КВАЛІФІКАЦІЙНА РОБОТА

КР.ІПЗ – 07.00.00.000 ПЗ

Група ІПЗс-2017

Купріянов В.Г.

2021

ЗАКЛАД ВИЩОЇ ОСВІТИ
УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних і прикладних наук
Кафедра інформаційних технологій

на правах рукопису

Купріянов Володимир Геннадійович

УДК 004.415

Розробка інтернет-ресурсу з продажу музичних інструментів за допомогою стеку технологій MERN

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття освітнього ступеню бакалавра

Науковий керівник
к.т.н., доц.
Пашкевич Олег Петрович

Івано-Франківськ – 2021

ЗВО «Університет Короля Данила»
Факультет суспільних і прикладних наук
Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»
Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«_____» _____ 202__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Купріянову Володимиру Геннадійовичу
(прізвище, ім'я, по-батькові)

1. Тема роботи
Розробка інтернет-ресурсу з продажу музичних інструментів за допомогою стеку технологій MERN

керівник роботи
Пашкевич Олег Петрович, кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «__» _____ 202_ року № ____

2. Строк подання студентом роботи
01.06.2021 р.

3. Зміст бакалаврської роботи (перелік питань, які потрібно розробити)
1. Основні тенденції розробки інтернет-аплікацій
2. Проектування та розробка інтернет ресурсу
3. Програмна реалізація інтернет ресурсу
4. Охорона праці

4. Дата видачі завдання
_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Основні тенденції розробки інтернет-аплікацій	25.03.2021	
2.	Проектування та розробка застосунку	15.04.2021	
3.	Програмна реалізація застосунку	15.05.2021	
4.	Формування висновків	23.05.2021	
5.	Охорона праці	25.05.2021	
6.	Оформлення пояснювальної записки	29.05.2021	
7.	Оформлення графічного матеріалу та підготовка до захисту роботи	01.06.2021	

Студент _____ _____
(підпис) **(прізвище та ініціали)**

Керівник роботи _____ _____
(підпис) **(прізвище та ініціали)**

Вихідні дані:

Мова програмування JavaScript, React.js, MongoDB, Express.js, Node.js, JWT, PayPal

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):

Сторінка	Опис граф. матеріалу	Сторінка	Опис граф. матеріалу
14	Головна сторінка JAZZ-CLUB-SERVICE	31	MVC модель
15	Категоризація на сайті JAZZ-CLUB-SERVICE	32	Головна сторінка
	Список товарів на сайті JAZZ-CLUB-SERVICE		
16	Сторінка окремого товару на сайті JAZZ-CLUB-SERVICE	35	Окремі елементи (товари)
	Покупка товару на сайті JAZZ-CLUB-SERVICE		
17	Кошик на сайті JAZZ-CLUB-SERVICE	36	Форма логінування
	Купівля в один клік на сайті JAZZ-CLUB-SERVICE		
18	Головна сторінка TOS.IN		Форма реєстрації
	Категоризація TOS.IN		
19	Список товарів TOS.IN	39	Сторінка окремого товару
	Сторінка окремого товару TOS.IN		Схожі товари
20	Кошик TOS.IN	40	Елемент в корзині
	Підтвердження замовлення TOS.IN		Корзина
24	Структура продукту	42	Модальне вікно ПейПал
			Історія покупок
26	Зразок документу з колекції Categories	43	Деталі окремої покупки
27	Зразок документу колекції Payments	44	Адмін панель
28	Зразок документу колекції Users	45	Меню категорій
29	Зразок документу колекції Products	46	Додавання товару
	Колекції даних у проекті		

Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 8
ВСТУП 9
РОЗДІЛ 1. ОСНОВНІ ТЕНДЕНЦІЇ РОЗРОБКИ ІНТЕРНЕТ-АПЛІКАЦІЙ 11
1.1 Поняття клієнт-серверної архітектури 11
1.2 Поняття інтернет-магазину та особливості його функціоналу 13
1.3 Розгляд аналогів продукту 13
Висновки до розділу 1 21
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНТЕРНЕТ РЕСУРСУ 22
2.1 Опис модулів і функціоналу сайту 22
2.2 Таблиця залежностей модулів 24
2.3 Проектування бази данних 26
Висновки до розділу 2 30
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРНЕТ РЕСУРСУ 31
3.1 Огляд структури проекту 31
3.2 Реалізація головної сторінки 32
3.3 Реалізація авторизації 35
3.4 Корзина та історія покупок 38
3.5 Реалізація панелі адміністратора 44
Висновок до розділу 3 47
РОЗДІЛ 4. ОХОРОНА ПРАЦІ 48
4.1 Умови роботи на робочому місці з комп’ютером 48
4.2 Мікроклімат та рівень іонізації повітря виробничого приміщення 50
4.3 Вимоги до безпечного користування комп’ютерною технікою 58

Висновок до розділу 4 58
ВИСНОВКИ 59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 60
ДОДАТОК 61
Код продукту 61

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

- API – прикладний програмний інтерфейс (Application Programming Interface);
- MERN – стек програмного забезпечення, що включає чотири технології з відкритим кодом: (MongoDB, Express.js, React та Node.js). Ці компоненти забезпечують наскрізну структуру для створення динамічних вебсайтів та вебдодатків;
- Back-end – код серверної сторони програми;
- CORS – Cross-Origin Resource Sharing (спільне використання ресурсів з різних джерел) механізм безпеки сучасних браузерів;
- CSS – каскадні таблиці стилів (Cascading Style Sheets);
- DOM – об'єктна модель документа (Document Object Model);
- Front-end – клієнтська сторона, будь-який компонент, яким керує користувач;
- HTML – мова розмітки гіпертексту (HyperText Markup Language);
- JS – мова програмування JavaScript;
- JSX – розширення до синтаксису JavaScript (JavaScript Syntax Extension) XML для JavaScript;
- JWT – стандарт токена доступу на основі JSON (JSON Web Token);
- MVC – архітектурний шаблон модель–вигляд–контролер (Model-view-controller);
- UI – інтерфейс користувача (user interface);
- БД – база даних;
- СКБД – система керування базами даних;
- Футер – нижня частина сторінки;
- Хук – функція, що дозволяє використовувати стан та інші можливості React без написання класу;

ВСТУП

Актуальність теми. У сучасному світі все більше людей починає надавати перевагу послугам, що надають інтернет-магазини. Можливість здійснювати покупки не виходячи з дому має тільки один практичний недолік – надійність. І навіть цей аспект часто зводиться до нуля, адже компанії змушені надавати людям гарантії якості та доставки, щоб вони були готові користуватися сервісом без страху бути обманутими. Розробку інтернет-магазину необхідно здійснювати згідно сучасних методолігії та підходів, тому що вони є необхідними для сучасних аплікацій. Відмова від сучасних підходів до розробки будь-якого програмного забезпечення може бути основною причиною потенційного провалу продукту на ринку.

Мета роботи. Розробка інтернет-ресурсу для продажу музичних інструментів з можливістю перегляду історії покупок, та збереження товарів в корзині, після того як клієнт залишив сайт, та можливістю здійснювати оплату використовуючи популярний сервіс PayPal.

Об’єкт роботи. Процес розробки сучасного вебсайту призначеного для постачання музичних інструментів.

Предмет роботи. Методика розробки інтернет-ресурсу згідно сучасних тенденцій розробки.

Завдання роботи. Відповідно до вибраної теми в роботі покладені такі задачі:

- пошук та аналіз уже існуючих застосунків аналогічного призначення;
- вибір мови програмування, технологій та інших суміжних програм;
- розроблення сучасного та зручного застосунку;
- проведення тестування продукту.

Методи роботи. Для вирішення поставленого завдання були використані: мова гіпертекстової розмітки HTML для розміщення блоків на сторінці,

каскадна таблиця стилів CSS для стилізації сторінкиии, MongoDB для створення та налаштування бази данних, мова програмування JavaScript та її бібліотека React.js для швидкого відображення компонетнів сторінки. Фреймворк Express, пакети данних cors, dotenv, bcrypt, cloudinary, jsonwebtoken(jwt), axios, paypal-express-checkout, cookie-parser, mongoose для роботи з базою в мові програмування JavaScript та середовище програмування Visual Studio Code. Розроблений проект залишає за собою можливість додавання у майбутньому нових можливостей та вдосконалення вже існуючого функціоналу та дизайну.

Результати роботи. Результатом дипломної роботи є інтернет-ресурс для продажу музичних інструментів з можливістю збереження історії про покупки здійснені з допомогою сервісу PayPal, та збереженням товарів в корзині при виході з сайту.

Структура роботи. Розділи – 4. Загальний обсяг основної частини – 55. Список використаних джерел містить – 16 позицій.

РОЗДІЛ 1. ОСНОВНІ ТЕНДЕНЦІЇ РОЗРОБКИ ІНТЕРНЕТ-АПЛІКАЦІЙ

1.1 Поняття клієнт-серверної архітектури

Для більшості сучасних інтернет-аплікацій дотримування клієнт-серверної архітектури є обов'язковим.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами;

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого.

Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів.

Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

Загальноприйнятим є положення, що клієнти та сервери – це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми – і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них;

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнтського та серверного.

В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.;

Недотримування цієї архітектури при розробці сучасної інтернет-аплікації може бути головною причиною провалу проекту.

1.2 Поняття інтернет-магазину та особливості його функціоналу

Інтернет-магазин являє собою спеціалізований вебсайт, який належить фірмі-товаровиробнику, торговій фірмі тощо та призначений для просування споживчих товарів на ринку, збільшення обсягів продажу, залучення нових покупців.

Характерними рисами Internet-магазинів є те, що вони можуть пропонувати значно більшу кількість товарів та послуг, ніж реальні магазини і забезпечувати споживачів значно більшим обсягом інформації, необхідної для прийняття рішення про покупку. Також завдяки використанню Internet-технологій є можливою персоналізація підходу до споживачів з врахуванням попередніх відвідувань магазину та зроблених в ньому покупок та використання Internet-магазину як ефективного способу маркетингових досліджень (анкетування, конференції покупців і т.п.).

Internet-магазини потребують значно менших витрат на утримання та організацію роботи, оскільки у ньому значно обмеженіша матеріально-технічна база (будівлі, споруди, приміщення) та кількість обслуговуючого персоналу.

Проте Internet-магазини мають і недоліки. Основними є невизначеність реального існування товару та відповідність його основним параметрам якості, шахрайства при проведенні грошових трансакцій, проблеми з доставкою.

Основними вимогами, які ставляться користувачами до Internet-магазину є:

- зрозумілий інтерфейс та зручна система навігації по магазину;
- зручна система посилань, що дозволяє оптимальним способом одержати необхідну користувачеві інформацію;
- мінімальна кількість дій користувача для здійснення покупки;

1.3 Розгляд аналогів продукту

Є не так вже і багато аналогів, що пропонують схожий сервіс для заданої

території. Тому для прикладу буде взяти більш маштабні сервіси, що розділяють спеціалізацію. Розглянемо два.

Ці два ресурси надають послуги значно більші, ніж просто продаж інструментів. Вони також спеціалізуються у пошуку уроків для початківців, реклами брендів та партнерів, ремонт, інше обладнення, на новини зі світу музики.

Першим розглянемо сайт JAZZ-CLUB-SERVICE. (https://jcs.ua/ua/)[14] (рис. 1.1)

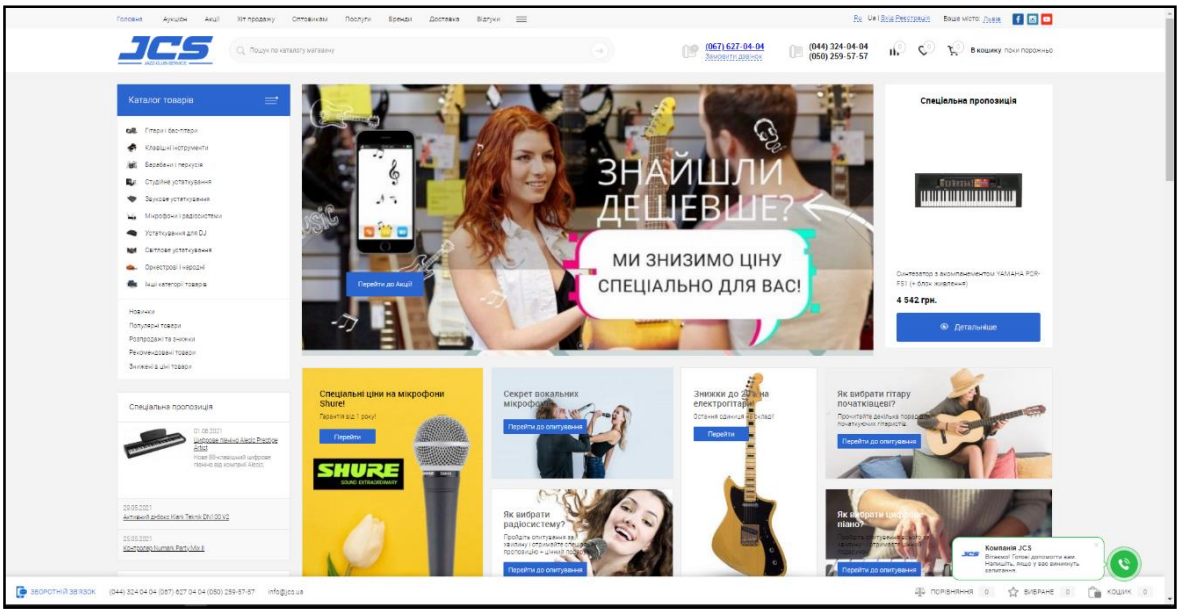

Рисунок 1.1 – Головна сторінка JAZZ-CLUB-SERVICE

Сервіс вітає користувачів яскравим інтерфейсом, на якому присутні більшість з важливих кнопок. Шапка сайту містить посилання на такі важливі секції, як новини, акції, відгуки про сервіс, та посилання для авторизації. Нижче присутнє поле для пошуку товарів по назві, логотип, та контактна інформація. Зліва можна побачити «гамбургер» меню, з ховер-ефектом для браузинга по різних категоріях. Зправа від меню можна побачити основну сторінку, що в принципі, заповнена посиланнями на цікаві матеріали, або акційні товари та пропозиції.

Велика концентрація реклами, та значна кількість згрупованих елементів

на сторінці може відштовхнути деяких користувачів. Це проблеми, які у додатку, що розробляється, будуть вирішені.

Меню категорії в розгорнутому вигляді складається з багатьох кнопок, при наведенні курсору мишки на них будуть з’являтися пов’язані підкатегорії. Кожна категорія має свою іконку, та кожна підкатегорія має фото товару, що її представляє. Приклад наведено на рисунку 1.2.

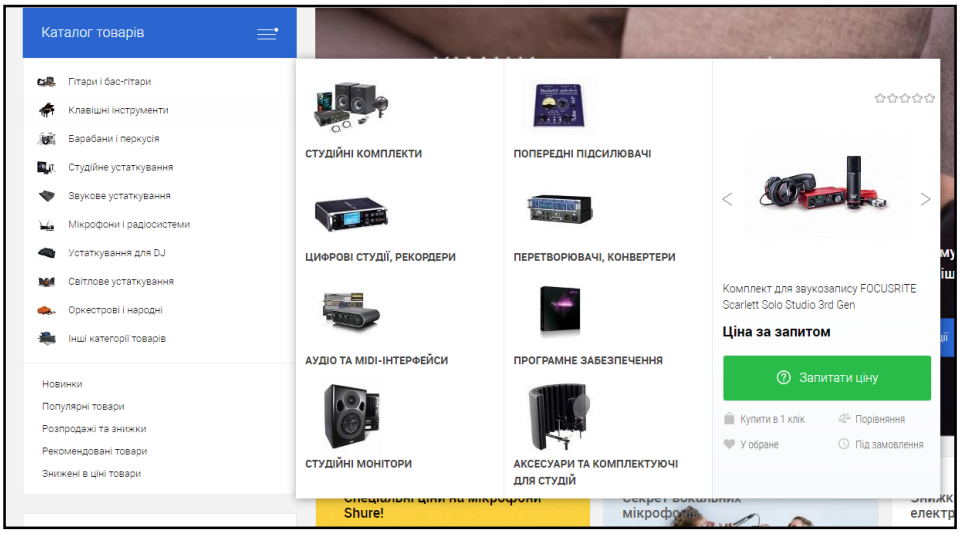

Рисунок 1.2 – Категоризація на сайті JAZZ-CLUB-SERVICE

Після вибору певної категорії, користувач попадає на сторінку з списком усіх товарів в заданій категорії. Виглядає вона так (рис. 1.3):

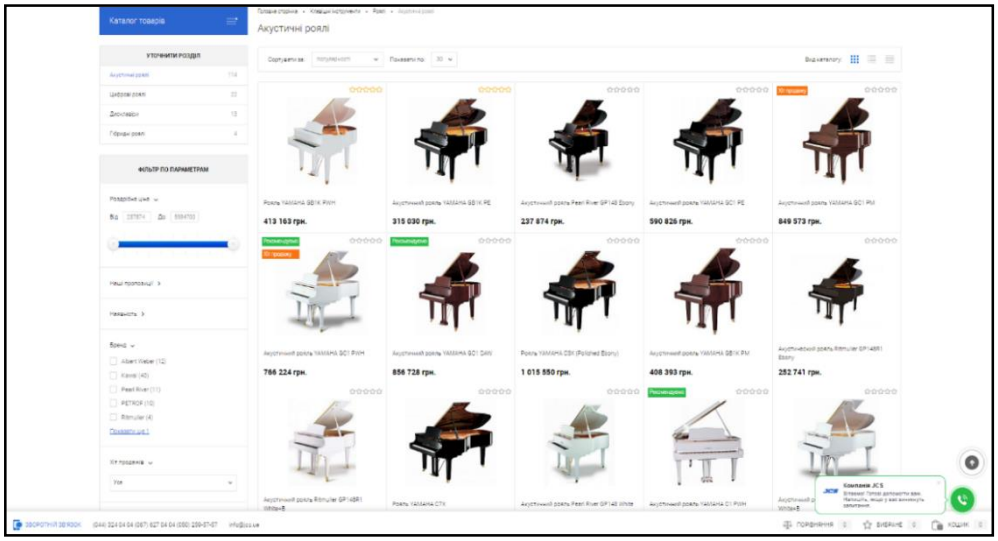

Рисунок 1.3 – Список товарів на сайті JAZZ-CLUB-SERVICE

Більше параметрів для сортування/фільтрування зліва та самі товари з цінником, фото, назвою, та позначками про унікальні пропозиції в центрі сторінки.

При виборі конкретного товару, користувач попаде на сторінку цього товару (рис. 1.4):

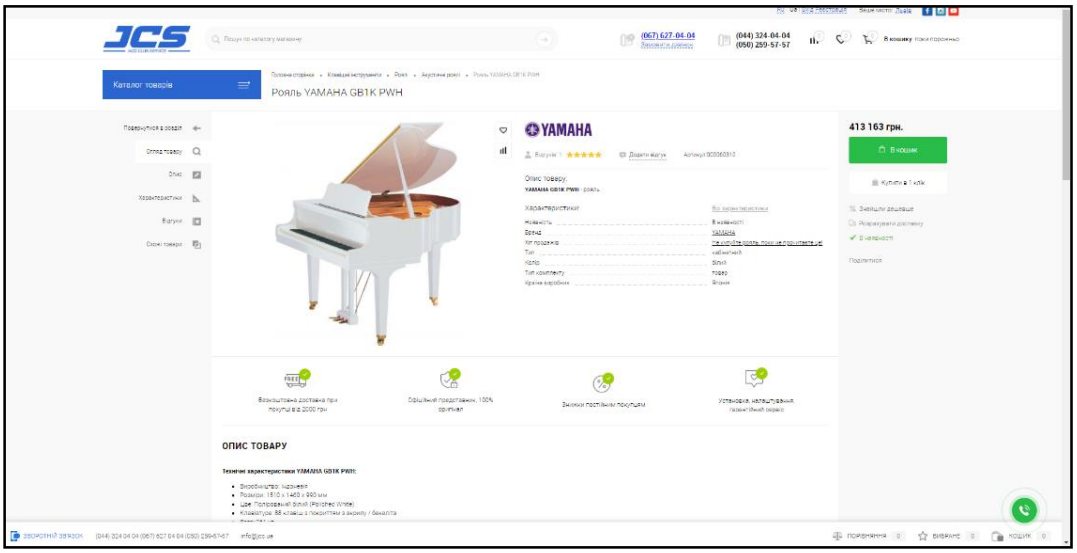

Рисунок 1.4 – Сторінка окремого товару на сайті JAZZ-CLUB-SERVICE

Тут зображено меню для швидкої навігації по секціях сторінки зліва, основну інформацію по центрі, секція з подібними товарами внизу сторінки, та меню для купівлі зправа.

При додаванні в кошик ми бачимо (рис. 1.5):

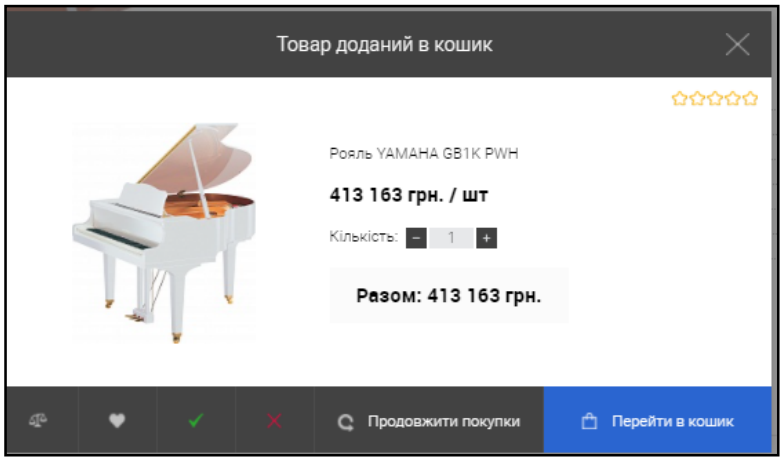

Рисунок 1.5 – Покупка товару на сайті JAZZ-CLUB-SERVICE

Після переходу в кошик, можна переглянути додані товари (рис. 1.6):

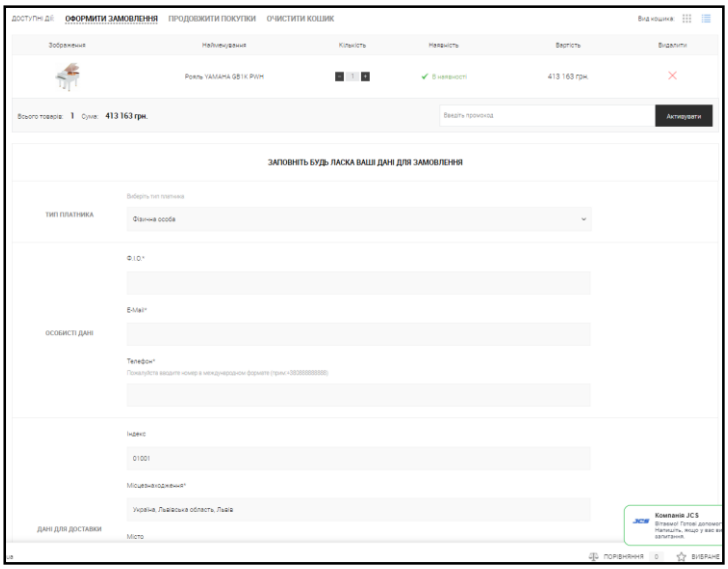

Рисунок 1.6 – Кошик на сайті JAZZ-CLUB-SERVICE

Для покупки є два способа. Перший – це заповнити дані про себе в формі нижче, та підтвердити замовлення. Другий – це налаштувати «купівлю в один клік». Для цього у відкритому меню потрібно ввести свої контактні дані, а далі з вами зв'яжуться напряму, для обговорення замовлення без потреби заповнення будь-яких форм на сайті (рис. 1.7).

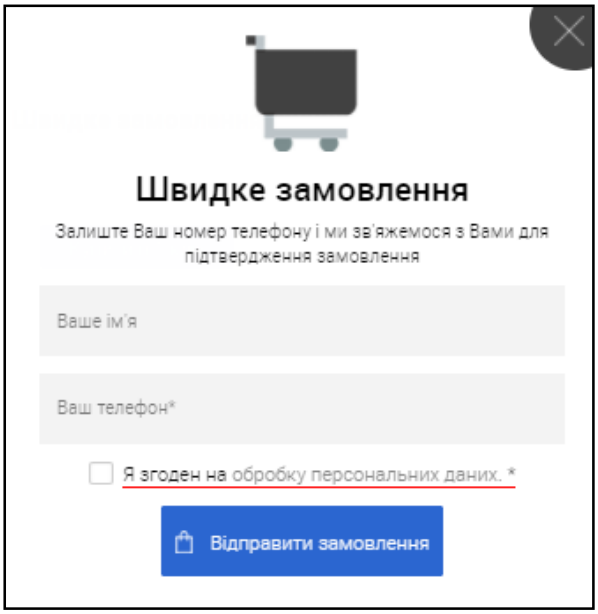

Рисунок 1.7 – Купівля в один клік на сайті JAZZ-CLUB-SERVICE

Для доповнення, швидко пробіжимося по ще одному аналогу сервісу, що розробляється.

Другим розглянемо TOS.IN. (https://tos.in.ua/)[13]

Сайт надає схожі послуги, тому краще, давайте швидко розглянути їх «зовні». Вдаватись в деталі займе деякий час, плюс, у цьому немає потреби, адже функціонал схожих модулів практично ідентичний.

Головна сторінка сайту (рис. 1.8):

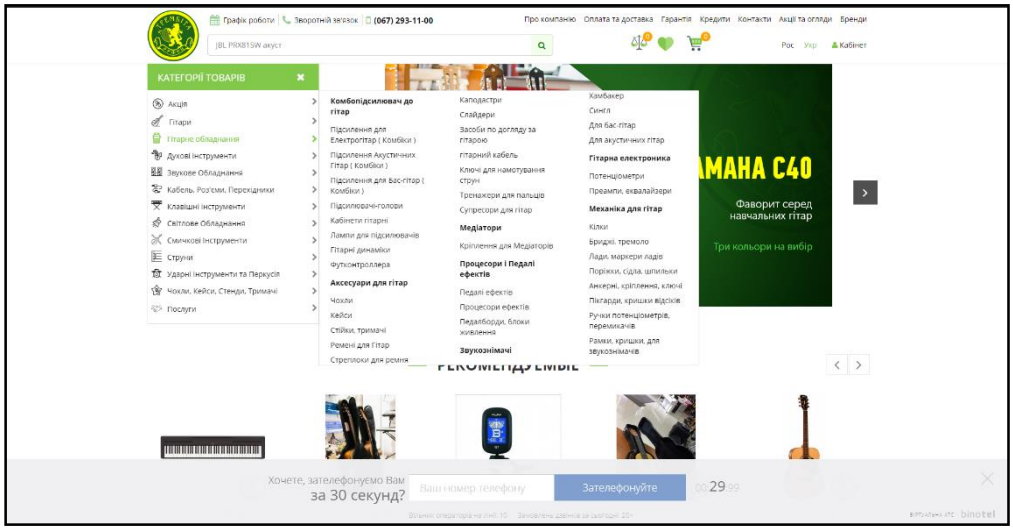

Рисунок 1.8 – Головна сторінка TOS.IN

Меню категорій (рис.1.9):

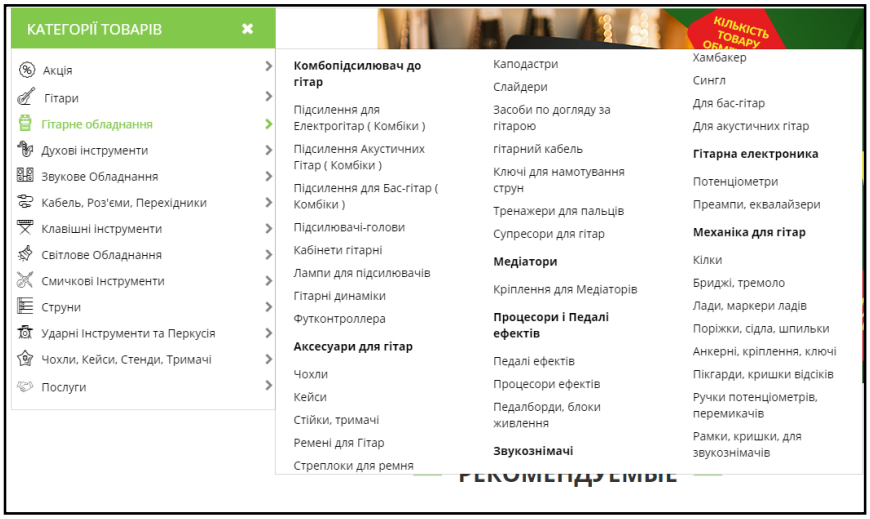

Рисунок 1.9 – Категоризація TOS.IN

Список товарів (рис. 1.10):

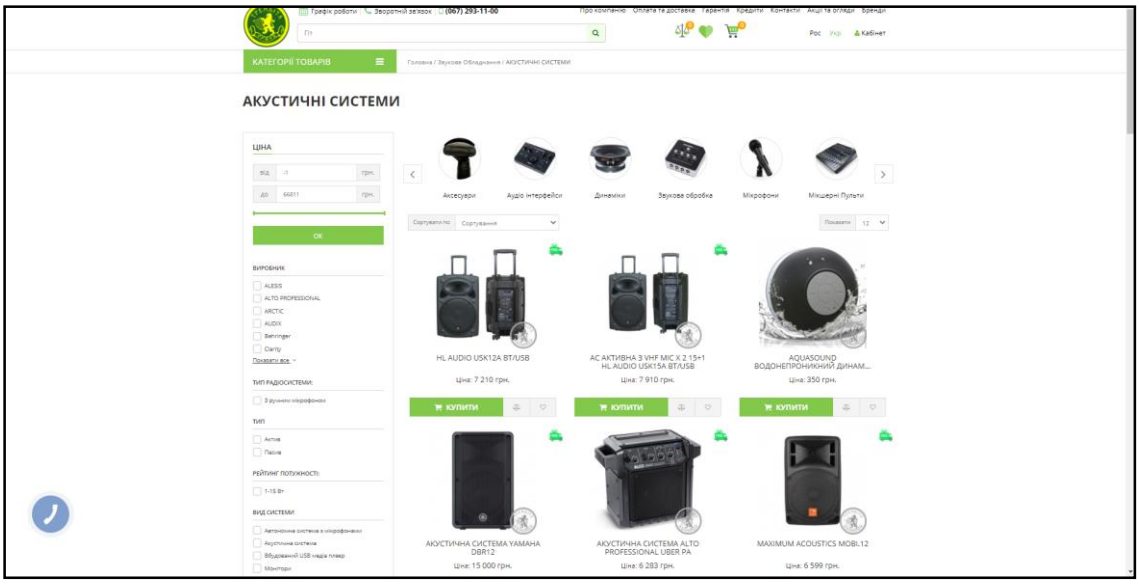

Рисунок 1.10 – Список товарів TOS.IN

Сторінка окремого товару (рис. 1.11):

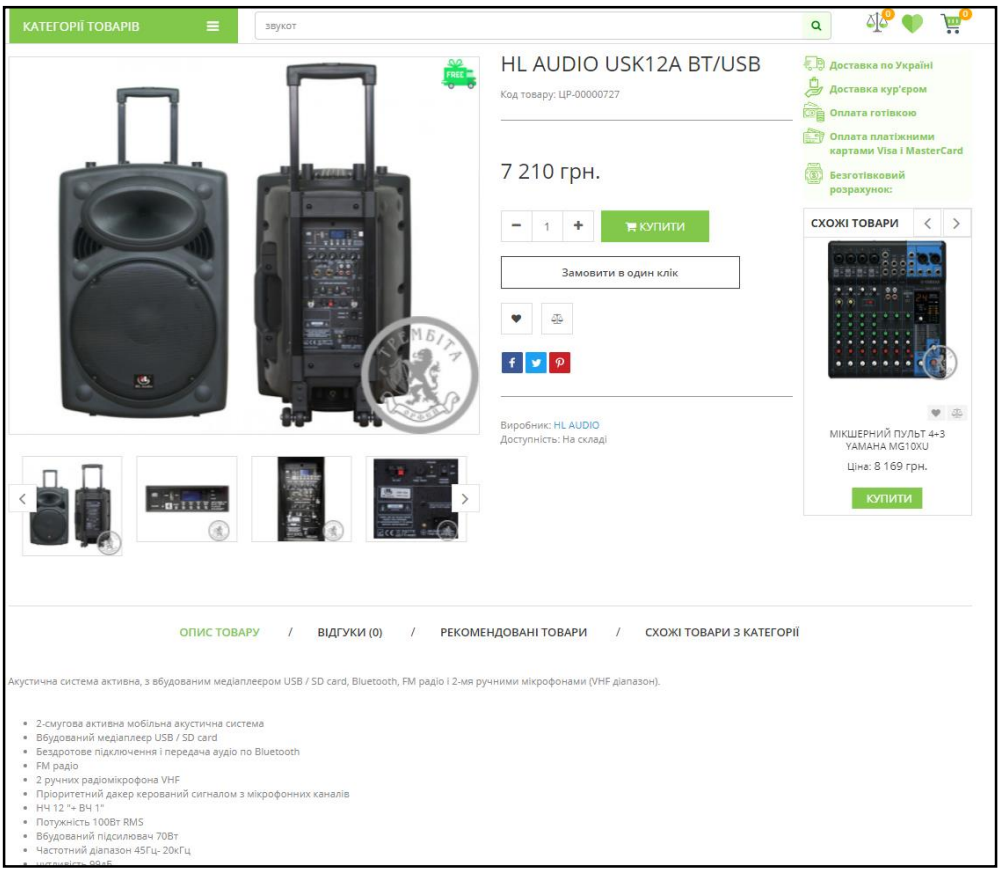

Рисунок 1.11 – Сторінка окремого товару TOS.IN

Кошик (рис. 1.12):

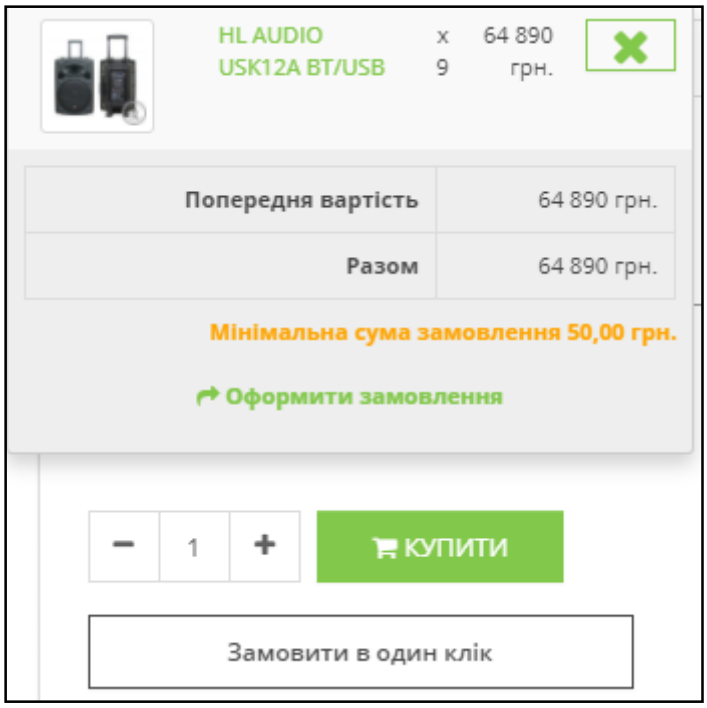

Рисунок 1.12 – Кошик TOS.IN

Підтвердження замовлення (рис. 1.13):

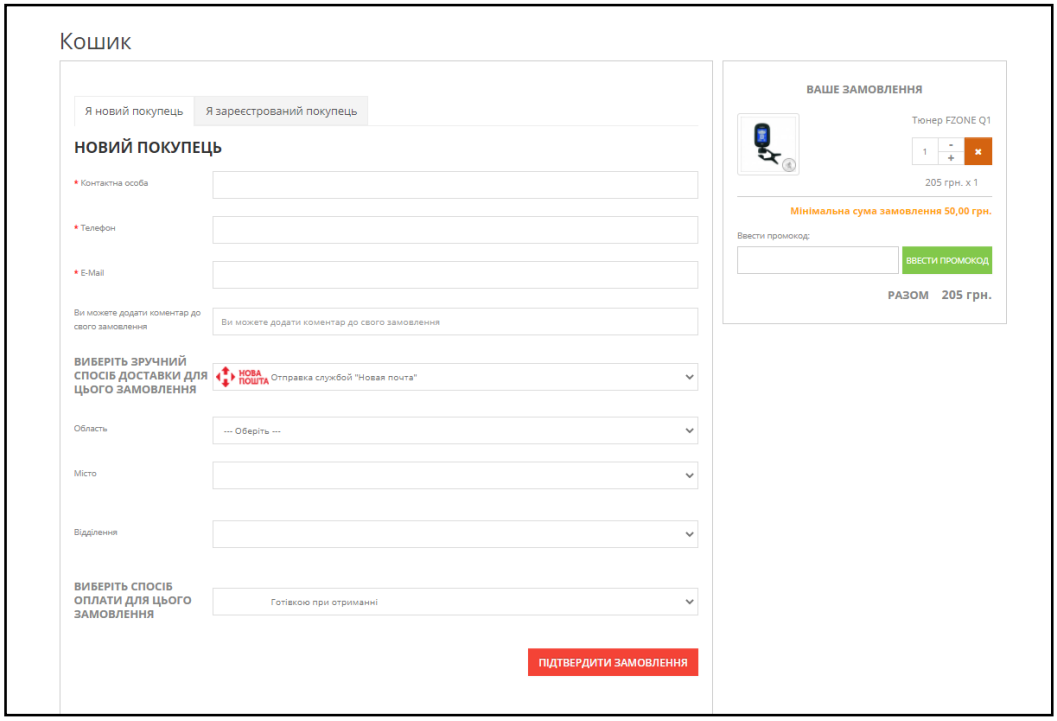

Рисунок 1.13 – Підтвердження замовлення TOS.IN

Висновки до розділу 1

Було розглянуто ключову архітектуру клієнт-сервер, наведено визначення інтернет-магазину, його переваги та недоліки. Розглянуто два ресурси-аналоги, їх функціонал, плюси та мінуси.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНТЕРНЕТ РЕСУРСУ

2.1 Опис модулів і функціоналу сайту

Розробка продукту буде відбуватися за допомогою стеку технологій MERN та декількох інших, менших бібліотек.

Переходячи до розробки ресурсу, буде виведено таблицю, де буде описано модулі та їх функіонал. У таблиці 2.1 буде зображенно:

– модуль;

– функціонал;

– короткий опис функціоналу;

Musicool**!** – платформа для продажу музичних інструментів. В порівнянні з аналогами, платформа значно менша, однак, у ній немає мінусів, присутніх у них.

Відсутність реклами, зрозумілий, мінімалістичний дизайн та навігація, дозволяють перейти до користування продуктом без попереднього ознайомлення з сайтом.

В принципі, дизайн та функціонал відповідає нормам, тому з мінімальними змінами в коді, дизайні та базі данних, сервіс може змінити своє призначення з продажу інструментів, на продаж чого завгодно.

Це добре для економічної складової сайту, оскільки при бажанні знадобиться мінімальна кількість часу і мінімальна кількість фінансових затрат для наступного оновлення, що правда це не відноситься до повної переробки сайту, так як при зміні структури сайту, прийдеться виділити набагато більше часу і грошей.

В кінцевому результаті буде отримано сучасний мінімалістичний дизайн, основний функціонал що потребує мінімальних дороблень, гнучкий інтерфейс для подальшої роботи дизайнера при потребі, та легко доповнюваний асортимент товарів.

Таблиця 2.1

Опис модулів проекту

Модуль	Функціонал	Опис
Головна сторінка	Сортування	Сортування товарів по певному критерію
	Додавання товару (Адмін)	Можливість добавити товар до БД.
	Додавання в кошик	Можливість придбати виставлений товар на головній сторінці
	Видалення	Можливість видалити товар з БД
	Пошук	Пошук товару по назві
	Фільтрування	Фільтрування по певному критерію
Профіль	Авторизація	Можливість користувачу авторизуватись
	Історія	Відображення історії здійснених покупок
	Корзина	Додавання різних товарів до корзини, щоб купити їх разом

Продовження таблиці 2.1

Корзина	Збільшення кількості	Збільшити кількість одного товару в покупці
	Видалення	Видалення товарів з корзини
	Покупка	Покупка товарів що знаходяться в корзині

Загалом, структура сайту продукту наступна (рис. 2.1):

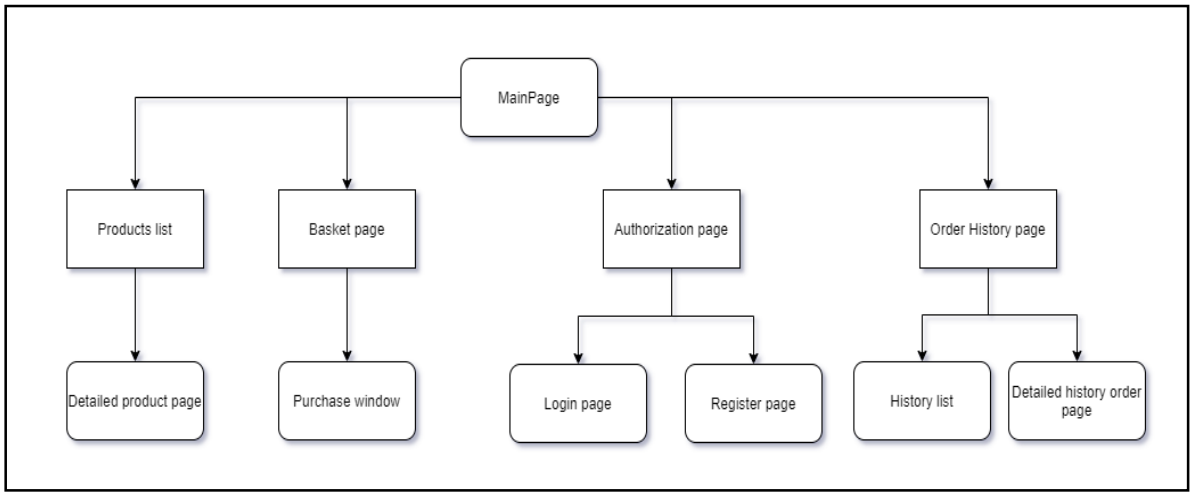

Рисунок 2.1 – Структура продукту

2.2 Таблиця залежностей модулів

Залежність модулів – пояснення чому необхідний той чи інший модуль по задумці автора, та як вони взаємодіють. Наведено це в таблиці 2.2.

Авторизація – модуль, що відповідає за автенфікацію користувача в системі.

Історія – лист з покупками авторизованого користувача за певний період часу.

Корзина – модуль що дозволяє переглянути тільки бажані товари, покупку яких можна оплатити разом.

Добавляння товарів – можливість постійного оновлення асортименту для

подальшої реалізації.

Видалення товарів – можливість видаляння товарів.

Список при бажанні в подальшому користуванні є легко доповнювальний.

Таблиця 2.2

Залежність модулів проекту

Модуль	Залежний модуль	Опис залежності
Головна сторінка	Авторизація	Можливість додавати в корзину, бачити історію покупок, зареєструватися чи залогінитися
	Корзина	
	Історія	
	Добавити(адміністратор)	Можливість додавати товари/категорію
	Видалити(адміністратор)	Можливість видаляти товари/категорію
Корзина	Авторизація	Необхідна для роботою з корзиною
	PayPal оплата	Можливість придбати товари в корзині
	Головна сторінка (Секція з товарами)	Відображення схожих товарів

Продовження таблиці 2.2

Історія	Авторизація	Відображення історії покупок конкретного користувача
	Головна сторінка (Секція з товарами)	Передача куплених товарів з корзини в історію
	PayPal	Передача куплених товарів в історію

2.3 Проектування бази данних

Використовуватись буде NoSQL БД – MongoDB. На офіційному сервісу хостингу БД MongoDB – MongoDB Atlas, було створено новий проект. Для цього проекту необхідно буде створити перший кластер, а в ньому першу БД.

Для коректної роботи усіх складових ресурсу, необхідні 4 різні колекції (таблиці) даних. Для зрозумілості їх було названо в довільному порядку:

– categories;
– cayments;
– users;
– products;

Categories – колекція з усіма існуючими категоріями товарів на ресурсі. Приймає дані (документи) у вигляді поданому на рисунку 2.2:

Рисунок 2.2 – Зразок документу з колекції Categories

Де:

– _id – автоматично згенерований СКБД для документа ІД;
– name – назва колекції, що задається адміністратором сайту чи БД;
– createdAt – автоматично згенерована дата створення документа в колекції;
– updatedAt – автоматично згенерована дата останньої зміни цього документа в колекції;

MongoDB дозволяє додавання документа любої структури в любу колекцію через середовище MongoDB Atlas, однак, зі сторони бек-енду, документи в колекції будуть добавлятися згідно заданого шаблону або моделі. Модель для цієї колекції виглядає наступним чином:

```
const categorySchema = new mongoose.Schema({
   name: {
      type: String,
      required: true,
      trim: true,
      unique: true
   }
}, {
   timestamps: true
})
```

Модель категорії для ручного вводу потребує тільки імені. Це поле в доменті повинно бути унікальним, і для зручності, система сама забере всі відступи на початку та в кінці.

Payments – колекція, документи якої, це записи про покупки користувачів. Її документи виглядають наступним чином (рис. 2.3):

Рисунок 2.3 – Зразок документу колекції Payments

– cart – масив даних, елементи якого – це товари придбані користувачем;
– status – ідентифікатор завершення транзакції. Стане правдивим в момент підтвердження користувачем отримання товару;
– user_id – ІД користувача, що знійснив покупку;
– name – ім’я користувача, що здійснив покупку;
– email – пошта користувача, що здійснив покупку;
– paymentID – ІД транзакції, автозгенерований системою ПейПал;
– address – дані про користувача системи ПейПал;

Users – колекція, що тримає дані про зареєстрованих користувачів сайту. Користувачі є двох видів: звичайні, та адміністратори.

Приклад документу подано на рисунку 2.4:

Рисунок 2.4 – Зразок документу колекції Users

– role – роль користувача. 0 = простий користувач, 1 = адміністратор;
– cart – масив товарів, що користувач добавив у корзину;
– name – ім’я користувача;
– email – пошта користувача;
– password – пароль користувача закодований за допомогою модуля bcrypt;

Products – колекція продуктів. Окремий продукт виглядає так (рис. 2.5):

```
_id: ObjectId("60b660401b0a0c3230236f10")
checked: false
sold: 13
product_id: "II MN Black"
title: "електрогітара fender player lead ii mn black"
price: 400
description: "Електрогітара, 6 струн, матеріал деки: вільха, матеріал і накладка гри..."
        "Особливості
content: Звукознімачі Player Series Alnico 5 Strat Single-Coil
        2-по..."
› images: Object
category: "60b656f7869e7f26a4c069b9"
createdAt: 2021-06-01T16:28:48.757+00:00
updatedAt: 2021-06-01T17:51:15.088+00:00
__v: 0
```


Рисунок 2.5 – Зразок документу колекції Products

– checked – параметр для адміністратора, характеризує чи вибраний товар для проведення над ним операцій;
– sold – кількість проданих товарів цього типу;
– product_id – ідентифікатор товару в колекції, не може повторюватись;
– title – назва товару;
– price – ціна товару;
– description – опис товару;
– content – характеристики, або більш детальний опис товару;
– images – об’єкт, що містить у собі посилання на фото, що демонструє товар. Хостингом для фото є сервіс Cloudinary;
– category – _id категорії до якої належить товар;

Загалом, ось як виглядають усі колекції в БД (рис. 2.6):

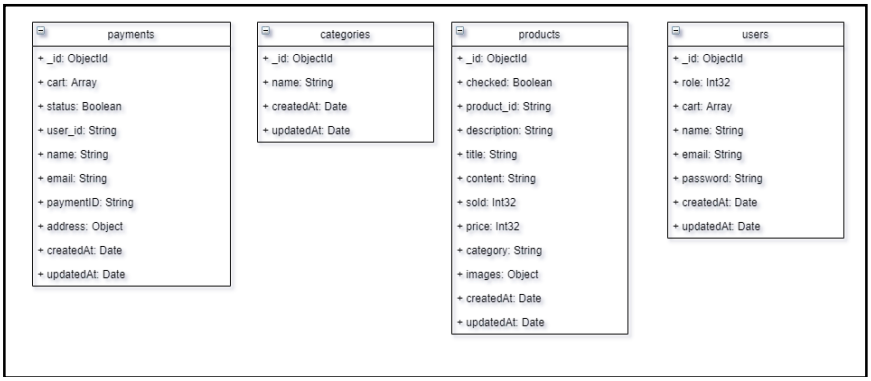

Рисунок 2.6 – Колекції даних у проекті

Висновки до розділу 2

Було розібрано структуру бази данних, доповнення що будуть використовуватися, та спроектовано модулі і їх залежності з коротким описом, що будуть реалізовані у проекті. Як БД було обрано MongoDB. Мовою програмування було обрано JavaScript та її бібліотеку React, що є одним з найбільш популярних рішень у наш час.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРНЕТ РЕСУРСУ

3.1 Огляд структури проекту

Для організації проекту було використано MVC підхід. Він полягає в розділенні функціоналу проекту на три ізольовані частини: Model-View-Controller. Model відповідає за збереження даних, ці дані відображає View. Користувач бачить View і за допомогою GUI, що надає View, подає команди на операції над Model. Ці операції описані в Controller, який модифікує Model.

Схема взаємодії виглядає наступним чином (рис. 3.1):

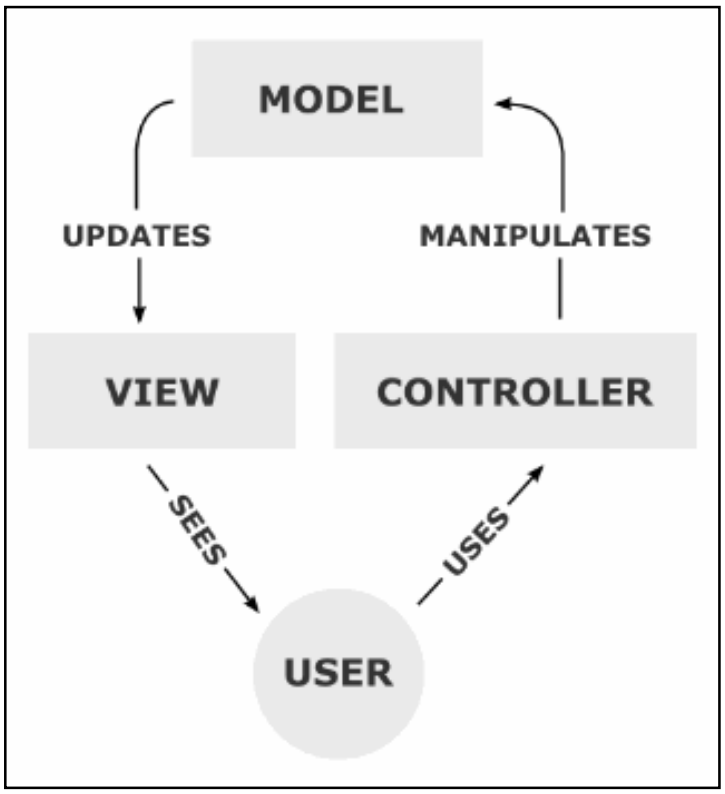

Рисунок 3.1 – MVC модель

Проект умовно поділено на дві частини, серверна(бек-енд), та клієнтська(фронт-енд). Серверна частина відповідає за:

– опис моделі даних для БД;

– шляхи(роути) для API, з якими може працювати проект;
– контроллери для цих роутів, що задають функціонал для окремих REST-запитів;
– запускає сервер;
– налаштовує зв’язок та взаємодію з БД;
– налаштовує необхідні middleware;

Клієнтська частина відповідає за:

– надання зручного UI для взаємодії з бек-ендом;
– відмальовку елементів DOM;

3.2 Реалізація головної сторінки

Головна сторінка виглядає наступним чином (рис. 3.2):

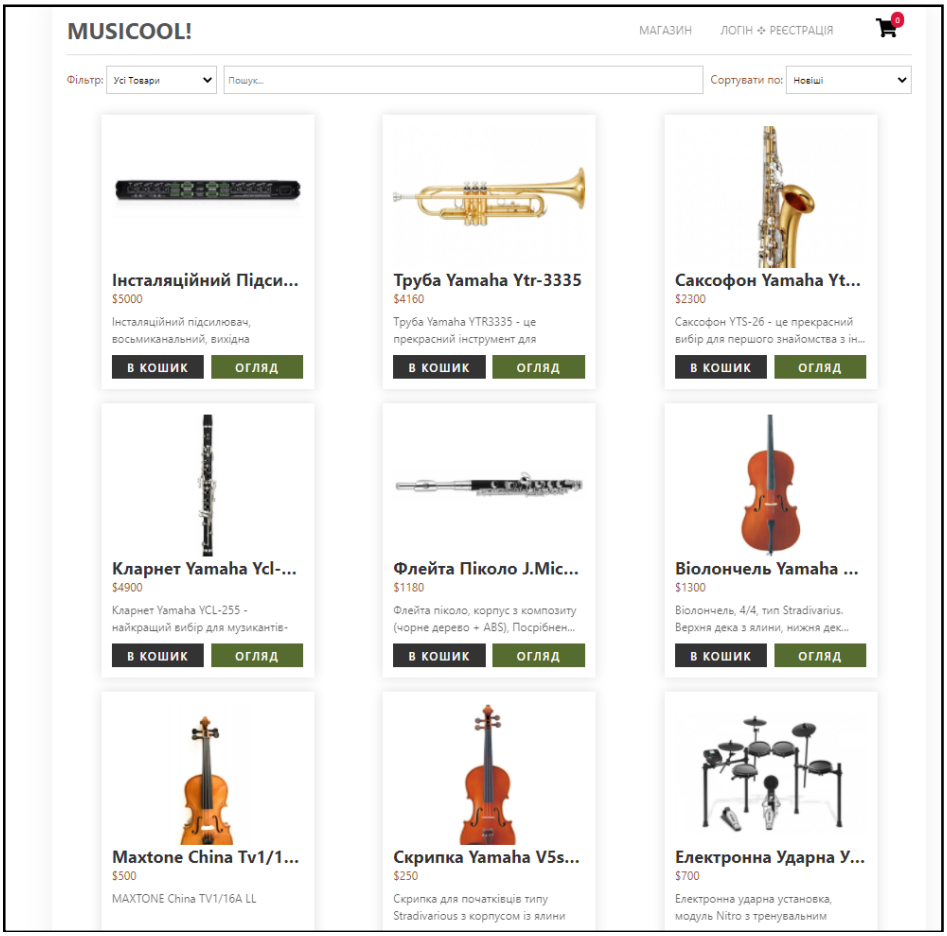

Рисунок 3.2 – Головна сторінка

На головній сторінці можна побачити шапку, яка буде спільна для всіх секцій сайту, і буде різнитися тільки в залежності від таких факторів як:

– чи є користувач адміністратором;

– чи є користувач залогіненим;

Нижче можна побачити поле, в якому зручно розташовані меню фільтрування, пошуку та сортування. Центральна частина сторінки, це список товарів, що належать до вибраної категорії.

Реалізацію будемо розглядати як з боку бек-енду, так і фронт-енду.

З боку бек-енду, потрібно зазначити, що функціонал відображення товарів описаний у одному з контроллерів продуктів. Цей функціонал, автоматично запускається на виконання, коли користувач переходить на головну сторінку, що відповідає базовій адресі «/».

```
router.route('/products')
    .get(productCtrl.getProducts)
    .post(auth, authAdmin, productCtrl.createProduct)


router.route('/products/:id')
    .delete(auth, authAdmin, productCtrl.deleteProduct)
    .put(auth, authAdmin, productCtrl.updateProduct)
```

Контроллер у цьому випадку productCtrl. Метод що викликається для відображення товарів – це запит get.

Як можна побачити, це єдиний з методів API продуктів, що не потребує ніякого middleware для роботи. Щоб не вдаватись в деталі, middleware auth та authAdmin відповідають за перевірки, чи є користувач авторизаваним, і чи є користувач адміністратором відповідно.

Метод про який було сказано це getProducts.

```
const productCtrl = {
    getProducts: async(req, res) =>{
        try {
            const features = new APIfeatures(Products.find(), req.query)
            .filtering().sorting().paginating()

            const products = await features.query

            res.json({
                status: 'success',
```

```
            result: products.length,
            products: products
        })

    } catch (err) {
        return res.status(500).json({msg: err.message})
    }
```

getProducts – асинхронний метод, що вибирає всі документи колекції Products, застосовує на результат методи фільтрування, сортування, та пагінації, та повертає відповідь, що містить статус, який дорівнює успіху, результат що дорівнює кількості товарів, та продукти, що є самими товарами.

Сам по собі, цей метод багато не дасть, оскільки без фронт-енду, єдиний спосіб викликати його це через спеціальний софт, такий як програма PostMan.

На фронт-енді, потрібно викликати цей метод, та відобразити продукти.

```
useEffect(() =>{
    const getProducts = async () => {
        const res = await axios.get(`/api/products?limit=
${page*9}&${category}&${sort}&title[regex]=${search}`)
        setProducts(res.data.products)
        setResult(res.data.result)
    }
    getProducts()
},[callback, category, sort, search, page])

}
export default ProductsAPI
```

Частина коду, що найбільш цікава, це хук useEffect. Цей хук буде виконуватись кожен раз як сторінка загрузиться, або хоча б одне зі значень, у переданому йому масиві, зміниться.

За допомогою бібліотеки axios, викликає метод описаний на бек-енді, здійснюючи get-запит на адресу роута, з якою контроллер зв’язаний. Таким чином, після виконання запиту, стейту продукти присвоюються продукти, які були отримані як відповідь, а стейту результати присвоюється кількість продуктів.

Залишилось це все зобразити в UI.

```
const deleteProduct = async(id, public_id) => {
    try {
        setLoading(true)
```

```
const destroyImg = axios.post('/api/destroy', {public_id},{
        headers: {Authorization: token}
})

const deleteProduct = axios.delete(`/api/products/${id}`, {
        headers: {Authorization: token}
})
```

Кожен продукт відмальовується як окремий компонент ProductItem, і разом вони відмальовуються на сторінці. ProductItem – це компонент, що зображений у вигляді блоку, та містить дані про товар. Відмальовка, або рендер, відбувається у методі return. Синтаксис цієї частини коду нагадує HTML, і це не дарма. Синтаксис бібліотеки React – JSX, дозволяє писати HTML-розмітку разом з скриптами JavaScript, прямо в тегах HTML.

Приклад наведено на рисунку 3.3.

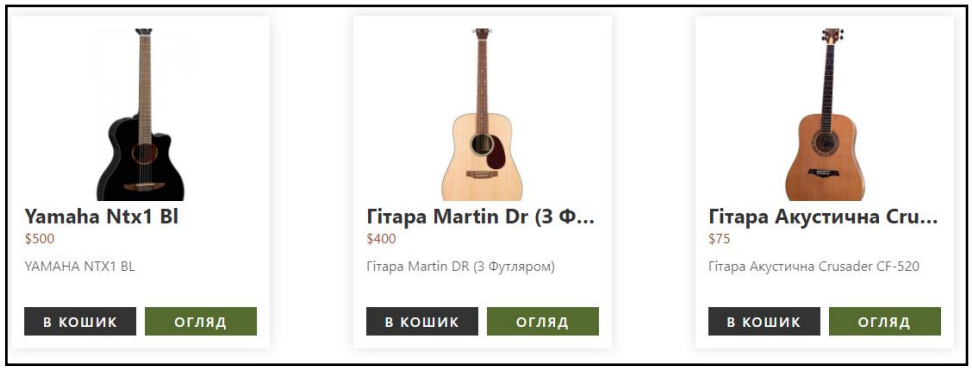

Рисунок 3.3 – Окремі елементи (товари)

3.3 Реалізація авторизації

На цьому етапі, для отримання можливості використовувати сайт по максимуму, користувач буде змушей авторизуватись. Для цього користувачу необхідно тільки ввести, своє ім’я, пошту та пароль.

Пароль повинен містити більше 6 символів, пошта повинна відповідати стандарту. На ім’я чіткої норми не встановлено, принаймі у поточній версії продукту.

Форма реєстрації наведена на рисунку 3.4.

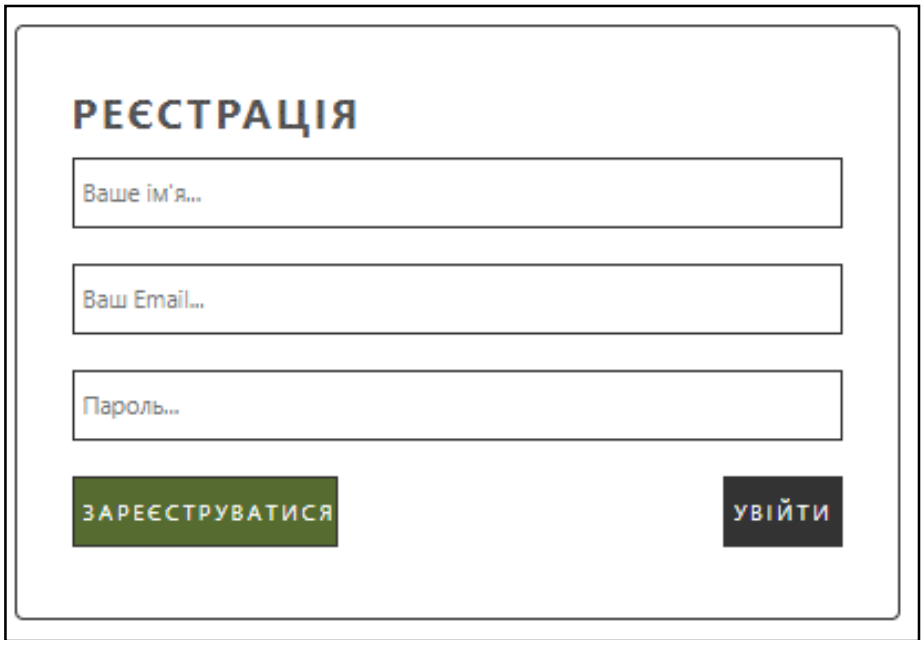

Рисунок 3.4 – Форма реєстрації

Форма входу ідентична, за вийнятком відсутності поля для імені (рис. 3.5).

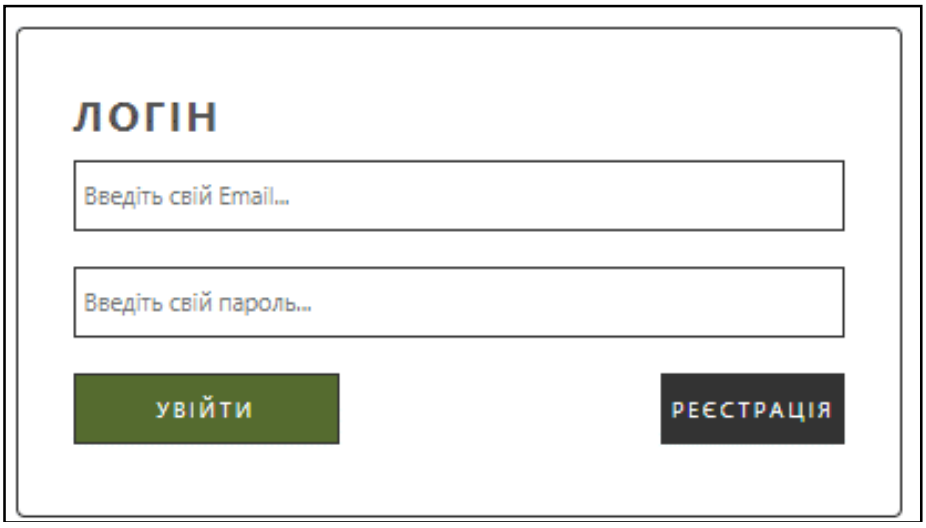

Рисунок 3.5 – Форма логінування

Роль фронт-енду у цій частині порівняно не велика. Клієнтська частина на цьому етапі здійснює тільки базову перевірку правильності заповнення полів, надсилає дані з форми на відповідний контроллер, та записує у LocalStorage браузера користувача змінну, для підтвердження того, що користувач авторизований.

Цікавіші речі відбуваються на бек-енді. На ньому висить відповідальність

за:

- звірення з БД для провірки наявності користувача;
- кодування паролю в цілях захисту;
- створення нового користувача згідно моделі відповідної колекції БД;
- збереження користувача в БД;
- надсилання браузеру в формі cookie JWT токена, для автенфікації користувача при його подальших діях на сайті;

В контроллері користувача для цього є два методи, для реєстрації та для логіну.

Для реєстрації код наступний:

```
register: async (req, res) =>{
      try {
        const {name, email, password} = req.body;
        const user = await Users.findOne({email})
        if(user) return res.status(400).json({msg: "Пошта уже зареєстрована!"})
        if(password.length < 6)
        return res.status(400).json({msg: "Пароль повинен містити
хоча 6 символів!"})
      } catch (err) {
        return res.status(500).json({msg: err.message})
      }
   },
```

Наступна частина коду для логіну:

```
login: async (req, res) =>{
      try {
         const {email, password} = req.body;
         const user = await Users.findOne({email})
         if(!user) return res.status(400).json({msg: "Користувача не знайдено!"})
         const isMatch = await bcrypt.compare(password, user.password)
         if(!isMatch) return res.status(400).json({msg: "Хибний пароль!"})
      } catch (err) {
         return res.status(500).json({msg: err.message})
}
```

Як можна побачити з коду, спільним для обох методів є зчитування даних з «тіла» запиту, пошук відповідного користувача в БД, та присвоєння JWT токена.

Для реєстрації проводиться кодування паролю, і закодований пароль надсилається в БД.

```
const passwordHash = await bcrypt.hash(password, 10)
        const newUser = new Users({
           name, email, password: passwordHash
        })
```

Для логіну тільки звіряється чи відповідає пароль, введений в формі на сайті, закодованому паролю, що зберігається в БД.

```
const isMatch = await bcrypt.compare(password, user.password)
        if(!isMatch) return res.status(400).json({msg: "Хибний пароль!"})
```

Обидва методи повертають токен, як знак того, що процесс завершився успішно.

```
res.cookie('refreshtoken', refreshtoken, {
            httpOnly: true,
            path: '/user/refresh_token',
            maxAge: 7*24*60*60*1000 })
        res.json({accesstoken})
```

Цей токен надалі використовується для підтвердження особи під час користування сайтом.

3.4 Корзина та історія покупок

Для покупки товару, його спершу потрібно добавити в корзину. Додати в корзину товар можна з головної сторінки, або зі сторінки окремого товару (рис. 3.6). В обох ситуаціях, присутня кнопка для цього. На сторінці окремого товару також можна переглянути схожі товари (рис. 3.7).

Значну частину сторінки буде займати фото товару, розташоване вгорі. Нижче, легко побачити назву товару. Зправа від назви, показано унікальний ідентифікатор товару. Нижче назви, наведений опис та характеристики товару, можна побачити скільки раз товар був придбаний, і звісно, сама кнопка, що добавляє товар в корзину.

Додавання товару в корзину (рис. 3.8) є необхідністю, тому що на даному етапі проекту, покупка здійснюється тільки з меню корзини, з використанням сервісу ПейПал.

Товари в корзині (рис. 3.9) розташовані вертикально, та займають

однакову кількість місця.

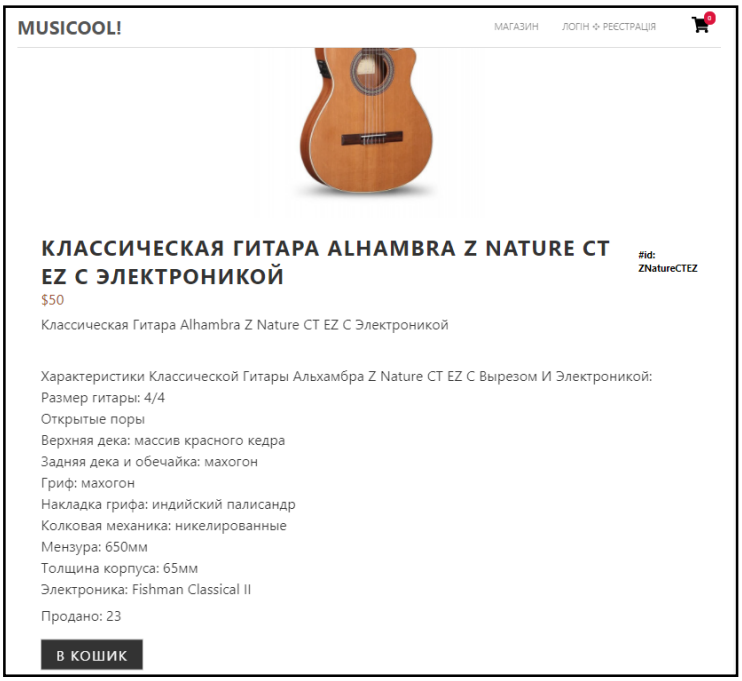

Рисунок 3.6 – Сторінка окремого товару

Кнопкою «В КОШИК» товар добавляється в кошик, а одже, і в базу даних, в масив cart, що присутній у кожному документі користувача.

Внизу сторінки, присутня секція зі схожими товарами (рис. 3.7).

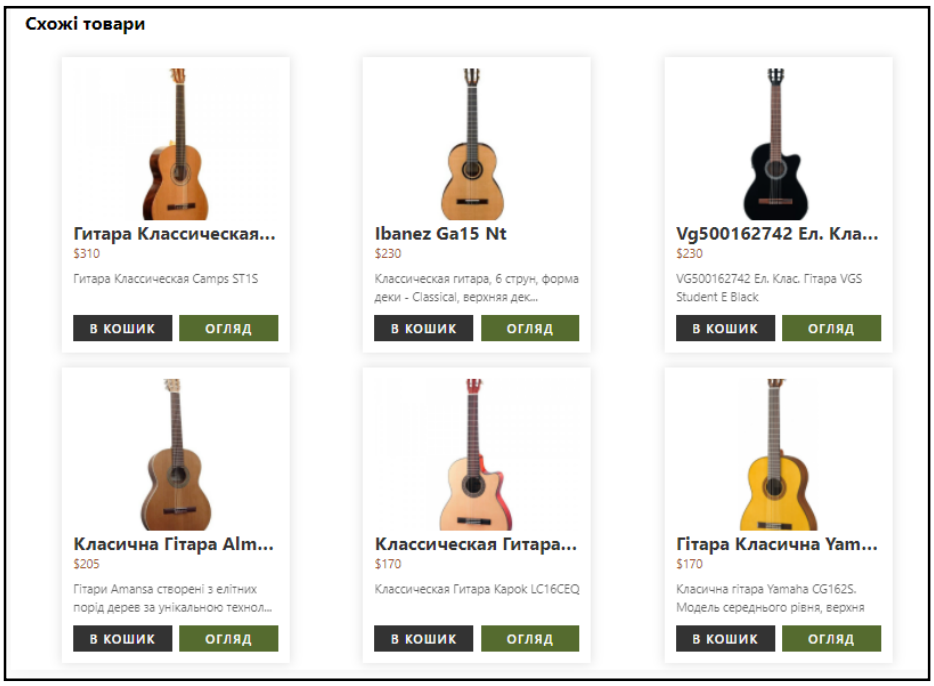

Рисунок 3.7 – Схожі товари

Окремий елемент в корзині виглядає наступним чином:

Рисунок 3.8 – Елемент в корзині

Товари в корзині можна видалити з корзини, або збільшити кількість копій окремого товару. Бек-енд реагує на товари на яких натиснули користувачі, і продукт з відповідним ІД додається в корзину користувача.

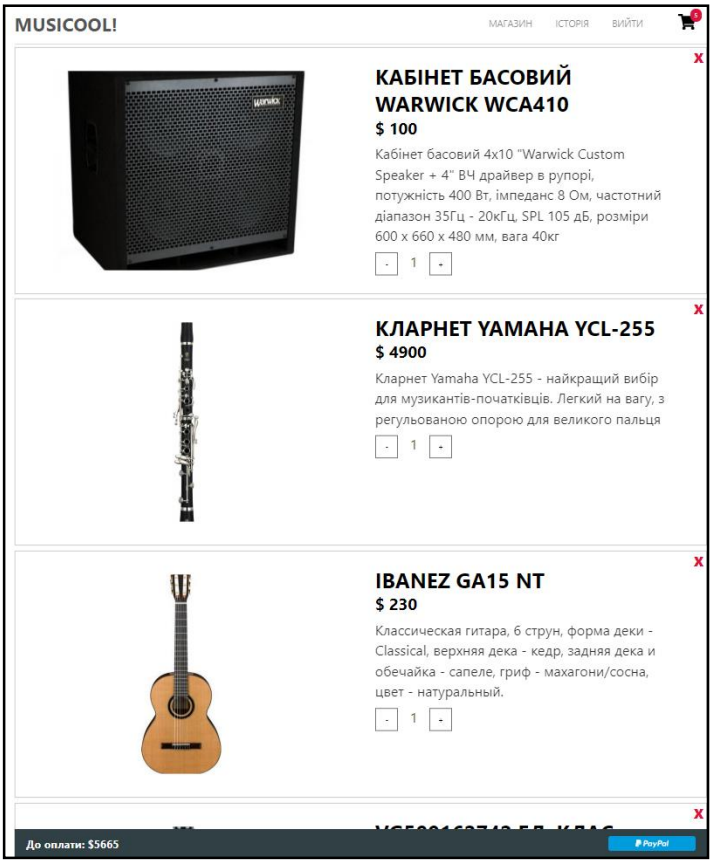

Рисунок 3.9 – Корзина

```
addCart: async (req, res) =>{
        try {
            const user = await Users.findById(req.user.id)
            if(!user) return res.status(400).json({msg: "Користувача не знайдено!"})
            await Users.findOneAndUpdate({_id: req.user.id}, {
                cart: req.body.cart
            })
            return res.json({msg: "Додано в кошик!"})
        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
},
```

На фронт-енді, проводиться перевірка, чи товар з цим ІД уже є в корзині. Якщо немає, товар додається в корзину, і дані про цю корзину віддаються на бек-енд, який цю корзину оновлює в базі.

```
const addCart = async (product) => {
        if(!isLogged) return alert("Авторизуйтесь для шопінгу!")
        const check = cart.every(item =>{
            return item._id !== product._id
        })
        if(check){
            setCart([...cart, {...product, quantity: 1}])

      await axios.patch('/user/addcart', {cart: [...cart, {...product, quantity: 1}]}, {
                headers: {Authorization: token}
            })
        }else{
            alert("Товар уже в корзині!")
        }
}
```

Покупка проводиться за допомогою сервісу ПейПал.

Після натиску відповідної кнопки, відкриється модальне вікно цієї системи. Тут користувач може авторизуватись, і підтвердити покупку згідно правил системи.

ПейПал, з нашого боку, отримає дані про загальну суму, яку необхідно сплатити, усі інші дані беруться з системи.

Наша система немає доступу до жодних даних, що вказуються користувачем у цій системі, вона отримує тільки результат транзакції, де буде вказано покупця, суму, та придбані товари.

Модальне вікно виглядає наступним чином (рис. 3.10):

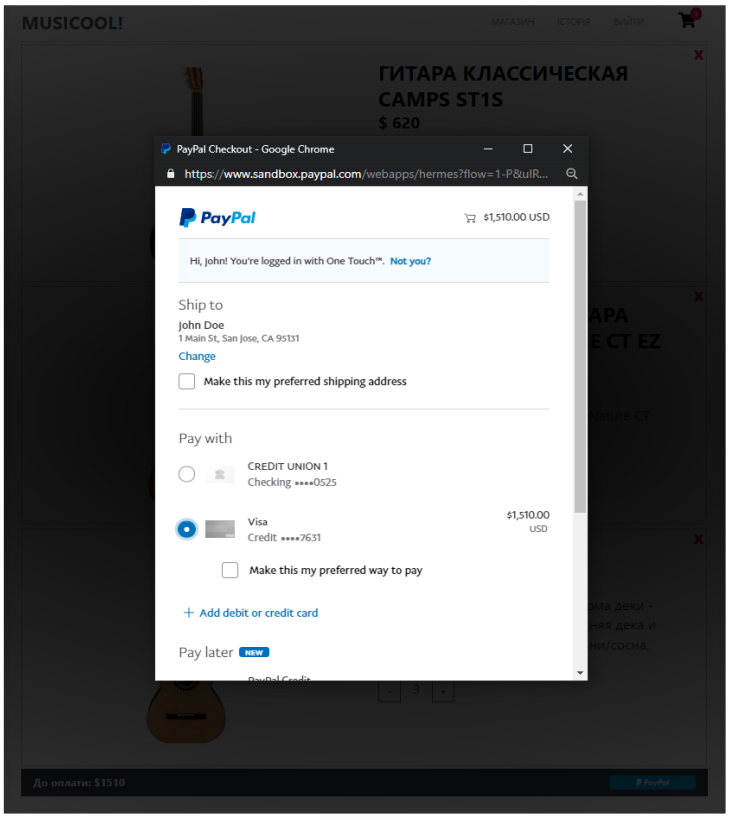

Рисунок 3.10 – Модальне вікно ПейПал

(Для демонстраційних цілей було використано сендбокс режим сервісу, щоб надати мені можливість начебто здійснювати покупку. В production режимі, сервіс працюватиме як задумано).

Після покупки, в історію буде добавлений новий запис, в якому можна переглянути що було куплено, на яку суму і т.д.

В базовому вигляді, на сторінці історії покупок буде показано дату покупки, та спеціальний ідентифікатор, що створює ПейПал для кожної транзакції що відбувається в системі. Приклад зображено на рисунку 3.11.

Рисунок 3.11 – Історія покупок

По натиску на кнопку «Переглянути» буде відкрито детальне представлення покупки. Деталі зображені на рисунку 3.12.

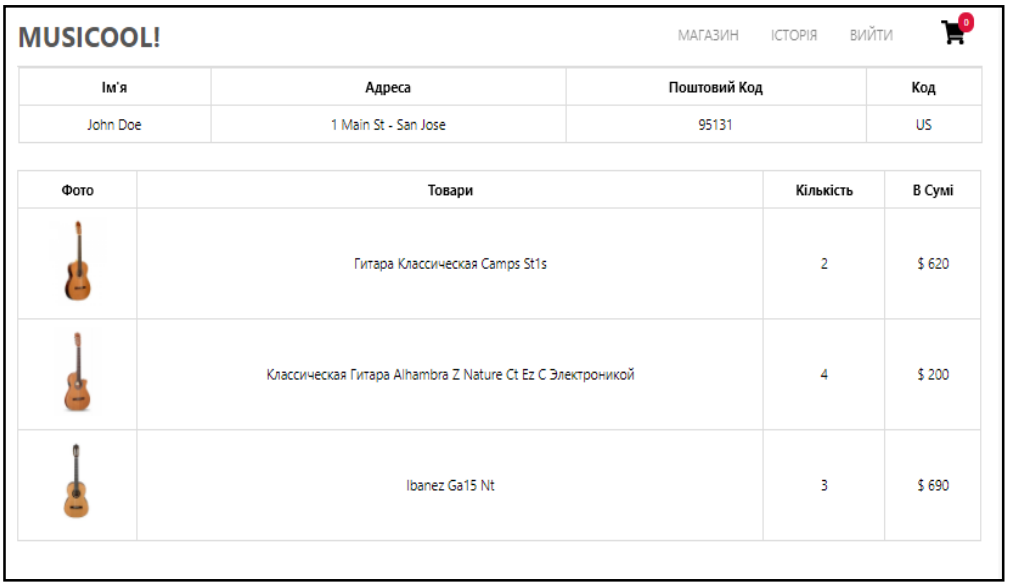

Рисунок 3.12 – Деталі окремої покупки

Основний функціонал надає ПейПал, з боку бек-енду, дані, що повертаються ПейПалом додаються в БД, та відмальовуються на фронт-енді.

```
createPayment: async(req, res) => {
      try {
         const user = await Users.findById(req.user.id).select('name email')
         if(!user) return res.status(400).json({msg: "Користувача не знайдено!"})
         const {cart, paymentID, address} = req.body;
         const {_id, name, email} = user;
         const newPayment = new Payments({
            user_id: _id, name, email, cart, paymentID, address
         })
         cart.filter(item => {
            return sold(item._id, item.quantity, item.sold)
         })
         await newPayment.save()
         res.json({msg: "Успішно придбано!"})
      } catch (err) {
         return res.status(500).json({msg: err.message})
}
```

З боку фронт-енду, також проводиться перевірка, чи є користувач адміністратором.

Адміністратор може переглядати всі записи, користувач тільки записи про свої покупки.

```
useEffect(() => {
      if(token){
         const getHistory = async() =>{
            if(isAdmin){
               const res = await axios.get('/api/payment', {
                  headers: {Authorization: token}
               })
               setHistory(res.data)
            }else{
               const res = await axios.get('/user/history', {
                  headers: {Authorization: token}
               })
               setHistory(res.data)
            }
         }
         getHistory()
      }
},[token, isAdmin, setHistory])
```

3.5 Реалізація панелі адміністратора

Головна сторінка адміністратора відрізняється (рис. 3.13). У адміністратора немає потреби у корзині, зате у нього є доступ до важливого функціоналу, такого як керування категоріями товарів, додавання, видалення, та оновлення товарів.

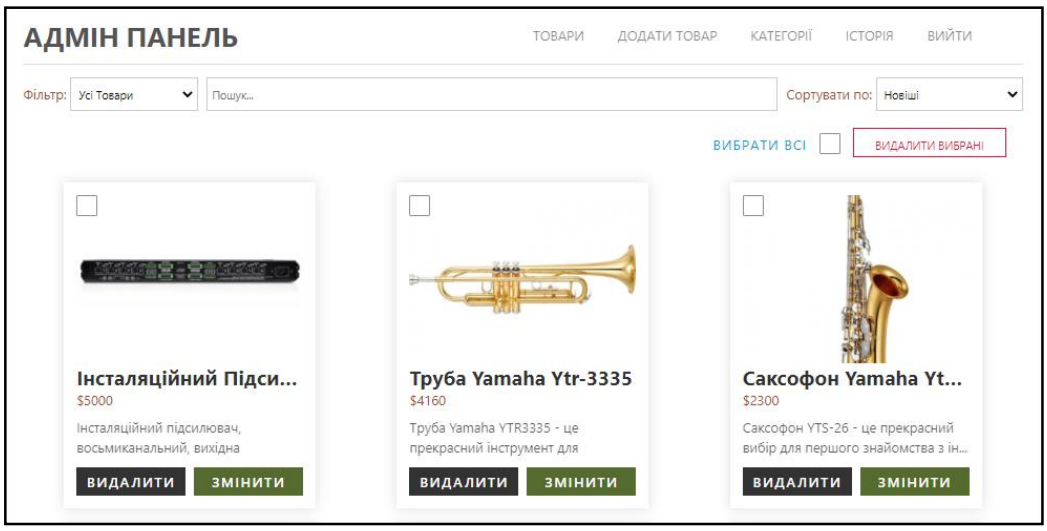

Рисунок 3.13 – Адмін панель

В залежності чи є користувач адміном чи ні, фронт-енд відображає різні сторінки.

Головна сторінка проводить перевірку на наявність прав адміна і додає функціонал видалення та оновлення товарів на головну сторінку.

Для доступу до меню додавання товарів, та керування категоріями, провірка на наявність прав адміна проводиться в шапці сайту.

Меню керування категоріями відносно просте. Адмін має доступ до списку усіх існуючих категорій, може видаляти будь-яку з них, та змінювати назву категорії. На рисунку 3.14 можна ознайомитись з цим меню.

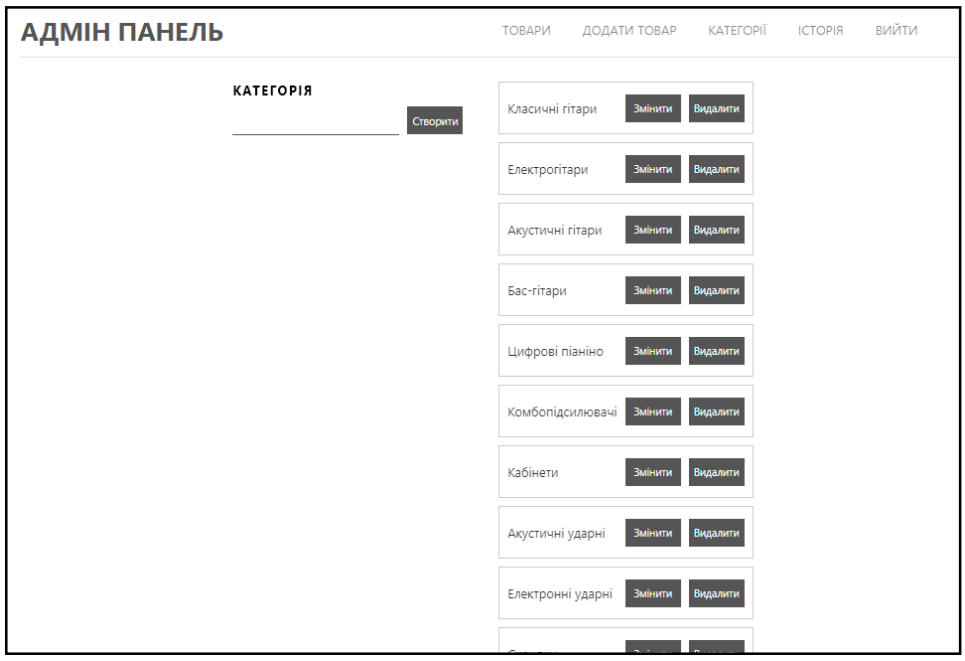

Рисунок 3.14 – Меню категорій

Додавання продукту має свій зрозумілий та зручний інтерфейс (рис. 3.15).

Область для завантаження фото, де після завантаження воно буде відображене, поле для назви, товару, його унікального ідентифікатора у БД, ціни у доларах США, короткий опис товару.

Поле для характеристик бере до уваги структуру заповнення, якщо у полі для опису весь текст буде відображений в лінійку, як звийний текст, то у полі для характеристик, відступи та розриви сторінки будуть відображатись теж. Спосіб відображення характеристик варто покращити у майбутніх версіях продукту.

```
const adminRouter = () =>{
      return(
         <>
            <li><Link to="/create_product">Додати товар</Link></li>
            <li><Link to="/category">Категорії</Link></li>
         </>
      )
   }
const loggedRouter = () =>{
      return(
         <>
            <li><Link to="/history">Історія</Link></li>
            <li><Link to="/" onClick={logoutUser}>Вийти</Link></li>
         </>
)
```

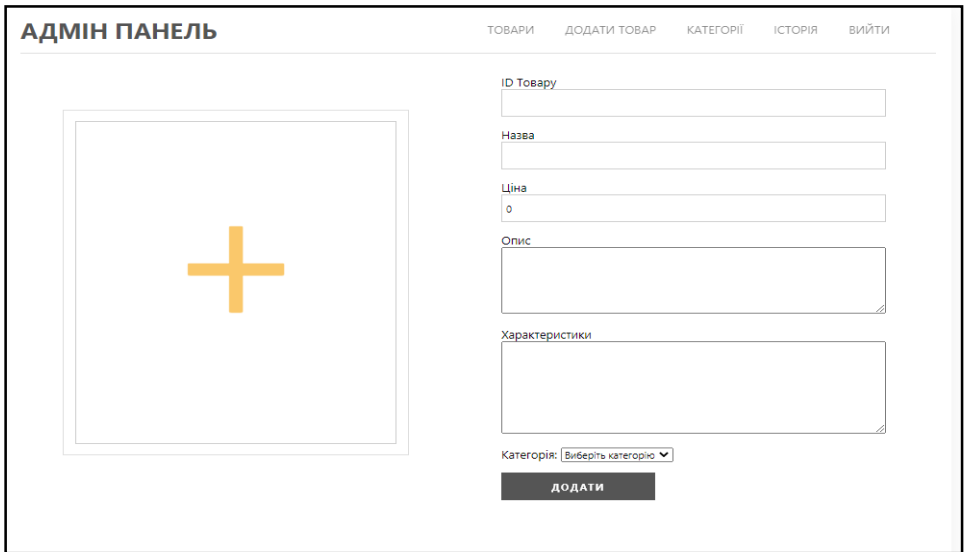

Рисунок 3.15 – Додавання товару

По заданій формі, буде створений новий запис в колекції товарів, з відповідними полями та картинками.

```
createProduct: async(req, res) =>{
      try {
      const {product_id, title, price, description, content, images, category} = req.body
         if(!images) return res.status(400).json({msg: "Не завантажено фото!"})
         const product = await Products.findOne({product_id})
         if(product)
            return res.status(400).json({msg: "Товар уже існує!"})
```

Окремо важливо пояснити, як працює завантаження фото. При добавленні файлу, він завантажується на сервіс Cloudinary. Цей сервіс, повертає нам посилання на фото, це посилання передається на сторінку, для відображення, і далі це посилання зберігається в БД.

```
router.post('/upload',auth , authAdmin, (req, res) =>{
    try {
        if(!req.files || Object.keys(req.files).length === 0)
            return res.status(400).json({msg: 'No files were uploaded.'})
        const file = req.files.file;
        if(file.size > 1024*1024 * 10) {
            removeTmp(file.tempFilePath)
            return res.status(400).json({msg: "Size too large"})
        }
        if(file.mimetype !== 'image/jpeg' && file.mimetype !== 'image/png'){
            removeTmp(file.tempFilePath)
            return res.status(400).json({msg: "File format is incorrect."})
}
```

Висновок до розділу 3

Було розібрано ключові елементи реалізації продукту, детально розглянуто взаємодію з БД, переглянуто основні сторінки проекту з аналізом принципів взаємодії різних модулів між собою. Продемонстровано дизайн та функціонал сайту на його поточному етапі.

РОЗДІЛ 4. ОХОРОНА ПРАЦІ

4.1 Умови роботи на робочому місці з комп’ютером

Навчальний процес з використанням комп’ютера, відбувається в середовищі, яке в певній мірі впливає на функціональний стан студентів, які перебувають у комп’ютерній аудиторії. Найважливішими несприятливими факторами середовища при роботі за комп'ютером є :

- фізичні параметри мікроклімату;
- освітлення;
- електромагнітні випромінювання різних частотних діапазонів;
- виробничий шум та вібрація; іонізація повітря; статична електрика;
- психофізіологічні – перенапруження зорового аналізатора;
- недостатня рухома діяльність;
- нервово-емоційне напруження;

Відповідно до НПАОП 0.00-1.28-10 є неприпустимим розташування приміщень, призначених для роботи з ВДТ у підвалах та цокольних поверхах. Також забороняється розташування вибухонебезпечних приміщень категорії А і Б (ОНТП 24-86) та виробництв з мокрими технологічними процесами поряд з приміщенням, де розташовуються ЕОМ (ПЕОМ), а також над такими приміщеннями, або під ними. Окрім того, виробничі приміщення для роботи з ВДТ не повинні межувати з приміщеннями, у яких рівень шуму та вібрації перевищує допустимі значення.

У процесі роботи з комп'ютером необхідно дотримувати правильний режим праці і відпочинку. У противному випадку в працюючого за ПК відзначаються значна напруга зорового апарата з появою скарг на незадоволеність роботою, головні болі, дратівливість, порушення сну, утома і хворобливі відчуття в очах, у попереку, в області шиї і руках.

Розташування робочого місця особи, що працює з ПК, в даному випадку

студента, повинне відповідати вимогам ДСТУ 22.269-79 «Рабоче місце оператора. Взаємне розташування элементів робочого місця».

- Облаштування робочого місці повинно забезпечувати:
- правильне розміщення робочого місця у виробничому приміщенні;
- належні умови освітлення приміщення і робочого місця, відсутність відблисків;
- належні ергономічні характеристики основних елементів робочого місця;
- характер та особливості трудової діяльності;

Для зменшення втоми, місця користувачів ЕОМ мають бути зручними.

Конструкція робочого місця користувача ПК, (при роботі сидячи) має забезпечувати підтримання оптимальної робочої пози з такими ергономічними характеристиками:

- ступні ніг – на підлозі або на підставці для ніг;
- стегна – в горизонтальній площині;
- передпліччя – вертикально;
- передпліччя – вертикально;
- лікті – під кутом 70-90 до вертикальної площини;
- зап'ястя зігнуті під кутом не більше 20 відносно горизонтальної площини;

Робоче місце користувача ПК, обладнується робочим столом, стільцем і підставкою для ніг. Висота робочого стола має бути в межах від 0,65 до 0,8 м, а ширина повинна забезпечувати можливість виконання операцій в зоні досяжності моторного поля.

Клавіатуру слід розташовувати на поверхні столу на відстані 200 мм від краю, звернутого до працюючого.

У конструкції клавіатури має передбачатися опорний пристрій (виготовлений із матеріалу з високим коефіцієнтом тертя, що перешкоджає його переміщенню), який дає змогу змінювати кут нахилу поверхні клавіатури у межах 5...15.

4.2 Мікроклімат та рівень іонізації повітря виробничого приміщення

Як фактор виробничого середовища, мікроклімат впливає на теплообмін організму людини з цим середовищем.

Необхідною умовою життєдіяльності людини є терморегуляція, тобто здатність організму регулювати віддачу тепла в оточуюче середовище. Цей процес визначається параметрами мікроклімату. Метеорологічні умови визначаються такими параметрами:

– температурою повітря в приміщенні, С;

– відносною вологістю повітря, %;

– рухливістю повітря, м/с;

– тепловим випромінюванням, Вт/м^3;

Принцип нормування мікроклімату – створення оптимальних умов для теплообміну тіла людини з навколишнім середовищем.

Параметри мікроклімату, вміст шкідливих речовин на робочих місцях, оснащених моніторами, відповідають вимогам ДСН 3.3.6.042-99 «Державні санітарні норми мікроклімату виробничих приміщень», ДСТУ 12.1.005-88 «Загальні санітарно-гігієнічні вимоги до повітря робочої зони».

Обчислювальна техніка є джерелом істотних тепловиділень, що може привести до підвищення температури і зниження відносної вологості в приміщенні.

В приміщеннях, де встановлені комп'ютери, повинні дотримуватися оптимальні параметри мікроклімату, які визначають комфортні умови. Ці параметри залежать від періоду року, категорії робіт за важкістю, і від теплових характеристик виробничого приміщення (табл. 4.1).

Робота за комп’ютером характеризується малими фізичним навантаженнями, цей вид діяльності належить до категорії легких робіт – Ia за критерієм загальних енерговитрат організму (ДСН 3.3.6.042-99)

Таблиця 4.1

Параметри мікроклімату для приміщень, де встановлені комп'ютери (ДСТУ 12.1.005-88; ДСН 3.3.6.042-99)

Період року	Категорія робіт	Параметр мікроклімату	Величина
Холодний	Роботи легкі Ia	Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	22÷24°С 40÷60% до 0,1м/с
Теплий		Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	23÷25°С 40÷60% 0,1.0,2м/с

Під час роботи комп’ютерної техніки в повітряному середовищі відбувається суттєва трансформація іонного складу, істотно знижується концентрація легких, середніх та важких негативно зарядних частинок. Така зміна балансу іонного складу призводить до негативного впливу на здоров’я працюючих.

Рівні іонізації повітря приміщень при роботі на персональних комп’ютерах визначені в таблиці 4.2 (відповідно до НПАОП 0.03-3.06-80).

Таблиця 4.2

Рівні іонізації повітря в приміщенні з ПК

Рівні	Кількість іонів в 1 см³ повітря	
	n+	n-
Мінімально необхідні	400	600
Максимально допустимі	50000	50000
Оптимальні	1500-3000	3000-5000

Для підтримки оптимальних значень мікроклімату та підтримання нормальної концентрації позитивних та негативних іонів в приміщенні аудиторії пропонується удосконалити системи опалення, природної вентиляції та встановити кондиціонер.

Дисплеї на основі ЕПТ є потенційним джерелом випромінювання кількох діапазонів електромагнітного спектра: рентгенівського, оптичного, радіочастотного. Кожний вид випромінювання відрізняється своїми особливими характеристиками впливу на організм людини.

Рентгенівське випромінювання. Дослідження показують, що відеотермінал не несе небезпеки для користувача ПЕОМ, оскільки інтенсивність такого випромінювання нижча за гранично допустимі норми (ГДН). Відповідно до «Норм радіаційної безпеки України» (НРБУ-97) гранично допустима потужність експозиційної дози рентгенівського випромінювання на відстані 5 см від поверхні екрана відеотермінала становить $7{,}74 \cdot 10^{-12}$ Кл/кг, що відповідає еквівалентній дозі 0,1 мбер/год. (100 мкР/год.).

Оптичне випромінювання. Оптичні види випромінювання виникають завдяки взаємодії електронів з шаром люмінофору, нанесеного на екран ВДТ. Область оптичного випромінювання включає ультрафіолетове (УФ), світлове та інфрачервоне (ІЧ) випромінювання.

УФ-випромінювання впливає на шкіру та очі людини. Такий вплив на шкірі проявляється досить швидко, а для очей характерним є період прихованої дії. Рівень УФ-випромінювання, який був виявлений, досить низький і становить 1 середньому 0,001Вт/м².

Світлове випромінювання впливає в основному на око і призводить до втоми очей, запалення райдужної оболонки. Однак ці симптоми швидко минають і не викликають патологічних змін.

ІЧ-випромінювання – довжина хвиль обмежена від 0,76мм до 1мм. Для більшості біологічних матеріалів випромінювання цього діапазону вважаються непрозорими. Інтенсивність інфрачервоних випромінювань нижча за значення, передбачені ДсанПіН 3.3.2.007-98.

Електромагнітні випромінювання (ЕМВ) радіочастотного діапазону. Джерелом ЕМВ є відеотермінал. Тому, обираючи робоче місце для комп'ютера, необхідно пам'ятати, що його задня і бокові стінки можуть бути джерелом значно більшого ЕМВ, ніж екран.

З метою профілактики несприятливого впливу електромагнітного випромінювання від ВДТ на користувача необхідно:

- встановити на робочому місці відеотермінал, що відповідає сучасним вимогам стосовно захисту від. випромінювань;
- не концентрувати на робочому місці великої кількості радіоелектронних пристроїв;

Гранично допустимі рівні випромінювань на робочих місцях з ПК наведено в таблиці 4.3.

Таблиця 4.3

Гранично допустимі рівні випромінювань

Вид випромінювання	Діапазон хвиль (частот)	Гранично допустимий рівень
Іонізуюче випромінювання		
М’яке рентгенівське випромінювання (на відстані 0,05 м від екрана та корпуса ВДТ)	0,01 – 1 нм	100мкР/год
Оптичні випромінювання		

Продовження таблиці 4.3

Ультрафіолетове випромінювання	315 – 400 нм	10 Вт/м^2
Видиме випромінювання (яскравість)	400 – 700 нм	1000 кд/м^2
Інфрачервоне випромінювання	700 нм – 1мм	100 Вт/м^2
Електромагнітні випромінювання (поля радіочастотного та низькочастотного діапазонів)		
Напруженість електромагнітного поля на відстані 0,5 м навколо монітору за електричною складовою	2 кГц – 400 кГц 5 Гц – 2 кГц	2,5 В/м 25 В/м
Щільність магнітного потоку	2 кГц – 400 кГц 5 Гц – 2 кГц	25 нТл 250 нТл
Електростатичні поля		
Поверхневий електростатичний потенціал	–	500 В

Відомо, що шум несприятливо діє на слуховий аналізатор та інші органи та системи організму людини. Визначальне значення щодо такої дії має інтенсивність шуму, його частотний склад, тривалість щоденного впливу, індивідуальні особливості людини, а також специфіка виробничої діяльності. Ті види діяльності, у яких поєднується напружена розумова робота та інтенсивне використання комп’ютера (редагування тексту, верстка оригіналу, "запуск" та відлагодження програм тощо) характеризується відчутним впливом навіть незначних рівнів шуму. Цей вплив виражається у зниженні розумової працездатності, швидкій втомлюваності, послабленні уваги, появі головного болю та ін.

Рівні звукового тиску в октавних смугах частот, рівні звуку та еквівалентні рівні звуку на робочих місцях, обладнаних ВДТ і ПК визначені ДСанПіН 3.3.2-007-98 не повинні перевищувати 65 дБА. Для приміщення

аудиторії, рівень шуму не перевищує 60 дБА.

Для забезпечення нормованих рівнів шуму у виробничих приміщеннях та на робочих місцях застосовуються шумопоглинальні засоби, вибір яких обґрунтовується спеціальними інженерно-акустичними розрахунками.

Рівні вібрації під час виконання робіт з ЕОМ у виробничих приміщеннях не повинні перевищувати допустимих значень – 70 дБ, визначених ДСТУ 12.1.012-90 та СН 3044-84 "Санітарні норми робочих місць"

Для зниження вібрації обладнання, пристрої, пристосування необхідно встановлювати на спеціальні амортизуючі прокладки, передбачені документами.

Близько 90% всієї інформації, що отримується людиною, приходиться на органи зору. Організація освітленості робочих місць грає велику роль у житті людини. Недостатнє та нераціональне освітлення веде до втомлення очей, розладу центральної нервової системи, зниженню розумової та фізичної працездатності, а у ряді випадків може бути причиною травматизму (близько 5% травм приходиться на частку нераціонального та недостатнього освітлення).

Щоб уникнути перевтоми, а також для профілактики професійних захворювань та виробничого травматизму потрібно дотримуватись наступних вимог:

- створювати на робочій поверхні освітленість, що відповідає характеру зорової роботи і не є нижчою за встановлені норми;
- забезпечити достатню рівномірність та постійність рівня освітленості у виробничих приміщеннях, щоб уникнути частої переадаптації органів зору;
- не створювати засліплювальної дії як від самих джерел освітлення, так і від інших предметів, що знаходяться в полі зору;
- не створювати на робочій поверхні різких та глибоких тіней (особливо рухомих);
- повинен бути достатній для розрізнення деталей контраст поверхонь, що освітлюються;

– не створювати небезпечних та шкідливих виробничих чинників (шум, теплові випромінювання, небезпека ураження струмом, пожежо та вибухонебезпека світильників);
– повинно бути надійним і простим в експлуатації, економічним та естетичним;

Штучне освітлення в приміщенні з комп'ютеризованим робочим місцем здійснюється системою загального освітлення.

Як джерела світла в разі штучного освітлення застосовуються світильники серії ЛПО353. Світильники укомплектовані високочастотними пускорегулювальними апаратами (ВЧ ПРА). Яскравість світильників загального освітлення в зоні кутів випромінювання від 50 до 90 град. з вертикаллю в повздовжній та поперечній площинах становить не більше 200 кд/м², захисний кут світильників – не менше ніж 40 град.

Рівень освітленості у аудиторії, Е=300лк.

ЕОМ, периферійні пристрої ЕОМ та устаткування для обслуговування, ремонту та налагодження ЕОМ, електропроводи та кабелі за виконанням та ступенем захисту мають відповідати класу зони за ПВЕ, мати апаратуру захисту від струму короткого замикання та інших аварійних режимів.

Використання нульового робочого провідника як нульового захисного провідника забороняється.

Нульовий захисний провід прокладається від склейки групового розподільчого щита до розеток живлення.

Не допускається підключення на щиті до одного контактного затискача нульового робочого та нульового захисного провідників.

Площа перерізу нульового робочого та нульового захисного провідника в груповій трипровідній мережі повинна бути не менше площі перерізу фазового провідника. Усі провідники повинні відповідати номінальним параметрам мережі та навантаження, умовам навколишнього середовища, умовам розподілу провідників, температурному режиму та типам апаратури захисту, вимогам ПВЕ.

З метою підвищення рівня електробезпеки в приміщенні аудиторії неприпустимим є:

- експлуатація кабелів та проводів з пошкодженою або такою, що втратила захисні властивості за час екплуатації, ізоляцією;
- залишення під напругою неізольованих кабелів та проводів;
- застосування саморобних подовжувачів, які не відповідають вимогам ПВЕ до переносних електропроводок;
- користування пошкодженими розетками, розгалужувальними та з'єднувальними коробками, вимикачами та іншими електровиробами;
- застосування для опалення приміщення нестандартного (саморобного) електронагрівального обладнання;

ВДТ є джерелом електростатичних зарядів.

Тривале перебування в електростатичному полі, створеному цими зарядами, негативно впливає на здоров'я працюючих: бронхо-легеневі захворювання, порушення серцево-судинної та нервової систем, ураження шкіри тощо.

Для запобігання створенню значної напруженості поля та захисту від статичної електрики необхідно:

- встановити нейтралізатори статичної електрики;
- підтримувати в приміщенні з ВДТ відносну вологість повітря не нижче 45-50% (чим сухіше повітря тим більше електростатичних зарядів); можна для цього використати навіть побутові зволожувачі;
- застелити підлогу в приміщеннях з ВДТ антистатичним лінолеумом і проводити щоденне вологе прибирання;
- складати всі полімерні покриття (чохли) ВДТ у найбільш віддаленому від користувачів місці розміщення;
- протирати екран та робоче місце спеціальною антистатичною серветкою або зволоженою тканиною;

4.3 Вимоги до безпечного користування комп’ютерною технікою

Найбільш імовірною причиною виникнення пожеж в приміщенні є порушення вимог при експлуатації комп'ютерної техніки та займання електропроводки внаслідок коротких замикань.

Для того щоб уникнути виникнення пожежі, потрібно дотримуватися наступних заходів:

– дотримання правил пожежної безпеки при роботі з комп’ютером; електро-обладнанням та освітлювальними приладами;
– періодичний контроль цілісності і надійності електроізоляції;
– наявність інструкцій з пожежної безпеки;
– навчання, атестація і переатестація персоналу з пожежної безпеки;
– наявність системи захисту від атмосферної електрики;
– періодичне зняття зарядів статичної електрики;
– заборона куріння в приміщенні.
– застосування будівельних конструкцій із ступенем вогнестійкості не нижче II, а також використання важкогорючих або негорючих матеріалів в інтер’єрі виробничого приміщення;
– наявність схеми евакуації;
– наявність пристроїв автоматичного вимкнення ПЕОМ та іншого електроустаткування на випадок пожежі;
– наявність первинних засобів пожежогасіння (вогнегасник ВВК-5 – 2шт);

Висновок до розділу 4

Було розібрано основні правила безпеки роботи за персональним комп'ютером, та необхідні умови в приміщенні в якому проводиться робота. Особливу увагу було надано до правил експлуатації ПК.

ВИСНОВКИ

В результаті виконання даної кваліфікаційної роботи було розроблено інтернет-ресурс з продажу музичних інструментів з повним функціоналом.

У результаті обліку всіх зроблених зауважень можливе покращення створеного програмного продукту, на який буде потрібно мінімум змін вихідного коду програми, мінімум змін в дизайні.

Було:

- розглянуто ключову архітектуру клієнт-сервер;
- наведено визначення інтернет-магазину, його переваги та недоліки;
- розглянуто два ресурси-аналоги, їх функціонал, плюси на мінуси;
- порівняно їх з розробленим продуктом;
- розібрано структуру бази данних;
- спроектовано модулі і їх залежності з коротким описом;
- розібрано ключові елементи реалізації продукту;
- розглянуто взаємодію з БД;
- переглянуто основні сторінки проекту, з аналізом принципів взаємодії різних модулів між собою;
- продемонстровано оформлення та функціонал сайту на його поточному етапі;
- розібрано основні правила безпеки під час роботи за персональним комп'ютером, та необхідні умови в приміщенні в якому проводиться робота;

Особливу увагу було надано до правил експлуатації ПК.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Музика. *Wikipedia:* вебсайт. URL: https://uk.wikipedia.org/Музика
2. Музичний інструмент. *Wikipedia:* вебсайт. URL: https://uk.wikipedia.org/wiki/Музичний_інструмент
3. Музичні стилі, жанри та напрями. *Wikipedia:* вебсайт. URL: https://uk.wikipedia.org/wiki/Музичні_стилі,_жанри_та_напрями
4. Музика та її жанри. *Wikipedia:* вебсайт. URL: https://madlife.webnode.com.ua/l/muzika-ta-jiji-zhanri/
5. React. *Wikipedia:* вебсайт. URL: https://ru.wikipedia.org/wiki/React
6. Node.js. *Wikipedia:* вебсайт. URL: https://ru.wikipedia.org/wiki/Node.js
7. Express.js. *Wikipedia:* вебсайт. URL: https://ru.wikipedia.org/wiki/Express.js
8. MongoDB. *Wikipedia:* вебсайт. URL: https://ru.wikipedia.org/wiki/MongoDB
9. JWT. *jwt.io:* вебсайт. URL: https://jwt.io/introduction
10. PayPal. *Wikipedia:* вебсайт. URL: https://uk.wikipedia.org/wiki/PayPal
11. JavaScript. *Wikipedia:* вебсайт. URL: https://uk.wikipedia.org/wiki/JavaScript
12. TOS.IN. *tos.in.ua:* вебсайт. URL: https://tos.in.ua/
13. JAZZ-CLUB-SERVICE. *jcs.ua* вебсайт. URL: https://jcs.ua/
14. MVC. *Wikipedia:* вебсайт. URL: https://ru.wikipedia.org/wiki/Model-View-Controller
15. Клієнт-серверна архітектура. *Wikipedia:* вебсайт. URL: https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура
16. Організація і технологія роботи інтернет-магазину. *pidru4niki.com:* вебсайт. URL: https://pidru4niki.com/10931123/informatika/organizatsiya_tehnologiya_roboti_internet-magazinu

ДОДАТОК

Код продукту

CategoryCtrl.js

```
const Category = require('../models/categoryModel')
const Products = require('../models/productModel')

const categoryCtrl = {
   getCategories: async(req, res) =>{
      try {
         const categories = await Category.find()
         res.json(categories)
      } catch (err) {
         return res.status(500).json({msg: err.message})
      }
   },
   createCategory: async (req, res) =>{
      try {
         const {name} = req.body;
         const category = await Category.findOne({name})
         if(category) return res.status(400).json({msg: "Категорія вже існує!"})

         const newCategory = new Category({name})

         await newCategory.save()
         res.json({msg: "Додано категорію!"})
      } catch (err) {
         return res.status(500).json({msg: err.message})
      }
   },
   deleteCategory: async(req, res) =>{
      try {
         const products = await Products.findOne({category: req.params.id})
         if(products) return res.status(400).json({
            msg: "Видаліть товари з цієї категорії!"
         })

         await Category.findByIdAndDelete(req.params.id)
         res.json({msg: "Видалено!"})
      } catch (err) {
         return res.status(500).json({msg: err.message})
      }
   },
   updateCategory: async(req, res) =>{
      try {
         const {name} = req.body;
         await Category.findOneAndUpdate({_id: req.params.id}, {name})

         res.json({msg: "Оновлено!"})
      } catch (err) {
         return res.status(500).json({msg: err.message})
      }
   }
}
```

```
module.exports = categoryCtrl
```

PaymentCtrl.js

```
const Payments = require('../models/paymentModel')
const Users = require('../models/userModel')
const Products = require('../models/productModel')

const paymentCtrl = {
    getPayments: async(req, res) =>{
        try {
            const payments = await Payments.find()
            res.json(payments)
        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    createPayment: async(req, res) => {
        try {
            const user = await Users.findById(req.user.id).select('name email')
            if(!user) return res.status(400).json({msg: "Користувача не знайдено!"})

            const {cart, paymentID, address} = req.body;

            const {_id, name, email} = user;

            const newPayment = new Payments({
                user_id: _id, name, email, cart, paymentID, address
            })

            cart.filter(item => {
                return sold(item._id, item.quantity, item.sold)
            })


            await newPayment.save()
            res.json({msg: "Успішно придбано!"})

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    }
}

const sold = async (id, quantity, oldSold) =>{
    await Products.findOneAndUpdate({_id: id}, {
        sold: quantity + oldSold
    })
}

module.exports = paymentCtrl
```

ProductCtrl.js

```
const Products = require('../models/productModel')
```

```
class APIfeatures {
   constructor(query, queryString){
      this.query = query;
      this.queryString = queryString;
   }
   filtering(){
     const queryObj = {...this.queryString}

     const excludedFields = ['page', 'sort', 'limit']
     excludedFields.forEach(el => delete(queryObj[el]))

     let queryStr = JSON.stringify(queryObj)
     queryStr = queryStr.replace(/\b(gte|gt|lt|lte|regex)\b/g, match => '$' + match)

   //    gte = greater than or equal
   //    lte = lesser than or equal
   //    lt = lesser than
   //    gt = greater than

     this.query.find(JSON.parse(queryStr))

     return this;
   }

   sorting(){
      if(this.queryString.sort){
         const sortBy = this.queryString.sort.split(',').join(' ')
         this.query = this.query.sort(sortBy)
      }else{
         this.query = this.query.sort('-createdAt')
      }

      return this;
   }

   paginating(){
      const page = this.queryString.page * 1 || 1
      const limit = this.queryString.limit * 1 || 9
      const skip = (page - 1) * limit;
      this.query = this.query.skip(skip).limit(limit)
      return this;
   }
}

const productCtrl = {
   getProducts: async(req, res) =>{
      try {
         const features = new APIfeatures(Products.find(), req.query)
         .filtering().sorting().paginating()

         const products = await features.query

         res.json({
            status: 'success',
            result: products.length,
            products: products
```

```
            })

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    createProduct: async(req, res) =>{
        try {
            const {product_id, title, price, description, content, images, category} = req.body;
            if(!images) return res.status(400).json({msg: "Не завантажено фото!"})

            const product = await Products.findOne({product_id})
            if(product)
                return res.status(400).json({msg: "Товар уже існує!"})
            if(!category) return res.status(400).json({msg: "Виберіть категорію"})

            const newProduct = new Products({
                product_id, title: title.toLowerCase(), price, description, content, images, category
            })

            await newProduct.save()
            res.json({msg: "Товар додано!"})

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    deleteProduct: async(req, res) =>{
        try {
            await Products.findByIdAndDelete(req.params.id)
            res.json({msg: "Видалено!"})
        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    updateProduct: async(req, res) =>{
        try {
            const {title, price, description, content, images, category} = req.body;
            if(!images) return res.status(400).json({msg: "Не завантажено фото!"})
            if(!category) return res.status(400).json({msg: "Виберіть категорію"})

            await Products.findOneAndUpdate({_id: req.params.id}, {
                title: title.toLowerCase(), price, description, content, images, category
            })

            res.json({msg: "Товар оновлено!"})
        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    }
}

module.exports = productCtrl
```

UserCtrl.js

```
const Users = require('../models/userModel')
```

```
const Payments = require('../models/paymentModel')
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')

const userCtrl = {
    register: async (req, res) =>{
        try {
            const {name, email, password} = req.body;

            const user = await Users.findOne({email})
            if(user) return res.status(400).json({msg: "Пошта уже зареєстрована!"})

            if(password.length < 6)
                return res.status(400).json({msg: "Пароль повинен містити хоча б 6 символів!"})

            const passwordHash = await bcrypt.hash(password, 10)
            const newUser = new Users({
                name, email, password: passwordHash
            })

            await newUser.save()

            const accesstoken = createAccessToken({id: newUser._id})
            const refreshtoken = createRefreshToken({id: newUser._id})

            res.cookie('refreshtoken', refreshtoken, {
                httpOnly: true,
                path: '/user/refresh_token',
                maxAge: 7*24*60*60*1000 // 7d
            })

            res.json({accesstoken})

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    login: async (req, res) =>{
        try {
            const {email, password} = req.body;

            const user = await Users.findOne({email})
            if(!user) return res.status(400).json({msg: "Користувача не знайдено!"})

            const isMatch = await bcrypt.compare(password, user.password)
            if(!isMatch) return res.status(400).json({msg: "Хибний пароль!"})

            const accesstoken = createAccessToken({id: user._id})
            const refreshtoken = createRefreshToken({id: user._id})

            res.cookie('refreshtoken', refreshtoken, {
                httpOnly: true,
                path: '/user/refresh_token',
                maxAge: 7*24*60*60*1000 // 7d
            })

            res.json({accesstoken})
```

```
    } catch (err) {
        return res.status(500).json({msg: err.message})
    }
},
logout: async (req, res) =>{
    try {
        res.clearCookie('refreshtoken', {path: '/user/refresh_token'})
        return res.json({msg: "Вихід успішний!"})
    } catch (err) {
        return res.status(500).json({msg: err.message})
    }
},
refreshToken: (req, res) =>{
    try {
        const rf_token = req.cookies.refreshtoken;
        if(!rf_token) return res.status(400).json({msg: "Авторизуйтесь!"})

        jwt.verify(rf_token, process.env.REFRESH_TOKEN_SECRET, (err, user) =>{
            if(err) return res.status(400).json({msg: "Авторизуйтесь!"})

            const accesstoken = createAccessToken({id: user.id})

            res.json({accesstoken})
        })

    } catch (err) {
        return res.status(500).json({msg: err.message})
    }

},
getUser: async (req, res) =>{
    try {
        const user = await Users.findById(req.user.id).select('-password')
        if(!user) return res.status(400).json({msg: "Користувача не знайдено!"})

        res.json(user)
    } catch (err) {
        return res.status(500).json({msg: err.message})
    }
},
addCart: async (req, res) =>{
    try {
        const user = await Users.findById(req.user.id)
        if(!user) return res.status(400).json({msg: "Користувача не знайдено!"})

        await Users.findOneAndUpdate({_id: req.user.id}, {
            cart: req.body.cart
        })

        return res.json({msg: "Додано в кошик!"})
    } catch (err) {
        return res.status(500).json({msg: err.message})
    }
},
history: async(req, res) =>{
    try {
```

```
            const history = await Payments.find({user_id: req.user.id})

            res.json(history)
        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    }
}

const createAccessToken = (user) =>{
    return jwt.sign(user, process.env.ACCESS_TOKEN_SECRET, {expiresIn: '11m'})
}
const createRefreshToken = (user) =>{
    return jwt.sign(user, process.env.REFRESH_TOKEN_SECRET, {expiresIn: '7d'})
}

module.exports = userCtrl
```

GlobalState.js

```
import React, {createContext, useState, useEffect} from 'react'
import ProductsAPI from './api/ProductsAPI'
import UserAPI from './api/UserAPI'
import CategoriesAPI from './api/CategoriesAPI'

import axios from 'axios'

export const GlobalState = createContext()

export const DataProvider = ({children}) =>{
    const [token, setToken] = useState(false)

    useEffect(() =>{
        const firstLogin = localStorage.getItem('firstLogin')
        if(firstLogin){
            const refreshToken = async () =>{
                const res = await axios.get('/user/refresh_token')

                setToken(res.data.accesstoken)

                setTimeout(() => {
                    refreshToken()
                }, 10 * 60 * 1000)
            }
            refreshToken()
        }
    },[])


    const state = {
        token: [token, setToken],
        productsAPI: ProductsAPI(),
        userAPI: UserAPI(token),
        categoriesAPI: CategoriesAPI()
    }

    return (
```

```
        <GlobalState.Provider value={state}>
            {children}
        </GlobalState.Provider>
    )
}
```

CategoriesAPI.js

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

function CategoriesAPI() {
    const [categories, setCategories] = useState([])
    const [callback, setCallback] = useState(false)

    useEffect(() =>{
        const getCategories = async () =>{
            const res = await axios.get('/api/category')
            setCategories(res.data)
        }

        getCategories()
    },[callback])
    return {
        categories: [categories, setCategories],
        callback: [callback, setCallback]
    }
}

export default CategoriesAPI
```

ProductAPI.js

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

function ProductsAPI() {
    const [products, setProducts] = useState([])
    const [callback, setCallback] = useState(false)
    const [category, setCategory] = useState('')
    const [sort, setSort] = useState('')
    const [search, setSearch] = useState('')
    const [page, setPage] = useState(1)
    const [result, setResult] = useState(0)

    useEffect(() =>{
        const getProducts = async () => {
            const res = await axios.get(`/api/products?limit=${page*9}&${category}&${sort}&title[regex]=${se
arch}`)
            setProducts(res.data.products)
            setResult(res.data.result)
        }
        getProducts()
    },[callback, category, sort, search, page])

    return {
        products: [products, setProducts],
        callback: [callback, setCallback],
```

```
        category: [category, setCategory],
        sort: [sort, setSort],
        search: [search, setSearch],
        page: [page, setPage],
        result: [result, setResult]
    }
}

export default ProductsAPI
```

UserAPI.js

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

function UserAPI(token) {
    const [isLogged, setIsLogged] = useState(false)
    const [isAdmin, setIsAdmin] = useState(false)
    const [cart, setCart] = useState([])
    const [history, setHistory] = useState([])

    useEffect(() =>{
        if(token){
            const getUser = async () =>{
                try {
                    const res = await axios.get('/user/infor', {
                        headers: {Authorization: token}
                    })

                    setIsLogged(true)
                    res.data.role === 1 ? setIsAdmin(true) : setIsAdmin(false)

                    setCart(res.data.cart)

                } catch (err) {
                    alert(err.response.data.msg)
                }
            }

            getUser()

        }
    },[token])



    const addCart = async (product) => {
        if(!isLogged) return alert("Авторизуйтесь для шопінгу!")

        const check = cart.every(item =>{
            return item._id !== product._id
        })

        if(check){
            setCart([...cart, {...product, quantity: 1}])

            await axios.patch('/user/addcart', {cart: [...cart, {...product, quantity: 1}]}, {
```

```
                headers: {Authorization: token}
            })

        }else{
            alert("Товар уже в корзині!")
        }
    }

    return {
        isLogged: [isLogged, setIsLogged],
        isAdmin: [isAdmin, setIsAdmin],
        cart: [cart, setCart],
        addCart: addCart,
        history: [history, setHistory]
    }
}

export default UserAPI
```