МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОЄКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ ЦИФРОВИХ ГАДЖЕТІВ

Здобувач вищої освіти гр. ІН.м-13 Олексій ПОКУТНІЙ

Науковий керівник:
кандидат технічних наук, доцент,
доцент кафедри комп'ютерних наук Ігор ШЕЛЕХОВ

В.о. завідувача кафедри
кандидат технічних наук, доцент,
доцент кафедри комп'ютерних наук Ігор ШЕЛЕХОВ

СУМИ 2022

Сумський державний університет

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:
В.о. зав.кафедри _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Покутньому Олексію Юрійовичу

1. Тема роботи *Інформаційна технологія проєктування інтернет-магазину цифрових гаджетів*
затверджую наказом по СумДУ від «____» _____ 20 ___ р. № _____
2. Термін здачі здобувачем кваліфікаційної роботи _____
3. Вхідні дані до кваліфікаційної роботи _____
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Вступ; 2) Інформаційний огляд; 3) Постановка завдання; 4) Вибір методу рішення;
5) Інформаційне та програмне забезпечення; 6)Висновок; 7) Література; 8) Додаток.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р. Керівник _____

Завдання прийняв до виконання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми та постановка задачі*		
2	*Вибір методу рішення*		
3	*Інформаційне та програмне забезпечення*		
4	*Оформлення кваліфікаційної магістерської роботи*		

Здобувач вищої освіти _____

Науковий керівник _____

РЕФЕРАТ

Записка: 52 стор., 20 рис., 17 джерел.

Об’єкт дослідження — процес інформаційного аналізу і синтезу системи електронної комерції.

Мета роботи – розробка комплексу методів, моделей та засобів проєктування системи електронної комерції.

Методи дослідження – методи проєктування інформаційних систем, методи нормалізації баз даних, засоби веб-дизайну, об’єктно-орієнтована парадигма програмування, методи тестування інформаційних систем.

Результати –проведено розробку та програмну реалізацію веб-орієнтованої системи електронної комерції. При цьому виконано аналітичний огляд сучасних технологій та фреймворків проєктування веб-орієнтованих систем, розроблено інформаційну модель системи електронної комерції з врахуванням особливості реалізації цифрових гаджетів, визначено структуру бази даних, перевірка працездатності розробки проведено шляхом реалізації інтернет-магазину «eShop».

ВЕБ-ДОДАТОК ДЛЯ ПОКУПКИ ГАДЖЕТІВ ОНЛАЙН, МЕТОД ПЕРЕГЛЯДУ, WEB APPLICATION FOR BUYING GADGETS ONLINE, JAVASCRIPT, REACTJS, SQL

ЗМІСТ

ВСТУП

За останніми даними аудиторія в інтернеті стрімко зростає, а продажі через інтернет у великих містах досягають до 25%, при цьому фахівці підкреслюють тенденцію до зростання продажів саме через інтернет. Щорічно кількість інтернет-магазинів збільшується, так як це дійсно вигідно і зручно для покупця, не кажучи про економії бюджету і часу. Інтернет-магазин працює цілодобово і може продавати певні товари в автоматичному режимі без участі продавця. До переваг так само можна віднести те, що не треба закуповувати товар заздалегідь, а це істотна економія, на складських приміщеннях. Потрібно лише домовитися з постачальниками, і в потрібний момент, просто викупити товар, який у вас замовлять. У порівнянні зі звичайним магазином, територія продажів якого обмежується населенням міста або району, територія охоплення інтернет-магазину збільшується на всю Україну і українську аудиторію в інших країнах, адже товар можна доставляти не тільки кур'єрською службою, але і поштою.

На даний момент існує багато інтернет-магазинів цифрових гаджетів. У кожного з них є красиве оформлення, достатньо багатий вибір різноманітних гаджетів та аксесуарів. Але в них у них є достатньо мінусів:

1. Реклама (при вході на сторінку, вспиливаюче вікно, при натисканні на невідому кнопку);
2. Ціна частіше за все перебільшені;
3. Недостатньо детальної інформації про товар;
4. Більшість сайтів неоптимізована, через це, користувач довго чекае завантаження сайту або навіть бачить перед собою пусту сторінку.

1 ЗАГАЛЬНІ ВІДОМОСТІ

1.1 Теоретичні дані

1.1.1 Визначення інтернет-магазину

Інтернет-магазин — це місце в Інтернет мережі, де здійснюється продаж продукту безпосередньо споживачеві, включно з доставкою. Розміщення споживчої інформації, замовлення відповідного товару і оплата відбуваються всередині самої мережи.

Найбільшою проблемою реалізації інтернет-магазину лежить між технологією продажу в Інтернеті та традиційної комерційної діяльності. Під час звичайної торгівлі покупець звик мати можливість оцінити товар візуально, «помацати» його, визначити якість. Нажаль, під час електронної купівлі він такої можливості немає. Частіше за все візуальної інформації достатньо, але в такому разу мають вплив емоційні і психологічні чинники.

1.1.2 Переваги та недоліки віртуальних магазинів

Переваги віртуальних магазинів

Інтернет-магазин має багато переваг перед звичайним магазином. Через зменшення обсягу взаємодії з клієнтами, скорочення персоналу дешевше і простіше орендувати дисковий простір і виставляти «електронні вітрини», ніж орендувати торгову площу і розставляти товари на полицях, не потрібно касове обслуговування тощо.

Для споживачів головною перевагою інтернет-магазину є економія часу.

Люди, які працюють 6 днів на тиждень з 10 ранку до 7 вечора, іноді не мають часу сходити в магазин. Інтернет-магазини дозволяють робити покупки в будь-який час, не виходячи з офісу, і якщо він точно знає, що хоче купити, вибір і замовлення товару займе у нього кілька хвилин. Служба доставки інтернет-магазину доставить вибрані товари в зручний час і місце. Крім того, вибір і оцінка атрибутів товару в інтернет-магазинах відбувається швидше і зручніше, ніж у звичайних магазинах. Уявіть, що ви прийшли в звичайний магазин побутової техніки і хочете порівняти параметри 20-30 мобільних

телефонів. Для цього варто вивчити всі цінники: запам'ятати характеристики, ціну і назву моделі. Якщо продавець-консультант не зайнятий, ви можете зв'язатися з ним і попросити представити всі ці товари. Але зазвичай жоден продавець не має стільки вільного часу, щоб мати справу з одним клієнтом, і не може бути гарантовано, що він володіє всією інформацією. В інтернет-магазині можна нескінченно збільшувати асортимент і кількість інформації.

Крім того, різноманітність інтернет-магазинів не підлягає жодним обмеженням (наприклад, різноманітність звичайних магазинів обмежена площею торгових залів). Якщо доступний пошук за параметрами, ви можете просто вказати характеристики, яким повинен відповідати товар, і вибрати зі списку моделей, які відповідають вашим вимогам. Крім того, інтернет-магазин також може публікувати рейтинги покупок, пропозиції та огляди продуктів, статті про продукти та іншу інформацію, щоб визначити ваш вибір. Наступний плюс **економія грошей.**

Операційні витрати інтернет-магазину, включаючи доставку, значно нижчі, ніж у звичайних магазинах. На відміну від звичайних магазинів, інтернет-магазини можуть обслуговувати сотні покупців одночасно. Хоча на практиці з такою ситуацією не зіткнешся. Крім того, якщо покупець проживає в іншому місті, він матиме можливість безкоштовно консультуватись за кордоном. Всю інформацію він може знайти на сторінках інтернет-магазину.

До недоліків інтернет-магазинів для покупців можна віднести:

• Ви не можете «доторкнутися», ви не можете дізнатися більше, ніж те, що можете написати (наприклад, меблі, одяг);

• Питання щодо гарантії та підтримки;

• Терміни доставки часто тривалі (до 6 тижнів), особливо якщо мова йде про міжнародні доставки.

Головний недолік електронної комерції – неякісні товари. Так, дослідження проводилося на початку 2010 року. Згідно з дослідженням

британської компанії ArmorGroup, 35% речей, що продаються в інтернет-магазинах, є підробками.

1.1.3 Класифікація віртуальних магазинів

Віртуальні магазини можна класифікувати за різними ознаками. Найважливішою є класифікація за моделлю бізнесу:

• онлайн-магазин;

• Поєднання офлайн-і онлайн-присутності (коли інтернет-магазин базується на вже діючій реальній торговій структурі).

Про постачальника:

• Власний склад (наявність фізичної інвентаризації)

• Робота за договором з постачальниками (великих власних складів немає).

Класифікація за асортиментом - книги, аудіо-, відеокасети, CD, DVD, комп'ютери, побутова техніка, цифрові товари.

Серед способів роздрібної торгівлі товарами в Інтернеті можна виділити наступні способи:

• інтернет-магазин;

• Web-вітрини;

• торговий автомат.

Інтернет-вітрина більше схожа на рекламний сервер. Інформація про товар відображається на вітрині і постійно оновлюється. Його створення та управління може бути дуже дешевим, а практична користь від такої презентації очевидна. Але це ще не угода. Відвідавши вітрину, потенційні покупці повинні зателефонувати в компанію, оплатити продукцію та домовитися про доставку. Тому інтернет-вітрина виправдана в тих випадках, коли покупця необхідно познайомити зі складним товаром, на вивчення якого він витратить багато часу в торговому залі.

Торговий автомат виконує не тільки функцію вітрини, але й приймає замовлення та передає їх менеджеру, тобто оформляє замовлення та виставляє рахунок за відсутності покупця.

Автомагазин - це ефективне та комплексне рішення в торговому бізнесі. Він не тільки виставляє рахунки, але й відстежує замовлення, приймає електронні платежі та створює заявки на доставку товарів клієнтам. Тут завдання адміністратора — контролювати роботу системи, яку важко підтримувати.

1.2 Інформація про бренд

1.2.1 Назва інтернет-магазину

Вибираючи назву для свого інтернет-магазину, слід враховувати наступні фактори:

• Назви не повинні бути плагіатом;

• Має бути креативним;

• Повинна легко запам'ятовуватися;

Після проведення дослідження ринку та виявлення назв компаній-конкурентів було обрано кілька варіантів назви магазину, з яких обрано найкращий – «eShop».

1.2.2 Асортимент товарів

Асортимент товарів, що продаються в інтернет-магазині - цифрова продукція. Але це будуть не прості комп’ютери та ноутбуки, а ігрові приставки, тобто оснащені потужними модулями для якісної та тривалої гри. Існує багато ігрових комп’ютерів, але не всі вони якісні і можуть використовуватися протягом тривалого часу.

Серед ігрових ноутбуків, що задовольняють потреби покупців, на сучасному ринку сяють деякі моделі «ASUS», «Acer», а також «HP» і «Lenovo». Такі моделі будуть виставлені на вітринах інтернет-магазину.

Коли мова заходить про ігрові ПК, важко вибрати найкращі бренди за всіма критеріями, оскільки всі вони достатньо потужні та якісні. Тому в

інтернет-магазині будуть представлені комп'ютери різних компаній, таких як: «ARTLine», «Everest», «Cobra» і «QUBE».

Також магазин може знайти прихильників серед людей, які хочуть придбати якісний телефон від відомого бренда “Apple”. Автор користувався різними телефонами: Lenovo, Sony, Nokia. Але, спробувавши техніку від Apple, ніякий телефон, на думку автора, не зможе працювати так як IPhone.

Отже асортимент товару в інтернет-магазині буде наступний:

- Ігрові комп’ютери;
- Ігрові ноутбуки;
- Телефони;

1.2.3 Унікальність

Кожен з всесвітньовідомих підприємств, незалежно від сфери його діяльності, приваблює своїх клієнтів однією серед трьох переваг, таких як:

- Якість;
- Ціна;
- Зручність;

Компанія MacDonald's виділяється своєю зручністю серед інших закладів швидкого харчування. По-перше, це мережа ресторанів, які мають філії по всьому світу в кожному великому місті. Тож у Києві, де б ви не були, MacDonald's завжди поруч. По-друге, в кожному з ресторанів дуже розвинений менеджмент: прийшовши до закладу, рідко помічаєш великі черги чи тривалий час, щоб оформлення замовлення. Також є можливість замовити їжу самостійно через електронний автомат, або MacDrive, під’їхавши на власному авто ви можете замовити їжу, не виходячи із власного автомобіля.

AliExpress приваблює клієнтів своєю ціною на товари. У магазині можна купити речі за невелику ціну, а вибір товарів достатньо великий. Із зручністю є невеликі проблеми: замовлення можна чекати більше місяця. Саме через привабливу ціну сайт, має поціновувачів у різних країнах світу.

Компанія Apple має товар першокласної якості. Вона з самого початку свого виходу на ринок відразу знайшла прихильників саме завдяки якості продукції. Згідно зі статистичними даними, серед 100 відсотків тільки 6 виходять з ладу не з вини користувача в перший рік використання. Так, ціна телефону значно вища порівняно з іншими брендами, та й фірмові модулі для заміни в разі поломки теж коштують недешево. Також, маючи телефон досить старої моделі, Apple перестає підтримувати оновлення операційної системи, яка не підтримує використання більшості додатків. Що змушує користувача купувати більш нову версію обладнання. Але, незважаючи на це, Apple є найпопулярнішим постачальником цифрових гаджетів у всьому світі.

Якби одна з перерахованих вище компаній вирішила зробити вибір на користь іншої переваги або навіть схрестити обидві відразу, вона зазнала б краху майже відразу.

Інтернет-магазин eShop матиме таку перевагу перед іншими магазинами як якість. На думку автора, така річ, як техніка, перш за все має бути надійною. Зараз смартфони – це те, без чого неможливо працювати, вчитися та підтримувати спілкування, тому без надійної та якісної техніки – як без рук.

1.2.4 Постачання

Оскільки основною перевагою компанії eShop буде якість, товари повинні бути надзвичайно надійними і обов'язково оригінальними. Тому постачальниками обладнання для вітрини інтернет-магазину будуть тільки компанії, які виробляють відповідний товар. У зв’язку з тим, що більшість техніки, яка буде виставлена – американського виробництва, варто враховувати ціни на доставку та розмитнення. Тому ціни за одиницю товару будуть вищі, ніж в інших магазинах. Магазин надасть ліцензії від компанії-виробника, тому клієнт буде впевнений в оригінальності гаджета. Також у разі несправності гаджет буде відправлено в сервісний центр компанії-виробника. Отже, скориставшись послугами нашого магазину, клієнт зрозуміє, що впевненість у власній техніці набагато важливіша за ціну.

1.3 Постановка задачі

1.3.1 Предмет розробки

Предметом розробки є фірмовий інтернет-магазин eShop з системою динамічного показу та пошуку цифрових гаджетів.

Призначення сайту:

• Подання інформації про компанію та її діяльність;

• Викладка товарів та інформація про них;

• Ведення торгівлі за допомогою інтернет-порталу;

Метою інтернет-магазину є прямі продажі, але також сайт буде використовуватися для створення іміджу компанії в очах цільової аудиторії та комунікації з клієнтами.

1.3.2 Структура інтернет-магазину

Вибір техніки складний, по-перше, матеріально, по-друге, не кожен клієнт має достатньо знань, щоб вибрати дійсно гідний гаджет. Тому структура сайту має бути лаконічною та зрозумілою для користувача. Тобто вона не повинна «загубитися» на сайті. Щоб уникнути таких випадків, необхідно продумати таку структуру (рис. 1.1).

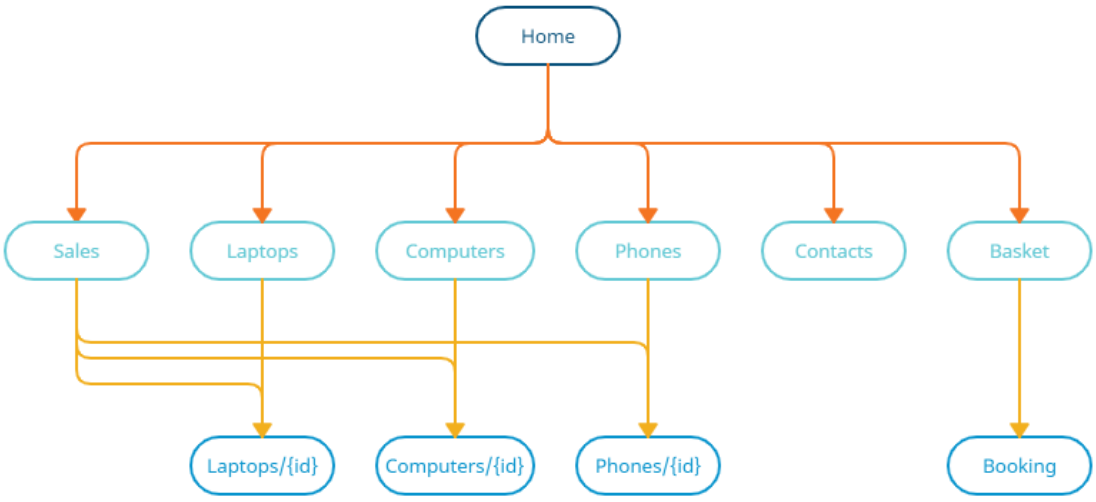

Рисунок 1.1 – Структура інтернет-магазину *eShop*

1.3.3 UX дизайн інтернет-магазину

UX дизайн (User Experience) – це те, який досвід чи враження користувач отримує від роботи з вашим інтерфейсом. Простими словами, це структура самої веб-сторінки без графічно красиво відображених елементів. UI дизайн створюється на основі UX дизайну (рис. 1.2). [1]

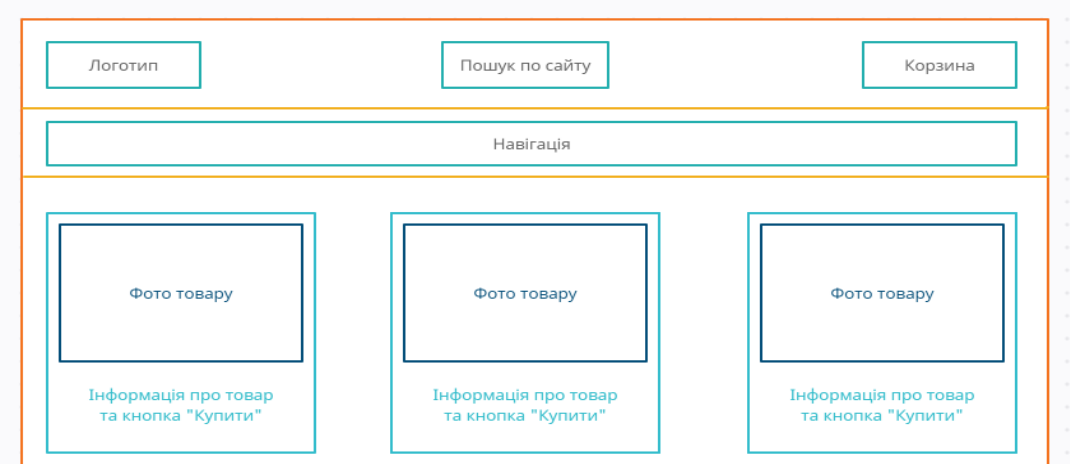

Рисунок1.2 – UX дизайн сайту

1.3.4 UI дизайн інтернет-магазину

UI дизайн (User Interface) — це те, як виглядає інтерфейс і яких фізичних характеристик він набуває. Тобто за допомогою графічних редакторів (PhotoShop / Figma), створення візуального шаблону сайту (як він має виглядати в браузері)(рис.1.3).

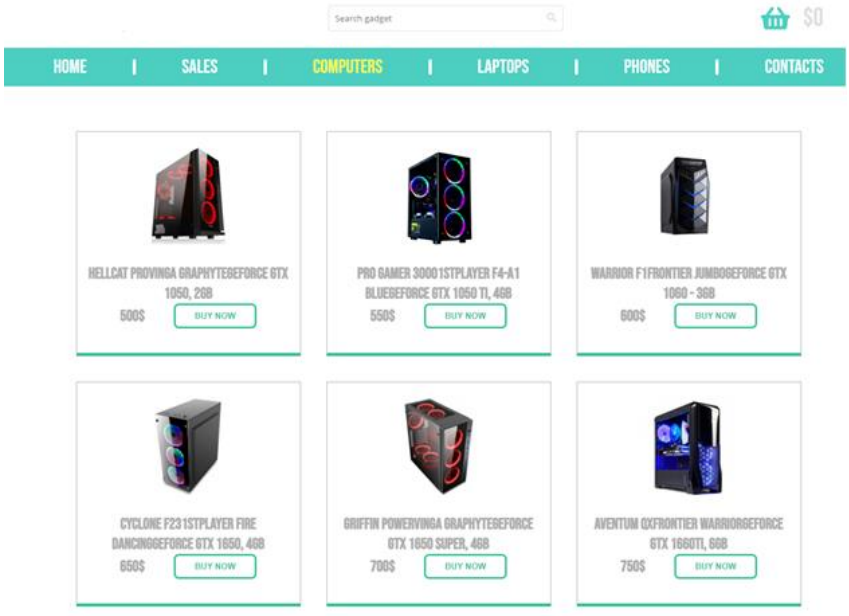

Рисунок 1.3 – UI дизайн

1.3.5 Технічні вимоги

Перелік технічних вимог:

• *Адаптивна верстка.* Сайт повинен виглядати красиво як в десктопній версії, так і в мобільній. Всі блоки не повинні виходити за межі сторінки і між ними не повинно бути зайвих відступів. Також слід попрацювати над текстом, щоб він не був занадто великим або, навпаки, маленьким для обох варіантів;

• *Пошукова система сайту*. Щоб користувач міг за бажанням ввести в поле введення назву товару і, натиснувши кнопку, переглянути список знайдених товарів;

• *Додавання товару в кошик*. На кожній сторінці з детальною інформацією про товар повинна бути кнопка «Купити», при натисканні на яку, відповідний товар повинен бути доданий в магазин;

• *Виймання товару з кошика*. На сторінці «кошик» повинні динамічно відображатися додані товари з інформацією про товар, а також кнопка «вилучити з кошика», при натисканні на яку товар повинен бути видалений зі сховища і з самого кошика.

• *Обробка замовлень*. На сторінці кошика в кінці списку доданих товарів повинна бути кнопка «оформити замовлення», при натисканні на яку користувач повинен перейти на сторінку з формою для заповнення. Поля введення повинні бути перевірені, щоб користувач вводив лише правдиву інформацію. Після натискання кнопки «оформити замовлення» форма з даними: ім'я, прізвище, номер телефону, поштова адреса для доставки, а також масив обраних користувачем товарів відправляється на сервер для обробки. Після відправки дійсної форми має прийти повідомлення у вигляді спливаючого вікна про те, що замовлення успішно оформлено, після чого список доданих товарів у кошик самоочищається.

• *Слайдер, плавні переходи між сторінками, анімація*. Щоб клієнту було цікаво відвідувати сайт, потрібно додати анімацію. Він добре приваблює користувачів своєю анімацією, яка сприяє покупці товару. Також при

завантаженні сторінки всі блоки повинні з'являтися на сайті плавно, що дуже порадує користувача.

1.3.6 Зміст інтернет-магазину

Дані сайту будуть динамічно оновлюватись з бази даних, розробленою розробником. Дані будуть взяті з офіційної характеристики товару, з онлайн ресурсу виробників.

1.3.7 Етапи створення інтернет-магазину

1. Створення концепції, розробка та узгодження технічного завдання;
2. Розробка макета сайту, який включає всі графічні та інтерактивні елементи;
3. Програмування модулів управління;
4. Створення, узгодження, оптимізація контенту;
5. Тест сайту та внесення корегувань;
6. Запуск проекту.

1.4 Висновок

Тому інтернет-магазин – найефективніший спосіб продати свій товар. Існує три основних типи віртуальних магазинів: інтернет-магазини, веб-вітрини та торгові автомати. Кожен має свої переваги і недоліки, інтернет-магазини є найзручнішими, адже для їх роботи менеджеру магазину не потрібно брати участь у оформленні замовлень.

У цьому розділі визначено назву магазину - eShop, плюси, мінуси та категорії товарів. Сьогодні неможливо працювати, вчитися і спілкуватися без новітніх технологій, тому смартфон потрібен кожному. Головна перевага – якість, адже надійність власного смартфона – пріоритет номер один. Постачальниками товарів для магазину будуть тільки компанії, які виробляють відповідну продукцію, що знизить відсоток бракованих гаджетів і підвищить задоволеність клієнтів.

Також у розділі вирішено технічне завдання, згідно з яким відбуватиметься розробка. Мета інтернет-магазину була визначена:

• Вивчити асортимент;

• Отримувати інформацію про товари бренду;

• Продавати товари;

Розроблено структуру сайту: легку для сприйняття користувачем та лаконічну. Було визначено та розроблено дизайн UX та UI. Перераховано та роз’яснено наступні вимоги до функціонування сайту:

• Адаптивна верстка;

• Пошукова система сайту;

• Додавання товару в кошик;

• Вилучення товару з кошика;

• Оформлення замовлення;

• Слайдер, плавні переходи між сторінками, анімація.

Також були описані етапи розвитку даного інтернет-магазину для компанії eShop.

2 ПРОЄКТУВАННЯ ІНТЕРНЕТ МАГАЗИНУ

2.1 Архітектура веб-додатку

Щоб магазин був оптимізований (швидко та якісно працював) і нормально функціонував, необхідно розробити програмну архітектуру.

Архітектура програмного забезпечення — це сукупність найважливіших рішень щодо організації програмної системи. Архітектура включає в себе:

• вибір структурних елементів та їх інтерфейсів, за допомогою яких збирається система, а також їх поведінка в рамках взаємодії структурних елементів;

• з'єднання обраних елементів структури та поведінки у все більші й більші системи;

• архітектурний стиль, який керує всією організацією – усіма елементами, їхніми інтерфейсами, їхньою співпрацею та зв’язком.

Документування архітектури ПЗ спрощує процес спілкування між розробниками, дозволяє документувати прийняті проектні рішення та надавати інформацію про них операторам системи, а також повторно використовувати компоненти та шаблони проекту в інших.

Правильна архітектура має такі характеристики:

• ефективність системи;

• гнучкість системи;

• можливість розширення системи;

• масштабування процесу розробки;

• тестування системи;

• багаторазове використання та простота обслуговування.

Під час розробки архітектури інтернет-магазину «eShop» врахували кілька головних аспектів: залучення до проекту бази даних для спрощення інформаційної роботи, використання фреймворку ReactJS і допоміжних бібліотек, таких як react-router-dom, react- slick, react-helmet тощо), а також

мова гіпертекстової розмітки (HTML) і каскадні таблиці стилів (CSS) (рис. 2.1) [1]

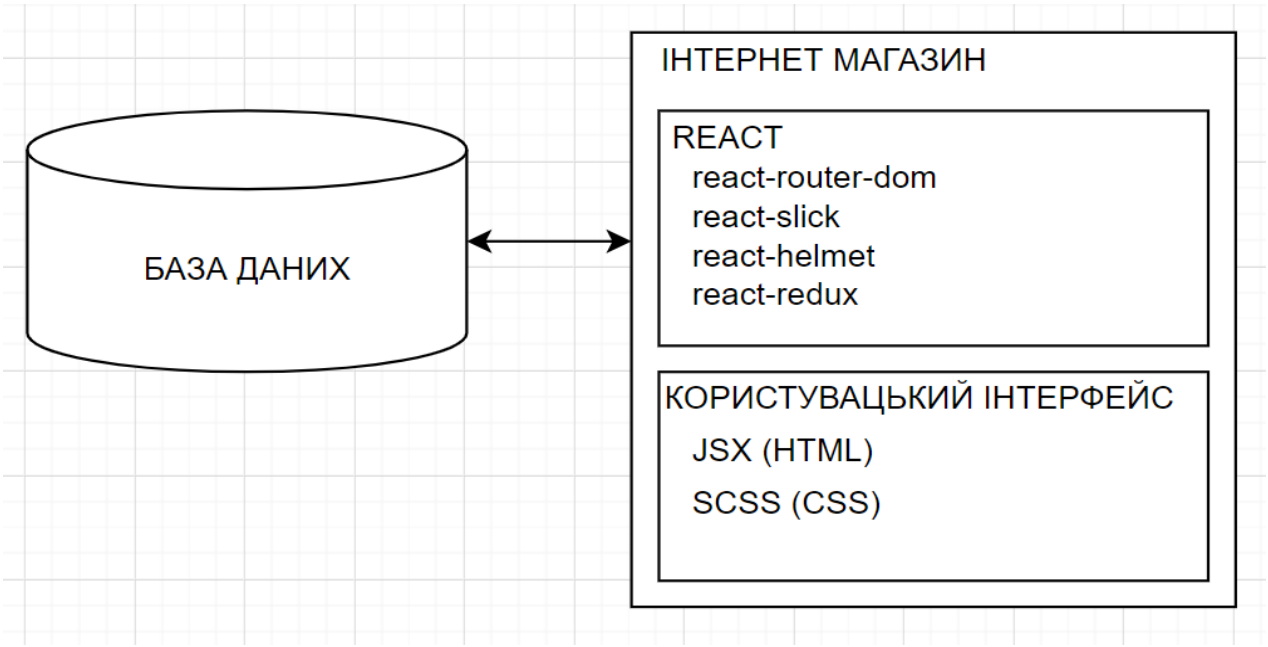

Рисунок 2.1 – Архітектура веб-додатку

При розробці системи, основним критерієм якості є доступність. Тобто структура системи розробки повинна бути добре зрозумілою іншим розробникам, тому що деякі проекти вимагають оновлення та виправлення, а проекти часто створюються та редагуються кількома розробниками.

Один із найкращих способів підійти до великого завдання (розробка інтернет-магазину) — розбити його на менші завдання, які, у свою чергу, розбиваються на менші завдання і так далі, доки завдання не стане достатньо простим. Це єдине правильне рішення для широкого кола завдань із забезпеченням зниженої складності, забезпечення гнучкості системи, надання легких можливостей для розширення та підвищення її стабільності.

2.2 Структура файлів проекту

Пишучи програмне забезпечення, звертайте особливу увагу на файлову структуру, оскільки правильна файлова структура є першою ознакою професіоналізму. Якщо правила формування не дотримуються, то проект

перетворюється на безлад, який важко зрозуміти не тільки іншим розробникам, але й автору програми.

Типовий веб-сайт містить: документи **.HTML, .CSS, .JS** та інші файли, наприклад зображення **(.jpg, .png**) і шрифти **(.woff, .ttf**). Для кожного з них потрібно створити окремі папки та розмістити файли **HTML** у корені папки (сайту). Але веб-додатки на **React** структуровані дещо інакше: кожен об’єкт на веб-сайті є окремим компонентом, імпортованим у головний файл, який уже відображає (рендерить) усі ці помпони на сторінці. Кожен компонент містить два компоненти: файл **JS** і файл **SCSS**. Однак логіка, згідно з якою кожен тип файлу має бути розміщений в окремій папці, залишається. [2]

Після ініціалізації фреймворк самостійно створює класичну структуру, побудовану на основі системи складання проекту Webpack. Ця система передбачає компіляцію будь-якого файлу, який не приймає документи HTML (таких **як SCSS, SASS, LESS FILES** — файли, які спрощують розробку **CSS**), у файли, що підтримують **HTML**.

Кожен веб-сайт починається з **HTML**-файлу, який, у свою чергу, складається з елементів мови розмітки (тегів), кожен з яких має власну роль і використовується в певних ситуаціях. Але при розробці масштабних проектів (наприклад, інтернет-магазинів, соціальних мереж, блогів тощо) тегів багато і схожість блоків висока. Фреймворки JavaScript були створені спеціально для вирішення таких проблем: **AngularJS** від Google, **ReactJS** від Facebook і **VueJS** від розробника Evan Y. Кожен фреймворк має свої особливості, але вони вирішують ту саму проблему. [3]

Розробка фреймворку передбачає створення компонентів, котрі замінять десятки або навіть сотні рядків коду. Наприклад, на нашій сторінці є такий товарний блок (рис. 2.2)

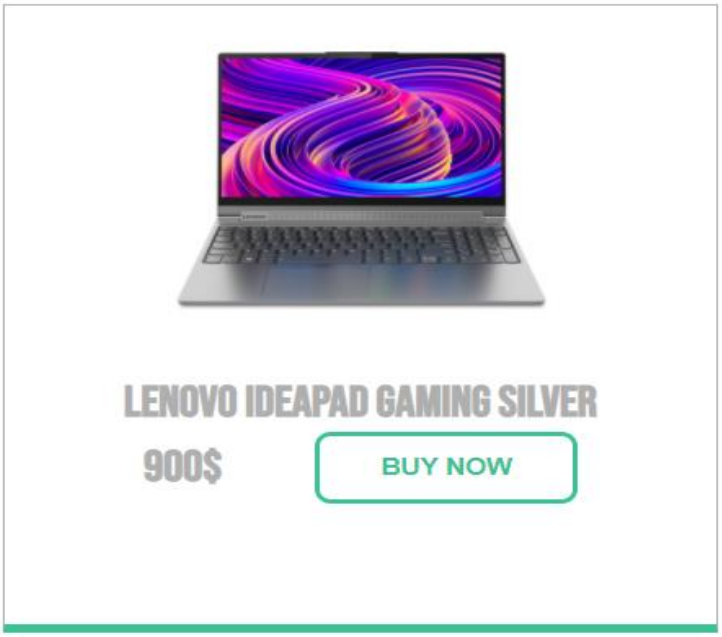

Рисунок 2.2 – Блок з товаром

Так він буде представлений у вигляді коду:

```
<div class = "product_block">
        <img src=”./img/laptop1.png” alt=”product”/>
        <div class = "product_block-content">
            <div>
                <h2> Lenovo Ideapad Gaming Silver </ h2>
            </ div>
            <div className = "product_block-content-button">
                <h2> 900$ </ h2>
                <button class = "button"> BUY NOW </ button>
            </ div>
        </ div>
    </ div>
```

Відображення блоку з товарами зайняло 12 рядків коду, але якщо на сторінці потрібно показати 200 таких товарів? То вже буде складно працювати з таким об’ємом коду. Саме **React** цю проблему може вирішити:

- Прописуємо блок один раз;
- Називаємо потрібним ім’ям;

- Викликаємо його, як свій власний тег (наприклад <*ProductBlock*/>).

Отже, веб-додаток буде складатися із компонентів, які будуть в окремих папках, та певній структурі (рис. 2.3):

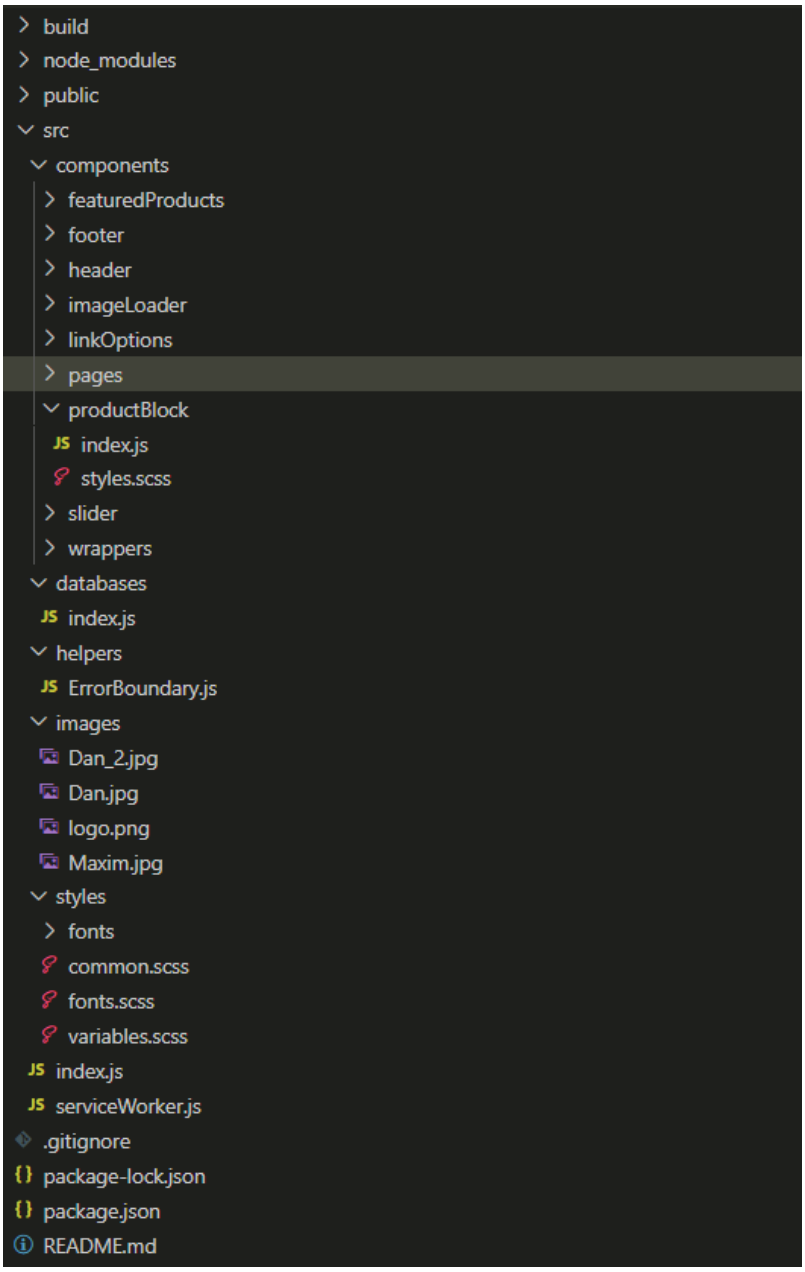

Рисунок 2.3 – Структура веб-додатку.

2.3 Структура регулювання інтернет магазину

На даному етапі слід визначити, ролі виконання за допомогою веб-додатку. Користувачі програми:

• *Відвідувач* – людина, котра зайшла на сторінку. Основною функцією веб-сайту, якою буде він користуватись - огляд товарів інтернет-магазину, після додавання товару в кошик відвідувач стає покупцем;

• *Покупець* – людина, що робить замовлення. Основні функції, які використовуються: додавати дані в кошик, змінювати його вміст, оформляти замовлення, заповнивши відповідну форму;

• *Менеджер (продавець)* – людина, яка допомогає покупцю під час оформлення замовлення. Продавець узгоджує деталі замовлення за телефоном, вказаним клієнтом. Менеджер уточнює деталі доставки і спосіб оплати, коли все вирішено та підтверджено, замовлення прямує до бухгалтера;

• *Бухгалтер* – людина, котра фіксує оплату замовлення, надає підтвердження менеджеру, а вже потім продавець відправляє замовлення клієнту. Бухгалтери також ведуть облік документації підприємства. Графічне зображення схеми. (рис.2.4)

Рисунок 2.4 – Структура регулювання

2.4 Послідовність оформлення замовлення.

На цьому етапі слід зрозуміти послідовність дій кожного з об’єктів (покупця, гаджет, продавець, замовлення), які приймають участь у замовленні гаджету. Отже маємо таку послідовність дій цих об’єктів:

1. Клієнт обирає гаджет;
2. Клієнт подає заявку;
3. Продавець оформлює замовлення;
4. Клієнт узгоджує замовлення з продавцем;
5. Продавець робить заявку на склад для отримання товару;
6. Продавець отримує товар;
7. Клієнт оплачує замовлення;
8. Продавець видає замовлення.

Дана послідовність у виді діаграми (рис. 2.5)

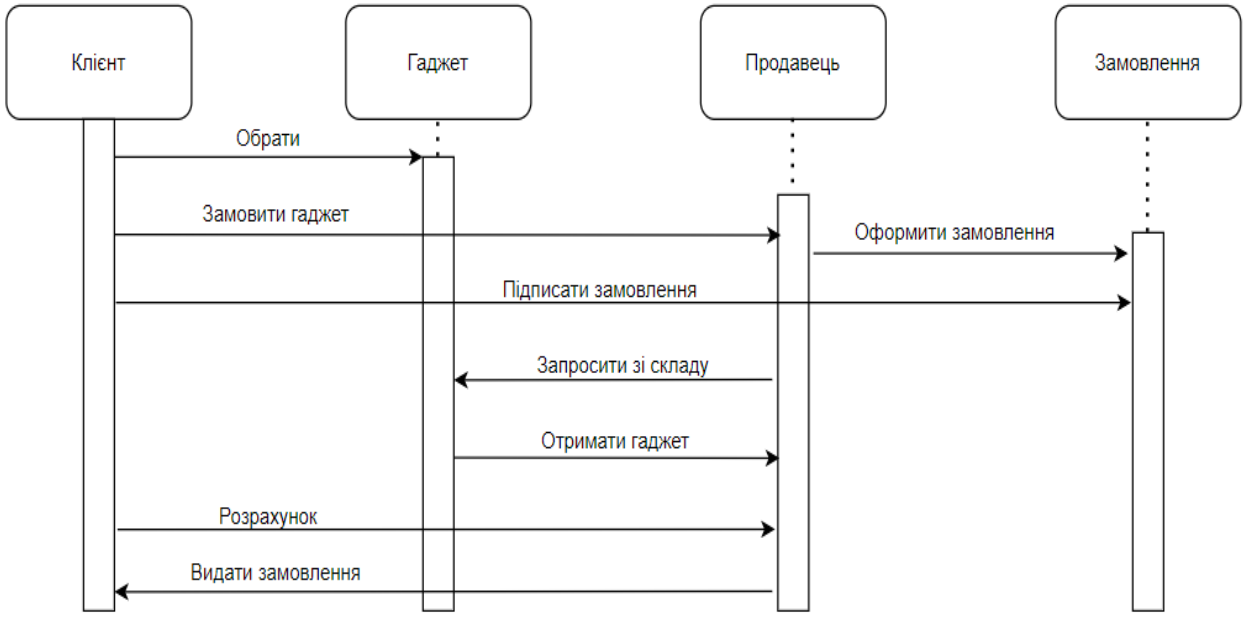

Рисунок 2.5 – Послідовність створення замовлення.

2.5 Структура замовлення

Щоб полегшити коригування замовлення та звітність про замовлення, його слід оформляти правильно. Необхідно визначити основні деталі замовлення, такі як: клієнт, продукт, менеджер і спосіб оплати. У свою чергу, кожен компонент має обов'язковий ідентифікатор, за яким можна швидко ідентифікувати та знайти той чи інший об'єкт у пошуковій базі. Приклад загальної структури замовлення та його компонентів зображено на рис.2.6.

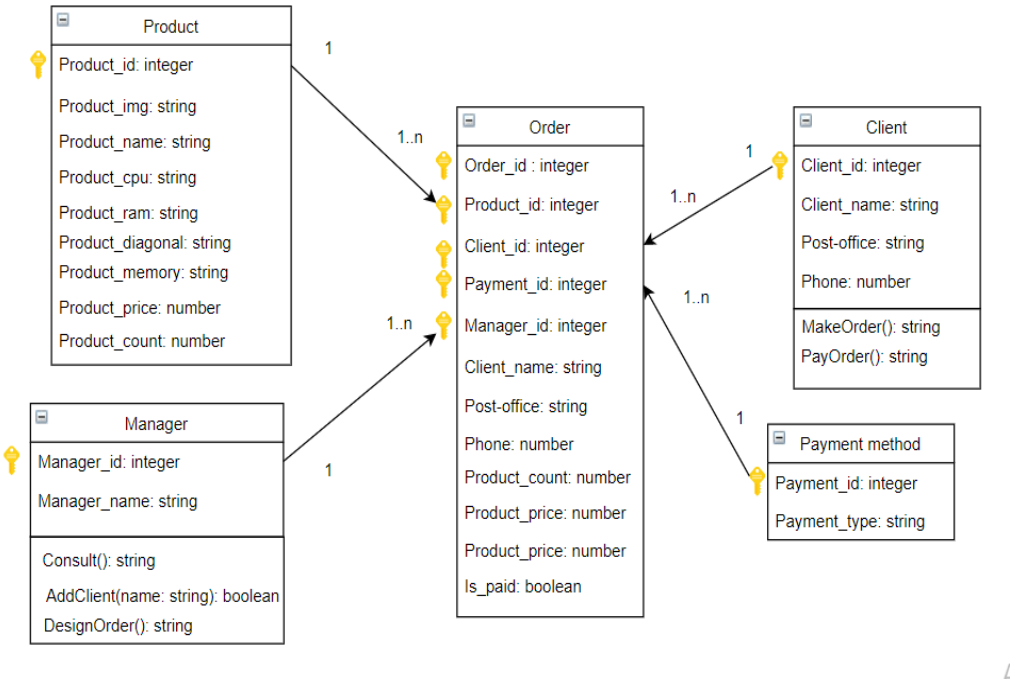

Рисунок 2.6 – Структура замовлення товару

2.6 Схема роботи клієнта з інтернет-магазином

Головною дійовою особою, яка використовує веб-сайт, є покупець. Щоб йому було зручно використовувати функції сайту, необхідно поставити розробника на місце замовника і з'ясувати: яку послідовність дій він буде виконувати, коли буде використовувати магазин в ролі покупця. Це один із найважливіших етапів, тому що від зручності використання залежить, зробить користувач замовлення або ні. На даній схемі показана послідовність дій клієнта з використанням інтернет-магазину. (рис. 2.7). [4]

Рисунок 2.7 – Схема взаємодії клієнта з веб-додатком

2.7 Висновок

Найважливішим та водночас найскладнішим етапом будь-якого проекту є початок. Формування технічного завдання, вибір архітектури додатку мають достатньо великий вплив на розвиток продукту, саме через це слід підходити серйозно з розумінням та баченням кінцевого продукту, ще до початку розробки. Саме після визначення архітектури додатку проект створюється та готується до активної фази розробки, що дає можливість розробити інтерфейс користувача за допомогою засобів середовища розробки Visual Studio Code.

3 РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ ЦИФРОВИХ ГАДЖЕТІВ

3.1 Середовище розробки

Розробка проекту з **React** виконується шляхом написання коду на **JavaScript** і його стилізації за допомогою препроцесора файлів **SCSS JavaScript**, тому що SCSS — це текстовий документ, а середовище розробки — це текстовий редактор. Для розробки продукту можна використовувати навіть програму «Блокнот», але головне у цьому питанні це якість та мобільність, які зможе нам надати редактор **Visual Studio Code**.

Visual Studio Code – це редактор вихідного коду. Він має багатомовний інтерфейс користувача та підтримує ряд мов програмування, підсвічування синтаксису, IntelliSense, рефакторинг, налагодження, навігацію за кодом, підтримку Git та інші можливості.

Visual Studio Code — це редактор вихідного коду, який має багато плагінів для прискореної розробки, багатомовний інтерфейс користувача та підтримує ряд мов програмування. Також має функції для виправлення синтаксичних помилок у коді, навігацію за кодом та підтримку Git. Крім цих функцій, редактор має набір тем, шрифтів, котрі змінюють інтерфейс для кращого та комфортного використання.

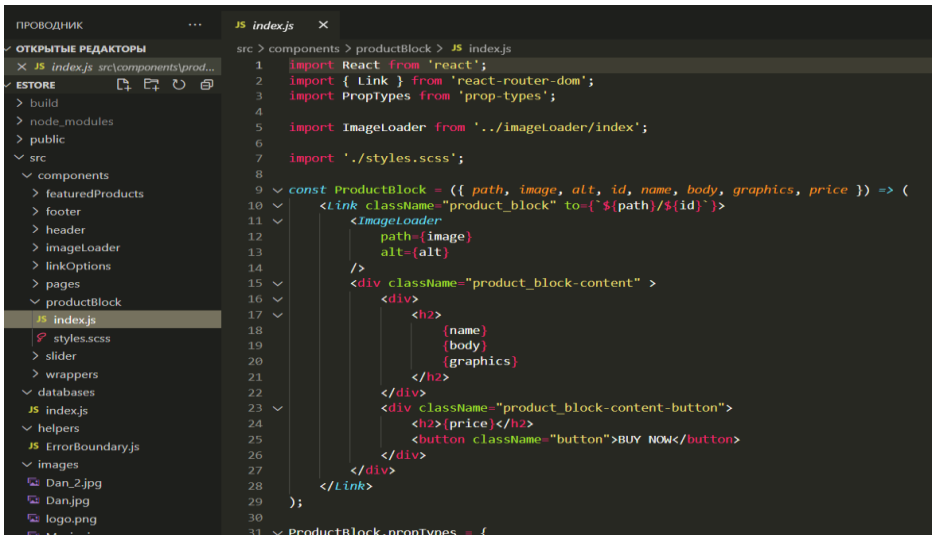

Рисунок 3.1 – Інтерфейс **Visual Studio Code**

3.2 Ініціалізація програми

Як уже згадувалося, додаток **React** побудовано на системі складання **Webpack**, яка налаштовується в консолі шляхом написання команд.

Щоб використовувати **Webpack** на своєму комп'ютері, ви повинні спочатку встановити програмну платформу **Node.js**, яка перетворює **JS** із вузькоспеціалізованої мови на мову загального призначення, і менеджер пакетів **NPM**. Маючи ці компоненти, ви можете почати створювати додаток **React**.

3.3 Розробка інтернет магазину

Спочатку видалимо все непотрібне з папки src, яка була створена автоматично. Далі створюємо глобальні папки зображень, стилів, компонентів, баз даних тощо.

3.3.1 Принцип роботи React

Додаток **React** побудований на компонентах, які імпортуються один в інший до одного батьківського компонента App, який в свою чергу імпортується в головний файл index.js який виконує рендер (прорисовку) усього вмісту App (рис. 3.2)

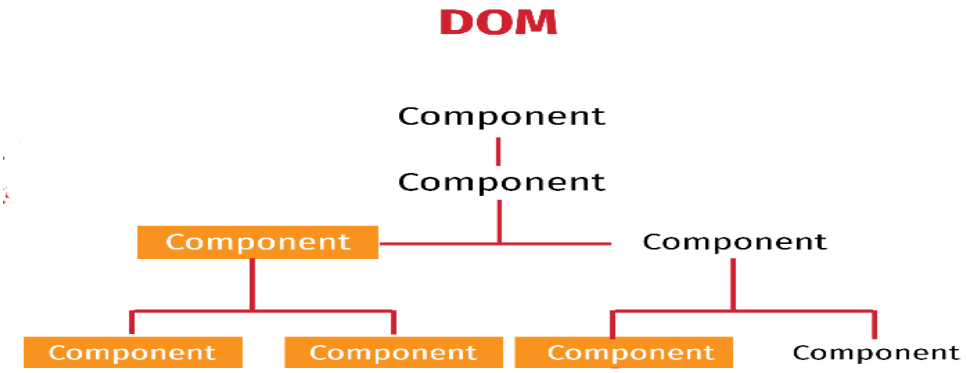

Рисунок 3.2 – Принцип роботи React

Щоб зберегти структуру додатку, у теці "*wrappers*" створюємо головний JS файл із назвою App, куди будуть імпортуватись глобальні файли зі стилями, та у якому будемо обгортати весь проект у компоненти, які зустрічаються на кожній сторінці, наприклад Header. [5]

3.3.2 Створення компонента

Існує два види компонентів: функціональні та класові. У кожного є свої переваги, але Facebook представив функціональні компоненти як стандарт, якому слід слідувати, а деякі розробники проектів навіть переписали свої проекти на функціональні компоненти. На думку автора, кожен тип компонента слід використовувати в залежності від ситуації: тип компонента не відповідає за оптимізацію проекту, необхідно використовувати той, який буде простіше написати з мінімальною кількістю коду. Більшість функціональних компонентів будуть задіяні в проекті, оскільки того вимагає ситуація.

Компонент – це звичайна **JavaScript** функція, яка повертає JSX код. Він в свою чергу це те, що повертає функція у вигляді класів, тегів тощо. Тобто простими словами JSX – це **HTML** код всередині **JavaScript** коду. Для створення компоненту треба:

1. Створити **.js** файл;
2. У даному файлі імпортуємо бібліотеку React командою “import React from “React”;
3. Імпортуємо допоміжні бібліотеки, компоненти та стилі;
4. Створюємо функцію, яка обов’язково повинна починатися з великої літери (в даному випадку ProductBlock);
5. В тілі функції вказуємо «return», відкриваємо круглі скобки та пишемо JSX код, але даний код має бути обгорнутий у один батьківський елемент, якщо функція буде повертати два елемента, то буде помилка;
6. Експортуємо даний компонент командою export default ProductBlock:

В результаті маємо такий код (рис. 3.3):

src > components > productBlock > JS index.js

```
import React from 'react';
import { Link } from 'react-router-dom';
import PropTypes from 'prop-types';

import ImageLoader from '../imageLoader/index';

import './styles.scss';

const ProductBlock = ({ path, image, alt, id, name, body, graphics, price }) => (
    <Link className="product_block" to={`${path}/${id}`}>
        <ImageLoader
            path={image}
            alt={alt}
        />
        <div className="product_block-content" >
            <div>
                <h2>
                    {name}
                    {body}
                    {graphics}
                </h2>
            </div>
            <div className="product_block-content-button">
                <h2>{price}</h2>
                <button className="button">BUY NOW</button>
            </div>
        </div>
    </Link>
);

export default ProductBlock
```

Рисунок 3.3 – Компонент ProductBlock

На даному рисунку (рис. 3.3) Link та ImageLoader – це також компоненти. Link компонент з бібліотеки react-router-dom, який робить навігацію по веб-додатку. ImageLoader – це компонент створений автором який, та залежності від того завантажилась картинка чи ні, відображає спочатку малюнок завантаження, а коли вона завантажилась – сам малюнок.

3.3.3 Props

На рис. 3.3 можна побачити декілька параметрів, які передаються у функцію ProductBlock. Ці параметри у бібліотеці **React** називаються props.

Таких блоків з товаром в інтернет-магазині може бути велика кількість і їх різниця мінімальна: назва товару, ціна товару, зображення та шлях до товару, який передається в компонент Link. Для того, щоб вирішити проблему повторення коду, ми створили props – це звичайний об'єкт JavaScript, який при виклику відображає його залежно від себе. Наприклад: якщо викликати компонент і передати в параметр name значення laptop, то в результаті ми відобразимо компонент з відповідним ім’ям (рис. 3.4).

```
< ProductBlock
    image={'https://www.lenovo.com/medias/
    lenovo-laptop-yoga-c940-hero-14.png?
    context=bWFzdGVyfHJvb3R8MTY4MTQxMHxpbWFnZS9wbmd8aDY0L2gzNy8xMDU0NzY
    4MjI3OTQ1NC5wbmd8OGEzMWFjMzgzMmQxNzY4ZDI1ZDUxZDg1NDIzMWFkY2M1NzhmYW
    NiM2VlMzNlZDViMGNlYjc3NDgzOWQ2OTdiNQ'}
    name="Laptop"
    price="1000$"
/>
```

Рисунок 3.4 – Виклик компонента

У полі image записана url адреса зображення, в name – ім’я гаджета, в price – його. Після збереження коду отримаємо результат на вказаний на рис. 3.5:

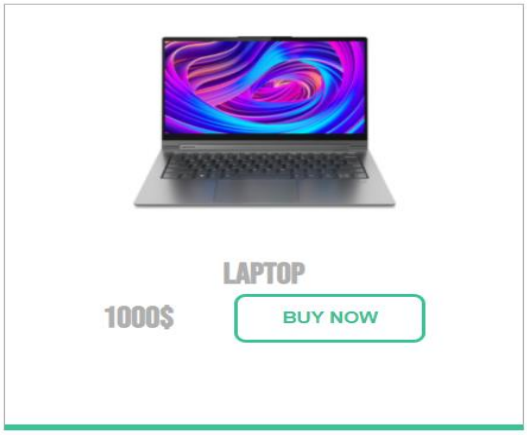

Рисунок 3.5 – Результат виклику компонента

3.3.4 Логіка компонентів

Так як компонент - це не просто код JSX, який, крім відображення HTML, інших функцій не має. Він також може мати власну логіку. Логіка компонента записується в самому компоненті або поза ним, але лише перед викликом "return". (рис. 3.6)

```
const Slider = ({arr}) => {

    const [x, setX] = useState(0);
    const goPrev = () => x === 0 ? setX(-100 * (arr.length - 1)) : setX(x + 100);
    const goNext = () => (x === -100 * (arr.length - 1)) ? setX(0) : setX(x - 100);

    return(
        <section className="slider">
            {
                arr && arr.map((item, key) => {
                    return(
                        <div key={key} className="slide" style={{transform:`translateX(${x}%)`}} to="/sales">
                            <div className="container">
                                <ImageLoader
                                    path={item.img}
```

Рисунок 3.6 – Логіка компонента Slider

3.3.5 База даних

Для оптимізації проекту та для швидкої його розробки створюємо базу даних за допомогою сервісу Microsoft SQL Server та мови SQL. Етапи створення бази даних:

1. CREATE DATABASE GadgetsDB, створюємо базу даних. CREATE DATABASE – два ключові слова для створення бази даних, після цих слів пишеться її назва в один рядок. Так ми створили базу даних з назвою GadgetsDB.
2. CREATE TABLE {Table_name} ({row} NOT NULL {type} (IDENTITY)) створюємо таблицю, в якій {Table_name} – назва таблиці; {row} – назва стовбця; {type} – тип стовбця; **NOTNULL** означає, що значення цього атрибута не може бути пустим при додаванні записів у нашу таблицю. «**IDENTITY**» - при додаванні записів, значення атрибута, зазначеного цим ключовим словом, зазначати не потрібно, оскільки значення будуть додаватися автоматично. Отже, ми створили декілька таблиць з назвами Computers, Laptops, Phones, Sales.
3. ALTER TABLE {Table_name} ADD PRIMARY KEY ({Table_name_id}) створюємо первинні ключі для кожної з таблиць. Первинний ключ – поле у таблиці, яке завжди має бути унікальним, частіше за все це id.
4. ALTER TABLE {Table_name} ADD FOREIGN KEY ({Table_name_id}) REFERENSES {Table_name} ({Table_name_id}) створюємо відношення між таблицями.
5. INSERT INTO {Table_name} ({…rows}) VALUES ({…values}) вносимо дані у нашу таблицю;
6. Браузери підтримують базу даних формату **.json**, тобто стандартне розширення бази даних **.mdf** не буде її відображати у веб-додатку. Для цього експортуємо базу даних у потрібний формат командою:

SELECT {row_name} AS {name} FROM {Table_name} FOR JSON PATH, ROOT ({variable});

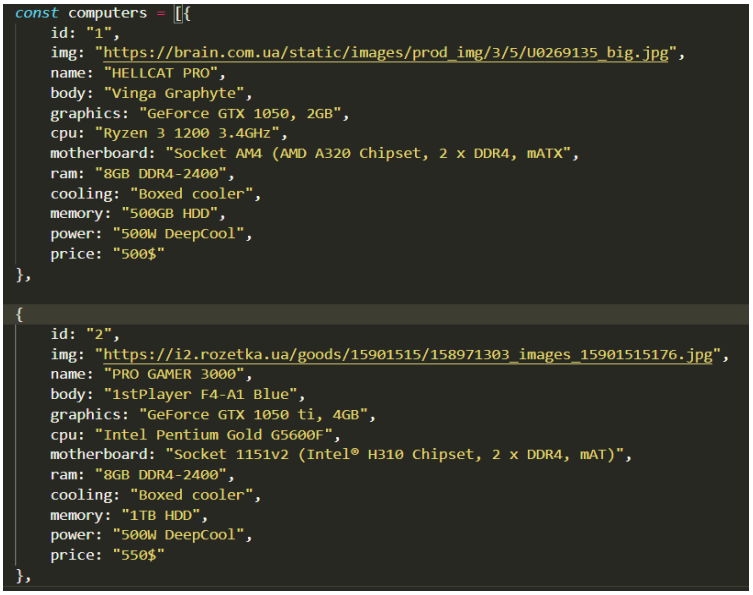

Рисунок 3.7 – База даних JSON

Підключити базу даних на фреймворку набагато легше, ніж у нативному JS. Щоб підключити базу даних в JS, потрібно зробити запити XMLHttp, це займе близько десяти рядків коду. React робить це набагато простіше та швидше з командою: “import data from '../../../databases/index';”, де '../../../databases/index' – це відносний шлях до файла JSON, де і лежить база даних. [7]

3.3.6 Навігація

Навігація веб-додатком у **React** не зовсім звичайний спосіб. Якщо в звичайному **HTML** ми створюємо тег з атрибутом href, значенням якого є відносний шлях до іншого файлу: <a href=”./computers.html”>

То у випадку з **React**, застосовується бібліотека **“react-router-dom”.**

“react-router-dom” – бібліотека **React**, що містить набір необхідних компонентів для навігації. Щоб її створити навігацію, потрібно:

1. Спочатку встановити бібліотеку локально написавши в консолі відповідну команду npm install react-router-dom;

2. Імпортувати дану бібліотеку за допомогою допоміжних компонентів Switch, Router;
3. На найвищому рівні, тобто батьківський компонент обгорнути у компонент даної бібліотеки. Найпоширеніший компонент – це BrowserRouter, завдяки оптимізації і зручності використання;
4. Всередині BrowserRouter вставляємо компонент Switch, дочірними компонентами якого будуть компоненти Route з атрибутами path, component та exact. Path – це url, в залежності від якого буде з'являтися та чи інша сторінка, за яку відповідальний атрибут component.

Пройшовши ці етапи, маємо такий компонент App (рис. 3.8).

```
const App = () => {
  const supportsHistory = 'pushState' in window.history;
  return (
    <BrowserRouter
      forceRefresh={!supportsHistory}
      basename="eStore"
    >
      <ErrorBoundary>
        <Helmet>
          <link rel="icon" href={logo}/>
        </Helmet>
        <Header>
          <Switch>
            <Route exact path="/" component={Home}/>
            <Route exact path="/sales" component={Sales}/>
            <Route exact path="/contactus" component={ContactUs}/>
            <Route exact path="/laptops" component={Laptops}/>
            <Route exact path="/computers" component={Computers}/>
            <Route exact path="/phones" component={Phones}/>
            <Route exact path="/search" component={Search}/>
            <Route exact path="/bag" component={Bag}/>
            <Route exact path="/booking" component={Booking}/>
            <Route path="/computers/:computerid"  component={ProductInfo}/>
            <Route path="/laptops/:laptopid" component={ProductInfo}/>
            <Route path="/phones/:phoneid" component={ProductInfo}/>
            <Route component={NotFound}/>
          </Switch>
        </Header>
      </ErrorBoundary>
    </BrowserRouter>
  )
}

export default App;
```

Рисунок 3.8 – Реалізація навігації додатку

Для того щоб перейти на потрібну сторінку, треба імпортувати такі компоненти з бібліотеки react-router-dom як Link або NavLink – виконують вони одну й ту саму роль: переходять на задану сторінку за допомогою атрибута to.

3.3.7 Відображення товарів на сторінці

У базі даних маємо чотири масиви **JS**: computers, laptops, phones та sales. Також маємо чотири сторінки computers, laptops, phones та sales. Тепер необхідно відобразити список товарів в залежності від обраної користувачем сторінки. Щоб не дублювати власний код (елементів можуть бути десятки, а то й сотні), а зробити це швидко та з найменшою кількістю коду, потрібно використовувати цикли.

На додаток до простих циклів forEach, for і while, з випуском ES7 (версія EcmaScript 7, стандарт для написання коду в **JS**), були додані нові цикли, які можуть не тільки проходити через масив, але й змінювати його або повернути новий залежно від відповідних умов. До таких циклів відносяться: сортування, фільтр, карта.

Сортування та фільтр проходять через масив і повертають потрібний елемент на кожній ітерації, а map повертає змінений масив, який слід використовувати для відображення продуктів на сторінці, оскільки нам потрібно перетворити звичайний об'єкт **JS** у **HTML**-код. (рис. 3.9).

```
import { laptops } from '../../../databases/index';

const Laptops = ({history}) => {
    const path = history.location.pathname;
    return (
        <>
            <Helmet>
                <title>Laptops</title>
            </Helmet>

            <div className="container">
                <div className="wrapper">
                    <div>
                        {
                            laptops.map((item, key) => {
                                return(
                                    < ProductBlock
                                    path={path}
                                    key={key}
                                    image={item.img}
                                    id={item.id}
                                    name={item.name}
                                    price={item.price}
                                    />
                                )
                            })
                        }
                    </div>
                </div>
            </div>
        </>
    )
}
export default Laptops
```

Рисунок 3.9 – Код відображення товарів на сторінці

3.3.8 Реалізація функціоналу корзини

Основне завдання інтернет-магазину - надіслати продавцю інформацію про замовлення, а саме: ім'я, прізвище, номер телефону, адресу, а головне - список замовлених товарів. Для реалізації цього функціоналу нам потрібне сховище, яке, незалежно від часу, відвіданої сторінки чи гаджета, запам’ятає таку інформацію. Існує три таких сховища: **LocalStorage, SessionStorage та Cookies**.

- **SessionStorage** – сховище, яке запам’ятовує набір інформації в межах сесії, тобто якщо закрити вкладку сторінки, то дані зітруться;
- **Cookie** – запам’ятовує дані на протягом потрібного часу, який встановлюється розробником;
- **LocalStorage** – сховище, яке запам’ятовує інформацію, поки користувач не зітре їх через консоль розробника або за допомогою функціонала розробленим на сайті.

Найліпшим варіантом для вирішення задачі є LocalStorage. Синтаксис LocalStorage це ключ – значення, тобто для кожного ключа є своє значення, наприклад **“time”: “12AM”**, де **“time”** це ключ, **“12AM”** – значення. LocalStorage - звичайний JavaScript об’єкт, який має методи, серед яких у додатку буде використано get та set.

Синтаксис get об’єкту LocalStorage: window.localStorage.get(‘key’); де key – ім’я ключа. Синтаксис set: window.localStorage.set(‘key’, ‘value’); де value – значення.

Отже, залишається обробити подію (рис. 3.10), що запустить відповідну функцію (рис. 3.11), котра додасть до сховище новий об’єкт – а саме, товар, що збирається придбати клієнт.

```
<div className="button_wrapper">
    <h3>{machedObj.lastPrice && machedObj.lastPrice}</h3>
    <button onClick={buyGadget}>{machedObj.price}</button>
</div>
```

Рисунок 3.10 – Обробник події “click”

```
const buyGadget = () => {
    modal.current.style.display = 'flex';
    setTimeout(() => {
        modal.current.classList.add('shown');
    },100);
    const buys = JSON.parse(window.localStorage.getItem('buys')) || [];
    const buy = {
        id: machedObj.id,
        img: machedObj.img,
        name: machedObj.name,
        price: machedObj.price,
        count: 1,
        type: type
    }
    const index = buys.findIndex(item => item.id === machedObj.id && item.name === machedObj.name);
    if (index > -1) buys[index].count++;
    else buys.unshift(buy);
    window.localStorage.setItem('buys', JSON.stringify(buys));
}
```

Рисунок 3.11 – Додавання обраного товару у сховище

Тепер, переходячи на сторінку для покупок, потрібно використовувати метод get, щоб відсортувати масив вибраних продуктів і відобразити їх. На останньому етапі ми розробляємо форму, в якій користувач залишить необхідні дані для оформлення замовлення, які потім будуть відправлені на пошту менеджера.

3.4 Висновок

У цьому розділі були повністю розглянуті засоби реалізації розробленого інтерфейсу та логіка програми. Також реалізовані головні можливості фреймворку **React**, а саме: ініціалізація проекту **React** та створення коректної структури; принципи роботи цієї рамки; створення компонента та його логіка; відображення компонентів за допомогою циклів; створення навігації, яка допомагає переходу між сторінками додатку. Завдяки базі даних створено інтернет-магазин цифрових гаджетів з великим асортиментом, можливістю переходу на потрібний товар для його аналізу та додавання в кошик. Також для сайту була розроблена система пошуку товару, що дозволяє обрати смартфон не тільки за назвою, але і за моделлю. Розроблено функціонал кошика, із відображається списку обраних товарів і кнопка «Купити», при натисканні користувач має можливість перейти на сторінку з потрібною формою, для оформлення замовлення

‘

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було створено інтернет-магазин eShop на базі комплексу методів та засобів сучасних інформаційних технологій. Асортимент товарів даного магазину також був підібраний під задану цільову аудиторію. При розробці eShop врахована архітектура програмного забезпечення, а саме: файлова структура, архітектура взаємодії користувача з ПЗ, послідовність розміщення замовлення та взаємодія клієнта та інтернет-магазином. Завдяки вибору фреймворку React був розроблений повноцінний інтернет-магазин цифрових гаджетів, який виділяється серед конкурентів своєю адаптивністю, оптимізацією та відсутністю реклами.

Були задіяні дані технології:

- **PhotoShop** – редактор для створення UX / UI веб-дизайну;

- Star UML – програма для розробки діаграм, з її допомогою створено візуальну структуру проекту;
- **React** – бібліотека **JavasScript**;
- **Webpack** – система зборки проекту;
- **SCSS – CSS** препроцесор, для комфортної та швидкої розробки;

Задіяні такі бібліотеки:

- **React-helmet** – бібліотека, яка дозволяє додавати імена та іконки для кожної із сторінок;
- **React-router-dom** – для навігації;
- **React-slick** – для реалізації спайдерів;
- **Prop-types** – для типізації компонентів, щоб уникнути помилок;

У процесі розробки інтернет-магазину були викладені та показані на практиці основні функції фреймворку React JS, що може дати впевнений старт для початку розробки веб-додатків починаючим програмістам.

СПИСОК ЛІТЕРАТУРИ

1. Fuller Matt, Moser Manfred, Traverso Martin. Trino: The Definitive Guide: SQL at Any Scale, on Any Storage, in Any Environment. 2nd Edition (Final). – O'Reilly Media, Inc., 2022. – 319 p.
2. Kline K., Obe R.O., Hsu L.S. SQL in a Nutshell. 4th Edition. – O'Reilly Media, 2022.
3. Rockoff Larry. The Language of SQL. 3rd Edition (Final). – Pearson Education, 2022. – 272 p.
4. Petkovic Dusan. Microsoft SQL Server 2019: A Beginner's Guide . 7th ed. – McGraw-Hill, 2020. – 825 p.
5. Muzny V. et al. SQL Server 2019 Administrator's Guide. – Packt, 2020. – 522 p.
6. Botros S. et al. High Performance MySQL: Proven Strategies for Operating at Scale . 4th Edition. – O'Reilly Media, 2022. – 388 p.
7. Bin Uzayr Sufyan (Ed.) Mastering MySQL for Web: A Beginner's Guide. – CRC Press, 2022. – 309 p.
8. Unhelkar B. Software Engineering with UML . – Auerbach Publications, 2018. – 427 p.
9. Donahoe L., Hartl M. Learn Enough Html, Css and Layout to Be Dangerous: An Introduction to Modern Website Creation and Templating Systems. – Pearson, 2022. – 672 p
10. McGregor M. JavaScript: The Hidden Parts: Building More Performant, Flexible, and Maintainable Applicationsю.– O'Reilly Media, 2022. – 142 p.
11. Scott A.D., MacDonald M., Powers S. JavaScript Cookbook: Programming the Web . 3rd edition. – O'Reilly Media, Inc., 2021. – 538
12. Shute Z. Advanced JavaScript.– Packt Publishing, 2019. – 330 p.
13. Meyer J., Nunney M. The Guide Of HTML5 and JAVA Script .– Amazon Digital Services LLC, 2019. – 394 p

14. Photoshop [Електронний ресурс] Режим доступу: https://www.adobe.com/ua/products/photoshop.html;
15. Webpack [Електронний ресурс] Режим доступу: https://webpack.js.org;
16. ReactJS[Електронний ресурс] Режим доступу: https://learn-reactjs.com/home
17. JavaScript[Електронний ресурс] Режим доступу: https://learn.javascript.com/

ДОДАТОК

index.html

```
<!DOCTYPE html>
<html lang="en">

  <head>

    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app" />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <script
src="https://bot.jaicp.com/chatwidget/ImQdWeOW:59a46d1e50e20992774c9070600c946990
a79686/justwidget.js?force=true" async></script>
    <!--This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

index.js (database)

```
export const computers = [{

        id: "1",
        img: "https://brain.com.ua/static/images/prod_img/3/5/U0269135_big.jpg",
        name: "HELLCAT PRO",
        body: "Vinga Graphyte",
        graphics: "GeForce GTX 1050, 2GB",
        cpu: "Ryzen 3 1200 3.4GHz",
        motherboard: "Socket AM4 (AMD A320 Chipset, 2 x DDR4, mATX",
        ram: "8GB DDR4-2400",
        cooling: "Boxed cooler",
        memory: "500GB HDD",
        power: "500W DeepCool",
        price: "20 500 грн"
    },

    {
        id: "2",
        img: "https://digitalfury.pro/media/component107917.png",
        name: "PRO GAMER 3000",
        body: "1stPlayer F4-A1 Blue",
        graphics: "GeForce GTX 1050 ti, 4GB",
        cpu: "Intel Pentium Gold G5600F",
        motherboard: "Socket 1151v2 (Intel® H310 Chipset, 2 x DDR4, mAT)",
        ram: "8GB DDR4-2400",
        cooling: "Boxed cooler",
        memory: "1TB HDD",
        power: "500W DeepCool",
        price: "22 550 грн"
    },

    {
        id: "3",
        img:
"https://content1.rozetka.com.ua/goods/images/original/60118139.jpg",
        name: "WARRIOR F1",
        body: "Frontier Jumbo",
        graphics: "GeForce GTX 1060 - 3GB",
        cpu: "Core i3-9100F",
        motherboard: "Socket 1151v2 (Intel® H310 Chipset, 2 x DDR4, mAT)",
        ram: "8GB DDR4-2400",
        cooling: "Boxed cooler",
        memory: "500GB HDD",
        ssd: "120 GB Patriot",
        power: " 550W 1stPlayer",
        price: "24 600 грн"
    },

    {
```

```
        id: "4",
        img: "https://digitalfury.pro/media/component109537.png",
        name: "CYCLONE F23",
        body: "1stPlayer Fire Dancing",
        graphics: "GeForce GTX 1650, 4GB",
        cpu: "Ryzen™ 5 1400 3.4GHz Turbo",
        motherboard: "Socket AM4 (AMD B450 Chipset, 2 x DDR4, mATX)",
        ram: "8GB DDR4-2400",
        cooling: "Boxed cooler",
        memory: "500GB HDD",
        ssd: "120 GB Patriot",
        power: "500W Frontier",
        price: "26 650 грн"
    },

    {
        id: "5",
        img: "https://hotline.ua/img/tx/163/1633659935.jpg",
        name: "GRIFFIN",
        body: "Vinga Graphyte",
        graphics: "GeForce GTX 1650 Super, 4GB",
        cpu: "Ryzen 3 1200 3.4GHz",
        motherboard: "Socket s1200 (Intel® B410 Chipset, 2 x DDR4, mATX)",
        ram: "8 GB [8 GB x1] DDR4-2666",
        cooling: "Boxed cooler",
        memory: "1TB HDD",
        ssd: "480 GB Kingston",
        power: "500W DeepCool",
        price: "28 700 грн"
    },

    {
        id: "6",
        img:
"https://i8.rozetka.ua/goods/15643715/158970981_images_15643715802.jpg",
        name: "AVENTUM QX",
        body: "Frontier Warrior",
        graphics: "GeForce GTX 1660TI, 6GB",
        cpu: "Intel Core i5-9400F",
        motherboard: "Socket AM4 (AMD A320 Chipset, 2 x DDR4, mATX",
        ram: "16 GB [8 GB x2] DDR4-2666",
        cooling: "Boxed cooler",
        memory: "1TB HDD",
        ssd: "120 GB Patriot",
        power: "550W 1stPlayer",
        price: "30 750 грн"
    },
```

index.js (слайдер)

```
import React, {useState} from 'react';
import PropTypes from 'prop-types';
import { Link } from 'react-router-dom';

import ImageLoader from '../imageLoader/index';

import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faArrowAltCircleRight, faArrowAltCircleLeft } from '@fortawesome/free-
solid-svg-icons';
import variant1 from '../../images/slider-variants-1.png';
import variant2 from '../../images/slider-variants-2.png';
import variant3 from '../../images/slider-variants-3.png';

import './styles.scss';

const Slider = ({arr}) => {

    const [x, setX] = useState(0);
    const goPrev = () => x === 0 ? setX(-100 * (arr.length - 1)) : setX(x + 100);
    const goNext = () => (x === -100 * (arr.length - 1)) ? setX(0) : setX(x -
100);

    return(
        <div className="slider__wrapper">
            <div className="container">
                <section className="slider">
                {
                    arr && arr.map((item, key) => {
                        return(
                            <div key={key} className="slide"
style={{transform:`translateX(${x}%)`}} to="/sales">
                                <div className="container">
                                    <ImageLoader
                                        path={item.img}
                                        alt='gatget'
                                        loaderGif="https://www.houseofdestiny.org
/wp-content/uploads/2020/01/ring-loader.gif"
                                    />
                                    <div className="slider-content">
                                        <h1>{item.name}</h1>
                                        <h3>{item.lastPrice}</h3>
                                        <h2>{item.price}</h2>
                                        <button to="/sales" className="slider-
content-btn"><Link to="/sales">Переглянути товар</Link></button>
                                    </div>
                                </div>
                            </div>
```

```
                        )
                    })
                }

                <button className="slider_button prev"
onClick={goPrev}><FontAwesomeIcon icon={faArrowAltCircleLeft} /></button>
                <button className="slider_button next"
onClick={goNext}><FontAwesomeIcon icon={faArrowAltCircleRight} /></button>
                </section>
                <div className="slider__variants">
                    <Link to="/phones" className="slider__variants__item">
                        <p>Новий <span>Iphone 14</span> вже в продажу</p>
                        <img src={variant1} alt="телефони"/>
                    </Link>
                    <Link to="/computers" className="slider__variants__item">
                        <p><span>Нові моделі</span> геймінгових пк </p>
                        <img src={variant2} alt="комп'ютери"/>
                    </Link>
                    <Link to="/laptops" className="slider__variants__item">
                        <p>знижка на ноути до першого вересня <span>-
10%</span></p>
                        <img src={variant3} alt="телефони"/>
                    </Link>
                </div>
            </div>
        </div>
    )
}
Slider.propTypes = {
    imgArr: PropTypes.array.isRequired
}
Slider.defaultProps = {
    imgArr: []
}
export default Slider
```

serciceWorker.js

```
// This optional code is used to register a service worker.

// register() is not called by default.

// This lets the app load faster on subsequent visits in production, and gives
// it offline capabilities. However, it also means that developers (and users)
// will only see deployed updates on subsequent visits to a page, after all the
// existing tabs open on the page have been closed, since previously cached
// resources are updated in the background.
```

```
// To learn more about the benefits of this model and instructions on how to
// opt-in, read https://bit.ly/CRA-PWA

const isLocalhost = Boolean(
  window.location.hostname === 'localhost' ||
    // [::1] is the IPv6 localhost address.
    window.location.hostname === '[::1]' ||
    // 127.0.0.0/8 are considered localhost for IPv4.
    window.location.hostname.match(
      /^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}$/
    )
);

export function register(config) {
  if (process.env.NODE_ENV === 'production' && 'serviceWorker' in navigator) {
    // The URL constructor is available in all browsers that support SW.
    const publicUrl = new URL(process.env.PUBLIC_URL, window.location.href);
    if (publicUrl.origin !== window.location.origin) {
      // Our service worker won't work if PUBLIC_URL is on a different origin
      // from what our page is served on. This might happen if a CDN is used to
      // serve assets; see https://github.com/facebook/create-react-
app/issues/2374
      return;
    }

    window.addEventListener('load', () => {
      const swUrl = `${process.env.PUBLIC_URL}/service-worker.js`;

      if (isLocalhost) {
        // This is running on localhost. Let's check if a service worker still
exists or not.
        checkValidServiceWorker(swUrl, config);

        // Add some additional logging to localhost, pointing developers to the
        // service worker/PWA documentation.
        navigator.serviceWorker.ready.then(() => {
          console.log(
            'This web app is being served cache-first by a service ' +
              'worker. To learn more, visit https://bit.ly/CRA-PWA'
          );
        });
      } else {
        // Is not localhost. Just register service worker
        registerValidSW(swUrl, config);
      }
    });
  }
}
```

```
function registerValidSW(swUrl, config) {
  navigator.serviceWorker
    .register(swUrl)
    .then(registration => {
      registration.onupdatefound = () => {
        const installingWorker = registration.installing;
        if (installingWorker == null) {
          return;
        }
        installingWorker.onstatechange = () => {
          if (installingWorker.state === 'installed') {
            if (navigator.serviceWorker.controller) {
              // At this point, the updated precached content has been fetched,
              // but the previous service worker will still serve the older
              // content until all client tabs are closed.
              console.log(
                'New content is available and will be used when all ' +
                  'tabs for this page are closed. See https://bit.ly/CRA-PWA.'
              );

              // Execute callback
              if (config && config.onUpdate) {
                config.onUpdate(registration);
              }
            } else {
              // At this point, everything has been precached.
              // It's the perfect time to display a
              // "Content is cached for offline use." message.
              console.log('Content is cached for offline use.');

              // Execute callback
              if (config && config.onSuccess) {
                config.onSuccess(registration);
              }
            }
          }
        };
      };
    })
    .catch(error => {
      console.error('Error during service worker registration:', error);
    });
}

function checkValidServiceWorker(swUrl, config) {
  // Check if the service worker can be found. If it can't reload the page.
  fetch(swUrl, {
    headers: { 'Service-Worker': 'script' },
  })
    .then(response => {
```

```
      // Ensure service worker exists, and that we really are getting a JS file.
      const contentType = response.headers.get('content-type');
      if (
        response.status === 404 ||
        (contentType != null && contentType.indexOf('javascript') === -1)
      ) {
        // No service worker found. Probably a different app. Reload the page.
        navigator.serviceWorker.ready.then(registration => {
          registration.unregister().then(() => {
            window.location.reload();
          });
        });
      } else {
        // Service worker found. Proceed as normal.
        registerValidSW(swUrl, config);
      }
    })
    .catch(() => {
      console.log(
        'No internet connection found. App is running in offline mode.'
      );
    });
}

export function unregister() {
  if ('serviceWorker' in navigator) {
    navigator.serviceWorker.ready
      .then(registration => {
        registration.unregister();
      })
      .catch(error => {
        console.error(error.message);
      });
  }
}
```

Common.scss

```
@import './fonts.scss';

@import './variables.scss';

body {
    margin: 0;
    padding: 0;
    font-family: 'Roboto Flex', sans-serif;
    scroll-behavior: smooth;
    background-color: #F4F6FA;
}
```

```scss
* {
    // font-family: 'Open Sans', sans-serif;
    box-sizing: border-box;
    margin: 0;
    padding: 0;
    outline: none;
}
h1{
    color: rgb(70, 70, 70);
}
ul{
    list-style: none;
}
a{
    color: #fff;
    text-decoration: none;
    &:hover{
        transition: all .15s linear;
        filter: brightness(0.95);
    }
}
button{
    outline: none;
}
.active{
    color: #fff94f;
    svg{
        color: #f8f368;
    }
}
.container{
    width: 100%;
    max-width: 1350px;
    margin: 0 auto;
}
a{
    transition: .15s;
}
.wrapper{
    display: flex;
    align-items: center;
    justify-content: center;
    flex-wrap: wrap;
    padding: 50px 0;
    &>div{
        display: flex;
        align-items: center;
        justify-content: center;
        flex-wrap: wrap;
    }
```

```
}

.heading{
    text-align: center;
    h1{
        font-size: 50px;
        font-weight: normal;
        color: $grey;
    }
}
*::-webkit-scrollbar-thumb{
    background: #7F7E89;
    border-radius: 4px;
    width: 8px;
}

*::-webkit-scrollbar{
    width: 8px;
}

.button {
    display: inline-block;
    margin-top: 16px;
    outline: none;
    cursor: pointer;
    letter-spacing: 3px;
    background: #FFCB05;
    border: 1px solid #FFCB05;
    border-radius: 16px;
    padding: 12px 28px;
    font-weight: 700;
    font-size: 18px;
    line-height: 28px;
    text-transform: uppercase;
    transition: 0.15s;
    color: #1A141F;
}
.button:hover {
    background: rgba(0, 0, 0, 0);
    box-shadow: inset 0 0 0 3px #ffcb05;
}
.button:hover:before {
    left: 162px;
    transition: .5s ease-in-out;
}
.justwidget--asst-pic .juswidget-fill {
    fill: #FFCB05 !important;
}
.justwidget .justwidget--headline{
    background-color: #FFCB05 !important;
```

```
}
.justwidget--powered_label{
    display: none !important;
}
.justwidget--actions textarea{
    min-height: 70px !important;
}

.justwidget .justwidget--buttons-inline{
    justify-content: start !important;
    margin-bottom: -30px !important;
}
```

fonts.scss

```
@import
url('https://fonts.googleapis.com/css2?family=Roboto+Flex:opsz,wght@8..144,300;8.
.144,400;8..144,500;8..144,600;8..144,700&display=swap');


@font-face {
    font-family: "Bebas Neue";
    src: url("./fonts/BebasNeue-Regular.ttf");
    font-style: normal;
    font-weight: 400;
}
@font-face {
    font-family: "Open Sans";
    src: url("./fonts/OpenSans-Light.ttf");
    font-style: normal;
    font-weight: 300;
}
@font-face {
    font-family: "Open Sans";
    src: url("./fonts/OpenSans-Regular.ttf");
    font-style: normal;
    font-weight: 400;
}
@font-face {
    font-family: "Open Sans";
    src: url("./fonts/OpenSans-Bold.ttf");
    font-style: normal;
    font-weight: 800;
}
@font-face {
    font-family: "Lobster";
    src: url("./fonts/Lobster-Regular.ttf");
    font-style: normal;
    font-weight: 400;
}
```