

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему РОЗРОБКА ІНТЕРНЕТ МАГАЗИНУ КОМП'ЮТЕРНИХ
КОМПЛЕКТУЮЧИХ
DEVELOPMENT OF A COMPUTER PARTS ONLINE STORE

Виконав: студент(ка) 4 курсу, групи КНТ-219сп
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

П'ятерня П.О.

(прізвище та ініціали)

Керівник Гладкова О.М.

(прізвище та ініціали)

Рецензент Зеленьова І.Я.

(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 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

П'ятерні Пилипа Олександровича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка інтернет магазину комп'ютерних
комплектуючих
Development of a Computer Parts Online Store

керівник проєкту (роботи) Гладкова Ольга Михайлівна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) червень 2022 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми.
3. Розробка програми. 4. Експлуатація та тестування програми.

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
1-4 Основна частина	Гладкова О.М., доцент		
Нормоконтролер	Дейнега Л.Ю., ст.викладач		

7. Дата видачі завдання « 04 » квітня 2022 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	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

(підпис)

П'ятерня П.О.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Гладкова О.М.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 84 с., 17 рис., 1 табл., 3 дод., 13 джерел.

ВЕБСАЙТ, INTERNET, BACKEND DESKTOP, ВЕБЗАСТОСУНОК, JAVASCRIPT, EXPRESS , NODE, EXPRESS, LAMP.

Об'єкт роботи – процес перегляду каталогу товарів з можливістю їх замовлення. Предмет роботи – програмне забезпечення перегляду каталогу товарів з можливістю їх замовлення онлайн. Мета роботи – розробити програмне забезпечення перегляду каталогу товарів з можливістю їх замовлення.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мови JavaScript, HTML, CSS, вебфреймворк React, Express, Tailwind система керування базами даних Postgres.

Результати. Розробку програмного забезпечення магазину комп'ютерних комплектуючих виконано за допомогою вебфреймворку Express, React, Tailwind з визначенням структури даних, вимог до програми, розробленням структури і алгоритму роботи програми на основі цих вимог, реалізацією відповідних одиниць. Програма дозволяє переглядати каталог товарів, сортувати товари за категоріями та ціною або використати пошук за назвою товару, користувач може звернутись за консультацією до менеджера через вбудований чат, для оформлення замовлення користувач має зареєструватись додати бажаний товар у корзину та заповнити форму доставки,

Висновки. Розроблене програмне забезпечення дозволяє підвищити зручність підбору та купівлі комп'ютерних комплектуючих онлайн.

Галузь використання. Програмне забезпечення може використовуватися в якості корисного веб порталу для підбору та купівлі комп'ютерних комплектуючих.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 84 pages, 17 figures, 1 table, 3 appendixes, 13 sources.

WEBSITE, INTERNET, BACKEND, DESKTOP, WEB APPLICATION, JAVASCRIPT, NODE, EXPRESS, LAMP.

The object of work is the process of viewing the catalog of goods with the possibility of ordering them. Subject of work - software for viewing the catalog of goods with the ability to order them online. The purpose of the work is to develop software for viewing the catalog of goods with the possibility of ordering them.

Materials, methods and technical tools: object-oriented programming, JavaScript, HTML, CSS, React web framework, Express, Tailwind Postgres database management system.

Results. Software development of the computer components store was performed using the web framework Express, React, Tailwind with the definition of data structure, program requirements, development of the structure and algorithm of the program based on these requirements, the implementation of appropriate units. The program allows you to view the catalog of goods, sort products by category and price or use the search by product name, the user can consult the manager through the built-in chat, to place an order the user must register to add the desired product to cart and fill out the delivery form.

Conclusions. Developed software can increase the convenience of selecting and purchasing computer components online.

Field of use. The software can be used as a useful web portal for selecting and purchasing computer components.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	9
Вступ.....	12
1 Аналіз предметної області.....	12
1.1 Світовий ринок інтернет магазинів.....	12
1.2 Огляд програмних аналогів.....	12
1.3 Висновки за розділом 1	15
2 Розробка архітектури програми.....	16
2.1 Вибір засобів проєктування і розробки	16
2.2 Визначення функціональних вимог до програмного забезпечення	17
2.3 Проєктування бази даних	21
2.4 Визначення структури програми.....	22
2.5 Висновки за розділом 2	24
3 Розробка програми	25
3.1 Алгоритм функціонування програми.....	25
3.2 Опис реалізованих одиниць вебзастосунку.....	29
3.3 Висновки за розділом 3	37
4 Експлуатація та тестування програми	35
4.1 Опис програми.....	35
4.2 Експлуатація програми.....	35
4.3 Тестування програми	43
4.4 Висновки за розділом 4	43
Висновки	44
Перелік джерел посилання	46
Додаток А Технічне завдання	46
Додаток Б Текст програми	50
Додаток В Слайди презентації.....	79

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CORS	–(англ. Cross-Origin Resource Sharing) - спільне використання ресурсів з різних джерел;
API	–(англ. application programming interface) - програмний інтерфейс програми, інтерфейс прикладного програмування;
CSS	–(англ. Cascading Style Sheets) - каскадні таблиці стилів;
HTML	–(англ. Hyper Text Markup Language) – мова розмітки гіпертексту;
HTTP	–(англ. Hyper Text Transfer Protocol) - протокол передачі даних, що використовується в комп'ютерних мережах;
IDE	–(англ. Integrated Development Environment) - паралельний інтерфейс підключення накопичувачів (жорстких дисків і оптичних приводів) до комп'ютера;
JSON	–(англ. JavaScript Object Notation) - текстовий формат обміну даними, заснований на JavaScript;
ORM	–(англ. <i>Object-relational mapping</i> , Об'єктно-реляційна проєкція) — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних»;
SEQUELIZE	–це сучасна ORM для роботи з Node.js яка надає можливість працювати з Postgres, MySQL, MariaDB, SQLite та SQL Server тощо;
NODE.JS	–програмна платформа, заснована на двигуні V8, що перетворює JavaScript з вузькоспеціалізованої

POSTGRES	–вільна об'єктно-реляційна система управління базами даних;
SASS	–це метамова на основі CSS, призначена для збільшення рівня абстракції CSS-коду та спрощення файлів каскадних таблиць стилів, вона має два синтаксиси: sass - відрізняється відсутністю фігурних дужок, у ньому вкладені елементи реалізовані за допомогою відступів;
REST	–це спосіб взаємодії сайтів та веб-додатків із сервером;
WEB SOCKET	–протокол зв'язку поверх TCP-з'єднання, призначений для обміну повідомленнями між браузером та веб-сервером у режимі реального часу;
PAGINATION	–порядкова нумерація сторінок.

ВСТУП

З моменту створення глобальної мережі Internet веб-розробка зазнала багато суттєвих змін. Розробникам треба було створювати окремі програми для кожної операційної системи та встановлювати їх на комп'ютер для зручного використання. Ці програми мали назву – десктоп додатки. На противагу цьому, веб-додатки неважко достати з браузера незалежно від операційної системи. Веб-додатки досягли величезного прогресу після того, як десктоп - додатки та статичні веб-сайти перейшли до сучасних, інтерактивних та інтелектуальних веб-додатків. Користувачі могли читати тільки вміст статичних вебсайтів. Однак через появу веб – додатків користувачі отримують можливість взаємодіяти з сервером, слухати аудіо і дивитися відео, а також малювати на екрані.

Розробка інтернет магазину - це поєднання фронтенд та бекенд розробки. У розпорядженні розробника велика кількість мов програмування та фреймворків для створення веб додатку. Раніше не існувало мов програмування, які могли б виконувати як фронтенд, так і бекендрозробку додатку. Довгий час найпопулярнішим стеком для повностекових веб-розробки була зв'язка LAMP stack: Linux, Apache, MySQL і PHP.

PHP в основному використовувався для бекенду, а HTML, CSS і JavaScript були основними стеками для розробки фронтенду, в той час як MySQL використовувався для баз даних. Apache використовувався як веб-сервер в операційній системі Linux у стеки LAMP. Проте для розробки веб – додатку з використанням стека LAMP розробникам потрібно було вивчити декілька мов, які використовують різний синтаксис і мають різну природу написання коду. Крім цього, з появою Node.js в якості серверної платформи JavaScript зміг забезпечити розробку повностекового програмного забезпечення в рамках однієї мови програмування. Незважаючи на те, що Node.js є найбільш поширеною технологією для розробки бекенд-систем, існує безліч фреймворків, таких як Angular, React, Vue.js, а також шаблонні

двигуни (шаблонизатор), такі як EJS, Handlebars або Pug. Метою дипломної роботи було вивчення різних аспектів веб-розробки за допомогою Full Stack JavaScript та розробка програми на її основі, а також використання Node.js, Express, Postgres та React при розробці веб-додатків Full Stack. Більше того, він також зосереджений на вивченні різних версій JavaScript та їх використання в платформі Node.js. Різні версії ECMAScript були детально вивчені, особливо ES6 як один з найрозповсюдженіших стандартів. Потім були вивчені особливості та реалізація платформ на основі JavaScript, таких як Node.js, Express та Postgres, та порівняно з іншими технологіями. Як результат, за допомогою Node.js та Express було розроблено інтернетмагазин комп'ютерних комплектуючих з використанням основних пакетів і модулів Node.js, а також проміжного програмного забезпечення Express. Інші сторонні проміжні програми також використовувались для розробки веб-застосунку. Повнотекстовий JavaScript вважається найкращою технологією для розробки сучасних, масштабованих та безпечних вебзастосунків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Світовий ринок інтернет магазинів

Продаж та купівля через всесвітню мережу Internet зростають у всьому світі (рис. 1.1), Лідером інтернет продаж є Східна Азія , вона генерує близько 50% доходів , а Китай за рівнем продажу обігнав США за рівнем інтернет-комерції.

Лідерами світового ринку є компанії з Китаю , США та Великобританії. Гіганти ринку заробляють мільйони на своїх сайтах .Тому ця ниша є не тільки цікавою, а й дуже перспективною.

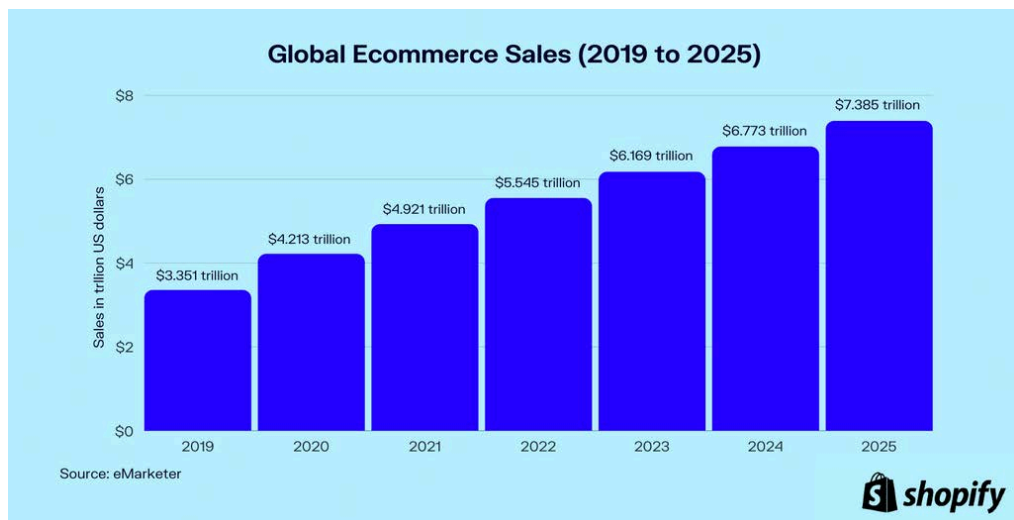


Рисунок 1.1 – Графік популярності продажів через Інтернет[1]

Одною з найпопулярніших сфер для інтернет продажу , є ринок комп'ютерних комплектуючих.Зокрема товари таких гігантів як AMD, NVIDIA та INTEL.

1.2 Огляд програмних аналогів

Оскільки, було проаналізовано раніше, продаж товарів через інтернет не тільки наших аналогів а й зарубіжних є одним з найпопулярніших видів збуту товарів існує багато вебсайтів, які надають можливість купівлі товарів.

У даному підрозділі вони будуть розглянуті в якості програмних аналогів і проаналізовані щодо можливостей які надають користувачу.

Першим програмним аналогом інтернет магазину Komputronik [2] (рис. 1.2).

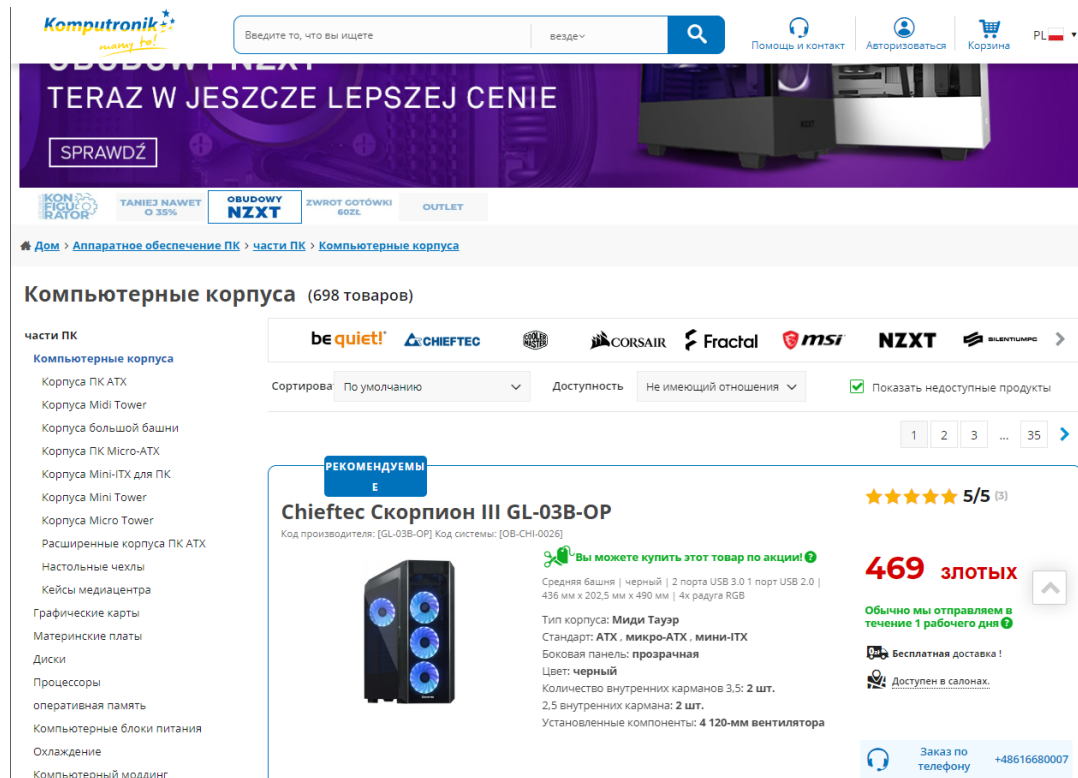


Рисунок 1.2 – Інтернет магазин Komputronik[2]

Інтернет магазин Komputronik надає такий функціонал:

- можливість перегляду асортименту товарів
- сортування та пошук товарів;
- можливість реєстрації/авторизації користувача ;
- додавання товарів у кошик;
- зміна мови локалізації сайту.

До недоліків даного інтернет магазину потрібно віднести відсутність можливості зв'язку з користувачем у режимі онлайн.

Другим аналогом є вебсайт Itplanet (рис. 1.3). Даний магазин має застарілий дизайн , відсутність спілкування з потенційним клієнтом [3].



Рисунок 1.3 – Інтернет магазин Itplanet[3]

Надається можливість переглядати список товарів , відсортувати товари або за потреби знайти потрібний товар за найменуванням, подивитись товар детальніше та якщо клієнт бажає придбати товар додати у корзину та заповнити контактну форму.

Даний вебсайт не має можливості реєстрації і авторизації.

На противагу другому аналогу і відповідній групі аналогів, які можна сюди додати, розглянемо інтернет магазин, повністю – Neon (рис. 1.4) [4].

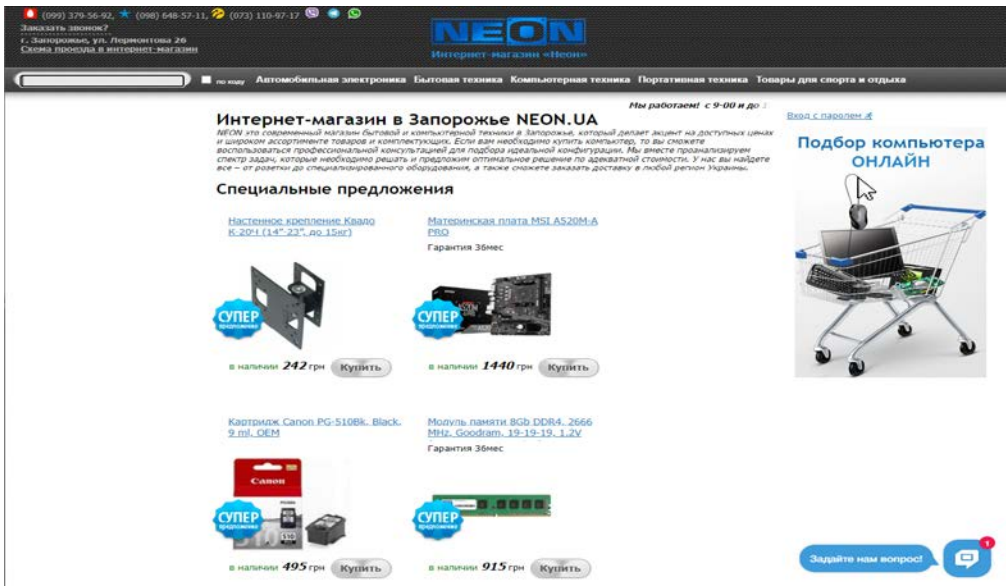


Рисунок 1.4 – Інтернет магазин Neon[4]

Даний вебсайт також не має засобів авторизації та реєстрації, але він має можливість зв'язку з клієнтом за допомогою чату, Окрім того можна виконувати пошук товарів не лише за найменуванням , а й за унікальним ідентификатором ID та є загальна можливість сортування товарів.

1.3 Висновки за розділом 1

Перший розділ направлений на аналіз предметної області, в межах якого було виділено основні поняття і невеликий екскурс в історію що до стандартів веб-розробки раніше, та взагалі рентабельність інтернет-магазину .

Інтернет-магазин – це ідеальне спасіння для підприємця, який хоче покращити та прискорити сервіс, отримати більше прибутку, та піклуватися про своїх клієнтів, особливо у часи війни чи пандемії.

Також було проведено огляд програмних аналогів, включаючи такі інтернет магазини: Komputronik, Itplanet, Neon. Проведений аналіз продемонстрував, що частина цих магазинів має деякі недоліки, а саме:

- відсутність зворотнього зв'язку з покупцем;
- не зручний чи застарілий інтерфейс;
- відсутність засобів індивідуального доступу.

Мені особливо сподобався магазин Komputronik за виключенням можливості зворотнього зв'язку.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів проєктування і розробки

Оскільки створення даного вебзастосунку має певну специфіку – передбачає роботу з не дуже великими наборами даних по розміру, таких як: товари, замовлення та користувачі, щоб накопичувати та обробляти ці данні, то є вірним рішенням обрати для розробки основної мови, яка на даний є дуже популярною та ефективною, – Javascript [5].

У подальшому виникло питання вибору конкретного вебфреймворку серед Express [6] і Коа [7], щодо якого за сукупністю показників (табл. 2.1) було обрано Express.

Таблиця 2.1 – Вибір фреймворку розробки вебзастосунку

Критерій	Express	Коа
Зручна робота з даними	Так	Так
Надає більше корисних інструментів з коробки	Так	Ні
Величезне ком'юніті та підтримка фреймворка	Так	Так
Просте налаштування роутингу	Так	Так
Швидкість роботи	Так	Ні

2.2 Визначення функціональних вимог до програмного забезпечення

Програмне забезпечення інтернет магазину комп'ютерних комплектуючих повинно надавати можливість роботи з цілим набором функціоналу, які мають використовуватися для зручної роботи:

- пошук товарів за найменуванням;
- сортування товарів за категоріями;
- сортування товарів за ціною;
- детальний перегляд товару;
- порівняння товарів;
- додавання товарів до кошика;
- розподілення ролей Адмін, Користувач та Менеджер;
- налаштування аккаунту;
- отримання повідомлення на електронну пошту при замовленні;
- додавання або редагування товарів Адміном;
- можливість зв'язку з менеджером;
- авторизація та реєстрація користувача;
- блокування користувача;
- можливість підтвердити чи відмінити замовлення;
- додавання або корегування знижок;
- контактна інформація ;
- можливість розрахунку в кредит;
- гарантійна інформація;
- видалення користувача.

У відповідності з даними функціональними можливостями визначено сценарії роботи неавторизованого користувача (рис. 2.1):

- пошук товарів;
- сортування товарів;
- детальний перегляд товару;

- контактна інформація;
- можливості розрахунку в кредит;
- гарантійна інформація;
- порівняння товарів;
- реєстрація користувача.



Рисунок 2.1 – Діаграма можливостей неавторизованого користувача

Сценарії роботи авторизованого користувача (рис. 2.2):

- додавання товарів у корзину;
- налаштування своїх персональних даних;
- налаштування адреси доставки;
- зв'язок з менеджером за допомогою чату;
- можливість відновлення паролю користувача;
- авторизація користувача.



Рисунок 2.2 – Діаграма можливостей авторизованого користувача

Сценарії роботи ролі менеджера (рис. 2.3):

- перегляд усіх товарів;
- сортування товарів;
- пошук товарів;
- налаштування персональних даних;
- зміна статусу замовлення;
- можливість відновлення паролю;
- детальний перегляд товару;
- редагування замовлення;
- відправлення повідомлення;
- авторизація менеджера.



Рисунок 2.3 – Діаграма прецедентів менеджера

Сценарії роботи ролі адміна (рис. 2.4):

- перегляд усіх товарів;
- сортування товарів;
- пошук товарів;
- налаштування персональних даних;
- детальний перегляд товару;
- перегляд усіх замовлень;
- можливість відновлення пароллю;
- видалення замовлення;
- редагування знижок;
- додавання товарів;
- редагування товарів;
- перегляд усіх користувачів;
- видалення користувачів;
- акційне розсилання;
- авторизація адміністратора.



Рисунок 2.4 – Діаграма прецедентів адміністратора

2.3 Проектування бази даних

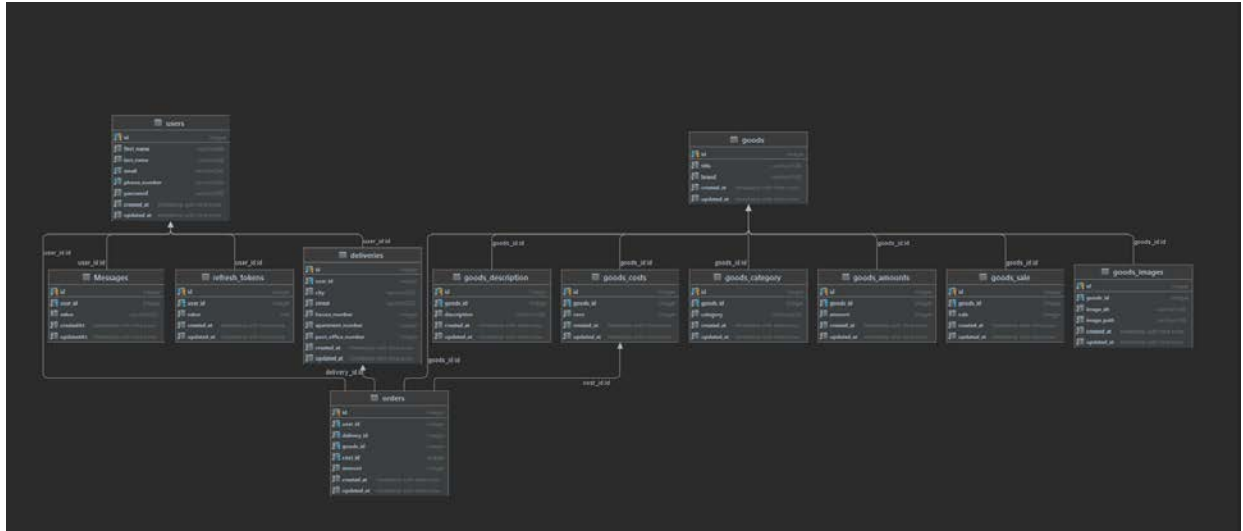
Базу даних було створено на основі системи керування базами даних Postgres [8].

За результатами вивчення предметної області, виконаного в розділі 1, було сформовано список основних сутностей предметної області зі створенням відповідних таблиць бази даних:

- goods – товар;
- goods_amounts – ціна товару;
- goods_category – кетегорія товару;
- goods_costs – ціна товару;
- goods_description опис товару;
- goods_images – зображення товару;
- goods_sale – значка на товар;
- users – користувачі;
- orders – замовлення;
- refresh_tokens – токени користувачів;

- deliveries – данні користувача о доставке;
- messages – повідомлення користувачів.

Схема створеної бази даних представлена на рис. 2.5.



Рисунку 2.5 – Схема бази даних

2.4 Визначення структури програми

Структуру програми яка зображена на рис. 2.6 побудовано на основі структури Rest api, яка передбачається застосуванням вебфреймворку Express.

Технологію REST API застосовують скрізь, де користувачеві сайту або веб-програми потрібно надати дані з сервера. Наприклад, при натисканні іконки з відео на відеохостингу REST API проводить операції та запускає ролик із сервера у браузері. В даний час це найпоширеніший спосіб організації API. Він витіснив раніше популярні методи SOAP і WSDL.

Rest api має 7 принципів роботи:

- відділення клієнта від сервера (клієнт — це інтерфейс сайту або програми, наприклад, пошуковий рядок відеохостингу, у REST API код запитів залишається на стороні клієнта, а код доступу до даних на стороні сервера. це спрощує організацію API, дозволяє легко переносити інтерфейс

користувача на іншу платформу і дає можливість краще масштабувати серверне зберігання даних);

- відсутність запису стану клієнта – сервер не повинен зберігати інформацію про стан (проведені операції) клієнта, кожен запит від клієнта повинен містити тільки інформацію, яка потрібна для отримання даних від сервера;

- кешування – у даних запиту має бути зазначено, чи потрібно кешувати дані (зберігати у спеціальному буфері для частих запитів), якщо така вказівка є, клієнт отримує право звертатися до цього буфера за потреби;

- єдність інтерфейсу (усі дані повинні запитуватись через одну URL-адресу стандартними протоколами, наприклад, HTTP, це спрощує архітектуру сайту або програми та робить взаємодію з сервером зрозумілішим);

- багаторівневність системи (у RESTful сервері можуть розташовуватися різних рівнях, у своїй кожен сервер взаємодіє лише з найближчими рівнями і пов'язаний запитами з іншими);

- надання коду на запит – сервери можуть надсилати клієнтові код (наприклад, скрипт для запуску відео), так загальний код програми або сайту стає складнішим лише за необхідності;

- початок від нуля (клієнт знає лише одну точку входу на сервер, подальші можливості взаємодії забезпечуються сервером).

Окрім безпосередньо самої методології Rest було використано фреймворк express та ORM sequelize для зручної роботи з базою даних Postgres, створено окремо налаштування застосунку та деякі статичні файли, до яких зокрема належать скрипти Javascript. Окрім базової роботи вони також забезпечують роботу з даними на стороні клієнта. Це пояснюється тим, що не завжди потрібно відображати одразу всі дані користувачу, а у відповідності до поставленої задачі, користувач сам обирає що йому потрібно відобразити, наприклад пагінація чи кількість товарів [9].



Рисунок 2.6 – Методологія Rest[9]

Розробники керуються принципами REST API для створення ефективної роботи з серверами для своїх сайтів та додатків. Принципи дозволяють будувати серверну архітектуру за допомогою інших протоколів: HTTP, URL, JSON та XML.

2.5 Висновки за розділом 2

Виконано вибір фреймворку Express для розробки вебзастосунку після порівняння з фреймворком Коа як найбільш популярних та ефективних фреймворків розробки вебзастосунків мовою JavaScript. Була розібрана методологія Rest. Виділено основні сценарії роботи авторизованого і неавторизованого користувачів, менеджера та адміністратора з вебзастосунком. Побудовано схему бази даних та визначено структуру програми.

3 РОЗРОБКА ПРОГРАМИ

3.1 Алгоритм функціонування програми

Функціональність, яка надається програмою для авторизованого і неавторизованого користувачів (рис.3.1), пов'язана в основному з відображенням певної інформації у відповідності до ролі. В даному підрозділі описано детально логіку роботи усіх чотирьох ролей менеджер, адміністратор користувач та не авторизований користувач для визначення алгоритму функціонування застосунку.

Під час роботи з програмою не авторизований користувач може переглядати такі розділи як каталог усіх товарів, детальний перегляд одного товару, порівняння товарів, може сортувати товари за категоріями та шукати за найменуванням товару, переглядати гарантійну інформацію, контактні данні та інформацію о розрахуванні у кредит і також має можливість зареєструватись.

Авторизований користувач має доступ до всіх тих розділів що й не авторизований користувач та ще до таких як додання товарів у корзину можливість спілкування з менеджером, налаштування даних користувача та адреси доставки товарів.

Менеджер має доступ до основного функціоналу програми. Вони передбачають можливість перегляду усіх замовлень, його пошук, та сортування по даті, редагування замовлень, а також можливість спілкування з користувачем. Після того як користувач зробив замовлення, менеджер може змінити статус замовлення, або якщо користувач допустив помилку наприклад у кількості замовленого товару відредагувати замовлення, чи видалити зайвий товар. Також менеджер має можливість корегувати знижки на якийсь окремий товар або групу товарів.

Адміністратор має доступ до усіх розділів що й не зареєстрований користувач та ще інші які відповідають його ролі, а саме: перегляд усіх замовлень, пошук та сортування замовлень, видалення замовлень,

налаштування свого профілю, зміна знижок, релагування замовлень включно з статусом, додавання товарів, видалення та редагування, має можливість блокування або видалення користувача та акційну розсилку зареєстрованим користувачам.

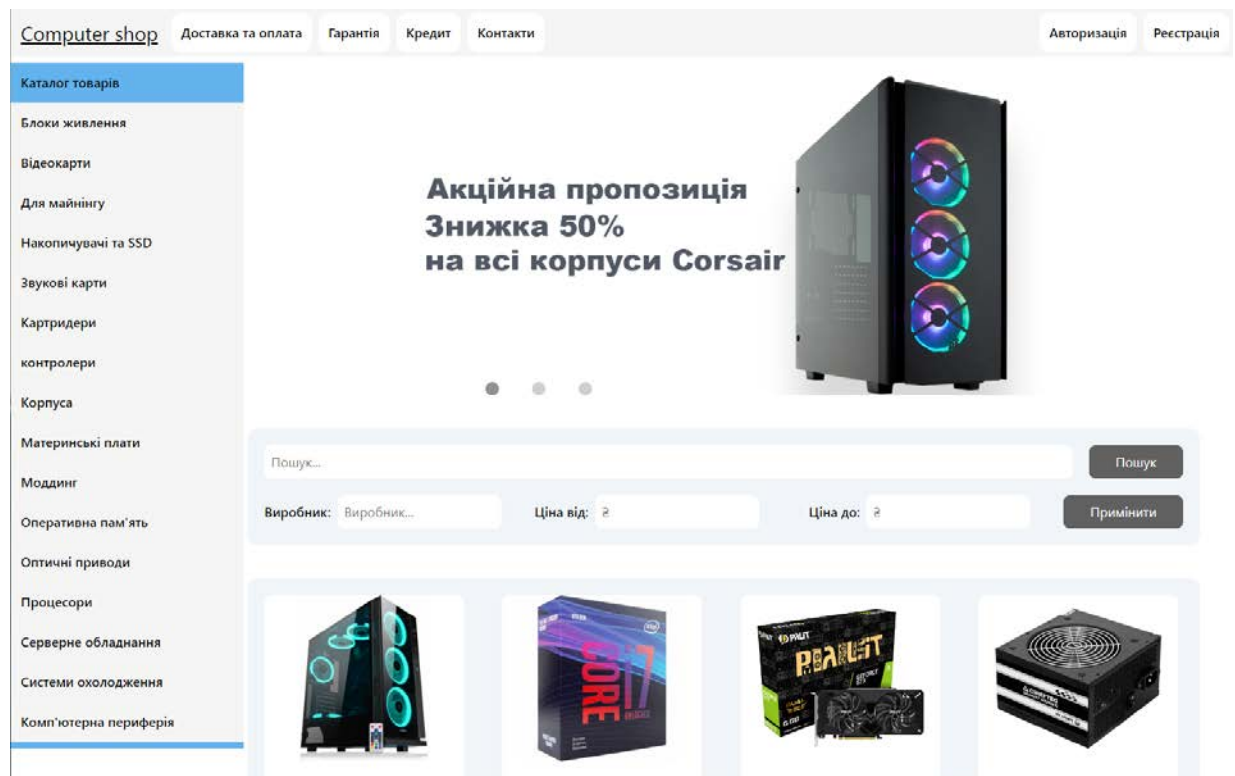


Рисунок 3.1 – Сторінка функціонування застосунку для неавторизованного користувача

3.2 Опис реалізованих одиниць вебзастосунку

Модель користувача організована за допомогою наступних її атрибутів:

- firstName – ім'я користувача що має тип DataTypes.STRING(64) та у неї присутні такі перевірки як : notNull: true та notEmpty: true, що дає гарантію що в базу не буде записано пусту строку чи взагалі нічого ці перевірки є обов'язковими і для наступних атрибутів моделі ;

- lastName – фамілія користувача також має тип DataTypes.STRING(64);

- email – електронна пошта користувача тип `DataTypes.STRING(256)`;
- role – роль користувача тип `DataTypes.ENUM('user', 'admin', 'manager')`;
- phoneNumber – номер користувача `DataTypes.INTEGER`;
- password – пароль користувача та наступний тип `DataTypes.STRING(256)` ще за допомогою ORM Sequelize зроблено хеширування паролю не на рівні запиту чи проміжного програмного забезпечення, а на рівні моделі за допомогою такої бібліотеки як `bcrypt`.

Модель доставки має наступні атрибути:

- `userId` – це поле зсилається на модель користувача за її унікальним ідентифікатором ID та має тип `type: DataTypes.INTEGER`, відношення між цими таблицями має назву один до одного;
- `city` – місто куди буде доставлене замовлення тип `DataTypes.STRING(128)`;
- `postOfficeNumber` – номер поштового відділення доставки `DataTypes.STRING(128)`;
- `apartmentNumber` – номер будівлі тип `DataTypes.INTEGER`;
- `houseNumber` – номер дому або квартири тип `DataTypes.INTEGER`;
- `street` – назва вулиці тип `DataTypes.STRING(128)`.

При замовленні користувач має можливість вибору доставки на відділення чи до дому, при замовленні ці данні будуть потрібні для оформлення накладної доставки.

Модель повідомлень має наступні атрибути :

- `userId` – це поле зсилається на модель користувача за її унікальним ідентифікатором ID та має тип `type: DataTypes.INTEGER`, відношення між цими таблицями має назву один до багатьох;
- `value`: поле самого повідомлення `DataTypes.STRING(512)`.

Модель refreshToken має наступні атрибути :

- userId – це поле зсилається на модель користувача за її унікальним ідентифікатором ID та має тип type: DataTypes.INTEGER, відношення між цими таблицями має назву один до багатьох;
- value: зберігається токен відновлення сесії.

Модель товарів має наступні атрибути :

- title –заголовок товару DataTypes.STRING(128) це поле та усі інші мають перевірку на пусту строку чи null;
- brand – DataTypes.STRING(128).

Модель goodsDescription має наступні атрибути :

- goodsId – це поле зсилається на модель товарів за її унікальним ідентифікатором ID та має тип type: DataTypes.INTEGER, відношення між цими таблицями має назву один до багатьох;
- description – дитальний опис товару його властивості DataTypes.STRING(128).

Модель goodsCategory має наступні атрибути :

- goodsId – це поле зсилається на модель товарів за її унікальним ідентифікатором ID та має тип type: DataTypes.INTEGER, відношення між цими таблицями має назву один до одного;
- category – категорія товару DataTypes.STRING(128).

Модель goodsCost має наступні атрибути:

- goodsId – це поле зсилається на модель товарів за її унікальним ідентифікатором ID та має тип type: DataTypes.INTEGER, відношення між цими таблицями має назву один до одного;
- cost – DataTypes.STRING(128) ціна товару.

Модель goodsAmount має наступні атрибути:

- goodsId – це поле зсилається на модель товарів за її унікальним ідентифікатором ID та має тип type: DataTypes.INTEGER, відношення між цими таблицями має назву один до одного;
- amount – DataTypes.STRING(128) відобрадає кількість доступних одениць товару.

Модель goodsImages має наступні атрибути:

- goodsId – це поле зсилається на модель товарів за її унікальним едентифікатором ID та має тип type: DataTypes.STRING(128), відношення між цими таблицями має назву один до багатьох;
- image – шлях до зображення DataTypes.STRING(128).

Модель goodsSale має наступні атрибути:

- goodsId – це поле зсилається на модель товарів за її унікальним ідентифікатором ID та має тип type: DataTypes.INTEGER, відношення між цими таблицями має назву один до одного;
- sale – DataTypes.STRING(128) відображає можливу знижку на товар це поли перевіряється на пусту строку та може буду 0.

Модель Замовлення має наступні атрибути :

- userId – це поле зсилається на модель користувача за її унікальним ідентифікатором ID та бере звідси данні користувача;
- deliveryId – це поле зсилається на модель доставок за її унікальним ідентифікатором ID та бере звідси данні що до відправлення замовлення;
- goodsId – це поле зсилається на модель товарів за її унікальним ідентифікатором ID та бере звідси данні що до товару;
- costId – це поле зсилається на модель товарів за її унікальним ідентифікатором ID та бере ціну товару;
- amount – відображає кількість замовлень у товарі;

- createdAt– відображає коли був створений заказ;
- updatedAt– відображає чи був він оновлений , за замовчанням дата оновлення якщо замовлення не редагувалось буде така як і створення.

Для роботи з базою даних були створені контролери за допомогою яких проходить взаємодія з самими моделями для реалізації функціональних можливостей, які були визначені в вимогах до програми.

В свою чергу для того щоб відпрацював контроллер були створені аі роути , при зверненні до котрих ми можемо отримати наприклад виборку з бази даних о всіх товарах або за одним товаром.

Клієнт робить запити до сервера за допомогою http клієнта Axios[10], отриманий запит обробляється саги і оновлюють стан завантажуючи данні до сховища та змінюють атрибути стану (рис. 3.2), такі як isLoading або у разі виникнення помилок то записують їх у атрибут error.

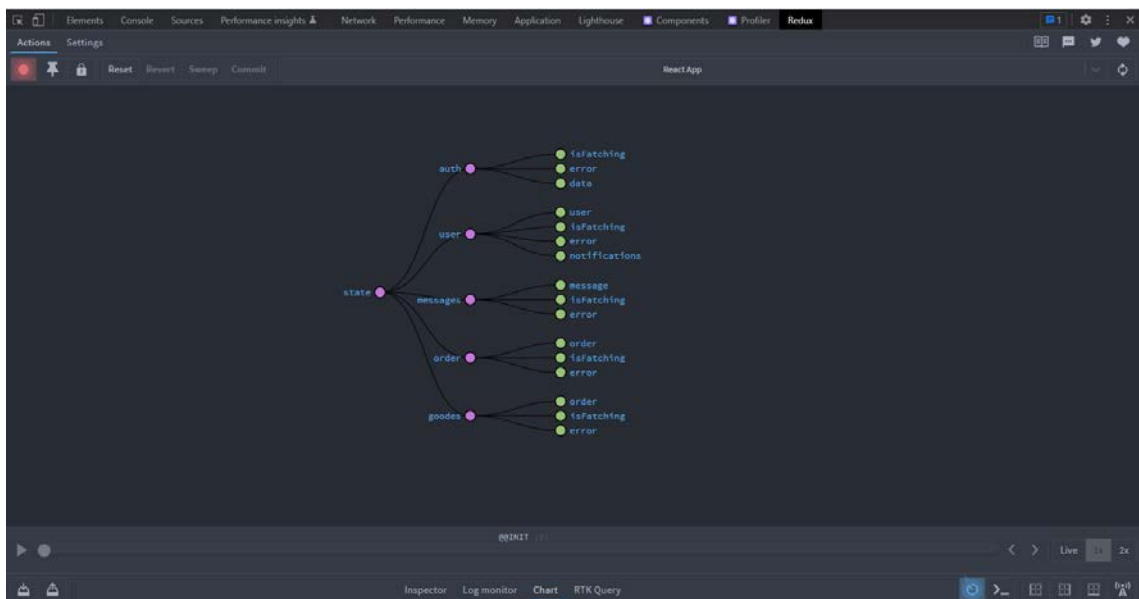


Рисунок 3.2 – Схема сховища стану

Коли данні вже отримані у стан Redux їх можна відобразити на самих сторінках за допомогою компонентів [11].

```
router.get('/get', GoodsController.getGoods);
```

```
router.get ('/get/:id', GoodsController.getAllGoods);
```

Для отримання інформації про товари був реалізований `goods.controller.js`.

```
module.exports.getGoods = async (req, res, next) => {  
  try {  
    const goods = await Goods.findAll({  
      include: [  
        {  
          model: GoodsCategory,  
          attributes: ['category'],  
  
        },  
        {  
          model: GoodsCost,  
          attributes: ['cost'],  
  
        },  
        {  
          model: GoodsDescription,  
          attributes: ['description'],  
  
        },  
        {  
          model: GoodsImages,  
          attributes: ['imageAlt', 'imagePath'],  
  
        },  
      ],  
    });  
    res.json(goods);  
  } catch (error) {  
    next(error);  
  }  
}
```

```

        model: GoodsSale,
        attributes: ['sale'],

    },
  ],
});

res.send(jane);
} catch (error) {
  next(error);
}
};

```

Доречі ось можливості ORM sequelize через те що таблиці зв'язані це дає нам можливість дуже зручної роботи з ними , не треба звертатись до кожної з таблиць а за допомогою викристання методів sequelize вони підключаються до одного запиту [12]. Отримані данні далі передаються на клієнтську частину та записуються у стан додатку де потім передаються у компонент і користувач може їх бачити.

Застосунок використовує такі типи HTTP запитів :

- GET – отримання ресурсу;
- POST – створення ресурсу;
- PATCH – часткове оновлення ресурсу;
- DELETE – видалення ресурсу.

Вебзастосунок має можливість відправки електронного листа на пошту користувача якщо він оформив замовлення, у листі будуть вказані деталі замовлення або користувач забув пароль новий пароль буде відправлено на пошту або відправка рекламної розсилки чи акційних пропозицій, що до відновлення паролю користувача буде необхідно заповнити форму відновлення вказавши пошту та номер телефону .

Всі ці маніпуляції з відправкою електронних листів доступні завдяки модулю nodeMailer [13].

Також окремо хотів би виділити реалізацію чату користувача та менеджера за допомогою вебсокитів (рис. 3.3). Коротко про них, це протокол, що призначений для обміну інформацією між браузером та вебсервером в режимі реального часу. Він забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет.

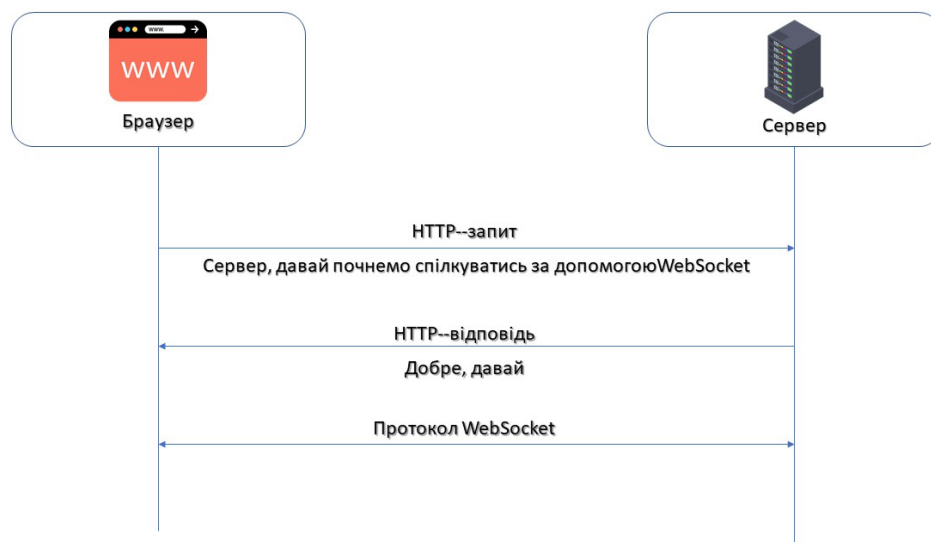


Рисунок 3.3 – Схема функціонування вебсокету

Фрагмент коду з файлу socket.js:

```

const io = SocketServer(server, cors)
const SocketServer = require('socket.io')
io.on('connection', socket => {
  console.log('connection to socket')
  socket.on(SOCKET_EVENTS.NEW_MESSAGE, async newMessage => {
    try {
      const savedMessage = await Message.create(newMessage)
      io.emit(SOCKET_EVENTS.NEW_MESSAGE, savedMessage)
    } catch (error) {
      io.emit(SOCKET_EVENTS.NEW_MESSAGE_ERROR, error)
    }
  })
})
  
```

```

    }
  }),
  socket.on(SOCKET_EVENTS.GET_MESSAGE, async () => {
    try {
      const message = await Message.find()
      socket.emit(SOCKET_EVENTS.GET_MESSAGE, message)
    } catch (error) {
      socket.emit(SOCKET_EVENTS.GET_MESSAGE_ERROR, error)
    }
  })
  socket.on('disconnect', reason => {
    console.log(reason);
  })
})

```

3.3 Висновки за розділом 3

Описано моделі , роути та контрооллери для взаємодії з даними на стороні сервера. Описано основні реалізовані одиниці програми з детальним описом їх структури та принципи роботи вебсокету та його реалізація у проекті.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Опис програми

Робота з каталогом твоарів є дуже важливою , оскільки від цієї інформації залежить взагалі роботопридатність інтернет магазину. Продаж товарів через інтернет є дуже розповсюдженим та перспективним, а тому такий інструмент для реалізації продукції є дуже корисним. Подібні програмні аналоги існують, проте вони не мають засобів індивідуальної роботи та зворинного зв'язку, що є важливим, оскільки це забезпечує комфортне використання магазину. Саме на реалізацію даної функціональності і направлено роботу данного додатку .

Для підтримки роботи програми необхідно, щоб апаратна частина включала процесор (4 ядра 2 потоки), оперативну пам'ять (від 4 Гб), жорсткий диск з наявним вільним простором (від 700 Мб).

4.2 Експлуатація програми

Для експлуатації програми потрібно звернутися за URL-адресою, після чого відкриється вебсторінка, яка визначає роботу користувача без авторизації. Дана сторінка має набір пунктів меню (рис. 4.1), що включають:

- каталог товарів: можна переглянути усі доступні пропозиції для купівлі їх;
- контактну інформацію: клієн може переглянути графік роботи та місце знаходження точок видачі;
- інформацію о кредитах: умови розстрочи за товар;
- гарантійна інформація: терм гарантії та які умови гарантійного повернення;
- порівняння товарів: користувач порівнювати декілька товарів, та обрати кращий.

Computer shop
Доставка та оплата
Гарантія
Кредит
Контакти
Профіль
Вихід

Каталог товарів
Блоки живлення
Відеокарти
Для майнінгу
Накопичувачі та SSD
Звукові карти
Картридери
контролери
Корпуса
Материнські плати
Моддинг
Оперативна пам'ять
Оптичні приводи
Процесори
Серверне обладнання
Системи охолодження
Комп'ютерна периферія

Налаштування

Будь ласка, введіть дані !
Піліп
П'ятерня
philipp@gmail.com
+380685765371
Password
Re-enter password
Оновити

Рисунок 4.1 – Налаштування аккаунту користувача

Як тільки користувач опиняється на сайті, він бачить виборку товарів з різних категорій, це зроблено для того щоб зацікавити користувача та показати асортимент магазину, для комфорту роботи з ресурсом зроблена можливість змінення категорій сортування за ціною та пошук за найменуванням товару, коли користувач знайшов цікаву йому пропозицію. Він може переглядати окремі товари (рис. 4.2).

Йому необхідно пройти процес реєстрації де необхідно заповнити форму .

Форма проходить валідацію за допомогою бібліотеки формік.

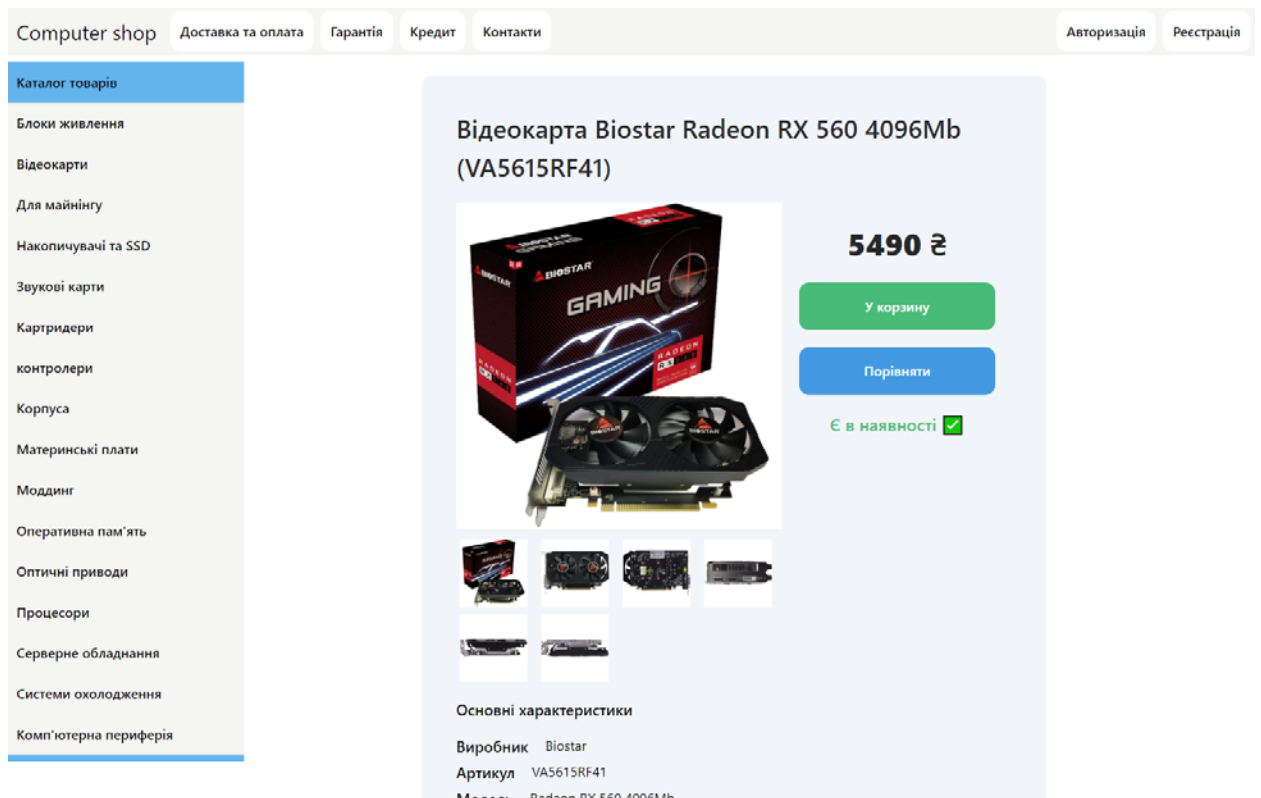


Рисунок 4.2 – Одиниця товару

У випадку коли користувач авторизовувався він може додавати товари до козину і оформити замовлення (рис. 4.3) після внесення інформації що до доставки важливо підмітити що ця інформація вноситься лише один раз і за потреби користувач може змінити адресу доставки чи свої данні аккаунту, якщо у користувача виникли якісь питання то у реальному часі він може написата у чат, де у свою чергу менеджер надасть своєчасну відповідь.

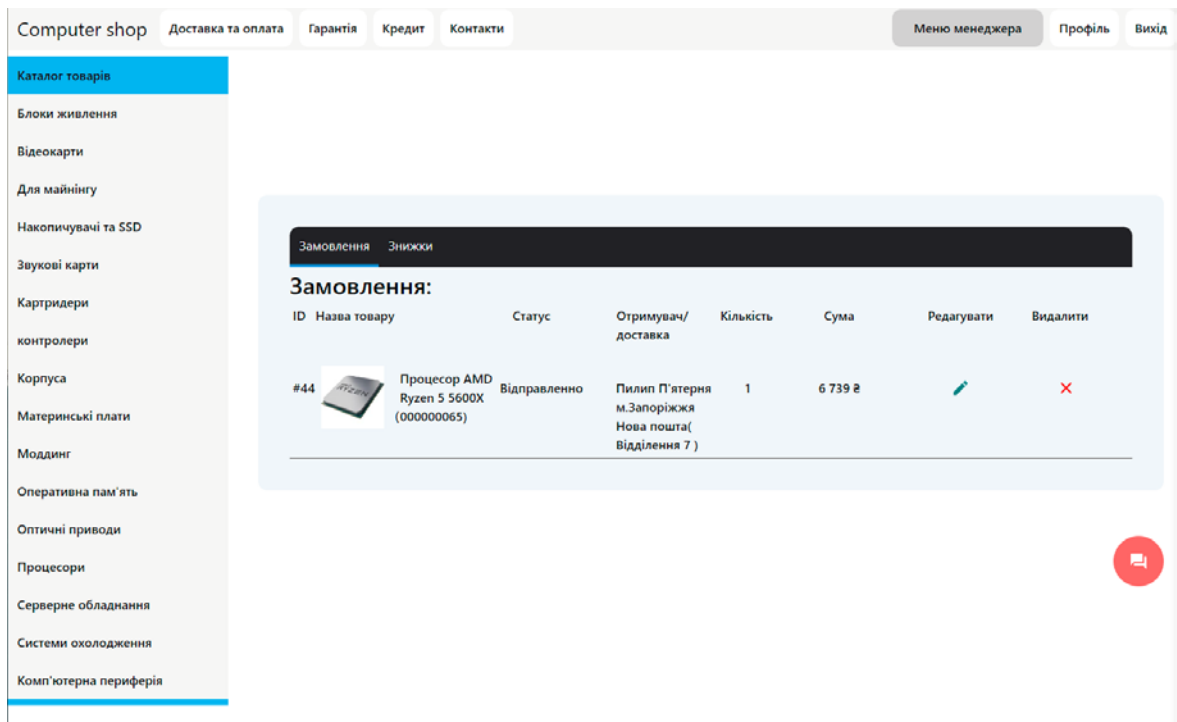


Рисунок 4.3 – Функціонал менеджера, перегляд замовлень

Якщо авторизований користувач є менеджером, він має наступні можливості (рис. 4.4):

- перегляд усіх товарів;
- детальний перегляд одного товару;
- сортування за категоріями;
- пошук товарів за найменуванням;
- сортування товарів за ціною;
- налаштування персональних даних;
- перегляд усіх замовлень;
- сортування замовлень;
- редагування замовлень;
- детальний перегляд одного замовлення;
- зміна статусу замовлення;
- можливість відправляти повідомлення.

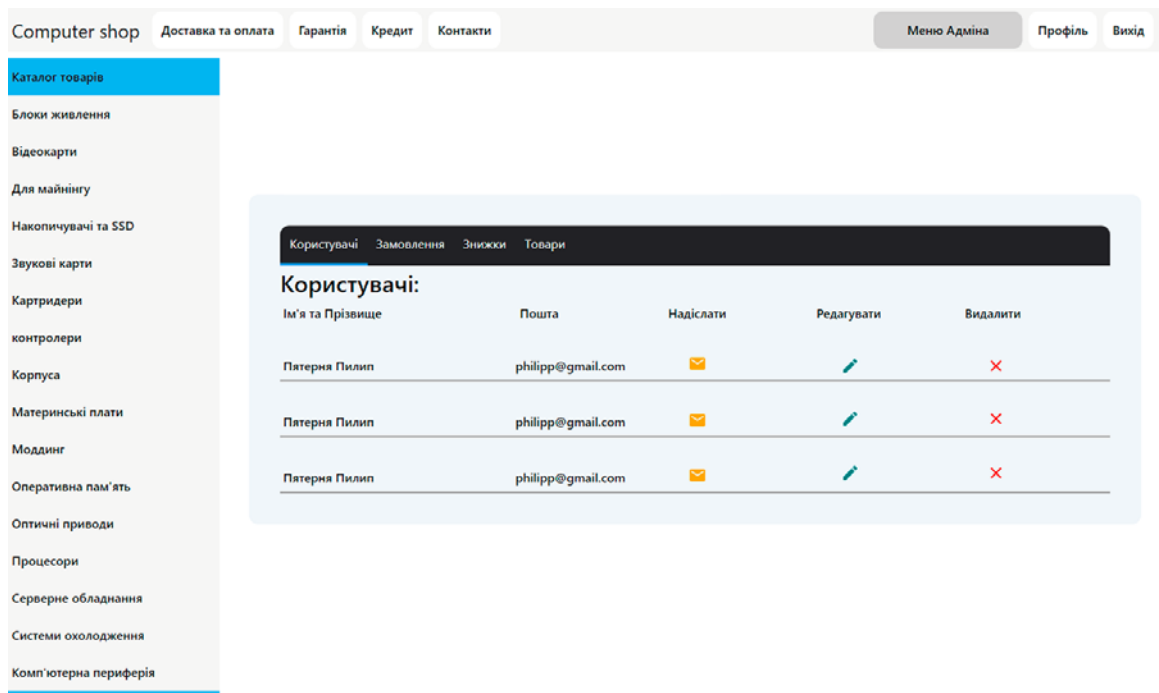


Рисунок 4.4 – Панель адміна , робота з користувачами

Якщо авторизований користувач є адміном, він має наступні можливості:

- перегляд усіх товарів;
- детальний перегляд одного товару;
- сортування за категоріями;
- пошук товарів за найменуванням;
- сортування товарів за ціною;
- налаштування персональних даних;
- перегляд усіх замовлень;
- сортування замовлень;
- редагування замовлень;
- видалення замовлень;
- детальний перегляд одного замовлення;
- додавання товарів;
- редагування товарів;
- видалення товарів;

- перегляд усіх користувачів;
- видалення користувачів;
- акційна розсилка на пошту користувачів.

4.3 Тестування програми

Тестування програми було здійснено за двома напрямками.

Перший напрямок включав перевірку реалізації функціональних вимог програмою. Для цього спочатку було розроблено набір тестів, що зокрема включали коректне відображення товарів, відображення одного товару,

Додавання товарів з ролі адміна, видалення та редагування товарів.

Та доже важливи частина тестування коректне відпрацювання ролей. Після завершення роботи було окремо перевірено повністю весь список функціональних вимог, підтвердивши дієздатність.

Другий напрямок включав роботу з вебзастосунком через різні браузери. Перевірка включала браузери Chrome (зокрема версія 102.0.5005.115), Opera (зокрема версія 75.0.3969.17), Microsoft Edge (зокрема версія 102.0.1245.41). Результати підтвердили, що програма є працездатною у всіх протестованих браузерах.

4.4 Висновки за розділом 4

Програма працює в режимах неавторизованого користувача, авторизованого користувача, менеджера та адміністратора кожен з яких було описано. Виконано успішне тестування вебзастосунку.

ВИСНОВКИ

Виконано аналіз предметної області, в межах якого було виділено основні поняття та методології що до розробки інтернет магазину, проведено огляд програмних аналогів, включаючи вебсайти ItPlanet, Komputronik, Neon. Враховуючи в них відсутність підтримки зворотнього зв'язку, реєстрації та застарілий дизайн, саме це було визначено в якості особливості розробленого вебзастосунку.

Розробку інтернет магазину комп'ютерних комплектуючих було виконано за допомогою вебфреймворку Express з визначенням структури даних на основі використання бази даних, вимог до програми, розробленням структури програми і алгоритму роботи на основі цих вимог, реалізацією відповідних одиниць вебзастосунку. Для розробки використано мови програмування JavaScript, мову розмітки HTML, мову стилю сторінок CSS, вебфреймворки Express, React, Tailwind, систему керування базами даних Postgres.

Програма дозволяє переглядати список усіх товарів сортувати їх за ціною чи категорією та шукати товари за найменуванням, реалізована реєстрація авторизація, зміна даних аккаунту користувача за потребою, детальний перегляд товарів, порівняння їх та додання в корзину, а також можливість спілкування з менеджером за допомогою чату.

Розроблене програмне забезпечення дозволяє підвищити зручність пошуку та замовлення товарів, та за потреби покупець має можливість проконсультуватись онлайн.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Global Ecommerce Sales statistic [Electronic resource]. – Access mode: <https://www.shopify.ca/blog/global-ecommerce-sales>.
2. Shop Komputronik [Electronic resource]. – Access mode: <https://www.komputronik.pl>.
3. Shop Itplanet [Електронний ресурс]. – Режим доступу: <https://www.itplanet.zp.ua/>.
4. Shop Neon [Електр. ресурс]. – Режим доступу: <https://neon.ua/>.
5. JavaScript electronic modern resource [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/>.
6. Express Fast, unopinionated, minimalist web framework for Node.js [Electronic resource]. – Access mode: <https://expressjs.com/>.
7. Koa next generation web framework for node.js Documentation [Electronic resource]. – Access mode: <http://koajs.cn>.
8. The World's Most Advanced Open Source Relational Database [Electronic resource]. – Access mode: <https://www.postgresql.org/docs/>.
9. Web services architecture [Electronic resource]. – Access mode: <https://legiacong.blogspot.com/2021/03/web-services-5-rest.html>.
10. Axios is a promise-based HTTP Client for node.js and the browser. [Electronic resource]. – Access mode: <https://axios-http.com/docs/intro>.
11. A Redux Predictable State Container for JS Apps [Electronic resource]. – Access mode: <https://redux.js.org/>.
12. Sequelize is a modern TypeScript and Node.js ORM for Postgres, MySQL, MariaDB, SQLite and SQL Server, and more. Featuring solid transaction support, relations, eager and lazy loading, read replication and more. [Electronic resource]. – Access mode: <https://sequelize.org/>.
13. Nodemailer is a module for Node.js applications to allow easy as cake email sending Documentation [Electronic resource]. – Access mode: <https://nodemailer.com/about/>.

ДОДАТОК А
Технічне завдання

Вступ

Реалізація товарів за допомогою онлайн торгівлі це перспектива розвитку магазину на довгі роки. Зараз існує низка мов програмування які цілком задовольняють потреби користувача.

А.1 Підстави для розробки

Дипломна кваліфікаційна робота виконується на основі завдання, виданого за темою «Розробка інтернет магазину комп'ютерних комплектуючих» у відповідності з наказом № 102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка».

А.2 Призначення розробки

Програма має використовуватися для отримання каталогу товарів можливості сортування, пошуку та замовлення товарів , а також можливістю отримання зворотнього зв'язку з користувачем.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Набір функціональних вимог, які має забезпечувати програма, включає:

- пошук товарів;
- сортування товарів за найменванням;
- сортування товарів за ціною;
- сортування товарів за категорією;
- перегляд контактних даних;
- інформація о кредетуванні;
- гарантійна інформація;

- порівняння товарів;
- реєстрація користувача;
- додавання товарів у корзину;
- авторизація користувача;
- можливість зворотнього зв'язку;
- розподілення ролей;
- додавання товарів;
- видалення товарів;
- перегляд усіх замовлень;
- пошук замовлень;
- сортування замовлень;
- можливість відновлення паролю користувача;
- видалення користувачів;
- зміна статусу замовлення;
- редагування замовлення;
- відображення списку усіх користувачів;
- можливість розсилки акційних пропозицій користувачам.

A.3.2 Умови експлуатації

Умовами експлуатації є виділення менеджера та адміністратора для обробки замовлень та додавання товарів. Менеджер може змінювати замовлення редагувати його за потреби та спілкуватись за потреби з користувачем, адміністратор може додавати редагувати та видаляти товари може видаляти користувачів та робити поштову розсилку.

А.3.3 Вимоги до складу та параметрів технічних засобів

Для підтримки роботи програми необхідно, щоб апаратна частина включала процесор (4 ядра, не менше 2,2 ГГц), оперативну пам'ять (від 4 Гб), жорсткий диск з наявним вільним простором (від 500 Мб).

А.4 Порядок контролю та приймання

Дипломна кваліфікаційна робота має виконуватися у відповідності з визначеним календарним планом згідно встановлених завдань, їх строків і результатів виконання.

ДОДАТОК Б
Текст програми

Б.1 Текст файла models.js

```
const { Model } = require('sequelize');

module.exports = (sequelize, DataTypes) => {
  class Delivery extends Model {
    static associate(models) {
      // Delivery.belongsToMany(models.User, {
      //   foreignKey: 'orders',
      //   targetKey: 'deliveryId'
      // });
    }
  }
  Delivery.init({
    userId: {
      type: DataTypes.INTEGER,
      field: 'user_id',
      allowNull: false,
      references: {
        model: 'User',
        key: 'id',
        onUpdate: 'cascade',
        onDelete: 'cascade'
      },
      validate: {
        notNull: true,
      }
    },
    city: {
      type: DataTypes.STRING(128),
      allowNull: false,
      validate: {
        notNull: true,
        notEmpty: true
      }
    },
    street: {
      type: DataTypes.STRING(128),
      allowNull: false,
      validate: {
        notNull: true,
```

```

        notEmpty: true
      }
    },
    houseNumber: {
      type: DataTypes.INTEGER,
      allowNull: false,
      validate: {
        notNull: true,
        notEmpty: true
      }
    },
    apartmentNumber: {
      type: DataTypes.INTEGER,
      allowNull: false,
      validate: {
        notNull: true,
        notEmpty: true
      }
    },
    postOfficeNumber: {
      type: DataTypes.INTEGER,
      allowNull: false,
      validate: {
        notNull: true,
        notEmpty: true
      }
    },
  }, {
    sequelize,
    modelName: 'Delivery',
    tableName: 'delivery',
    underscored: true
  });
  return Delivery;
};

'use strict';
module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.createTable('deliveries', {
      id: {
        allowNull: false,

```

```
    autoIncrement: true,
    primaryKey: true,
    type: Sequelize.INTEGER
  },
  userId: {
    type: Sequelize.INTEGER,
    field: 'user_id',
    allowNull: false,
    references: {
      model: 'users',
      key: 'id',
      onUpdate: 'cascade',
      onDelete: 'cascade'
    }
  },
  city: {
    type: Sequelize.STRING,
    allowNull: false,
  },
  street: {
    type: Sequelize.STRING,
    allowNull: false,
  },
  houseNumber: {
    field: 'house_number',
    type: Sequelize.INTEGER,
    allowNull: false,
  },
  apartmentNumber: {
    field: 'apartment_number',
    type: Sequelize.INTEGER,
    allowNull: false,
  },
  postOfficeNumber: {
    field: 'post_office_number',
    type: Sequelize.INTEGER,
    allowNull: false,
  },
  createdAt: {
    allowNull: false,
    type: Sequelize.DATE,
```

```

        field: 'created_at',
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE,
        field: 'updated_at',
      }
    });
  },
  async down(queryInterface, Sequelize) {
    await queryInterface.dropTable('deliveries');
  }
};

module.exports = (sequelize, DataTypes) => {
  class Goods extends Model {
    static associate(models) {
      Goods.hasMany(models.GoodsImages, {
        foreignKey: 'goodsId',
        targetKey: 'id'
      });
      Goods.hasOne(models.GoodsSale, {
        foreignKey: 'goodsId',
        targetKey: 'id'
      });
      Goods.hasMany(models.GoodsCategory, {
        foreignKey: 'goodsId',
        targetKey: 'id'
      });
      Goods.hasMany(models.GoodsDescription, {
        foreignKey: 'goodsId',
        targetKey: 'id'
      });
      Goods.hasOne(models.GoodsAmount, {
        foreignKey: 'goodsId',
        targetKey: 'id'
      });
      Goods.hasOne(models.GoodsCost, {
        foreignKey: 'goodsId',
        targetKey: 'id'
      });
      // Goods.belongsToMany(models.User, {

```

```

        //   foreignKey: 'orders',
        //   targetKey: 'goodsId'
        // });
    }
}
Goods.init({
  title: {
    type: DataTypes.STRING(128),
    allowNull: false,
    validate: {
      notNull: true,
      notEmpty: true
    }
  },
  brand: {
    type: DataTypes.STRING(128),
    allowNull: false,
    validate: {
      notNull: true,
      notEmpty: true
    }
  },
}, {
  sequelize,
  modelName: 'Goods',
  tableName: 'goods',
  underscored: true
});
return Goods;
};
'use strict';
module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.createTable('goods', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      title: {

```

```

        type: Sequelize.STRING(128),
        allowNull: false,
      },
      brand: {
        type: Sequelize.STRING(128),
        allowNull: false,
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE,
        field: 'created_at',
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE,
        field: 'updated_at',
      }
    });
  },
  async down(queryInterface, Sequelize) {
    await queryInterface.dropTable('goods');
  }
};

module.exports = (sequelize, DataTypes) => {
  class GoodsAmount extends Model {
    static associate(models) {
      GoodsAmount.belongsTo(models.Goods, {
        foreignKey: 'goodsId',
        targetKey: 'id',
      })
    }
  }
}

GoodsAmount.init({
  goodsId: {
    type: DataTypes.INTEGER,
    field: 'goods_id',
    allowNull: false,
    references: {
      model: 'Goods',
      key: 'id',
      onUpdate: 'cascade',
    },
  },

```

```

        onDelete: 'cascade'
      },
      validate: {
        notNull: true,
      }
    },
    amount: {
      type: DataTypes.INTEGER,
      allowNull: false,
    },
  }, {
    sequelize,
    modelName: 'GoodsAmount',
    tableName: 'goods_amounts',
    underscored: true,
  });
  return GoodsAmount;
};

'use strict';
module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.createTable('goods_amounts', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      goodsId: {
        type: Sequelize.INTEGER,
        field: 'goods_id',
        allowNull: false,
        references: {
          model: 'goods',
          key: 'id',
          onUpdate: 'cascade',
          onDelete: 'cascade'
        }
      },
      amount: {
        type: Sequelize.INTEGER,

```

```

        allowNull: false,
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE,
        field: 'created_at',
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE,
        field: 'updated_at',
      }
    });
  },
  async down(queryInterface, Sequelize) {
    await queryInterface.dropTable('goods_amounts');
  }
};

module.exports = (sequelize, DataTypes) => {
  class GoodsCategory extends Model {
    static associate(models) {
      GoodsCategory.belongsTo(models.Goods, {
        foreignKey: 'goodsId',
      })
    }
  }
  GoodsCategory.init({
    goodsId: {
      type: DataTypes.INTEGER,
      field: 'goods_id',
      allowNull: false,
      references: {
        model: 'Goods',
        key: 'id',
        onUpdate: 'cascade',
        onDelete: 'cascade'
      },
    },
    validate: {
      notNull: true,
    }
  },
  },

```

```

    category: {
      type: DataTypes.STRING(64),
      allowNull: false,
    }
  }, {
    sequelize,
    modelName: 'GoodsCategory',
    tableName: 'goods_category',
    underscored: true,
  });
  return GoodsCategory;
};

'use strict';
module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.createTable('goods_category', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      goodsId: {
        type: Sequelize.INTEGER,
        field: 'goods_id',
        allowNull: false,
        references: {
          model: 'goods',
          key: 'id',
          onUpdate: 'cascade',
          onDelete: 'cascade'
        }
      },
      category: {
        type: Sequelize.STRING,
        allowNull: false,
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE,
        field: 'created_at',

```

```

    },
    updatedAt: {
      allowNull: false,
      type: Sequelize.DATE,
      field: 'updated_at',
    }
  });
},
async down(queryInterface, Sequelize) {
  await queryInterface.dropTable('goods_category');
}
};

module.exports = (sequelize, DataTypes) => {
  class GoodsCost extends Model {
    static associate(models) {
      GoodsCost.belongsTo(models.Goods, {
        foreignKey: 'goodsId',
        targetKey: 'id',
      })
    }
  }
}

GoodsCost.init({
  goodsId: {
    type: DataTypes.INTEGER,
    field: 'goods_id',
    allowNull: false,
    references: {
      model: 'Goods',
      key: 'id',
      onUpdate: 'cascade',
      onDelete: 'cascade'
    },
  },
  validate: {
    notNull: true,
  }
},
cost: {
  type: DataTypes.INTEGER,
  allowNull: false,
}

```

```

    }, {
      sequelize,
      modelName: 'GoodsCost',
      tableName: 'goods_costs',
      underscored: true,
    });
    return GoodsCost;
  };
  'use strict';
  module.exports = {
    async up(queryInterface, Sequelize) {
      await queryInterface.createTable('goods_costs', {
        id: {
          allowNull: false,
          autoIncrement: true,
          primaryKey: true,
          type: Sequelize.INTEGER
        },
        goodsId: {
          type: Sequelize.INTEGER,
          field: 'goods_id',
          allowNull: false,
          references: {
            model: 'goods',
            key: 'id',
            onUpdate: 'cascade',
            onDelete: 'cascade'
          }
        },
        cost: {
          type: Sequelize.INTEGER,
          allowNull: false,
        },
        createdAt: {
          allowNull: false,
          type: Sequelize.DATE,
          field: 'created_at',
        },
        updatedAt: {
          allowNull: false,
          type: Sequelize.DATE,

```

```

        field: 'updated_at',
      }
    });
  },
  async down(queryInterface, Sequelize) {
    await queryInterface.dropTable('goods_costs');
  }
};

module.exports = (sequelize, DataTypes) => {
  class GoodsDescription extends Model {
    static associate(models) {
      GoodsDescription.belongsTo(models.Goods, {
        foreignKey: 'goodsId',
        targetKey: 'id',
      })
    }
  }
  GoodsDescription.init({
    goodsId: {
      type: DataTypes.INTEGER,
      field: 'goods_id',
      allowNull: false,
      references: {
        model: 'Goods',
        key: 'id',
        onUpdate: 'cascade',
        onDelete: 'cascade'
      },
    },
    validate: {
      notNull: true,
    }
  }, {
    description: {
      type: DataTypes.STRING(128),
      allowNull: false,
    }
  }, {
    sequelize,
    modelName: 'GoodsDescription',
    tableName: 'goods_description',
    underscored: true,
  });
};

```

```

    });
    return GoodsDescription;
  };
  'use strict';
  module.exports = {
    async up(queryInterface, Sequelize) {
      await queryInterface.createTable('goods_description', {
        id: {
          allowNull: false,
          autoIncrement: true,
          primaryKey: true,
          type: Sequelize.INTEGER
        },
        goodsId: {
          type: Sequelize.INTEGER,
          field: 'goods_id',
          allowNull: false,
          references: {
            model: 'goods',
            key: 'id',
            onUpdate: 'cascade',
            onDelete: 'cascade'
          }
        },
        description: {
          type: Sequelize.STRING(128),
          allowNull: false,
        },
        createdAt: {
          allowNull: false,
          type: Sequelize.DATE,
          field: 'created_at',
        },
        updatedAt: {
          allowNull: false,
          type: Sequelize.DATE,
          field: 'updated_at',
        }
      });
    },
    async down(queryInterface, Sequelize) {

```

```

    await queryInterface.dropTable('goods_description');
  }
};

module.exports = (sequelize, DataTypes) => {
  class GoodsImages extends Model {
    static associate(models) {
      GoodsImages.belongsTo(models.Goods, {
        foreignKey: 'goodsId',
        targetKey: 'id',
      })
    }
  }
  GoodsImages.init({
    goodsId: {
      type: DataTypes.INTEGER,
      field: 'goods_id',
      allowNull: false,
      references: {
        model: 'Goods',
        key: 'id',
        onUpdate: 'cascade',
        onDelete: 'cascade'
      },
      validate: {
        notNull: true,
      }
    },
    imageAlt: {
      field: 'image_alt',
      type: DataTypes.STRING(128),
      allowNull: false,
    },
    imagePath: {
      field: 'image_path',
      type: DataTypes.STRING(128),
      allowNull: false,
    },
  }, {
    sequelize,
    modelName: 'GoodsImages',
    tableName: 'goods_images',
  })
}

```

```

    underscored: true,
  });
  return GoodsImages;
};

module.exports = (sequelize, DataTypes) => {
  class GoodsSale extends Model {
    static associate(models) {
      GoodsSale.belongsTo(models.Goods, {
        foreignKey: 'goodsId',
        targetKey: 'id',
      })
    }
  }
  GoodsSale.init({
    goodsId: {
      type: DataTypes.INTEGER,
      field: 'goods_id',
      allowNull: false,
      references: {
        model: 'Goods',
        key: 'id',
        onUpdate: 'cascade',
        onDelete: 'cascade'
      },
      validate: {
        notNull: true,
      }
    },
    sale: {
      type: DataTypes.INTEGER,
      validate: {
        min: 5,
        max: 80
      }
    }
  }, {
    sequelize,
    modelName: 'GoodsSale',
    tableName: 'goods_sale',
    underscored: true,
  });
};

```

```

    });
    return GoodsSale;
  };
  'use strict';
  module.exports = {
    async up(queryInterface, Sequelize) {
      await queryInterface.createTable('goods_sale', {
        id: {
          allowNull: false,
          autoIncrement: true,
          primaryKey: true,
          type: Sequelize.INTEGER
        },
        goodsId: {
          type: Sequelize.INTEGER,
          field: 'goods_id',
          allowNull: false,
          references: {
            model: 'goods',
            key: 'id',
            onUpdate: 'cascade',
            onDelete: 'cascade'
          }
        },
        sale: {
          type: Sequelize.INTEGER
        },
        createdAt: {
          allowNull: false,
          type: Sequelize.DATE,
          field: 'created_at',
        },
        updatedAt: {
          allowNull: false,
          type: Sequelize.DATE,
          field: 'updated_at',
        }
      });
    },
    async down(queryInterface, Sequelize) {
      await queryInterface.dropTable('goods_sale');
    }
  };

```

```

    }
  };
  module.exports = (sequelize, DataTypes) => {
    class Messages extends Model {

      automatically.
        static associate(models) {
          }
        }
      Messages.init({
        userId: DataTypes.STRING,
        value: DataTypes.STRING
      }, {
        sequelize,
        modelName: 'Messages',
      });
      return Messages;
    };
    'use strict';
    module.exports = {
      async up(queryInterface, Sequelize) {
        await queryInterface.createTable('messages', {
          id: {
            allowNull: false,
            autoIncrement: true,
            primaryKey: true,
            type: Sequelize.INTEGER
          },
          userId: {
            type: Sequelize.INTEGER,
            field: 'user_id',
            allowNull: false,
            references: {
              model: 'users',
              key: 'id',
              onUpdate: 'cascade',
              onDelete: 'cascade'
            }
          },
          value: {
            type: Sequelize.STRING

```

```

    },
    createdAt: {
      allowNull: false,
      type: Sequelize.DATE
    },
    updatedAt: {
      allowNull: false,
      type: Sequelize.DATE
    }
  });
},
async down(queryInterface, Sequelize) {
  await queryInterface.dropTable('messages');
}
};
module.exports = (sequelize, DataTypes) => {
  class RefreshToken extends Model {
    static associate({ User }) {
      RefreshToken.belongsTo(User, {
        foreignKey: 'userId',
      })
    }
  }
};
RefreshToken.init({
  userId: {
    type: DataTypes.INTEGER,
    field: 'user_id',
    allowNull: false,
    references: {
      model: 'User',
      key: 'id',
      onUpdate: 'cascade',
      onDelete: 'cascade'
    },
  },
  validate: {
    notNull: true,
  }
},
{
  value: {
    type: DataTypes.TEXT,
    allowNull: false,

```

```

        validate: {
          notNull: false,
        }

      },

    }, {
      sequelize,
      modelName: 'RefreshToken',
      tableName: 'refresh_tokens',
      underscored: true,
    });
    return RefreshToken;
  };
  'use strict';
  module.exports = {
    up: async (queryInterface, Sequelize) => {
      await queryInterface.createTable('refresh_tokens', {
        id: {
          allowNull: false,
          autoIncrement: true,
          primaryKey: true,
          type: Sequelize.INTEGER
        },
        userId: {
          type: Sequelize.INTEGER,
          field: 'user_id',
          allowNull: false,
          references: {
            model: 'users',
            key: 'id',
            onUpdate: 'cascade',
            onDelete: 'cascade'
          }
        },
        value: {
          type: Sequelize.TEXT,
          allowNull: false,
        },
        createdAt: {
          field: 'created_at',

```

```

        allowNull: false,
        type: Sequelize.DATE
      },
      updatedAt: {
        field: 'updated_at',
        allowNull: false,
        type: Sequelize.DATE
      }
    });
  },
  down: async (queryInterface, Sequelize) => {
    await queryInterface.dropTable('refresh_tokens');
  }
};

const bcrypt = require('bcrypt');
const { SALT_ROUNDS } = require('../constants')

async function hashPassword(user, options) {
  if (user.changed('password')) {
    const { password } = user
    const hashedPassword = await bcrypt.hash(password, SALT_ROUNDS)
    user.password = hashedPassword
  }
}

module.exports = (sequelize, DataTypes) => {
  class User extends Model {
    static associate(models) {
      User.hasOne(models.Delivery, {
        foreignKey: 'orders',
        targetKey: 'userId'
      });

      User.hasMany(models.RefreshToken, {
        foreignKey: 'userId',
        targetKey: 'id'
      });

      User.hasMany(models.Messages, {
        foreignKey: 'userId',
        targetKey: 'id'
      });
    }
  }
};

```

```

    });
  }
  async comparePassword(password) {
    return await bcrypt.compare(password, this.getDataValue('password'))
  }
}
User.init({
  firstName: {
    type: DataTypes.STRING(64),
    field: 'first_name',
    allowNull: false,
    validate: {
      notNull: true,
      notEmpty: true
    }
  },
  lastName: {
    type: DataTypes.STRING(64),
    field: 'last_name',
    allowNull: false,
    validate: {
      notNull: true,
      notEmpty: true
    }
  },
  email: {
    type: DataTypes.STRING(256),
    allowNull: false,
    unique: true,
    validate: {
      notNull: true,
      notEmpty: true,
      isEmail: true,
    }
  },
  phoneNumber: {
    field: 'phone_number',
    type: DataTypes.STRING(32),
    allowNull: false,
    unique: true,
    validate: {

```

```

        notNull: true,
        notEmpty: true,
      }
    },
    password: {
      type: DataTypes.STRING(256),
      allowNull: false,
      validate: {
        notNull: true,
        notEmpty: true,
      }
    },
  }, {
    sequelize,
    modelName: 'User',
    tableName: 'users',
    underscored: true
  });
User.beforeCreate(hashPassword);

User.beforeBulkUpdate(async (user, options) => {
  if (user.attributes.password) {
    const { attributes: { password } } = user
    const hashedPassword = await bcrypt.hash(password, SALT_ROUNDS)
    user.attributes.password = hashedPassword
  }
});

return User;
};

'use strict';

const fs = require('fs');
const path = require('path');
const Sequelize = require('sequelize');
const basename = path.basename(__filename);
const env = process.env.NODE_ENV || 'development';
const config = require(__dirname + '/../config/config.json')[env];
const db = {};

```

```

let sequelize;
if (config.use_env_variable) {
  sequelize = new Sequelize(process.env[config.use_env_variable], config);
} else {
  sequelize = new Sequelize(config.database, config.username, config.password,
config);
}

fs
  .readdirSync(__dirname)
  .filter(file => {
    return (file.indexOf('.') !== 0) && (file !== basename) && (file.slice(-
3) === '.js');
  })
  .forEach(file => {
    const model = require(path.join(__dirname, file))(sequelize,
Sequelize.DataTypes);
    db[model.name] = model;
  });

Object.keys(db).forEach(modelName => {
  if (db[modelName].associate) {
    db[modelName].associate(db);
  }
});

db.sequelize = sequelize;
db.Sequelize = Sequelize;

module.exports = db;

```

Б.2 Текст файла /controller.js

```

const createHttpError = require('http-errors');
const { User, RefreshToken } = require('../models');
const AuthService = require('../services/authService');

module.exports.signIn = async (req, res, next) => {
  try {

```

```

    const {
      body: { email, password },
    } = req;
    const user = await User.findOne({
      where: { email },
    });
    if (user && (await user.comparePassword(password))) {
      const data = await AuthService.createSession(user);
      return res.send({ data });
    }
    next(createHttpError(401, 'Invalid credentials'));
  } catch (error) {
    console.log(error);
    next(error);
  }
};

module.exports.signUp = async (req, res, next) => {
  try {
    const { body } = req;
    const user = await User.create(body);
    if (user) {
      const data = await AuthService.createSession(user);
      return res.send({ data });
    }
    next(createHttpError(406, 'User already exists'));
  } catch (error) {
    next(error);
  }
};

module.exports.refresh = async (req, res, next) => {
  try {
    const { body: { refreshToken } } = req;

    const refreshTokenInstance = await RefreshToken.findOne({
      where: {
        value: refreshToken
      }
    })
  }
};

```

```

        const data = await AuthService.refreshSession(refreshTokenInstance);

        res.send({ data });
    } catch (error) {
        next(error);
    }
};

const { createTokenPair } = require('./jwtService')
const { MAX_DEVICES_AMOUNT } = require('../constants')

module.exports.createSession = async user => {
    const tokenPair = await createTokenPair(user)
    if ((await user.countRefreshTokens()) >= MAX_DEVICES_AMOUNT) {
        const [oldestToken] = await user.getRefreshTokens({
            order: [['updatedAt', 'ASC']],
            limit: 1,
        })
        await oldestToken.update({ value: tokenPair.refresh })
    } else {
        await user.createRefreshToken({ value: tokenPair.refresh })
    }
    return { tokenPair, user }
}

module.exports.refreshSession = async refreshTokenInstance => {
    const user = await refreshTokenInstance.getUser()
    const tokenPair = await createTokenPair(user)
    await refreshTokenInstance.update({ value: tokenPair.refresh })
    return { tokenPair, user }
}

const { promisify } = require('util');
const jwt = require('jsonwebtoken');
const {
    ACCESS_TOKEN_SECRET,
    ACCESS_TOKEN_TIME,
    REFRESH_TOKEN_SECRET,
    REFRESH_TOKEN_TIME,
} = require('../constants');
```

```

const signJWT = promisify(jwt.sign);
const verifyJWT = promisify(jwt.verify);

const tokenConfig = {
  access: {
    secret: ACCESS_TOKEN_SECRET,
    time: ACCESS_TOKEN_TIME,
  },
  refresh: {
    secret: REFRESH_TOKEN_SECRET,
    time: REFRESH_TOKEN_TIME,
  },
};

const createToken = async (payload, { time, secret }) => {
  const accessToken = await signJWT(
    {
      userId: payload.id,
      email: payload.email,
    },
    secret,
    { expiresIn: time },
  );

  return accessToken;
};

module.exports.createTokenPair = async payload => {
  return {
    access: await createToken(payload, tokenConfig.access),
    refresh: await createToken(payload, tokenConfig.refresh),
  };
};

const verifyToken = (token, { secret }) => verifyJWT(token, secret);

module.exports.decodeToken = token => jwt.decode(token);

module.exports.verifyRefreshToken = token =>
  verifyToken(token, tokenConfig.refresh);

```

```

module.exports.verifyAccessToken = token =>
  verifyToken(token, tokenConfig.access);
const JwtService = require('../services/jwtService');
const createHttpError = require('http-errors');

module.exports.checkRefreshToken = async (req, res, next) => {
  try {
    const {
      body: { refreshToken },
    } = req;

    req.tokenData = await JwtService.verifyRefreshToken(refreshToken);

    next();
  } catch (error) {
    console.log(error);
    next(createHttpError(401, 'Wrong refresh token'));
  }
};

module.exports.checkAccessToken = async (req, res, next) => {
  try {
    const {
      headers: { authorization: accessToken },
    } = req;

    if (accessToken) {
      req.tokenData = await JwtService.verifyAccessToken(accessToken);
      return next();
    }
    next(createHttpError(419, 'Need token'));
  } catch (error) {
    next(error);
  }
};

const createHttpError = require('http-errors');
const schemas = require('../validationSchemes/schemas');

module.exports.validateSignUp = async (req, res, next) => {

```

```

    const validationResult = await schemas.signUpSchem.isValid(req.body);
    if (validationResult) {
        return next();
    }
    next(createHttpError('Invalid data for signUp'));
};

```

```

module.exports.validateSignIn = async (req, res, next) => {
    const validationResult = await schemas.signInSchem.isValid(req.body);
    if (validationResult) {
        return next();
    }
    next(createHttpError('Invalid data for login'));
};

```

```

const yup = require('yup');

```

```

module.exports.signUpSchem = yup.object().shape({
    firstName: yup.string().required().min(1),
    lastName: yup.string().required().min(1),
    email: yup.string().email().required().min(4),
    phoneNumber: yup.string().required().min(1),
    password: yup.string().required().min(8),
});

```

```

module.exports.signInSchem = yup.object().shape({
    email: yup.string().email().required().min(4),
    password: yup.string().required().min(8),
});

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»

Дипломна кваліфікаційна робота бакалавра

**Інтернет магазин комп'ютерних комплектуючих Node.js
Online store of computer components node.js**

Виконав

студент групи КНТ-219сп

П.О. П'ятерня

Керівник

О.М. Гладкова

2022

Рисунок В.1 – Слайд № 1

Об'єкт, предмет, мета, завдання роботи

Об'єкт роботи – процес надання користувачу каталогу товарів на основі котрих може бути сформоване замовлення

Предмет роботи – інтернет магазин комп'ютерних комплектуючих

Мета роботи – розробити програмне забезпечення для надання користувачу каталогу товарів для формування замовлення, можливості спілкування з менеджером та .

Завдання роботи:

- ✓ аналіз програмних аналогів;
- ✓ визначення функціональних вимог до програми надання;
- ✓ проєктування програми;
- ✓ реалізація і налагодження програми.

2

Рисунок В.2 – Слайд № 2

Вибір фреймворку розробки вебзастосунку		
Критерій	Express	Koa
Зручна робота з даними	Так	Так
Надає більше корисних інструментів з коробки	Так	Ні
Величезне ком'юніті та підтримка фреймворку	Так	Так
Просте налаштування роутингу	Так	Так
Швидкість роботи	Так	Ні

3

Рисунок В.3 – Слайд № 3



Рисунок В.4 – Слайд № 4



Рисунок В.5 – Слайд № 5

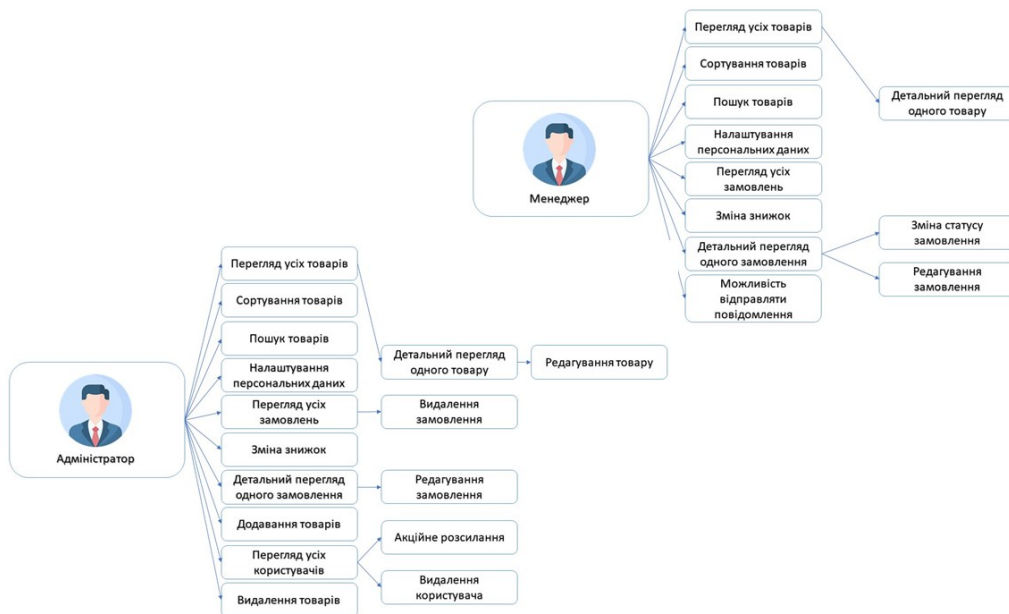


Рисунок В.6 – Слайд № 6

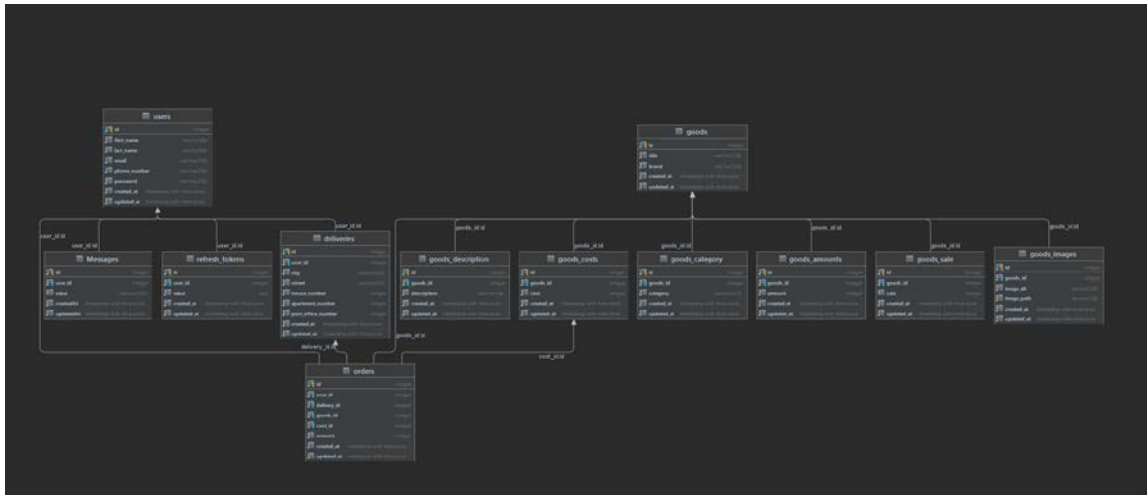


Рисунок В.7 – Слайд № 7

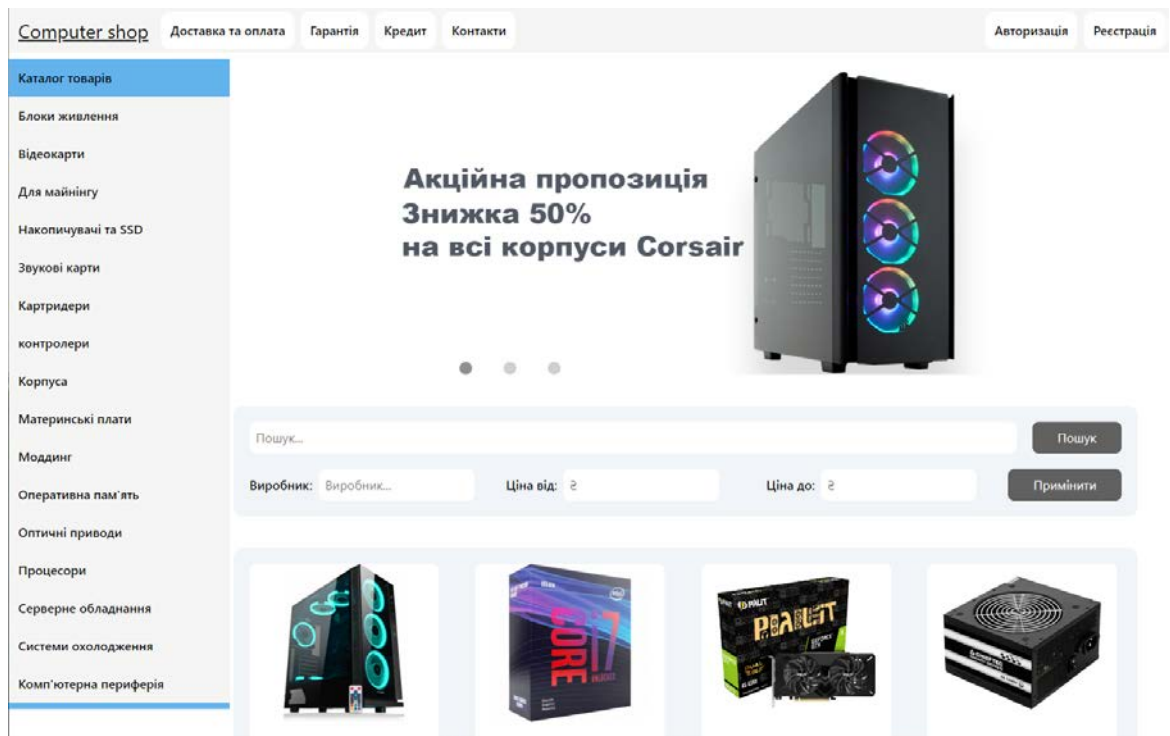


Рисунок В.8 – Слайд № 8

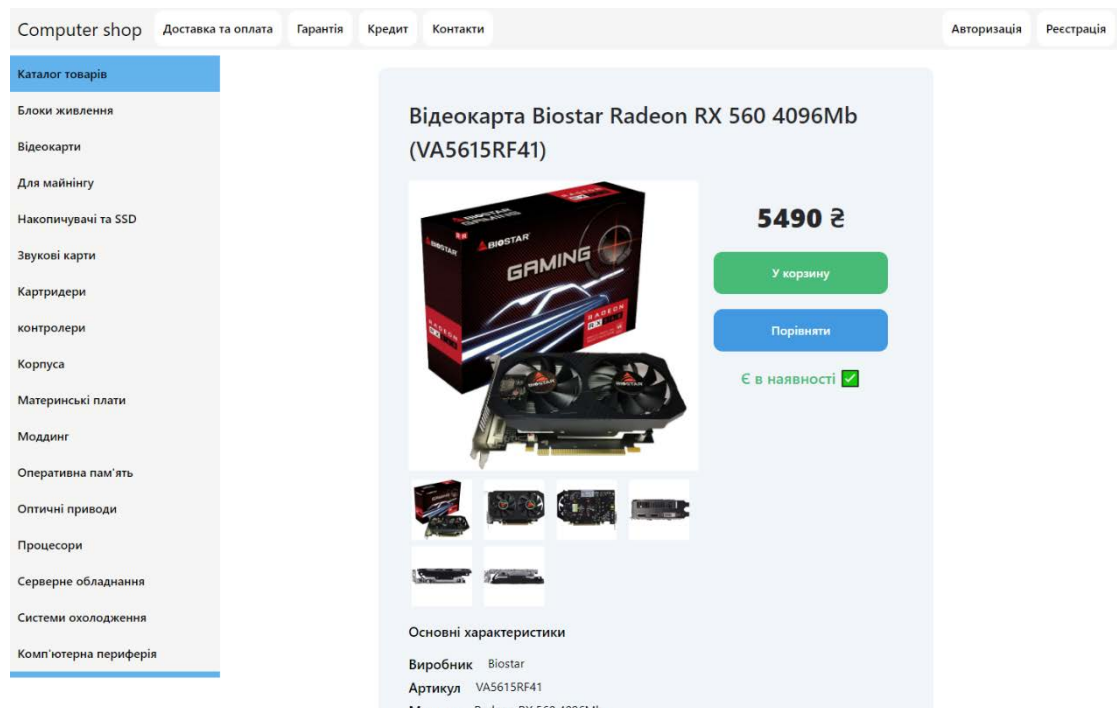


Рисунок В.9 – Слайд № 9

Computer shop Доставка та оплата Гарантія Кредит Контакти Авторизація Реєстрація

Каталог товарів

- Блоки живлення
- Відеокарти
- Для майнінгу
- Накопичувачі та SSD
- Звукові карти
- Картридери
- контролери
- Корпуса
- Материнські плати
- Моддинг
- Оперативна пам'ять
- Оптичні приводи
- Процесори
- Серверне обладнання
- Системи охолодження
- Комп'ютерна периферія

Будь ласка, введіть дані

First name

Last name

Email

Phone number

Password

Re-enter password

Submit

Рисунок В.10 – Слайд № 10

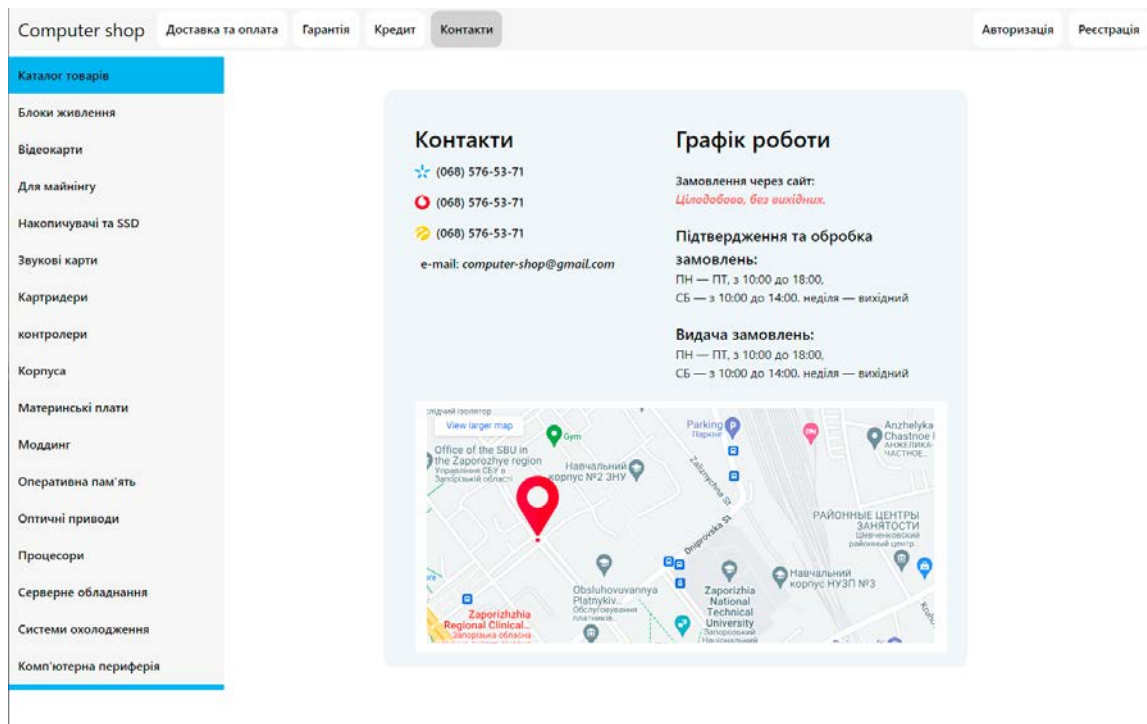


Рисунок В.11 – Слайд № 11

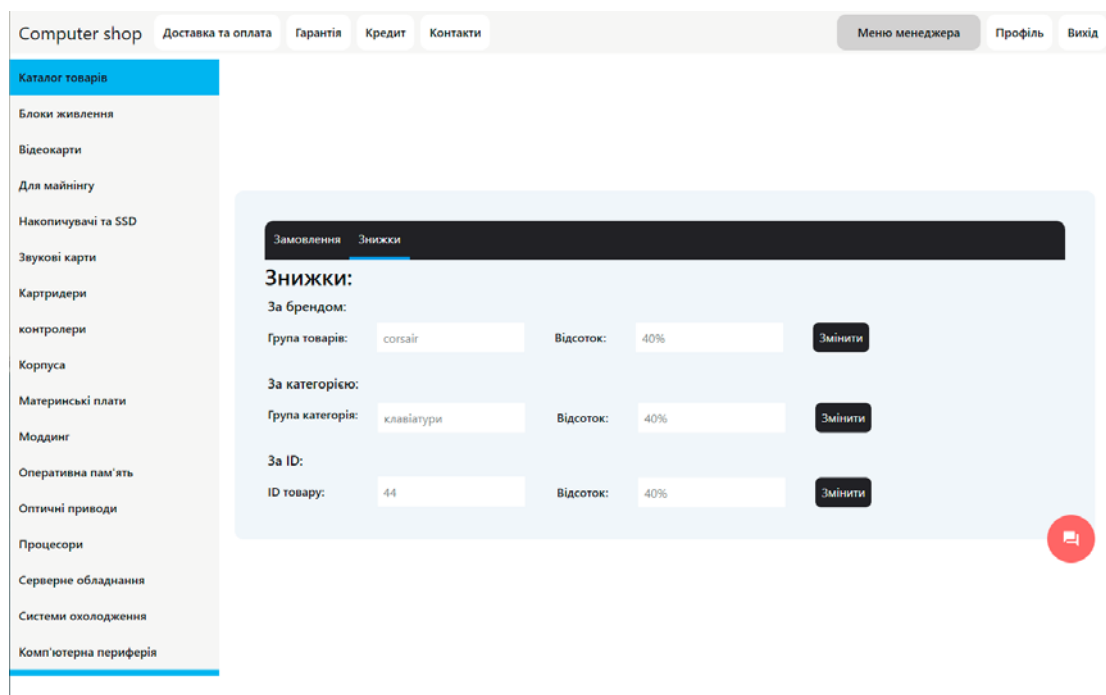


Рисунок В.12 – Слайд № 12

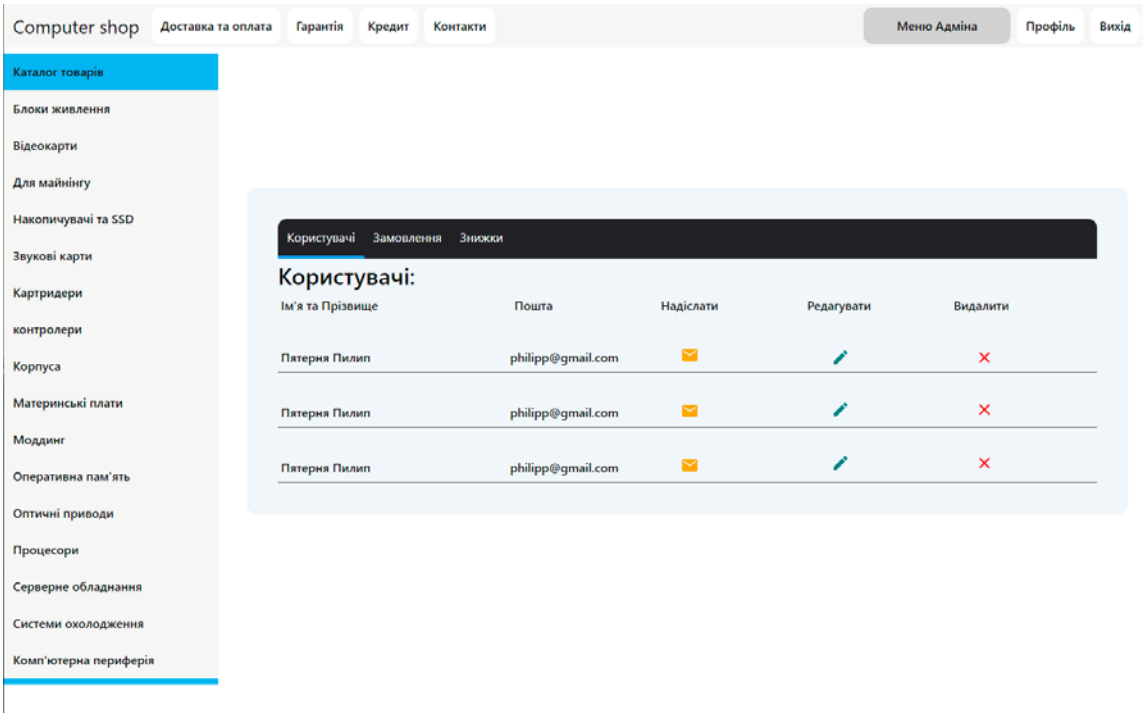


Рисунок В.13 – Слайд № 13

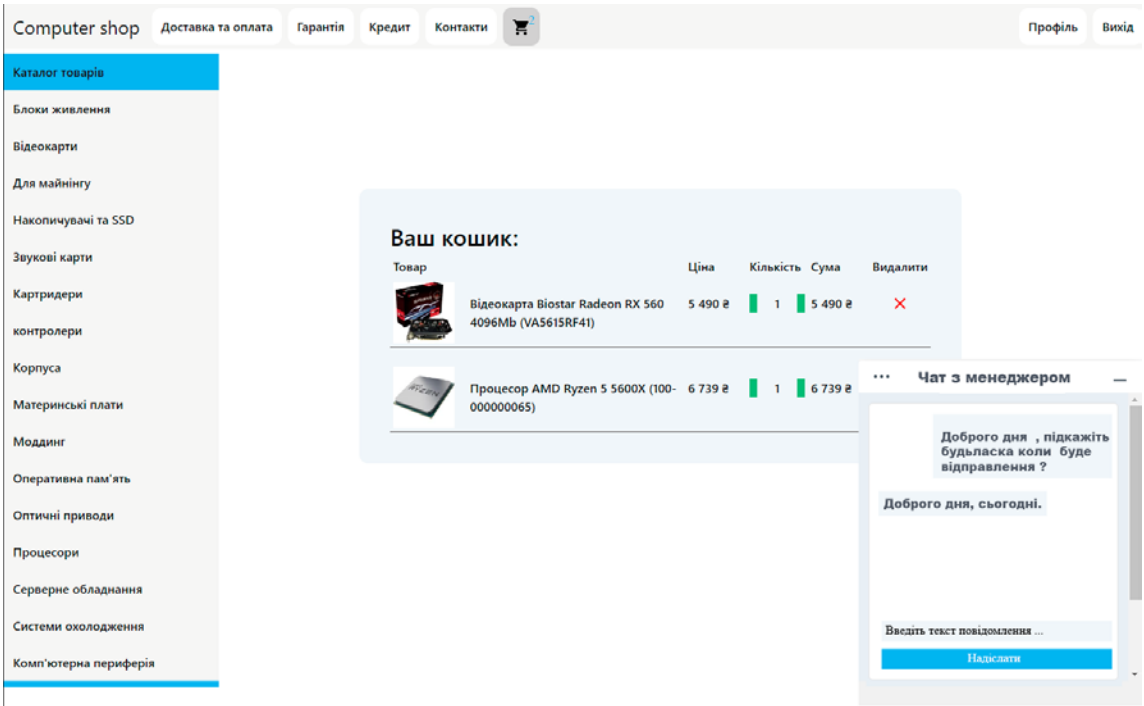


Рисунок В.14 – Слайд № 14

Висновки

Розробку Інтернет магазин комп'ютерних комплектуючих Node.js виконано за допомогою фреймворку Express з визначенням структури даних на основі використання бази даних, вимог до програми, розробленням структури програми і алгоритму роботи програми на основі цих вимог, реалізацією відповідних одиниць вебзастосунку. Для розробки використано мову програмування, JavaScript, мову розмітки HTML, мову стилю сторінок CSS, фреймворки Express, React та Фреймворк для стилізування Tailwind, систему керування базами даних Postgres.

Програма дозволяє переглядати каталог товарів, шукати товари за найменуванням чи категорією, додавати товари у корзину, налаштувати аккаунт, порівнювати товари, користувач має можливість скористатись чатом для консультації з менеджером.

12

Рисунок В.15 – Слайд № 15