

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 комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ ПОБУТОВОЇ
ТЕХНІКИ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ REACTJS ТА
NODEJS

Виконав: студент(ка) 4 курсу, групи КНТ-527
Спеціальності 123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерна інженерія

Сіренко В.В.

(прізвище та ініціали)

Керівник Касьян К.М.

(прізвище та ініціали)

Рецензент Морщавка С.В.

(прізвище та ініціали)

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

Інститут, факультет ІРЕ, ФКНТ
 Кафедра «Комп'ютерні системи та мережі»
 Ступінь вищої освіти бакалаврський
 Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
 Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри Кудерметов Р.К.
 « » 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Сіренко Володимира Володимировича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка інтернет-магазину побутової техніки за допомогою технологій ReactJS та NodeJS.

керівник проєкту (роботи) Касьян Костянтин Миколайович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «17» березня 2021 року № 81

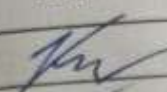
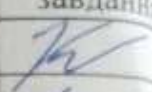
2. Строк подання студентом проєкту (роботи) 5 травня 2021 року

3. Вихідні дані до проєкту (роботи) веб-додаток повинен мати усі доступні товари у базі, меню з категоріями товарів, кошик для товарів, меню фільтрів при виборі категорії, пошуковий рядок, оформлення замовлення

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та вибір технологій для розробки. 2. Проектування системи. 3. Програмна реалізація та тестування системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) огляд аналогів інтернет-магазину побутової техніки, структура бази даних веб-додатку, алгоритм роботи веб-додатку, інтерфейс веб-додатку

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконання завдання
1-3 Основна частина	Касьян К.М., доцент		
Нормоконтроль	Щербак Н.В., ст. викладач		

7. Дата видачі завдання "28" лютого 2021 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

Керівник проєкту (роботи)


(підпис)


(підпис)

Сіренко В.В.
(прізвище та ініціали)

Касьян К.М.
(прізвище та ініціали)

РЕФЕРАТ

ПЗ: 64 с., 26 рис., 1 табл., 22 лістинги., 18 джерел.

INTELLIJ IDEA, NODEJS, REACTJS, ВЕБ-ДОДАТОК, ІНТЕРНЕТ-МАГАЗИН, КОМПОНЕНТ, ПОЯСНЮВАЛЬНА ЗАПИСКА

Об'єктом дослідження є процеси пошуку та оформлення замовлення необхідних товарів у інтернет-магазині побутової техніки за допомогою веб-додатку.

Метою даної роботи є розробка та програмна реалізація інтернет-магазину побутової техніки за допомогою технологій ReactJS та NodeJS. Веб-додаток розробляється для вдосконалення системи продажу товарів через мережу інтернет. Вдосконалення включає в себе: зручний та зрозумілий інтерфейс, приємний дизайн, швидку роботу як клієнтської частини додатку, так і серверної.

Для досягнення мети, в роботі були вирішені наступні задачі:

- розглянуто існуючі методи розробки веб-додатку;
- розроблена клієнтська частина веб-додатку;
- розроблена серверна частина веб-додатку;
- створена база даних інтернет-магазину побутової техніки.

В результаті роботи здійснена програмна реалізація інтернет магазину побутової техніки в середовищі IntelliJ IDEA, що призначений для здійснення онлайн покупок через середовище інтернет.

Дана розробка призначена для застосування в бізнес галузі.

ЗМІСТ

Скорочення та умовні позначки	5
Вступ.....	6
1 Аналіз предметної області та вибір технологій для розробки	7
1.1 Аналіз предметної області	7
1.2 Аналіз веб-додатків аналогів	9
1.3 Реляційні бази SQL	11
1.4 Нереляційні бази NoSQL	12
1.5 Порівняння SQL та NoSQL	14
1.6 Огляд існуючих реляційних субд.....	15
1.7 Порівняння PostgreSQL, SQLite.....	17
1.8 Огляд мов програмування	18
1.9 Огляд існуючих фреймворків	20
1.10 Порівняння та вибір фреймворку	22
1.11 Вибір засобів розробки серверної частини.....	24
1.12 Середовище розробки	26
2 Проектування системи.....	26
2.1 Загальний опис	26
2.2 Моделювання сценаріїв роботи.....	27
2.3 Діаграми послідовностей	28
2.4 Проектування графічного інтерфейсу користувача	31
3 Програмна реалізація та тестування системи	33
3.1 Клієнтська частина.....	33
3.2 База даних	44
3.3 Серверна частина	47
3.4 Тестування розробленої системи.....	56
Висновки	62
Перелік джерел посилання	63

Перелік графічних матеріалів:

Алгоритм 1 - Алгоритм роботи веб-додатку;

Плакат 1 - Огляд аналогів інтернет-магазину побутової техніки;

Плакат 2 - Структура бази даних веб-додатку;

Плакат 3 - Інтерфейс веб-додатку.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД	–	База даних
ПЗ	–	Програмний застосунок
API	–	Application Programming Interface
CLI	–	Command Line Interface
CSS	–	Cascading Style Sheets
DOM	–	Document Object Model
HTML	–	HyperText Markup Language
IDE	–	Integrated Development Environment
JS	–	JavaScript
JSX	–	JavaScript XML
MVC	–	Model View Controller
MVVM	–	Model View View Model
ORM	–	Object-Relational Mapping
SQL	–	Structured Query Language
UML	–	Unified Modeling Language

ВСТУП

У наш час люди все менше користуються реальними магазинами, віддаючи перевагу онлайн сервісам надання послуг. Тому усі компанії активно переносять свій бізнес у простори інтернету.

На сьогоднішній день яскраве зображення не значного розміру зазиває потенційного клієнта скористатись послугами компанії, а привабливий опис, зручний та привабливий дизайн, зрозумілий інтерфейс надання та отримання бажаних послуг спонукає користувача повертатися до сервісу знову. Для цього компанії витрачають значні кошти на спеціалістів цифрової галузі, таких як програмістів, Web-дизайнерів, SEO спеціалістів.

Саме це стало причиною створення веб-додатку інтернет магазину побутової техніки. З переваг якого буде унікальний, зручний інтерфейс, за допомогою якого користувачі зможуть швидко орієнтуватись в системі та швидко підбирати необхідний товар. Оформлення замовлення буде дуже простим, потрібно буде лише визначитися з товарами, додати їх до кошика, натиснути оформити замовлення та заповнити дуже просту форму, у якій потрібно вказати свої контактні данні. Через деякий час з клієнтом зв'яжеться менеджер компанії.

У веб-додатку повинна бути зручна панель адміністратора, для полегшення прийому, підтвердження замовлень та редагування товарів у базі.

Веб-додаток повинен робити вибір та замовлення відповідних товарів більш зручним ніж у компаній конкурентів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

1.1 Аналіз предметної області

В наш час розвинених технологій існує безліч методів продажу товарів: на ринку, у магазині, через онлайн платформу, або у інтернет магазині. Тобто, фактично методи продажу можна розділити на два види:

- онлайн;
- офлайн.

Кожен з цих методів має свої переваги та недоліки. На перший погляд все доволі просто. В один магазин можна зайти фізично, побродити між полицями і стелажми, взяти товари в руки. Другий не має приміщення. Замість нього – веб-додаток з каталогами. Але на цьому відмінності не закінчуються.

Бізнес процеси онлайн та офлайн магазинів – однакові. Власник створює бізнес-план, відкриває магазин, купує товари для продажу, налагоджує взаємодію з покупцями. Крім цього, йому доводиться вирішувати проблеми з організацією реклами, проведенням маркетингових компаній, орендою складу, розрахунком і сплатою податків і т. д.

Інтернет-магазин не передбачає наявності особистого спілкування з клієнтами. Але ніхто не відміняв розсилки листів на електронну пошту, дзвінки і СМС-повідомлення, спілкування через месенджери. В інтернет-магазинах навіть комунікації досить автоматизовані. Існують чат-боти, що імітують діалог з продавцем і спонукають покупця купити більше, ніж він планував спочатку. Способів взаємодії з клієнтами багато. Пріоритетне завдання власника магазину - вибір оптимальних методів, автоматизація їх і спрощення. В ідеалі комунікаціями повинно займатися мінімум людей-працівників магазину.

Переваги інтернет-магазину:

- торгівля в електронному варіанті - мінімум накладних витрат. Не потрібні великі офіси. На першому етапі можна обходитися взагалі без найманих

працівників. Можна працювати і без складу. Метод дропшипінг досить популярний. Ви берете на себе роль посередника і продаєте товар з інших магазинів. Ви допомагаєте зустрітися покупцеві і продавцеві, отримуючи за це винагороду;

- можливість тримати під контролем дані по проданих товарах, рекламі і т. д. В офлайн-версії така аналітика зустрічається рідко;

- товари коштують дешевше, ніж в стаціонарних торгових точках. Ви економите на накладних витратах, а покупець радіє невеликим знижкам і в наступний раз вибирає саме вас. А безкоштовна доставка - ще один плюс на користь користувача;

- час покупців. Мало хто вже хоче витрачати дні та години на походи по магазинах. Все те саме можна виконати зі смартфона під час щоденної поїздки на роботу або додому, або пізно ввечері, коли звичайні магазини вже закриті. Багато хто віддає перевагу купувати в інтернеті тільки через це;

- географія. Звичайний магазин розрахований на жителів одного кварталу, району або міста. Покупку в онлайн магазині можна зробити з будь-якого міста чи області. Нова пошта, або інша кур'єрська служба доставить посилку навіть в саму далеку точку;

- анонімність. Багато людей соромляться купувати в реалі нижню білизну, предмети інтимної гігієни. В інтернеті ніхто не бачить обличчя покупця. А на посилці буде позначено, що знаходиться всередині.

Переваги звичайного магазину:

- товар можна взяти в руки, помацати, подивитися. Жодне докладний опис не дає такої можливості. До того ж багато хто воліє все ж міряти одяг, подивитися, як працює праска або пілосос. Інтернет-магазини такої можливості не дають;

- товар можна забрати відразу, не доводиться чекати доставки. Іноді така можливість просто необхідна, наприклад, якщо зламався каблук на туфлях;

- повернути товар можна в будь-якому магазині. Але в звичайному процедура простіше і проходить швидше. В онлайн час займають процеси пересилання;
- віртуальні магазини дають гарантію на свої товари. Але, якщо пральна машина зламалася, вирішити проблему простіше, якщо вона придбана в сусідньому торговому центрі;
- універсальність. Все більше людей вважають за краще робити покупки в мережі. Але велика і частка тих, хто ще вважає за краще ходити по торговим залам. Причина - безліч шахраїв, які працюють на просторах інтернету. Та й заявлена якість не завжди відповідає реальності.

1.2 Аналіз веб-додатків аналогів

1.2.1 Інтернет магазин побутової техніки «ФБК»

ФБК – Запорізький магазин побутової техніки що має свій веб-додаток (показано на рис. 1.1).



Рисунок 1.1 – Інтернет-магазин побутової техніки «ФБК»

Веб-додаток дозволяє переглядати товар магазину, відфільтрувавши його за категоріями, що дуже зручно. Також є можливість обрати ціновий діапазон товару, бренд, та інші критерії фільтрації. Веб-додаток дозволяє обрати товар, додати його до корзини, а також оформити заказ.

Із недоліків: не дуже привабливий зовнішній вигляд сайту та незручність використання. В деяких частинах додатку дизайн інтуїтивно незрозумілий.

Переглянути веб-додаток можна за посиланням [1].

1.2.2 Byttechnika – інтернет-магазин побутової техніки.

Переглянути веб-додаток можна за посиланням [2].

Веб-додаток магазину виглядає дуже привабливо, має зручний інтерфейс (рис 1.2). Функціонал дозволяє обрати певну категорію товару, відфільтрувати його. Магазин надає послуги гарантії та доставки.

Але фільтрація товару виглядає дуже «зім'ято», та не зрозуміло.

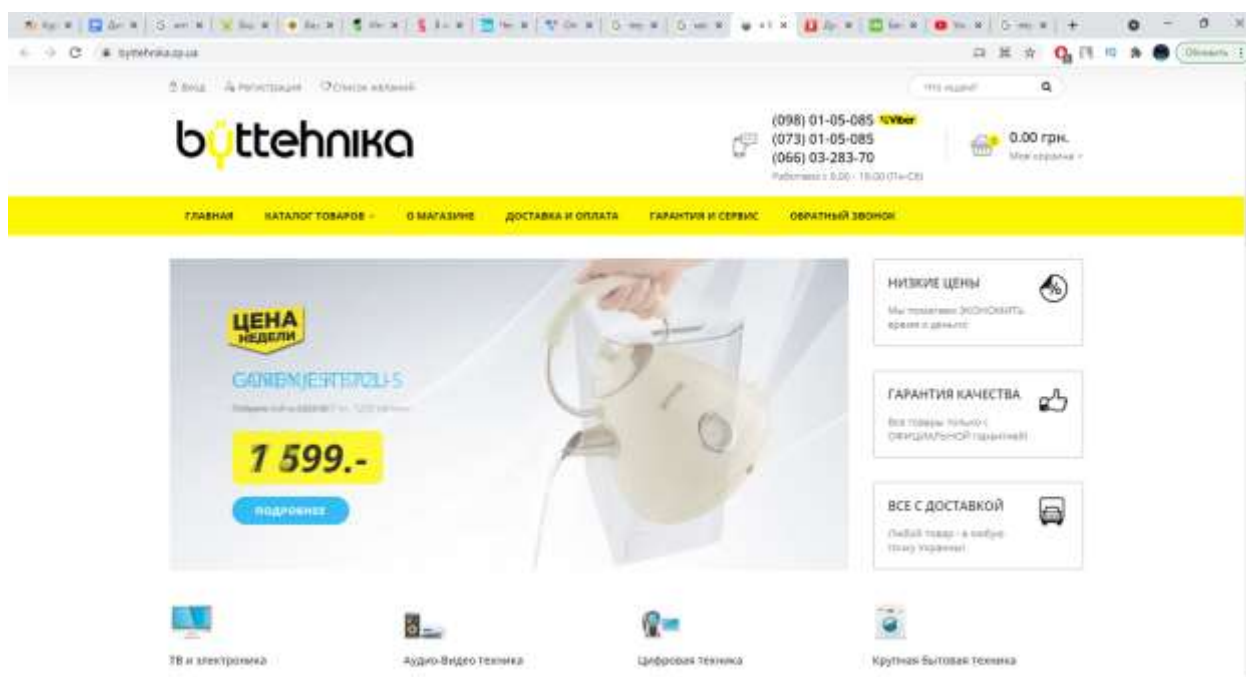


Рисунок 1.2 – Ітерфейс веб-додатку Byttechnika

1.2.3 Інтернет-магазин XCOM

XCOM - магазин не тільки побутової техніки, а також комп'ютерної, портативної, авто-електроніки, тощо (рис. 1.3). Магазин має великий вибір товару ,

але дизайн веб-додатку дуже не зручний та інтуїтивно не зрозумілий. Багато товарів відображається не коректно. Зображення товару на сторінці каталогу перекриває опис товару, що також виглядає дуже не професійно.

Веб-додаток має інтегрований чат, для зв'язку з продавцем, що є великою перевагою.



Рисунок 1.3 - Интернет магазин XCOM

Переглянути веб-додаток можна за посиланням [3].

1.3 Реляційні бази SQL

Реляційна база даних, або база даних SQL, написана в Structured Query Language (SQL) - це більш жорсткий, структурований спосіб зберігання даних, як телефонна книга. Розроблена IBM в 1970-х роках, реляційна база даних складається з двох або більше таблиць із стовпцями та рядками. Кожен рядок представляє запис, а кожен стовпець сортує дуже конкретний тип інформації, як-от ім'я, адресу та номер телефону. Зв'язок між таблицями та типами полів називається схемою. У

реляційній базі даних схема повинна бути чітко визначена перед додаванням будь-якої інформації [4].

Щоб реляційна база даних була ефективною, дані, які зберігаються в ній, повинні бути структуровані в дуже організованому вигляді. Добре розроблена схема мінімізує надмірність даних та запобігає несинхронізації таблиць, що є важливою функцією для багатьох підприємств, особливо тих, що реєструють фінансові операції. Погано розроблена схема може спричинити помилки в організації через її жорсткість. Наприклад, для стовпця, призначеного для зберігання телефонних номерів, може знадобитися 10 цифр, оскільки це стандарт для телефонних номерів в Україні. Ця перевага полягає у відхиленні будь-яких недійсних значень (наприклад, якщо в номері відсутній код міста). Однак якщо вам потрібно змінити схему (наприклад, якщо вам потрібно включити запис міжнародного телефонного номера, що містить більше 10 цифр), тоді всю базу даних потрібно відредагувати [4].

Мова структурованих запитів (SQL) - це мова програмування, що використовується архітекторами баз даних для проектування реляційних баз даних. У базі даних SQL, наприклад MySQL, Sybase, Oracle або IBM DB2, SQL виконує запити, отримує дані та редагує дані шляхом оновлення, видалення або створення нових записів. SQL - це легка декларативна мова, яка важко піднімає реляційну базу даних, діючи як версія бази даних сценарію на стороні сервера. Однією з особливих переваг SQL є його простий, але потужний пункт JOIN, який дозволяє розробникам отримувати відповідні дані, що зберігаються в декількох таблицях, за допомогою однієї команди [4].

1.4 Нереляційні бази NoSQL

Технології NoSQL з'явилися ще в 1960-х і носили різні назви, але нинішній сплеск їх популярності пов'язаний з тим, що картина даних змінюється, а

організатори повинні адаптуватися до величезних обсягів і масивів даних, що надходять із хмари, мобільних пристроїв, соціальних мереж і систем обробки великих даних.

Якщо вимоги до даних не зрозумілі з самого початку, або якщо мається справа з величезними обсягами неструктурованих даних, може не бути можливості створювати реляційну базу даних з чітко визначеною схемою. Використовуються нереляційні бази даних, які пропонують набагато більшу гнучкість, ніж їхні традиційні аналоги. Нереляційні бази даних, більше схожі на папки файлів, що збирають відповідну інформацію всіх типів. Якщо в блозі WordPress використовується база даних NoSQL, кожен файл може зберігати дані для публікації в блозі: соціальні лайки, фотографії, текст, показники, посилання [4].

Неструктуровані дані з Інтернету можуть включати дані датчиків, соціальний доступ, особисті налаштування, фотографії, інформацію на основі місцезнаходження, діяльність в Інтернеті, показники використання тощо. Спроба зберігати, обробляти та аналізувати всі ці неструктуровані дані призвела до розробки без схем альтернатив SQL. У сукупності ці альтернативи називаються NoSQL, що означає "Не тільки SQL". Хоча термін NoSQL охоплює широкий спектр альтернатив реляційним базам даних, загальним для них є те, що вони дозволяють обробляти дані більш гнучко [4].

Замість таблиць бази даних NoSQL орієнтовані на документи. Таким чином, неструктуровані дані (такі як статті, фотографії, дані в соціальних мережах, відео чи вміст у дописі блогу) можуть зберігатися в одному документі, який можна легко знайти, але не обов'язково класифікувати на такі поля, як робить реляційна база даних. Для масового зберігання даних потрібні додаткові зусилля з обробки та більше місця, ніж високоорганізовані дані SQL. Ось чому Hadoop, платформа обчислень та аналізу даних з відкритим кодом, здатна обробляти величезні обсяги даних у хмарі, популярна у поєднанні зі стеками баз даних NoSQL [4].

Бази даних NoSQL пропонують ще одну велику перевагу, особливо для розробників додатків: простоту доступу. Реляційні бази даних пов'язані з програмами, написаними об'єктно-орієнтованими мовами програмування, такими

як Java, PHP та Python. Бази даних NoSQL часто можуть уникнути цієї проблеми за допомогою API, які дозволяють розробникам виконувати запити без необхідності вивчати SQL або розуміти основну архітектуру їхньої системи баз даних [4].

1.5 Порівняння SQL та NoSQL

Що стосується технологій баз даних, то тут не існує універсального рішення. Ось чому багато підприємств покладаються як на реляційні, так і на нереляційні бази даних для різних завдань. Незважаючи на те, що бази даних NoSQL набувають популярності завдяки своїй швидкості та масштабованості, все ще існують ситуації, коли високоструктурована база даних SQL може бути кращою [4].

Ось кілька причин, чому можна вибрати базу даних SQL [4]:

- якщо потрібно забезпечити відповідність ACID (атомарність, узгодженість, ізолюваність, довговічність). Відповідність ACID зменшує аномалії та захищає цілісність вашої бази даних, визначаючи, як саме взаємодіють транзакції з базою даних. Як правило, бази даних NoSQL жертвують відповідністю ACID заради гнучкості та швидкості обробки, але для багатьох електронних комерцій та фінансових додатків кращим варіантом залишається база даних, сумісна з ACID;

- дані структуровані та незмінні. Якщо бізнес не переживає значного зростання, що вимагатиме більшої кількості серверів, і відбувається праця лише з послідовними даними, тоді може не бути причин використовувати систему, призначену для підтримки різноманітних типів даних та великого обсягу трафіку.

Коли всі інші компоненти серверної програми розроблені так, щоб вони були швидкими та безперебійними, NoSQL БД не дають даним бути «вузьким місцем». NoSQL можуть працювати з великими даними, роблячи те, що традиційні реляційні бази даних не можуть. Це сприяє популярності баз даних NoSQL, таких як MongoDB, CouchDB, Cassandra та HBase [4].

Чому можна вибрати базу даних NoSQL [4]:

- зберігання великих обсягів даних, які часто майже не мають структури. База даних NoSQL не встановлює обмежень на типи даних, які ви можете зберігати разом, і дозволяє додавати різні нові типи, коли ваші потреби змінюються. За допомогою баз даних, заснованих на документах, ви можете зберігати дані в одному місці без необхідності заздалегідь визначати, які «типи» цих даних;
- максимальне використання хмарного зберігання та обчислень. Хмарне сховище - це відмінне рішення для економії коштів, але воно потребує дані, що легко поширювати на декількох серверах. Використання доступного апаратного забезпечення на місці або в хмарі економить клопоти з додатковим програмним забезпеченням, а бази даних NoSQL, такі як Кассандра, розроблені для масштабування в декількох центрах обробки даних з коробки без особливих проблем;
- швидкий розвиток. Якщо розробка виконується впродовж двох тижнів, виконуються швидкі ітерації або необхідно часто оновлювати структуру даних без великих простоїв між версіями, реляційна база даних сповільнить процес розробки. Дані NoSQL не потрібно готувати заздалегідь.

1.6 Огляд існуючих реляційних субд

Oracle - об'єктно-реляційна СУБД, написана мовою C ++. Якщо є бюджет, це варіант повного спектру послуг із чудовим обслуговуванням та надійністю [4].

Актуальна версія Oracle призначена для хмарних середовищ і може бути розміщена на одному або декількох серверах, це дозволяє управляти базами даних, які містять мільярди записів. Деякі з функцій новітньої версії Oracle включають в себе grid framework і використання як фізичних, так і логічних структур.

Це означає, що фізичне управління даними не впливає на доступ до логічних структур. Крім того, безпека доведена до найвищого рівня, тому що кожна транзакція ізольована від інших.

Sybase - серверний продукт реляційної моделі баз даних для підприємств, що в основному використовується в ОС Unix, яка була першою СУБД на рівні підприємства для Linux [4].

MS SQL Server - розроблена корпорацією Майкрософт СУБД для баз даних на рівні підприємства, що підтримує архітектури SQL і NoSQL [4].

Специфіка роботи сервера бази даних SQL server полягає в транзакційній обробці даних. Це означає, що по кожен запит від СУБД обробляється і зберігається невелика кількість інформації.

Застосування SQL server дозволяє автоматизувати рішення різних бізнес-задач, підтримувати проведення аналітики даних в режимі онлайн, відслідковувати напрямки ресурсів СУБД, управляти транзакціями (операціями з обробки даних).

PostgreSQL - об'єктно-реляційна СУБД на рівні підприємства, яка використовує процедурні мови, такі як Perl та Python [4].

Двигун БД може бути розміщений в ряді середовищ, в тому числі віртуальних, фізичних і хмарних. Найсвіжіша версія пропонує обробку великих обсягів даних і збільшення числа одночасно працюючих користувачів. Безпека була покращена завдяки підтримці DBMS_SESSION.

SQLite - компактна вбудована СУБД. Двигун SQLite не є окремо працюючим процесом, з яким взаємодіє програма, а надає бібліотеку, з якої програма komponується і двигун стає складовою частиною програми. Таким чином, в якості протоколу обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому комп'ютері, на якому виконується програма [5].

Кілька процесів або потоків можуть одночасно без будь-яких проблем читати дані з однієї бази. Запис в базу можна здійснити тільки в тому випадку, якщо ніяких інших запитів в даний момент не обслуговується; в іншому випадку спроба запису

закінчується невдачею, і в програму повертається код помилки. Іншим варіантом розвитку подій є автоматичне повторення спроб запису протягом заданого інтервалу часу. Можна, також, ввести таймаут операцій. Тоді підключення, зіткнувшись із зайнятістю БД, чекатиме декілька секунд до того, як відвалитися з помилкою `SQLITE_BUSY` [5].

1.7 Порівняння PostgreSQL, SQLite

Бібліотека SQLite - одна з найбільш компактних бібліотек у цьому списку, де розмір бібліотеки може бути легше ніж 600 КБ. Завдяки дуже малому розміру та природі СУБД, він дуже добре підходить для вбудованих пристроїв [6].

Деякі інші випадки хорошого використання включають веб-сайти з низьким та середнім трафіком (~ 100 тис. Запитів на день) [6].

Однією з головних переваг SQLite є те, як він може виступати додатковим рішенням для клієнт-серверних СУБД. Наприклад, він може кешувати дані з клієнт-серверних СУБД локально, тим самим зменшуючи час очікування запитів та підтримуючи кінцеву програму в разі відключення корпоративних СУБД [6].

Завдяки своїм чудовим можливостям паралельної обробки PostgreSQL виходить на перше місце (порівняно з MySQL) при запуску довгих SELECT, як в аналітичних додатках. PostgreSQL завжди вважався найкращим для аналітичних процесів, таких як зберігання даних. Яскравим прикладом є БД Timescale, яка може дозволити вставити 1 мільйон записів в секунду. Приклад використання також включає додатки OLTP та додатки у фінансовій галузі завдяки суворому дотриманню вимог ACID. Розширюваність бази даних PostgreSQL також робить її ідеальним кандидатом для дослідницьких та наукових проектів [6].

Отже, виходячи з усіх вище перерахованих факторів, було вирішено обрати СУБД PostgreSQL.

1.8 Огляд мов програмування

1.8.1 Мова розмітки - HTML та CSS

HTML - це мова розмітки гіпертексту. Вона дозволяє користувачеві створювати і структурувати розділи, параграфи, заголовки, посилання і блоки для веб-сторінок і додатків [7].

HTML не є мовою програмування, тобто не має можливості створювати динамічні функції. Замість цього дозволяє організовувати і формувати документи, аналогічно Microsoft Word [7].

При роботі з HTML використовуються прості структури коду (теги і атрибути), щоб розмітити сторінку веб-сайту [7].

HTML теги - це спеціальні команди для браузера, які вказують, що, наприклад, слід вважати заголовком сторінки, а що абзацом [8].

Теги будуються за таким принципом: відкриваючий куточок, потім ім'я тега, а потім закриваючий куточок, ось так: <ім'я тега>. Ім'я тега може складатися з англійських букв і цифр. Приклади тегів: <h1>, <p>, [8].

Теги зазвичай пишуться парами - відкриваючий тег і відповідний йому закриваючий. Різниця між відкриваючими і закриваючими тегамі в тому, що в закриваючому тезі перед закриваючим куточком ставиться слеш, ось так: <ім'я тега/> [8].

CSS (Cascading Style Sheets, каскадні таблиці стилів) - мова опису зовнішнього вигляду HTML-документа. Це одна з базових технологій в сучасному інтернеті. Практично жоден сайт не обходиться без CSS, тому HTML і CSS діють в єдиній зв'язці [9].

Каскадні таблиці стилів працюють з HTML, але це зовсім інша мова. HTML структурує документ і впорядковує інформацію, а CSS взаємодіє з браузером, щоб надати документу оформлення [9].

1.8.2 JavaScript

При генерації сторінок в Web виникає дилема, пов'язана з архітектурою «клієнт-сервер». Сторінки можна генерувати як на стороні клієнта, так і на стороні сервера. У 1995 році фахівці компанії Netscape створили механізм управління сторінками на клієнтській стороні, розробивши мову програмування JS [10].

Таким чином, JavaScript - це мова керування сценаріями перегляду гіпертекстових сторінок Web на стороні клієнта [10].

Основна ідея JavaScript полягає в можливості зміни значень атрибутів HTML-контейнерів і властивостей середовища відображення в процесі перегляду HTML-сторінки користувачем. Перезавантаження сторінки не відбувається [10].

JavaScript - це мова програмування, яка [11]:

- має високий рівень: забезпечує абстракції, які дозволяють ігнорувати деталі машини, де вона працює. Автоматично керує пам'яттю за допомогою збирача сміття, тому можна зосередитись на коді, а не керувати пам'яттю, як це було б потрібно іншим мовам, як C, і надає безліч конструкцій, які дозволяють мати справу з надзвичайно потужними змінними та об'єктами;

- динамічна: на відміну від статичних мов програмування, динамічна мова виконує під час виконання багато речей, які робить статична мова під час компіляції. У цьому є плюси і мінуси, і це дає нам потужні функції, такі як динамічне введення тексту, пізнє прив'язування, відображення, функціональне програмування, зміна часу виконання об'єктів, закриття та багато іншого;

- динамічно набрана: змінна не застосовує тип. Можна перепризначити змінну будь-якого типу, наприклад, присвоївши ціле число змінній, що містить рядок;

- інтерпретується: це загальновідома мова як інтерпретована, що означає, що їй не потрібен етап компіляції перед запуском програми, на відміну від C, Java або Go, наприклад. На практиці браузері компілюють JavaScript перед його виконанням з міркувань продуктивності, але це виглядає прозоро - додаткових кроків не передбачено;

– мультипарадигма: мова не застосовує жодної конкретної парадигми програмування, на відміну від Java, наприклад, яка змушує використовувати об'єктно-орієнтоване програмування, або C, що змушує імперативне програмування. Можна писати JavaScript за допомогою об'єктно-орієнтованої парадигми, використовуючи прототипи та новий (на ES6) синтаксис класів. Можна писати JavaScript у функціональному стилі програмування з його першокласними функціями або навіть у імперативному стилі (подібний до C).

1.9 Огляд існуючих фреймворків

Vue.js - це прогресивний фреймворк, що підходить для створення користувацьких інтерфейсів [12].

Назва фреймворка співзвучна з view, тобто з поданням (якщо говорити про моделі MVC) [12].

Vue.js - це фреймворк, який використовується для вирішення завдань саме рівня уявлення, його просто інтегрувати з іншими бібліотеками та проектами. Іншими словами, Vue.js - це інструмент, який можна впроваджувати поступово. Можливість впровадження пов'язана з тим, що Vue.js прагне до прогресивності: підтримку Vue можна додати у вже існуючий проект, завдяки чому його функціональність буде значно розширена. І це відрізняє його від інших фреймворків [12].

Взагалі простота інтеграції - це одне з основних переваг даного фреймворку, особливо в поєднанні з можливістю інтеграції з бекенд фреймворками [12].

Інші сильні сторони Vue.js - це простота в освоєнні і хороша документація, а також висока продуктивність [12].

Також Vue.js можна використовувати для створення односторінкових додатків [12].

Центральна концепція Vue.js - це концепція компонентів, тобто невеликих частин інтерфейсу користувача, які можна використовувати повторно. Таким чином, і сам додаток складається з частин-компонентів. Один компонент може включати кілька інших компонентів, тобто використовується деревоподібна ієрархія [12].

AngularJS - структурований фреймворк для динамічних web-додатків. Дозволяє використовувати HTML в якості мови шаблонів, а також розширювати HTML-синтаксис, щоб код програми виглядав коротко і лаконічно. Використовуючи Data-binding і Dependency injection (що йдуть з коробки), також зменшується кількість коду. І все це на браузерному JavaScript і працює з будь-якою серверною технологією [13].

Angular - це те, чим був би HTML, будь він призначений для додатків. HTML - чудова декларативна мова для статичних документів. Вона не містить багато чого для створення додатків, і, як результат, побудова в ньому веб-додатків зводиться до того щоб змусити браузер робити те, що потрібно.

ReactJs - це JS фреймворк, представлений розробниками Facebook. Існує багато різних за між собою думок з приводу користі і доцільності використання даного продукту [14].

ReactJs постійно працює з DOM, перемальовуючи його при зміні умов (та частина DOM, яку змінює ReactJs, називається компонентом). Раніше подібна практика сильно б відбилася на продуктивності додатка, але розробники ReactJs підійшли до вирішення даного питання кардинально: вони повністю переписали DOM на Javascript [14].

Важливою особливістю ReactJs є використання JSX. Це надбудова на JS, що дозволяє використовувати про-XML синтаксис в Javascript коді. JSX - це поєднання javascript і html, які в зв'язці є незвичним синтаксисом для більшості розробників. Стандартом вважається поділ JS частини від розмітки, що ускладнює спостереження за змінами HTML, JS, HTML. JSX дозволяє бачити всі процеси в одному місці, не відволікаючись на складності грамотного і валідного коду. Після компіляції JSX виходить чистий JS [14].

1.10 Порівняння та вибір фреймворку

Переваги Angular [15]:

- Angular-language-service - забезпечує інтелектуальні можливості і автозаповнення шаблону HTML-компонента;
- нові функції, такі як generation Angular, що використовують бібліотеки npm з CLI, generation, і розробка компонентів, що використовує Angular;
- Angular використовується разом з Typescript. Він має виняткову підтримку для цього;
- детальна документація, що дозволяє розробнику отримати всю необхідну інформацію, не вдаючись до допомоги його колег. Однак це вимагає більше часу для навчання;
- одностороння прив'язка даних, яка забезпечує виняткову поведінку додатка, що зводить до мінімуму ризик можливих помилок;
- впровадження залежностей від компонентів, пов'язаних з модулями і модульність в цілому;
- структура і архітектура, спеціально створені для великої масштабованості проекту;
- MVVM (Model-View-ViewModel), яка дозволяє розробникам працювати окремо над одним і тим же розділом програми, використовуючи один і той же набір даних.

Переваги VueJs [15]:

- посилений HTML. Це означає, що Vue.js має багато характеристик схожих з Angular, а це, завдяки використанню різних компонентів, допомагає оптимізації HTML- блоків;
- детальна документація. Vue.js має дуже детальну документацію, яка може прискорити процес навчання для розробників і заощадити багато часу на розробку програми, використовуючи тільки базові знання HTML і JavaScript;

- адаптивність. Може бути здійснений швидкий перехід від інших фреймворків до Vue.js через схожість з Angular і React з точки зору дизайну і архітектури;
- чудова інтеграція. Vue.js можна використовувати як для створення односторінкових додатків, так і для більш складних веб-інтерфейсів додатків. Важливо, що невеликі інтерактивні елементи можна легко інтегрувати в існуючу інфраструктуру без негативних наслідків;
- масштабування. Vue.js може допомогти в розробці досить великих шаблонів багаторазового використання, які можуть бути зроблені майже за той же час, що і більш прості;
- не значний розмір. Vue.js важить близько 20 КБ, зберігаючи при цьому свою швидкість і гнучкість, що дозволяє досягти набагато кращої продуктивності в порівнянні з іншими платформами.

Переваги ReactJS [15]:

- легко вивчити, завдяки простому дизайну, використання JSX (HTML-подібний синтаксис) для шаблонів і дуже докладної документації. Розробники витрачають більше часу на написання сучасного JavaScript і менше турбуються про код, специфічному для фреймворка;
- дуже швидка, завдяки реалізації React Virtual DOM і різним оптимізаціям рендеринга;
- відмінна підтримка рендеринга на стороні сервера, що робить його потужною платформою для контент-орієнтованих додатків.
- першокласна підтримка Progressive Web App (PWA) завдяки генератору додатків `create-react-app`;
- прив'язка даних є односторонньою, що означає менше небажаних побічних ефектів;
- Redux, найпопулярніша платформа для управління станом додатків в React, її легко вчити і використовувати;
- React реалізує концепції функціонального програмування (FP), створюючи простий в тестуванні і багаторазово використовуваний код;

- додатки можуть бути створені за допомогою TypeScript або Facebook's Flow, що мають вбудовану підтримку JSX;
- перехід між версіями, як правило, дуже простий: Facebook надає «кодові модулі» для автоматизації більшої частини процесу.

Отже, виходячи з усіх вище перерахованих факторів, було вирішено обрати фреймворк ReactJs.

1.11 Вибір засобів розробки серверної частини

Для розробки серверної частини додатку була обрана мова програмування JavaScript за допомогою середовища виконання NodeJs. В якості фреймворку NodeJs використовувався ExpressJs. Для зв'язку бази даних з сервером використовувалась ORM-бібліотека Sequelize.

Node.js - це крос-платформна середовище виконання JavaScript з відкритим вихідним кодом, яка виконує код JavaScript поза браузера.

В основному JavaScript використовується для створення сценаріїв на стороні клієнта, які вбудовані в HTML-код сторінки веб-сайту і запускаються двигуном JavaScript в браузері.

За допомогою Node.js розробники можуть використовувати JavaScript для отримання інструментів командного рядка. На стороні сервера він запускає необхідні сценарії для обробки динамічного вмісту веб-сторінки, перш ніж вона буде доступна в браузері користувача. Таким чином, Node.js уособлює собою концепцію "JavaScript всюди", дозволяючи розробляти веб-додатки на одній мові, як для сценаріїв з боку сервера, так і з боку клієнта.

Express.js - це самий простий і швидкий фреймворк Node.js, який використовується в якості проміжного обробника для управління серверами і маршрутами.

Express.js підходить для розробки простих додатків, які можуть обробляти декілька запитів одночасно і спираються на можливості технології Express.

Для цього фреймворка існує велика кількість докладних інструкції та описів, які складені розробниками, перевірившими його ефективність на практиці [16].

Основна особливість цього фреймворка полягає в тому, що для Express характерний невеликий обсяг базового функціоналу. Всі інші необхідні функції потрібно буде добирати за рахунок зовнішніх модулів. По суті, Express в чистому вигляді - це сервер і у нього може не бути жодного модуля [16].

Завдяки такому мінімалізму розробник одразу отримує в своє розпорядження легкий і швидкий інструмент, який він може розширювати і розвивати [16].

Тому саме з Express рекомендується починати роботу, якщо мається намір навчитися створювати додатки на платформі Node.js.

Sequelize - це ORM-бібліотека для додатків на Node.js, яка здійснює зіставлення таблиць в бд і відносин між ними з класами. При використанні Sequelize можна не писати SQL-запити, а працювати з даними як зі звичайними об'єктами [17].

ORM або Object-relational mapping (Об'єктно-реляційне відображення) - 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 [18].

1.12 Середовище розробки

В якості середовища розробки була обрана IntelliJ IDEA.

IntelliJ IDEA - це безкоштовна комерційне середовище Java IDE від JetBrains. Його дизайн орієнтований на продуктивність програміста. Ряд його функцій призначений для прискорення розробки і дозволяє програмістам зосередитися на функціональності, в той час як IntelliJ IDEA виконує рутинні завдання кодування. IntelliJ IDEA доступний в двох редакціях: Ultimate (платна) і Community (безкоштовна і з відкритим вихідним кодом). IDEA підтримує різні мови програмування з розширеними функціями перевірки коду і рефакторінга, такі як Groovy, Javascript, Actionscript, Ruby / JRuby, PHP і інші. Community Edition є відкритим вихідним кодом, але пропускає більшість функцій для серверної і веб-розробки.

IntelliJ IDEA аналізує код в пошуках зв'язків між символами у всіх файлах і на всіх мовах, що використовуються в проекті. На основі цього аналізу IntelliJ IDEA надає допомогу при написанні коду, зручну навігацію, перевірку помилок в коді і, звичайно, рефакторинг.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Загальний опис

2.1.1 Контекст продукту

Створюваний програмний продукт – веб-додаток для здійснення покупок побутової техніки за допомогою мережі Інтернет. Dodatok надає асортимент товарів та можливість зробити замовлення..

2.1.2 Функції продукту

Основні функції веб-додатку:

- вибір категорії товару
- пошук необхідного товару;
- фільтрація за характеристиками товарів;
- додавання товару до кошику;
- відправлення замовлення;

2.1.3 Характеристики користувачів

Гість - користувач, який здійснює пошук необхідних товарів та робить замовлення.

Адміністратор – особа, яка має стежити за відсутністю помилок системи, підтверджувати замовлення та додавати відповідну інформацію у веб-додаток.

2.2 Моделювання сценаріїв роботи

Для моделювання ПЗ був проведений детальний аналіз предметної області та вимог до програмного продукту, в ході чого були виявлені основні ролі та функції, які вони виконують (наведені у табл. 2.1). Також були розроблені діаграми використання за допомогою мови UML (Unified Modeling Language) - Уніфікована мова моделювання (показано на рис. 2.1-2.2).

Таблиця 2.1 – Основні ролі та їх функції

Роль	Функції
Гість	– перегляд товарів; – вибір категорії товарів; – пошук конкретного товару; – додавання товару до кошика; – замовлення;

Продовження таблиці 2.1

Роль	Функції
Адміністратор	– додавання нових товарів; – додавання нових категорій; – прийом замовлень; – підтвердження замовлень;



Рисунок 2.1 – UML-діаграма використання для ролі Гість

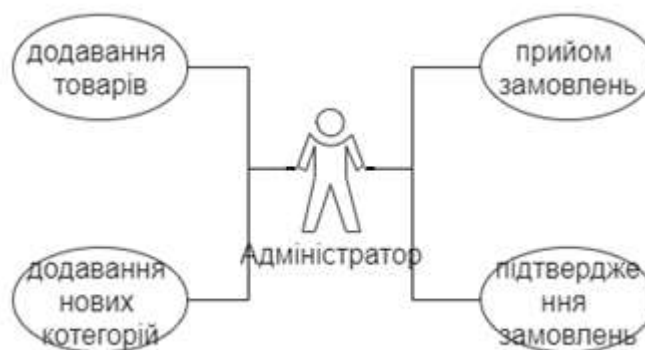


Рисунок 2.2 – UML-діаграма використання для ролі Адміністратор

2.3 Діаграми послідовностей

Діаграми послідовностей відображають взаємодії об'єктів впорядкованих за часом. Ці діаграми відображають задіяні об'єкти та послідовність відправлених

повідомлень. На діаграмі послідовностей зображено у вигляді вертикальних ліній процеси, які існують в один момент часу. Надіслані повідомлення зображуються у вигляді горизонтальних ліній, у порядку надіслання.

2.3.1 Перегляд товарів

Прецидент: перегляд товарів після завантаження головної сторінки веб-додатку (показано на рис. 2.3).

Актор: гість.

Передумови: веб-додаток працює коректно.

Післяумови: усі доступні товари відображаються гостю.

Основний сценарій:

- гість переходить до головної сторінки веб-додатку;
- очікуються завантаження товарів;
- товари відображаються користувачеві.

Альтернативний сценарій: товари не відображаються.

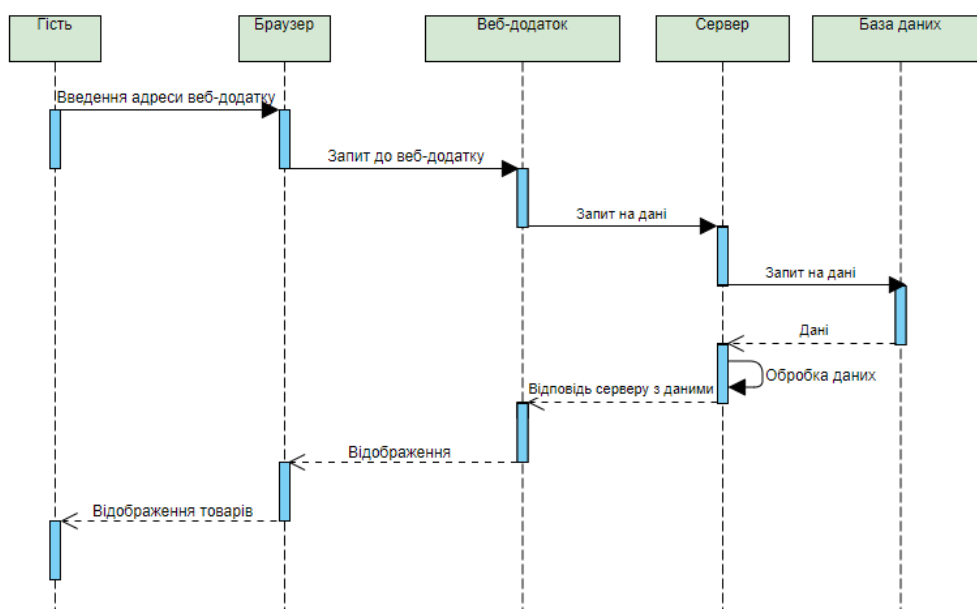


Рисунок 2.3 – UML-діаграма послідовності прециденту «Перегляд товарів»

2.3.2 Пошук товарів

Прецидент: пошук товарів через пошуковий рядок (показано на рис. 2.4).

Актор: гість.

Передумови: гість вводить назву товару у пошуковий рядок.

Післяумови: веб-додаток видає відповідні товари.

Основний сценарій:

- гість переходить до головної сторінки веб-додатку;
- гість вводить назву товару у пошуковий рядок;
- товари відображаються користувачеві.

Альтернативний сценарій: пошуковий рядок не працює.

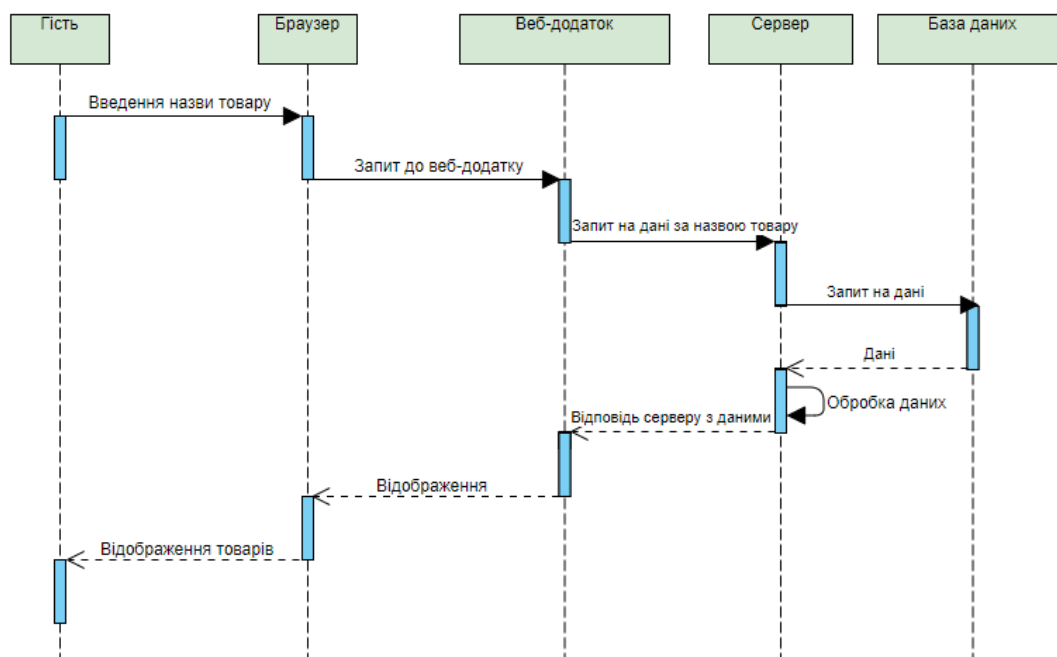


Рисунок 2.4 – UML-діаграма послідовності прецеденту «Пошук товарів»

2.3.3 Оформлення замовлення

Прецидент: оформлення замовлення (показано на рис. 2.5).

Актор: гість.

Передумови: гість додав товари до кошику та перейшов до оформлення замовлення.

Післяумови: замовлення успішно створено.

Основний сценарій:

- гість додає необхідні товари до кошику;

- гість переглядає товари у кошику та переходить до створення замовлення;
- гість заповнює форму та надсилає замовлення;
- веб-додаток відображає статус.

Альтернативний сценарій: оформлення замовлення не працює.

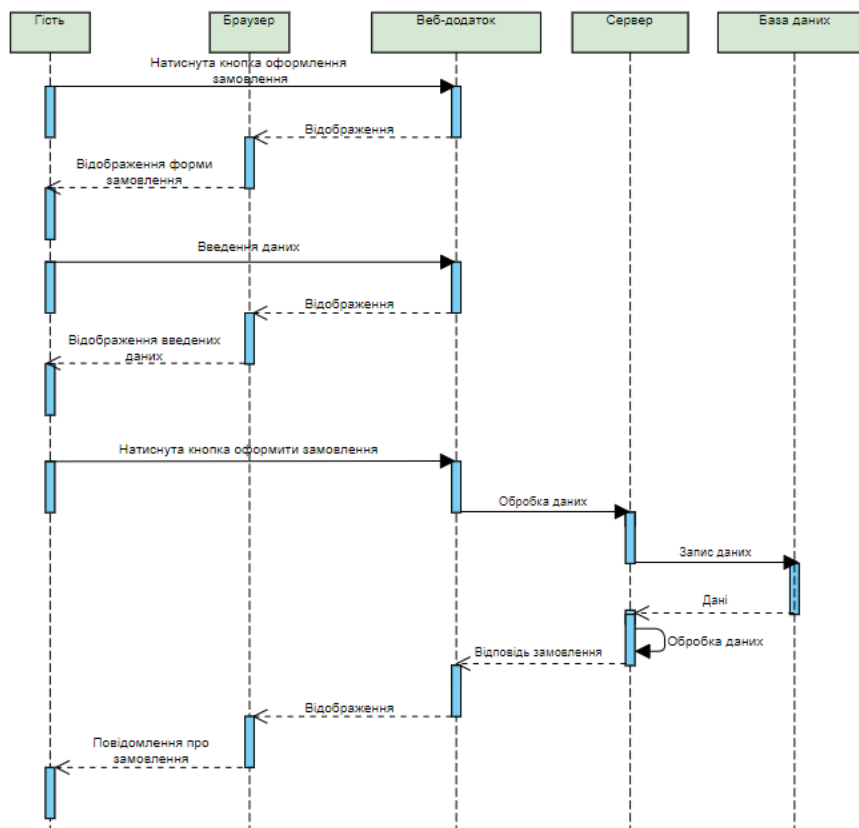


Рисунок 2.5 – UML-діаграма послідовності прецеденту «Оформлення замовлення»

2.4 Проектування графічного інтерфейсу користувача

Графічний інтерфейс користувача - одне з найважливіших нововведень в історії персональних обчислень. Графічні інтерфейси забезпечують

фундаментальну платформу для взаємодії людина-комп'ютер, і майже неможливо уявити, як люди могли б жити без цього.

При оформленні веб-додатку, необхідно враховувати велику кількість незначних факторів. В загалом створюють основний стиль веб-додатку і його привабливість для користувачів.

Одним із найважливіших факторів є кольорова палітра. Велику роль грає людське сприйняття кольору.

Важливо дотримуватися простоти та не використовувати більше 4 кольорів, підбирати між собою відтінки. Викорстовуються додаткові кольори, для виділення важливих елементів веб-додатку, таких як: навігаційне меню, кошик, каталог та ін.

Одним із не менш важливих факторів є зрозумілий інтерфейс. У веб додатку повинні бути елементи, що повторюються, наприклад: підвал веб-додатку, шапка з навігаційним меню.

В результаті проектування графічного інтерфесу було отримано сучасний простий та зручний інтерфейс (показано на рис. 2.6).

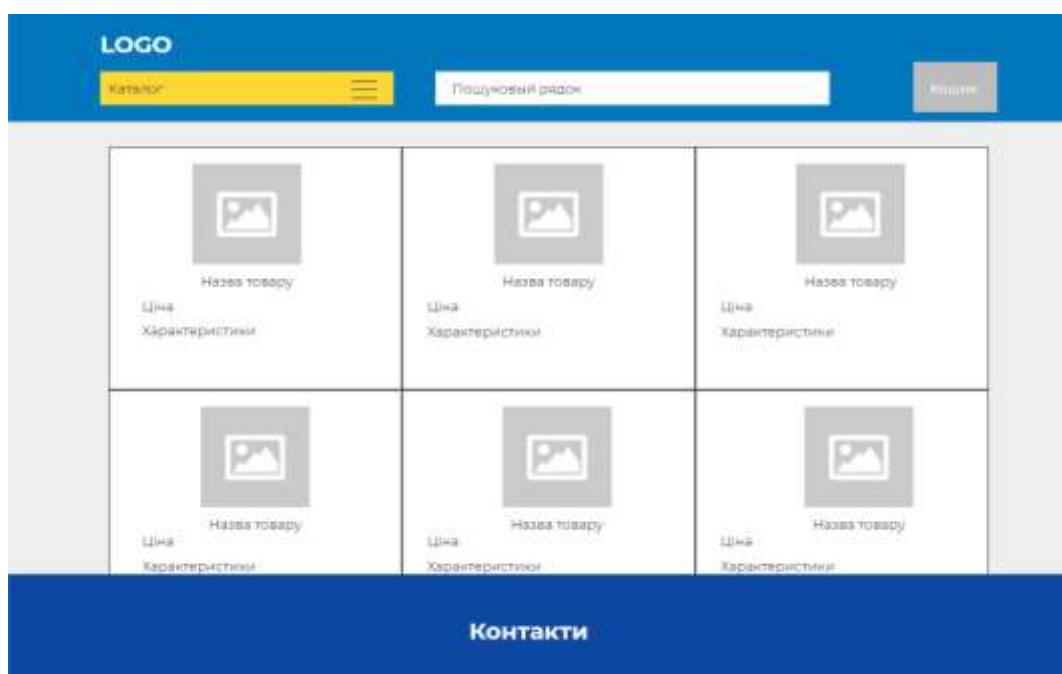


Рисунок 2.6 – Макет головної сторінки веб-додатку

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Клієнтська частина

Розробка клієнтської частини починається зі створення самої структури проекту (показано на рис. 3.1). Автоматизоване створення структури проекту при використанні фреймворку ReactJs виконується за допомогою команди: `npm create-react-app my-app`.

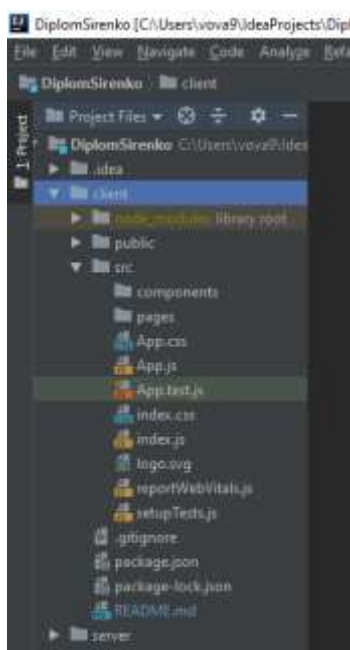


Рисунок 3.1 – Структура клієнтської частини проекту

Оскільки основна ідея ReactJS це використання та перевикористання компонентів, тому у веб-додатку було створено 10 незалежних компонентів (показано на рис. 3.2).

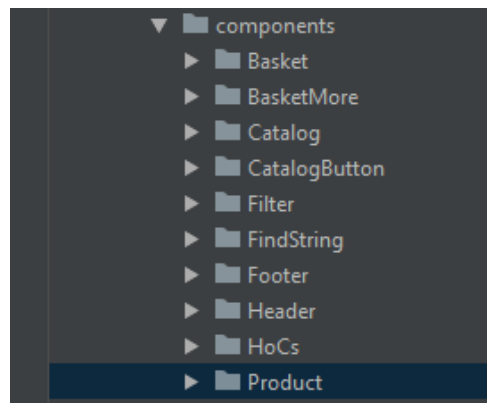


Рисунок 3.2 – Компоненти клієнтської частини проекту

До основних компонентів, що будуть постійно відображатися у веб-додатку, не залежно від шляху відносяться: Header та Footer.

У компоненті Header знаходяться компоненти: кнопка каталогу, пошуковий рядок, кошик, відкритий кошик (лістинг 3.1).

Лістинг 3.1 – Компонент Header

```
import React from 'react';
import styles from './HStyle.module.sass'
import CatalogButton from "../CatalogButton/CatalogButton";
import FindString from "../FindString/FindString";
import Basket from "../Basket/Basket";
import withContext from "../HoCs/widthContext";
import BasketMore from "../BasketMore/BasketMore";

class Header extends React.Component{
  constructor(props) {
    super(props);
  }
  render () {
    const {catalogButtonIsOpen, basketIsOpen} = this.props;

    return (
      <div className={styles.headerContainer}>
        <div className={styles.label}>ButTechno</div>
        <div className={styles.subBottomContainer}>
          <CatalogButton/>
          <FindString/>
          <Basket/>
          {
            basketIsOpen && <BasketMore/>
          }
        </div>
      </div>
    )
  }
}
```

```

    );
  }
}
export default withRouter(Header);

```

У компоненті Footer містяться контакти організації та додаткова інформація (лістинг 3.2).

Лістинг 3.2 – Компонент Footer

```

import React      from 'react';
import styles     from './FooterStyle.module.sass';
import Icon       from "@mdi/react";
import { mdiPhone } from "@mdi/js";

function Footer( props ) {
  return (
    <div className={styles.footerContainer}>
      <div className={styles.contacts}>
        <span><Icon      path={mdiPhone}      color="white"
size={0.8}/></span>
          <span>+380614441022</span>
          <span>+380954333011</span>
        </div>
        <div className={styles.year}>
          © 2021 buttechno.net.ua
        </div>
      </div>
    );
  }

export default Footer;

```

Для зв'язку клієнту з сервером у проекті було створено файл .../api/index.js (лістинг 3.3).

У файлі прописана конфігурація серверу, тобто дані куди необхідно надсилати запити, та заголовки, з якими будуть надсилатися запити. Також у файлі є функції з запитами, які приймають відповідь та повертають її.

Лістинг 3.3 – Вміст файлу .../api/index.js

```

import axios from 'axios';
import React from "react";

```

```

const config = {
  baseURL: 'http://localhost:3001/',
  headers: {
    'Content-type': 'application/json',
  },
};
const axiosInstance = axios.create(config);

const getQuery = async (url) => {
  try {
    const response = await axiosInstance.get(url);
    return response;

  } catch (e) {
    throw e;
  }
};

export const getProducts = async (url) => getQuery(url);
export const getCategories = async (url) => getQuery(url);
export const getFilters = async (url) => getQuery(url);

```

В ході розробки, для відображення у веб-додатку списку існуючих товарів у файлі головної сторінки була створена функція завантаження товарів (лістинг 3.4).

Функція завантаження товарів може приймати параметр у вигляді ідентифікатору категорії. У самій функції є умова, якщо виклик функції приймає параметр то змінюється рядок посилання, він доповнюється параметром ідентифікатору категорії. Далі параметр стану завантаження переводиться в одиницю та виконується виклик функції запиту до серверу з параметром посилання. В результаті обробки запиту отримані товари та інформація про закінчення завантаження записуються у стан компоненту.

Лістинг 3.4 – Функція завантаження товарів

```

loadProducts = (parameters) => {
  if (parameters) {
    let tmpNext =
`products/?categoryId=${parameters.categoryId}`;
    this.setState({
      products: [],
      productsCopy: [],
      next:
`products/?categoryId=${parameters.categoryId}`,
      categoryIdFilter: parameters.categoryId
    });
  }
}

```

```

    this.setState({isFetching: true});
    getProducts(tmpNext).then(value => {

        return this.setState({
            products: [...value.data],
            productsCopy: [...value.data],
            isFetching: false
        },
        );
    }).catch((e) => this.setState({isFetching: false}));

} else {
    this.setState({isFetching: true});
    getProducts(this.state.next).then(value => {
        console.log(value.data);
        return this.setState({
            products: [...value.data],
            isFetching: false
        },
        );
    }).catch((e) => this.setState({isFetching: false}))
}
};

```

Після виконання запиту на товари та запису у стан, дані розподіляються у компоненти Product. У кожен компонент Product передається свій об'єкт товару.

Компонент Product містить функцію додавання об'єкту «товар» до кошика через контекст веб-додатку. Додатково відбувається збереження інформації про вміст кошика у локальне сховище.

Товар містить такі дані:

- зображення товару;
- назва товару;
- ціна на товар;
- характеристики товару;
- бренд товару.

У компоненті також міститься кнопка, яка відображається за умови, що курсор миші наведений на компонент Product. Кнопка містить обробник натискання, який викликає функцію додавання товару до кошика (лістинг 3.5).

Лістинг 3.5 – Компонент Product

```

function Product(props) {
  const {basket, setBasket} = props;
  const addToBasket = () => {
    setBasket(JSON.stringify([props.product, ...JSON.parse(basket)]))
  };
  localStorage.setItem('basket', JSON.stringify([props.product, ...JSON.parse(basket)]));
  return (
    <div className={styles.productContainer}>
      <div className={styles.imageWrapper}>
        <img src={props.product.productImageUrl}
alt=""/>
      </div>
      <div className={styles.tovar}>
        <span
className={styles.nameProduct}>{props.product.productName}</span>
        <span className={styles.cina}>Цена:</span>
        <span
className={styles.cinaValue}>{props.product.cost} грн</span>
      </div>
      <div className={styles.characteristicsContainer}>
        <span
className={styles.cina}>Характеристики:</span>
        {props.product.Product_characteristics.map(
productCharacteristic => (
          <span
key={productCharacteristic.id}>{productCharacteristic.Characteristic
.characteristicName}: {productCharacteristic.characteristicValue}
{productCharacteristic.Characteristic.Measure.measureName}</span>
        ) ) }
        <span>Бренд:
{props.product.Firm.firmName}</span>
      </div>
      <button className={styles.toBasketBtn}
onClick={addToBasket}>В корзину</button>
    </div>
  );
}
export default withContext(Product);

```

Для зручності пошуку товарів було прийнято рішення зробити фільтрацію за параметрами товарів у відповідній категорії.

Функція завантаження фільтрів складається з функції запиту до серверу з відповідним посиланням з параметром ідентифікатору категорії товарів. В результаті виконання функції з запитом дані що повернулись зберігаються у стані компоненту Filter та відображаються на сторінці (лістинг 3.6).

Лістинг 3.6 – Функція завантаження фільтрів

```
loadFilters = () => {
  getFilters(this.state.next).then(value => {
    for (let firm of value.data.rangeFirms) {
      this.setState({
        checkBoxItems: [
          ...this.state.checkBoxItems,
          {
            id: firm.id,
            firmName: firm.firmName,
            isChecked: false
          }
        ]
      });
    }
  })
  return this.setState({
    params: {...value.data},
    marksStatic: {...value.data},
  })
});
```

Фільтри у компоненті відображаються як набір повзунків з мінімальними та максимальними значеннями характеристик товарів. В кінці фільтру відображається перелік фірм товарів у вигляді чекбокс елементів. При зміні будь-якого елементу з фільтрів відбувається фільтрація по усім поточним параметрам фільтру. Якщо при заданому наборі фільтрів не знайдеться відповідних товарів, то у контейнері з товарами відобразиться текст, що товарів не знайдено (лістинг 3.7).

Лістинг 3.7 – Функція фільтрування товарів

```
const loadFilteredProductsCost = (event, value) => {
  this.props.newProductsFunc(this.props.productsCopy);
  let newProducts = this.props.allProducts;
  let copy = this.state.copyNew;
  let checkBoxes = this.state.checkBoxItems;
  if (value) {
    for (let checkB of checkBoxes) {
      if (checkB.id === value) {
        checkB.isChecked = event.target.checked;
      }
    }
  }

  this.setState({
    ...this.state,
    checkBoxItems: checkBoxes
  });
  let isOneOfChecked = false;
```

```

    for (let checkBoxItem of checkBoxes) {
        if (checkBoxItem.isChecked === true) {
            isOneOfChecked = true;
            break;
        }
    }
    if (isOneOfChecked) {
        newProducts = copy.filter(product => {
            let tmp = true;
            for(let item of checkBoxes){
                if (item.id === product.Firm.id) {
                    tmp = tmp && item.isChecked;}}
            return tmp;});
    }else {newProducts = copy;}
    newProducts = newProducts.filter(product => {
        return product.cost >=
this.state.params.rangeCost[0].minCost && product.cost <=
this.state.params.rangeCost[0].maxCost; });
    newProducts = newProducts.filter(product => {
        let tmp = true;
        for (let product_characteristic of
product.Product_characteristics) {
            for (let item of
this.state.params.rangeValues) {
                if
(item.Characteristic.characteristicName ===
product_characteristic.Characteristic.characteristicName) {
                    tmp = tmp &&
product_characteristic.characteristicValue >=
item.minCharacteristicValue &&
product_characteristic.characteristicValue <=
item.maxCharacteristicValue; }}}
            return tmp; });
    this.props.newProductsFunc(newProducts);
};

```

Важливою частиною веб-додатку інтернет-магазину є кошик. Через кошик користувач може подивитися товари, які були обрані, загальну суму обраних товарів. Якщо товар обраний помилково, або користувач перехотів його замовляти, цей товар можна легко видалити з кошика за допомогою натискання на кнопки у вигляді хрестика навпроти відповідного товару (лістинг 3.9). Компонент кнопки кошика відображає у реальному часі кількість товарів, що знаходяться у кошику (лістинг 3.8). Якщо кількість товарів у кошику менше одного, кількість товарів не відображається, а у самому кошику відображається текст, що в кошику товарів немає.

ЛІСТИНГ 3.8 – КОМПОНЕНТ КНОПКИ КОШИКА Basket

```

class Basket extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    const {basket, setBasket, basketIsOpen, setBasketIsOpen}
= this.props;
    const lengthBasket = () => {
      return JSON.parse(basket).length;
    };
    const changeBasketIsOpen = () => {
      setBasketIsOpen(!basketIsOpen);
    };

    return (
      <>
        <div className={styles.basketContainer}
onClick={changeBasketIsOpen}>
          <Icon className={styles.icon}
path={mdiShopping} size={3}/>
          {
            JSON.parse(basket) !== null &&
JSON.parse(basket).length !== 0 &&
            <div
className={styles.kolItems}>{lengthBasket()}</div>
          }
        </div>
      </>
    )
  }
}
export default withContext(Basket)

```

ЛІСТИНГ 3.9 – КОМПОНЕНТ ВМІСТУ КОШИКА BasketMore

```

function BasketMore(props) {
  const {basket, setBasket ,setBasketIsOpen} = props;
  const deleteItemWithIdx = (index) => {
    let newMass = [];
    for (let i = 0; i < JSON.parse(basket).length; i++){
      if(i !== index){
newMass.push(JSON.parse(basket)[i]); }
      setBasket(JSON.stringify(newMass));
    }
    localStorage.setItem('basket',JSON.stringify(newMass));};
  const sumOreder = () => {
    let sum = 0;
    for (let item of JSON.parse(basket)){ sum += item.cost;
}return sum};
  return (
    <div className={Styles.mainContainer}>

```

```

        {JSON.parse(basket).map( (item, index) => (
            <div                                className={Styles.basketItem}
key={index}>
                <span
className={Styles.imageWrapper}><img                src={item.productImageUrl}
alt=""/></span>
                <span
className={Styles.productName}>{item.productName}</span>
                <span className={Styles.tyr}>-</span>
                <span className={Styles.tyr}>{item.cost}
грн</span>
                <Icon                                onClick={ ()                                =>
deleteItemWithIdx(index)}    className={Styles.close}    path={mdiClose}
size={0.8}/>
            </div>
        ) ) }
        {JSON.parse(basket).length === 0 ?
            <div
className={Styles.doOrderContainer}>Корзина пуста</div>
            :
            <div className={Styles.doOrderContainer}>
                <span>Сумма                                заказа:
{sumOreder()}грн</span>
                <button>Оформить заказ</button>
            </div>
        }
    </div>
    );
    export default withContext(BasketMore);

```

У веб-додатку бувають такі змінні, значення яких необхідно використовувати у різних частинах додатку. Для цього у ReactJS існує контекст, який зберігає дані для загального використання улюбій частині додатку. Для реалізації контексту необхідно обернути весь додаток у Provider, тобто постачальник та компоненти, що приймають контекст необхідно обернути у Consumer, тобто споживач. Для того щоб не було потреби у кожному компоненті де необхідний контекст додавати Consumer було створено функцію withContext (лістинг 3.10). Функція приймає у параметрах компонент, який необхідно обернути. Обертає компонент та розподіляє у ньому дані з контексту. Отже для того щоб отримати доступ до змінної у компоненті, необхідно прописати викликати функцію withContext та у параметрі передати компонент.

Лістинг 3.10 – Функція withContext

```
const withContext = (WrappedComponent) => (props => (
  <AppContext.Consumer>
    {
      value      =>      (<WrappedComponent      {...props}
{...value}/>)
    }
  </AppContext.Consumer>
))
);
```

Головний файл проекту містить ініціалізацію параметрів для контексту. Далі ці параметри передаються у об'єкт `contextValue`. У подальшому цей об'єкт передається у контекст `Provider`. Також у головному файлі міститься метод `useEffect`, для одноразової перевірки при завантаженні веб-додатку, чи були якісь дані у кошику після виходу з веб-додатку. Якщо дані були, вони записуються у контекст та веб-додаток одразу має інформацію про товари, які були у кошику. Якщо локальне сховище не має даних про товари у кошику, то параметр `basket` у контексті та у локальному сховищі ініціалізуються порожнім масивом.

Що стосовно інших компонентів, у головному компоненті веб-додатку міститься `Header` та `Footer`, що залишаються незмінними при будь якому посиланні у веб-додатку. Компоненти що змінюються в залежності від посилання у веб-додатку: `HomePage`, `CheckoutPage`. Компонент головної сторінки `HomePage` з'являється при завантаженні додатку, або при посиланні на корінь веб-додатку. Компонент оформлення замовлення `CheckoutPage` з'являється при переходженні до оформлення замовлення через кошик веб-додатку, або за посиланням `checkout` у додатку (лістинг 3.11).

Лістинг 3.11 – Головний файл проекту

```
import './App.css';
import React, {Component, lazy, Suspense, useEffect, useState}
from 'react';
import {Switch, Route, HashRouter} from 'react-router-dom';
import AppContext from './store';
import Header from './components/Header/Header';
import Footer from './components/Footer/Footer';
import CheckoutPage from './pages/CheckoutPage/CheckoutPage';
```

```

    const fallbackElem = null;
    const HomePage = lazy(() => import(
'../pages/HomePage/HomePage.js' ));
    function App() {
        const [catalogButtonIsOpen, setCatalogButtonIsOpen] =
useState(null);
        const [basket, setBasket] = useState(null);
        const [basketIsOpen, setBasketIsOpen] = useState(null);
        const [parameters, setParameters] = useState(null);
        const contextValue = {catalogButtonIsOpen,
setCatalogButtonIsOpen, basket, setBasket, basketIsOpen,
setBasketIsOpen, parameters, setParameters,};

        useEffect(() => {
            if(localStorage.getItem('basket')){
                setBasket(localStorage.getItem('basket'));
            }else{
                setBasket(JSON.stringify([]));
                localStorage.setItem('basket',JSON.stringify([]));
            }, []);

        return (
            <AppContext.Provider value={contextValue}>
                <div className="appContainer">
                    <HashRouter>
                        <Header/>
                        <Suspense fallback={fallbackElem}>
                            <Switch>
                                <Route exact path="/"
component={HomePage}/>
                                <Route exact path="/checkout"
component={CheckoutPage}/>
                            </Switch>
                        </Suspense>
                        <Footer/>
                    </HashRouter>
                </div>
            </AppContext.Provider>
        );}
    export default App;

```

3.2 База даних

Перед підключенням до бази даних, вона створювалась за допомогою графічного клієнту pgAdmin (показано на рис. 3.3).

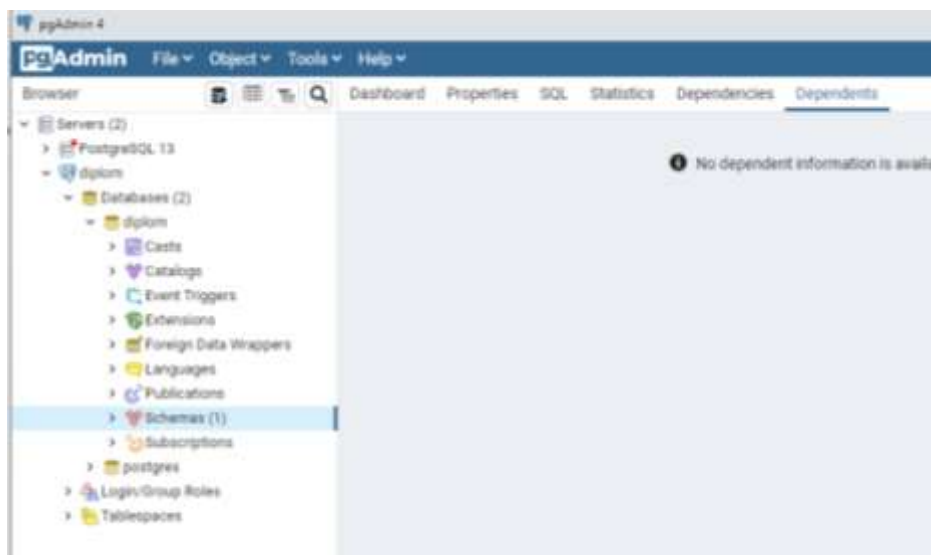


Рисунок 3.3 – Інтерфейс графічного клієнту pgAdmin

Для підключення до бази даних у проєкті прописувався конфігураційний файл бази даних, який називається `postgresConfig.json` (лістинг 3.12). В цьому файлі прописані дані бази даних до якої необхідно підключитися. Присутні три режими: розробка, тестування, виробництво.

Лістинг 3.12 – Вміст файлу `postgresConfig.json`

```
{
  "development": {
    "username": "postgres",
    "password": "12345",
    "database": "diplom",
    "host": "localhost",
    "dialect": "postgres",
    "operatorsAliases": "Op"
  },
  "test": {
    "username": "postgres",
    "password": "12345",
    "database": "diplom",
    "host": "localhost",
    "dialect": "postgres",
    "operatorsAliases": "Op"
  },
  "production": {
    "username": "postgres",
    "password": "12345",
    "database": "diplom",
    "host": "localhost",
    "dialect": "postgres",
```

```

    "operatorsAliases": "Op"
  }
}

```

Для створення таблиць та встановлення залежностей необхідно спочатку створити моделі та міграції у проекті. Щоб згенерувати модель і міграцію одночасно необхідно ввести у консоль команду «sequelize model:create --name [назва таблиці] --attributes [атрибут]:[тип даних]».

Таким чином у проекті було створено 9 міграцій (показано на рис. 3.4). У кожній міграції для полів майбутньої таблиці були прописані: типи даних, чи може поле бути нульовим, залежність поля, якщо є.

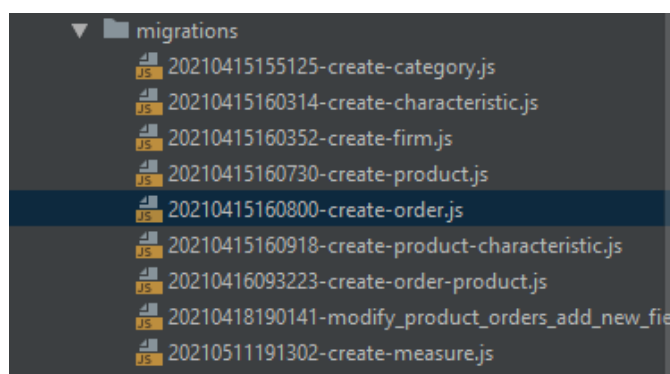


Рисунок 3.4 – Міграції проекту

Для того щоб оновити базу даних та виконати міграції проекту, необхідно у консолі ввести команду «sequelize db:migrate:all». Усі таблиці та залежності автоматично створяться в базі.

Таким чином було створено 8 таблиць та таблиця SequelizeMeta, яка створилась автоматично (показано на рис. 3.5).

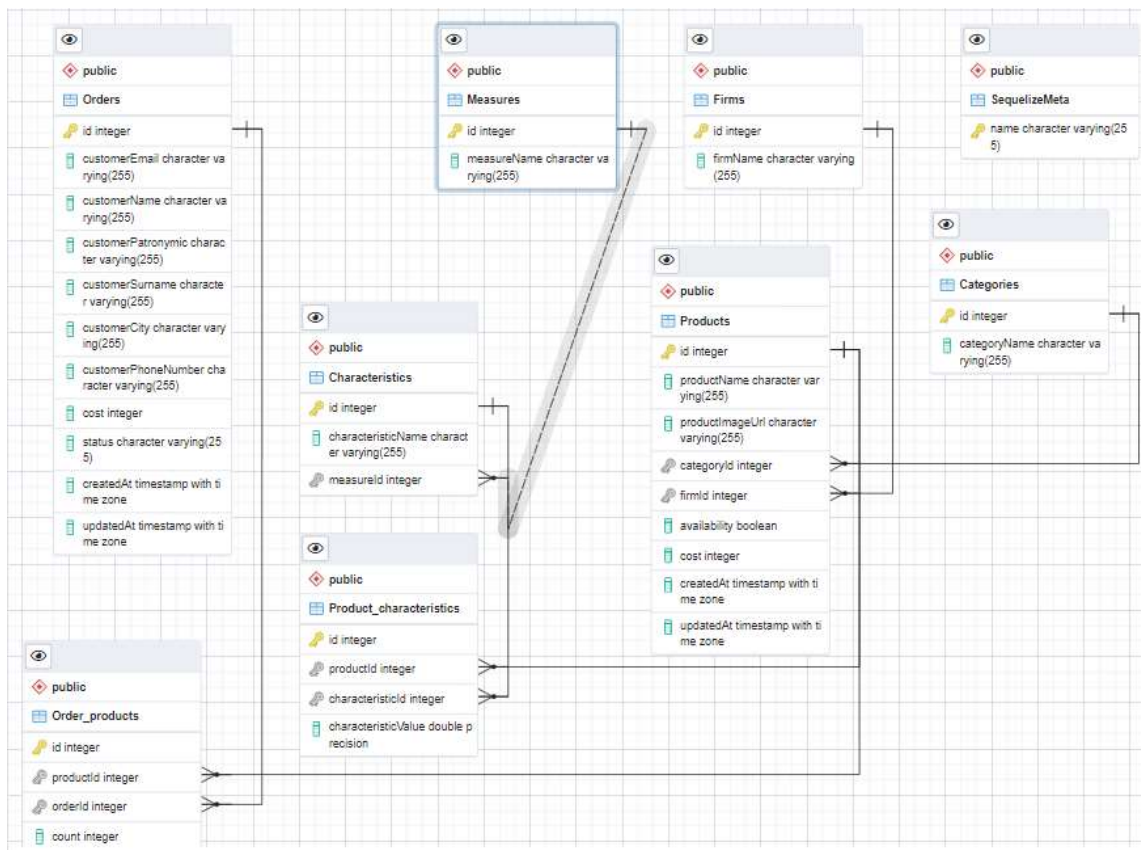


Рисунок 3.5 – Схема бази даних

3.3 Серверна частина

Для коректної роботи серверної частини з базою даних необхідно відновити моделі таблиць на стороні сервера.

В результаті було створено 8 моделей бази даних та вказані всі залежності (показано на рис. 3.6).

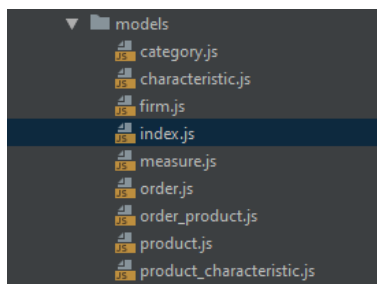


Рисунок 3.6 – Моделі бази даних

Для використання моделей було створено головний файл `index.js` в моделях, де відбувається поєднання усіх моделей в один об'єкт, та встановлення залежностей між ними (лістинг 3.13).

Лістинг 3.13 – Вміст головного файлу моделей

```
const fs = require('fs');
const path = require('path');
const Sequelize = require('sequelize');
const basename = path.basename(__filename);
const env = process.env.NODE_ENV || 'development';
const config = require(__dirname +
'../../config/postgresConfig.json')[env];
const db = {}; let sequelize;
if (config.use_env_variable) {sequelize = new
Sequelize(process.env[config.use_env_variable], config);
} else {sequelize = new Sequelize(config.database,
config.username, config.password, config);}
fs
  .readdirSync(__dirname)
  .filter(file => {
    return (file.indexOf('.') !== 0) && (file !== basename) &&
(file.slice(-3) === '.js');})
  .forEach(file => {
    const model = require(path.join(__dirname, file))(sequelize,
Sequelize.DataTypes);
    db[model.name] = model;});
db['Categories'].hasMany(db['Products'],
  {foreignKey: 'categoryId', sourceKey: 'id'});
db['Characteristics'].hasMany(db['Product_characteristics'],
  {foreignKey: 'characteristicId', sourceKey: 'id'});
db['Characteristics'].belongsTo(db['Measures'],
  {foreignKey: 'measureId', sourceKey: 'id'});
db['Firms'].hasMany(db['Products'],
  {foreignKey: 'firmId', sourceKey: 'id'});
db['Orders'].hasMany(db['Order_products'],
  {foreignKey: 'orderId', sourceKey: 'id'});
db['Order_products'].belongsTo(db['Products'],
  {foreignKey: 'productId', sourceKey: 'id'});
db['Order_products'].belongsTo(db['Orders'],
  {foreignKey: 'orderId', sourceKey: 'id'});
db['Products'].belongsTo(db['Categories'],
  {foreignKey: 'categoryId', sourceKey: 'id'});
db['Products'].belongsTo(db['Firms'],
  {foreignKey: 'firmId', sourceKey: 'id'});
db['Products'].hasMany(db['Product_characteristics'],
  {foreignKey: 'productId', sourceKey: 'id'});
db['Products'].hasMany(db['Order_products'],
  {foreignKey: 'productId', sourceKey: 'id'});
db['Product_characteristics'].belongsTo(db['Characteristics'],
```

```

    {foreignKey: 'characteristicId', sourceKey: 'id'}});
db['Product_characteristics'].belongsTo(db['Products'],
    {foreignKey: 'productId', sourceKey: 'id'});
db['Measures'].hasMany(db['Characteristics'],
    {foreignKey: 'measureId', sourceKey: 'id'});
db.sequelize = sequelize;
db.Sequelize = Sequelize;
module.exports = db;

```

При надходженні на сервер, запит потрапляє до роутеру. Роутер у свою чергу перевіряє адресу на яку прийшов запит. В залежності від цього надає відповідний контроллер для обробки запиту (лістинг 3.14).

Лістинг 3.14 – Вміст файлу роутеру

```

const productsController = require(
"../controllers/productsController");
const categoriesController = require(
"../controllers/categoriesController");
const orderController =
require("../controllers/orderController");
const filterController =
require("../controllers/filterController");
const express = require('express');
const router = express.Router();

router.post(
  '/createOrder',
  (req,res,next) => {
    console.log(req.body);
    next();
  },
  orderController.newOrder
);

router.get(
  '/products',
  productsController.getAllProducts
);

router.get(
  '/categories',
  categoriesController.getAllCategories
);

router.get(
  '/filters',
  filterController.getAllFilterParams
);

```

```
module.exports = router;
```

Контроллер має доступ до моделі бази даних у вигляді об'єкту та проводить відповідні зміни у базі. Після обробки запиту, результат відсилається клієнту. Усього було створено 4 контроллери, які містять у собі функції з які приймають запити, взаємодіють з базою та повертають оброблене значення (рис. 3.6).

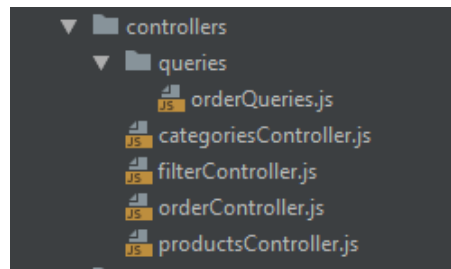


Рисунок 3.6 – Контроллери

Контроллер `productsController` містить запит на отримання усіх товарів (лістинг 3.15). Відбувається пошук усіх товарів включаючи пов'язані дані з таблиць: фірма, категорія, характеристики.

Лістинг 3.15 – Вміст файлу контролеру `productsController`

```
const querystringConverter = require('sequelize-querystring-converter');
const sqs = require('sequelize-querystring');
const Sequelize = require('sequelize');
const db = require('../models/index');
const Op = Sequelize.Op;

module.exports.getAllProducts = async (req, res, next) => {
  try {
    const products = await db.Products.findAll({
      where: req.query,
      include: [{
        model: db.Firms,
        as: 'Firm',
      },
      {
        model: db.Categories,
        as: 'Category',
      },
    ],
  },
```

```

        model: db.Product_characteristics,
        include: [{
            model: db.Characteristics,
            include: [
                {
                    model: db.Measures
                }
            ]
        }]
    }

    ],
    order: [
        ['id', 'asc'],
        [{model: db.Product_characteristics}, {model:
db.Characteristics}, 'id', 'asc']
    ]
    });
    res.send(products);
} catch (err) {
    next(err);
}
};

```

Контроллер `categoriesController` містить запит на отримання усіх категорій (лістинг 3.16). Відбувається пошук в базі усіх категорій з таблиці категорії за допомогою методу `findAll`.

Лістинг 3.16 – Вміст файлу контролеру `categoriesController`

```

const Sequelize = require('sequelize');
const db = require('../models/index');

module.exports.getAllCategories = async (req, res, next) => {
    try {
        const categories = await db.Categories.findAll();

        res.send(categories);
    } catch (err) {
        next(err);
    }
};

```

Контроллер `filterController` містить запит на фільтри для відповідної категорії товарів (лістинг 3.17). Відбувається пошук мінімальних та максимальних значень ціни та характеристик. Також знаходяться усі можливі фірми товарів. Усі отримані дані з бази компонуються та посилаються у відповідь запиту.

Лістинг 3.17 – Вміст файлу контролера filterController

```

const Sequelize = require('sequelize');
const db = require('../models/index');

module.exports.getAllFilterParams = async (req, res, next) => {
  try {
    console.log(req.query);
    let filterCost = await db.Products.findAll({
      where: req.query,
      attributes: [
        [Sequelize.fn('max',
Sequelize.col('cost')), 'maxCost'],
        [Sequelize.fn('min',
Sequelize.col('cost')), 'minCost'],
      ],
    });
    let filterValues = await
db.Product_characteristics.findAll({
      attributes: [
        [Sequelize.fn('max',
Sequelize.col('characteristicValue')), 'maxCharacteristicValue'],
        [Sequelize.fn('min',
Sequelize.col('characteristicValue')), 'minCharacteristicValue'],
      ],
      include: [{
        model: db.Characteristics,
        attributes: ['characteristicName']
      }],
      group: [
        `Characteristic.id`
      ]
    });
    let filterFirms = await db.Firms.findAll({
    });
    res.send({rangeCost: [...filterCost], rangeValues:
filterValues, rangeFirms: [...filterFirms]});
  } catch (err) {
    next(err);
  }
};

```

Контролер orderController містить запити на створення нового замовлення (лістинг 3.20).

Перший запит містить створення нового замовлення з інформацією клієнта (лістинг 3.18).

Лістинг 3.18 – Функція створення нового замовлення

```
module.exports.orderCreation = async (data) => {
  const newOrder = await db.Orders.create(data);

  if (!newOrder) {
    throw new ServerError('server error on order creation');
  } else {
    return newOrder.get({plain: true});
  }
};
```

Далі відбувається пошук усіх товарів із замовлення у базі даних. Формується масив з товарами у замовленні з ідентифікатором замовлення та відбувається створення товарів у відповідному замовленні (лістинг 3.19).

Лістинг 3.19 – Функція створення товарів у замовленні

```
module.exports.orderProductsCreation = async (data) => {
  const orderProducts = await
db.Order_products.bulkCreate(data);

  if (!orderProducts) {
    throw new ServerError('server error on order creation');
  } else {
    return orderProducts;
  }
};
```

Лістинг 3.20 – Вміст файлу контролера orderController

```
const orderQueries = require('./queries/orderQueries');
const Sequelize = require('sequelize');
const Op = Sequelize.Op;
const db = require('../models/index');

module.exports.newOrder = async (req, res, next) => {
  try {
    const newOrder = await
orderQueries.orderCreation(req.body);

    const products = await db.Products.findAll({
      where: {
        id: {
          [Op.or]: [...req.body.productsId]
        }
      }
    });
  }
};
```

```

        let data = [];
        products.forEach(p =>{
            data.push({'orderId':    newOrder.id,    'productId':
p.id}))
        });

        const            orderProducts            =            await
orderQueries.orderProductsCreation(data);

        res.send(orderProducts);
    }catch (err) {
        next(err);
    }
};

```

У головному файлі серверної частини відбувається створення серверу (лістинг 3.22). Прописаний конфіг для адмін панелі серверу, вказуються дані для входу в адмін панель (лістинг 3.21). Для коректної роботи серверу підключаються необхідні модулі. У кінці сервер починає прослуховування на вказаному порті.

Лістинг 3.21 – Конфігурація адмін панелі

```

const adminConfig = new AdminBro({
  databases: [db],
  rootPath: '/admin',
  resources:[
    {resource: db.Firms,
      options: {
        properties: {
          firmName: {
            isTitle: true,
          }},
    },
    {resource: db.Categories,
      options: {
        properties: {
          categoryName: {
            isTitle: true,
          }},
    },
    {resource: db.Characteristics,
      options: {
        properties: {
          characteristicName: {
            isTitle: true,
          }},
    },
    {resource: db.Products,
      options: {
        properties: {
          productName: {
            isTitle: true,

```

```

        }},
    ],
    branding: {
        logo: '',
        companyName: 'Sirenko admin panel'
    }
});

```

Лістинг 3.22 – Вміст головного файлу серверу

```

const express = require('express');
const http = require('http');
const app = express();
const port = process.env.PORT || 3001;
const router = require('./server/router');
const bodyParser = require('body-parser');
const db = require('./server/models');
const AdminBro = require('admin-bro');
const AdminBroSequelize = require('@admin-bro/sequelize');
const AdminBroExpress = require('@admin-bro/express');
const cors = require('cors');
const morgan = require('morgan');

AdminBro.registerAdapter(AdminBroSequelize);
const ADMIN = {
    email: process.env.ADMIN_EMAIL || 'admin@gmail.com',
    password: process.env.ADMIN_EMAIL || 'adminadmin',
};

const routerAB =
AdminBroExpress.buildAuthenticatedRouter(adminConfig, {
    cookieName: process.env.ADMIN_COOKIE_NAME || 'what',
    cookiePassword: process.env.ADMIN_COOKIE_PASSWORD ||
'sdfwrsjh73y91491428798174ibiBFEBFJHBWEjebGRBJRESB',
    authenticate: async (email, password) => {
        if (email === ADMIN.email && password === ADMIN.password)
        {
            return ADMIN;
        }
        return null;
    }
});

app.use(cors());
app.use(adminConfig.options.rootPath, routerAB);
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));
app.use(morgan('dev'));
app.use(router);

const server = http.createServer(app);
server.listen(port, () => {
    console.log(`App listening at http://localhost:${port}`);
});

```

3.4 Тестування розробленої системи

3.4.1 Функціональне тестування

У результаті перевірки можна оцінити працездатність розробленої системи. Було проведено тестування функціональності веб-додатку.

При завантаженні веб-додатку відбувається очікування завантаження товарів. В результаті можна побачити усі доступні товари в системі (рис. 3.7).

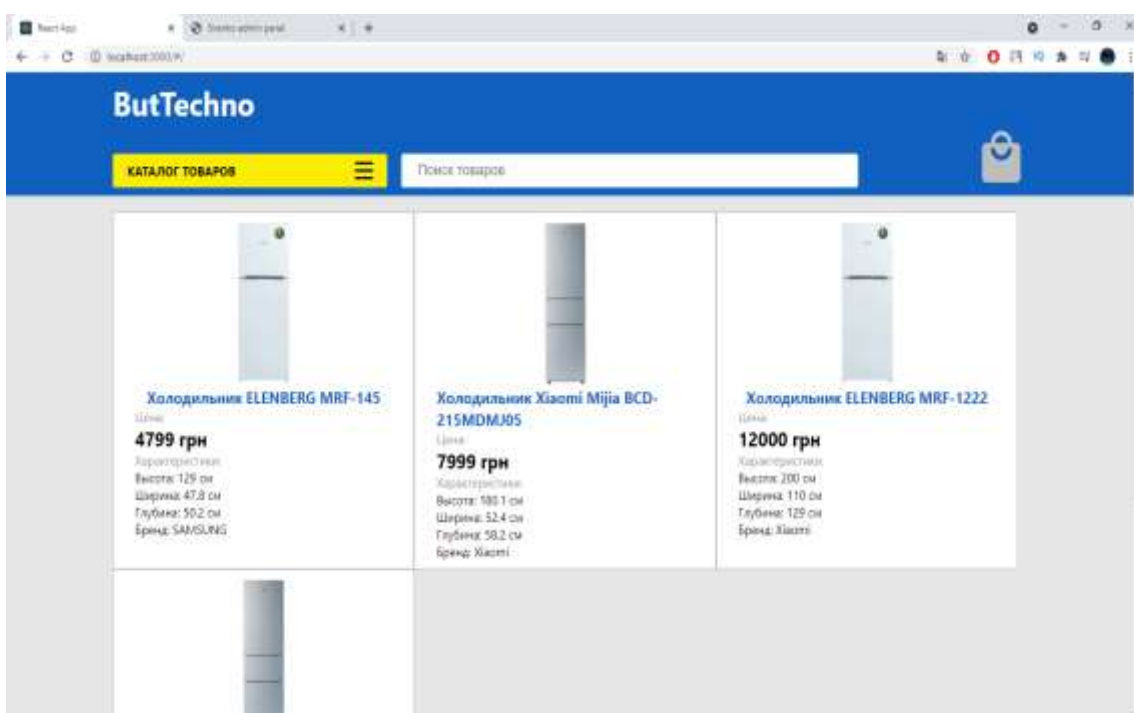


Рисунок 3.7 – Головна сторінка веб-додатку

Також, усі товари що відображаються на головній сторінці можна переглянути через адмін панель веб-додаку (показано на рис. 3.8).

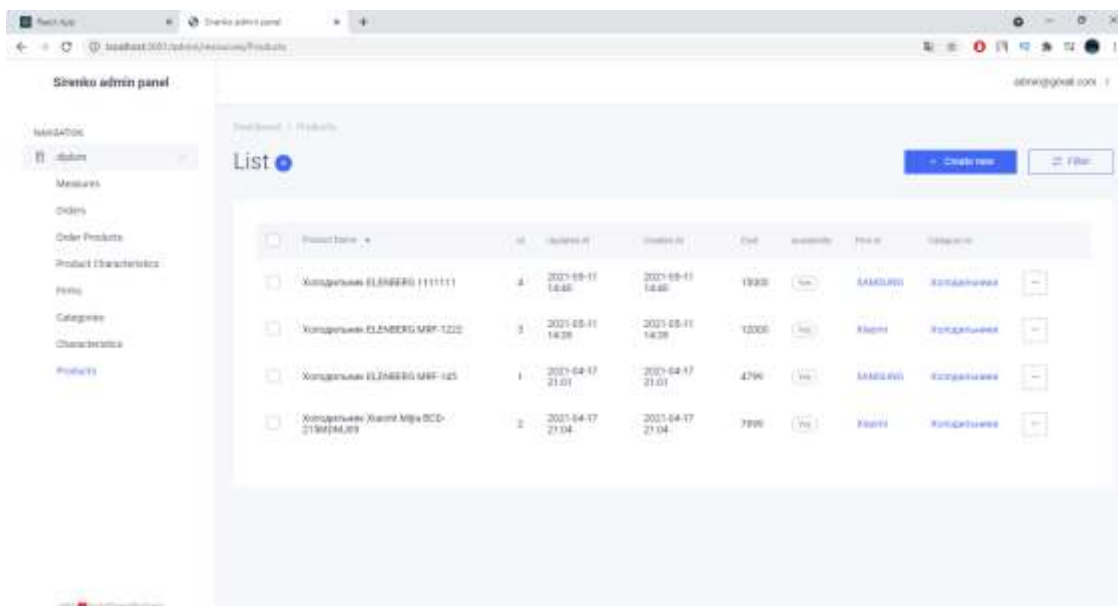


Рисунок 3.8 – Таблиця товарів в адмін панелі

Якщо натиснути на кнопку «Катгорії товарів» відобразиться список доступних категорій веб-додатку. Після натискання на категорію відобразяться усі доступні товари у відповідній категорії. Також з'являється додаткове меню фільтрів товарів, яке містить усі можливі фільтри за товарами цієї категорії (показано на рис. 3.9).

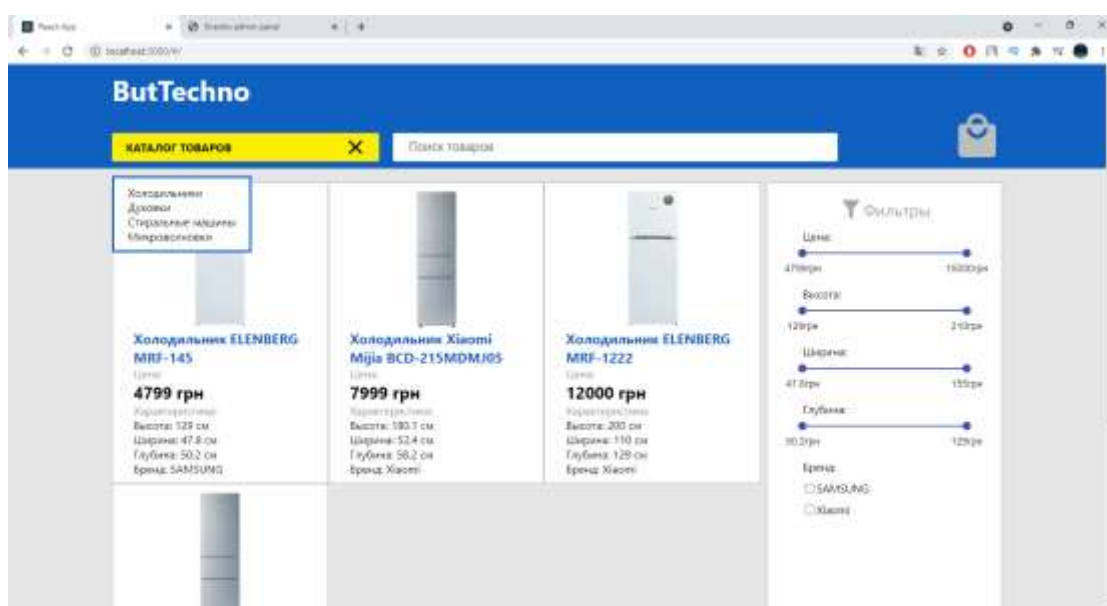


Рисунок 3.9 – Вибір категорії товарів

При встановленні фільтрів можна побачити як відбувається фільтрація товарів (показано на рис. 3.10).

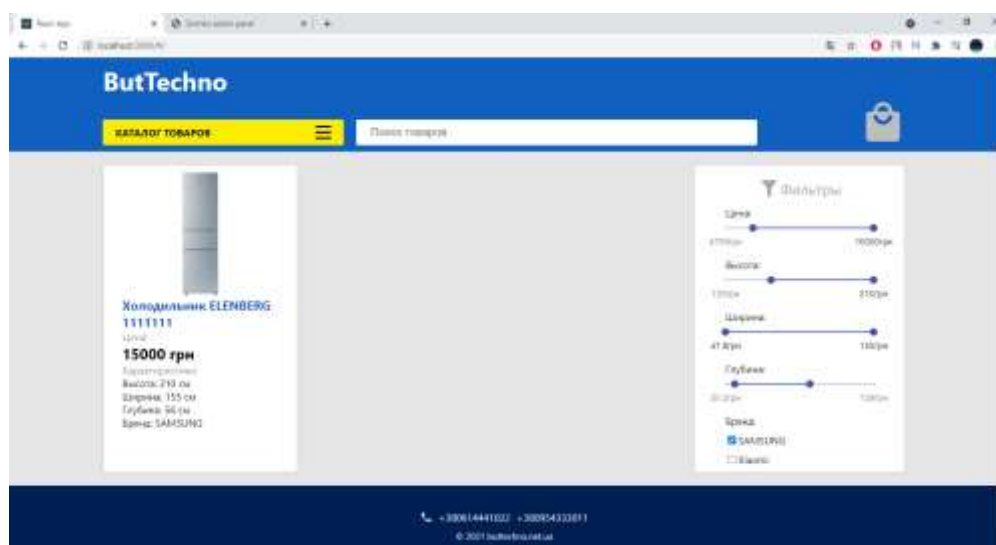


Рисунок 3.10 – Фільтрація товарів

Якщо навести курсором миші на товар, з'явиться кнопка додавання відповідного товару до кошика. При натисканні на кнопку товар з'явиться у кошику, а у замого значку з'явиться кількість товарів, що знаходяться у кошику (показано на рис. 3.11).

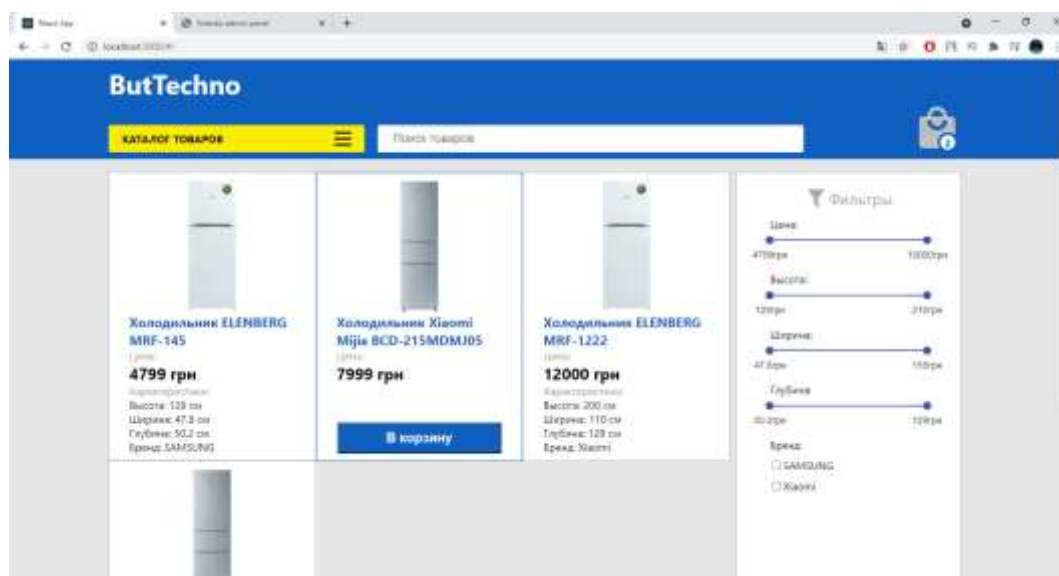


Рисунок 3.11 – Додавання товару до кошика

Якщо натиснути на значок кошика, відобразяться усі товари які присутні у кошику, загальна сума замовлення та кнопка переходження до створення замовлення (показано на рис. 3.12).

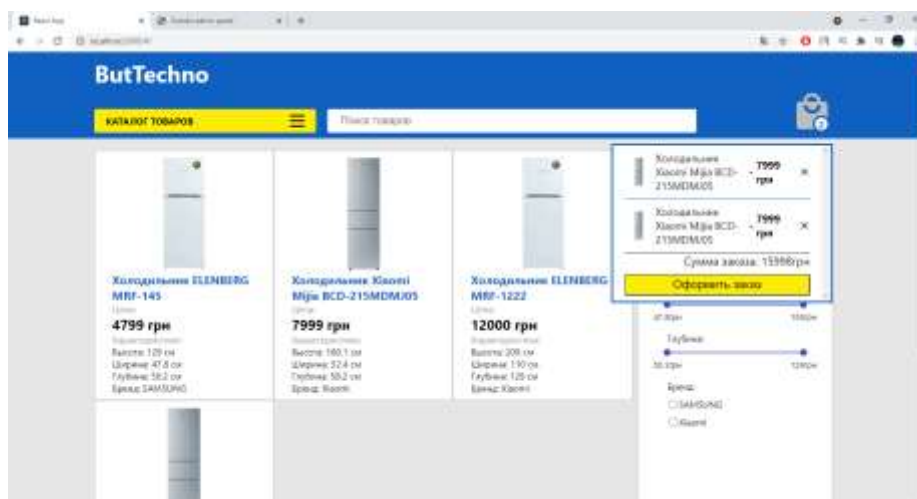


Рисунок 3.12 – Кошик

3.4.2 Тестування корсбраузерності

Процес оптимізації системи передбачає не тільки роботу над його вмістом, а й відповідальний підхід до аналізу його технічного стану. Необхідно обов'язково провести тестування веб-додатку на кросбраузерність.

Веб-додаток без будь-яких зауважень працює у браузері Microsoft Edge (показано на рис. 3.13).

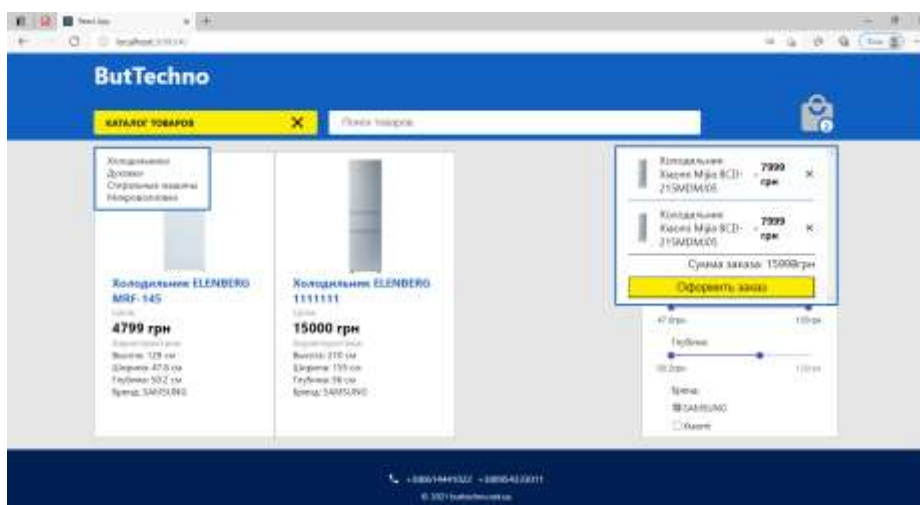


Рисунок 3.13 – Веб-додаток у браузері Microsoft Edge

Також, веб-додаток працює у браузері Opera (показано на рис. 3.14).

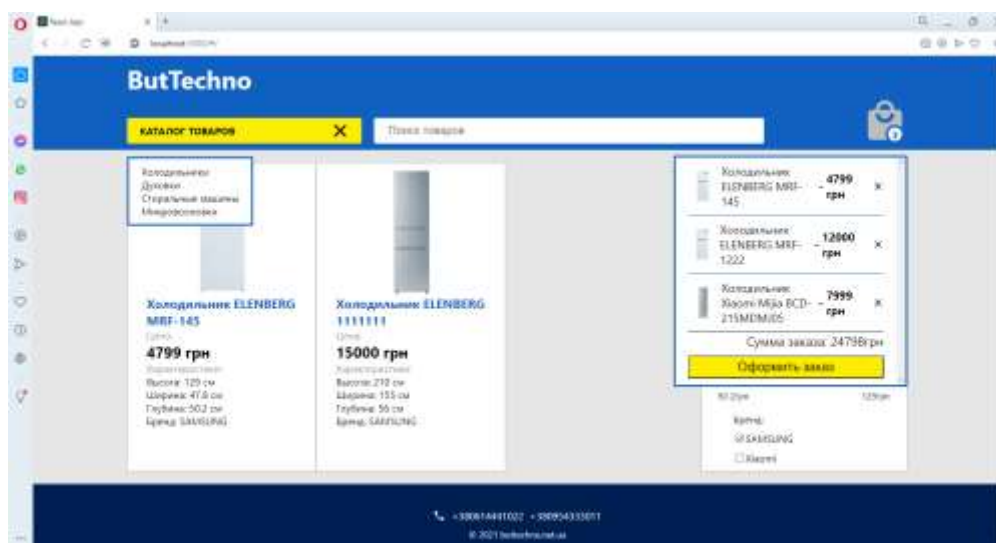


Рисунок 3.14 – Веб-додаток у браузері Opera

3.4.2 Тестування адаптивності

Адаптивність веб-додатку це дуже важлива складова розробки, оскільки користувачі мають різні розміри гаджетів та екранів.

В результаті тестування була перевірена адаптивність веб-додатку до екранів маленького ноутбуку, планшету та самого маленького смартфона.

Веб-додаток коректно відображає блоки даних на маленькому ноутбуці з шириною екрану 1024 пікселі (показано на рис. 3.15).

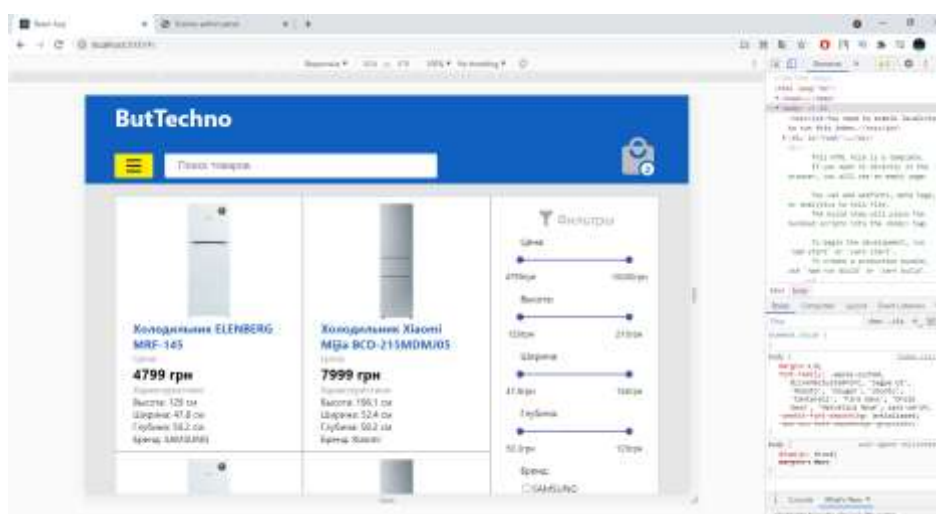


Рисунок 3.15 – Відображення веб-додатку на екрані маленького ноутбуку

На екрані планшету, що має ширину 768 пікселів немає ніяких дефектів (показано на рис. 3.16).

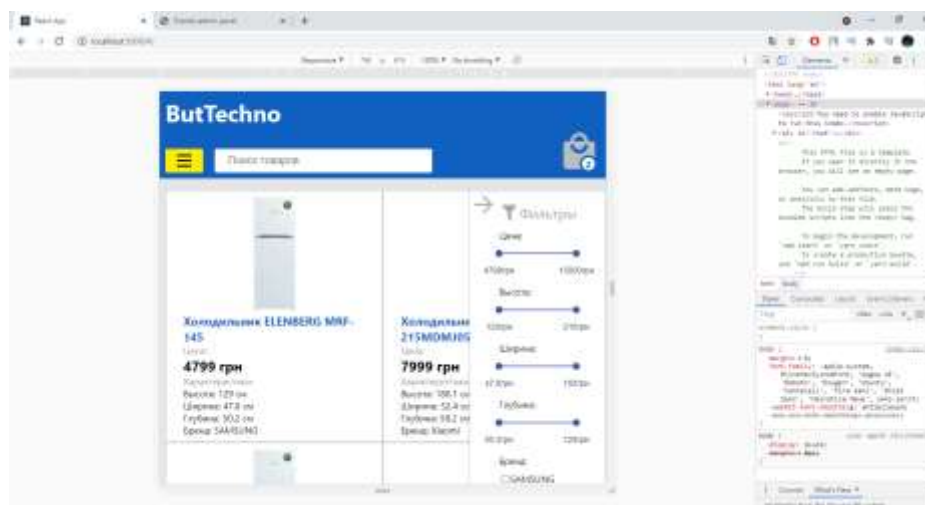


Рисунок 3.16 – Відображення веб-додатку на екрані планшету

На самому маленькому смартфоні, що має ширину екрану 320 пікселів усі блоки даних відображаються коректно (показано на рис. 3.17).

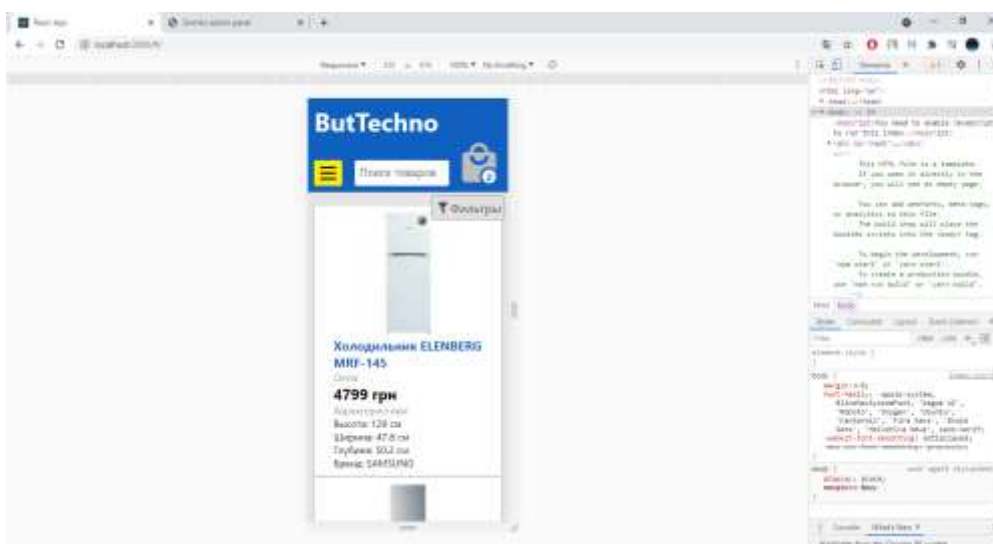


Рисунок 3.17 – Відображення веб-додатку на екрані самого маленького смартфона

ВИСНОВКИ

В ході аналізу предметної області були розглянуті типи магазинів, переваги інтернет-магазину перед оффлайн, та веб-додатки аналоги.

Основні переваги інтернет-магазину:

- торгівля в електронному варіанті - мінімум накладних витрат;
- можливість тримати під контролем дані по проданих товарах, рекламі;
- товари коштують дешевше, ніж в стаціонарних торгових точках;
- зручність для покупців. Мінімальні часові затрати;
- анонімність.

В результаті порівняння SQL та NoSQL баз даних було обрано реляційну. В ході проведення порівняння СУБД PostgreSQL та SQLite було обрано PostgreSQL.

Були розглянуті можливості та зручності мови програмування JavaScript.

В результаті порівняння існуючих фреймворків для створення веб-додатків в якості фреймворку фронтової частини додатку був обраний ReactJS.

В якості мови програмування серверної частини веб-додатку використовувався JavaScript, який виконується за допомогою середовища виконання NodeJS.

В якості фреймворку серверної частини веб-додатку був обраний Express.

В якості ORM-бібліотеки для зв'язку об'єктно-орієнтованого коду з базою даних був Sequelize.

В ході проектування системи були змодельовані сценарії роботи для ролей: гість, адміністратор. Створені діаграми послідовностей, які відображають взаємодії об'єктів впорядкованих за часом.

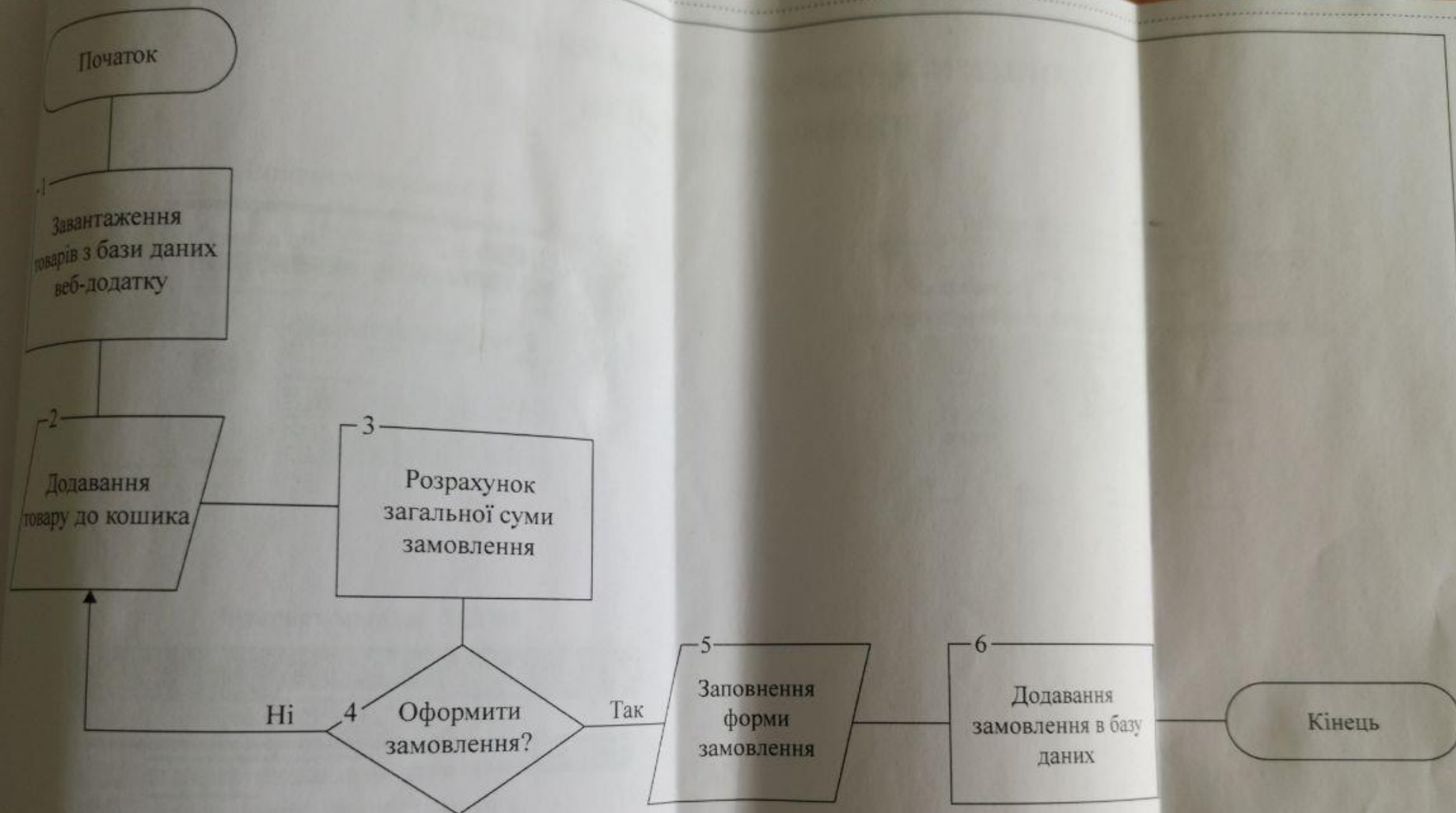
В результаті проектування графічного інтерфейсу був створений унікальний, привабливий та зручний інтерфейс.

В результаті програмної реалізації був розроблений сучасний веб-додаток інтернет-магазину побутової техніки зі зручним, зрозумілим інтерфейсом, високою швидкістю роботи та унікальним дизайном.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1 Інтернет-магазин ФБК. URL: <http://www.xn--90as2a.zp.ua/holodilniki.html>
- 2 Інтернет-магазин buttehnika. URL: <https://byttehnika.zp.ua/>
- 3 Інтернет-магазин XCOM URL: <https://xcom.ua/>
- 4 SQL vs. NoSQL Databases: What's the Difference? URL: <https://www.upwork.com/resources/sql-vs-nosql-databases-whats-the-difference>
- 5 SQLite. URL: <https://lecturesdb.readthedocs.io/databases/sqlite.html>
- 6 Sqlite vs. MySQL vs. PostgreSQL: A Comparison of Relational Databases. URL: <https://logz.io/blog/relational-database-comparison/>
- 7 Що таке HTML? URL: <https://futurenow.com.ua/shho-take-html/>
- 8 Учебник HTML и CSS от Трепачёва Дмитрия. URL: <http://old.code.mu/books/css/>
- 9 Учимся верстать: что такое CSS. URL: https://skillbox.ru/media/code/chto_takoe_css/
- 10 Основы JavaScript - Вступ в JavaScript. URL: <https://www.coursehero.com/file/61100735/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%B8-JavaScriptdocx/>
- 11 The JavaScript Beginner's Handbook (2020 Edition). URL: <https://www.freecodecamp.org/news/the-complete-javascript-handbook-f26b2c71719c/>
- 12 Обзор Vue.js. URL: <https://timeweb.com/ru/community/articles/obzor-vue-js-1>
- 13 AngularJS. URL: <https://www.whitelabeldevelopers.ru/angularjs>
- 14 Обзор ReactJS. URL: https://club.cnews.ru/blogs/entry/obzor_reactjs
- 15 React или Angular или Vue.js. URL: <https://habr.com/ru/post/476312/>
- 16 Выбор лучшего Node.js фреймворка. URL: <https://umbrellait.com/ru/blog/choosing-the-best-nodejs-framework/>
- 17 Sequelize. URL: <https://metanit.com/web/nodejs/9.1.php>

18 Введение в ORM (Object Relational Mapping) URL:
<http://internetka.in.ua/orm-intro/>



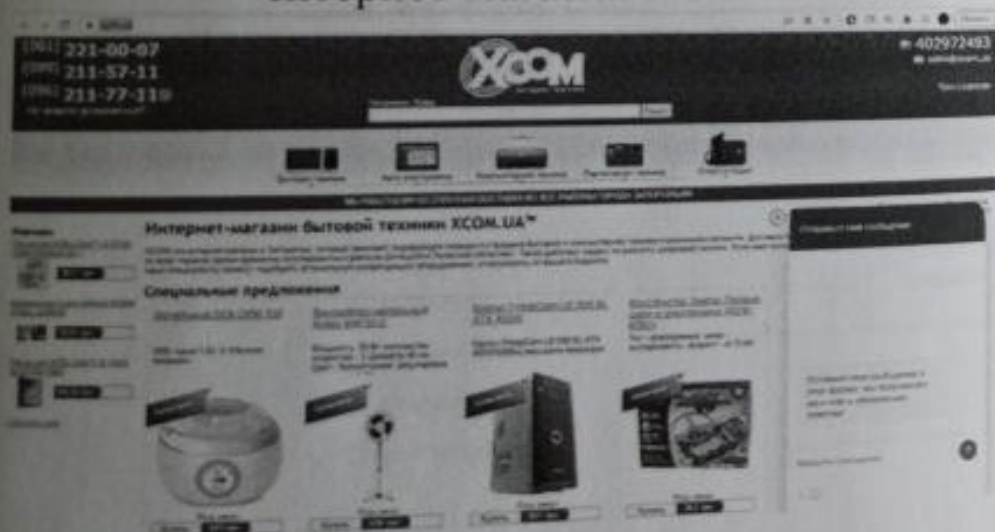
					13.02070849.00041 A1		
Зам.	Лист.	№ докум.	Підпис	Дата	Розробка інтернет-магазину побутової техніки за допомогою технологій ReactJS та NodeJS Алгоритм роботи веб-додатку		
Розроб.		Степанко В.В.	<i>В.В.</i>				
Перевір.		Касьян К.М.	<i>К.М.</i>				
Техніч.							
Наступ.		Шербак Н.В.	<i>Н.В.</i>		Лист 1 Листів 1		
Заверш.		Куценко Р.К.	<i>Р.К.</i>				
					НУ "Запорізька політехніка" КНТ-527		

Огляд аналогів інтернет-магазину побутової техніки

Интернет-магазин ФБК



Интернет-магазин XCOM



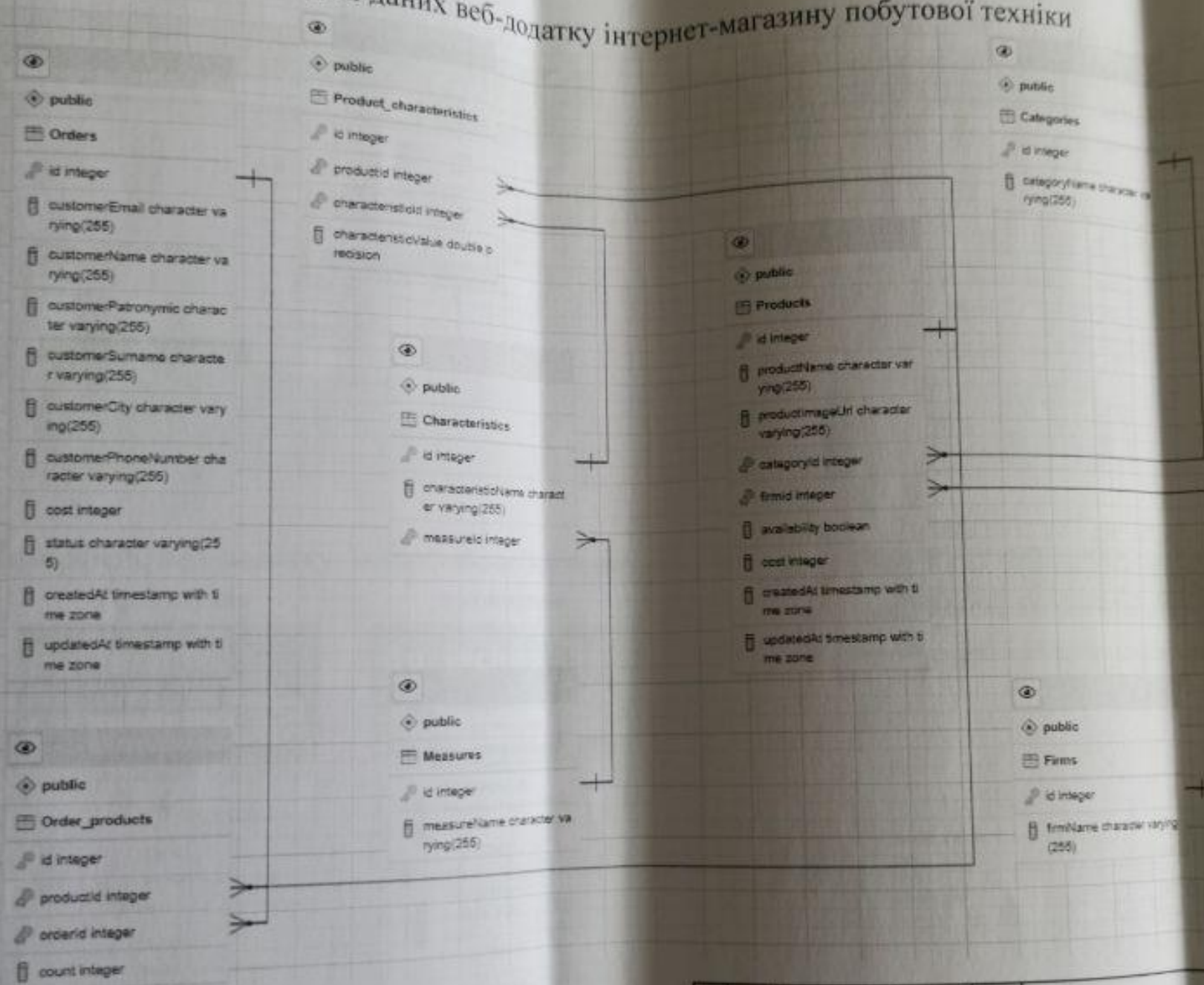
Интернет-магазин Byttechnika



						13.02070849.00041 ПЛ1					
						Розробка інтернет-магазину побутової техніки за допомогою технологій ReactJS та NodeJS Огляд аналогів інтернет-магазину побутової техніки	Лист		Маса	Масштаб	
Зак.	Лист	№ докум.	Підпис	Дата			Лист 1		Листів 1		
Розроб.		Степан В.В.									
Перевір.		Косин К.М.									
Техніч.											
Начальн.		Шербак Н.В.							НУ "Запорізька політехніка" КНТ-527		
Затвердж.		Кудриченко Р.К.									

Структура бази даних веб-додатку

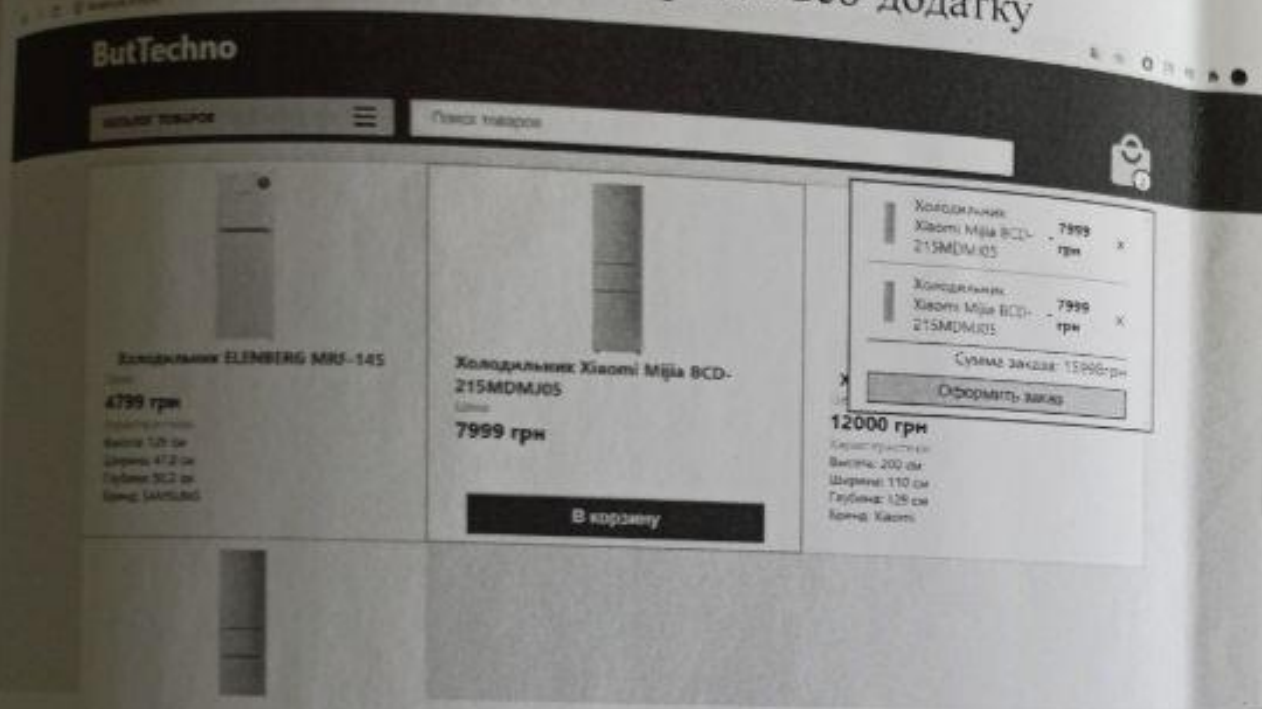
Схема бази даних веб-додатку інтернет-магазину побутової техніки



13.02070849.00041 ПЛ2													
Зам.	Лист	№ докум.	Підпис	Дата									
Розроб.		Сиренко В.В.	<i>[Signature]</i>										
Перевір.		Касьян К.М.	<i>[Signature]</i>										
Т.напр.													
Н.напр.		Шербат Н.В.	<i>[Signature]</i>										
Заверш.		Кудерметов Р.К.	<i>[Signature]</i>										
Розробка інтернет-магазину побутової техніки за допомогою технологій ReactJS та NodeJS Структура бази даних веб-додатку													
<table border="1"> <tr> <td>Лист</td> <td>Маса</td> <td>Масштаб</td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td>Лист 1</td> <td colspan="2">Листів 1</td> </tr> </table>					Лист	Маса	Масштаб				Лист 1	Листів 1	
Лист	Маса	Масштаб											
Лист 1	Листів 1												
НУ "Запорізька політехніка" КНТ-527													

Інтерфейс веб-додатку

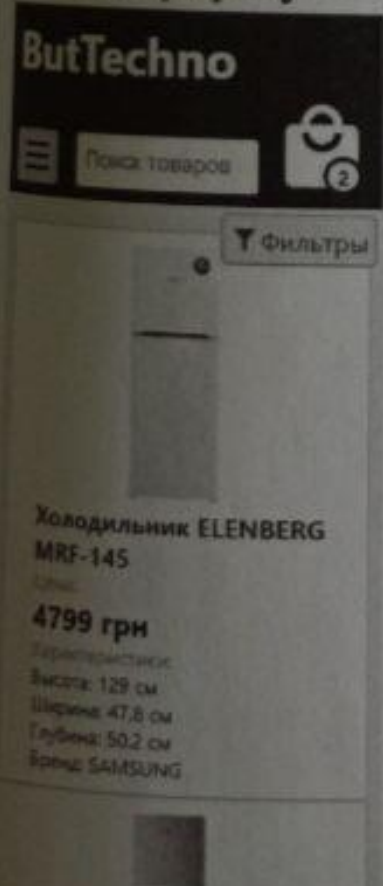
Інтерфейс головної сторінки веб-додатку



Інтерфейс сторінки веб-додатку за категорією товару



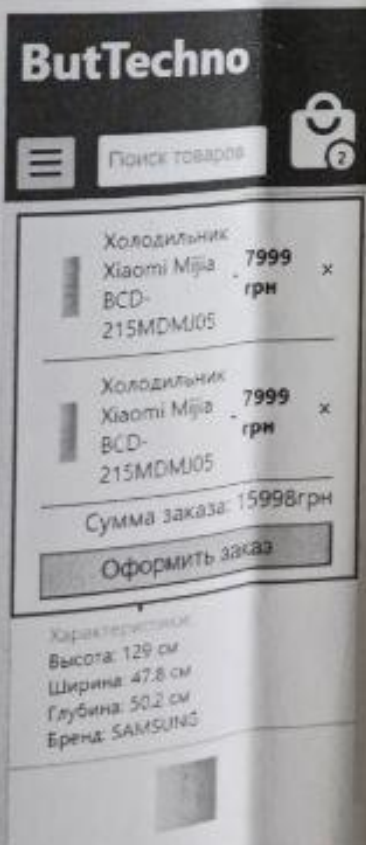
Інтерфейс веб-додатку зі смартфона



Інтерфейс веб-додатку з фільтром



Інтерфейс веб-додатку з кошиком



Інтерфейс веб-додатку з планшета



						13.02070849.00041 ПЛЗ			
						Розробка інтернет-магазину побутової техніки за допомогою технологій ReactJS та NodeJS Інтерфейс веб-додатку			
Вам.	Лист	№ докум.	Підпис	Дата		Лист	Маса	Масшт.	
Розроб.		Сиренко В.В.							
Перевір.		Косиш К.М.							
Тестув.						Лист 1	Листів 1		
Начальн.		Шербак Н.В.				НУ "Запорізька політехніка" КНТ-527			
Замовляюч.		Кудеристов Р.К.							