

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ МАЙСТЕРНІ МУЗИЧНИХ
ІНСТРУМЕНТІВ

DEVELOPMENT OF WEB APPLICATION FOR A WORKSHOP OF
OF MUSICAL INSTRUMENTS

Виконав: студент(ка) 4 курсу, групи КНТ-
121сп

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КУЛЬЧЕНКО Я.Е.

(прізвище та ініціали)

Керівник КАПЛІЄНКО Т.І.

(прізвище та ініціали)

Рецензент ШИТІКОВА О.В.

(прізвище та ініціали)

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

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

ЗАТВЕРДЖУЮ

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

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КУЛЬЧЕНКА Ярослава Едуардовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка вебзастосунку для майстерні музичних інструментів.

Development of a Web Application for a Workshop of Musical Instruments

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

затверджені наказом закладу вищої освіти від “24” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Характеристика об'єкта проєктування.

2. Вибір структури системи та його обґрунтування. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво системного адміністратора. 6. Додатки

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6 Основна частина	КАПЛІЄНКО Т.І., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання “ 15 ” квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	3 тиждень	Розділ 3
5	Розробка програми.	4 тиждень	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Ярослав КУЛЬЧЕНКО
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна КАПЛІЄНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи
бакалавра: 115 с., 4 табл., 33 рис., 2 дод., 13 джерел.

ВЕБЗАСТОСУНОК, КАСТОМІЗАЦІЯ ІНСТРУМЕНТІВ, РЕМОНТНІ
ПОСЛУГИ, АВТОМАТИЗАЦІЯ, ЕЛЕКТРОННА КОМЕРЦІЯ.

Об'єкт дослідження – вебзастосунки для майстерень музичних
інструментів.

Мета роботи – проектування та створення вебзастосунку, який
дозволяє обирати музичні інструменти, профілактичні послуги для
інструмента та кастомізацію зручно для будь-якого користувача.

Методи дослідження: аналіз предметної області, розробка методів,
моделей, бази даних, алгоритмів. Вдосконалення методів, моделей, бази даних
та алгоритмів. Проектування та розробка архітектури вебзастосунка, розробка
основних та додаткових елементів, програмування основи вебзастосунку та
бази даних, тестування програмного забезпечення та експериментальне
дослідження вебзастосунку.

Результати: створено вебзастосунок майстерні музичних інструментів,
в якому є функціонал, який дозволяє купувати музичні інструменти, замовляти
послуги обробки та кастомізації свого інструмента звичайному користувачу в
інтернеті.

Висновки: розроблений вебзастосунок адаптує, робить доступним та
прискорює процес зв'язку користувача з майстернею, взаємодії між ними,
придбання товарів та послуг.

ABSTRACT

Explanatory note to the bachelor's degree qualification work: 115 pages, 4 tables, 33 figures, 2 appendices, 13 sources.

WEB APPLICATION, INSTRUMENT CUSTOMIZATION, REPAIR SERVICES, AUTOMATION, E-COMMERCE.

Research Object– the object of the research is web applications for musical instrument workshops.

Purpose of the work – the purpose of the work is to design and create a web application that allows users to choose musical instruments, preventive services for the instrument, and customization options in a user-friendly manner.

Research methods: analysis of the subject area, development of methods, models, database, and algorithms. Improvement of methods, models, database, and algorithms. Designing and developing the architecture of the web application, development of core and additional elements, programming the core of the web application and the database, software testing and experimental study of the web application.

Results: a web application for a musical instrument workshop has been created, featuring functionalities that allow ordinary users to purchase musical instruments, order processing services, and customize their instruments online.

Conclusions: the developed web application adapts, makes accessible, and accelerates the process of communication between the user and the workshop, facilitating interactions between them, and the purchase of goods and services.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Характеристика об'єкта проектування	10
1.1 Поняття вебзастосунка для майстерні музичних інструментів.....	10
1.2 Функціональні можливості вебзастосунку.....	11
1.3 Аналіз аналогів.....	12
2 Вибір структури системи та його обґрунтування	15
2.1 Обґрунтування вибору середовища для серверної частини	15
2.2 Обґрунтування вибору середовища для клієнтської частини	17
2.3 Обґрунтування вибору бази даних.....	21
3 Основні рішення щодо реалізації компонентів системи.....	25
3.1 Алгоритм роботи головної сторінки.....	25
3.2 Алгоритм роботи сторінки бронювання.....	26
3.3 Алгоритм роботи сторінки реєстрації та сторінки авторизації.....	26
3.4 Алгоритм роботи сторінки адміністратора.....	28
4 Керівництво програміста.....	30
4.1 Призначення і умови використання вебзастосунку.....	30
4.2 Характеристики продукту.....	30
4.3 Структура бази даних.....	31
4.4 Структура застосунку.....	34
5 Керівництво системного адміністратора.....	35
5.1 Розгортання сайту.....	35
5.1.1 Серверна частина.....	35
5.1.2 Клієнтська частина.....	36
5.2 Адміністраторська сторінка.....	37
6 Керівництво користувача.....	40
6.1 Призначення сайту.....	40
6.2 Заовлення інструментів та авторизація.....	40

Висновки	45
Перелік джерел посилання	46
Додаток А Текст програми.....	47
Додаток Б Слайди презентації	108

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

DOM – Document Object Model;

БД – база даних;

ВСТУП

В сучасному світі, де постійно лунає музика й де кожного дня утворюються нові музичні групи та сольні музиканти, автоматизація процесів майстерні музичних інструментів є важливою складовою ефективного функціонування музичного бізнесу та підвищення задоволеності клієнтів. Однією з таких інновацій є розробка вебзастосунків майстерень музичних інструментів та музичних магазинів, які надають можливість користувачам за короткий відрізок часу замовити музичний інструмент чи послугу майстра.

Ця дипломна робота присвячена розробці вебзастосунку майстерні музичних інструментів. Мета роботи - створення зручної та надійної системи, яка полегшує процес замовлення музичних інструментів чи послуг майстра.

1 ХАРАКТЕРИСТИКА ОБ'ЄКТА ПРОЄКТУВАННЯ

1.1 Поняття вебзастосунку майстерні музичних інструментів

Вебзастосунок для майстерні музичних інструментів – це застосунок, який в реальному часі дозволяє клієнту (юридичній або фізичній особі) обрати музичний інструмент або послугу майстра. Якщо інструмент та послуги з оплатою, то клієнт сплачує їх за допомогою банківської картки або накладеною платежем на пошті.

Замовлення – це документ, яким реєструється бажання покупця придбати певний товар[1].

Вебзастосунки майстерень музичних інструментів існують ще з минулого століття. За цей час популярність таких вебзастосунків і не тільки таких, а й всієї мережі інтернет постійно зростала. Наразі майже кожна майстерня музичних інструментів замовляє собі вебзастосунки, або створює акаунти в соціальних мережах, а ті, хто бажають скористатися їхніми послугами можуть скористатися вебзастосунком.

Електронна комерція (е-комерція) – це процес виробництва, розподілу, маркетингу, продажу або доставки товарів і послуг за допомогою електронних засобів [2]. Вебзастосунок майстерні музичних інструментів також є прикладом електронної комерції.

При замовленні інструмента або послуги клієнтом здійснюється електронний правочин.

Електронний правочин - дія особи, спрямована на набуття, зміну або припинення цивільних прав та обов'язків, здійснена з використанням інформаційно-телекомунікаційних систем (п. 7 ч. 1 ст. 3 Закону України «Про електронну комерцію») [3].

Електронний договір – це домовленість двох або більше сторін, спрямована на встановлення, зміну або припинення цивільних прав та обов'язків і оформлена в електронній формі (пункт 5 частини 1 статті 3 Закону України «Про електронну комерцію») [4].

Електронна торгівля відноситься до «електронної комерції», що визначена у пп. 1 год. 1 ст. 3 Закону № 675: відносини, спрямовані на отримання прибутку, що виникають при скоєнні угод щодо придбання, зміни або припинення цивільних прав та обов'язків, здійснені дистанційно з використанням інформаційно-телекомунікаційних систем, унаслідок чого в учасників таких відносин виникають права та обов'язки майнового характеру [5].

Коли клієнт замовляє музичний інструмент чи послугу майстра, відбувається онлайн-покупка. Прикладом такого вебзастосунку є muztorg.ua, український вебзастосунок для продажу музичних інструментів та обслуговування їх. За допомогою подібного вебзастосунку користувач легко і швидко зможе замовити музичні інструменти та периферію, знаходячись при цьому де завгодно.

1.2 Функціональні можливості вебзастосунку

При плануванні вебзастосунку основною його метою було поставлено надання користувачу можливість обрати та придбати музичний інструмент, замовлення послуг майстрів, як для профілактики свого інструмента, так і для кастомізації. Головними критеріями при розробці були швидкість роботи, зручність для користувача, та функціональність. Для проєкта були поставлені для виконання та реалізовані можливості:

- інтуїтивно зрозуміла навігація ;
- вибір та придбання інструменту або послуги майстра;
- можливість ставити оцінку товару;

Володарі музичних інструментів, музиканти та новачки, які хочуть придбати собі перший інструмент скориставшись цим вебзастосунком зможуть витратити менше часу для задоволення своїх потреб, аніж як би вони звернулися до фізичної майстерні та магазину. Функціонал цього

вебзастосунку хоч і невеликий, але дозволить придбати інструмент або замовити послугу більш комфортніше та швидше.

1.3 Аналіз аналогічних вебзастосунків

Аналізуючи три достатньо великих та популярних вебзастосунків для майстерень музичних інструментів в Україні, виділяються наступні аспекти та відмінності.

Аналіз вебзастосунку asropolis.ua (рис 1.1) [6]:

- великий асортимент товарів як інших брендів, так і свого виробництва;
- зручна навігація;
- проведення різноманітних акцій, розпродажів;
- відповідність функціоналу мобільного застосунку та повноцінного десктопного вебзастосунку;
- привабливий дизайн.

Аналіз вебзастосунку tadyka-guitars.com (рис 1.2) [7]:

- спеціалізація на ремонтних роботах та відновлення старих інструментів;
- наявність новин та блогу майстерні;
- програма лояльності для військових та постійних клієнтів;
- цілодобовий зв'язок з службою підтримки;
- широкий спектр послуг.

Аналіз вебзастосунку muztorg.ua (рис 1.3) [8]:

- гарний дизайн інтерфейсу та зручна навігація;
- наявність відділу б/в товарів по низьким цінам;
- зручна фільтрація товарів та послуг;
- наявність відеоблогу з оглядом на інструменти та обладнання;
- наявність власного сервіс-центру.

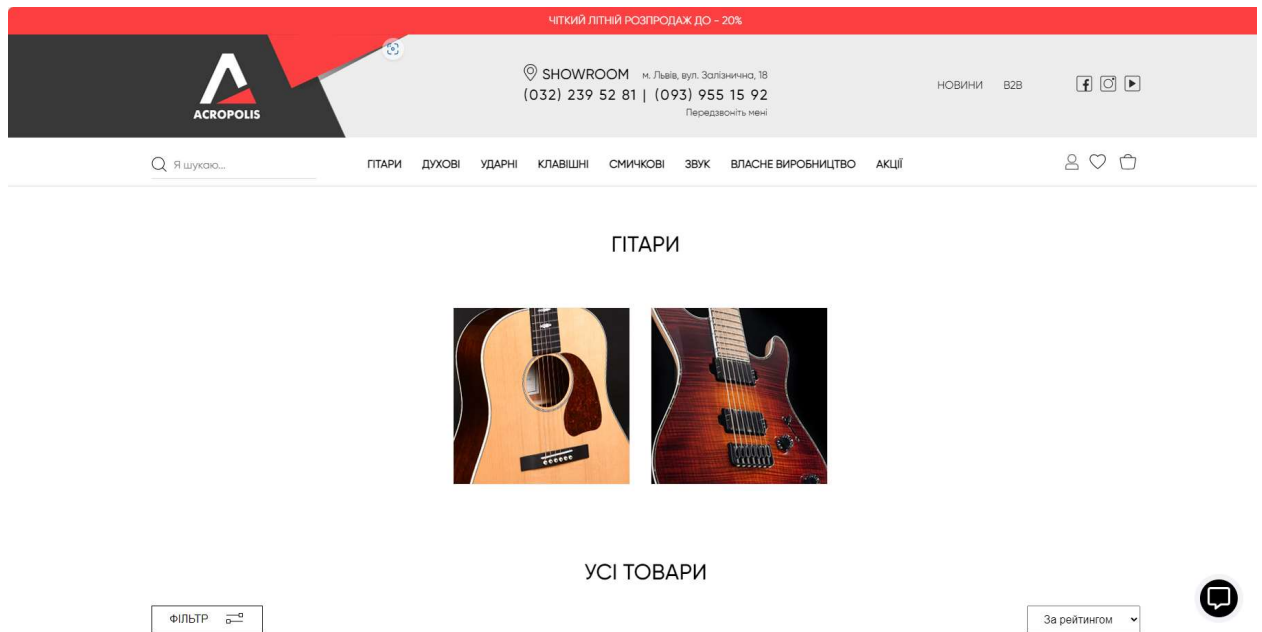


Рисунок 1.1 – Сайт acropolis.ua.

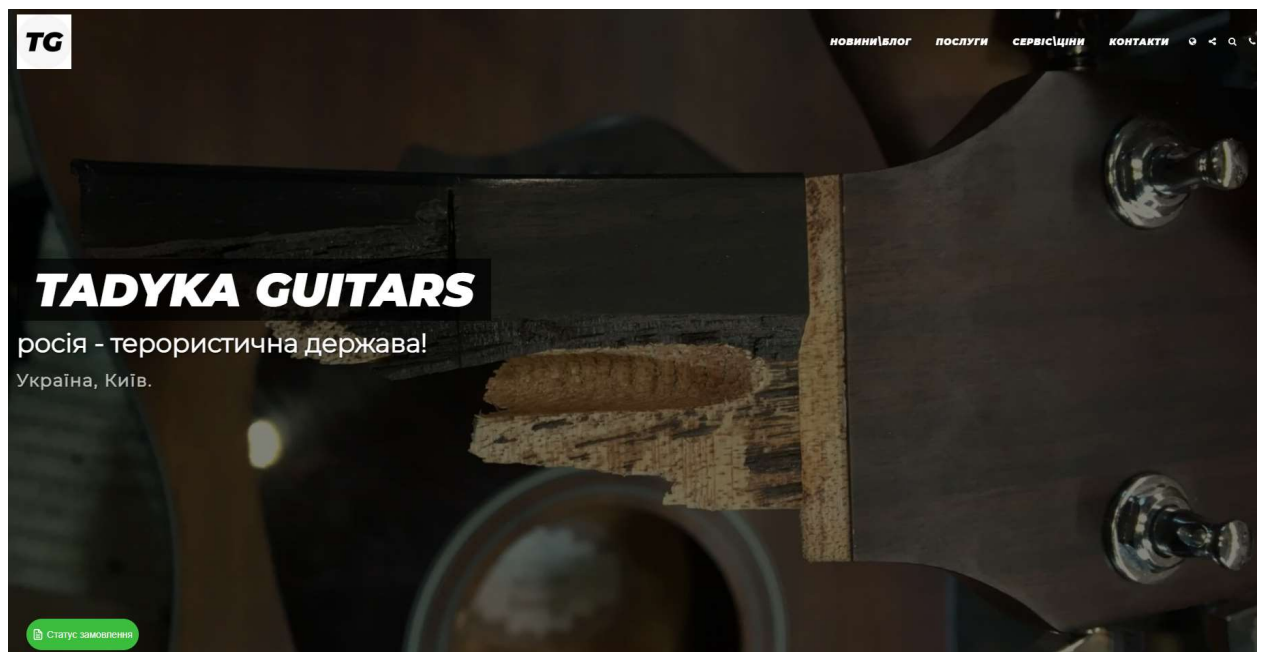


Рисунок 1.2 – Сайт tadyka-guitars.com.

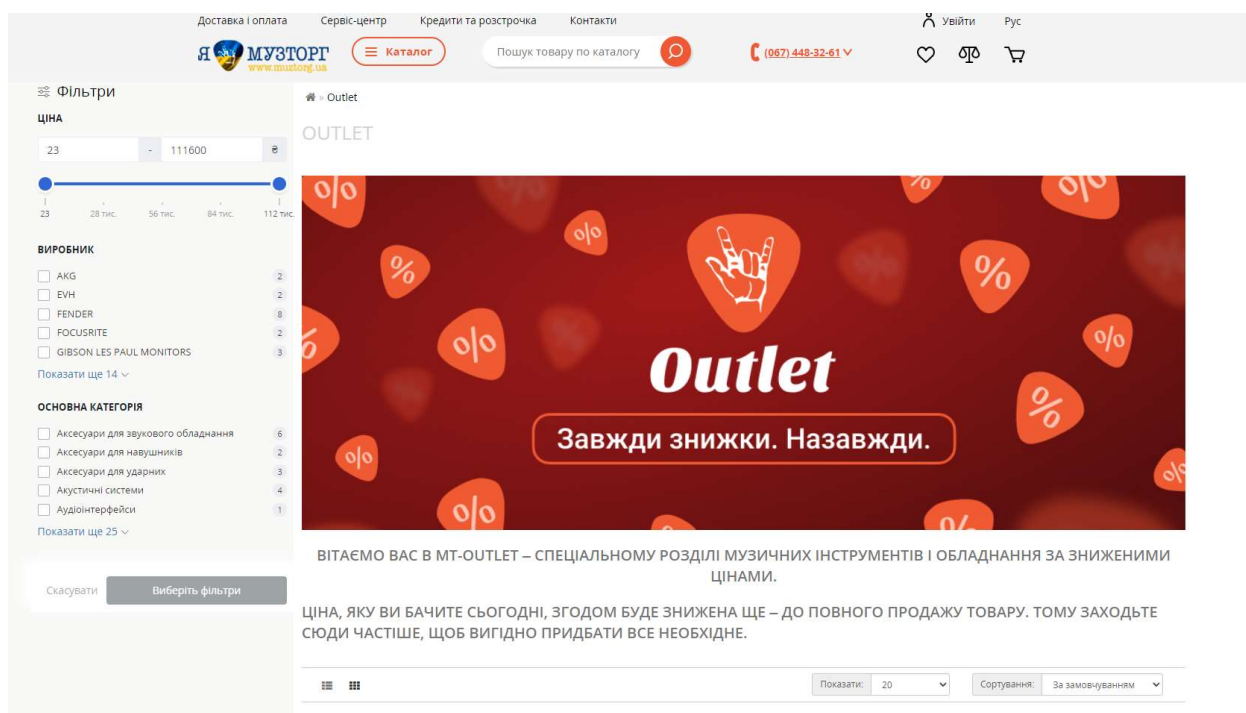


Рисунок 1.3 – muztorg.ua.

Аналізуючи ці вебзастосунки можна прийти до висновку, що всі вони мають гарний дизайн, зручну навігацію, мають великий асортимент послуг з кастомізації та ремонту музичних інструментів. Також вони мають послуги з виготовлення нових власних товарів. Різниця між ними є тільки в наявності асортименту інструментів інших брендів та ще невеликими унікальними функціями, а взагалі вони забезпечують швидкий та надійний сервіс .

1.4 Висновки за розділом

Дослідивши об'єкт проектування та проаналізувавши аналогічні проекти, було виділено набір термінів, які характеризують та підкреслюють функції вебзастосунку для майстерні музичних інструментів. Вебзастосунки для майстерні музичних інструментів в сучасному світі є необхідними та незамінними вебзастосунками для комфортного життя сучасного музиканта. Вебзастосунки для майстерень музичних інструментів дозволяє швидко та зручно придбати інструмент знаходячись де завгодно та в будь-який момент часу,

а також відправити свій інструмент для ремонту або кастомізації. Такі застосунки є необхідним зручним інструментом для сучасного музиканта.

2. ВИБІР СТРУКТУРИ СИСТЕМИ ТА ЙОГО ОБҐРУНТУВАННЯ

2.1 Обґрунтування вибору середовища для клієнтської частини

Для розробки клієнтської частини було використано HTML, CSS, JavaScript та ReactJS.

HTML (HyperText Markup Language) — це стандартна мова розмітки документів, яка використовується для створення вебсторінок. Основна функція HTML полягає у структуризації контенту на вебсторінці за допомогою різних елементів та тегів[9].

CSS (Cascading Style Sheets) – мова стилів, що використовується для форматування та опису зовнішнього вигляду HTML-документів[10].

JavaScript - це мова програмування, яка використовується для створення динамічного та інтерактивного контенту на вебсторінках. Вона є однією з основних технологій веброзробки поряд з HTML і CSS. JavaScript дозволяє змінювати контент вебсторінки без перезавантаження, що забезпечує інтерактивність користувацького інтерфейсу, працює в будь-якому сучасному веббраузері, незалежно від операційної системи, може реагувати на події, такі як кліки миші, введення тексту, завантаження сторінок тощо. JavaScript підтримує принципи ООП, що дозволяє створювати об'єкти з властивостями та методами, підтримує асинхронне програмування, що дозволяє виконувати операції (наприклад, завантаження даних з сервера) у фоновому режимі, не блокуючи основний потік виконання.

React – це потужна JavaScript бібліотека з відкритим вихідним кодом, яка призначена для створення динамічних користувацьких інтерфейсів. Вона допомагає розробникам ефективно вирішувати проблеми, пов'язані з частковим оновленням вмісту вебсторінок, що є особливо важливим при розробці односторінкових застосунків. Завдяки React можна створювати масштабовані вебзастосунки, які працюють з даними, що змінюються в часі, без необхідності повного перезавантаження сторінки. Ключовими перевагами React є його швидкість, простота використання та можливість масштабування.

Ця бібліотека дозволяє розробникам ефективно керувати станом застосунку, забезпечуючи плавне і реактивне оновлення інтерфейсу відповідно до змін у даних [11].

React є однією з найпопулярніших бібліотек для розробки клієнтської частини вебзастосунків, і є низка причин, чому він є чудовим вибором для цієї мети:

- компонентний підхід – React дозволяє розбивати інтерфейс на незалежні, повторно використовувані компоненти. Це спрощує розробку і підтримку застосунків, дозволяючи створювати модульні і легко масштабовані проєкти;
- Virtual DOM – React використовує віртуальний DOM, що підвищує продуктивність застосунків. Віртуальний DOM мінімізує кількість змін в реальному DOM, оновлюючи лише ті частини, які потребують змін, що робить рендеринг швидшим і ефективнішим;
- одностороннє спрямування даних – React дотримується принципу одностороннього потоку даних, що спрощує розуміння і налагодження застосунків. Це також дозволяє краще контролювати потік даних і зменшує ймовірність помилок, пов'язаних з неконтрольованими змінами стану;
- JSX (JavaScript XML) – JSX дозволяє писати структуру інтерфейсу у вигляді HTML-подібного синтаксису прямо в JavaScript. Це підвищує зручність і зрозумілість коду, а також спрощує процес розробки, роблячи його більш інтуїтивно зрозумілим;
- розвинена екосистема – React має велику кількість додаткових бібліотек та інструментів, які допомагають у розробці, тестуванні і деплої застосунків. Це включає Redux для керування станом, React Router для маршрутизації і багато інших;
- легка інтеграція – React можна легко інтегрувати з іншими бібліотеками та фреймворками. Він може бути використаний як в уже існуючих проєктах, так і для створення нових застосунків з нуля;
- підтримка мобільних застосунків – завдяки React Native,

розробники можуть створювати мобільні застосунки для iOS та Android, використовуючи ті ж самі принципи і компоненти, що й у вебзастосунках;

- активна спільнота та підтримка – React має велику і активну спільноту розробників, що забезпечує постійну підтримку і швидке вирішення проблем. Крім того, React підтримується компанією Facebook, що гарантує стабільні оновлення і розвиток бібліотеки;

- SEO-оптимізація – Завдяки можливості серверного рендерингу з такими фреймворками, як Next.js, застосунки на React можуть бути оптимізовані для пошукових систем, що важливо для поліпшення видимості сайту в пошукових результатах;

- гнучкість і масштабованість – React дозволяє створювати як прості, так і складні застосунки, що можуть рости і масштабуватися разом з потребами бізнесу. Завдяки компонентному підходу, кожен частину застосунку можна легко модернізувати і розширювати.

Далі наведено порівняння React, Angular, Vue.js, Svelte у таблиці 2.1.

Таблиця 2.1 – Порівняння React з його аналогами

Характеристика	React	Angular	Vue.js	Svelte
Тип	Бібліотека	Фреймворк	Фреймворк	Компілер
Розробник	Facebook	Google	Evan You та спільнота	Rich Harris та спільнота
Перший реліз	2013	2010	2014	2016
Основна концепція	Компоненти, Virtual DOM	MVC, двостороннє зв'язування	Компоненти, реактивні дані	Компоненти, компіляція в чистий JS
JSX / Шаблони	JSX	HTML-шаблони	HTML-шаблони з директивами	HTML-шаблони з реактивними змінними
Крива навчання	Середня	Висока	Низька	Низька

Продовження таблиці 2.1

Характеристика	React	Angular	Vue.js	Svelte
Продуктивність	Висока, залежить від Virtual DOM	Висока, оптимізовані шаблони	Висока, залежить від Virtual DOM	Дуже висока, немає Virtual DOM
Розмір	Відносно невеликий	Великий	Малий	Дуже малий
Компанії, що використовують	Facebook, Instagram, Airbnb	Google, Microsoft, Upwork	Alibaba, Xiaomi, Grammarly	The New York Times, Spotify, 3commas
Інструменти розробника	React DevTools	Angular DevTools	Vue DevTools	Svelte DevTools
Серверний рендеринг (SSR)	Next.js	Angular Universal	Nuxt.js	Sapper (застарілий), SvelteKit

За наявності інших фреймворків та бібліотек для розробки клієнтської частини вебзастосунків, React може виділитись простотою в розробці та гнучкістю. Завдяки великій кількості переваг React є одним з лідируючих інструментів для розробки вебзастосунків.

2.2 Обґрунтування вибору бази даних

Для проєкту було обрано систему управління базами даних PostgreSQL.

PostgreSQL (або скорочено Postgres) - це об'єктно-реляційна база даних з відкритим вихідним кодом, яка відзначається надійністю, масштабованістю та розширюваністю. Вона підтримує широкий спектр функцій, включаючи складні SQL-запити, індексацію, транзакції з ACID-властивостями

(атомарність, консистентність, ізоляція, стійкість), реплікацію даних та багато іншого[12].

PostgreSQL є однією з найпопулярніших баз даних у світі, особливо серед розробників відкритого програмного забезпечення та великих корпорацій. Вона підтримується та активно розвивається великою спільнотою розробників, що забезпечує швидкий розвиток, виправлення помилок та додавання нових функцій. PostgreSQL може бути використаний для різноманітних завдань, від невеликих вебзастосунків до великих корпоративних систем з великим обсягом даних.

- надійність – PostgreSQL відомий своєю високою стабільністю та надійністю. Він забезпечує механізми контролю цілісності даних, а також має вбудовані засоби відновлення після відмов;
- масштабованість – PostgreSQL може легко масштабуватися від невеликих проєктів до великих корпоративних систем. Він підтримує розподілені операції, горизонтальне та вертикальне масштабування;
- розширюваність – PostgreSQL має широкий спектр вбудованих та сторонніх розширень, які дозволяють розширити його функціональність, включаючи геопросторові функції, повнотекстовий пошук, роботу з JSON та багато іншого;
- підтримка ACID – PostgreSQL гарантує виконання ACID-властивостей (атомарність, консистентність, ізоляція, стійкість) для транзакцій, що забезпечує надійність та цілісність даних;
- широкі можливості запитів – PostgreSQL підтримує розширений SQL, що дозволяє виконувати складні запити та операції з даними;
- відкритий вихідний код – Будучи вільним програмним забезпеченням з відкритим вихідним кодом, PostgreSQL надає можливість користуватися його функціональністю безкоштовно та змінювати код під власні потреби;
- активна спільнота – PostgreSQL підтримується широкою та активною спільнотою розробників, яка забезпечує швидкий розвиток,

виправлення помилок та підтримку користувачів;

- підтримка розподіленої розробки – PostgreSQL підтримує розподілені операції, що дозволяє працювати з базами даних на різних серверах та в різних локаціях;

- крос-платформеність – PostgreSQL доступний для різних операційних систем, включаючи Linux, Windows та macOS, що робить його універсальним рішенням для різних середовищ розробки;

- гнучкість конфігурації – PostgreSQL має велику кількість налаштувань, які дозволяють налаштовувати його під конкретні потреби проєкту або вимоги користувача.

Далі наведено порівняння PostgreSQL, MySQL, SQLite, MongoDB у таблиці 2.2.

Таблиця 2.2 – Порівняння PostgreSQL з його аналогами.

Характеристика	PostgreSQL	MySQL	SQLite	MongoDB
Тип	Об'єктно-реляційна	Об'єктно-реляційна	Локальна	NoSQL
Мова запитів	SQL	SQL	SQL	BSON (Binary JSON)
Відкритий код	Так	Так	Так	Ні
Масштабованість	Висока	Висока	Низька	Висока
Підтримка ACID	Так	Так	Так	Ні
Швидкість	Висока	Висока	Висока	Залежить від конфігурації
Гнучкість конфігурації	Висока	Висока	Низька	Середня

Продовження таблиці 2.3

Характеристика	PostgreSQL	MySQL	SQLite	MongoDB
Реплікація	Так	Так	Ні	Так
Шифрування даних	Так	Так	Ні	Так
Розширюваність	Висока	Висока	Низька	Висока
Спільнота та підтримка	Велика	Велика	Середня	Велика
Використання	Комплексні проєкти, великі корпорації	Вебзастосунки, середні проєкти	Мобільні застосунки, прототипи	Застосунки з великим обсягом даних, IoT

Хоча існує багато реляційних і нереляційних баз даних, PostgreSQL відрізняється своєю надійністю, масштабованістю, відповідністю стандартам і розширюваністю. Завдяки цим характеристикам, PostgreSQL є ідеальним вибором для розробки складних і високонавантажених вебзастосунків, які потребують надійного та гнучкого управління даними.

2.3 Обґрунтування вибору середовища для серверної частини

Для розробки серверної частини вебзастосунку було обрано платформу NodeJS.

Node.js - це середовище виконання JavaScript, яке дозволяє запускати JavaScript-код поза веб-браузером. Node.js побудований на основі двигуна V8, розробленого компанією Google, який використовується в браузері Chrome для виконання JavaScript[13].

Програмна платформа NodeJS має велику кількість переваг, а саме:

— асинхронна, подійно-орієнтована архітектура — NodeJS використовує неблокуючу, асинхронну модель введення-виведення, що

дозволяє ефективно обробляти багато одночасних з'єднань з мінімальним використанням пам'яті та ресурсів процесора. Це особливо важливо для застосунків реального часу, таких як чати або ігри;

- висока продуктивність – Завдяки V8, JavaScript-движку Google, Node.js забезпечує швидке виконання коду. V8 компілює JavaScript безпосередньо в машинний код, що підвищує ефективність роботи;

- єдина мова для клієнта і сервера – Node.js дозволяє використовувати JavaScript як на клієнтській, так і на серверній стороні, що спрощує процес розробки, оскільки розробникам не потрібно використовувати різні мови для різних частин застосунку

- велика екосистема пакетів – Менеджер пакетів npm надає доступ до величезної кількості бібліотек і модулів, що значно спрощує та прискорює розробку застосунків.

- масштабованість – Node.js дозволяє легко масштабувати застосунки за допомогою кластеризації, що дозволяє використовувати переваги багатоядерних систем і забезпечувати високу продуктивність під великим навантаженням.

- швидкий цикл розробки – Завдяки своїй швидкості та гнучкості, Node.js дозволяє розробникам швидко ітерувати та вносити зміни, що сприяє прискоренню процесу розробки.

- підтримка різних баз даних – Node.js підтримує підключення до різноманітних типів баз даних, включаючи реляційні (MySQL, PostgreSQL), NoSQL (MongoDB, Cassandra) та кешуючі системи (Redis).

- активна спільнота – Велика і активна спільнота розробників забезпечує постійну підтримку, оновлення та створення нових інструментів і бібліотек для Node.js, що робить його одним із найбільш динамічно розвиваються інструментів для розробки

- легкість навчання – JavaScript є однією з найпопулярніших мов програмування, і багато розробників вже мають з нею досвід. Це робить процес навчання Node.js швидким і простим для тих, хто вже знайомий з

JavaScript.

Нижче наведено таблицю 2.3 з порівнянням Node.js, Python (Django/Flask), Ruby (Ruby on Rails), PHP (Laravel)

Таблиця 2.3 – Порівняння характеристик Node.js з його основними аналогами.

Характеристика	Node.js	Python (Django/Flask)	Ruby (Ruby on Rails)	PHP (Laravel)
Мова програмування	JavaScript	Python	Ruby	PHP
Асинхронність	Підтримується	Обмежена (через бібліотеки)	Обмежена (через бібліотеки)	Обмежена (через бібліотеки)
Архітектура	Асинхронна, подійно-орієнтована	Синхронна (Django), асинхронна (Flask)	Синхронна	Синхронна
Популярність	Висока	Висока	Середня	Висока
Продуктивність	Висока	Середня	Середня	Середня
Масштабованість	Висока	Висока	Висока	Висока
Крива навчання	Низька для JS розробників	Низька (Flask), середня (Django)	Середня	Середня
Наявність бібліотек	Величезна (npm)	Величезна (PyPI)	Велика (Gems)	Величезна (Packagist)
Документація	Відмінна	Відмінна	Відмінна	Відмінна

Спільнота	Величезна і активна	Величезна і активна	Середня і активна	Величезна і активна
-----------	---------------------	---------------------	-------------------	---------------------

Продовження таблиці 2.3

Характеристика	Node.js	Python (Django/Flask)	Ruby (Ruby on Rails)	PHP (Laravel)
Підтримка різних БД	Так	Так	Так	Так
Мікросервіси	Підходить	Підходить	Можливо, але не поширено	Можливо, але не поширено
Інструменти розробки	Добре розвинені	Добре розвинені	Добре розвинені	Добре розвинені
Крос-платформеність	Висока	Висока	Висока	Висока
Використання	Вебзастосунки, реальні часи	Вебзастосунки, наука про дані	Вебзастосунки, стартапи	Вебзастосунки, CMS

2.4 Висновок за розділом

Для розробки клієнтської частини було обрано HTML, CSS, ReactJS,. ReactJS був обраний завдяки легкій інтеграції, використання JavaScript XML, Virtual DOM та розвиненій екосистемі.

Для розробки бази даних для вебзастосунка була обрана PostgreSQL через її розширюваність, активну спільноту, гнучкість конфігурацій та підтримку розподіленої розробки.

Для розробки серверної частини вебзастосунку використовувався NodeJS завдяки його асинхронній, подійно-орієнтованій архітектурі, єдиній мові для клієнта та сервера та великій екосистемі.

Загалом, цей набір технологій відрізняється високою швидкістю, масштабованістю, продуктивністю, гнучкістю та активною підтримкою спільноти, що робить їх ідеальними для створення сучасних і високопродуктивних вебзастосунків.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Алгоритм роботи головної сторінки

Розглянемо алгоритм роботи головної сторінки. Після відкриття головної сторінки, користувач бачить колонки для фільтрацій по брендам та по типам інструментам та послуг, при натисканні яких відбувається фільтрація асортименту в базі даних та виводу на клієнтську частину, поле для пошуку по назві, в якому відбувається пошук за назвою інструменту або послуги . Користувач обирає необхідну для нього позицію в асортименті, або знаходить через систему фільтрації . Після виставлення необхідних фільтрів, автоматично відправляється запит до бази даних та виведення товарів. При натисканні на відповідну кнопку «Додати товар до кошика» в карті інструмента, користувач має змогу перейти до кошика .

Схема алгоритму роботи головної сторінки наведено нижче на рисунку 3.1.



Рисунок 3.1 – Схема основного алгоритму роботи головної сторінки.

3.2 Алгоритм роботи сторінки авторизації

При переході з головної до кошика, туди передаються дані про обрані товари, але до кошика має доступ тільки авторизований користувач. Користувач вводить свої особисті дані, при вдалій авторизації до БД додається новий юзер, у разі якщо користувач авторизований. Після введення даних та натискання на кнопку «Перейти до оплати», робиться запит до БД, аби додати дані про нового клієнта та оновити дані про зайняте місце.

Схема основного алгоритму цієї сторінки надана нижче на рисунку 3.2.

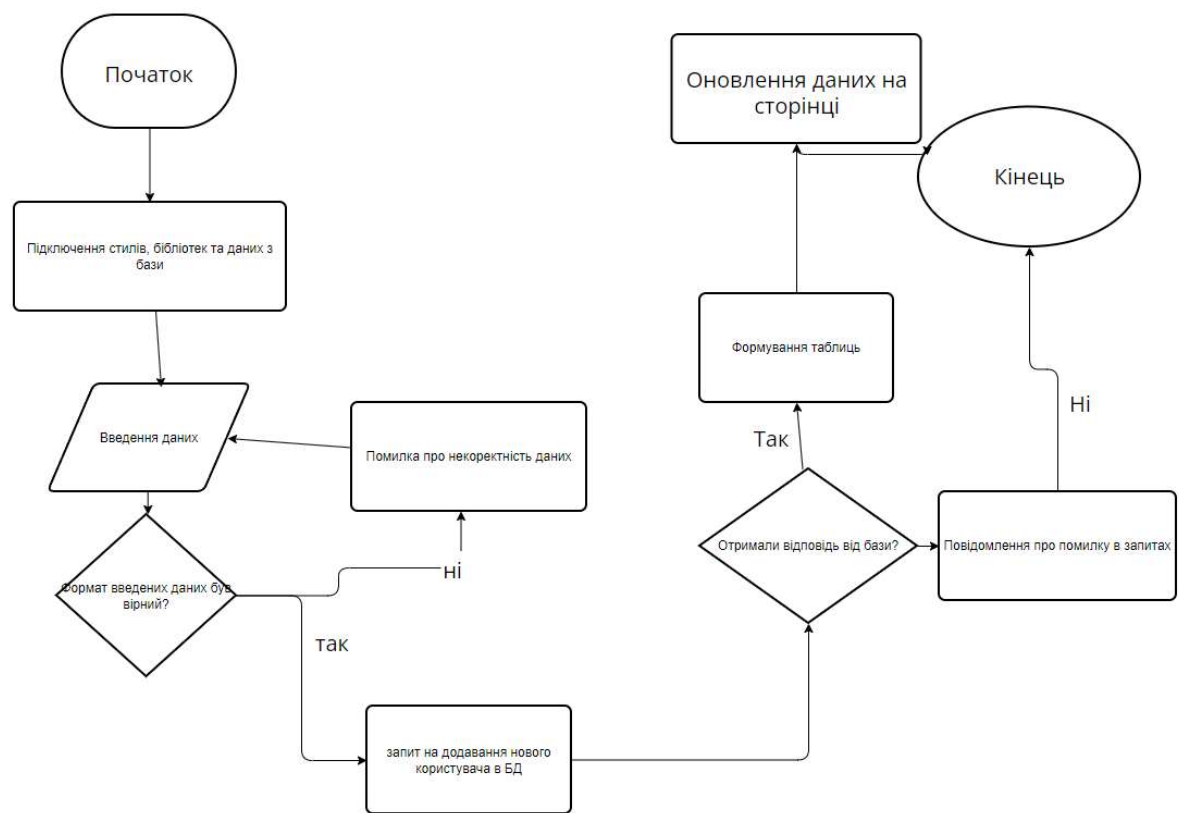


Рисунок 3.2 – Схема основного алгоритму роботи сторінки авторизації.

3.3 Алгоритм роботи сторінки кошика

Потрапити до кошика можливо в карточці товару або в хедері вебзастосунку. Після того як клієнт додає до кошика товар в кошику додається функція збільшення кількості обраного товару, та створення заказу. Після

натискання на кнопку «створити заказ» сформується запит до бази даних для додавання нових даних.

Схему алгоритму роботи сторінки кошику надано далі на рисунку 3.3.

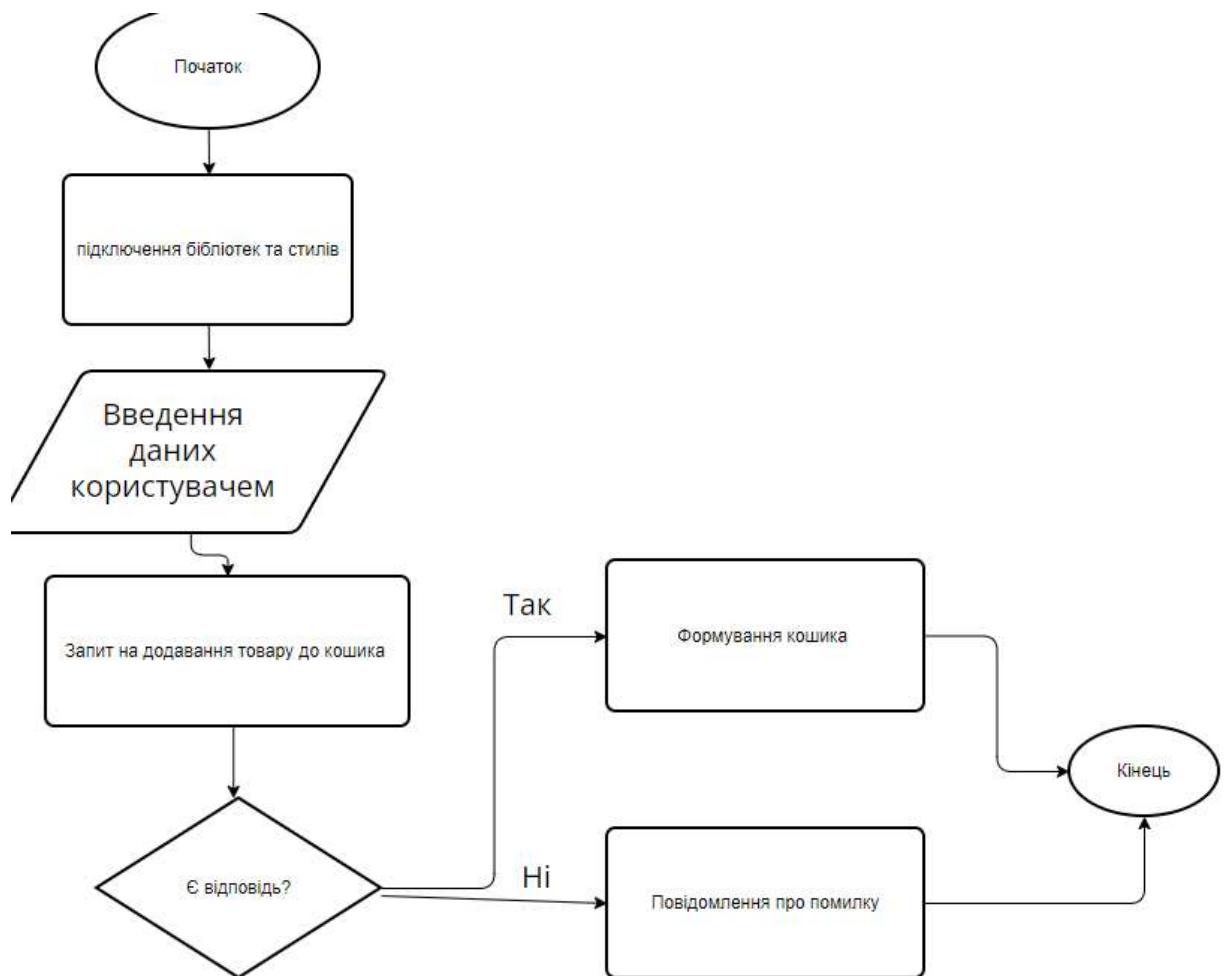


Рисунок 3.3 – Схема алгоритму сторінки кошика.

3.4 Алгоритм роботи сторінки адміністратора

Перейти на сторінку адміністратора можна через навігаційне меню якщо ввести на сторінці авторизації дані користувача Адмін. У тілі на цій формі 6 кнопок: «додати бренд», «додати тип», «додати інструмент-послугу», «Видалити тип», «Видалити бренд», «Видалити інструмент-послугу».

При натисненні на кнопку виходу, користувача буде перенаправлено на головну сторінку. Кнопки формують відповідні запити до бази даних.

Після натиснення однієї з них буде відображено присутні для них поля для створення або. При натисканні на кнопку «Додати бренд» з'являється форма для введення назви бренду. При натисканні на кнопку «Додати тип» з'являється форма для введення назви типу. При натисканні на кнопку «Додати інструмент-послугу» з'являється форма для введення назви, картинок, характеристик, ціни та власності інструмента або послуги. При натисканні на кнопку «Видалити бренд» з'являється форма для введення назви бренду. При натисканні на кнопку «Видалити тип» з'являється форма для введення назви типу. При натисканні на кнопку «Видалити інструмент-послугу» з'являється форма для введення назви інструмента або послуги.

Схема алгоритму роботи сторінки адміністратора наведена далі на рисунку 3.5.

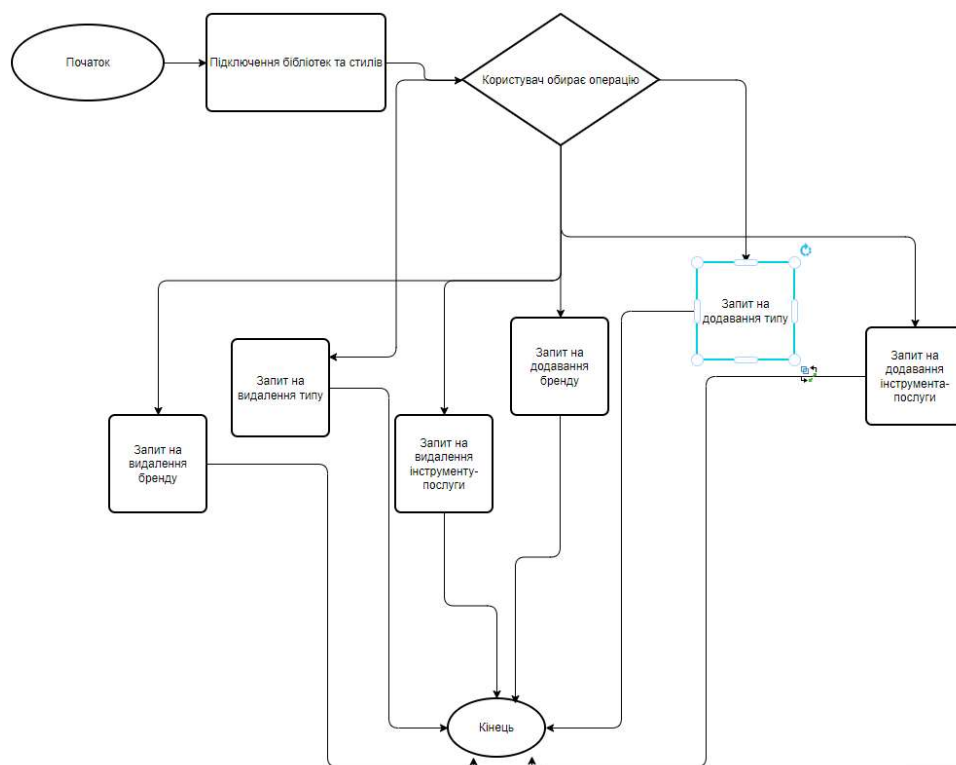


Рисунок 3.4 – Схема алгоритму роботи сторінки адміністратора.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення і умови використання вебзастосунку

Вебзастосунок створений для автоматизації продажу товарів та послуг в майстерні музичних інструментів

Частина клієнта була розроблена за допомогою HTML, CSS та фреймворку ReactJS. Сервер вебзастосунку був написаний за допомогою платформи Node.js та використовує базу даних PostgreSQL.

Для того, щоб працювати з цим вебзастосунком потрібен вебсервер та підключення до інтернету.

Для того, щоб вебзастосунок міг працювати потрібен пристрій, який може вводити дані, має вихід до мережі Інтернет, має встановлений веббраузер Google Chrome, Microsoft Edge та інші. Ще пристрій повинен вміти відображати вебсторінки.

4.2 Характеристики продукту

Під час розробки застосунку були зроблені кілька логічних блоків, на які можна умовно поділити сайт:

- AuthComponent/index.jsx – логічний блок, у якому відбувається реєстрація або авторизація;
- BasketDeviceItem/index.jsx – логічний блок для роботи кошика;
- DeviceItem/index.jsx – блок для управління карти товару.
- AdminComponent/index.jsx – блок для управління сторінки адміністратора та внесення змін до бази даних.

Деякі функції будуть описані нижче.

Функція пошуку інструмента або послуги необхідна, щоб користувач міг легко знайти в великій кількості елементів потрібну йому позицію. Схема цієї функції на рисунку 4.1.

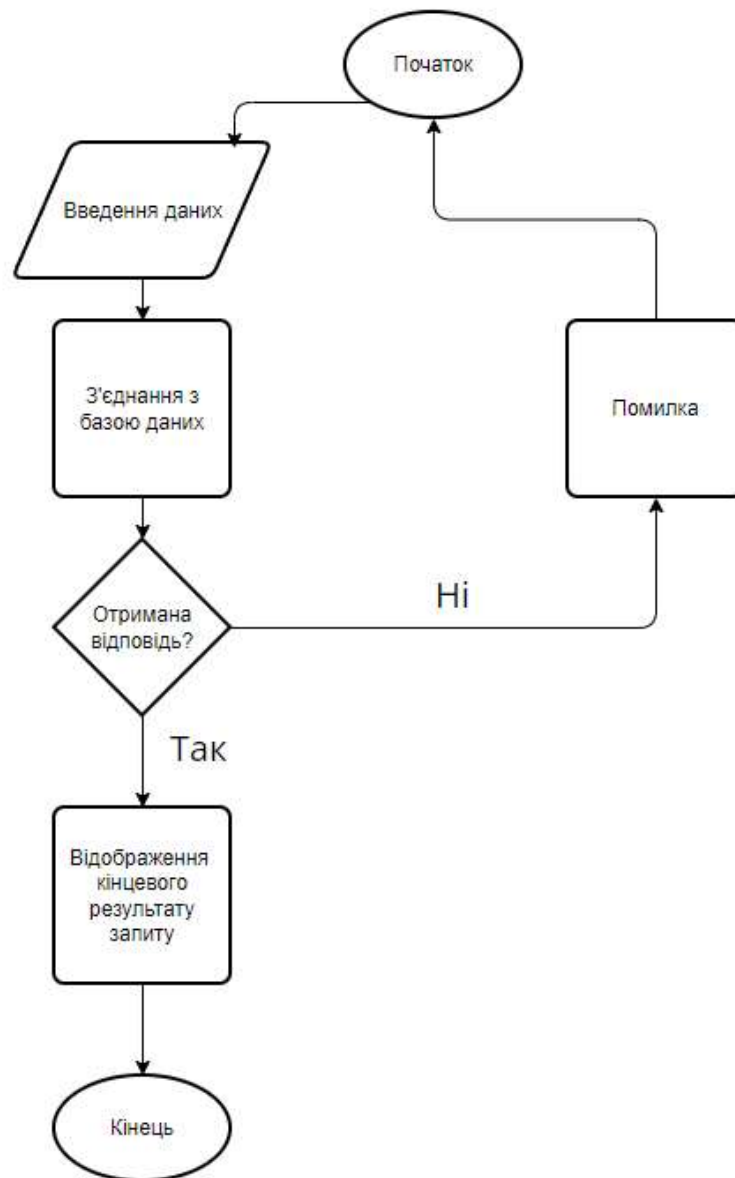


Рисунок 4.1 – Схема функції пошуку інструментів або послуг.

Авторизація як функція є дуже корисною для користувачів вебзастосунка, адже вона дозволяє системі самій заходити у ваш зареєстрований профіль.

Функція додавання до кошику розроблена для додавання до бази запису про обрані товари та послуги перед купівлею.

4.3 Структура бази даних

В цього вебзастосунка присутня база даних. БД складається з 8 таблиць зв'язаних «один до багатьох»:

- instruments – таблиця інструментів та послуг;
- type – таблиця типів інструментів або послуг;
- brand – таблиця брендів інструментів;
- rating – таблиця рейтингу інструмента або послуги;
- user – таблиця користувачів;
- basket – таблиця кошику;
- basket_device – проміжна таблиця кошика та інструмента;
- instrument_info-таблиця інформації про послугу або інструмент.

Як саме зв'язані між собою таблиці представлено на схемі нижче, рисунок 4.2.

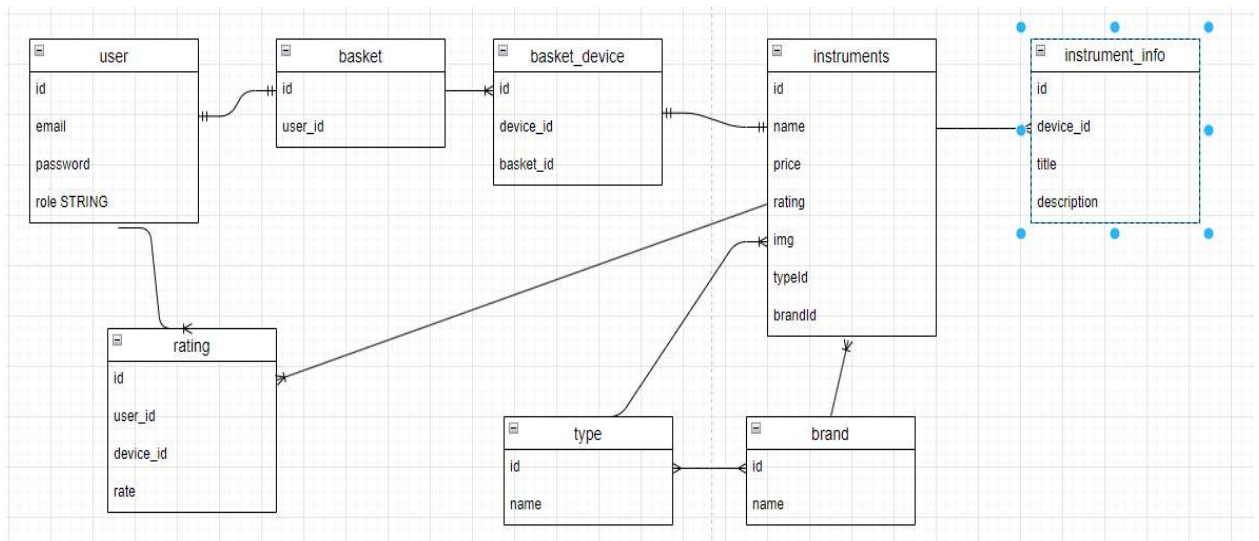


Рисунок 4.2 – Схема структури бази даних.

Зв'язок з базою даних та сервером відбувався за допомогою бібліотеки sequelize та SQL запитів. Приклади запитів наведено у таблиці 4.1 нижче.

Таблиця 4.1 – Приклади запитів у застосунку.

SQL запит	Опис запиту
SELECT "id", "name", "price", "rating", "description", "createdAt", "updatedAt", "typeId", "brandId" FROM "devices" AS "device" WHERE "device"."name" = 'g200';	Пошук даних про наявність інструмента
INSERT INTO "devices" ("id","name","price","rating","description","createdAt","updatedAt","typeId","brandId") VALUES (DEFAULT,\$1,\$2,\$3,\$4,\$5,\$6,\$7,\$8) RETURNING "id","name","price","rating","description","createdAt","updatedAt","typeId","brandId";	Додавання інструмента
SELECT * FROM profiles WHERE email = \$1 AND profile_password = \$2	Пошук даних аккаунту людини для порівняння при авторизації
SELECT "device".*, "photos"."id" AS "photos.id", "photos"."url" AS "photos.url", "photos"."deviceId" AS "photos.deviceId", "photos"."createdAt" AS "photos.createdAt", "photos"."updatedAt" AS "photos.updatedAt", "type"."id" AS "type.id", "type"."name" AS "type.name", "type"."createdAt" AS	Перевірка списку інструментів

Продовження таблиці 4.1

<pre> "type.createdAt", "type"."updatedAt" AS "type.updatedAt", "brand"."id" AS "brand.id", "brand"."name" AS "brand.name", "brand"."createdAt" AS "brand.createdAt", "brand"."updatedAt" AS "brand.updatedAt" FROM (SELECT "device"."id", "device"."name", "device"."price", "device"."rating", "device"."description", "device"."createdAt", "device"."updatedAt", "device"."typeId", "device"."brandId" FROM "devices" AS "device" WHERE "device"."typeId" = '1' AND "device"."brandId" = '1' ORDER BY "device"."rating" DESC LIMIT '4' OFFSET 0) AS "device" LEFT OUTER JOIN "device_photos" AS "photos" ON "device"."id" = "photos"."deviceId" LEFT OUTER JOIN "types" AS "type" ON "device"."typeId" = "type"."id" LEFT OUTER JOIN "brands" AS "brand" ON "device"."brandId" = "brand"."id" ORDER BY "device"."rating" DESC; </pre>	
---	--

Дані з програми у запит записувались через змінні, які мають такий вигляд: «\$1».

4.4 Структура застосунку

Вебзастосунок поділяється на дві частини: клієнтську та адміністраторську. Клієнтська частина відповідає за роботу з клієнтом, здійснює пошук товарів та послуг, здійснює заповнення кошика товарами та послугами. Адміністраторська частина відповідає за додаванням та видаленням інформації з бази даних. Тільки адміністратор зможе скористуватися нею. На сторінці адміністратора можна додавати типи, бренди, інструменти та послуги, а також їх видаляти. Схематично вебзастосунок буде показаний на рисунку 4.5.

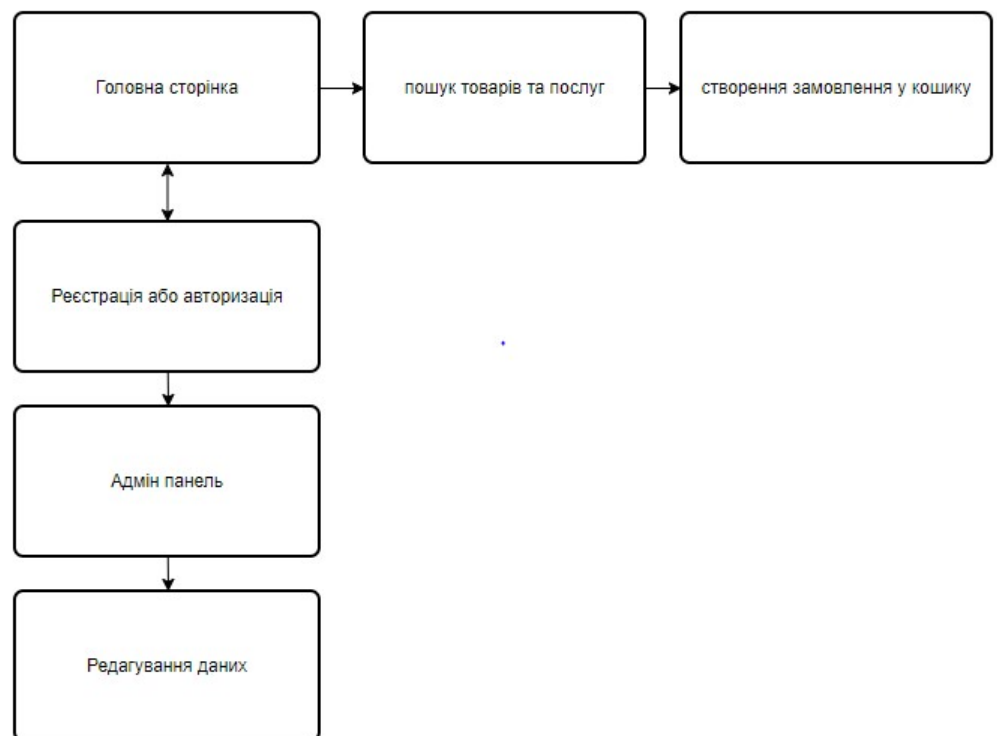


Рисунок 4.5 – Схема структури сайту.

5 КЕРІВНИЦТВО СИСТЕМНОГО АДМІНІСТРАТОРА

5.1 Розгортання сайту

5.1.1 Серверна частина

Спочатку необхідно взяти флешку й перенести на пристрій архів «MusicWorkShop.rar»

Наступним кроком буде перенесення папки MusicWorkShop в будь-яку папку на пристрої. Необхідно працювати всередині папки MusicWorkShop, адже всередині 2 основні папки client та server.

Слід зауважити, що необхідно перевірити, чи встановлені такі програми:

- Node.js і npm;
- PostgreSQL;
- Nginx

Наступним кроком буде налаштування серверної частини та встановлення залежностей.

```
cd server
```

```
npm install
```

У папці server є файл .env , в якому написана конфігурація для з'єднання з базою даних.

```
PORT=5000
```

```
DB_NAME=online_store
```

```
DB_USER=postgres
```

```
DB_PASSWORD=perekatipole12312
```

```
DB_HOST=localhost
```

```
DB_PORT=5432
```

```
SECRET_KEY=secret_key_123
```

Після цих дій потрібно запустити сервер. Введемо команду запуску.

```
npm run dev
```

5.1.2 Клієнтська частина

Наступним кроком буде перехід в папку `client` та встановлення залежностей для клієнтської частини коду.

```
cd..  
cd client  
npm install
```

Наступним кроком буде створення `production build` застосунку.

```
npm run build
```

Перевірте та переконайтеся, що Nginx встановлений правильно. Тепер необхідно налаштувати Nginx для проксування запитів до серверу та обслуговування статичних файлів клієнтської частини. Створіть файл конфігурацій. Це робиться за допомогою наступної команди.

```
sudo nano /etc/nginx/sites-available/tickets
```

Конфігурації:

```
server {  
    listen 80;  
    server_name yourdomain.com;  
  
    location / {  
        root /path/to/your/project/frontend/build;  
        try_files $uri /index.html;  
    }  
  
    location /api {  
        proxy_pass http://localhost:3000;  
        proxy_http_version 1.1;  
        proxy_set_header Upgrade $http_upgrade;  
        proxy_set_header Connection 'upgrade';
```

```

        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}

```

Активуйте конфігурацію і перезапустіть Nginx:

```

sudo ln -s /etc/nginx/sites-available/tickets
/etc/nginx/sites-enabled/
sudo systemctl restart nginx

```

Для керування процесами Node.js потрібно користуватися pm2:

Тепер вебзастосунок буде доступний за вказаним доменом. Налаштуйте DNS для домену, щоб він вказував на IP-адресу сервера.

5.2 Адміністраторська сторінка

Для входу в адміністраторську частину вебзастосунку авторизуйтеся як адміністратор. Введіть в поля «Email» та «пароль» наступне:

Email: perekatipole123123@gmail.com

Пароль: perekatipole123123

як показано на рисунку 5.1.

ТвояНайкращаМайстерня

Авторизація

Авторизація

perekatipole123123@gmail.com

.....

Немає акаунта? Зареєструйтесь!

Увійти

Рисунок 5.1 – Вхід на адміністраторську сторінку.

Після авторизації потрапимо на панель адміністратора як на рисунку 5.2.

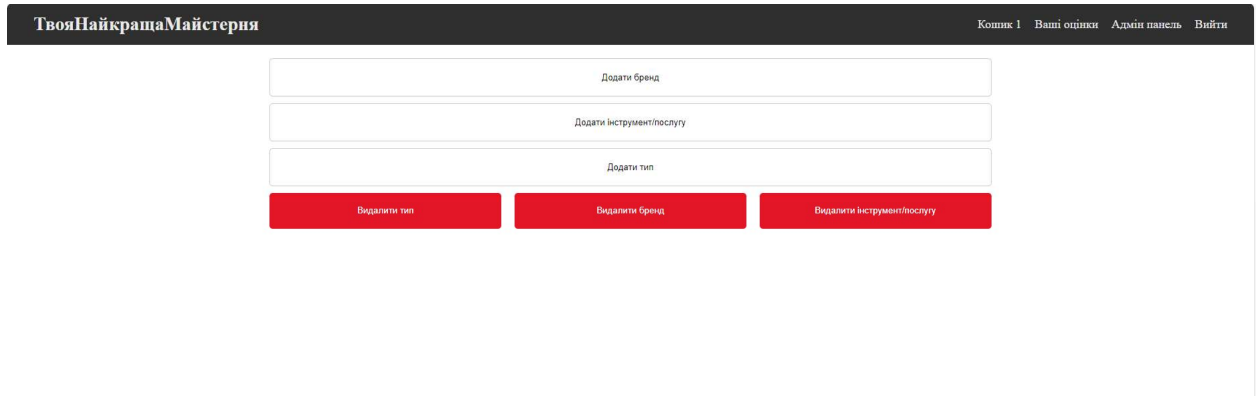


Рисунок 5.2 – Панель адміністратора.

На цій панелі знаходяться кнопки, які розгортають форми для заповнення даними й відправленні їх на сервер. Структури в них майже однакові тому розглянемо форму для додавання інструменту або послуги. Побачити її можна на рисунку 5.3.

The image displays a form titled 'Додати інструмент / послугу'. At the top, there are two dropdown menus: 'Оберіть тип' and 'Оберіть бренд'. Below these are three text input fields with placeholder text: 'Введіть назву інструмента', 'Введіть ціну інструмента', and 'Введіть опис інструмента'. The description field has a small red icon in the bottom right corner. Below the input fields is a file selection area with a button labeled 'Выбор файлов' and the text 'Не выбран ни один файл'. At the bottom of the form, there are two buttons: 'Додати нову власність' and 'Додати власність з варіантами'. In the bottom right corner, there are two more buttons: 'Додати' and 'Закрити'.

Рисунок 5.3 – Вигляд форми для додавання інструментів або послуг.

На цій формі можна додавати інструменти та послуги. Щоб додати інструмент або послугу необхідно заповнити поля, обрати фотографію та натиснути на кнопку «Додати», інструмент або послуга відправляється в базу даних й будуть відображатися на сайті.

Для редагування необхідно самому змінювати значення в базі даних. Для видалення інструменту є своя форма в яку необхідно написати назву інструмента.

Побачити її можна на рисунку 5.4.

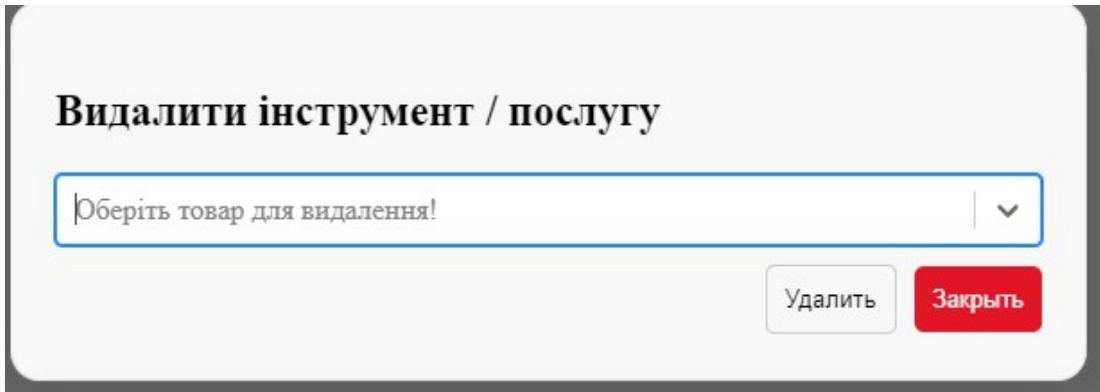
The image shows a web form with a light gray background and rounded corners. At the top, the title 'Видалити інструмент / послугу' is displayed in a bold, black font. Below the title is a text input field with a blue border and a light gray background. Inside the field, the text 'Оберіть товар для видалення!' is written in a gray font, and a small downward-pointing arrow is visible on the right side of the field. Below the input field, there are two buttons: a light gray button with the text 'Удалить' and a red button with the text 'Закреть' in white.

Рисунок 5.4 – Вигляд форми для видалення інструментів або послуг

Так само працюють й інші форми. Якщо панель адміністратора більш не потрібна, можна натиснути «ТвояНайкращаМайстерня», і система поверне користувача на головну сторінку.

6 КЕРІВНИЦТВО КОРИСТУВАЧА

6.1 Призначення вебзастосунку

Вебзастосунок призначений для пошуку та придбання музичних інструментів та послуг для них. Він розроблений для того, щоб спростити в цій сфері роботу та зекономити час користувачів й працівників цієї сфери, автоматизувавши її частину.

6.2 Вибір музичного інструменту або послуги майстра

Після переходу за адресою вебзастосунка, користувач потрапляє на головну сторінку. Головна сторінка продемонстрована далі на рисунку 6.1.

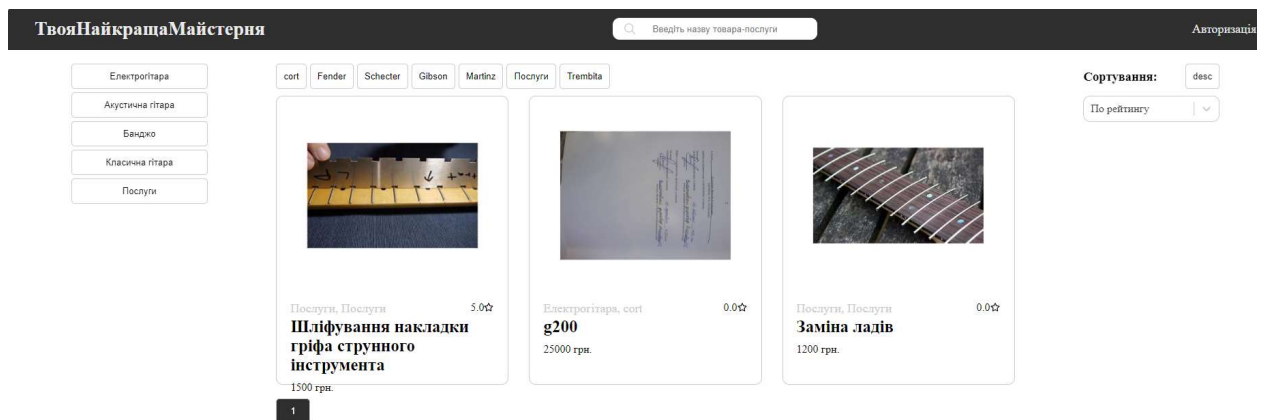


Рисунок 6.1 – Головна сторінка.

Після переходу до головної сторінки необхідно зареєструватись для того, щоб з'явилась можливість користуватися кошиком. Для цього потрібно натиснути на кнопку «Авторизація». Вона переадресує клієнта на сторінку авторизації, яку буде показана далі на рисунку 6.2.

The screenshot shows a login form with the title 'Авторизація' in bold. It contains two input fields: the first contains the email 'perekatipole123123@gmail.com' and the second contains masked characters '.....'. Below the fields, there is a link 'Немає аккаунта? Зареєструйтесь!' and a green button labeled 'Увійти'.

Рисунок 6.2 – Сторінка авторизації.

Користувач повинен ввести свою електронну пошту та пароль, після чого натиснути на кнопку «Увійти». Тоді він авторизується після чого його переадресує на головну сторінку. Якщо у користувача відсутній акаунт, необхідно натиснути на кнопку «Зареєструйтесь!». Після чого його переадресує на сторінку реєстрації, яка зображена на рисунку 6.3.

The screenshot shows a registration form with the title 'Реєстрація' in bold. It contains two input fields: the first contains the email 'perekatipole123123@gmail.com' and the second contains masked characters '.....'. Below the fields, there is a link 'Є аккаунт? Ввійдіть!', a checkbox labeled 'Зареєструватися як адміністратор?' which is currently unchecked, and a green button labeled 'Реєстрація'.

Рисунок 6.3 – Сторінка реєстрації.

На цій сторінці користувачу потрібно ввести свої дані та натиснути на кнопку «Зареєструватися». Потім система збереже його особисті дані в базу даних, авторизує та переадресує на головну сторінку.

У авторизованого користувача зміниться навігаційна панель вебзастосунку і з'явиться можливість обрати інструмент та підтвердити замовлення в кошику. Кошик зображений на рисунку 6.4.

	Заміна ладів	Рейтинг: 0.0	-	1	+	Всього: 1200 Грн.	
	Шліфування накладки грифа струнного інструмента	Рейтинг: 5.0	-	1	+	Всього: 1500 Грн.	

Всього: 2700 грн.

Всього в кошику: 2

Всього позицій: 2

[Створити заказ](#)

Рисунок 6.4 – Кошик.

Для того щоб ознайомитись з інструментом чи послугою майстра детальніше користувачу потрібно на головній сторінці натиснути на один з представлених в асортименті інструментів. Після чого користувача перекине на сторінку інструмента . Ця сторінка продемонстрована на рисунку 6.5.

ТвояНайкращаМайстерня

Кошик 1 Ваші оцінки



g520

Рейтинг: 0.0

Ваша оцінка:

★★★★★

Електрогітара Електрогітара Електрогітара Електрогітара
Електрогітара Електрогітара Електрогітара Електрогітара
Електрогітара Електрогітара Електрогітара Електрогітара
Електрогітара

47200 ГРН.

Додати товар до кошика

g520-7:

g520

Характеристики

g520-7 : g520

Електрогітара :

Електрогітара :

Рисунок 6.5 – Редагування особистих даних.

Щоб повернутися на головну потрібно натиснути на іконку в шапці сайту.

ВИСНОВКИ

В результаті виконання дипломної роботи було проведено детальний аналіз вимог до вебзастосунку та його функціоналу. На основі цих даних було розроблено вебзастосунок майстерні музичних інструментів.

Проведений аналіз включав:

- визначення потреб користувачів;
- аналіз аналогів;
- технічний аналіз вебзастосунку;
- аналіз результатів тестування.

Розроблений вебзастосунок надає користувачам можливість швидко і комфортно замовляти музичні інструменти та послуги для них, використовуючи інтуїтивно зрозумілий, простий та зручний інтерфейс. Вебзастосунок може забезпечувати необхідним функціоналом користувачів для реєстрації, авторизації, вибору музичних інструментів, послуг майстра та замовлення інструмента.

В ході проведення тестування вебзастосунок показав високу ефективність та надійність. Вебзастосунок відповідає всім поставленим вимогам і готовий до впровадження та використання в реальних умовах.

Таким чином, розроблений вебзастосунок не тільки полегшує процес замовлення музичних інструментів та послуг, але й підвищує ефективність роботи майстерні музичних інструментів, забезпечуючи високий рівень задоволеності користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Замовлення [Електрон. ресурс]. – Режим доступу:
<https://skynum.ua/help/orders>
2. Електронна комерція [Електрон. ресурс]. – Режим доступу:
<https://www.me.gov.ua/Documents/Detail?lang=uk-UA&id=6d3d0ba1-219b-470a-a9b9-a88f7dbf02ec&title=ElektronnaKomertsiiia#:~:text=>
3. Електронний правочин [Електрон. ресурс]. – Режим доступу:
<https://instaco.com.ua/sposoby-ukladannia-elektronnuh-dogovoriv#:~:text=>
4. Електронний договір [Електрон. ресурс]. – Режим доступу:
https://biz.ligazakon.net/analytics/220170_elektronniy-dogovr-pravila-ukladennya-ta-pdpisu
5. Електронна торгівля [Електрон. ресурс]. – Режим доступу:
<https://ips.ligazakon.net/document/bz013345?an=0>
6. acropolis.ua [Електрон. ресурс]. – Режим доступу:
<https://acropolis.ua/>
7. tadyka-guitars.com [Електрон. ресурс]. – Режим доступу:
<https://www.tadyka-guitars.com/>
8. muztorg.ua [Електрон. ресурс]. – Режим доступу:
<https://muztorg.ua/uk/>
9. HTML [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/HTML>
10. CSS [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/CSS>
11. React [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/React>
12. PostgreSQL [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/PostgreSQL>
13. NodeJS [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/Node.js>

ДОДАТОК А
Текст програми

Серверна частина

index.js

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

const PORT = process.env.PORT || 3001;

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

app.use(errorHandler);

app.get('/', (req, res) => {
  res.status(200).json({ message: 'Working' });
});

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();
```

Контроллери

basketController.js

```
const {
  BasketDevice,
  Device,
  DevicePhoto,
  Type,
  Brand,
  DeviceVariant,
```

```

    BasketDeviceVariant,
  } = require('../models/models');
  const ApiError = require('../error/ApiError');

  class BasketController {
    async create(req, res, next) {
      try {
        let { deviceId, basketId, variantsId } = req.body;
        /* typeof(JSON.parse(variantsId)) array */

        const basketDevice = await BasketDevice.create({
          deviceId,
          basketId,
          count: 1,
        });

        if (variantsId) {
          variantsId = JSON.parse(variantsId);

          if (variantsId.length > 0) {
            variantsId.forEach((variantId) => {
              BasketDeviceVariant.create({
                deviceVariantId: variantId,
                basketDeviceId: basketDevice.id,
              });
            });
          }
        }

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

    async update(req, res, next) {
      try {
        const { id, count } = req.body;
        const device = await BasketDevice.findOne({
          where: { id },
        });

        if (!device) {
          return next(ApiError.badRequest('Device not found'));
        } else {
          await BasketDevice.update({ count }, { where: { id } });
          return res.json('basket device count updated');
        }
      } catch (e) {
        next(ApiError.badRequest(e.message));
      }
    }
  }

```

```

}

async getAll(req, res, next) {
  try {
    const { basketId } = req.query;

    const basketsDevices = await BasketDevice.findAll({
      where: { basketId },
      include: [
        {
          model: Device,
          include: [
            { model: DevicePhoto, as: 'photos' },
            { model: Type, as: 'type' },
            { model: Brand, as: 'brand' },
          ],
        },
        {
          model: BasketDeviceVariant,
          as: 'variants',
          include: { model: DeviceVariant, as: 'device_variant' },
        },
      ],
    });
    return res.json(basketsDevices);
  } catch (e) {
    next(ApiError.badRequest(e.message));
  }
}

async delete(req, res, next) {
  try {
    const { id } = req.query;

    if (!id) {
      return next(ApiError.badRequest('id not found'));
    }

    await BasketDeviceVariant.destroy({
      where: { basketDeviceId: id },
    });

    await BasketDevice.destroy({
      where: { id },
    });

    return res.json('basket device deleted');
  } catch (e) {
    next(ApiError.badRequest(e.message));
  }
}

```

```

}

module.exports = new BasketController();

```

brandController.js

```

const { Brand } = require('../models/models');
const ApiError = require('../error/ApiError');

class BrandController {
  async create(req, res) {
    try {
      const { name } = req.body;
      const brand = await Brand.create({ name });
      return res.json(brand);
    } catch (e) {
      next(ApiError.badRequest(e.message));
    }
  }

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

  async delete(req, res, next) {
    try {
      const { id } = req.query;
      await Brand.destroy({ where: { id } });
      return res.json('brand deleted');
    } catch (e) {
      next(ApiError.badRequest(e.message));
    }
  }
}

module.exports = new BrandController();

```

deviceController.js

```

require('multer');
const ApiError = require('../error/ApiError');
const { Op } = require('sequelize');
const uuid = require('uuid');
const path = require('path');
const fs = require('fs');
const { Device, DeviceInfo, DevicePhoto, DeviceVariant, Type, Brand } =
require('../models/models');

const includeArr = [

```

```

    { model: DevicePhoto, as: 'photos' },
    { model: Type, as: 'type' },
    { model: Brand, as: 'brand' },
  ];

class DeviceController {
  async create(req, res, next) {
    try {
      let { name, price, brandId, typeId, info, description } = req.body;
      const prevDevice = await Device.findOne({ where: { name } });
      const images = req.files;
      const imageNames = [];

      if (prevDevice) {
        return next(ApiError.badRequest('Device with this name already exists'));
      }

      if (!images) {
        return next(ApiError.badRequest('No images'));
      }

      const device = await Device.create({
        name,
        price,
        brandId,
        typeId,
        description,
      });

      if (images.img.length > 1) {
        images.img.forEach((image) => {
          const fileName = uuid.v4() + '.jpg';
          image.mv(path.resolve(__dirname, '..', 'static', fileName));
          imageNames.push(fileName);
        });
      } else {
        const fileName = uuid.v4() + '.jpg';
        await images.img.mv(path.resolve(__dirname, '..', 'static', fileName));
        imageNames.push(fileName);
      }

      imageNames.forEach((imageName) => {
        DevicePhoto.create({
          url: imageName,
          deviceId: device.id,
        });
      });

      if (info) {
        info = JSON.parse(info);
      }
    }
  }
}

```

```

    if (info.properties.length > 0) {
      info.properties.forEach((i) => {
        DeviceInfo.create({
          title: i.title,
          description: i.description,
          deviceId: device.id,
        });
      });
    }

    if (info.propertiesWithVariants.length > 0) {
      info.propertiesWithVariants.forEach((property) => {
        (async () => {
          const deviceInfo = await DeviceInfo.create({
            title: property.title,
            description: property.description,
            deviceId: device.id,
          });

          property.variants.forEach((variant) => {
            DeviceVariant.create({
              value: variant.value,
              cost: variant.cost,
              colorHex: variant.colorHex,
              deviceInfoId: deviceInfo.id,
              deviceId: device.id,
            });
          });
        })();
      });
    }
  }

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

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

  if (!brandId && !typeId && !search) {
    devices = await Device.findAll({
      order: [[sort, order]],

```

```

        include: includeArr,
        limit,
        offset,
    });

    count = await Device.count();
}

if (!brandId && !typeId && search) {
    devices = await Device.findAll({
        where: { name: { [Op.iRegexp]: search } },
        order: [[sort, order]],
        include: includeArr,
        limit,
        offset,
    });

    count = await Device.count({
        where: { name: { [Op.iRegexp]: search } },
    });
}

if (brandId && !typeId && !search) {
    devices = await Device.findAll({
        where: { brandId },
        order: [[sort, order]],
        include: includeArr,
        limit,
        offset,
    });

    count = await Device.count({
        where: { brandId },
    });
}

if (brandId && !typeId && search) {
    devices = await Device.findAll({
        where: { brandId, name: { [Op.iRegexp]: search } },
        order: [[sort, order]],
        include: includeArr,
        limit,
        offset,
    });

    count = await Device.count({
        where: { brandId, name: { [Op.iRegexp]: search } },
    });
}

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

```

```

    devices = await Device.findAll({
      where: { typeId },
      order: [[sort, order]],
      include: includeArr,
      limit,
      offset,
    });

    count = await Device.count({
      where: { typeId },
    });
  }

  if (!brandId && typeId && search) {
    devices = await Device.findAll({
      where: { typeId, name: { [Op.iRegexp]: search } },
      order: [[sort, order]],
      include: includeArr,
      limit,
      offset,
    });

    count = await Device.count({
      where: { typeId, name: { [Op.iRegexp]: search } },
    });
  }

  if (brandId && typeId && !search) {
    devices = await Device.findAll({
      where: { typeId, brandId },
      order: [[sort, order]],
      include: includeArr,
      limit,
      offset,
    });

    count = await Device.count({
      where: { typeId, brandId },
    });
  }

  if (brandId && typeId && search) {
    devices = await Device.findAll({
      where: { typeId, brandId, name: { [Op.iRegexp]: search } },
      order: [[sort, order]],
      include: includeArr,
      limit,
      offset,
    });

    count = await Device.count({

```

```

        where: { typeId, brandId, name: { [Op.iRegexp]: search } },
      });
    }

    return res.json({ rows: devices, count });
  }

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

    return res.json(device);
  }

  async delete(req, res, next) {
    try {
      const { id } = req.query;
      const device = await Device.findOne({
        where: { id },
        include: [{ model: DevicePhoto, as: 'photos' }],
      });

      if (!device) {
        return next(ApiError.badRequest('Device not found'));
      }

      if (device.photos.length > 0) {
        device.photos.forEach((photo) => {
          fs.unlink(path.resolve(__dirname, '..', 'static', photo.url), (err) =>
{
            err && console.log(err);
          });
        });
      }

      await Device.destroy({ where: { id } });

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

```

```
module.exports = new DeviceController();
```

ratingController.js

```
const { Rating, Device, DevicePhoto, Type, Brand } = require('../models/models');
const ApiError = require('../error/ApiError');
const { Sequelize } = require('sequelize');

const updateRating = async (deviceId) => {
  const avgRating = await Rating.findAll({
    where: { deviceId },
    attributes: [[Sequelize.fn('AVG', Sequelize.col('rate')), 'avgRating']],
  });

  const result = avgRating.map((r) => r.get('avgRating'))[0];
  const newRating = result ? result : 0;

  Device.update({ rating: newRating }, { where: { id: deviceId } });

  return {
    newRating: newRating,
    message: 'Rating saved',
  };
};

class RatingController {
  async create(req, res, next) {
    try {
      const { deviceId, userId, rate } = req.body;
      const numberRate = Number(rate);
      const rating = await Rating.findOne({ where: { deviceId, userId } });

      if (rating) {
        if (Number(rating.rate) === numberRate) {
          return res.json('You have already rated this device');
        } else {
          await Rating.update({ rate }, { where: { deviceId, userId } });
        }
      } else {
        await Rating.create({ deviceId, userId, rate });
      }

      const result = await updateRating(deviceId);
      return res.json(result);
    } catch (e) {
      next(ApiError.badRequest(e.message));
    }
  }

  async getAllByUser(req, res, next) {
    try {

```

```

const { userId } = req.query;
const ratings = await Rating.findAll({
  where: { userId },
  include: [
    {
      model: Device,
      include: [
        { model: DevicePhoto, as: 'photos' },
        { model: Type, as: 'type' },
        { model: Brand, as: 'brand' },
      ],
    },
  ],
});
return res.json(ratings);
} catch (e) {
  next(ApiError.badRequest(e.message));
}
}

async getOne(req, res, next) {
  try {
    const { userId, deviceId } = req.params;
    const rating = await Rating.findOne({
      where: { userId, deviceId },
    });
    if (rating) {
      return res.json(rating);
    } else {
      return res.json(null);
    }
  } catch (e) {
    next(ApiError.badRequest(e.message));
  }
}

async delete(req, res, next) {
  try {
    const { id, deviceId } = req.query;
    await Rating.destroy({ where: { id } });
    const result = await updateRating(deviceId);
    return res.json(result);
  } catch (e) {
    next(ApiError.badRequest(e.message));
  }
}
}

module.exports = new RatingController();

```

typeController.js

```

const { Type } = require('../models/models');
const ApiError = require('../error/ApiError');

class TypeController {
  async create(req, res, next) {
    try {
      const { name } = req.body;
      const type = await Type.create({ name });
      return res.json(type);
    } catch (e) {
      next(ApiError.badRequest(e.message));
    }
  }

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

  async delete(req, res, next) {
    try {
      const { id } = req.query;

      await Type.destroy({ where: { id } });
      return res.json('type deleted');
    } catch (e) {
      next(ApiError.badRequest(e.message));
    }
  }
}

module.exports = new TypeController();

```

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) {

```

```

try {
  const { email, password, role } = req.body;

  if (!email || !password) {
    return next(ApiError.badRequest('Некорректный email или пароль'));
  }

  const candidate = await User.findOne({ where: { email } });

  if (candidate) {
    return next(ApiError.badRequest('Пользователь с таким email уже существует'));
  }

  const hashPassword = await bcrypt.hash(password, 5);
  const user = await User.create({ email, role, password: hashPassword });
  await Basket.create({ userId: user.id });
  const token = generateJwt(user.id, user.email, user.role);
  return res.json({ token });
} catch (e) {
  next(ApiError.badRequest(e.message));
}
}

async login(req, res, next) {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ where: { email } });

    if (!user) {
      return next(ApiError.internal('Пользователь не найден'));
    }

    let comparePassword = bcrypt.compareSync(password, user.password);

    if (!comparePassword) {
      return next(ApiError.internal('Указан неверный пароль'));
    }

    const token = generateJwt(user.id, user.email, user.role);
    return res.json({ token });
  } catch (e) {
    next(ApiError.badRequest(e.message));
  }
}

async check(req, res, next) {
  try {
    const token = generateJwt(req.user.id, req.user.email, req.user.role);
    return res.json({ token });
  } catch (e) {
    next(ApiError.badRequest(e.message));
  }
}

```

```

    }
  }
}

module.exports = new UserController();

```

Poymepu *basketRouter.js*

```

const Router = require('express');
const router = new Router();
const basketController = require('../controllers/BasketController');
const authMiddleware = require('../middleware/authMiddleware');

router.post('/', authMiddleware, basketController.create);
router.delete('/', authMiddleware, basketController.delete);
router.get('/', authMiddleware, basketController.getAll);
// router.get('/:id', basketController.getOne);
router.put('/', authMiddleware, basketController.update);

module.exports = router;

```

brandRouter.js

```

const Router = require('express');
const router = new Router();
const brandController = require('../controllers/brandController');
const checkRole = require('../middleware/checkRoleMiddleware');

router.post('/', checkRole('ADMIN'), brandController.create);
router.get('/', brandController.getAll);
router.delete('/', checkRole('ADMIN'), brandController.delete);

module.exports = router;

```

deviceRouter.js

```

const Router = require('express');
const router = new Router();
const deviceController = require('../controllers/deviceController');
const checkRole = require('../middleware/checkRoleMiddleware');

router.post('/', checkRole('ADMIN'), deviceController.create);
router.delete('/', checkRole('ADMIN'), deviceController.delete);
router.get('/', deviceController.getAll);
router.get('/:id', deviceController.getOne);

module.exports = router;

```

index.js

```

const Router = require('express');
const router = new Router();
const userRouter = require('./userRouter');
const deviceRouter = require('./deviceRouter');
const brandRouter = require('./brandRouter');
const typeRouter = require('./typeRouter');
const basketRouter = require('./basketRouter');
const ratingRouter = require('./ratingRouter');

router.use('/user', userRouter);
router.use('/type', typeRouter);
router.use('/brand', brandRouter);
router.use('/device', deviceRouter);
router.use('/basket', basketRouter);
router.use('/rating', ratingRouter);

module.exports = router;

```

ratingRouter.js

```

const Router = require('express');
const router = new Router();
const ratingController = require('../controllers/ratingController');
const authMiddleware = require('../middleware/authMiddleware');

router.post('/', authMiddleware, ratingController.create);
router.get('/', authMiddleware, ratingController.getAllByUser);
router.get('/:userId/:deviceId', authMiddleware, ratingController.getOne);
router.delete('/', authMiddleware, ratingController.delete);

module.exports = router;

```

typeRouter.js

```

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);
router.delete('/', checkRole('ADMIN'), typeController.delete);

module.exports = router;

```

userRouter.js

```

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;
```

Мідлвеїри authMiddleware.js

```
const jwt = require('jsonwebtoken');

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

checkRoleMiddleware.js

```
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 TOKEN
      if (!token) {
        return res.status(401).json({ message: 'Не авторизован' });
      }
      const decoded = jwt.verify(token, process.env.SECRET_KEY);
      if (decoded.role !== role) {
        return res.status(403).json({ message: 'Нет доступа' });
      }
      req.user = decoded;
      next();
    } catch (e) {

```

```

    res.status(401).json({ message: 'Не авторизован' });
  }
};
};

```

ErrorHandlingMiddleware.js

```

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

module.exports = function (err, req, res) {
  if (err instanceof ApiError) {
    return res.status(err.status).json({ message: err.message });
  }
  return res.status(500).json({ message: 'Непредвиденная ошибка' });
};

```

API помилок

ApiError.js

```

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;

```

Керування БД

Db.js

```

const { Sequelize } = require('sequelize');

```

```
module.exports = new Sequelize(process.env.DB_NAME, process.env.DB_USER,
process.env.DB_PASSWORD, {
  dialect: 'postgres',
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
});
```

Моделі бази даних models.js

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

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

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

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

const BasketDeviceVariant = sequelize.define('basket_device_variant', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
});

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

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

```

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

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

const DeviceInfo = sequelize.define('device_info', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  title: { type: DataTypes.STRING, allowNull: false },
  description: { type: DataTypes.STRING, allowNull: false },
  deviceId: { type: DataTypes.INTEGER, allowNull: false },
});

const DevicePhoto = sequelize.define('device_photo', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  url: { type: DataTypes.STRING, allowNull: false },
  deviceId: { type: DataTypes.INTEGER, allowNull: false },
});

const DeviceVariant = sequelize.define('device_variant', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  value: { type: DataTypes.STRING, allowNull: false },
  cost: { type: DataTypes.INTEGER, allowNull: false },
  colorHex: { type: DataTypes.STRING, allowNull: false },
  deviceInfoId: { type: DataTypes.INTEGER, allowNull: false },
  deviceId: { type: DataTypes.INTEGER, allowNull: false },
});

const TypeBrand = sequelize.define('type_brand', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
});

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

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

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

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

Brand.hasMany(Device);
Device.belongsTo(Brand);

```

```

Device.hasMany(Rating);
Rating.belongsTo(Device, { onDelete: 'cascade' });

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

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

Device.hasMany(DevicePhoto, { as: 'photos' });
DevicePhoto.belongsTo(Device, { onDelete: 'cascade' });

Device.hasMany(DeviceVariant, { as: 'variants' });
DeviceVariant.belongsTo(Device, { onDelete: 'cascade' });

DeviceInfo.hasMany(DeviceVariant, { as: 'variants' });
DeviceVariant.belongsTo(DeviceInfo, { onDelete: 'cascade' });

BasketDevice.hasMany(BasketDeviceVariant, { as: 'variants' });
BasketDeviceVariant.belongsTo(BasketDevice, { onDelete: 'cascade' });

DeviceVariant.hasMany(BasketDeviceVariant);
BasketDeviceVariant.belongsTo(DeviceVariant);

Type.belongsToMany(Brand, { through: TypeBrand });
Brand.belongsToMany(Type, { through: TypeBrand });

module.exports = {
  User,
  Basket,
  BasketDevice,
  BasketDeviceVariant,
  Device,
  Type,
  Brand,
  Rating,
  TypeBrand,
  DeviceInfo,
  DevicePhoto,
  DeviceVariant,
};

```

Клієнтська частина

App.jsx

```

import { observer } from 'mobx-react-lite';
import { useContext, useEffect, useState } from 'react';

```

```

import { BrowserRouter } from 'react-router-dom';
import { Context } from '.';
import './scss/main.scss';
import AppRouter from './components/AppRouter';
import Navbar from './components/Navbar';
import { check } from './http/userApi';
import Loader from './theme/Loader/Loader';

const App = observer(() => {
  const { user } = useContext(Context);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    check()
      .then((data) => {
        user.setUser(data);
        user.setIsAuth(true);
      })
      .catch((e) => {
        user.setUser({});
        user.setIsAuth(false);
        console.log(e);
      })
      .finally(() => setIsLoading(false));
  }, []);

  if (isLoading) {
    return <Loader />;
  }

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

export default App;

```

index.js

```

import React, { createContext } from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import BasketStore from './store/BasketStore';
import DeviceStore from './store/DeviceStore';
import RatingStore from './store/RatingStore';
import UserStore from './store/UserStore';

```

```

export const Context = createContext(null);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Context.Provider
    value={{
      user: new UserStore(),
      device: new DeviceStore(),
      basket: new BasketStore(),
      rating: new RatingStore(),
    }}>
    <React.StrictMode>
      <App />
    </React.StrictMode>
  </Context.Provider>,
);

```

Poymu *routes.js*

```

import Auth from './pages/Auth';
import Basket from './pages/Basket';
import DevicePage from './pages/DevicePage';
import Shop from './pages/Shop';
import RatedDevices from './pages/RatedDevices';
import {
  ADMIN_ROUTE,
  BASKET_ROUTE,
  DEVICE_ROUTE,
  LOGIN_ROUTE,
  RATING_ROUTE,
  REGISTRATION_ROUTE,
  SHOP_ROUTE,
} from './utils/consts';
import Admin from './pages/Admin';

export const authRoutes = [
  {
    path: BASKET_ROUTE,
    component: Basket,
  },

  {
    path: RATING_ROUTE,
    component: RatedDevices,
  },
];

export const adminRoutes = [
  {

```

```

    path: ADMIN_ROUTE,
    component: Admin,
  },
  {
    path: BASKET_ROUTE,
    component: Basket,
  },

  {
    path: RATING_ROUTE,
    component: RatedDevices,
  },
];

export const publicRoutes = [
  {
    path: SHOP_ROUTE,
    component: Shop,
  },

  {
    path: LOGIN_ROUTE,
    component: Auth,
  },

  {
    path: REGISTRATION_ROUTE,
    component: Auth,
  },

  {
    path: DEVICE_ROUTE + '/:id',
    component: DevicePage,
  },
];

```

consts.js

```

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';
export const RATING_ROUTE = '/rating';
export const SORT = {
  RATING: 'rating',
  PRICE: 'price',
  NAME: 'name',
  TYPE: 'typeId',
};

```

```

    BRAND: 'brandId',
  };
export const ANIMATION_TIME = 200;
export const ADMIN_ROLE = 'ADMIN';
export const USER_ROLE = 'USER';

```

calcTotalBasketPrice.js

```

export const calcTotalBasketPrice = (basketDevices) => {
  return basketDevices.reduce((final, position) => {
    let addedPrice = 0;
    if (position.variants.length) {
      addedPrice = position.variants.reduce(
        (total, variant) => total + variant.device_variant.cost,
        0,
      );
    }

    return final + (!position.count ? 1 : position.count * (position.device.price
+ addedPrice));
  }, 0);
};

```

calcTotalBasketCount.js

```

export const calcTotalBasketCount = (basketDevices) => {
  return basketDevices.reduce((total, device) => total + device.count, 0);
};

```

AddToBasketBtn.jsx

```

import { Link } from 'react-router-dom';
import { BASKET_ROUTE, LOGIN_ROUTE } from '../../utils/consts';
import styles from './AddToBasketBtn.module.scss';
import btnStyles from '../Button/Button.module.scss';

const AddToBasketBtn = ({ isAuth, isAdded, isLoading, onAddToBasket, onClick,
className }) => {
  const classes = `${styles.btn} ${className ? className : ''}`;

  if (!isAuth) {
    return (
      <span className={` ${styles.notAuth} ${classes}`}>
        <Link to={LOGIN_ROUTE}>Ввійдіть</Link> щоб додати в кошик
      </span>
    );
  }

  if (isAdded) {

```

```

    return (
      <Link to={BASKET_ROUTE} className={` ${styles.added} ${classes}`}
onClick={onClick}>
        Товар в кошику. <span className={styles.accent}>Перейти</span>
      </Link>
    );
  }
  return (
    <button
      className={` ${classes} ${isLoading ? btnStyles.loading + ' ' +
styles.unHovered : ''}`}
      onClick={onAddToBasket}>
      Додати товар до кошика
    </button>
  );
};

export default AddToBasketBtn;

```

AlertModal.jsx

```

import Modal from '../components/Modal';
import styles from './AlertModal.module.scss';
import Button from '../Button';

const AlertModal = ({ title, desc, opened, onClose, children }) => {
  return (
    <Modal onClose={onClose} opened={opened}>
      <div className={styles.root}>
        <h2 className={styles.title}>{title}</h2>
        <p className={styles.desc}>{desc}</p>
        <div className={styles.children}>{children}</div>
        <Button variant="danger" onClick={onClose} className={styles.btn}>
          Закрити
        </Button>
      </div>
    </Modal>
  );
};

export default AlertModal;

```

BasketInfo.jsx

```

import { useState } from 'react';
import Skeleton from 'react-loading-skeleton';
import Button from '../Button';
import AlertModal from '../AlertModal';

```

```

import styles from './BasketBottom.module.scss';
import 'react-loading-skeleton/dist/skeleton.css';

const BasketBottom = ({ totalPrice, totalCount, totalPositions, isLoading }) => {
  const [opened, setOpened] = useState(false);
  return (
    <div className={styles.root}>
      <ul className={styles.list}>
        {isLoading ? (
          <Skeleton className={` ${styles.listItem} ${styles.noBorder}`} count={3} />
        ) : (
          <>
            <li className={styles.listItem}>
              <b className={styles.accent}>Всього:</b> {totalPrice} грн.
            </li>
            <li className={styles.listItem}>
              <b className={styles.accent}>Всього в кошику:</b> {totalCount}
            </li>
            <li className={styles.listItem}>
              <b className={styles.accent}>Всього позицій:</b> {totalPositions}
            </li>
          </>
        )}
      </ul>

      <Button
        disabled={isLoading}
        variant="success"
        className={styles.btn}
        onClick={() => setOpened(true)}>
        Створити заказ
      </Button>

      <AlertModal opened={opened} onClose={() => setOpened(false)} title="Щось не так">
        <p className={styles.modalDesc}>
          Це некомерційна робота для тестування вмінь та навичок, тому створення
          заказів неможливе
        </p>
        <p className={styles.modalDesc}>
          Буду радий найвищому балу =)
        </p>
        <p className={styles.modalDesc}>
          Репозиторій с проектом:
          <a
            className={styles.modallink}
            href="ццццц"
            target="_blank">
            Github
          </a>
        </p>
      </AlertModal>
    </div>
  );
};

```

```

        </a>
      </p>
    </AlertModal>
  </div>
);
};

export default BasketBottom;

```

Button.jsx

```

import styles from './Button.module.scss';

const Button = ({
  children,
  onClick,
  disabled = false,
  variant = 'outline', // outline / primary / secondary
  className = '',
  hover = true,
  type = 'button',
  active = false,
  isLoading = false,
  ...props
}) => {
  const classesMap = {
    outline: styles.outline,
    primary: styles.primary,
    secondary: styles.secondary,
    danger: styles.danger,
    success: styles.success,
  };

  return (
    <button
      {...props}
      type={type}
      className={` ${styles.root} ${className} ${disabled ? styles.disabled : ''}
${
      !hover ? styles.unHovered : ''
    } ${classesMap[variant]} ${active ? styles.active : ''} ${isLoading ?
styles.loading : ''}`}
      onClick={onClick}>
      {children}
    </button>
  );
};

```

```
export default Button;
```

Input.jsx

```
import styles from './Input.module.scss';

import React from 'react';

const Input = ({ value, onChange, placeholder, required, type, unfocused,
...props }) => {
  return (
    <input
      {...props}
      type={type || 'text'}
      className={` ${styles.root} ${unfocused ? styles.unfocused : ''} `}
      value={value}
      onChange={onChange}
      placeholder={placeholder}
      required={required || false}
    />
  );
};

export default Input;
```

Loader.jsx

```
import styles from './Loader.module.scss';

const Loader = () => {
  return (
    <div className={styles.root}>
      <svg width="60px" height="60px" viewBox="0 0 60 60">
        <path
          fill="#000"
          d="M25.251,6.461c-10.318,0-18.683,8.365-18.683,18.683h4.068c0-8.071,6.543-14.615,14.615-14.615V6.461z">
          <animateTransform
            attributeType="xml"
            attributeName="transform"
            type="rotate"
            from="0 25 25"
            to="360 25 25"
            dur="0.6s"
            repeatCount="indefinite"
          />
        </path>
      </svg>
    </div>
  );
};
```

```
export default Loader;
```

NoItems.jsx

```
import styles from './NoItems.module.scss';

const NoItems = ({ title, desc }) => {
  return (
    <div className={styles.root}>
      <h2 className={styles.title}>{title}</h2>
      <p className={styles.desc}>{desc}</p>
    </div>
  );
};

export default NoItems;
```

TextArea.jsx

```
import styles from './Textarea.module.scss';

import React from 'react';

const Textarea = ({
  value,
  onChange,
  placeholder,
  required,
  type,
  resize,
  unFocused,
  ...props
}) => {
  const classesMap = {
    vertical: styles.vertical,
    horizontal: styles.horizontal,
    none: styles.none,
  };
  return (
    <textarea
      {...props}
      className={` ${styles.root} ${unFocused ? styles.unFocused : ''}
${classesMap[resize]} `}
      value={value}
      onChange={onChange}
      placeholder={placeholder}
      required={required || false}
```

```

    />
  );
};

export default Textarea;

```

BasketStore.js

```

import { makeAutoObservable } from 'mobx';
import { calcTotalBasketPrice } from '../utils/calcTotalBasketPrice';
import { calcTotalBasketCount } from '../utils/calcTotalBasketCount';

export default class BasketStore {
  constructor() {
    this._basketDevices = [];
    this._basketTotalCount = 0;
    this._basketTotalPrice = 0;
    this._basketTotalPositions = 0;
    makeAutoObservable(this);
  }

  setBasketDevices(devices) {
    this._basketDevices = devices;
    this._basketTotalPrice = calcTotalBasketPrice(devices);
    this._basketTotalCount = calcTotalBasketCount(devices);
  }

  setBasketTotalCount(count) {
    this._basketTotalCount = count;
  }

  setBasketTotalPrice(price) {
    this._basketTotalPrice = price;
  }

  setBasketTotalPositions(positions) {
    this._basketTotalPositions = positions;
  }

  get basketDevices() {
    return this._basketDevices;
  }

  get basketTotalCount() {
    return this._basketTotalCount;
  }

  get basketTotalPrice() {
    return this._basketTotalPrice;
  }
}

```

```

    get basketTotalPositions() {
      return this._basketTotalPositions;
    }
  }
}

```

DeviceStore.js

```

import { makeAutoObservable } from 'mobx';
import { SORT } from '../utils/consts';

export default class DeviceStore {
  constructor() {
    this._types = [];
    this._brands = [];
    this._selectedType = {};
    this._selectedBrand = {};
    this._selectedSort = SORT.RATING;
    this._selectedOrder = 'desc';
    this._search = '';
    this._devices = [];
    this._page = 1;
    this._totalCount = 0;
    this._limit = 4;

    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;
  }
}

```

```
setSelectedSort(sort) {  
  this._selectedSort = sort;  
}  
  
setSelectedOrder(order) {  
  this._selectedOrder = order;  
}  
  
setSearch(search) {  
  this._search = search;  
}  
  
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 selectedSort() {  
  return this._selectedSort;  
}  
  
get selectedOrder() {  
  return this._selectedOrder;  
}  
  
get search() {  
  return this._search;  
}
```

```

    }

    get page() {
      return this._page;
    }

    get totalCount() {
      return this._totalCount;
    }

    get limit() {
      return this._limit;
    }
  }
}

```

RatingStore.js

```

import { makeAutoObservable } from 'mobx';

export default class RatingStore {
  constructor() {
    this._ratedDevices = [];
    this._ratedDevicesCount = 0;
    makeAutoObservable(this);
  }

  setRatedDevices(devices) {
    this._ratedDevices = devices;
  }

  setRatedDevicesCount(count) {
    this._ratedDevicesCount = count;
  }

  get ratedDevices() {
    return this._ratedDevices;
  }

  get ratedDevicesCount() {
    return this._ratedDevicesCount;
  }
}

```

UserStore.js

```

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;
}

get userId() {
  return this._user.id;
}

get userRole() {
  return this._user.role;
}
}

```

Admin.jsx

```

import AdminComponent from '../components/admin/AdminComponent';

const Admin = () => {
  return (
    <div className="container">
      <AdminComponent />
    </div>
  );
};

export default Admin;

```

Auth.jsx

```
import AuthComponent from '../components/AuthComponent';

const Auth = () => {
  return <AuthComponent />;
};

export default Auth;
```

Basket.jsx

```
import { observer } from 'mobx-react-lite';
import { useContext, useEffect, useState } from 'react';
import { Context } from '..';
import BasketDevicesList from '../components/basket/BasketDevicesList';
import { fetchBasketDevices } from '../http/basketApi';
import BasketBottom from '../theme/BasketInfo';
import NoItems from '../theme/NoItems';

const Basket = observer(() => {
  const { basket, user } = useContext(Context);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    if (user.isAuth) {
      fetchBasketDevices(user.userId)
        .then((data) => {
          basket.setBasketDevices(data);
          basket.setBasketTotalPositions(data.length);
        })
        .catch((e) => {
          alert('Помилка при отриманні кошика');
          console.log(e);
        })
        .finally(() => setIsLoading(false));
    }
  }, []);

  if (!basket.basketTotalPositions && !isLoading) {
    return (
      <div className="container">
        <NoItems
          title="Кошик порожній ="
          desc="Додайте хоча б один товар та повертайтеся!"
        />
      </div>
    );
  }

  return (
```

```

    <div className="container">
      <BasketDevicesList isLoading={isLoading} />
      <BasketBottom
        totalPrice={basket.basketTotalPrice}
        totalCount={basket.basketTotalCount}
        totalPositions={basket.basketTotalPositions}
        isLoading={isLoading}
      />
    </div>
  );
});

export default Basket;

```

DevicePage.jsx

```

import { observer } from 'mobx-react-lite';
import { useContext, useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';
import { Context } from '..';
import DevicePageComponent from '../components/device/DevicePageComponent';
import { createBasketDevice, fetchBasketDevices } from '../http/basketApi';
import { fetchOneDevice } from '../http/deviceApi';
import { fetchOneRating } from '../http/ratingApi';
import DevicePageLoader from
'../components/device/DevicePageComponent/DevicePageLoader';

const DevicePage = observer(() => {
  const { id } = useParams();
  const { user, basket } = useContext(Context);
  const [device, setDevice] = useState([]);
  const [userRate, setUserRate] = useState(null);
  const [isLoading, setIsLoading] = useState(true);
  const [isUserDataLoading, setIsUserDataLoading] = useState(true);
  const [isBasketUpdating, setIsBasketUpdating] = useState(false);

  const addDeviceToBasket = (variantsObj, callback) => {
    setIsBasketUpdating(true);

    createBasketDevice({
      deviceId: +id,
      basketId: +user.userId,
      variantsId: variantsObj.map((variant) => variant.id),
    })
      .then(() => {
        fetchBasketDevices(user.userId)
          .then((data) => {
            basket.setBasketDevices(data);

```

```

        basket.setBasketTotalPositions(data.length);
        callback();
    })
    .finally(() => setIsBasketUpdating(false));
})
.catch((e) => {
    alert('Ошибка при добавлении товара в корзину');
    setIsBasketUpdating(false);
    console.log(e);
});
});

useEffect(() => {
    try {
        fetchOneDevice(id)
            .then((data) => {
                setDevice(data);
            })
            .finally(() => setIsLoading(false));
    } catch (e) {
        alert('Помилка при отриманні товара');
        console.log(e);
    }

    if (user.isAuth) {
        Promise.all([
            fetchBasketDevices(user.userId),
            fetchOneRating({ userId: user.userId, deviceId: id }),
        ])
            .then(([basketData, ratingData]) => {
                if (basketData) {
                    basket.setBasketDevices(basketData);
                    basket.setBasketTotalPositions(basketData.length);
                }

                if (ratingData) {
                    setUserRate(ratingData.rate);
                }
            })
            .catch((e) => {
                alert('Помилка при отриманні користувацьких даних');
                console.log(e);
            })
            .finally(() => {
                setIsUserDataLoading(false);
            });
    } else {
        setIsUserDataLoading(false);
    }
}, []);

```

```

if (isLoading || isUserDataLoading) {
  return (
    <div className="container">
      <DevicePageLoader />
    </div>
  );
}

return (
  <div className="container">
    <DevicePageComponent
      {...device}
      addDeviceToBasket={addDeviceToBasket}
      userRate={userRate}
      isBasketUpdating={isBasketUpdating}
      setUserRate={setUserRate}
    />
  </div>
);
});

export default DevicePage;

```

RatedDevices.jsx

```

import RatedDevicesList from '../components/rating/RatedDevicesList';

const RatedDevices = () => {
  return (
    <div className="container">
      <RatedDevicesList />
    </div>
  );
};

export default RatedDevices;

```

Shop.jsx

```

import { observer } from 'mobx-react-lite';
import { useContext, useEffect, useState } from 'react';
import { Context } from '..';
import BrandsBar from '../components/BrandsBar';
import DevicesList from '../components/device/DevicesList';
import Pagination from '../components/Pagination';
import TypesSidebar from '../components/TypesSidebar';
import { fetchBasketDevices } from '../http/basketApi';
import { fetchBrands, fetchDevices, fetchTypes } from '../http/deviceApi';

```

```

import { fetchRatings } from '../http/ratingApi';
import Sort from '../components/Sort';

const Shop = observer(() => {
  const { device, basket, rating, user } = useContext(Context);
  const [deviceInfoLoading, setDeviceInfoLoading] = useState(true);
  const [isOptionsLoading, setIsOptionsLoading] = useState(true);
  const [isDevicesLoading, setIsDevicesLoading] = useState(true);

  useEffect(() => {
    setIsOptionsLoading(true);

    Promise.all([fetchTypes(), fetchBrands()])
      .then(([types, brands]) => {
        device.setTypes(types);
        device.setBrands(brands);
      })
      .catch((e) => {
        alert('Помилка при завантаженні брендів та типів');
        console.log(e);
      })
      .finally(() => setIsOptionsLoading(false));
  }, []);

  useEffect(() => {
    setIsDevicesLoading(true);

    fetchDevices({
      typeId: device.selectedType.id,
      brandId: device.selectedBrand.id,
      page: device.page,
      limit: device.limit,
      search: device.search,
      sort: device.selectedSort,
      order: device.selectedOrder,
    })
      .then((data) => {
        device.setDevices(data.rows);
        device.setTotalCount(data.count);
      })
      .catch((e) => {
        alert('Помилка при завантаженні товарів');
        console.log(e);
      })
      .finally(() => {
        setIsDevicesLoading(false);
      });
  }, [
    device.selectedType,
    device.selectedBrand,
    device.page,
  ]);
});

```

```

    device.search,
    device.selectedSort,
    device.selectedOrder,
  ]);

  useEffect(() => {
    setDeviceInfoLoading(true);
    if (user.isAuth) {
      Promise.all([fetchBasketDevices(user.userId), fetchRatings(user.userId)])
        .then(([basketDevices, ratings]) => {
          basket.setBasketDevices(basketDevices);
          basket.setBasketTotalPositions(basketDevices.length);

          rating.setRatedDevices(ratings);
          rating.setRatedDevicesCount(ratings.length);
        })
        .catch((e) => {
          alert('Помилка при завантаженні кошика та рейтингу');
          console.log(e);
        })
        .finally(() => setDeviceInfoLoading((prev) => prev + 1));
    }
  }, []);

  return (
    <div className="shop-container">
      <div className="shop-sidebar-left">
        <TypesSidebar isLoading={isLoading} />
      </div>

      <div className="shop-sidebar-right">
        <Sort isLoading={isLoading} />
      </div>

      <div className="container">
        <BrandsBar isLoading={isLoading} />
        <DevicesList isLoading={isLoading && deviceInfoLoading} />
        <Pagination isLoading={isLoading && deviceInfoLoading} />
      </div>
    </div>
  );
});

export default Shop;

```

basketApi.js

```
import { $authHost } from './index';
```

```

export const createBasketDevice = async (device) => {
  // noinspection CommaExpressionJS
  const formData = Object.entries(device).reduce(
    (formData, [key, value]) => (formData.append(key, JSON.stringify(value)),
    formData),
    new FormData(),
  );

  const { data } = await $authHost.post('/api/basket', formData);
  return data;
};

export const fetchBasketDevices = async (basketId) => {
  const { data } = await $authHost.get('/api/basket', {
    params: {
      basketId,
    },
  });
  return data;
};

export const updateBasketDevice = async (device) => {
  const { data } = await $authHost.put('/api/basket', device);
  return data;
};

export const deleteBasketDevice = async ({ id }) => {
  const { data } = await $authHost.delete('/api/basket', { params: { id } });
  return data;
};

```

deviceApi.js

```

import { $authHost, $host } from './index';
import { SORT } from '../utils/consts';

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 deleteType = async (id) => {
  const { data } = await $authHost.delete('/api/type', { params: { id } });
  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 deleteBrand = async (id) => {
  const { data } = await $authHost.delete('/api/brand', { params: { id } });
  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 = 4,
  search,
  sort = SORT.RATING,
  order = 'DESC',
}) => {
  const { data } = await $host.get('/api/device', {
    params: {
      typeId,
      brandId,
      search,
      sort,
      order,
      page,
      limit,
    },
  });
  return data;
};

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

export const deleteDevice = async (id) => {

```

```
const { data } = await $authHost.delete('/api/device', { params: { id } });
return data;
};
```

httpindex.js

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

$authHost.interceptors.request.use(authInstance);

export { $host, $authHost };
```

ratingApi.js

```
import { $authHost } from './index';

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

export const fetchRatings = async (userId) => {
  const { data } = await $authHost.get('/api/rating', { params: { userId } });
  return data;
};

export const fetchOneRating = async ({ userId, deviceId }) => {
  const { data } = await $authHost.get(`api/rating/${userId}/${deviceId}`);
  return data;
};

export const deleteRating = async (id, deviceId) => {
  const { data } = await $authHost.delete(`api/rating`, { params: { id, deviceId } });
  return data;
};
```

```
};
```

userApi.js

```
import jwt_decode from 'jwt-decode';
import { $authHost, $host } from './index';

export const registration = async (email, password, role) => {
  const { data } = await $host.post('/api/user/registration', { email, password,
    role: role });
  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);
};
```

useDebounce.js

```
import { useCallback, useRef } from 'react';

export default function useDebounce(callback, delay) {
  const timer = useRef();

  return useCallback(
    (...args) => {
      if (timer.current) {
        clearTimeout(timer.current);
      }

      timer.current = setTimeout(() => {
        callback(...args);
      }, delay);
    },
    [callback, delay],
  );
}
```

useMount.js

```
import { useEffect, useState } from 'react';

export const useMount = ({ opened, animationTime }) => {
  const [mounted, setMounted] = useState(false);

  useEffect(() => {
    if (opened && !mounted) {
      setMounted(true);
    } else if (!opened && mounted) {
      setTimeout(() => setMounted(false), animationTime);
    }
  }, [opened]);

  return {
    mounted,
  };
};
```

AdminComponent.jsx

```
import { useState } from 'react';
import CreateBrand from '../modals/CreateBrand';
import CreateDevice from '../modals/CreateDevice';
import CreateType from '../modals/CreateType';

import styles from './AdminComponent.module.scss';
import Button from '../../../theme/Button';
import DeleteType from '../modals/DeleteType';
import DeleteBrand from '../modals/DeleteBrand';
import DeleteDevice from '../modals/DeleteDevice';

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

  const [deleteTypeVisible, setDeleteTypeVisible] = useState(false);
  const [deleteBrandVisible, setDeleteBrandVisible] = useState(false);
  const [deleteDeviceVisible, setDeleteDeviceVisible] = useState(false);

  return (
    <>
      <div className={styles.wrapper}>
        <Button onClick={() => setBrandVisible(true)} className={styles.btn}>
          Додати бренд
        </Button>
        <Button onClick={() => setDeviceVisible(true)} className={styles.btn}>
          Додати інструмент/послугу
        </Button>
      </div>
    </>
  );
};
```

```

    </Button>
    <Button onClick={() => setTypeVisible(true)} className={styles.btn}>
      Додати тип
    </Button>
    <div className={styles.btnGroup}>
      <Button
        variant="danger"
        onClick={() => setDeleteTypeVisible(true)}
        className={styles.btn}>
        Видалити тип
      </Button>
      <Button
        variant="danger"
        onClick={() => setDeleteBrandVisible(true)}
        className={styles.btn}>
        Видалити бренд
      </Button>

      <Button
        variant="danger"
        onClick={() => setDeleteDeviceVisible(true)}
        className={styles.btn}>
        Видалити інструмент/послугу
      </Button>
    </div>
  </div>
  <CreateBrand opened={brandVisible} onClose={() => setBrandVisible(false)}
/>

  <CreateDevice opened={deviceVisible} onClose={() =>
setDeviceVisible(false)} />
  <CreateType opened={typeVisible} onClose={() => setTypeVisible(false)} />

  <DeleteType opened={deleteTypeVisible} onClose={() =>
setDeleteTypeVisible(false)} />
  <DeleteBrand opened={deleteBrandVisible} onClose={() =>
setDeleteBrandVisible(false)} />
  <DeleteDevice opened={deleteDeviceVisible} onClose={() =>
setDeleteDeviceVisible(false)} />
  </>
  );
};

export default AdminComponent;

```

DeviceInfoBlock.jsx

```

import Button from '../../theme/Button';
import styles from './DeviceInfoBlock.module.scss';

```

```

const DeviceInfoBlock = ({ title, description, number, changeInfo, removeInfo })
=> {
  return (
    <div className={styles.infoBlockItem} key={number}>
      <input
        type="text"
        className={styles.input}
        placeholder="Введіть назву свойства"
        required={true}
        value={title}
        onChange={(e) => changeInfo('title', e.target.value, number)}
      />
      <input
        type="text"
        className={styles.input}
        placeholder="Введіть опис свойства"
        required={true}
        value={description}
        onChange={(e) => changeInfo('description', e.target.value, number)}
      />
      <Button variant="danger" onClick={() => removeInfo(number)}>
        Видалити
      </Button>
    </div>
  );
};

export default DeviceInfoBlock;

```

DeviceMetaSelect.jsx

```

import { useContext } from 'react';
import { observer } from 'mobx-react-lite';
import { Context } from '../../index';
import Select from 'react-select';

import styles from './DeviceMetaSelect.module.scss';

const DeviceMetaSelect = observer(({ rootClassname }) => {
  const { device } = useContext(Context);

  return (
    <div className={` ${styles.root} ${rootClassname}`}>
      <Select
        options={device.types}
        getOptionLabel={(type) => type.name}
        getOptionValue={(type) => type.id}
        onChange={(selectedOption) => device.setSelectedType(selectedOption)}
        placeholder="Оберіть тип"
      />
    </div>
  );
});

```

```

    />

    <Select
      options={device.brands}
      getOptionLabel={brand => brand.name}
      getOptionValue={brand => brand.id}
      onChange={(selectedOption) => device.setSelectedBrand(selectedOption)}
      placeholder="Оберіть бренд"
    />
  </div>
);
});

export default DeviceMetaSelect;

```

CreateBrand.jsx

```

import { useState } from 'react';
import { createBrand } from '../../http/deviceApi';
import Modal from '../../Modal';
import styles from './modals.module.scss';
import Button from '../../theme/Button';
import Input from '../../theme/Input';

const CreateBrand = ({ opened, onClose }) => {
  const [value, setValue] = useState('');
  const [isLoading, setIsLoading] = useState(false);

  const addBrand = (e) => {
    e.preventDefault();
    setIsLoading(true);

    if (!value) {
      alert('Введіть назву бренда!');
      return;
    }

    createBrand({ name: value })
      .then(() => {
        setValue('');
        onClose();
      })
      .catch((e) => {
        alert('Помилка при створенні бренда!');
        console.log(e);
      })
      .finally(() => {
        setIsLoading(false);
      });
  };

```

```

};

return (
  <Modal opened={opened} onClose={onClose}>
    <h2>Додати бренд</h2>
    <form className={styles.form} onSubmit={addBrand}>
      <div className={styles.inputsBlock}>
        <Input
          className={styles.input}
          value={value}
          onChange={(e) => setValue(e.target.value)}
          type="text"
          placeholder="Введіть назву бренда"
          required={true}
        />
      </div>

      <div className={styles.footer}>
        <Button type="submit" isLoading={isLoading}>
          Додати
        </Button>
        <Button variant="danger" className={styles.closeBtn} onClick={onClose}>
          Закрити
        </Button>
      </div>
    </form>
  </Modal>
);
};

export default CreateBrand;

```

CreateDevice.jsx

```

import { observer } from 'mobx-react-lite';
import { useContext, useEffect, useState } from 'react';
import { Context } from '../../../index';
import { createDevice, fetchBrands, fetchTypes } from '../../../http/deviceApi';
import Modal from '../../Modal';
import styles from './modals.module.scss';
import Button from '../../../theme/Button';
import DeviceMetaSelect from '../DeviceMetaSelect';
import DeviceInfoBlock from '../DeviceInfoBlock';
import Input from '../../../theme/Input';
import Textarea from '../../../theme/Textarea';

const CreateDevice = observer(({ opened, onClose }) => {
  const { device } = useContext(Context);
  const [info, setInfo] = useState([]);

```

```

const [infoWithVar, setInfoWithVar] = useState([]);
const [name, setName] = useState('');
const [desc, setDesc] = useState('');
const [price, setPrice] = useState(0);
const [files, setFiles] = useState(null);
const [isLoading, setIsLoading] = useState(false);

useEffect(() => {
  try {
    fetchTypes().then((data) => device.setTypes(data));
    fetchBrands().then((data) => device.setBrands(data));
  } catch (error) {
    console.log(error);
    alert('Помилка при завантаженні класифікацій (типів/брендів)');
  }
}, []);

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 addInfoWithVar = () => {
  setInfoWithVar([
    ...infoWithVar,
    {
      title: '',
      description: '',
      variants: [{ value: '', cost: '', colorHex: '', number: Date.now() }],
      number: Date.now(),
    },
  ]);
};

const setInfoVariant = (number) => {
  const item = infoWithVar.find((i) => i.number === number);

  item.variants.push({ value: '', cost: '', colorHex: '', number: Date.now() });
  setInfoWithVar([...infoWithVar]);
};

const removeInfoVariant = (number) => {
  const item = infoWithVar.find((i) => i.number === number);

```

```

    item.variants.pop();
    setInfoWithVar([...infoWithVar]);
  };

  const updateInfoVariant = (key, value, infoNumber, variantNumber) => {
    const item = infoWithVar.find((i) => i.number === infoNumber);
    const variant = item.variants.find((i) => i.number === variantNumber);

    variant[key] = value;
    setInfoWithVar([...infoWithVar]);
  };

  const removeInfoWithVar = (number) => {
    setInfoWithVar(infoWithVar.filter((i) => i.number !== number));
  };

  const changeInfoWithVar = (key, value, number) => {
    setInfoWithVar(infoWithVar.map((i) => (i.number === number ? { ...i, [key]:
value } : i)));
  };

  const selectFile = (e) => {
    setFiles(e.target.files);
  };

  const addDevice = (e) => {
    e.preventDefault();
    setIsLoading(true);

    if (!device.selectedBrand.id || !device.selectedType.id) {
      alert('Оберіть бренд і тип');
      return;
    }

    const formData = new FormData();
    const total = {
      properties: info,
      propertiesWithVariants: infoWithVar,
    };

    formData.append('name', name);
    formData.append('price', `${price}`);
    formData.append('brandId', device.selectedBrand.id);
    formData.append('typeId', device.selectedType.id);
    formData.append('info', JSON.stringify(total));
    formData.append('description', desc);

    for (let i = 0; i < files.length; i++) {
      formData.append('img', files[i]);
    }
  }

```

```

createDevice(formData)
  .then(() => {
    onClose();
  })
  .catch((e) => {
    alert('Помилка при створенні інструмента');
    console.log(e);
  })
  .finally(() => {
    setIsLoading(false);
  });
};

return (
  <Modal opened={opened} onClose={onClose}>
    <h2>Додати інструмент / послугу</h2>
    <form className={styles.form} onSubmit={addDevice}>
      <DeviceMetaSelect />

      <div className={styles.inputsBlock}>
        <Input
          placeholder="Введіть назву інструмента"
          required={true}
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
        <Input
          type="number"
          placeholder="Введіть ціну інструмента"
          required={true}
          value={price || ''}
          onChange={(e) => setPrice(Number(e.target.value))}
        />
        <Textarea
          value={desc}
          onChange={(e) => setDesc(e.target.value)}
          className={styles.input}
          required={true}
          placeholder="Введіть опис інструмента"
          style={{ resize: 'vertical' }}
        />
        <Input type="file" multiple={true} onChange={selectFile}
required={true} />
      </div>

      <div className={styles.buttons}>
        <Button onClick={addInfo}>Додати нову власність </Button>
        <Button onClick={addInfoWithVar}>Додати власність з варіантами</Button>
      </div>

      <div className={styles.infoBlock}>

```

```

        {info.map((item) => (
            <DeviceInfoBlock
                key={item.number}
                {...item}
                changeInfo={changeInfo}
                removeInfo={removeInfo}
            />
        ))}
    </div>

    <div>
        {infoWithVar.map((item) => (
            <div
                className={` ${styles.infoBlockItem} ${styles.infoWidthVarBlock}`}
                key={item.number}>
                <div className={styles.infoWithVarItems}>
                    <Input
                        placeholder="Введіть назву інструмента-послуги"
                        required={true}
                        value={item.title}
                        onChange={(e) => changeInfoWithVar('title', e.target.value,
item.number)}
                    />
                    <Input
                        placeholder="Введіть опис свойства"
                        value={item.description}
                        onChange={(e) => changeInfoWithVar('description',
e.target.value, item.number)}
                    />
                    <Button variant="danger" onClick={() =>
removeInfoWithVar(item.number)}>
                        Видалити свойство
                    </Button>
                </div>

                <div className={styles.variantWrapper}>
                    Введіть варіанти:
                    <Button onClick={() => setInfoVariant(item.number)}>Додати
варіант</Button>
                    {item.variants.map((obj) => (
                        <div key={obj.number} className={styles.variants}>
                            <Input
                                placeholder="Введіть значення"
                                required={true}
                                value={item.variants.value}
                                onChange={(e) =>
updateInfoVariant('value', e.target.value, item.number,
obj.number)
                            }
                        />
                            <Input

```

```

        placeholder="Введіть націнку цього варіанту"
        required={true}
        value={item.variants.cost}
        onChange={(e) =>
            updateInfoVariant('cost', e.target.value, item.number,
obj.number)
        }
    />
    <Input
        placeholder="Введіть колір в форматі rgb, hex, etc (необ.)"
        value={item.variants.colorHex}
        onChange={(e) =>
            updateInfoVariant('colorHex', e.target.value,
item.number, obj.number)
        }
    />
    <Button variant="danger" onClick={() =>
removeInfoVariant(item.number)}>
        Видалити варіант
    </Button>
</div>
    )})
</div>
    )})
</div>

<div className={styles.footer}>
    <Button type="submit" isLoading={isLoading}>
        Додати
    </Button>
    <Button variant="danger" className={styles.closeBtn} onClick={onClose}>
        Закрити
    </Button>
</div>
</form>
</Modal>
);
});
export default CreateDevice;

```

CreateType.jsx

```

import { useState } from 'react';
import { createType } from '../../http/deviceApi';
import Modal from '../../Modal';
import styles from './modals.module.scss';
import Button from '../../theme/Button';
import Input from '../../theme/Input';

```

```

const CreateType = ({ opened, onClose }) => {
  const [value, setValue] = useState('');
  const [isLoading, setIsLoading] = useState(false);

  const addType = (e) => {
    e.preventDefault();
    setIsLoading(true);

    if (!value) {
      alert('Введіть назву типа!');
      return;
    }

    createType({ name: value })
      .then(() => {
        setValue('');
        onClose();
      })
      .catch((e) => {
        alert('Помилка при створенні типа!');
        console.log(e);
      })
      .finally(() => {
        setIsLoading(false);
      });
  };

  return (
    <Modal opened={opened} onClose={onClose}>
      <h2>Додати тип</h2>
      <form className={styles.form} onSubmit={addType}>
        <div className={styles.inputsBlock}>
          <Input
            className={styles.input}
            value={value}
            onChange={(e) => setValue(e.target.value)}
            type="text"
            placeholder="Введіть назву типа"
            required={true}
          />
        </div>

        <div className={styles.footer}>
          <Button type="submit" isLoading={isLoading}>
            Додати
          </Button>
          <Button variant="danger" className={styles.closeBtn} onClick={onClose}>
            Закрити
          </Button>
        </div>
      </form>
    </Modal>
  );
};

```

```

    </form>
  </Modal>
);
};

export default CreateType;

```

DeleteBrand.jsx

```

import Modal from '../..//Modal';
import Button from '../..//theme/Button';
import Select from 'react-select';
import { useContext, useEffect, useState } from 'react';
import { deleteBrand, fetchBrands } from '../..//http/deviceApi';
import { Context } from '../..//index';

import styles from './modals.module.scss';

const DeleteBrand = ({ opened, onClose }) => {
  const { device } = useContext(Context);
  const [selectedBrand, setSelectedBrand] = useState(null);
  const [isLoading, setIsLoading] = useState(false);

  useEffect(() => {
    try {
      fetchBrands().then((data) => device.setBrands(data));
    } catch (e) {
      alert('Виникла помилка при загрузці брендів!');
      console.log(e);
    }
  }, []);

  const handleDelete = () => {
    setIsLoading(true);

    if (!selectedBrand) {
      alert('Оберіть бренд!');
      return;
    }

    deleteBrand(selectedBrand.id)
      .then(() => {
        onClose();
      })
      .catch((e) => {
        alert('Виникла помилка при видаленні бренду!');
        console.log(e);
      })
      .finally(() => {

```

```

        setIsLoading(false);
    });
};

return (
    <Modal opened={opened} onClose={onClose}>
        <h2>Видалити бренд</h2>
        <div className={styles.form}>
            <Select
                options={device.brands}
                getOptionLabel={(brand) => brand.name}
                getOptionValue={(brand) => brand.id}
                onChange={(selectedOption) => setSelectedBrand(selectedOption)}
                placeholder="Оберіть бренд для видалення!"
            />

            <div className={styles.footer}>
                <Button onClick={handleDelete} isLoading={isLoading}>
                    Видалити
                </Button>
                <Button variant="danger" className={styles.closeBtn} onClick={onClose}>
                    Закрити
                </Button>
            </div>
        </div>
    </Modal>
);
};

export default DeleteBrand;

```

DeleteDevice.jsx

```

import Modal from '../../Modal';
import Button from '../../theme/Button';
import Select from 'react-select';
import { useContext, useEffect, useState } from 'react';
import { deleteDevice, fetchDevices } from '../../http/deviceApi';
import { Context } from '../../index';

import styles from './modals.module.scss';

const DeleteDevice = ({ opened, onClose }) => {
    const { device } = useContext(Context);
    const [selectedDevice, setSelectedDevice] = useState(null);
    const [isLoading, setIsLoading] = useState(false);

    useEffect(() => {
        try {

```

```

    fetchDevices({
      page: 1,
      limit: device.totalCount + 10,
      sort: device.selectedSort,
      order: device.selectedOrder,
    }).then((data) => {
      device.setDevices(data.rows);
      device.setTotalCount(data.count);
    });
  } catch (e) {
    alert('Виникла помилка при загрузці товарів-послуг!');
    console.log(e);
  }
}, []);

const handleDelete = () => {
  setIsLoading(true);

  if (!selectedDevice) {
    alert('Оберіть інструмент-послугу!');
    return;
  }

  deleteDevice(selectedDevice.id)
    .then(() => {
      onClose();
    })
    .catch((e) => {
      alert('Виникла помилка при видаленні товару-послуги!');
      console.log(e);
    })
    .finally(() => {
      setIsLoading(false);
    });
};

return (
  <Modal opened={opened} onClose={onClose}>
    <h2>Видалити інструмент / послугу</h2>
    <div className={styles.form}>
      <Select
        options={device.devices}
        getOptionLabel={(device) => device.name}
        getOptionValue={(device) => device.id}
        onChange={(selectedOption) => setSelectedDevice(selectedOption)}
        placeholder="Оберіть товар для видалення!"
      />

      <div className={styles.footer}>
        <Button onClick={handleDelete} isLoading={isLoading}>
          Удалить

```

```

        </Button>
        <Button variant="danger" className={styles.closeBtn} onClick={onClose}>
            Заккрыть
        </Button>
    </div>
</div>
</Modal>
);
};

export default DeleteDevice;

```

DeleteType.jsx

```

import Modal from '../..../Modal';
import Button from '../..../theme/Button';
import Select from 'react-select';
import { useContext, useEffect, useState } from 'react';
import { deleteType, fetchTypes } from '../..../http/deviceApi';
import { Context } from '../..../index';

import styles from './modals.module.scss';

const DeleteType = ({ opened, onClose }) => {
    const { device } = useContext(Context);
    const [selectedType, setSelectedType] = useState(null);
    const [isLoading, setIsLoading] = useState(false);

    useEffect(() => {
        try {
            fetchTypes().then((data) => device.setTypes(data));
        } catch (e) {
            alert('Виникла помилка при загрузці типів!');
            console.log(e);
        }
    }, []);

    const handleDelete = () => {
        setIsLoading(true);

        if (!selectedType) {
            alert('Оберіть тип!');
            return;
        }

        deleteType(selectedType.id)
            .then(() => {
                onClose();
            })
    }

```

```

        .catch((e) => {
            alert('Виникла помилка при видаленні типа!');
            console.log(e);
        })
        .finally(() => {
            setIsLoading(false);
        });
    };

    return (
        <Modal opened={opened} onClose={onClose}>
            <h2>Видалити тип</h2>
            <div className={styles.form}>
                <Select
                    options={device.types}
                    getOptionLabel={(type) => type.name}
                    getOptionValue={(type) => type.id}
                    onChange={(selectedOption) => setSelectedType(selectedOption)}
                    placeholder="Оберіть тип для видалення!"
                />

                <div className={styles.footer}>
                    <Button onClick={handleDelete} isLoading={isLoading}>
                        Видалити
                    </Button>
                    <Button variant="danger" className={styles.closeBtn} onClick={onClose}>
                        Закрити
                    </Button>
                </div>
            </div>
        </Modal>
    );
};

export default DeleteType;

```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька Політехніка»
Кафедра програмних засобів

Розробка вебзастосунку майстерні музичних інструментів

Виконав:
ст. гр. КНТ-121сп

Кульченко Я.Е.

Керівник:
к.т.н., доцент

Каплієнко Т.І.

2024

1

Рисунок Б.1 – Слайд №1

Інформація про роботу

Мета роботи – проєктування та створення вебзастосунку, який дозволяє обирати музичні інструменти, профілактичні послуги для інструмента та кастомізацію зручно для любого користувача.

Об'єкт дослідження – вебзастосунки для майстерень музичних інструментів.

Завдання на дипломну роботу :

- Аналіз аналогічних вебзастосунків;
- Порівняння програмного забезпечення;
- Вибір програмного забезпечення для розробки вебзастосунку;
- Створення архітектури вебзастосунку;
- Створення вебзастосунку;
- Тестування продукту.

2

Рисунок Б.2 – Слайд №2

Порівняння переваг аналогів

acropolis.ua	tadyka-guitars.com	muztorg.ua
великий асортимент товарів як інших брендів, так і свого виробництва	наявність новин та блогу майстерні	гарний дизайн інтерфейсу та зручна навігація
зручна навігація	програма лояльності для військових та постійних клієнтів	наявність відділу б/в товарів по низьким цінам
відповідність функціоналу мобільного застосунку та повноцінного десктопного вебзастосунку	цілодобовий зв'язок з службою підтримки	зручна фільтрація товарів та послуг
привабливий дизайн	широкий спектр послуг	наявність відеоблогу з оглядом на інструменти та обладнання

3

Рисунок Б.3 – Слайд №3

Порівняння технологій для клієнтської частини

Характеристика	 React JS	 ANGULAR	 V	 SVELTE
Тип	Бібліотека	Фреймворк	Фреймворк	Компілер
Розробник	Facebook	Google	Evan You та спільнота	Rich Harris та спільнота
Основна концепція	Компоненти, Virtual DOM	MVC, двостороннє зв'язування	Компоненти, реактивні дані	Компоненти, компіляція в чистий JS
JSX / Шаблони	JSX	HTML-шаблони	HTML-шаблони з директивами	HTML-шаблони з реактивними змінними
Продуктивність	Висока, залежить від Virtual DOM	Висока, оптимізовані шаблони	Висока, залежить від Virtual DOM	Дуже висока, немає Virtual DOM
Розмір	Відносно невеликий	Великий	Малий	Дуже малий

4

Рисунок Б.4 – Слайд №4

Порівняння технологій баз даних

Характеристика				
Тип	Об'єктно-реляційна	Об'єктно-реляційна	Локальна	NoSQL
Мова запитів	SQL	SQL	SQL	BSON (Binary JSON)
Відкритий код	Так	Так	Так	Ні
Масштабованість	Висока	Висока	Низька	Висока
Підтримка ACID	Так	Так	Так	Ні
Швидкість	Висока	Висока	Висока	Залежить від конфігурації

5

Рисунок Б.5 – Слайд №5

Порівняння технологій серверної частини

Характеристика				
Мова програмування	JavaScript	Python	Ruby	PHP
Асинхронність	Підтримується	Обмежена (через бібліотеки)	Обмежена (через бібліотеки)	Обмежена (через бібліотеки)
Архітектура	Асинхронна, подійно-орієнтована	Синхронна (Django), асинхронна (Flask)	Синхронна	Синхронна
Популярність	Висока	Висока	Середня	Висока
Продуктивність	Висока	Середня	Середня	Середня
Масштабованість	Висока	Висока	Висока	Висока

6

Рисунок Б.6 – Слайд №6

Вибір технологій розробки

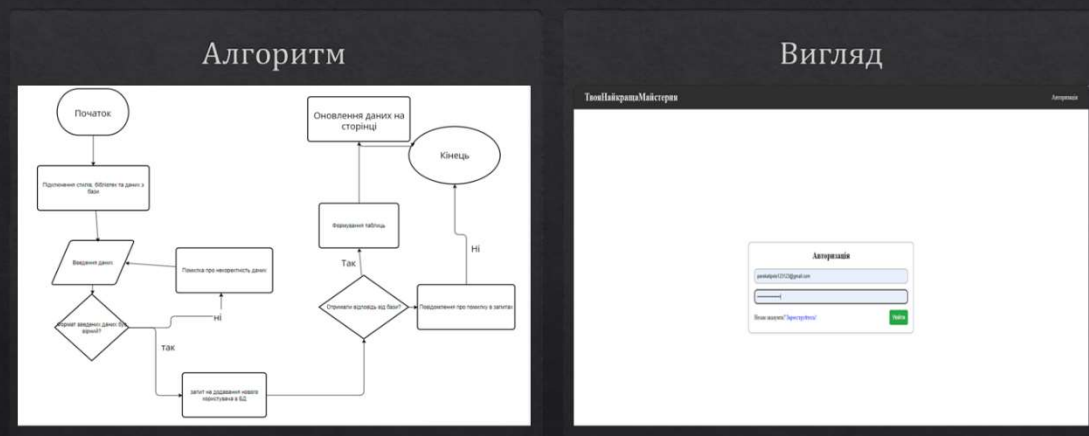
Для розробки вебзастосунку було обрано такі технології:

- HTML
- CSS
- JavaScript
- ReactJS
- NodeJS
- PostgreSQL

7

Рисунок Б.7 – Слайд №7

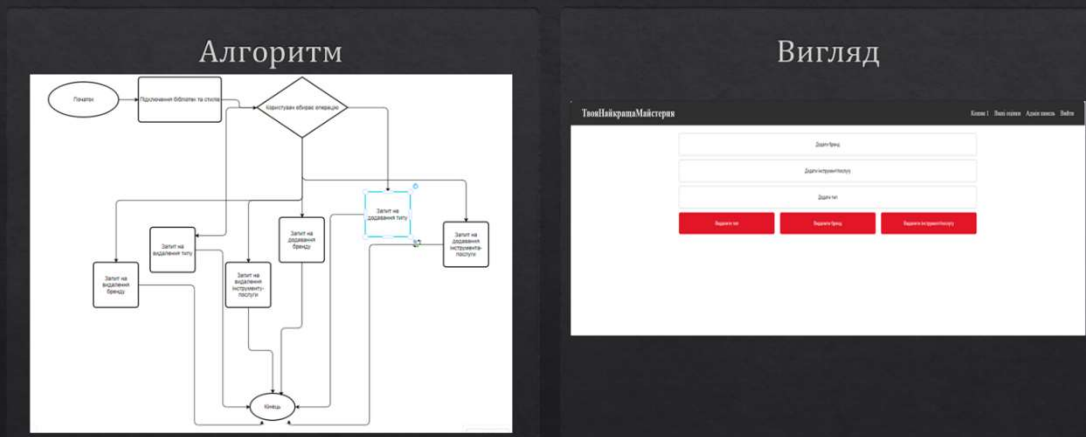
Сторінка авторизації



8

Рисунок Б.8 – Слайд №8

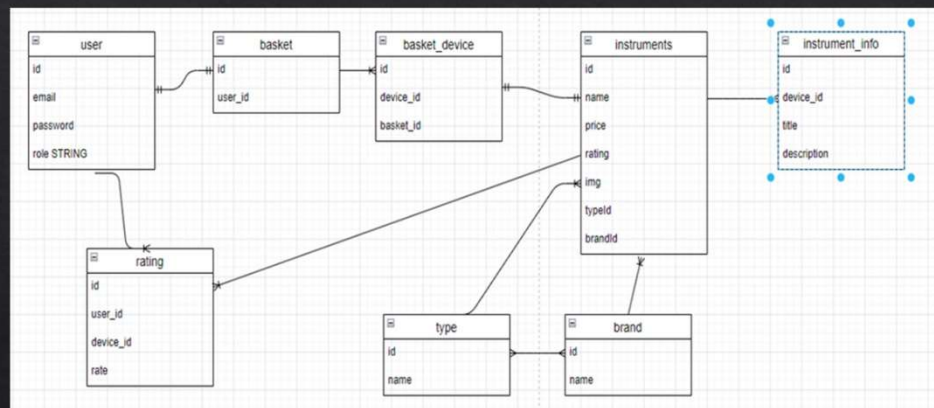
Сторінка адміністратора



9

Рисунок Б.9 – Слайд №9

Структура бази даних



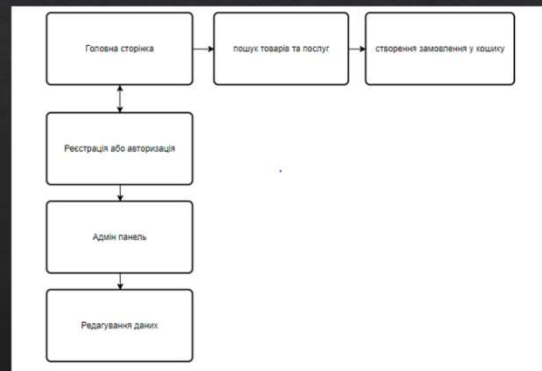
10

Рисунок Б.10 – Слайд №10

Структура вебзастосунку

Під час розробки застосунку були зроблені кілька логічних блоків, на які можна умовно поділити сайт:

- AuthComponent/index.jsx – логічний блок, у якому відбувається реєстрація або авторизація;
- BasketDeviceItem/index.jsx – логічний блок для роботи кошика;
- DeviceItem/index.jsx – блок для управління карти товару;
- AdminComponent/index.jsx – блок для управління сторінки адміністратора та внесення змін до бази даних.



11

Рисунок Б.11 – Слайд №11

Форма додавання даних до БД

Запити форми

- ♦ Executing (default): INSERT INTO "devices" ("id","name","price","rating","description","createdAt","updatedAt","typeId","brandId") VALUES (DEFAULT,\$1,\$2,\$3,\$4,\$5,\$6,\$7,\$8) RETURNING "id","name","price","rating","description","createdAt","updatedAt","typeId","brandId";
- ♦ Executing (default): INSERT INTO "device_photos" ("id","url","deviceId","createdAt","updatedAt") VALUES (DEFAULT,\$1,\$2,\$3,\$4) RETURNING "id","url","deviceId","createdAt","updatedAt";

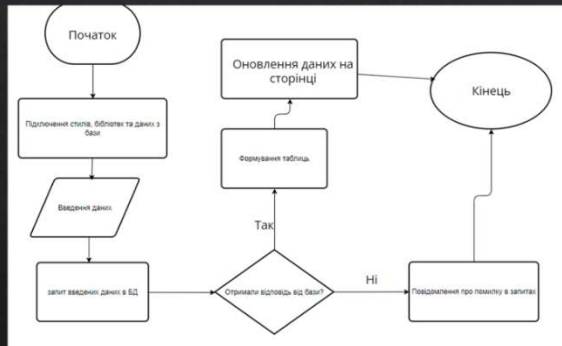
Вигляд

12

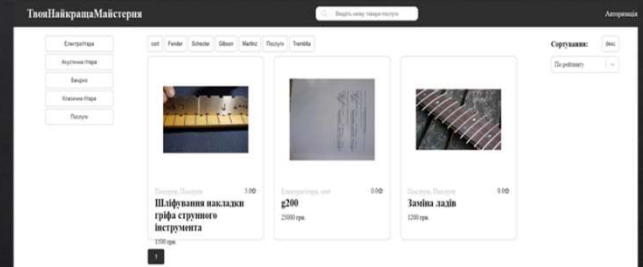
Рисунок Б.12 – Слайд №12

Головна сторінка вебзастосунку

Алгоритм



Вигляд сторінки



13

Рисунок Б.13 – Слайд №13

Висновок

- ✦ В результаті виконання дипломної роботи було проведено детальний аналіз вимог до вебзастосунку та його функціоналу. На основі цих даних було розроблено вебзастосунок майстерні музичних інструментів.
- ✦ Проведений аналіз включав:
 - ✦ – визначення потреб користувачів;
 - ✦ – аналіз аналогів;
 - ✦ – технічний аналіз вебзастосунку;
 - ✦ – аналіз результатів тестування.
- ✦ Розроблений вебзастосунок надає користувачам можливість швидко і комфортно замовляти музичні інструменти та послуги для них, використовуючи інтуїтивно зрозумілий, простий та зручний інтерфейс. Вебзастосунок може забезпечувати необхідним функціоналом користувачів для реєстрації, авторизації, вибору музичних інструментів, послуг майстра та замовлення інструмента.

14

Рисунок Б.14 – Слайд №14