрунтуючись на цілі проєкту були обрані такі технології:

- Node.js;
- Express;
- PostgreSQL;
- Sequelize;
- React.js.

Головною і критичною перевагою JS в порівнянні з іншими мовами прикладного програмування є універсальна сумісність для всіх ОС створених додатків. Ігри та програми, написані за допомогою JS, виглядають і працюють однаково на всіх ОС. Єдина вимога – наявність сучасного браузера, який підтримує HTML5. У той час як програми, написані на Java, C #, C / C ++, Flash вимагають заздалегідь встановлених Java Runtime Environment, .NET Framework, Visual C ++ Redistributable Package або Adobe Flash Player. І тому мова програмування JavaScript має великі переваги, як мова для веброзробки.

JS входить в перелік найпопулярніших мов програмування, яка використовується для розробки вебсайтів. Тому вона зустрічається майже всюди. JS часто використовується для створення анімації, завантаження нових зображень, сценаріїв або об'єктів на вебсторінки та створення гнучкого інтерфейсу користувача. JS здатна виконувати ряд функцій, включаючи управління браузером, редагування вмісту документа, не кажучи вже про клієнтські сценарії та комунікації з користувачем, а також асинхронний обмін даними.

З випуском ECMAScript 6 та таких фреймворків, як Angular, Node, Express і React, розробники почали використовувати JS для створення клієнтського та серверного програмного забезпечення [10].

Node.js – це серверна технологія заснована на подіях яка використовує рушій Google V8 JS. Одна з переваг Node.js полягає в можливості розробки Node додатків тільки для однієї реалізації JS, а не для великої кількості різних браузерів і їх версій. Технологія Node була задумана як платформа створення додатків, орієнтованих на високу інтенсивність введення-виведення і невисоку інтенсивність обчислень. Асинхронний ввід-висновок означає, що програма не чекає завершення введення-виведення перед переходом до наступного кроку в коді програми. Що ще важливіше, Node пропонує цю функціональність в повністю готовому вигляді. Тобто не доведеться турбуватися про те, що додаток заблокує всю іншу роботу, чекаючи завершення завантаження файлу або поновлення бази даних, оскільки велика частина функціональності за замовчуванням ставиться до асинхронного вводу-виводу. І також не потрібно хвилюватися з приводу програмних потоків, оскільки Node додаток виконується в єдиному програмному потоці [11].

Express – це мінімалістичний і гнучкий вебфреймворк для додатків Node.js, що надає Великий набір функцій для мобільних і вебдодатків [12].

Цей фреймворк чиста форма сервера і може не мати модулів. Головною особливістю фреймворку є те, що Express має кілька основних функцій. Всі інші функції треба брати за допомогою зовнішніх модулів. Завдяки цьому розробник спочатку отримує простий і потужний інструмент, який він може розширити та вдосконалити. Важливо, щоб вибір модулів для Express не підпадав під будь-які обмеження: ні кількісні, ні функціональні. В результаті цей фреймворк дозволяє розробнику вирішити будь-яку проблему, не обмежуючи його у виборі інструментів. З одного боку, добре те, що відсутність готових узагальнених рішень насправді означає, що кожна створена програма буде унікальною. З іншого боку, розробник повинен самостійно вибирати та організовувати модулі, що є великою роботою та вимагає від програміста більше зусиль та часу.

PostgreSQL – це популярна безкоштовна об'єктно-реляційна система

керування базами даних. PostgreSQL заснований на мові SQL і підтримує багато функцій [13].

Завдяки цій СКБД розробник має багатий набір інструментів, що дозволяє реалізувати будь-який тип додатка:

- використання різних мов для програмування на стороні сервера, включаючи вбудований PL / pgSQL (зручна щільна інтеграція з SQL), C для критично важливих завдань, Perl, Python, Tcl та JS, Java та інші;

- програмні інтерфейси для звернення до СКБД з додатків на будь-якій мові, включаючи стандартні інтерфейси ODBC і JDBC;

- набір об'єктів бази даних, що забезпечує ефективну реалізацію логіки будь-якої складності на стороні сервера: таблиці і індекси, обмеження цілісності, уявлення і матеріалізовані уявлення, послідовності, секціонування, під запити і with-запити (в тому числі рекурсивні), агрегатні і віконні функції, збережені функції, тригери і т. д.;

- вбудована гнучка система повнотекстового пошуку з підтримкою всіх європейських мов, доповнена спеціально розробленим ефективним індексним доступом;

- підтримка слабо структурованих даних в дусі NoSQL: сховище пар «ключ-значення» hstore, xml, json (Як в текстовому, так і в ефективному довічному поданні jsonb);

- підключення зовнішніх джерел даних, включаючи всі основні СКБД, як таблиць з можливістю їх повноцінного використання, в тому числі для запису і розподіленого виконання запитів (Foreign Data Wrappers) [14].

Sequelize – це бібліотека ORM для додатків написаних на Node.js, яка відображає таблиці БД та їх зв'язок із класами. Використовуючи Sequelize, можна не писати запити SQL, але працювати з даними, як це було б зі звичайними об'єктами[5]. Крім того, Sequelize може працювати з декількома СКБД – MariaDB, MySQL, SQLite, Postgres, MS SQL Server.

ORM – це технологія програмування, яка дозволяє перетворювати несумісні типи моделей в ООП, зокрема, між сховищем даних і об'єктами

програмування. ORM використовується для спрощення процесу збереження об'єктів в реляційну базу даних і їх вилучення, при цьому ORM сама піклується про перетворення даних між двома несумісними станами. Більшість ORM інструментів значною мірою покладаються на метадані бази даних і об'єктів, так що об'єктам нічого не потрібно знати про структуру бази даних, а базі даних – нічого про те, як дані організовані в додатку. ORM забезпечує повне розділення завдань в добре спроектованих додатках, при якому і база даних, і додаток можуть працювати з даними кожен у своїй вихідній формі [15].

React.js – це JS-бібліотека для створення користувацьких інтерфейсів на різних платформах. Вона надає потужну ментальну модель для роботи і допомагає створювати декларативні і орієнтовані на компоненти призначені для користувача інтерфейси. Ментальна модель React широко використовує функціональні, а також об'єктно-орієнтовані концепції програмування і фокусується на компонентах як єдиної основи для розробки. У React-додатках ви створюєте з них інтерфейси. Система рендеринга React управляє ними і автоматично синхронізує подання додатка. Компоненти часто відповідають елементам користувацького інтерфейсу, наприклад календарів, заголовкам, панелям навігації. І тому є важливою частиною клієнтської програми.

2.2 Визначення структури бази даних

Основні сутності предметної області:

- user (користувач);
- basket (кошик);
- device (пристрій);
- type (тип);
- color (бренд).

Таблиця user включає поля:

- id (ідентифікаційний номер);
- email (пошта);
- password (пароль);
- role STRING (роль користувача).

Таблиця basket включає поля:

- id (ідентифікаційний номер);
- user_id (ідентифікаційний номер користувача).

Таблиця device включає поля:

- id (ідентифікаційний номер);
- name (назва);
- price (ціна);
- rating (рейтинг);
- img (ілюстрація до товару);
- typeid (ідентифікаційний номер типу);
- colorid (ідентифікаційний номер бренду).

Таблиця type включає поля:

- id (ідентифікаційний номер);
- name (назва).

Таблиця color включає поля:

- id (ідентифікаційний номер);
- name (назва).

Таблиця device_info включає поля:

- id (ідентифікаційний номер);
- device_id (ідентифікаційний номер пристрою);
- title (назва);
- description (опис).

Таблиця basket_device включає поля:

- id (ідентифікаційний номер);
- device_id (ідентифікаційний номер пристрою);

– basket_id (ідентифікаційний номер кошика).

Таблиця rating включає поля:

- id (ідентифікаційний номер);
- user_id (ідентифікаційний номер користувача);
- device_id (ідентифікаційний номер пристрою);
- rate (оцінка).

Завершаючи, схему БД представлено на рис. 2.1.

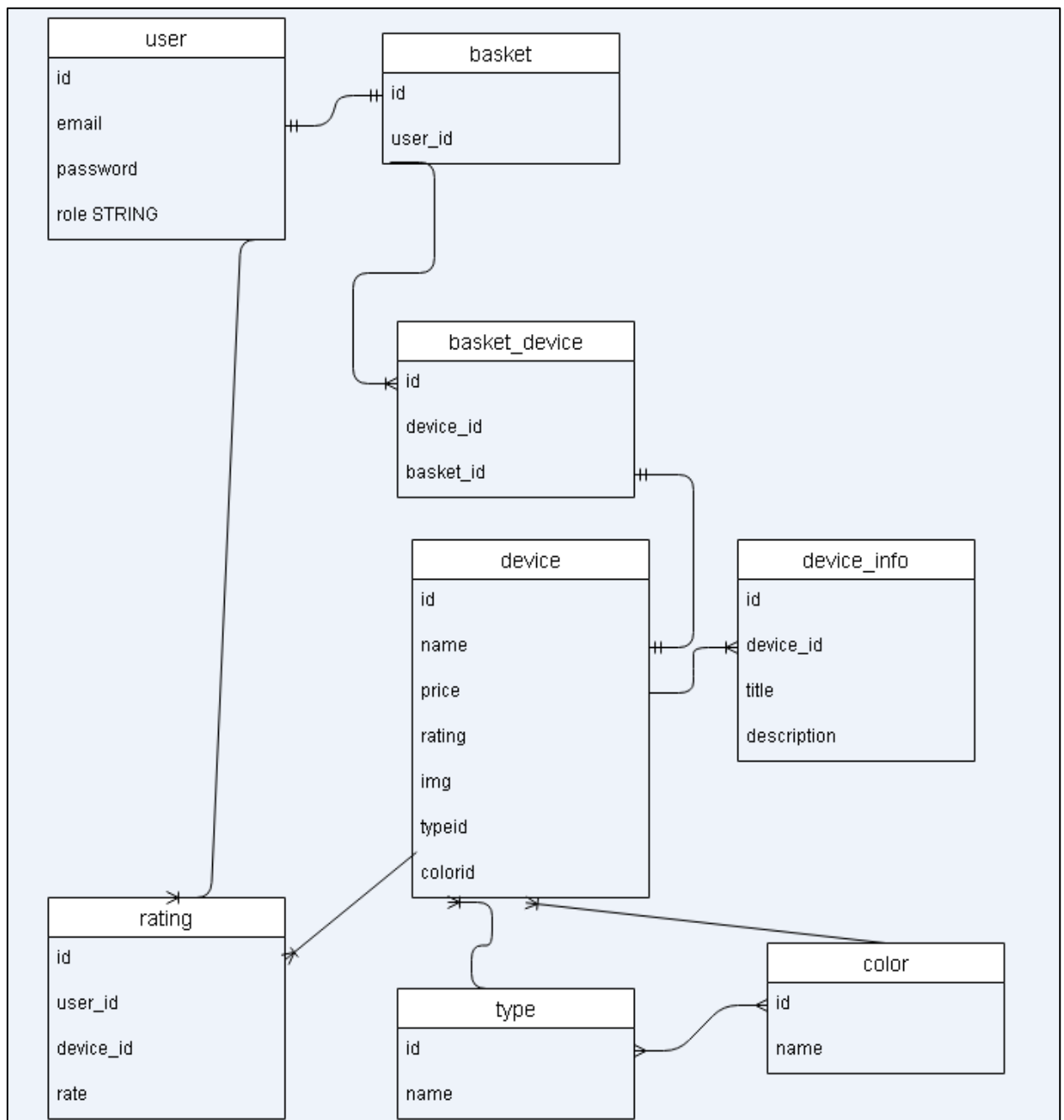


Рисунок 2.1 – Схема БД

Реалізацію бази даних виконано за допомогою СКБД PostgreSQL.

2.3 Визначення структури клієнтської частини

Всі взаємодії вебдодатку з сервером зводиться до 4 операцій:

- отримання даних з сервера (в форматі JSON);
- додавання нових даних на сервер;
- зміна існуючих даних на сервері;
- видалення даних на сервері.

Операція отримання даних не може приводити до зміни статусу сервера.

Структура роботи клієнта з сервером зображено на рисунку (рис. 2.2).

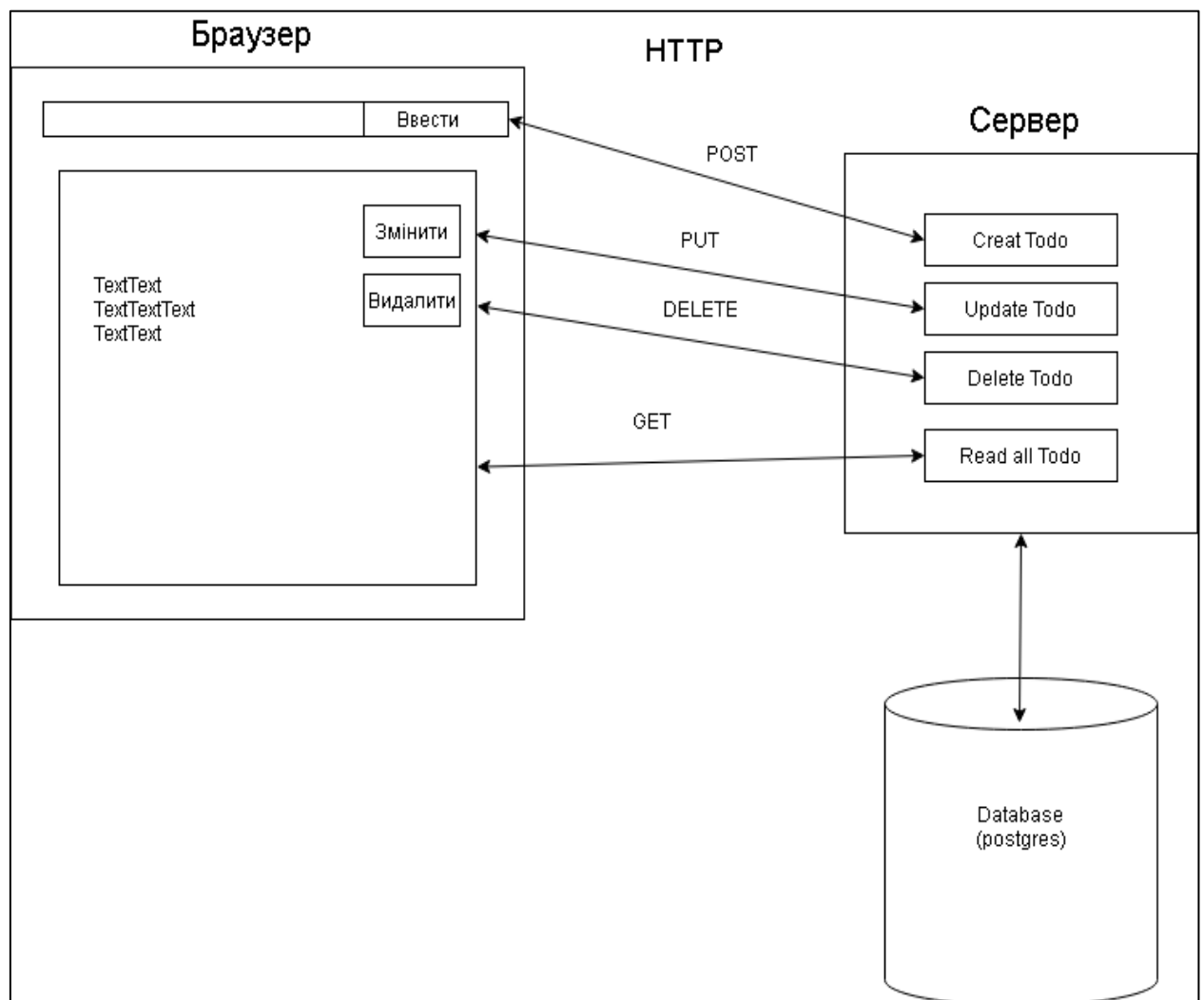


Рисунок 2.2 – Структура роботи вебсайту

Для кожного типу операції використовується свій метод HTTP-запиту:

- отримання – GET;
- додавання – POST;
- модифікація – PUT;
- видалення – DELETE.

Так сервер інтерпретує запити:

- GET-запит /api/users/1 – отримання інформації про всіх користувачів з id 1;
- GET-запит /api/users – отримання інформації про всіх користувачів;
- POST- запит /api/users/ – додавання нового користувача ;
- PUT- запит /api/users – зміна інформації про користувача;
- DELETE- запит /api/users – видалення користувача;
- GET-запит /api/devices/1 – отримання інформації про всі пристрої з id 1;
- GET-запит /api/devices – отримання інформації про всі пристрої;
- POST- запит /api/ devices – додавання нового пристрою.

2.4 Висновки за розділом 2

В даному розділі була описана мова програмування та технології для реалізації , виконано проектування бази даних на основі використання PostgreSQL в якості системи управління базами даних та визначено структуру вебсайту з побудовою відповідної структурної схеми.

3 РОЗРОБКА ПРОГРАМИ

3.1 Опис розробленої серверної частини

На першому етапі відбувається ініціалізація проєкт. Це потрібно для опису проєкту та залежностей які в ньому використовуються. Ініціалізація відбувається за допомогою команди в термінал `npm init -y`, та завантажуюмо ці залежності командою `npm install express pg pg-hstore sequelize cors dotenv`.

Також треба завантажити залежність для розробки `npm -D nodemon`. Це потрібно, щоб для кожної зміни у коді не треба було перезавантажувати сервер, бо команда робить це автоматично.

В описі проєкту та залежностей треба прописати `"dev": "nodemon index.js"`, це скрипт який буде перезавантажувати додаток автоматично при кожній зміні у середовищі розробки (рис. 3.1).

```
main folder > server > npm package.json > {} scripts
1  {
2    "name": "server",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "dev": "nodemon index.js"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "bcrypt": "^5.0.1",
14     "cors": "^2.8.5",
15     "dotenv": "^8.2.0",
16     "express": "^4.17.1",
17     "express-fileupload": "^1.2.1",
18     "jsonwebtoken": "^8.5.1",
19     "pg": "^8.6.0",
20     "pg-hstore": "^2.3.3",
21     "sequelize": "^6.6.2",
22     "uuid": "^8.3.2"
23   },
24   "devDependencies": {
25     "nodemon": "^2.0.7"
26   }
27 }
28
```

Рисунок 3.1 – Файл опису проєкту

У корінному файлі, через який починається запуск додатка, треба імпортувати express, sequelize та інші модулі. Далі, необхідно створити об'єкт у якому зазначаємо порт на якому буде працювати додаток(рис. 3.2). Вказана конфігурація вноситься у змінне оточення (рис. 3.3).

```

JS index.js U x .env U
main folder > server > JS index.js > PORT
1  require('dotenv').config()
2  const express = require('express')
3  const sequelize = require('./db')
4  const models = require('./models/models')
5  const cors = require('cors')
6  const fileUpload = require('express-fileupload')
7  const router = require('./routes/index')
8  const errorHandler = require('./middleware/ErrorHandlerMiddleware')
9
10 const path = require('path')
11 const PORT = process.env.PORT || 5000
12
13 const app = express()
14 app.use(cors())
15 app.use(express.json())
16 app.use(express.static(path.resolve(__dirname, 'static')))
17 app.use(fileUpload({}))
18 app.use('/api', router)
19
20 app.use(errorHandler)
21
22 Complexity is 4 Everything is cool!
23 const start = async () => {
24   try {
25     await sequelize.authenticate()
26     await sequelize.sync()
27     app.listen(PORT, () => console.log(`Server started on port ${PORT}`))
28   } catch (e) {
29     console.log(e)
30   }
31 }
32
33 start()

```

Рисунок 3.2 – Кінцевий вигляд корінного файлу

Далі треба додати БД до СКБД PostgreSQL. Верхній частині інтерфейсу тиснемо на «File» та «Create new». Після просто вказується назва БД та користувач, і БД з'являється у панелі зліва (рис. 3.4).

```

1  PORT=5000
2  DB_NAME=online-store
3  DB_USER=postgres
4  DB_PASSWORD=LwLr59BTec
5  DB_HOST=localhost
6  DB_PORT=5432
7  SECRET_KEY=cock

```

Рисунок 3.3 – Змінне оточення

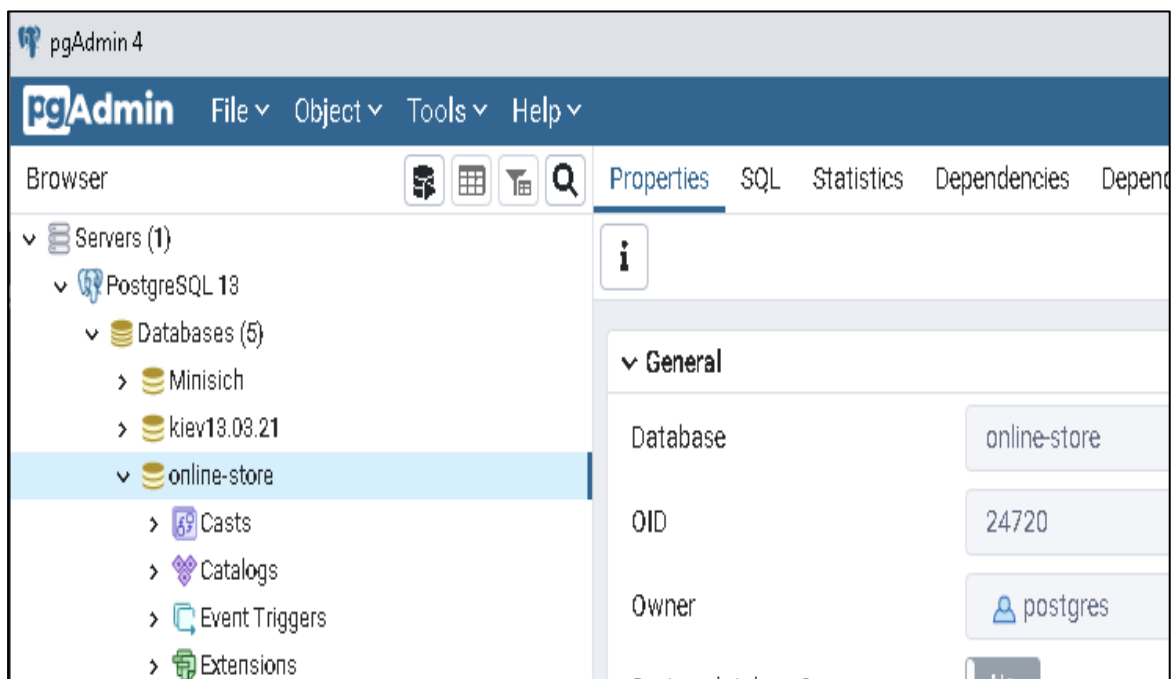


Рисунок 3.4 – Наша додана БД у СКБД

Далі проводиться підключення до бази даних. Імпортуємо у файлі БД клас `sequelize` та всі налаштування винесемо з змінного оточення. Передаються такі налаштування, як назву БД, користувача та паролі. Також вказується діалект `postgres`, і вказується хост та порт нашого додатку (рис. 3.5).

Далі потрібно реалізувати схему даних для нашої БД. Для цього треба описати всі моделі даних. Це потрібно для того, щоб визначати логічну структуру бази даних і принципово визначати спосіб збереження, організації

та обробки даних. У ORM БД передбачає об'єктно-орієнтовану архітектуру фактичної бази даних, використовуючи нові типи відомих та пов'язаних даних. Цей зв'язок встановлюється шляхом створення методів з використанням тригерів та збережених процедур. Розглянемо першу модель для прикладу (рис. 3.6).

```

1  const {Sequelize} = require(['sequelize'])
2
3  module.exports = new Sequelize (
4
5      process.env.DB_NAME,
6      process.env.DB_USER,
7      process.env.DB_PASSWORD,
8      {
9          dialect: 'postgres',
10         host: process.env.DB_HOST,
11         port: process.env.DB_PORT
12     }
13 )

```

Рисунок 3.5 – БД додатка

```

const User = sequelize.define('user', {
  id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
  email: {type: DataTypes.STRING, unique: true},
  password: {type: DataTypes.STRING},
  role: {type: DataTypes.STRING, defaultValue: "USER"},
})

```

Рисунок 3.6 – Модель користувача

Поле id має первісний ключ, який має властивість створювати завжди новий ідентифікаційний код. Пошта має строковий тип та має бути завжди унікальним. Пароль теж буде мати строковий тип, але вже не буде унікальною, бо користувачі можуть мати унікальні паролі. Далі іде роль користувача, яка за замовченням завжди буде звичайна, тобто без прав адміністратора.

Далі необхідно встановити зв'язок з усіма моделями. Наприклад користувач має один кошик, який йому належить (рис. 3.7).

```

User.hasOne(Basket)
Basket.belongsTo(User)

User.hasMany(Rating)
Rating.belongsTo(User)

Basket.hasMany(BasketDevice)
BasketDevice.belongsTo(Basket)

Type.hasMany(Device)
Device.belongsTo(Type)

Color.hasMany(Device)
Device.belongsTo(Color)

Device.hasMany(Rating)
Rating.belongsTo(Device)

Device.hasMany(BasketDevice)
BasketDevice.belongsTo(Device)

Device.hasMany(DeviceInfo, {as: 'info'});
DeviceInfo.belongsTo(Device)

Type.belongsToMany(Color, {through: TypeColor })
Color.belongsToMany(Type, {through: TypeColor })

```

Рисунок 3.7 – Зв'язок моделей між собою

Далі потрібно переконатись, що моделі внесені таблицями до СКБД, але перед цим треба імпортувати файл з моделями у корінний файл (рис. 3.8).

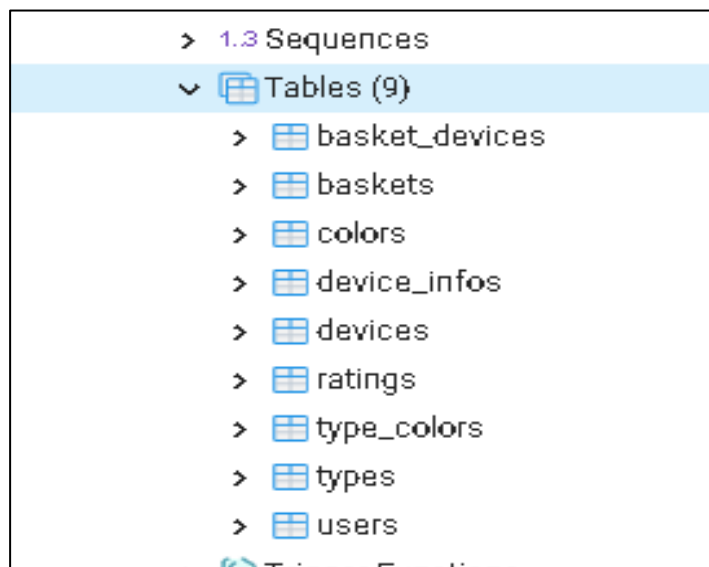


Рисунок 3.8 – Таблиці у СКБД

Маршрутизація визначає, як додаток відповідає на клієнтський запит до конкретної адреси. Шляхи маршрутів, в поєднанні з методом запиту,

визначають конкретні адреси (кінцеві точки), в яких можуть бути створені запити[15].

В основному маршруті нашого додатку буде отримано маршрут із express, далі створюється об'єкт цього маршруту і необхідно вказати під-маршрути (рис. 3.9).

```
const Router = require('express')
const router = new Router()
const deviceRouter = require('./deviceRouter')
const userRouter = require('./userRouter')
const colorRouter = require('./colorRouter')
const typeRouter = require('./typeRouter')

router.use('/user', userRouter)
router.use('/type', typeRouter)
router.use('/color', colorRouter)
router.use('/device', deviceRouter)

module.exports = router
```

Рисунок 3.9 – Маршрутизація

Кожен маршрут має використовувати різні типи HTTP запиту. Наприклад користувач буде мати два методи POST, це реєстрація та авторизація. Ще буде один метод GET для перевірки та обробки помилок через middleware (рис.3.10, 3.11).

```
router.post('/registration', userController.registration)
router.post('/login', userController.login)
router.get('/auth', authMiddleware, userController.check)
```

Рисунок 3.10 – Маршрути користувача

У контролері маршруту треба реалізувати реєстрацію і авторизацію. Це дозволить нам застосування токенів без стану, які можна додавати в чорні списки. У тілі реєстрації отримується запит пошти, паролю та роль з другорядним характером. Остання роль ніде не задається і тому її будуть отримувати теж у запити для створення окремих ролей користувача.

```
class UserController {
  Complexity is 7 It's time to do something...
  async registration (req, res) {
    const {email, password, role} = req.body
    if (!email || !password) {
      return next(ApiError.badRequest('Некорректний email або password'))
    }
    const candidate = await User.findOne({where: {email}})
    if (candidate) {
      return next(ApiError.badRequest('Користувач з таким email вже існує'))
    }
    const hashPassword = await bcrypt.hash(password, 5)
    const user = await User.create({email, role, password: hashPassword})
    const basket = await Basket.create({userId: user.id})
    const token = generateJwt(user.id, user.email, user.role)
    return res.json({token})
  }
}
```

Рисунок 3.11 – Контролер маршруту

В умовах перевіряється чи вірна пошта і пароль, введений він в принципи. Також перевіряємо чи реєструвався такий користувач раніше. Далі, якщо новий користувач заданий вірно ми можемо створити нового користувача. У першому параметрі захищуємо пароль, і потім створюється користувач з вказаними паролем та поштою. Далі для користувача створюється свій кошик та передається для нього ідентифікаційний номер користувача.

Далі визивається модуль JWT та передається у зазначені параметри (рис. 3.12).

Middleware – це функції, які мають доступ до об'єкта запиту (req), об'єкту відповіді (res) і до наступної функції тимчасової роботи в циклі "запит-відповідь" додатка. Наступна функція тимчасової роботи, як правило, позначається змінної next.

```
const hashPassword = await bcrypt.hash(password, 5)
const user = await User.create({email, role, password: hashPassword})
const basket = await Basket.create({userId: user.id})
const token = generateJwt(user.id, user.email, user.role)
return res.json({token})
```

Рисунок 3.12 – Реалізація JWT у контролері користувача

Функції тимчасової роботи можуть виконувати такі завдання:

- виконання будь-якого коду;
- внесення змін до об'єкти запитів і відповідей;
- завершення циклу "запит-відповідь";
- виклик наступної функції тимчасової роботи з стека.

Якщо поточна функція тимчасової роботи завершує цикл "запит-відповідь", вона повинна викликати next () для передачі управління наступної функції тимчасової роботи. В іншому випадку запит зависне. Далі розробляється middleware для маршрута «/ auth», вона буде обробляти тільки запит «http: // localhost: 3000 / auth» (рис. 3.13).

Далі реалізується клас ApiError. Конструктор параметрами буде приймати статус коду та відправляти повідомлення помилки клієнту (рис. 3.14).

```

class ApiError extends Error{
  constructor(status, message) {
    super();
    this.status = status
    this.message = message
  }

  static badRequest(message) {
    return new ApiError(404, message)
  }

  static internal(message) {
    return new ApiError(500, message)
  }

  static forbidden(message) {
    return new ApiError(403, message)
  }
}

```

Рисунок 3.13 – Клас ApiError

```

Complexity is 6 It's time to do something...
module.exports = function (req, res, next) {
  if (req.method === "OPTIONS") {
    next()
  }
  try {
    const token = req.headers.authorization.split(' ')[1]
    if (!token) {
      return res.status(401).json({message: "Не авторізований"})
    }
    const decoded = jwt.verify(token, process.env.SECRET_KEY)
    req.user = decoded
    next()
  } catch (e) {
    res.status(401).json({message: "Не авторізований"})
  }
};

```

Рисунок 3.14 – Middleware маршруту auth

3.2 Опис розробленої клієнтської частини

Після повністю розробленої серверної частини розробляється фронт, тобто обличчя нашого додатка. Для поліпшення розробки зовнішнього

вигляду сайту використовується бібліотека react-bootstrap. Це бібліотека React компонентів, підходящі для повторного використання, яка реалізує можливості популярного шаблону Bootstrap.

Спочатку треба створити та наповнити компоненти клієнтської частини (рис. 3.15). Це будуть сторінки інтернет-магазину (рис 3.16).

Далі на прикладі сторінки адміністратора розглянемо додаткові компоненти API цієї сторінки, які будуть створювати новий тип, бренд та пристрій через модальні вікна для інтернет-магазину (рис. 3.17, 3.18).

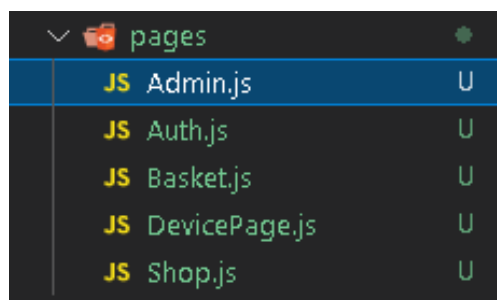


Рисунок 3.15 – Усі основні компоненти нашого додатка

```
return (
  <Container className="d-flex flex-column">
    <Button
      variant={"outline-dark"}
      className="mt-4 p-2"
      onClick={() => setTypeVisible(true)}
    >
      Додати тип
    </Button>
    <Button
      variant={"outline-dark"}
      className="mt-4 p-2"
      onClick={() => setColorVisible(true)}
    >
      Додати бренд
    </Button>
    <Button
      variant={"outline-dark"}
      className="mt-4 p-2"
      onClick={() => setDeviceVisible(true)}
    >
      Додати пристрій
    </Button>
    <CreateColor show={colorVisible} onHide={() => setColorVisible(false)} />
    <CreateDevice show={deviceVisible} onHide={() => setDeviceVisible(false)} />
    <CreateType show={typeVisible} onHide={() => setTypeVisible(false)} />
  </Container>
);
```

Рисунок 3.16 – Вигляд панелі адміністратора з використанням шаблонів Bootstrap

```

export const createType = async (type) => {
  const {data} = await $authHost.post('api/type', type)
  return data
}

export const fetchTypes = async () => {
  const {data} = await $host.get('api/type')
  return data
}

export const createColor = async (color) => {
  const {data} = await $authHost.post('api/color', color)
  return data
}

```

Рисунок 3.17 – Реалізація API панелі адміністратора -1

```

export const fetchColors = async () => {
  const {data} = await $host.get('api/color', )
  return data
}

export const createDevice = async (device) => {
  const {data} = await $authHost.post('api/device', device)
  return data
}

export const fetchDevices = async (typeId, colorId, page, limit= 5) => {
  const {data} = await $host.get('api/device', {params: {
    typeId, colorId, page, limit
  }})
  return data
}

export const fetchOneDevice = async (id) => {
  const {data} = await $host.get('api/device/' + id)
  return data
}

```

Рисунок 3.18 – Реалізація API панелі адміністратора -2

3.3 Висновки за розділом 3

У цьому розділі було описано основні рішення щодо розробки інтернет-магазину. Описано модулі та компоненти серверної та клієнтської частини, реалізацію основних модулів додатку, опис розроблених і використаних у ході роботи методів. Представлено реалізацію Rest API, JWT та діорам баз даних для реляційного відображення.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ПРОГРАМИ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосування

Розроблений додаток представляє собою простий інтернет-магазин з каталогом товару та розподілом даних. Також була використана об'єктно-реляційна база даних, яка відображає інформацію в найбільш простий для користувача формі та дозволяє маніпулювати даними на рівні вихідний БД, і надає можливість зміни.

4.2 Умови запуску

Для використання вебдодатку необхідно мати встановлене програмне забезпечення, таке як Node.js, PostgreSQL v13. Також комп'ютер повинен мати:

- ОС Windows 7 x64 або новіші;
- ОЗУ 2Гб;
- доступ до інтернету;
- процесор приблизно 3 ГГц.

Для роботи вебдодатка потрібно приблизно 500 МБ вільного місця в оперативній пам'яті.

4.3 Робота з сайтом

Після запуску серверу і додатка інтернет-магазину користувачі мають доступ до додатка. Спочатку відкривається головна сторінка, вона ж і сторінка з товарами (рис. 4.1).

З головної сторінки відкривається інтерфейс сайту. Звідси користувач потрапляє до сторінки авторизації та реєстрації (рис. 4.2).

Після того, як користувач буде авторизований та отримує свій токен, він буде відправлений знову на головну сторінку. Користувач має права адміністратора і тому він може наповнювати товари в магазині через панель адміністратора (рис. 4.3).

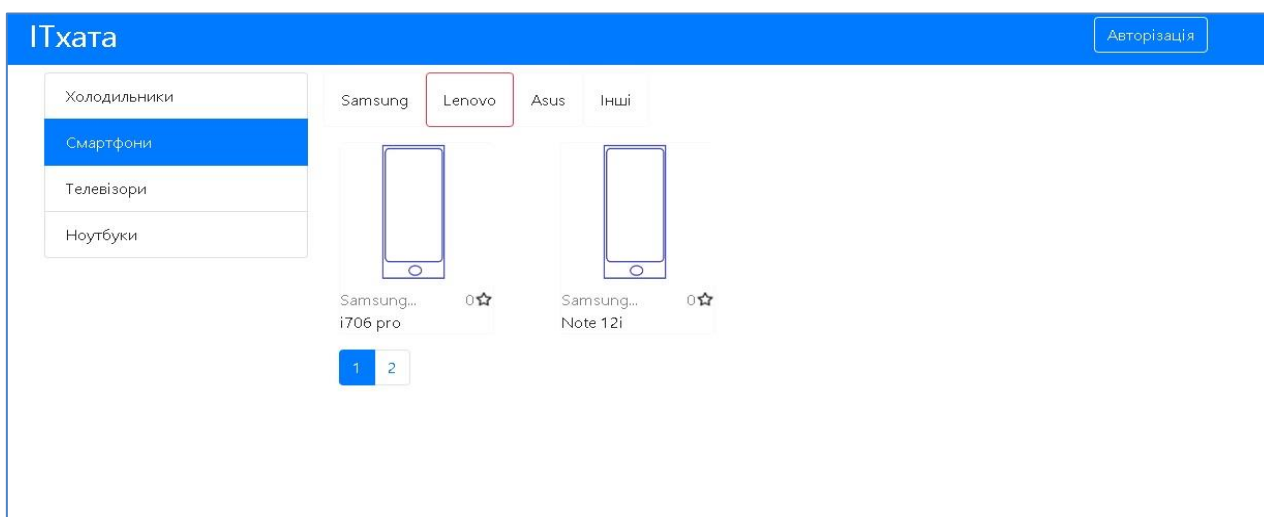


Рисунок 4.1 – Головна сторінка додатка

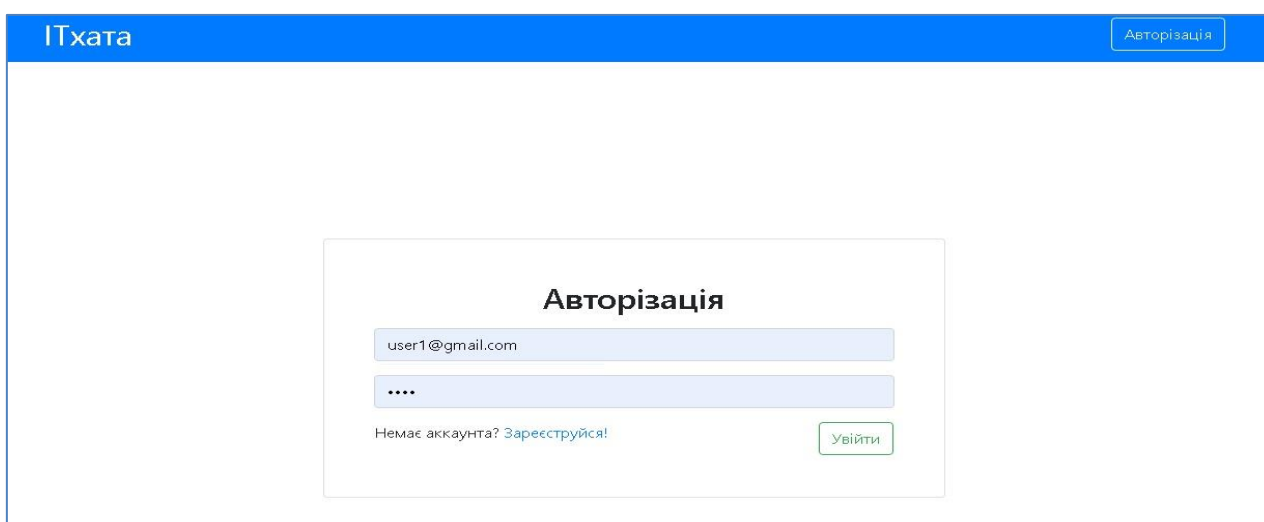


Рисунок 4.2 – Сторінка авторизації

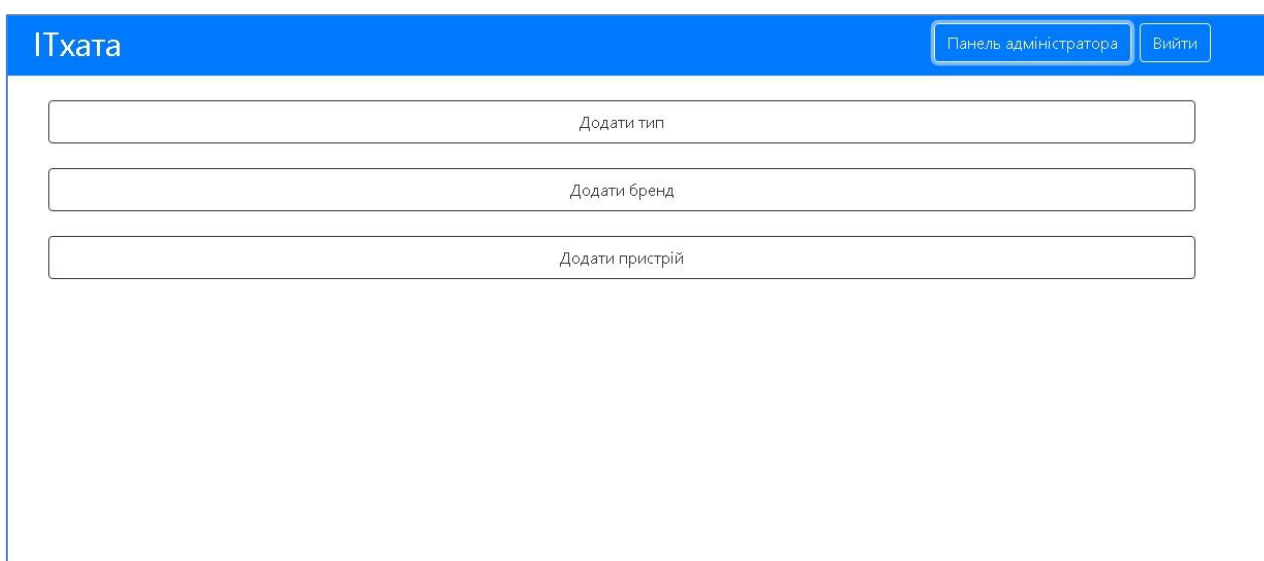
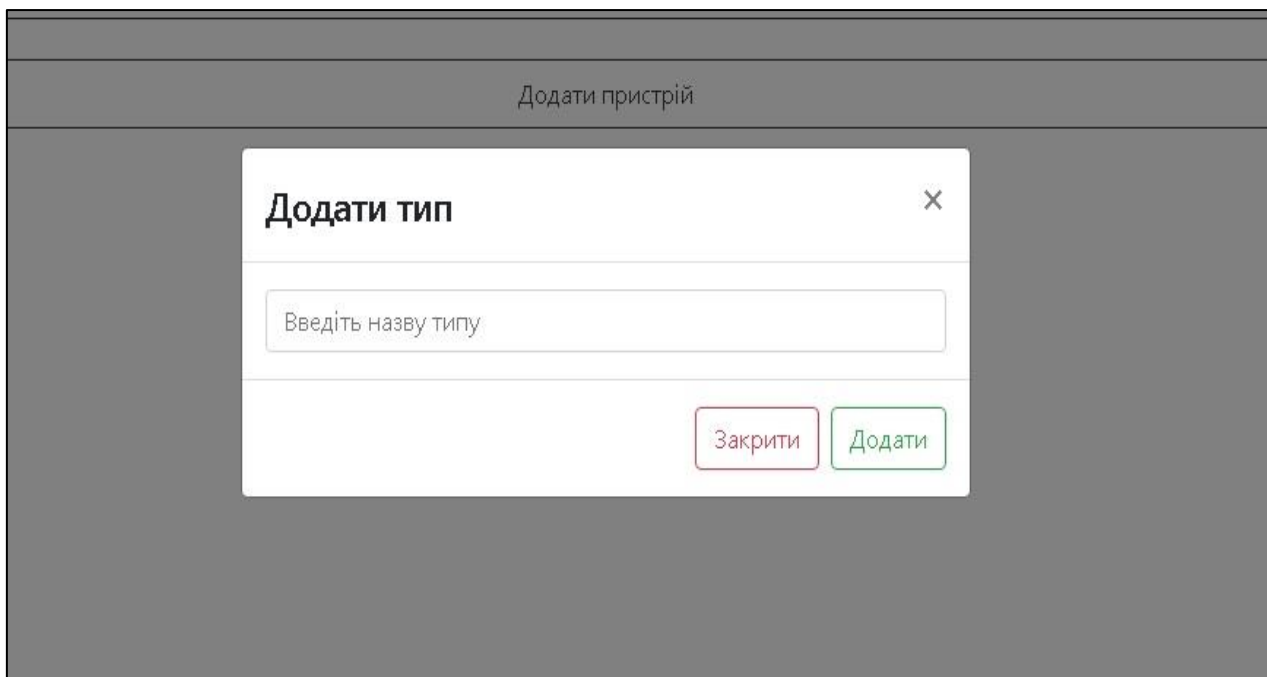


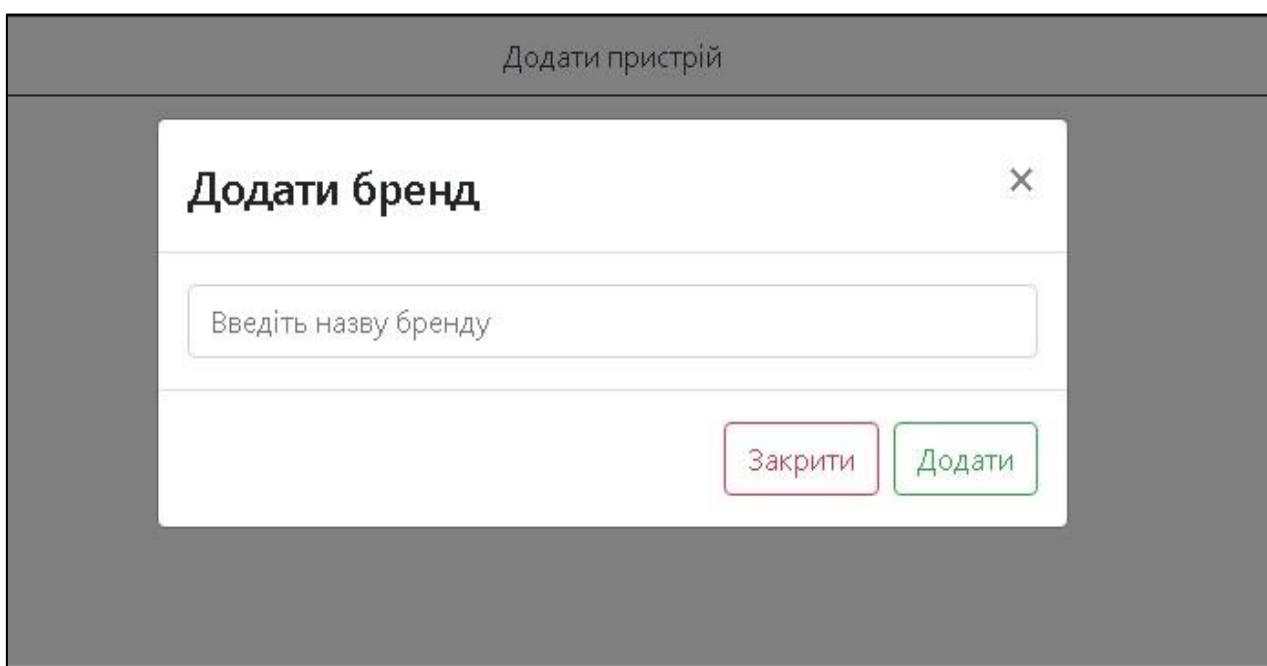
Рисунок 4.3 – Панель адміністратора

Панель адміністратора може додавати через спливаючі модулі нові типи (рис. 4.4) та бренди (рис. 4.5) до інтернет магазину. Також вона додає нові пристрої до магазину з вказанням ціни, типу, бренду, зображення та характеристик (рис. 4.6).



The screenshot shows a web application interface. At the top, there is a dark gray header bar with the text "Додати пристрій" (Add device) in white. Below this, a modal window titled "Додати тип" (Add type) is displayed. The modal has a close button (X) in the top right corner. Inside the modal, there is a text input field with the placeholder text "Введіть назву типу" (Enter the type name). At the bottom right of the modal, there are two buttons: "Закрити" (Close) in a red box and "Додати" (Add) in a green box.

Рисунок 4.4 – Додавання типу



The screenshot shows the same web application interface as Figure 4.4. The modal window is now titled "Додати бренд" (Add brand). It has a close button (X) in the top right corner. Inside the modal, there is a text input field with the placeholder text "Введіть назву бренду" (Enter the brand name). At the bottom right of the modal, there are two buttons: "Закрити" (Close) in a red box and "Додати" (Add) in a green box.

Рисунок 4.5 – Додавання бренду

Рисунок 4.6 – Додавання пристрою

Через головну сторінку користувачі потрапляють на сторінки пристроїв вказуючи через навігаційну панель ті бренди та типи товару, якими цікавляться користувачі. Після переходу на сторінку товару ознайомлюються з його характеристиками та рейтингом (рис. 4.7). Також через сторінку товару додається товар до кошика. Коли додається товар до кошика, то користувач потрапляє на головну сторінку (рис. 4.8), а в інтерфейсі зверху з'являється можливість потрапити до кошика (рис. 4.9).

Характеристики	
Діагональ екрана:	6.65
Ємність акумулятора:	5000 мА·год
Оперативна пам'ять:	12 ГБ
Кількість мегапікселів основної камери:	64 Мп + 16 Мп

Рисунок 4.7 – Сторінка товару

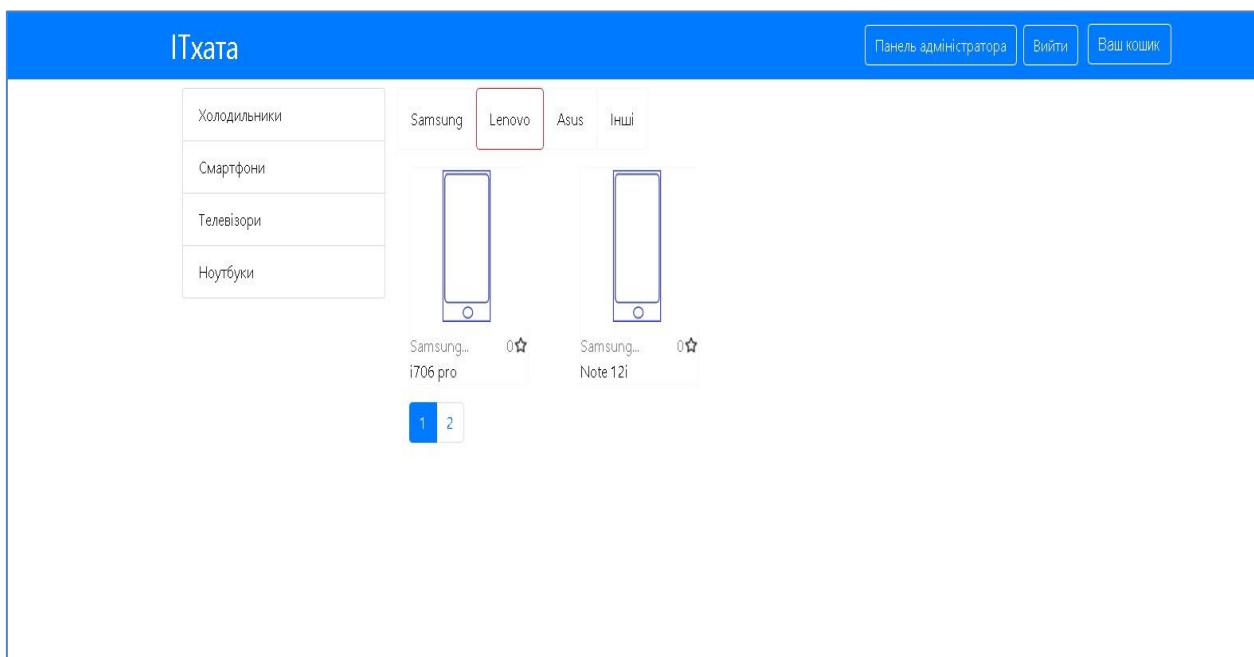


Рисунок 4.8 – З’являється кошик на головній сторінці

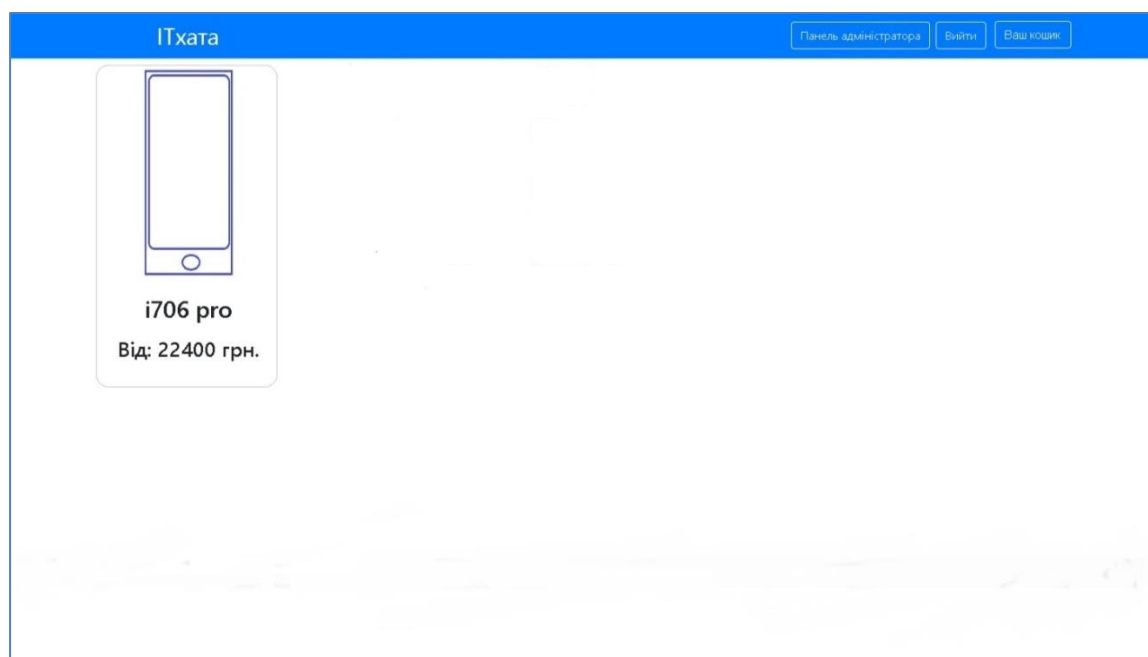


Рисунок 4.9 – Кошик користувача з товарами

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було створено інтернет-магазин та реалізовано Rest API, авторизацію по JWT та базу даних для реляційного відображення. Було розв'язано такі завдання:

- аналіз предметної області;
- розробка технічного завдання;
- опис обраних технологій;
- визначення структури вебсайту з побудовою відповідної структурної схеми;
- розробка інтернет-магазину;
- тестування розробленого додатку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Інтернет-тенденції. Статистика та факти в усьому світі [Електрон. ресурс] – Режим доступу: <https://uk.vpnmentor.com/blog>.
2. Створення інтернет-магазину [Електрон. ресурс] – Режим доступу: <https://koloro.ua/ua/sozdanie-internet-magazina.html>.
3. Царьов, Р. Ю. Електронна комерція: навчальний посібник з підготовки бакалаврів / Царьов Р. Ю. - Одеса: ОНАЗ ім. О. С. Попова, 2010. – 112 с.
4. Система управління сайтом [Електрон. ресурс] – Режим доступу: <https://web24.pro/rozrobka-sajtiv-blog/systema-upravlinnya-sajtom>.
5. Створення інтернет-магазину [Електрон. ресурс] – Режим доступу: <https://koloro.ua/ua/services.html>.
6. Юзабилити Тестирование Сайта [Электрон. ресурс] – Режим доступа: <https://www.plerdy.com/ru/blog/what-is-website-usability-testing/>.
7. Круг, С. Веб-дизайн: книга Стива Круга или «не заставляйте меня думать!». 2-е издание. / Круг С. - СПб.: Символ-Плюс, 2008. – 214 с.
8. ROZETKA//ЛігаДосьє 2020. [Электрон. ресурс] – Режим доступа: <https://file.liga.net/companies/rozetka>.
9. Інтернет-супермаркет РОЗЕТКА (rozetka.com.ua) – відгуки [Электрон. ресурс].– Режим доступа: <https://irecommend.ru/content/internet-supermarket-rozetka-httprozetkacomua>.
10. ECMAScript 6 [Electronic resource]. – Access mode: <https://rauchg.com/2015/ecmascript-6>.
11. Майк Кантелон, Марк Хартер, Т. Дж. Головайчук та Натан Райліх. Node.js в действии. 2-е изд./ М. Кантелон, М. Хартер, Т. Дж. Головайчук, Н. Райліх — СПб.: Питер, 2018. – 432 с.
12. Express4.17.1 Fast, unopinionated, minimalist web framework for Node.js [Electronic resource]. – Access mode: <https://expressjs.com/>.
13. What is PostgreSQL? [Electronic resource]. – Access mode:

<https://www.postgresql.org/about/>.

14. Sequelize [Electronic resource]. – Access mode: <https://metanit.com/web/nodejs/9.1.php>.

15. Mapping Objects to Relational Databases: O/R Mapping In Detail [Electronic resource]. – Access mode: <http://www.agiledata.org/essays/mappingObjects.html#MappingRelationships>.

16. Routing [Electronic resource]. – Access mode: <http://expressjs.com/en/guide/routing.html>.

ДОДАТОК А
Технічне завдання

Вступ

Інтернет-магазин «ІТхата» може вільно використовуватися фахівцям для особистого користування, з метою полегшення розробки типового комерційного додатку.

A.1 Підстави для розробки

Підставою для розробки програмного забезпечення є завдання на дипломну кваліфікаційну роботу бакалавра на тему: «Розробка інтернет магазину з використанням об'єктно-реляційного відображення», що затверджене наказом № 103 від 30 березня 2021 р. по Національному університету «Запорізька політехніка».

A.2 Призначення розробки

Функціональне призначення: додаток дозволяє вести облік пристроїв, додавати нові пристрої, типи і бренди пристроїв, реєструватись користувачам.

A.3 Вимоги до програми чи програмному виробу

A.3.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати:

- виведення списку товарів;
- пошук по категоріям пристроїв;
- реєструвати нових користувачів;
- введення інформації про конкретний пристрій;
- можливість додати пристрій у особистий кошик користувача;
- відображення рейтингу пристрою;
- виведення категорій пристрою.

A.3.2 Вимоги до надійності

Надійне функціонування додатку має забезпечуватися наступними факторами:

- джерело безперебійного живлення апаратно-технічних засобів;
- необхідний рівень підготовки обслуговуючого персоналу.

A.3.3 Умови експлуатації

Для забезпечення експлуатації програмного забезпечення необхідно встановити наступні програмні застосунки:

- Node.js;
- PostgreSQL;
- ОС: Windows;
- Браузер Google Chrome.

A.3.4 Вимоги до складу й параметрів технічних засобів

Мінімальні системні вимоги:

- ОС Windows 7 x64 або новіші;
- ОЗУ 1ГБ;
- доступ до інтернету;
- процесор приблизно 3 GHz.

A.3.5 Вимоги маркування й упакування

Вимоги до транспортування та зберігання схожі на носій, на якому зберігається програма, тобто оптичний, дискета або жорсткий диск.

A.3.6 Вимоги транспортування й збереження

Вимоги щодо доставки та зберігання точно такі ж, як вимоги щодо доставки та зберігання цифрових багатоцільових дисків.

A.4 Стадії та етапи розробки

Для роботи над дипломною кваліфікаційною роботою бакалавра заплановано виконання наступних етапів:

- аналіз проблеми;
- проєктування програмного забезпечення;
- розробка програмного забезпечення;
- тестування і налаштування програмного забезпечення;
- оформлення документації.

A.5 Порядок контролю й приймання

Дипломна кваліфікаційна робота бакалавра повинна бути виконана у відповідності з календарним планом. Дипломна кваліфікаційна робота допускається до захисту керівником, нормоконтролером, затверджується завідувачем кафедри і захищається в присутності членів екзаменаційної комісії.

ДОДАТОК Б
Текст програми

Б.1 Текст файла index.js

```

require('dotenv').config()
const express = require ('express')
const sequelize = require('./db')
const models = require('./models/models')
const cors = require('cors')
const fileUpload = require('express-fileupload')
const router = require('./routes/index')
const errorHandler = require('./middleware/ErrorHandlerMiddleware')

const path = require('path')
const PORT = process.env.PORT || 5000

const app = express ()
app.use(cors())
app.use(express.json())
app.use(express.static(path.resolve(__dirname, 'static')))
app.use(fileUpload({}))
app.use('/api', router)

app.use(errorHandler)

const start = async () => {
  try {
    await sequelize.authenticate()
    await sequelize.sync()
    app.listen(PORT, () => console.log(`Server started on port ${PORT}`))
  } catch (e) {
    console.log(e)
  }
}

start()

```

Б.2 Текст файла db.js

```

const {Sequelize} = require ('sequelize')

module.exports = new Sequelize (

  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  {

```

```

        dialect: 'postgres',
        host: process.env.DB_HOST,
        port: process.env.DB_PORT
    }
)

```

Б.3 Текст файла deviceRouter.js

```

const Router = require('express')
const router = new Router()
const deviceController = require('../controllers/deviceController')

router.post('/', deviceController.create)
router.get('/', deviceController.getAll)
router.get('/:id', deviceController.getOne)

module.exports = router

```

Б.4 Текст файла deviceController.js

```

const uuid = require('uuid')
const path = require('path');
const {Device, DeviceInfo} = require('../models/models')
const ApiError = require('../error/ApiError');

class DeviceController {
  async create (req, res, next) {
    try {

      let { name, price, colorId, typeId, info} = req.body
      const {img} = req.files
      let fileName = uuid.v4() + ".jpg"
      img.mv(path.resolve(__dirname, '..', 'static', fileName))
    }

    const device = await Device.create({name, price, colorId, typeId, img: fileName});

    if (info) {
      info = JSON.parse(info)
      info.forEach(i =>
        DeviceInfo.create({
          title: i.title,
          description: i.description,
          deviceId: device.id
        })
      )
    }
  }
}

```

```

        return res.json(device)

    } catch (error) {
        next(ApiError.badRequest(e.message))
    }

}

async getALL (req, res) {
    let {colorId, typeId, limit, page} = req.query
    page = page || 1
    limit = limit || 9
    let offset = page * limit - limit
    let devices;
    if (!colorId && !typeId) {
        devices = await Device.findAndCountAll({limit, offset})
    }
    if (colorId && !typeId) {
        devices = await Device.findAndCountAll({where:{colorId}, limit, offset})
    }
    if (!colorId && typeId) {
        devices = await Device.findAndCountAll({where:{typeId}, limit, offset})
    }
    if (colorId && typeId) {
        devices = await Device.findAndCountAll({where:{typeId, colorId}, limit, offset})
    }
    return res.json(devices)
}

async getOne(req, res) {
    const {id} = req.params
    const device = await Device.findOne(
        {
            where: {id},
            include: [{model: DeviceInfo, as: 'info'}]
        },
    )
    return res.json(device)
}

}

module.exports = new DeviceController()

```

Б.5 Текст файла models.js

```

const sequelize = require('../db')
const {DataTypes} = require('sequelize')

```

```

const User = sequelize.define('user', {
  id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
  email: {type: DataTypes.STRING, unique: true},
  password: {type: DataTypes.STRING},
  role: {type: DataTypes.STRING, defaultValue: "USER"},
})

const Basket = sequelize.define('basket', {
  id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
})

const BasketDevice = sequelize.define('basket_device', {
  id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
})

const Device = sequelize.define('device', {
  id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
  name: {type: DataTypes.STRING, unique: true, allowNull: false},
  price: {type: DataTypes.INTEGER, allowNull: false},
  rating: {type: DataTypes.INTEGER, defaultValue: 0},
  img: {type: DataTypes.STRING, allowNull: false},
})

const Type = sequelize.define('type', {
  id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
  name: {type: DataTypes.STRING, unique: true, allowNull: false},
})

const Color = sequelize.define('color', {
  id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
  name: {type: DataTypes.STRING, unique: true, allowNull: false},
})

const Rating = sequelize.define('rating', {
  id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
  rate: {type: DataTypes.INTEGER, allowNull: false},
})

const DeviceInfo = sequelize.define('device_info', {
  id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},

```

```

    title: {type: DataTypes.STRING, allowNull: false},
    description: {type: DataTypes.STRING, allowNull: false},
  })

const TypeColor = sequelize.define('type_color', {
  id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
})

User.hasOne(Basket)
Basket.belongsTo(User)

User.hasMany(Rating)
Rating.belongsTo(User)

Basket.hasMany(BasketDevice)
BasketDevice.belongsTo(Basket)

Type.hasMany(Device)
Device.belongsTo(Type)

Color.hasMany(Device)
Device.belongsTo(Color)

Device.hasMany(Rating)
Rating.belongsTo(Device)

Device.hasMany(BasketDevice)
BasketDevice.belongsTo(Device)

Device.hasMany(DeviceInfo, {as: 'info'});
DeviceInfo.belongsTo(Device)

Type.belongsToMany(Color, {through: TypeColor })
Color.belongsToMany(Type, {through: TypeColor })

module.exports = {
  User,
  Basket,
  BasketDevice,
  Device,
  Type,
  Color,
  Rating,
  TypeColor,
  DeviceInfo
}
```

Б.6 Текст файла App.js

```
import React, {useContext, useEffect, useState} from 'react';
import {BrowserRouter} from "react-router-dom";
import AppRouter from "../components/AppRouter";
import NavBar from "../components/NavBar";
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import {check} from "../http/userAPI";
import {Spinner} from "react-bootstrap";

const App = observer(() => {
  const {user} = useContext(Context)
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    check().then(data => {
      user.setUser(true)
      user.setIsAuth(true)
    }).finally(() => setLoading(false))
  }, [])

  if (loading) {
    return <Spinner animation={"grow"}/>
  }

  return (
    <BrowserRouter>
      <NavBar />
      <AppRouter />
    </BrowserRouter>
  );
});

export default App;
```

Б.7 Текст файла DeviceStore.js

```
import {makeAutoObservable} from "mobx";

export default class DeviceStore {
  constructor() {
    this._types = [

    ]
    this._colors = [

    ]
  }
}
```

```

        this._devices = [

            ]
        this._selectedType = {}
        this._selectedColor = {}
        this._page = 1
        this._totalCount = 0
        this._limit = 3
        makeAutoObservable(this)
    }

    setTypes(types) {
        this._types = types
    }
    setColors(colors) {
        this._colors = colors
    }
    setDevices(devices) {
        this._devices = devices
    }

    setSelectedType(type) {
        this.setPage(1)
        this._selectedType = type
    }
    setSelectedColor(color) {
        this.setPage(1)
        this._selectedColor = color
    }
    setPage(page) {
        this._page = page
    }
    setTotalCount(count) {
        this._totalCount = count
    }

    get types() {
        return this._types
    }
    get colors() {
        return this._colors
    }
    get devices() {
        return this._devices
    }
    get selectedType() {
        return this._selectedType
    }
    get selectedColor() {
        return this._selectedColor
    }
}

```

```

    get totalCount() {
      return this._totalCount
    }
    get page() {
      return this._page
    }
    get limit() {
      return this._limit
    }
  }
}

```

Б.8 Текст файлу DevicePage.js

```

import React, {useEffect, useState} from 'react';
import {Button, Card, Col, Container, Image, Row} from "react-
bootstrap";
import bigStar from '../assets/bigStar.png'
import {useParams} from 'react-router-dom'
import {fetchOneDevice} from "../http/deviceAPI";

const DevicePage = () => {
  const [device, setDevice] = useState({info: []})
  const {id} = useParams()
  useEffect(() => {
    fetchOneDevice(id).then(data => setDevice(data))
  }, [])

  return (
    <Container className="mt-3">
      <Row>
        <Col md={4}>
          <Image width={300} height={300} src={process
.env.REACT_APP_API_URL + device.img}/>
        </Col>
        <Col md={4}>
          <Row className="d-flex flex-column align-
items-center">
            <h2>{device.name}</h2>
            <div
              className="d-flex align-items-
center justify-content-center"
              style={{background: `url(${bigStar})
no-
repeat center center`, width:240, height: 240, backgroundSize: '
cover', fontSize:64}}
            >
              {device.rating}
            </div>
          </Row>
        </Col>
      </Row>
    </Container>
  )
}

```

```

        <Col md={4}>
          <Card
            className="d-flex flex-column align-items-center justify-content-around"
            style={{width: 300, height: 300, fontSize: 32, border: '5px solid lightgray'}}
          >
            <h3>Від: {device.price} грн.</h3>
            <Button variant={"outline-dark"}>Додати до кошика</Button>
          </Card>
        </Col>
      </Row>
      <Row className="d-flex flex-column m-3">
        <h1>Характеристики</h1>
        {device.info.map((info, index) =>
          <Row key={info.id} style={{background: index % 2 === 0 ? 'lightgray' : 'transparent', padding: 10}}>
            {info.title}: {info.description}
          </Row>
        )}
      </Row>
    </Container>
  );
};

export default DevicePage;

```

Б.9 Текст файлу deviceAPI.js

```

import {$authHost, $host} from "../index";

export const createType = async (type) => {
  const {data} = await $authHost.post('api/type', type)
  return data
}

export const fetchTypes = async () => {
  const {data} = await $host.get('api/type')
  return data
}

export const createColor = async (color) => {
  const {data} = await $authHost.post('api/color', color)
  return data
}

export const fetchColors = async () => {
  const {data} = await $host.get('api/color', )

```

```
        return data
    }

    export const createDevice = async (device) => {
        const {data} = await $authHost.post('api/device', device)
        return data
    }

    export const fetchDevices = async (typeId, colorId, page, limit=
5) => {
        const {data} = await $host.get('api/device', {params: {
            typeId, colorId, page, limit
        }})
        return data
    }

    export const fetchOneDevice = async (id) => {
        const {data} = await $host.get('api/device/' + id)
        return data
    }
}
```

ДОДАТОК В
Слайди презентації

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ

Дипломна кваліфікаційна робота бакалавра за темою «Веброзробка інтернет-магазину з використанням об'єктно-реляційного відображення»

Підготував: Зосенко Кирило Віталійович, ст. гр. КНТ-127

Керівник: Зайко Тетяна Анатоліївна, к.т.н, доцент НУ"ЗП"

2021

Рисунок В.1 – Слайд 1

Об'єкт, предмет дослідження, та мета роботи

Об'єкт дослідження - процес реалізації вебсайтів інтернет-магазинів.

Предмет дослідження - методи розробки вебсайтів інтернет-магазинів з використанням JavaScript, Node.js та React.js.

Мета роботи - розробка інтернет-магазину електроніки з реалізацією клієнтської і серверної частини з додаванням технології ORM.

Рисунок В.2 – Слайд 2

Завдання до роботи

В процесі виконання роботи були поставленні такі задачі:

- аналіз предметної області
- розробка технічного завдання
- опис обраних технологій
- визначення структури вебсайту з побудовою відповідної структурної схеми.
- розробка інтернет-магазину
- тестування розробленого додатку

Рисунок В.3 – Слайд 3

Вибір мови програмування

JavaScript - мультіпарадігменна мова програмування. Підтримує об'єктно-орієнтована, імперативний і функціональний стилі. JS входить в перелік найпопулярніших мов програмування, яка використовується для розробки вебсайтів. Тому вона зустрічається майже всюди. JS часто використовується для створення анімації, завантаження нових зображень, сценаріїв або об'єктів на веб-сторінки та створення гнучкого інтерфейсу користувача.

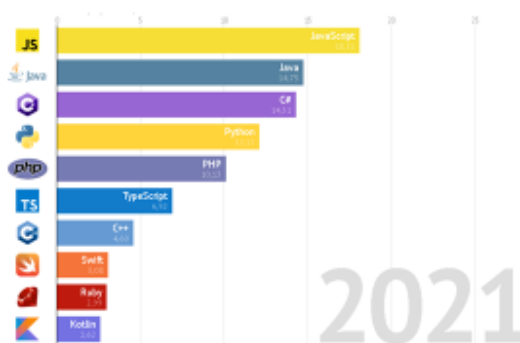


Рисунок В.4 – Слайд 4

Використанні технології при розробці

Для реалізації проєкту були обрані наступні технології:

- node.js;
- express;
- sequelize;
- postgresql;
- react.js;
- bootstrap.

Рисунок В.5 – Слайд 5

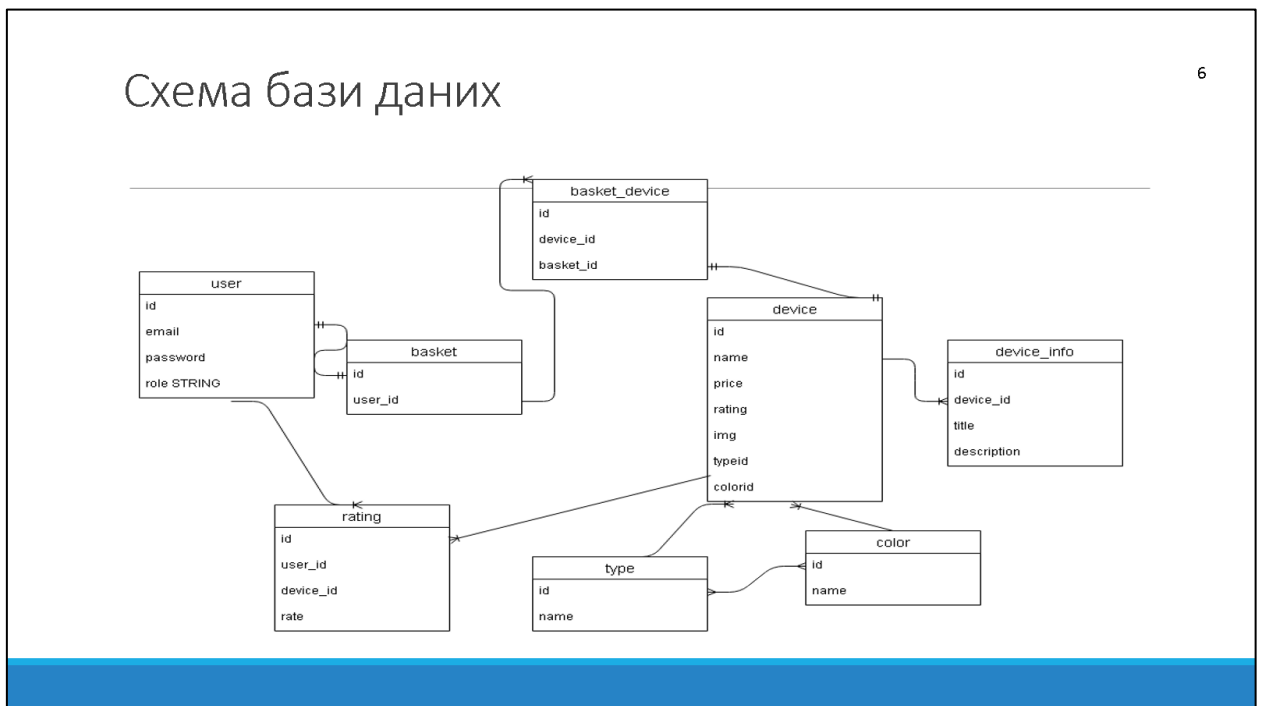


Рисунок В.6 – Слайд 6

Структура роботи вебсайту

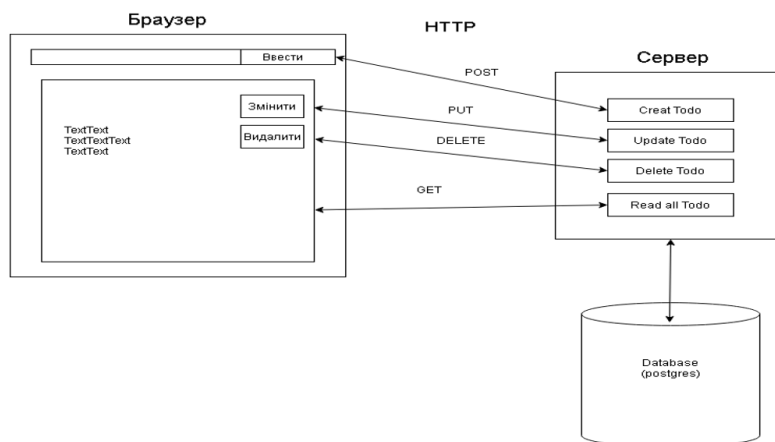


Рисунок В.7 – Слайд 7

Головна сторінка та сторінка товару

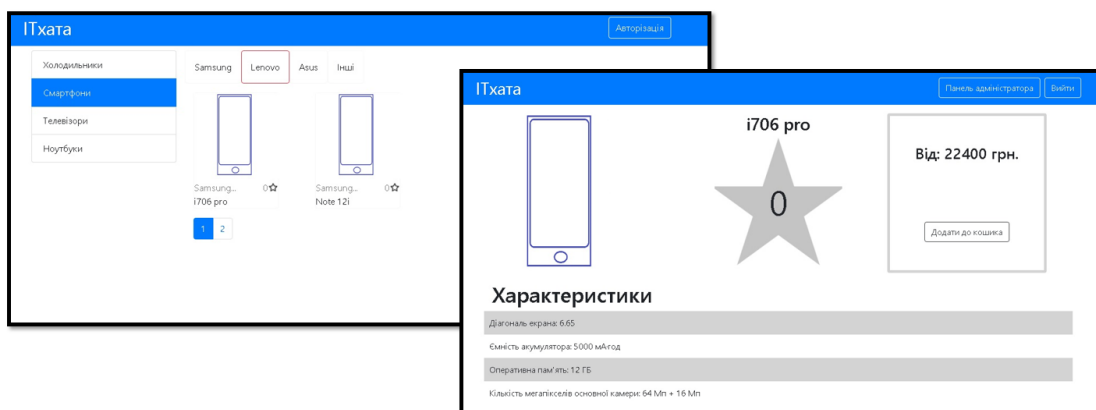


Рисунок В.8 – Слайд 8

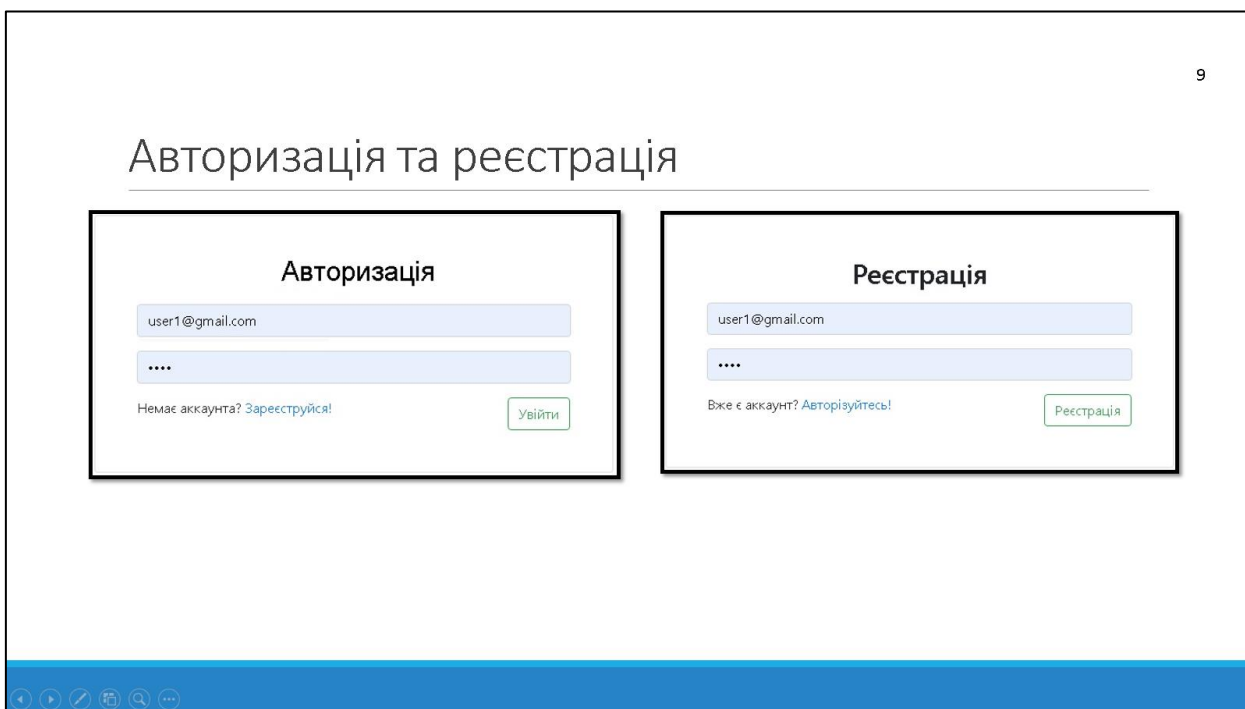


Рисунок В.9 – Слайд 9

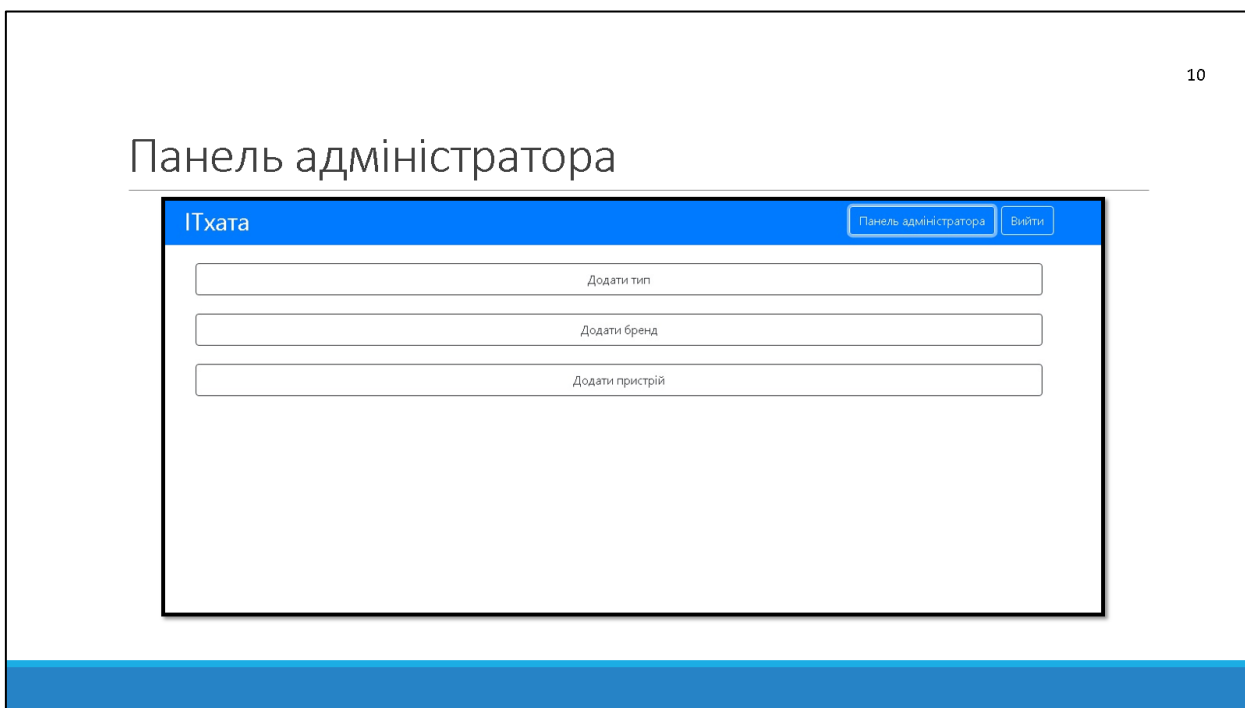


Рисунок В.10 – Слайд 10

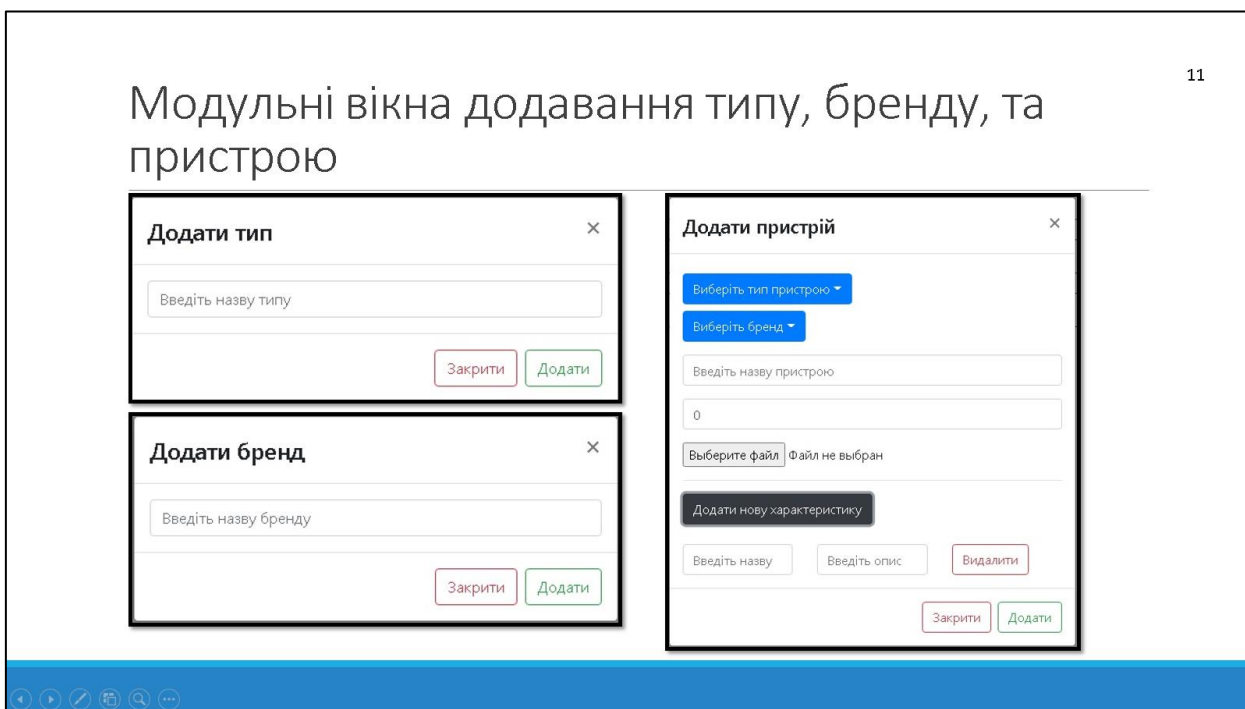


Рисунок В.11 – Слайд 11

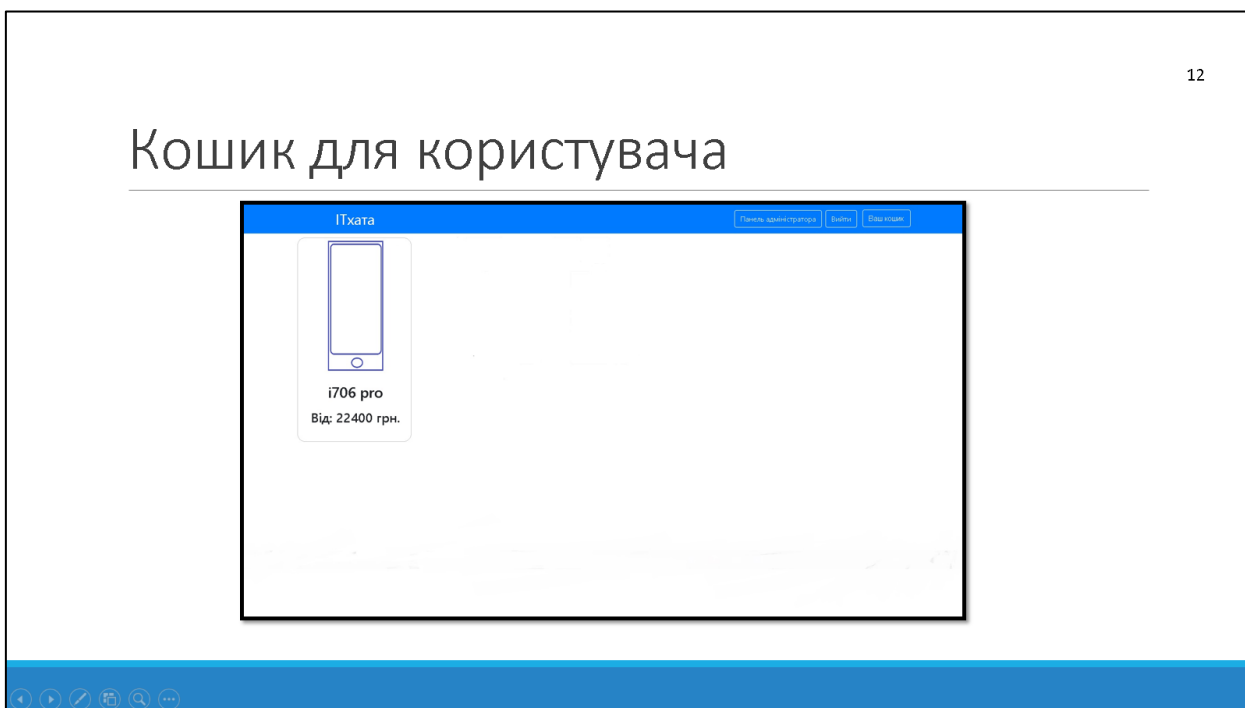


Рисунок В.12 – Слайд 12

Висновки

Під час виконання кваліфікаційної роботи бакалавра було створено інтернет-магазин та реалізовано Rest API, авторизацію по JWT та базу даних для реляційного відображення. Було розв'язано такі завдання:

- аналіз предметної області;
- розробка технічного завдання;
- опис обраних технологій ;
- визначення структури вебсайту з побудовою відповідної структурної схеми;
- розробка інтернет-магазину;
- тестування розробленого додатку.



Рисунок В.13 – Слайд 13