Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Ігнатенка Сергія Івановича*
(ПІБ)

академічної групи *122-18-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми Комп'ютерні науки
(назва освітньої програми)

на тему: *Розробка веб-магазину з продажу комп'ютерної периферії за допомогою мови JavaScript*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц.Приходченко С.Д.*			
розділів:				
спеціальний	*доц. Приходченко С.Д.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-18-2* *Ігнатенка С.І.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-магазину з продажу комп'ютерної периферії за допомогою мови JavaScript*

затверджена наказом ректора НТУ «ДП» від *08.06.2022 р.* №

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *доц. Приходченко С.Д.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Ігнатенко С.І.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 72 с., 13 рис., 3 дод., 22 джерела.

Об'єкт розробки: сайт з продажу комплектуючих для комп’ютерів.

Мета кваліфікаційної роботи: створення сайту з продажу комплектуючих для комп’ютерів, використовуючи мову програмування JavaScript і ReactJS.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточняється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні мобільного додатка, що дозволяє вимірювати з точністю реальні об’єкти, зберігати створені заміри.

Актуальність даного програмного продукту визначається великим попитом на подібні розробки, що оптимізують та спрощують дії щодо вимірювання у компаніях для маркетингового відділу або в особистих цілях окремого користувача.

Список ключових слів: САЙТ, JavaScript, ReactJS, КОМП’ЮТЕР, ІНТЕРФЕЙС, АЛГОРИТМ, ПРОЕКТУВАННЯ, НАВІГАЦІЯ, WEB, РОЗРОБКА, БАЗА ДАНИХ.

ABSTRACT

Explanatory note: 72 pp., 13 fig, 3 appendix, 22 sources.

Object of development: site for the sale of computer components.

The purpose of the qualification work: to create a site for the sale of computer components using the JavaScript and ReactJS programming languages.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic and clarifies the task.

In the first section the subject branch is analyzed, the urgency of the task and the purpose of development are determined, the statement of the task is formulated, the requirements to the software implementation, technologies and software are specified.

The second section analyzes the available solutions, selected platforms for development, designed and developed the program, describes the program, algorithm and structure of its operation, as well as calling and loading the program, determines the input and output data, describes the parameters of hardware.

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical value is to create a mobile application that allows you to accurately measure real objects, save the created measurements.

The relevance of this software product is determined by the high demand for such developments that optimize and simplify measurement activities in companies for the marketing department or for personal purposes of an individual user.

List of keywords: SITE, JavaScript, ReactJS, COMPUTER, INTERFACE, ALGORITHM, DESIGN, NAVIGATION, WEB, DEVELOPMENT, DATABASE.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

SDK – Software Development Kit;

ЕОМ – електронно-обчислювальна машина;

ПЗ – програмне забезпечення;

ПК – портативний комп’ютер.

ВСТУП

Темою даної кваліфікаційної роботи є створення сайту з продажу комплектуючих для комп'ютерів, використовуючи мову програмування JavaScript і ReactJS.

Метою даної кваліфікаційної роботи є вивчення інструментальних засобів прграмми WebStorm, вивчення і визначення різних методів створення WEB-сторінок, зокрема за допомогою ReactJS.

Ідея створення першого інтернет-магазину належить ДжеффуБізосу, який у 1994 році припустив, що люди, які спілкуються за допомогою інтернету, також захочуть замовляти товари та послуги онлайн. Розмірковуючи над тим, які товари найзручніше купувати в інтернеті, тобто не псуються і не тендітні, він почав продаж книг, аудіо- і відеокасет і дисків. Так у 1995 році відкрився перший інтернет-магазин Amazon, який досі є одним із найбільших та найпопулярніших у світі.

Зараз же в інтернет-магазині можна купити практично все: техніку, авіаквитки, продукти, одяг. взуття; 28% - парфумерію та косметику; 25% - побутову техніку та електроніку; 24% найчастіше замовляють в інтернеті їжу з кафе та ресторанів, а 20% купують товари для дітей. Цікаво, що до рейтингу не увійшли книги, з яких розпочинався розвиток онлайн-торгівлі.

Найпопулярніші товари – це одяг та взуття, а отже, найбільші та найуспішніші магазини займаються продажем саме цієї категорії продукції.

Велику роль для розвитку інтернет-магазинів відіграла поява банківських карток, за допомогою яких можна робити покупки дистанційно. Якщо раніше багато хто писав номер своєї карти в інтернеті з обережністю, то зараз все більше людей довіряє цьому способу оплати і не сумнівається в його безпеці.

Але, мабуть, найважливіша причина популярності магазинів у мережі – це економія часу, яку вони нам забезпечують. Здійснюючи покупки в інтернеті, не потрібно нікуди їхати, стояти у черзі чи пробці. Весь час, який ви витрачаєте на покупку, - це вибір потрібного вам товару в магазині. Не дивно, що онлайн-

шопінг дуже зручний для ділових людей, тих, хто не має можливості їздити в магазин, і просто для тих, хто цінує свій час..

Саме тому, на сьогодні, інтернет-магазини – невід’ємна частина будь-якої компанії, їх створення-необхідний крок для розвитку бізнесу

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Інтернет-магазин – це форма електронної торгівлі, яка дозволяє споживачам купувати товари або послуги за допомогою веб-браузера. Споживачі знаходять товар, що цікавить, відвідуючи сайт роздрібного продавця безпосередньо або шляхом пошуку серед альтернативних постачальників. Торгівля може відбуватися через невеликий локальний магазин, великого роздрібного продавця, магазин електронної комерції або приватну особу, яка продає товари через сторонній сервіс. Інтернет магазин має досить складну структуру, яка включає як основне меню, так і елементи підменю. В основному меню має відображатися найважливіша інформація про категорії товарів та акції. Один з кращих способів дізнатися, які сторінки включити в панель навігації, а які пропустити або використовувати як частину списку, - переглянути дані потоку користувачів в Google Analytics.

Приклад вдало створеної структури можна простежити на веб-сайтах багатьох спортивних брендів. На кожному сайті відображається дуже проста навігація, незважаючи на те, що сайт пропонує дуже багато різноманітних товарів. Ця простота має сенс. Коли ви хочете придбати шорти, ви не хотіли б почати з категорії шортів, тому що там будуть показані товари як для чоловіків, так і для жінок. Замість цього, коли ви потрапляєте на сайт світових спортивних брендів, ви можете почати з прокручування мишею за необхідною основною категорією, а потім варіанти стають все більш і більш конкретними. І тут основна категорія — це пів-клієнта.

Структура інтернет-магазину заснована на моделі "від простого до складного", де кожна категорія містить ще більшу кількість категорій. До структури хорошого сервісу входить зручний пошук за категоріями, який має відображатися як на головній сторінці, так і підкатегоріях.

Процес запуску інтернет-магазину включає просування, кваліфікацію товару, підвищення зацікавленості клієнтів, налаштування обробки замовлень.

Аби відкрити інтернет-магазин, необхідно зробити такі кроки:

При виборі предмета торгівлі важливо оцінити місцевий та глобальний попит. Покупці завжди воліють купувати товари у місцевого продавця, навіть якщо це інтернет-магазин.

Виберіть бізнес-модель

Як власник інтернет-магазину, ви можете вибрати свою бізнес-модель відповідно до ваших вимог.

Виберіть доменне ім'я

Ім'я домену має відповідати вашим продуктам і бути легко запам'ятовуватися для вашої цільової аудиторії.

Створіть базу даних та WEB-сторінку магазину

Налаштуйте платіжний шлюз та захист сайту

Клієнт повинен мати можливість здійснювати платежі за допомогою кредитної картки, дебетової картки, онлайн-гаманців, інтернет-банкінгу, післяплати. Сертифікат SSL забезпечує безпеку вашого сайту і підвищує довіру ваших клієнтів. В даний час Google рекомендує мати сертифікат SSL для кожного сайту.

1.2. Призначення розробки та галузь застосування

Інтернет охоплює весь світ, тому шукати клієнтів та надавати послуги можна практично у будь-який куточок Землі. На сьогоднішній день чисельність населення нашої планети становить 7,5 мільярдів, з яких станом на 2017 рік інтернет-користувачами є 3,9 мільярдів. Нехай ці цифри розчинять ваш страх перед конкуренцією – у таких масштабах добрий продукт точно знайде свою аудиторію. Працюючи в інтернеті, ви самі можете визначити, коли та скільки годин присвячувати своїй діяльності. Це може бути 4 години на день, а може 20, якщо раптом хочете заробити більше і рухатися швидше. Але другий варіант

категорично не радимо - він дуже шкодить здоров'ю. Як і з клієнтами, в інтернет-бізнесі куди простіше знаходити партнерів та укладати угоди, бо фізична присутність не є обов'язковою. Над початком підприємницької діяльності часто замислюються люди, які відчувають, що не можуть реалізувати свій потенціал на найманій роботі. Мрія бути бізнес-коучем, психологом або втілювати круті ідеї створення сайтів стає реальною з приходом Інтернету в наше життя. Тут кожен може знайти свою нішу та свою аудиторію. А кожен споживач – власний контент.

Саме тому інтернет-магазин на сьогодні – необхідна частина будь-якого підприємства.

1.3. Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується з наказом ректора.

Таким чином підставами для розробки (виконанням кваліфікаційної роботи) є:

– освітня програма спеціальності 122 «Комп’ютерні науки»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка»;

– завдання на кваліфікаційну роботу на тему «Розробка веб-магазину з продажу комп'ютерної периферії за допомогою мови JavaScript».

1.4. Постановка завдання

Завданням даної роботи є створення сайту з продажу комплектуючих для ПК за допомогою мови JavaScript. Основними характеристиками розробки повинні бути:

– Зручна навігація по сайту

– Зберігання та доповнення даних про товари;

– Створення розмітки та дизайну.

Поставлена задача може бути досягнута при виконанні наступних вимог:

– вивчення предметної області завдання;

– проведення порівняльної характеристики можливостей аналогічних продуктів;

– вибір платформи розробки;

– написання програмного коду.

Кінцевим результатом має бути WEB-сторінка зі зручною навігацією, можливістю вибрати, отримати інформацію та купити товар.

1.5. Вимоги до програми або програмного виробу

В результаті роботи ми повинні отримати повноцінний інтернет-магазин з продажу компютерної периферії, написаний за допомогою мови JavaScript, на основі NodeJS і з використанням фреймворку ReactJS.

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт повинен дотримуватися наступних функціональних вимог:

– інтуїтивно зрозумілий інтерфейс користувача;

– можливість оперувати товарами в корзині;

– система фільтрації товарів.

1.5.2. Вимоги до інформаційної безпеки

Для безпеки нашого сайту необхідно зробити наступні кроки:

- Використання протоколу HTTPS;
- Вхід адміністратора тільки через SSH;

- Захист від DDOS – атак;
- Захист від XSS - атак;
- Захист від SQL - ін’єкцій;

1.5.3. Вимоги до складу та параметрів технічних засобів

Для коректної роботи сайту вам потрібні наступні параметри:

- WEB-браузер;
- IE > 11;
- ОЗУ > 1 ГБ

React 16 покладається на типи колекцій Map та Set. У багатьох старих пристроях та браузерах (наприклад, IE < 11) ці типи колекцій відсутні. А в інших вони мають невідповідну реалізацію (скажімо, IE 11).

1.5.4. Вимоги до інформаційної та програмної сумісності

Для роботи з системою не потрібне встановлення жодного додаткового забезпечення на комп'ютери користувачів. Користувачі просто переглядають сторінки сайту у браузері.

Як основна мова програмування використовувався JavaScript.

Додаток було написано в WebStorm (4.1).

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом даної кваліфікаційної роботи має бути інтернет-магазин з продажу комплектуючих для пк.

Основний функціонал додатка полягає в можливості клієнта знайти, вибрати, купити товар, використовуючи систему фільтрації та пошук.

Крім цього, для кращої і якісної роботи програми, повинен бути присутнім наступний додатковий функціонал:

– можливість доповнювати та редагувати базу даних;

– налаштування корзини, аби користувач міг оперувати вибраними товарами;

– налаштування авторизації на сайті.

2.2. Опис застосованих математичних методів

При створенні інтернет-магазину не було використано складних математичних методів. При організації БД були використані знання про побудову алгоритму дій.

2.3. Опис використаних технологій та мов програмування

Дворівнева архітектура веб-сайту передбачає поділ розміщення інформації у двох різних місцях - на рівні представлення (або інтерфейсу) на стороні клієнта та на рівні даних на стороні сервера. Поділ цих двох компонентів на окремі розташування і є дворівневою архітектурою.

Клієнтська програма зазвичай запускається на клієнтському комп'ютері для збору даних від клієнта і передачі їх назад на сервер бази даних, створюючи тим самим узгоджену взаємодію між двома рівнями.

Переваги дворівневої архітектури:

- Зниження навантаження на сервер та клієнтські машини
- Зниження мережевого трафіку та підвищення ефективності обробки інформації за рахунок оптимізації та буферизації введення-виводу
- Захист даних за допомогою системи управління базами даних (СУБД), що дозволяє блокувати дії, дозволені користувачеві
- Сервер реалізує контроль транзакцій та може блокувати спроби одночасної зміни ідентичних записів.

Недоліки дворівневої архітектури:

- Бізнес-логіка функціональної обробки та подання даних може бути однаковою для кількох клієнтських програм, що збільшує потребу в ресурсах (повторення програмного коду та запитів).
- Оскільки клієнт бачить більшу частину логіки програми, виникають проблеми з контролем версії ПЗ та повторним розповсюдженням нових версій.
- Двохрівневої моделі не вистачає масштабованості, оскільки вона підтримує лише обмежену кількість користувачів. Коли кількість одночасних клієнтських запитів збільшується, продуктивність програми швидко знижується через те, що клієнтам потрібні окремі з'єднання і пам'ять ЦП.

Рис.2.1. Типова модель архітектури для інтернет-магазину

За основу взято мову програмування JavaScript.

В серверній частині проекту використано Express.js, або просто Express — програмний каркас розробки серверної частини вебзастосунків для Node.js, реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT. Він спроєктований для створення вебзастосунків і API. Де-факто є стандартним каркасом для Node.js.

Імпортуємо express в наш node-застосунок

```
const express = require('express');
```

тепер ініціалізуємо застосунок

```
const app = express();
```

тепер ми можемо зареєструвати функцію зворотного виклику для певного GET-запиту і надавати текстову відповідь

```
app.get('/', (req,res)=>{
   res.send('Привіт, ми отримали ваш запит')
})
```

тепер почнемо слухати порт

```
app.listen(3000, ()=>{
   console.log('слухаємо https://localhost:3000')
})
```

Як асинхронне подієве JS–оточення, Node.js спроектований для побудови масштабованих мережевих додатків. Нижче наведений приклад "hello world", який може одночасно обробляти багато з'єднань. Для кожного з'єднання викликається функція зворотнього виклику, проте коли з'єднань немає Node.js засинає.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Рис. 2.2. Функція зворотнього виклику

Це контрастує з більш загальною моделлю в якій використовуються паралельні OS потоки. Такий підхід є відносно неефективним та дуже важким у використанні. Більше того, користувачі Node.js можуть не турбуватись про блокування процесів, оскільки немає жодних блокувань. Майже жодна з функцій у Node.js не працює напряму з I/O, тому процес не блокується ніколи. Оскільки нічого не блокується на Node.js легко розробляти масштабовані системи.

Node.js створений під впливом таких систем як Event Machine в Ruby або Twisted в Python. Node.js використовує подієву модель значно ширше, він приймає цикл подій (event loop) за основу оточення, замість того, щоб використовувати його в якості бібліотеки. В інших системах завжди стається блокування виклику, щоб запустити цикл подій.

Зазвичай поведінка визначається через функції зворотнього виклику на початку скрипта і в кінці запускає сервер через блокуючий виклик, як от EventMachine::run(). В Node.js немає нічого подібного на виклик початку циклу подій. Node.js просто входить в подієвий цикл після запуску скрипта на виконання. Node.js виходить з подієвого циклу тоді, коли не залишається зареєстрованих функцій зворотнього виклику. Така поведінка схожа на поведінку браузерного JavaScript: подієвий цикл прихований від користувача.

HTTP є об'єктом першого роду в Node.js, розробленим з потоковістю та малою затримкою. Це робить Node.js хорошою основою для веб–бібліотеки або фреймворку.

Використано Sequelize — це інструмент Node.js ORM на основі рішень для Postgres, MySQL, MariaDB, SQLite, DB2 та Microsoft SQL Server. Він має надійну підтримку транзакцій, швидке завантаження, тощо. Sequelize підтримує Node v10 і вище.

Для керування базами даних використано PostgreSQL. Це система керування базами даних (СКБД). Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

Порівняно з іншими проєктами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СКБД та впроваджувати у неї найновіші досягнення.

У фронтенді взято фреймворк ReactJS. Він спрощує створення інтерактивних інтерфейсів. Потрібно лише описати, як різні частини інтерфейсу виглядають у кожному стані додатку і React ефективно оновить та відрендерить лише потрібні компоненти, коли дані зміняться. React також може рендеритись на сервері, використовуючи Node, і приводити в дію мобільні додатки, які використовують React Native.

2.4. Опис структури програми та алгоритми її функціонування

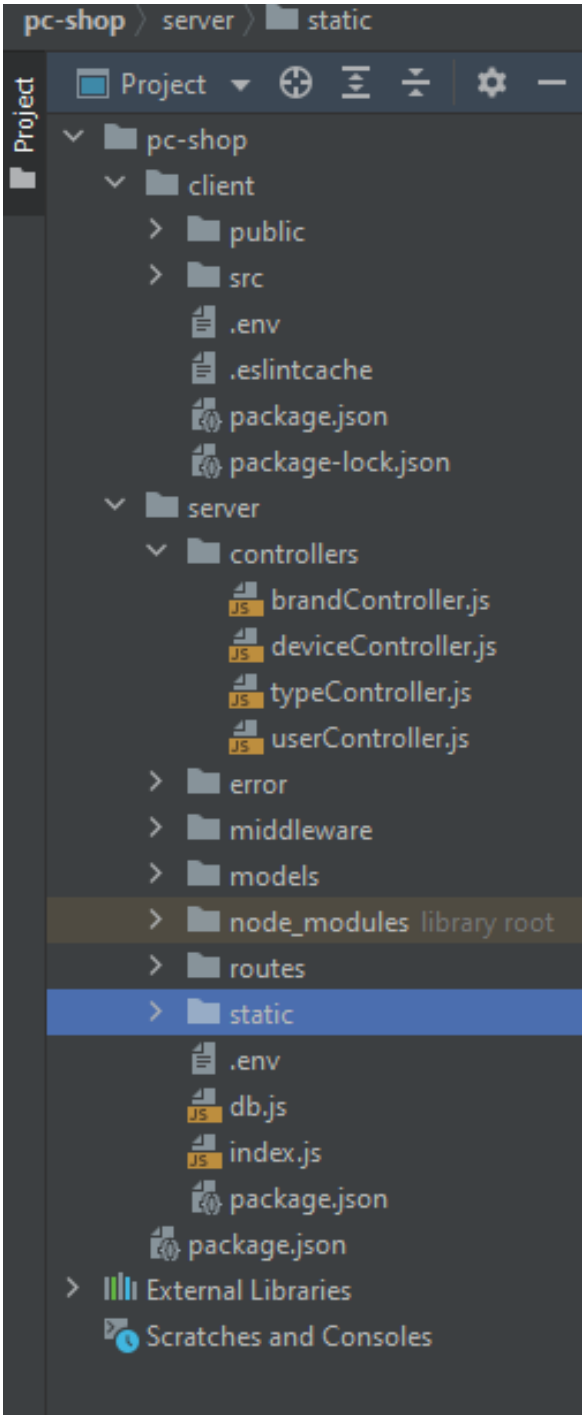

Рис.2.3. Структура програми в додатку WebStorm

Чим менше часу і сил користувач витрачає на роботу з сайтом, тим лояльнішим він стає до компанії. Існують сотні інтернет-магазинів, і ніхто не «перемагатиме» і терпітиме незручності, щоб скористатися саме вашою

пропозицією. Користувачеві простіше закрити вкладку і знайти «дружній» портал.

Щоб користувачеві було зручно працювати із сайтом, потрібно розробити правильну структуру. Це «скелет» порталу, який відіграє ключову роль зручності використання. Як приклад візьмемо текст. Він може бути страшенно корисним, але відсутність абзаців і підрозділів приведе читача до зневіри: ніхто не хоче мати справу з «полотном». Те саме відбувається з сайтами: нікому не подобаються портали, що нагадують лабіринт.

Структура сайту – це логічний розділ порталу на блоки, розташовані в ієрархічному порядку. Простіше кажучи, це схема розміщення товарних розділів, інформаційних сторінок, карток товарів, службових сторінок. Відобразивши структуру на папері чи моніторі комп'ютера, можна за допомогою одного погляду оцінити цільну картину інтернет-магазину. Приклад структури – малюнку.

Поясню на прикладі мого інтернет-магазину. Візьмемо по одному пункту кожного ієрархічного блоку. Головна – це стартова сторінка, категорія – процесори, товар – Intel Core i7-11700KF, властивість – 5MHz. Властивістю може бути частота, сокет, кеш-память. Після натискання на певну властивість система видасть, наприклад, найпопулярніші процесори від виробників Intel та AMD. (рис. 2.4.).

Також реалізовано сортування за зростанням (рис. 2.5), за спаданням ціни (рис. 2.6)

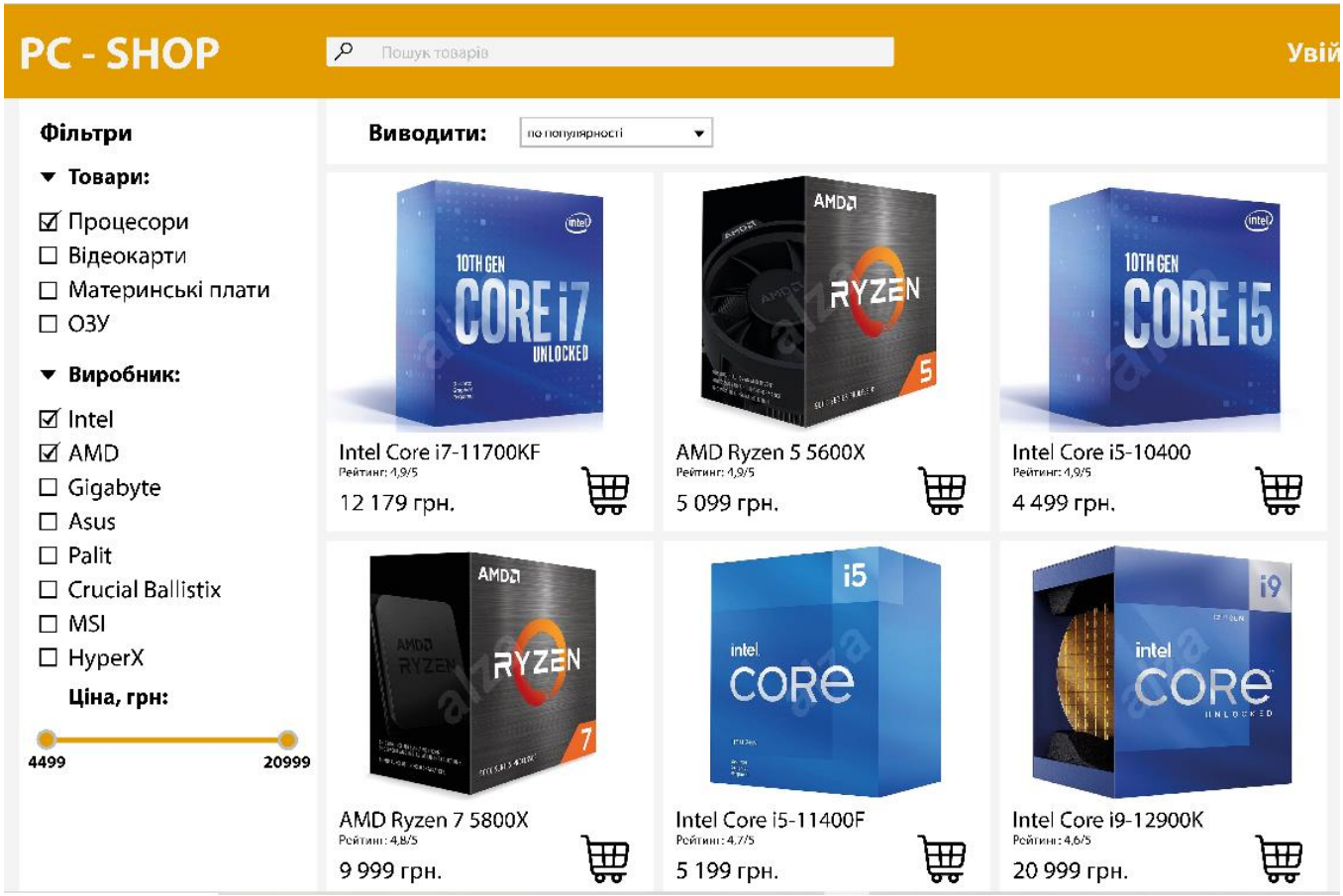

Рис.2.4. Сортування процесорів за популярністю

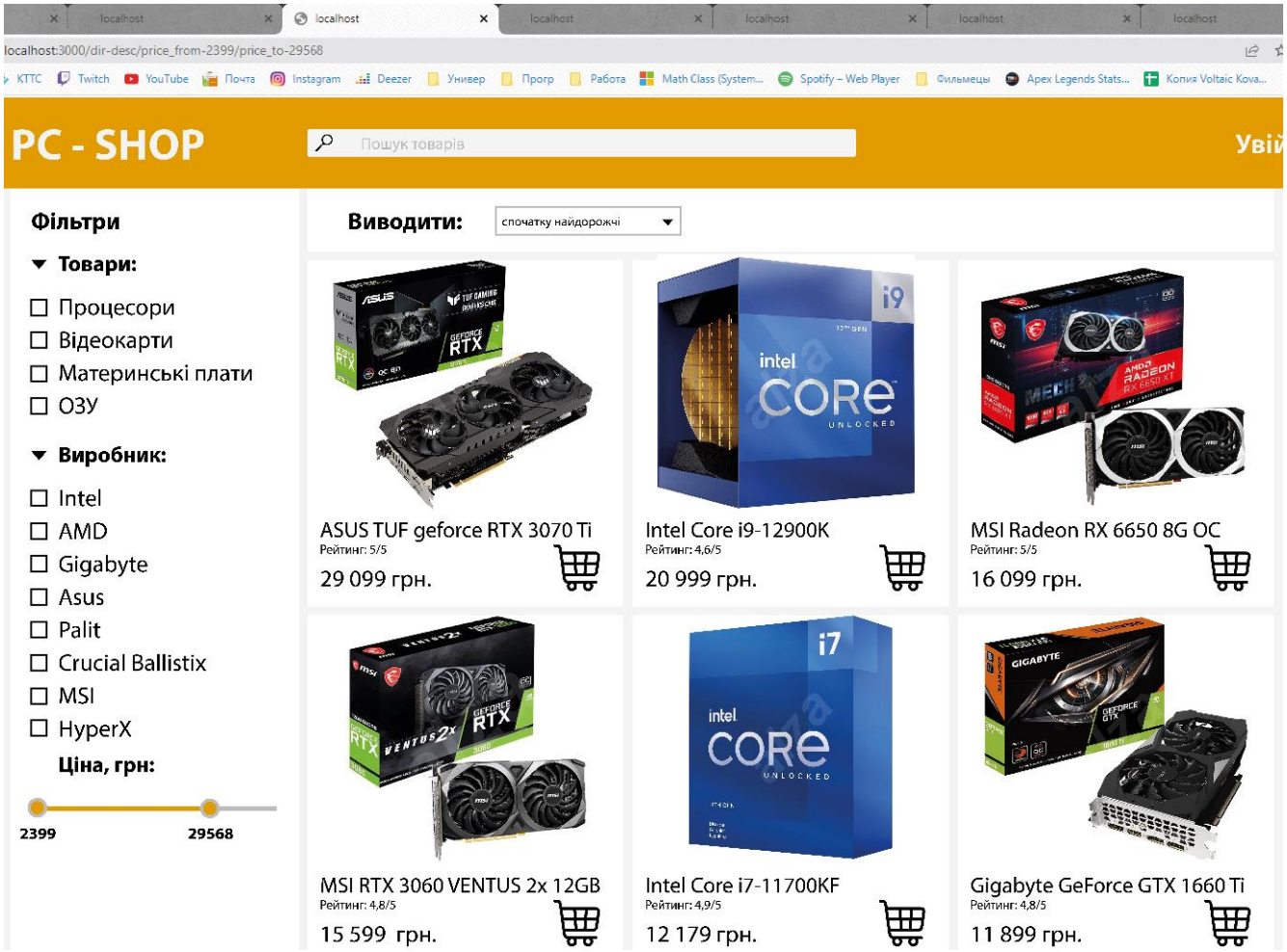

Рис.2.5. Сортування товарів за ціною з найдорожчих

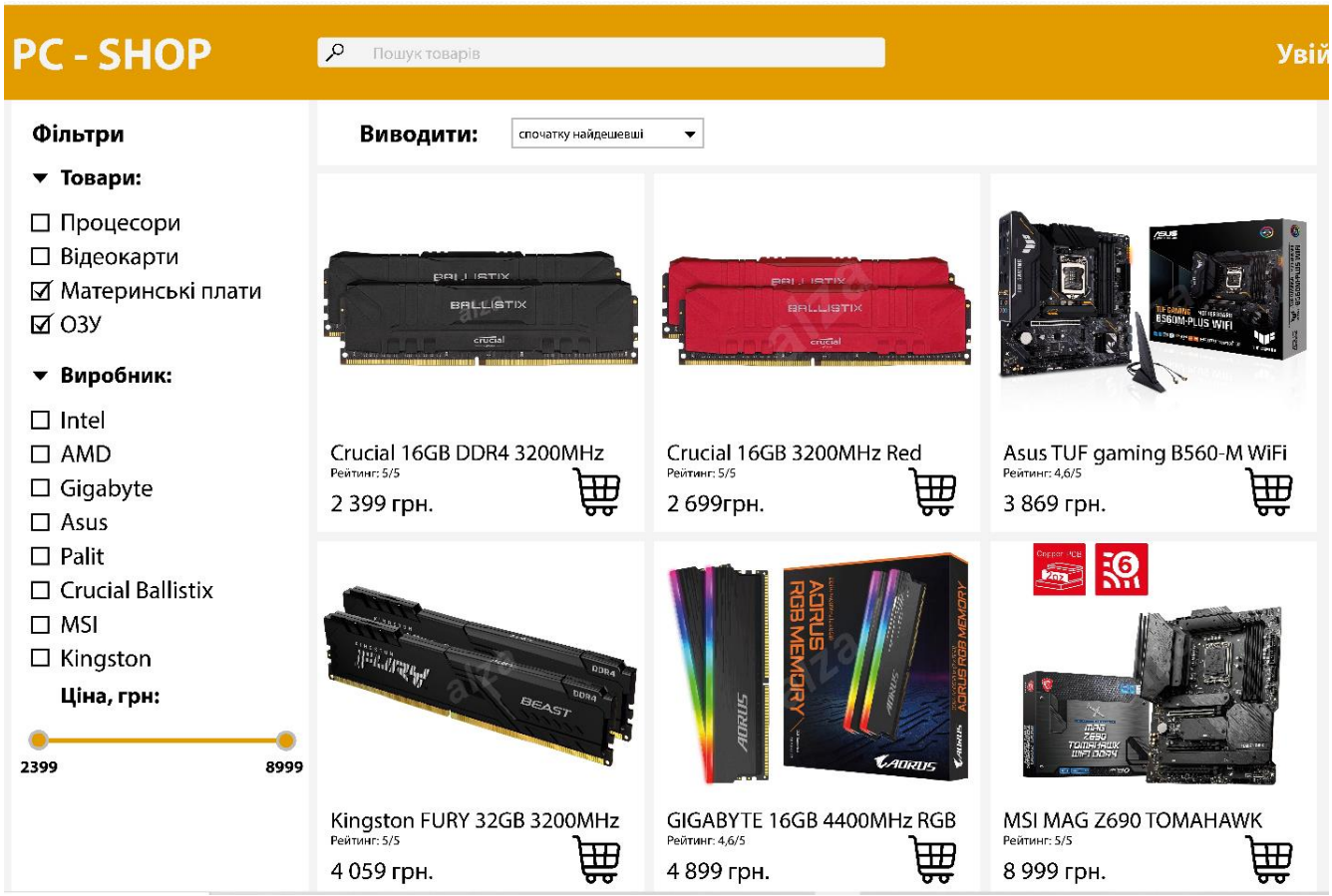

Рис.2.6. Сортування деяких категорій за зростанням ціни

JavaScript витягує всі елементи на сторінці, які мають атрибут data-sort:

```
collection = document.querySelectorAll("[data-sort]");
```

Наразі елементи записані у тому порядку, в якому вони є на сторінці (3, 4, 2, 1). Але мені потрібно, щоб у колекції вони зберігалися у порядку, вказаному в самому атрибуті data-sort.

Я вирішив використати функцію sort(). Але ця функція доступна лише масивів. Давайте отримаємо масив із атрибутів:

```
dataAttrs = [];
collection.forEach(function(item, i, coll) {
  dataAttrs[i] = item.getAttribute("data-sort");
```

});

Тепер маємо масив виду 3, 4, 2, 1. Його вже можна відсортувати:

```
dataAttrs.sort(function(a, b) {
  return a - b;
});
```

Принцип роботи сортування показано на рис. 2.5.

```
document.querySelector('#sort-asc').onclick = function () {
    mySort('data-price');
}
document.querySelector('#sort-desc').onclick = function () {
    mySortDesc('data-price');
}
document.querySelector('#sort-rating').onclick = function () {
    mySortDesc('data-rating');
}
```

Рис.2.7. Приклад коду сортування

Також було реалізовано сортування за ціною і слайдер (рис. 2.4.). Приклад коду та принцип роботи описаний нижче:

```
const rangeSlider = document.getElementById('range-slider');
if (rangeSlider) {
noUiSlider.create(rangeSlider, {
    start: [500, 999999],
connect: true
step: 1
    range: {
'min': [500],
'max': [999999]
    }
});
const input0 = document.getElementById('input-0');
const input1 = document.getElementById('input-1');
const inputs = [input0, input1];
rangeSlider.noUiSlider.on('update', function(values, handle){
inputs[handle].value = Math.round(values[handle]);
});
```

```
const setRangeSlider = (i, value) => {
let arr = [null, null];
arr[i] = value;
console log(arr);
rangeSlider.noUiSlider.set(arr);
};
inputs.forEach((el, index) => {
el.addEventListener('change', (e) => {
console log(index);
setRangeSlider(index, e.currentTarget.value);
});
});
}
```

Наступний пункт – авторизація в системі. Реалізовано за допомогою JWT Access Token, bcrypt

```
const User = require('./models/User')
const Role = require('./models/Role')
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { validationResult } = require('express-validator')
const {secret} = require("./config")

const generateAccessToken = (id, roles) => {
   const payload = {
      id,
      roles
   }
   return jwt.sign(payload, secret, {expiresIn: "24h"} )
}

class authController {
   async registration(req, res) {
      try {
         const errors = validationResult(req)
         if (!errors.isEmpty()) {
            return res.status(400).json({message: "Помилка при регістрації", errors})
         }
         const {username, password} = req.body;
         const candidate = await User.findOne({username})
         if (candidate) {
            return res.status(400).json({message: "Користувач з таким ім'ям вже
існує"})
         }
```

Рис.2.8.1. Приклад роботи JWT

```
const hashPassword = bcrypt.hashSync(password, 7);
            const userRole = await Role.findOne({value: "USER"})
            const user = new User({username, password: hashPassword, roles: [userRole.value]})
            await user.save()
            return res.json({message: "Користувач успішно зареєстрований"})
        } catch (e) {
            console.log(e)
            res.status(400).json({message: 'Registration error'})
        }
    }

    async login(req, res) {
        try {
            const {username, password} = req.body
            const user = await User.findOne({username})
            if (!user) {
                return res.status(400).json({message: `Користувача ${username} не знайдено`})
            }
            const validPassword = bcrypt.compareSync(password, user.password)
            if (!validPassword) {
                return res.status(400).json({message: `Невірний пароль`})
            }
            const token = generateAccessToken(user._id, user.roles)
            return res.json({token})
        } catch (e) {
            console.log(e)
            res.status(400).json({message: 'Login error'})
        }
    }

    async getUsers(req, res) {
        try {
            const users = await User.find()
            res.json(users)
        } catch (e) {
            console.log(e)
        }
    }
}

module.exports = new authController()
```

Рис.2.8.2. Приклад коду Jwt (частина 2)

Також на сайті реалізовано пошукове вікно для зручної новігації, приклад коду та роботи описано на рис.2.5.

```
document.querySelector('#elastic').oninput = function () {
    let val = this.value.trim();
    let elasticItems = document.querySelectorAll('.elastic li');
    if (val != '') {
        elasticItems.forEach(function (elem) {
            if (elem.innerText.search(val) == -1) {
                elem.classList.add('hide');
                elem.innerHTML = elem.innerText;
            }
            else {
                elem.classList.remove('hide');
                let str = elem.innerText;
                elem.innerHTML = insertMark(str,elem.innerText.search(val), val.length);
            }
        });
    }
    else {
        elasticItems.forEach(function (elem) {
            elem.classList.remove('hide');

            elem.innerHTML = elem.innerText;
```

Рис.2.9. Приклад коду вікна пошуку товарів

Алгоритм пошукових систем влаштований в такий спосіб, що сторінки інтернет-магазинів без описів товарів мало індексуються. Це означає, що товари без описів не потраплять у пошукову видачу, а клієнти відповідно їх менше купуватимуть. Додатково за допомогою зрозумілих і корисних описів, можливо, вдасться схилити до покупки клієнта, який ще не визначився, що саме йому потрібно і де це купити. На моєму сайті вікно перегляду конкретного товару показано на рис. 2.9.

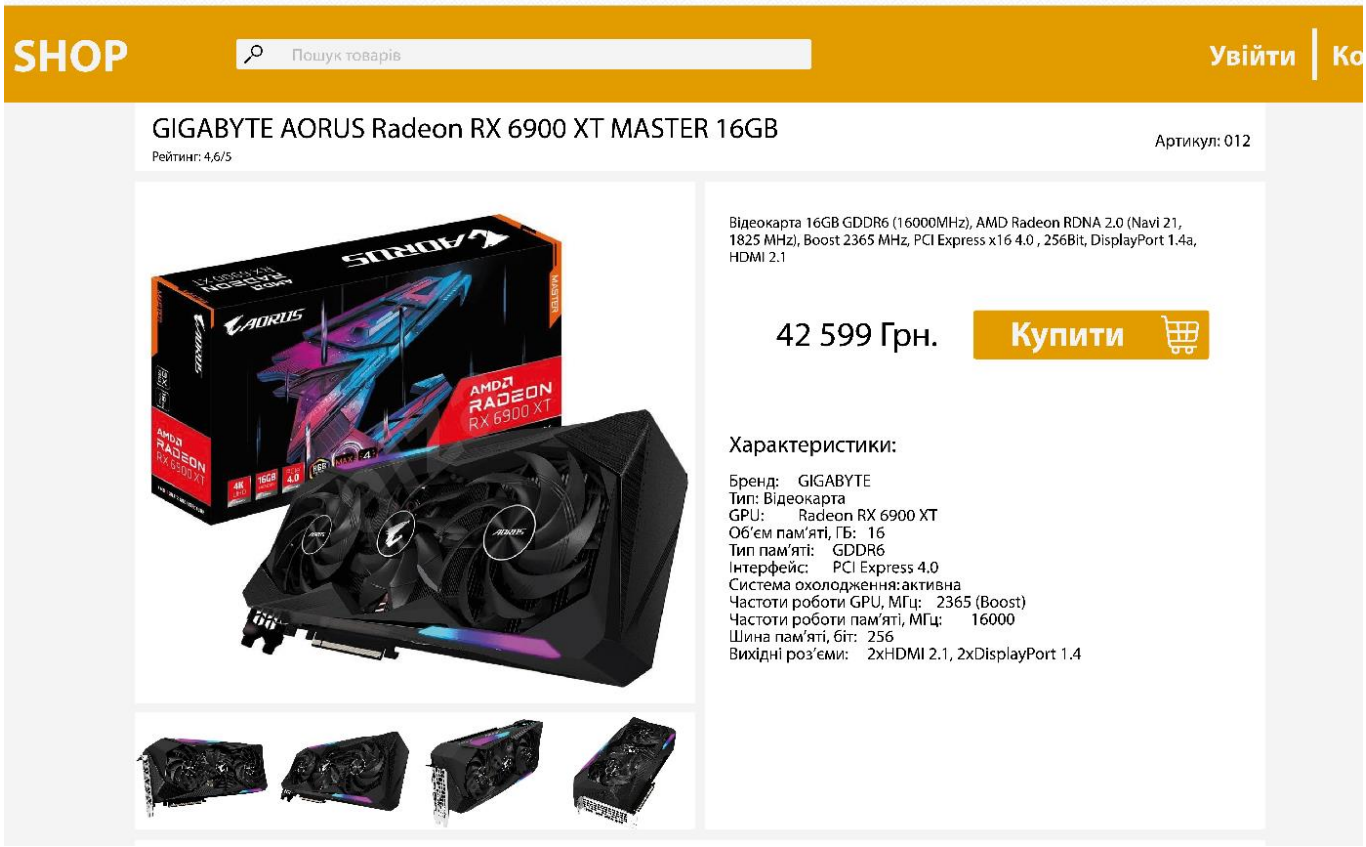

Рис.2.9. Сторінка перегляду товара

Щоб зробити кошик товарів, на першому кроці нам знадобиться інформація про товари. Як правило, це масив даних, який зберігається, відповідно в БД.

Підключаємо скрипт кошика до сторінки. Виводимо товари звичним способом у циклі:

```
<button type="button" onclick='cartLS.add(<?php echo json_encode($item, JSON_UNESCAPED_UNICODE); ?>)'>
```

Потім йде кнопка, яка додає товар у кошик, тобто LocalStorage у форматі JSON.

```
const $block_cart = document.querySelector(".cart");
const $total = document.querySelector(".total");
const $count = document.querySelector("#count");
$block_cart.innerHTML = items.map((item) => `
<img src="${item.img}"/>
<p>${item.name}</p>
<p>Колір: ${item.color}</p>
```

```
<button type="button" onClick="cartLS.quantity(${item.id},-1)">-</button>
<div>${item.quantity}</div>
<button type="button" onClick="cartLS.quantity(${item.id},1)">+</button>
<h5>$${item.price}</h5>
<a href="#!" onClick="cartLS.remove(${item.id})">Видалити</a>`).join("")
$total.innerHTML = "$" + cartLS.total();
$count.innerHTML = cartLS.list().length;
renderCart(cartLS.list())
cartLS.onChange(renderCart)
```

Параметр JSON_UNESCAPED_UNICODE потрібен, щоб звернути дані (кирилицю) в кодування, тобто в UTF-8.

Тепер напишемо JavaScript функцію, яка формуватиме товари в кошику і виводитиме їх на сторінку.

Якщо потрібно порахувати загальну суму кожного товару окремо (якщо їх кілька), то пишемо наступне:

```
var items = cartLS.list();
var tot = items.reduce((sum, item) => sum + item.quantity, 0);
console.log(tot);
```

Тобто вартість множимо на кількість. Також не зайвим буде дізнатися загальну кількість одиниць товарів усіх разом. У цьому нам допоможе цей фрагмент коду:

```
var items = cartLS.list();
var tot = items.reduce((sum, item) => sum + item.quantity, 0);
console.log(tot);
item.price * item.quantity
```

Авторизація реалізована через JWT та bscript

JSON Web Token (JWT) – це JSON об'єкт, який визначено у відкритому стандарті RFC 7519. Він вважається одним із безпечних способів передачі інформації між двома учасниками. Для створення необхідно визначити заголовок (header) із загальною інформацією по токену, корисні дані (payload), такі як id користувача, його роль тощо. та підписи (signature).

header.payload.signature.

Припустимо, що хочемо зареєструватися на сайті. У нашому випадку є три учасники - користувач user, сервер application server і сервер автентифікації authentication server. Сервер автентифікації забезпечуватиме користувача токеном, за допомогою якого він пізніше зможе взаємодіяти з програмою.
JWT складається з трьох частин: заголовок header, корисні дані платioload і підпис підпису.

1. Створюємо HEADER

Хедер JWT містить інформацію про те, як повинен обчислюватися JWT підпис. Хедер це теж JSON об'єкт, який виглядає наступним чином:

header = {"alg": "HS256", "type": "JWT"}

Поле typ не каже нам нічого нового, тільки те, що це JSON Web Token. Цікавішим тут буде поле alg, яке визначає алгоритм хешування. Він використовуватиметься під час створення підпису. HS256 - не що інше, як HMAC-SHA256, для його обчислення потрібен лише один секретний ключ. Ще може використовуватися інший алгоритм RS256 — на відміну від попереднього, він є асиметричним та створює два ключі: публічний та приватний. За допомогою приватного ключа створюється підпис, а за допомогою публічного лише перевіряється справжність підпису, тому нам не потрібно турбуватися про його безпеку.

2. Створюємо PAYLOAD

Payload – це корисні дані, що зберігаються всередині JWT. Ці дані також називають JWT-claims (заявки). У прикладі, який ми розглядаємо, сервер автентифікації створює JWT з інформацією про id користувача — userId.

payload = { "userId": "b08f86af-35da-48f2-8fab-cef3904660bd" }

Існує список стандартних заявок для JWT payload – ось деякі з них:

iss (issuer) - визначає програму, з якої відправляється токен.

sub (subject) - визначає тему токена.

exp (expiration time) - час життя токена.

3. Створюємо SIGNATURE

Підпис обчислюється з використанням наступного коду:

```
const SECRET_KEY = 'cAtwa1kkEy'
const unsignedToken = base64urlEncode(header) + '.' + base64urlEncode(payload)
const signature = HMAC-SHA256(unsignedToken, SECRET_KEY)
```

Алгоритм base64url кодує хедер та payload, створені на 1 та 2 кроці. Він з'єднує кодовані рядки через точку. Потім отриманий рядок хешується алгоритмом, заданим у хедері на основі нашого секретного ключа.

4. Тепер об'єднаємо всі три JWT компоненти разом

Тепер, коли ми маємо всі три складові, ми можемо створити наш JWT. Це досить просто, ми поєднуємо всі отримані елементи в рядок через точку.

Повний код реалізації авторизації на сайті (userController.js):

```
const ApiError = require('../error/ApiError');
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')
const {User, Basket} = require('../models/models')

const generateJwt = (id, email, role) => {
return jwt.sign(
{id, email, role},
process.env.SECRET_KEY,
{expiresIn: '24h'}
    )
}

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

async login(req, res, next) {
const {email, password} = req.body
const user = await User.findOne({where: {email}})
if (!user) {
return next(ApiError.internal('Користувача не знайдено'))
        }
let comparePassword = bcrypt.compareSync(password, user.password)
if (!comparePassword) {
return next(ApiError.internal('Вказано невірний пароль'))
        }
const token = generateJwt(user.id, user.email, user.role)
return res.json({token})
    }
```

Рис.2.10. Код авторизації на сайті

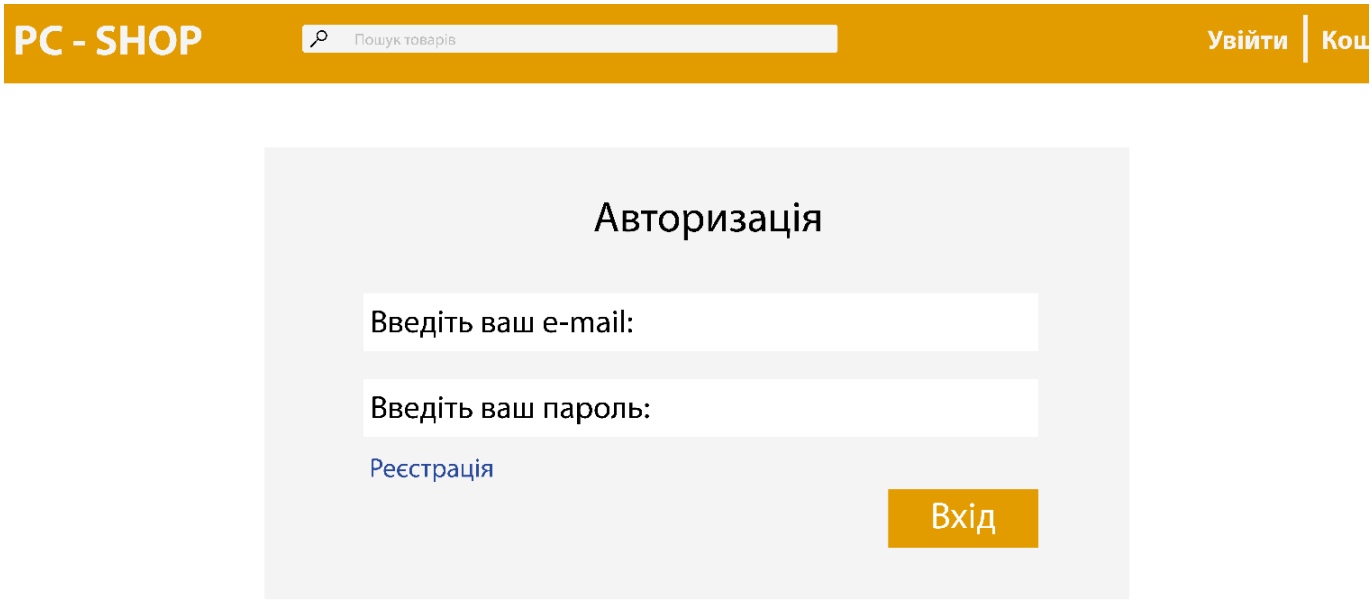

Рис.2.11. Авторизація на сайті

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Інтерфейс сайту простий і не потребує для користування додаткового обладнання.

Механізм введення представляє собою матеріальний пристрій для прямої взаємодіх користувача з сайтом. Тобто, цими механізмами є стандартні периферійні пристрої: клавіатура, миша, сенсорний дисплей телефону тощо.

Механізм виведення – візуальний. Користувач одразу після виконання якоїсь дії (нажаття на кнопку, вібір фільтрів тощо) отримує інформацію. Головні функції виводу даних – відображення інформації про товари, їх сортування, оперування їми в корзині, можливість придбати.

2.6. Опис розробленого програмного продукту

Сайт має головну сторінку з товарами. Можливість сортування за брендами, типами товарів, ціною, популярністю. Має такоє можливість регістрації та авторизації, оперування товарами в корзині.

Сам сайт, додатки, сервіси та дані цього магазину потрібно десь розмістити, тобто потрібно побудувати інфраструктуру.

В роботі я вибрав хмарний хостинг. Тут я беру в оренду частину ресурсів платформи, в якій реалізовано високий рівень автоматизації та можливостей самообслуговування. Можна розгорнути віртуальний сервер, створювати мережі, користуватися платформою контейнеризації, СУБД, чергами повідомлення та іншими сервісами без витрат часу та праці. Віртуальний сервер можна настроювати на свій розсуд, вибирати потрібну потужність, підключати додаткові заходи захисту.

Відкрити сторінку інтернет-магазину з продажу комплектуючих для ПК може користувач з будь-якого пристрою, який відповідає мінімальним вимогам для запуску браузера

2.6.1. Використані технічні засоби

При розробці програми була використана персональна ЕОМ з наступними характеристиками:

– Процесор Intel Core i7-8700k;

– Відеопроцесор Nvidia GeForce RTX 3060Ti (8 ГБ GDDR6);

– Оперативна пам'ять 32 ГБ DDR4-2666.

Тестування проводилося на таких пристроях:

– ПК на Windows 10;

– Mi 10 Note;

– iPhone 12;

2.6.2. Використані програмні засоби

Додаток було написано в WebStorm - інтегрованому середовищі розробки для JavaScript, HTML та CSS від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA. WebStorm є спеціалізованою версією PhpStorm, пропонуючи підмножину з його можливостей. WebStorm постачається з перед-установленим плагінами JavaScript (такими як для Node.js).

2.6.3. Виклик та завантаження програми

Користувачу необхідно відкрити WEB-сторінку інтернет-магазину у браузері.

2.6.4. Опис інтерфейсу користувача

Інтерфейс магазина є стандартним та зручним для розуміння користувачу будь-якого рівня. Після відкриття посилання користувач може побачити головну сторінку с переліком товарів з найдорожчих (рис.2.9). Також можна побачити поле пошуку товарів, кнопка авторизації, PC-SHOP (головна), кнопка корзини. Увесь функціонал сайта є інтуїтивно зрозумілим і не передбачає спеціалізованих знань у даній сфері.

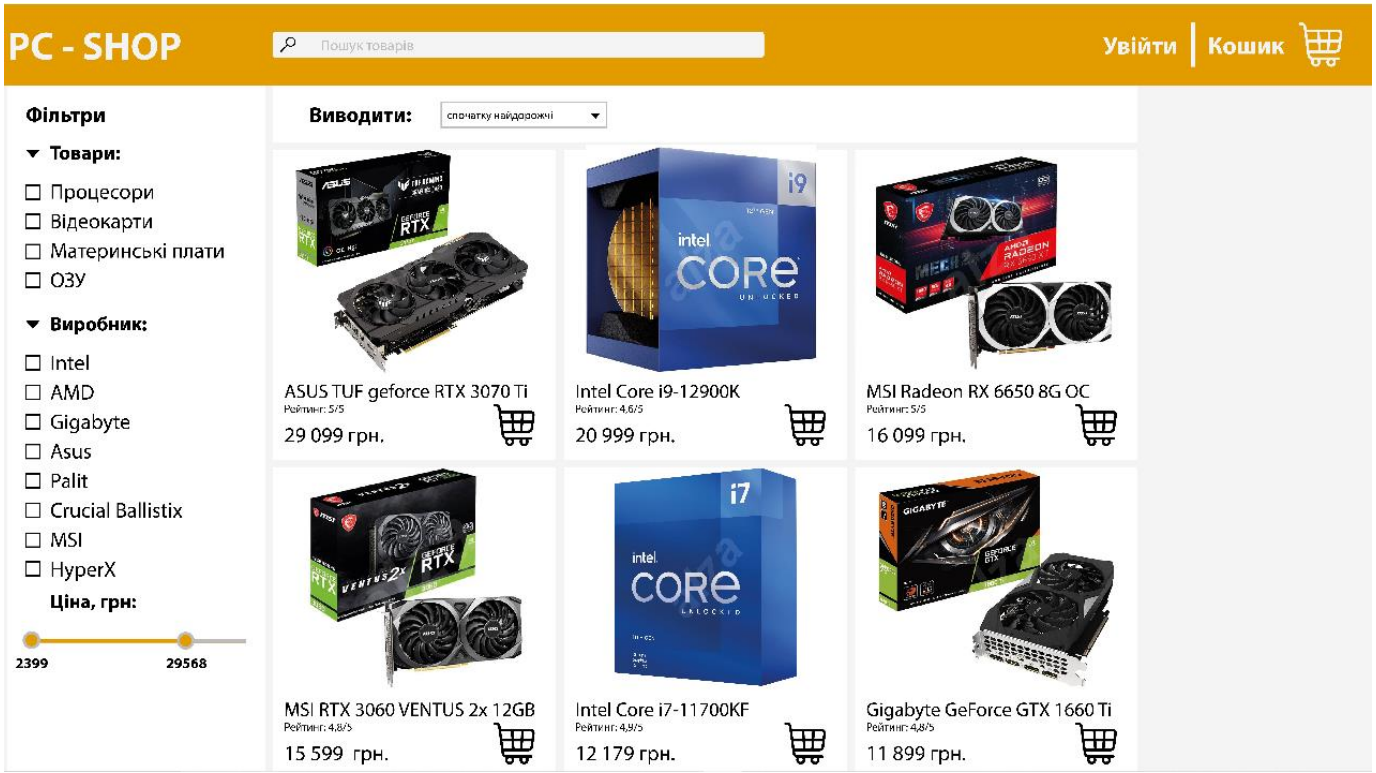

Рис.2.12. Головна сторінка сайта

РОЗДІЛ 3
ЕКОНОМІЧНІЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вихідні дані:

1. передбачуване число операторів програми – 1435;
2. коефіцієнт складності програми – 1,3;
3. коефіцієнт корекції програми в ході її розробки – 0,1;
4. годинна заробітна плата full stuck developer – 269 грн/год (за версією сайта WorkUA) - https://www.work.ua/ru/salary-full+stack+developer/;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8;
7. вартість машино-години ЕОМ – 12 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \qquad (3.2)$$

де q – передбачуване число операторів (1435);

C – коефіцієнт складності програми (1,3);

p – коефіцієнт корекції програми в ході її розробки (0,1).

$$Q = 1435 \cdot 1{,}3 \cdot (1 + 0{,}1) = 2052;$$

Витрати праці на вивчення опису задачі ti визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot K}, \text{ людино-годин} \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,2);

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності (0,8);

$$t_u = \frac{2052 \cdot 1{,}2}{85 \cdot 0{,}8} = 36{,}21 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20 \ldots 25) \cdot K}; \qquad (3.4)$$

$$t_a = \frac{2052}{20 \cdot 0{,}8} = 128{,}25, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25) \cdot K}; \quad (3.5)$$

$$t_n = \frac{2052}{25 \cdot 0{,}8} = 102{,}6, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4...5) \cdot K}; \quad (3.6)$$

$$t_{\text{отл}} = \frac{2052}{5 \cdot 0{,}8} = 513, \text{ людино-годин,}$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{\text{к}} = 1{,}2 \cdot t_{\text{отл}}; \quad (3.7)$$

$$t_{\text{отл}}^{\text{к}} = 1{,}2 \cdot 513 = 615{,}6, \text{ людино-годин,}$$

Витрати праці на підготовку документації:

$$t_{\partial} = t_{\partial p} + t_{\partial o}; \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису;

$$t_{\partial} = \frac{Q}{(15...20) \cdot K}; \quad (3.9)$$

$$t_{\partial} = \frac{2052}{20 \cdot 0{,}8} = 128{,}25, \text{ людино-годин,}$$

де $t_{до}$ – трудомісткість редагування, печатки й оформлення документації;

$$t_{до} = 0{,}75 \cdot t_{др}; \qquad (3.10)$$

$$t_{до} = 0{,}75 \cdot 128{,}5 = 96{,}38, \text{ людино-годин.}$$

$$t_{\partial} = 128{,}25 + 96{,}38 = 224{,}63, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 36{,}21 + 128{,}25 + 102{,}6 + 513 + 224{,}63 = 1054{,}69, \text{ людино-годин.}$$

У результаті ми розрахували, що в загальній складності необхідно 1054,69 людино-годин для розробки даного програмного забезпечення.

3.2. Рахунок витрат на створення програми

Витрати на створення ПЗ *Кпо* включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \qquad (3.11)$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \qquad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата full stuck developer-а становить 269 грн/год, то отримаємо:

$$З_{ЗП} = 1054.69 \cdot 269 = 283\,712, \text{ грн.}$$

Вартість машинного часу $З_{МВ,}$ необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} \cdot C_{М}, \text{ грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$C_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 513 \cdot 12 = 6156 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 283712 + 5156 = 288868 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес.} \qquad (3.14)$$

де B_k – число виконавців;

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

Витрати на створення программного продукту:

$$T = \frac{1054{,}69}{1 \cdot 176} = 6 \text{ міс.}$$

Висновки. Інтернет-магазин має вартість 288868 грн. Ймовірний очікуваний час розробки – 6 місяці при стандартному 40-годинному робочому тижні і 178-годинному робочому місяці. Цей термін пов’язаний з кількістю операторів і включає в себе час для дослідження та розробку алгоритму розв’язання задачі, розробку дизайну і створення документації. На розробку інтернет-магазину буде витрачено 1055 людино-годин.

ВИСНОВКИ

Метою кваліфікаційної роботи бакалавра є розробка веб-магазину з продажу комп'ютерної периферії за допомогою мови JavaScript. В процесі написання кваліфікаційної роботи було встановлено, що інтернет-магазин – невід’ємна частина будь-якої компанії, його створення-необхідний крок для розвитку бізнесу

Сайт має додатковий функціонал для спрощення користуванням, а саме:

– пошук товарів через спеціальне вікно;

– фільтрація товарів;

– сортування за ціною та популярністю;

– оперування товарами в корзині.

Актуальність поставленої задачі обумовлюється широким попитом на такі програмні продукти. Створення інтернет магазину допоможе Вашій компанії зміцнити свої позиції на традиційних ринках і виходити на нові, за допомогою інтернет магазину Ви зможете збільшити свої продажі, а відповідно оборот і прибуток.

Особливо створення інтернет магазину, корисно, якщо ви підприємець-початківець, ви можете з мінімальними витратами організувати свій власний бізнес, а інтернет-магазин допоможе збільшити продажі та відкрити нові ринки збуту товарів.

В «Економічному розділі» визначено трудомісткість розробки програмного забезпечення (1054,69 чол-год), підраховані витрати на створення програмного забезпечення (288 868 грн.) і гаданий період розробки (6 міс.).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://ilion.digital/pravilnaya-struktura-internet-magazina/
2. https://uk.reactjs.org/
3. https://uk.wikipedia.org/wiki/Express.js
4. https://stackoverflow.com/
5. JavaScript: The Good Parts - Douglas Crockford (2008)
6. JavaScript and JQuery: Interactive Front-End Web Development - John Duckett (2013)
7. Learning JavaScript Design Patterns - Addie Osmani (2012)
8. https://nodejs.dev/learn
9. https://www.w3schools.com/nodejs/nodejs_intro.asp
10. https://github.com/nodejs/node
11. https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm
12. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction
13. https://www.itnetwork.cz/javascript/react/zaklady/uvod-do-react/
14. https://www.w3schools.com/REACT/DEFAULT.ASP
15. https://www.w3schools.com/bootstrap/
16. https://www.itnetwork.cz/html-css/bootstrap/kurz/uvod-do-css-frameworku-bootstrap
17. https://www.postgresql.org/about/
18. https://www.postgresqltutorial.com/
19. https://www.npmjs.com/package/sequelize
20. https://github.com/sequelize/sequelize
21. https://mobx.js.org/README.html
22. https://v5.reactrouter.com/web/api/Switch

ДОДАТОК А

КОД ПРОГРАМИ

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link
          rel="stylesheet"
          href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
          integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk"
          crossorigin="anonymous"
    />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
{
  "short_name": "Pc.shop",
  "name": "Create pc shop",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
# https://www.robotstxt.org/robotstxt.html
User-agent: *
Disallow:
import React, {useState} from 'react';
import Modal from "react-bootstrap/Modal";
import {Button, Form} from "react-bootstrap";
import {createBrand, createType} from "../../http/deviceAPI";

const CreateBrand = ({show, onHide}) => {
    const [value, setValue] = useState('')

    const addBrand = () => {
        createBrand({name: value}).then(data => {
            setValue('')
```

```
            onHide()
        })
    }
    return (
        <Modal
            show={show}
            onHide={onHide}
            centered
        >
            <Modal.Header closeButton>
                <Modal.Title id="contained-modal-title-vcenter">
                    Добавить тип
                </Modal.Title>
            </Modal.Header>
            <Modal.Body>
                <Form>
                    <Form.Control
                        value={value}
                        onChange={e => setValue(e.target.value)}
                        placeholder={"Введіть назву типу "}
                    />
                </Form>
            </Modal.Body>
            <Modal.Footer>
                <Button variant="outline-danger" onClick={onHide}>Закрити</Button>
                <Button variant="outline-success" onClick={addBrand}>Додати</Button>
            </Modal.Footer>
        </Modal>
    );
};

export default CreateBrand;
import React, {useContext, useEffect, useState} from 'react';
import Modal from "react-bootstrap/Modal";
import {Button, Dropdown, Form, Row, Col} from "react-bootstrap";
import {Context} from "../../index";
import {createDevice, fetchBrands, fetchDevices, fetchTypes} from "../../http/deviceAPI";
import {observer} from "mobx-react-lite";

const CreateDevice = observer(({show, onHide}) => {
    const {device} = useContext(Context)
    const [name, setName] = useState('')
    const [price, setPrice] = useState(0)
    const [file, setFile] = useState(null)
    const [info, setInfo] = useState([])

    useEffect(() => {
        fetchTypes().then(data => device.setTypes(data))
        fetchBrands().then(data => device.setBrands(data))
    }, [])

    const addInfo = () => {
        setInfo([...info, {title: '', description: '', number: Date.now()}])
    }
    const removeInfo = (number) => {
        setInfo(info.filter(i => i.number !== number))
    }
    const changeInfo = (key, value, number) => {
        setInfo(info.map(i => i.number === number ? {...i, [key]: value} : i))
    }

    const selectFile = e => {
        setFile(e.target.files[0])
```

```
}

const addDevice = () => {
    const formData = new FormData()
    formData.append('name', name)
    formData.append('price', `${price}`)
    formData.append('img', file)
    formData.append('brandId', device.selectedBrand.id)
    formData.append('typeId', device.selectedType.id)
    formData.append('info', JSON.stringify(info))
    createDevice(formData).then(data => onHide())
}

return (
    <Modal
        show={show}
        onHide={onHide}
        centered
    >
        <Modal.Header closeButton>
            <Modal.Title id="contained-modal-title-vcenter">
                Додати пристрій
            </Modal.Title>
        </Modal.Header>
        <Modal.Body>
            <Form>
                <Dropdown className="mt-2 mb-2">
                    <Dropdown.Toggle>{device.selectedType.name || "Виберіть тип"}</Dropdown.Toggle>
                    <Dropdown.Menu>
                        {device.types.map(type =>
                            <Dropdown.Item
                                onClick={() => device.setSelectedType(type)}
                                key={type.id}
                            >
                                {type.name}
                            </Dropdown.Item>
                        )}
                    </Dropdown.Menu>
                </Dropdown>
                <Dropdown className="mt-2 mb-2">
                    <Dropdown.Toggle>{device.selectedBrand.name || "Виберіть тип"}</Dropdown.Toggle>
                    <Dropdown.Menu>
                        {device.brands.map(brand =>
                            <Dropdown.Item
                                onClick={() => device.setSelectedBrand(brand)}
                                key={brand.id}
                            >
                                {brand.name}
                            </Dropdown.Item>
                        )}
                    </Dropdown.Menu>
                </Dropdown>
                <Form.Control
                    value={name}
                    onChange={e => setName(e.target.value)}
                    className="mt-3"
                    placeholder="Введіть назву пристрою"
                />
                <Form.Control
                    value={price}
                    onChange={e => setPrice(Number(e.target.value))}
                    className="mt-3"
                    placeholder="Введіть назву пристрою "
```

```
                    type="number"
                />
                <Form.Control
                    className="mt-3"
                    type="file"
                    onChange={selectFile}
                />
                <hr/>
                <Button
                    variant={"outline-dark"}
                    onClick={addInfo}
                >
                    Додати нову властивість
                </Button>
                {info.map(i =>
                    <Row className="mt-4" key={i.number}>
                        <Col md={4}>
                            <Form.Control
                                value={i.title}
                                onChange={(e) => changeInfo('title', e.target.value, i.number)}
                                placeholder="Введіть назву пристрою"
                            />
                        </Col>
                        <Col md={4}>
                            <Form.Control
                                value={i.description}
                                onChange={(e) => changeInfo('description', e.target.value, i.number)}
                                placeholder="Введіть описання"
                            />
                        </Col>
                        <Col md={4}>
                            <Button
                                onClick={() => removeInfo(i.number)}
                                variant={"outline-danger"}
                            >
                                Удалить
                            </Button>
                        </Col>
                    </Row>
                )}
            </Form>
        </Modal.Body>
        <Modal.Footer>
            <Button variant="outline-danger" onClick={onHide}>Закрити</Button>
            <Button variant="outline-success" onClick={addDevice}>Додати</Button>
        </Modal.Footer>
    </Modal>
  );
});

export default CreateDevice;
import React, {useState} from 'react';
import Modal from "react-bootstrap/Modal";
import {Form, Button} from "react-bootstrap";
import {createType} from "../../http/deviceAPI";

const CreateType = ({show, onHide}) => {
    const [value, setValue] = useState('')

    const addType = () => {
        createType({name: value}).then(data => {
            setValue('')
            onHide()
```

```
    })
  }

  return (
    <Modal
      show={show}
      onHide={onHide}
      centered
    >
      <Modal.Header closeButton>
        <Modal.Title id="contained-modal-title-vcenter">
          Додати тип
        </Modal.Title>
      </Modal.Header>
      <Modal.Body>
        <Form>
          <Form.Control
            value={value}
            onChange={e => setValue(e.target.value)}
            placeholder={"Введіть назву типу"}
          />
        </Form>
      </Modal.Body>
      <Modal.Footer>
        <Button variant="outline-danger" onClick={onHide}>Закрити</Button>
        <Button variant="outline-success" onClick={addType}>Додати</Button>
      </Modal.Footer>
    </Modal>
  );
};

export default CreateType;
import React, {useContext} from 'react';
import {Switch, Route, Redirect} from 'react-router-dom'
import {authRoutes, publicRoutes} from "../routes";
import {SHOP_ROUTE} from "../utils/consts";
import {Context} from "../index";
import {observer} from "mobx-react-lite";

const AppRouter = observer(() => {
  const {user} = useContext(Context)

  console.log(user)
  return (
    <Switch>
      {user.isAuth && authRoutes.map(({path, Component}) =>
        <Route key={path} path={path} component={Component} exact/>
      )}
      {publicRoutes.map(({path, Component}) =>
        <Route key={path} path={path} component={Component} exact/>
      )}
      <Redirect to={SHOP_ROUTE}/>
    </Switch>
  );
});

export default AppRouter;
import React, {useContext} from 'react';
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import {Card, Row} from "react-bootstrap";

const BrandBar = observer(() => {
```

```
    const {device} = useContext(Context)

    return (
        <Row className="d-flex">
            {device.brands.map(brand =>
                <Card
                    style={{cursor:'pointer'}}
                    key={brand.id}
                    className="p-3"
                    onClick={() => device.setSelectedBrand(brand)}
                    border={brand.id === device.selectedBrand.id ? 'danger' : 'light'}
                >
                    {brand.name}
                </Card>
            )}
        </Row>
    );
});

export default BrandBar;
import React from 'react';
import {Card, Col} from "react-bootstrap";
import Image from "react-bootstrap/Image";
import star from '../assets/star.png'
import {useHistory} from "react-router-dom"
import {DEVICE_ROUTE} from "../utils/consts";

const DeviceItem = ({device}) => {
    const history = useHistory()
    return (
        <Col md={3} className={"mt-3"} onClick={() => history.push(DEVICE_ROUTE + '/' + device.id)}>
            <Card style={{width: 150, cursor: 'pointer'}} border={"light"}>
                <Image width={150} height={150} src={process.env.REACT_APP_API_URL + device.img}/>
                <div className="text-black-50 mt-1 d-flex justify-content-between align-items-center">
                    <div>Samsung...</div>
                    <div className="d-flex align-items-center">
                        <div>{device.rating}</div>
                        <Image width={18} height={18} src={star}/>
                    </div>
                </div>
                <div>{device.name}</div>
            </Card>
        </Col>
    );
};

export default DeviceItem;
import React, {useContext} from 'react';
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import {Row} from "react-bootstrap";
import DeviceItem from "./DeviceItem";

const DeviceList = observer(() => {
    const {device} = useContext(Context)

    return (
        <Row className="d-flex">
            {device.devices.map(device =>
                <DeviceItem key={device.id} device={device}/>
            )}
        </Row>
    );
```

```
});

export default DeviceList;
import React, {useContext} from 'react';
import {Context} from "../index";
import Navbar from "react-bootstrap/Navbar";
import Nav from "react-bootstrap/Nav";
import {NavLink} from "react-router-dom";
import {ADMIN_ROUTE, LOGIN_ROUTE, SHOP_ROUTE} from "../utils/consts";
import {Button} from "react-bootstrap";
import {observer} from "mobx-react-lite";
import Container from "react-bootstrap/Container";
import {useHistory} from 'react-router-dom'
const NavBar = observer(() => {
    const {user} = useContext(Context)
    const history = useHistory()

    const logOut = () => {
        user.setUser({})
        user.setIsAuth(false)
    }

    return (
        <Navbar bg="dark" variant="dark">
            <Container>
                <NavLink style={{color:'white'}} to={SHOP_ROUTE}>PC_SHOP </NavLink>
                {user.isAuth ?
                    <Nav className="ml-auto" style={{color: 'white'}}>
                        <Button
                            variant={"outline-light"}
                            onClick={() => history.push(ADMIN_ROUTE)}
                        >
                            Адмін панель
                        </Button>
                        <Button
                            variant={"outline-light"}
                            onClick={() => logOut()}
                            className="ml-2"
                        >
                            Выйти
                        </Button>
                    </Nav>
                    :
                    <Nav className="ml-auto" style={{color: 'white'}}>
                        <Button variant={"outline-light"} onClick={() =>
history.push(LOGIN_ROUTE)}>Авторизація</Button>
                    </Nav>
                }
            </Container>
        </Navbar>

    );
});

export default NavBar;
import React, {useContext} from 'react';
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import {Pagination} from "react-bootstrap";

const Pages = observer(() => {
    const {device} = useContext(Context)
    const pageCount = Math.ceil(device.totalCount / device.limit)
```

```
    const pages = []

    for (let i = 0; i < pageCount; i++) {
        pages.push(i + 1)
    }

    return (
        <Pagination className="mt-3">
            {pages.map(page =>
                <Pagination.Item
                    key={page}
                    active={device.page === page}
                    onClick={() => device.setPage(page)}
                >
                    {page}
                </Pagination.Item>
            )}
        </Pagination>
    );
});

export default Pages;
import React, {useContext} from 'react';
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import Col from "react-bootstrap/Col";
import ListGroup from "react-bootstrap/ListGroup";

const TypeBar = observer(() => {
    const {device} = useContext(Context)
    return (
        <ListGroup>
            {device.types.map(type =>
                <ListGroup.Item
                    style={{cursor: 'pointer'}}
                    active={type.id === device.selectedType.id}
                    onClick={() => device.setSelectedType(type)}
                    key={type.id}
                >
                    {type.name}
                </ListGroup.Item>
            )}
        </ListGroup>
    );
});

export default TypeBar;
import {$authHost, $host} from "./index";
import jwt  decode from "jwt-decode";

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

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

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

```
}

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

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

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

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

const $host = axios.create({
    baseURL: process.env.REACT_APP_API_URL
})

const $authHost = axios.create({
    baseURL: process.env.REACT_APP_API_URL
})

const authInterceptor = config => {
    config.headers.authorization = `Bearer ${localStorage.getItem('token')}`
    return config
}

$authHost.interceptors.request.use(authInterceptor)

export {
    $host,
    $authHost
}
import {$authHost, $host} from "./index";
import jwt  decode from "jwt-decode";

export const registration = async (email, password) => {
    const {data} = await $host.post('api/user/registration', {email, password, role: 'ADMIN'})
    localStorage.setItem('token', data.token)
    return jwt_decode(data.token)
}

export const login = async (email, password) => {
    const {data} = await $host.post('api/user/login', {email, password})
    localStorage.setItem('token', data.token)
    return jwt_decode(data.token)
}

export const check = async () => {
    const {data} = await $authHost.get('api/user/auth' )
    localStorage.setItem('token', data.token)
```

```
    return jwt_decode(data.token)
}
import React, {useState} from 'react';
import {Button, Container} from "react-bootstrap";
import CreateBrand from "../components/modals/CreateBrand";
import CreateDevice from "../components/modals/CreateDevice";
import CreateType from "../components/modals/CreateType";

const Admin = () => {
    const [brandVisible, setBrandVisible] = useState(false)
    const [typeVisible, setTypeVisible] = useState(false)
    const [deviceVisible, setDeviceVisible] = useState(false)

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

export default Admin;
import React, {useContext, useState} from 'react';
import {Container, Form} from "react-bootstrap";
import Card from "react-bootstrap/Card";
import Button from "react-bootstrap/Button";
import Row from "react-bootstrap/Row";
import {NavLink, useLocation, useHistory} from "react-router-dom";
import {LOGIN_ROUTE, REGISTRATION_ROUTE, SHOP_ROUTE} from "../utils/consts";
import {login, registration} from "../http/userAPI";
import {observer} from "mobx-react-lite";
import {Context} from "../index";

const Auth = observer(() => {
    const {user} = useContext(Context)
    const location = useLocation()
    const history = useHistory()
    const isLogin = location.pathname === LOGIN_ROUTE
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')
```

```
    const click = async () => {
        try {
            let data;
            if (isLogin) {
                data = await login(email, password);
            } else {
                data = await registration(email, password);
            }
            user.setUser(user)
            user.setIsAuth(true)
            history.push(SHOP_ROUTE)
        } catch (e) {
            alert(e.response.data.message)
        }

    }

    return (
        <Container
            className="d-flex justify-content-center align-items-center"
            style={{height: window.innerHeight - 54}}
        >
            <Card style={{width: 600}} className="p-5">
                <h2 className="m-auto">{isLogin ? 'Авторизація' : "Реєстрація"}</h2>
                <Form className="d-flex flex-column">
                    <Form.Control
                        className="mt-3"
                        placeholder="Введіть ваш email..."
                        value={email}
                        onChange={e => setEmail(e.target.value)}
                    />
                    <Form.Control
                        className="mt-3"
                        placeholder="Введіть ваш пароль..."
                        value={password}
                        onChange={e => setPassword(e.target.value)}
                        type="password"
                    />
                    <Row className="d-flex justify-content-between mt-3 pl-3 pr-3">
                        {isLogin ?
                            <div>
                                Нет аккаунта? <NavLink to={REGISTRATION_ROUTE}>Зареєструйтеся!</NavLink>
                            </div>
                            :
                            <div>
                                Есть аккаунт? <NavLink to={LOGIN_ROUTE}>Увійдіть!</NavLink>
                            </div>
                        }
                        <Button
                            variant={"outline-success"}
                            onClick={click}
                        >
                            {isLogin ? 'Увійти' : 'Реєстрація'}
                        </Button>
                    </Row>

                </Form>
            </Card>
        </Container>
    );
});

export default Auth;
```

```
import React from 'react';

const Basket = () => {
    return (
        <div>
            basket
        </div>
    );
};

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

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

    return (
        <Container className="mt-3">
            <Row>
                <Col md={4}>
                    <Image width={300} height={300} src={process.env.REACT_APP_API_URL + device.img}/>
                </Col>
                <Col md={4}>
                    <Row className="d-flex flex-column align-items-center">
                        <h2>{device.name}</h2>
                        <div
                            className="d-flex align-items-center justify-content-center"
                            style={{background: `url(${bigStar}) no-repeat center center`, width:240, height: 240,
backgroundSize: 'cover', fontSize:64}}
                        >
                            {device.rating}
                        </div>
                    </Row>
                </Col>
                <Col md={4}>
                    <Card
                        className="d-flex flex-column align-items-center justify-content-around"
                        style={{width: 300, height: 300, fontSize: 32, border: '5px solid lightgray'}}
                    >
                        <h3>От: {device.price} руб.</h3>
                        <Button variant={"outline-dark"}>Додати в корзину</Button>
                    </Card>
                </Col>
            </Row>
            <Row className="d-flex flex-column m-3">
                <h1>Характеристики</h1>
                {device.info.map((info, index) =>
                    <Row key={info.id} style={{background: index % 2 === 0 ? 'lightgray' : 'transparent', padding: 10}}>
                        {info.title}: {info.description}
                    </Row>
                )}
            </Row>
        </Container>
    );
};
```

```
export default DevicePage;
import React, {useContext, useEffect} from 'react';
import {Container} from "react-bootstrap";
import Row from "react-bootstrap/Row";
import Col from "react-bootstrap/Col";
import TypeBar from "../components/TypeBar";
import BrandBar from "../components/BrandBar";
import DeviceList from "../components/DeviceList";
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import {fetchBrands, fetchDevices, fetchTypes} from "../http/deviceAPI";
import Pages from "../components/Pages";

const Shop = observer(() => {
    const {device} = useContext(Context)

    useEffect(() => {
        fetchTypes().then(data => device.setTypes(data))
        fetchBrands().then(data => device.setBrands(data))
        fetchDevices(null, null, 1, 2).then(data => {
            device.setDevices(data.rows)
            device.setTotalCount(data.count)
        })
    }, [])

    useEffect(() => {
        fetchDevices(device.selectedType.id, device.selectedBrand.id, device.page, 2).then(data => {
            device.setDevices(data.rows)
            device.setTotalCount(data.count)
        })
    }, [device.page, device.selectedType, device.selectedBrand,])

    return (
        <Container>
            <Row className="mt-2">
                <Col md={3}>
                    <TypeBar/>
                </Col>
                <Col md={9}>
                    <BrandBar/>
                    <DeviceList/>
                    <Pages/>
                </Col>
            </Row>
        </Container>
    );
});

export default Shop;
import {makeAutoObservable} from "mobx";

export default class DeviceStore {
    constructor() {
        this._types = []
        this._brands = []
        this._devices = []
        this._selectedType = {}
        this._selectedBrand = {}
        this._page = 1
        this._totalCount = 0
        this._limit = 3
        makeAutoObservable(this)
```

```
    }

    setTypes(types) {
        this._types = types
    }
    setBrands(brands) {
        this._brands = brands
    }
    setDevices(devices) {
        this.  devices = devices
    }

    setSelectedType(type) {
        this.setPage(1)
        this._selectedType = type
    }
    setSelectedBrand(brand) {
        this.setPage(1)
        this._selectedBrand = brand
    }
    setPage(page) {
        this._page = page
    }
    setTotalCount(count) {
        this.  totalCount = count
    }

    get types() {
        return this.  types
    }
    get brands() {
        return this._brands
    }
    get devices() {
        return this._devices
    }
    get selectedType() {
        return this._selectedType
    }
    get selectedBrand() {
        return this._selectedBrand
    }
    get totalCount() {
        return this.  totalCount
    }
    get page() {
        return this._page
    }
    get limit() {
        return this._limit
    }
}
import {makeAutoObservable} from "mobx";

export default class UserStore {
    constructor() {
        this._isAuth = false
        this.  user = {}
        makeAutoObservable(this)
    }

    setIsAuth(bool) {
        this._isAuth = bool
```

```
    }
    setUser(user) {
        this._user = user
    }

    get isAuth() {
        return this._isAuth
    }
    get user() {
        return this.  user
    }
}
export const ADMIN_ROUTE = '/admin'
export const LOGIN  ROUTE = '/login'
export const REGISTRATION_ROUTE = '/registration'
export const SHOP_ROUTE = '/'
export const BASKET_ROUTE = '/basket'
export const DEVICE  ROUTE = '/device'
import React, {useContext, useEffect, useState} from 'react';
import {BrowserRouter} from "react-router-dom";
import AppRouter from "./components/AppRouter";
import NavBar from "./components/NavBar";
import {observer} from "mobx-react-lite";
import {Context} from "./index";
import {check} from "./http/userAPI";
import {Spinner} from "react-bootstrap";

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

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

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

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

export default App;
import React, {createContext} from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import UserStore from "./store/UserStore";
import DeviceStore from "./store/DeviceStore";

export const Context = createContext(null)

ReactDOM.render(
    <Context.Provider value={{
        user: new UserStore(),


 device: new DeviceStore(),
```

```
    }}>
      <App />
    </Context.Provider>,
  document.getElementById('root')
);
import Admin from "./pages/Admin";
import {ADMIN_ROUTE, BASKET_ROUTE, DEVICE_ROUTE, LOGIN_ROUTE, REGISTRATION_ROUTE,
SHOP_ROUTE} from "./utils/consts";
import Basket from "./pages/Basket";
import Shop from "./pages/Shop";
import Auth from "./pages/Auth";
import DevicePage from "./pages/DevicePage";

export const authRoutes = [
    {
      path: ADMIN_ROUTE,
      Component: Admin
    },
    {
      path: BASKET_ROUTE,
      Component: Basket
    },
]

export const publicRoutes = [
    {
      path: SHOP_ROUTE,
      Component: Shop
    },
    {
      path: LOGIN_ROUTE,
      Component: Auth
    },
    {
      path: REGISTRATION_ROUTE,
      Component: Auth
    },
    {
      path: DEVICE_ROUTE + '/:id',
      Component: DevicePage
    },
]
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.11.9",
    "@testing-library/react": "^11.2.3",
    "@testing-library/user-event": "^12.6.2",
    "axios": "^0.21.1",
    "bootstrap": "^4.6.0",
    "jwt-decode": "^3.1.2",
    "mobx": "^6.0.5",
    "mobx-react-lite": "^3.1.7",
    "react": "^17.0.1",
    "react-bootstrap": "^1.4.3",
    "react-dom": "^17.0.1",
    "react-router-dom": "^5.2.0",
    "react-scripts": "4.0.1",
    "web-vitals": "^0.2.4"
  },
```

```
"scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.11.9",
    "@testing-library/react": "^11.2.3",
    "@testing-library/user-event": "^12.6.2",
    "axios": "^0.21.1",
    "bootstrap": "^4.6.0",
    "jwt-decode": "^3.1.2",
    "mobx": "^6.0.5",
    "mobx-react-lite": "^3.1.7",
    "react": "^17.0.1",
    "react-bootstrap": "^1.4.3",
    "react-dom": "^17.0.1",
    "react-router-dom": "^5.2.0",
    "react-scripts": "4.0.1",
    "web-vitals": "^0.2.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
```

```
    "last 1 chrome version",


     "last 1 chrome version",
     "last 1 firefox version",
     "last 1 safari version"
   ]
  }
}
const {Brand} = require('../models/models')
const ApiError = require('../error/ApiError');

class BrandController {
    async create(req, res) {
        const {name} = req.body
        const brand = await Brand.create({name})
        return res.json(brand)
    }

    async getAll(req, res) {
        const brands = await Brand.findAll()
        return res.json(brands)
    }

}

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

class DeviceController {
    async create(req, res, next) {
        try {
            let {name, price, brandId, typeId, info} = req.body
            const {img} = req.files
            let fileName = uuid.v4() + ".jpg"
            img.mv(path.resolve(__dirname, '..', 'static', fileName))
            const device = await Device.create({name, price, brandId, typeId, img: fileName});

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

            return res.json(device)
        } catch (e) {
            next(ApiError.badRequest(e.message))
        }

    }

    async getAll(req, res) {
        let {brandId, typeId, limit, page} = req.query
        page = page || 1
        limit = limit || 9
        let offset = page * limit - limit
        let devices;
```

```
  if (!brandId && !typeId) {

        devices = await Device.findAndCountAll({limit, offset})
      }
      if (brandId && !typeId) {
        devices = await Device.findAndCountAll({where:{brandId}, limit, offset})
      }
      if (!brandId && typeId) {
        devices = await Device.findAndCountAll({where:{typeId}, limit, offset})
      }
      if (brandId && typeId) {
        devices = await Device.findAndCountAll({where:{typeId, brandId}, limit, offset})
      }
      return res.json(devices)
    }

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

module.exports = new DeviceController()
const {Type} = require('../models/models')
const ApiError = require('../error/ApiError');

class TypeController {
    async create(req, res) {
      const {name} = req.body
      const type = await Type.create({name})
      return res.json(type)
    }

    async getAll(req, res) {
      const types = await Type.findAll()
      return res.json(types)
    }

}

module.exports = new TypeController()
const ApiError = require('../error/ApiError');
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')
const {User, Basket} = require('../models/models')

const generateJwt = (id, email, role) => {
    return jwt.sign(
        {id, email, role},
        process.env.SECRET_KEY,
        {expiresIn: '24h'}
    )
}

class UserController {
    async registration(req, res, next) {
        const {email, password, role} = req.body
        if (!email || !password) {
```

```
        }
        const candidate = await User.findOne({where: {email}})
        if (candidate) {
            return next(ApiError.badRequest(Користувач з таким email вже існує))
        }
        const hashPassword = await bcrypt.hash(password, 5)
        const user = await User.create({email, role, password: hashPassword})
        const basket = await Basket.create({userId: user.id})
        const token = generateJwt(user.id, user.email, user.role)
        return res.json({token})
    }

    async login(req, res, next) {
        const {email, password} = req.body
        const user = await User.findOne({where: {email}})
        if (!user) {
            return next(ApiError.internal('Користувач не знайдений'))
        }
        let comparePassword = bcrypt.compareSync(password, user.password)
        if (!comparePassword) {
            return next(ApiError.internal('Невірний пароль'))
        }
        const token = generateJwt(user.id, user.email, user.role)
        return res.json({token})
    }

    async check(req, res, next) {
        const token = generateJwt(req.user.id, req.user.email, req.user.role)
        return res.json({token})
    }
}

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

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

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

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

module.exports = ApiError
const jwt = require('jsonwebtoken')

module.exports = function (req, res, next) {
    if (req.method === "OPTIONS") {
        next()
    }
    try {
        const token = req.headers.authorization.split(' ')[1] // Bearer asfasnfkajsfnjk
```

```
            return res.status(401).json({message: "Не авторизовано"})
        }
        const decoded = jwt.verify(token, process.env.SECRET_KEY)
        req.user = decoded
        next()
    } catch (e) {
        res.status(401).json({message: "Не авторизовано"})
    }
};
const jwt = require('jsonwebtoken')

module.exports = function(role) {
    return function (req, res, next) {
        if (req.method === "OPTIONS") {
            next()
        }
        try {
            const token = req.headers.authorization.split(' ')[1] // Bearer asfasnfkajsfnjk
            if (!token) {
                return res.status(401).json({message: "Не авторизовано"})
            }
            const decoded = jwt.verify(token, process.env.SECRET_KEY)
            if (decoded.role !== role) {
                return res.status(403).json({message: "Немає доступу"})
            }
            req.user = decoded;
            next()
        } catch (e) {
            res.status(401).json({message: "Не авторизовано"})
        }
    };
}


const ApiError = require('../error/ApiError');

module.exports = function (err, req, res, next) {
    if (err instanceof ApiError) {
        return res.status(err.status).json({message: err.message})
    }
    return res.status(500).json({message: "Помилка!"})
}
const Router = require('express')
const router = new Router()
const brandController = require('../controllers/brandController')

router.post('/', brandController.create)
router.get('/', brandController.getAll)

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

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

module.exports = router
const Router = require('express')
const router = new Router()
const deviceRouter = require('./deviceRouter')
```

```
const userRouter = require('./userRouter')
    const brandRouter = require('./brandRouter')
    const typeRouter = require('./typeRouter')

    router.use('/user', userRouter)
    router.use('/type', typeRouter)
    router.use('/brand', brandRouter)
    router.use('/device', deviceRouter)

    module.exports = router
    const Router = require('express')
    const router = new Router()
    const typeController = require('../controllers/typeController')
    const checkRole = require('../middleware/checkRoleMiddleware')

    router.post('/', checkRole('ADMIN'), typeController.create)
    router.get('/', typeController.getAll)

    module.exports = router
    const Router = require('express')
    const router = new Router()
    const userController = require('../controllers/userController')
    const authMiddleware = require('../middleware/authMiddleware')

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

    module.exports = router
    const {Sequelize} = require('sequelize')

    module.exports = new Sequelize(
      process.env.DB_NAME, // Название БД
      process.env.DB_USER, // Пользователь
      process.env.DB_PASSWORD, // ПАРОЛЬ
      {
        dialect: 'postgres',
        host: process.env.DB_HOST,
        port: process.env.DB_PORT
      }
    )
    require('dotenv').config()
    const express = require('express')
    const sequelize = require('./db')
    const models = require('./models/models')
    const cors = require('cors')
    const fileUpload = require('express-fileupload')
    const router = require('./routes/index')
    const errorHandler = require('./middleware/ErrorHandlingMiddleware')
    const path = require('path')

    const PORT = process.env.PORT || 5000

    const app = express()
    app.use(cors())
    app.use(express.json())
    app.use(express.static(path.resolve(__dirname, 'static')))
    app.use(fileUpload({}))
    app.use('/api', router)
    const start = async () => {
      try {
        await sequelize.authenticate()
        await sequelize.sync()    app.listen(PORT, () => console.log(`Server started on port ${PORT}`))
```

```
    } catch (e) {
        console.log(e)
    }
}

start()
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Кваліфікаційна_робота_Ігнатенко_С.І.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна_робота_Ігнатенко_С.І.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
pc-shop.zip	Архів. Містить коди програми.
Презентація	
Презентація_Ігнатенко.ppt	Презентація кваліфікаційної роботи.

ВІДГУК

на кваліфікаційну роботу бакалавра
на тему:
" Розробка веб-магазину з продажу комп'ютерної периферії за допомогою мови JavaScript "
студента групи 122-18-2 Ігнатенка Сергія Івановича

Розроблена в кваліфікаційній роботі програма призначена для популяризації та продажу товару для підприємства з продажу одягу.

Як інструмент для проектування і реалізації було використане середовище розробки WebStorm за допомогою мови програмування JavaScript.

Практична значимість створення даної інформаційної системи полягає в можливості демонстрації потенційному користувачеві асортименту інтернет-магазину.

Працездатність представленої інформаційної системи підтверджена налагоджувальними випробуваннями та тестуванням програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Тема кваліфікаційної роботи безпосередньо пов'язана з об'єктом діяльності бакалавра за спеціальністю 122 Комп’ютерні науки.

Оформлення пояснювальної записки до роботи виконано відповідно до стандартів на програмну документацію.

Кваліфікаційну роботу виконано самостійно та заслуговує оцінки 74 бала «добре», а студент Ігнатенко Сергій Іванович заслуговує присвоєння йому кваліфікації бакалавра з комп’ютерних наук.

Керівник кваліфікаційної роботи
доцент каф. ПЗКС, к.т.н. **С.Д. Приходченко**

РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра
на тему:
" Розробка веб-магазину з продажу комп'ютерної периферії за допомогою мови JavaScript "
студента групи 122-18-2 Ігнатенка Сергія Івановича

Кваліфікаційну роботу на тему «Розробка веб-магазину з продажу комп'ютерної периферії за допомогою мови JavaScript» виконано в повному обсязі, відповідно до технічного завдання.

Мета кваліфікаційної роботи: метою дипломного проекту є аналіз доступних технологій і методів розробки для створення інформаційної системи для продажу товару в інтернеті у середовищі WebStorm.

У пояснювальній записці розглянуто необхідність створення і сфера застосування розробленої, виконано постановку завдання, опис вхідних і вихідних даних, розроблено інформаційне забезпечення системи, наведені загальні відомості про роботу, визначені джерела, використані при розробці.

Вважаю завдання і зміст кваліфікаційної роботи відповідним для перевірки ступеня підготовленості бакалавр за напрямом 122 Комп’ютерні науки.

Список літератури, наведений в роботі, налічує 22 джерела, що свідчить про вміння автора працювати з літературою та іншими джерелами інформації. Якість оформлення кваліфікаційної роботи можна визнати «добре», з незначними недоліками.

Кваліфікаційну роботу виконано самостійно та заслуговує оцінки «добре», а студент Ігнатенко Сергій Іванович заслуговує присвоєння йому кваліфікації бакалавра з комп’ютерних наук.

Рецензент кваліфікаційної роботи