МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОЄКТУВАННЯ СИСТЕМ ПРОВЕДЕННЯ ОНЛАЙН-АУКЦІОНІВ

Здобувач освіти гр. ІН.м–12ан/2у Андрій ВАЦЕНКО

Науковий керівник,
доцент, к. т. н. Наталія БАРЧЕНКО

Завідувач кафедри
доцент, к. т. н. Ігор ШЕЛЕХОВ

Суми 2022

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 - Комп'ютерні науки»*

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Ваценку Андрію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи *Інформаційна технологія проєктування систем проведення онлайн-аукціонів*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Огляд технологій, що дозволять реалізувати поставлену задачу; 2) Постановка завдання й формування завданння дослідження; 3) Огляд технологій, що використовуються під час розробки додатків на Nestjs, React; 4) Моделювання та проектування; 5) Програмна реалізація; 6) Аналіз результатів.

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Огляд наявних рішень		
2.	Постановка задачі та формування завдань дослідження.		
3.	Проектування та моделювання		
4.	Програмна реалізація		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 72 стор., 22 рис., 2 таблиці, 1 додаток, 14 літературних джерел.

Об’єкт дослідження — інформаційна технологія проєктування систем проведення онлайн-аукціонів.

Мета роботи — розробка веб додатку, який забезпечить асинхронну комунікацію та консистентність даних.

Результати — було проведено аналіз предметної області, який складався з огляду літератури та аналізу наявних рішень. Досліджено наявні способи комунікації за допомогою асинхронних протоколів, методи побудови веб-додатків та способи взаємодії з базами даних. Проведено аналіз засобів програмної реалізації, які дозволять розробити додатки з допомогою асинхронної комунікації. Після аналізу предметної області та засобів реалізації було проведено проектування та моделювання технології під час, якої була розроблена діаграма варіантів використання, створений прототип веб-інтерфейсу та спроектована структура бази даних. На завершальному етапі було проведено програмну реалізацію. Результатом роботи є платформа аукціону, яка забезпечує основні можливості з продажу товарів на платформі використанням протоколів асинхронної комунікації.

Інформаційна технологія проектування систем проведення онлайн-аукціонів

ЗМІСТ

ВСТУП

На сьогоднішній день платформи з продажу товарів набули широкої популярності завдяки попиту на зростаючому ринку, при зручному інтерфейсі який дозволяє купувати товари. Таким чином відбувається пов’язування бізнес процесів зі сторони бізнесу та попиту зі сторони клієнтів. Одним з найголовніших факторі є зручний інтерфейс та доступність сервісу в будь який час доби, а отже клієнт може скористатися послугами платформи без залежності від годин праці, а також скористатися платформою незалежно від допомоги інших людей, для прикладу навчених операторів. Це дає змогу зекономити кошти та покращити бізнес процеси.

В сучасному світі більшість платформ для онлайн продаж не можуть витримати високих навантажень для прикладу під час свят. Таким чином більшість бізнесів втрачають клієнтів та програють в конкурентній боротьбі. Для вирішення цієї проблеми потрібно використовувати асинхронну комунікацію між сервісами та розділити бекенд за доменними зонами.

Найефективніший способом для вирішення задач бізнесу по продаж та розміщення товарів є онлайн платформи . На щастя, платформа вирішує поставлені цілі. Використання подібної платформи значно підсилить бізнес та дозволить знайти нових клієнтів, а також буде витримувати великі навантаження при великому попиті на товари.

Метою даної роботи є дослідження асинхронної комунікації, яка дозволяє значно підвищити кількість запитів що надходять до сервісу та зробити його стійким до відмов.

Новизна роботи полягає в вирішенні наступних недоліків, що має переважна частина наявних рішень:

- безкоштовні рішення не мають потрібного функціоналу, що не дозволяє великій кількості користувачів відвідувати платформу;
- більшість рішень мають громіздкий та незрозумілий інтерфейс;
- деякі з існуючих рішень працюють в хмарах без можливості розгорнутися на власних серверах;
- більшість рішень це платні додатки, які не вирішують проблеми;
- більшість платних рішень мають занадто високу вартість для невеликих компаній.

На основі виявлених недоліків, які мають переважна частина наявних рішень, в даній роботі буде розроблено систему, що дозволить:

- переглядати товари на платформі;
- додавати товари до платформи за допомогою API;
- переглядати товари аукціону;
- відслідковувати історію покупок;
- робити ставки на товари аукціону;
- розгортати систему на власних серверах;

Для досягнення поставленої мети треба вирішити наступні задачі:

- провести аналіз предметної області;
- обрати засоби та методи для вирішення задачі;
- виконати проектування системи;
- здійснити програмну реалізацію.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Літературний огляд

1.1.1. Огляд технологій мережевої взаємодії

Технологічна взаємодія між серверною та клієнтською частиною веб додатків відбувається за рахунок протоколу HTTP та TCP. TCP це контракт який гарантує доставлення повідомлень через мережу Інтернет. Він надсилає повідомлення забезпечуючи гарантовану та успішну відповідь. TCP один з основних стандартів, який використовують для взаємодії в мережі Інтернету, і входить до стандартів, визначених Інженерною групою з розробки Інтернету (IETF)[1]. Протокол слугує базою для прикладних додатків та інших контрактів обміну(HTTP).

Через непередбачувану поведінку мережі IP-пакети можуть бути втрачені або доставлені не в порядку; TCP виявляє та мінімізує ці проблеми, змінюючи порядок пакетних даних або запитуючи повторну доставку. Ця точність приходить із компромісом у швидкості. TCP більше відомий своєю надійністю, але ця точність пояснюється швидкістю торгівлі, яка іноді надходить із затримкою в кілька секунд.

Протокол використовують, щоб відправки та отримування повідомлень через електронну пошту за допомогою IMAP та (POP). Веб-протокол HTTP використовує для взаємодії також TCP для забезпечення гарантій доставки. [2]

HTTP (протокол передачі гіпертексту) — це набір правил для передачі файлів, таких як текст, зображення, звук, відео та інші мультимедійні файли, через Інтернет. При відкритті веб-браузера, користувач опосередковано використовує HTTP. HTTP — це приклад прикладного протоколу. Останньою

версією HTTP є HTTP/2, яка була опублікована в травні 2015 року. Це альтернатива своєму попереднику HTTP 1.1, але не робить її застарілою.

За допомогою протоколу HTTP відбувається обмін ресурсами між клієнтськими пристроями та серверами через Інтернет. Клієнтські пристрої надсилають серверам запити на ресурси, необхідні для завантаження веб-сторінки; сервери надсилають відповідь клієнту для виконання запитів. Запити та відповіді спільно використовують піддокументи, такі як дані про зображення, текст, макети тексту тощо, які об’єднуються клієнтським веб-браузером для відображення повного файлу веб-сторінки.

На додаток до файлів веб-сторінок, які він може обслуговувати, веб-сервер містить HTTP-демон, програму, яка очікує HTTP-запитів і обробляє їх, коли вони надходять. Веб-браузер — це HTTP-клієнт, який надсилає запити на сервери. Коли користувач браузера вводить запити на файли, «відкриваючи» веб-файл, вводячи URL-адресу або натискаючи гіпертекстове посилання, браузер створює HTTP-запит і надсилає його на адресу Інтернет-протоколу (IP-адресу), указану в URL-адресі. HTTP-демон на сервері призначення отримує запит і надсилає запитуваний файл або файли, пов’язані із запитом.[3]

Асинхронний обмін повідомленнями та зв’язок, керований подіями, є критично важливими під час поширення змін між кількома мікросервісами та їх пов’язаними моделями домену. Як згадувалося раніше в обговоренні мікросервісів і обмежених контекстів (BC), моделі (користувач, клієнт, продукт, обліковий запис тощо) можуть означати різні речі для різних мікросервісів або BC. Це означає, що коли відбуваються зміни, вам потрібен певний спосіб узгодити зміни в різних моделях. Рішенням є остаточна узгодженість і зв’язок, керований подіями, на основі асинхронного обміну повідомленнями.

Під час використання обміну повідомленнями процеси спілкуються шляхом асинхронного обміну повідомленнями. Клієнт робить команду або

запит до служби, надсилаючи їй повідомлення. Якщо служба потребує відповіді, вона надсилає клієнту інше повідомлення. Оскільки це спілкування на основі повідомлень, клієнт припускає, що відповідь не буде отримано негайно, і що відповіді може не бути взагалі.

Повідомлення складається із заголовка (метаданих, таких як ідентифікаційна інформація або інформація про безпеку) і тіла. Повідомлення зазвичай надсилаються через асинхронні протоколи, такі як AMQP.

Бажаною інфраструктурою для цього типу зв’язку в спільноті мікросервісів є легкий брокер повідомлень, який відрізняється від великих брокерів і оркестровців, що використовуються в SOA. У легкому посереднику повідомлень інфраструктура зазвичай «тупа», діючи лише як посередник повідомлень, із простими реалізаціями, такими як RabbitMQ або масштабованою службовою шиною в хмарі, як-от Azure Service Bus. У цьому сценарії більшість «розумного» мислення все ще живе в кінцевих точках, які створюють і споживають повідомлення, тобто в мікросервісах.

Інше правило, якого ви повинні намагатися дотримуватися, наскільки це можливо, полягає в тому, щоб використовувати лише асинхронний обмін повідомленнями між внутрішніми службами та використовувати синхронний зв’язок (наприклад, HTTP) лише від клієнтських програм до зовнішніх служб (шлюзи API плюс перший рівень мікросервісів).

Advanced Message Queuing Protocol (AMQP) — це відкритий стандартний протокол прикладного рівня для проміжного програмного забезпечення, орієнтованого на повідомлення. Визначальними особливостями AMQP є орієнтація повідомлень, черги, маршрутизація (включаючи точка-точка та публікація та підписка), надійність і безпека.[1]

AMQP визначає поведінку постачальника повідомлень і клієнта в тій мірі, в якій реалізації від різних постачальників є сумісними, так само, як SMTP, HTTP, FTP тощо створили сумісні системи. Попередня стандартизація

проміжного ПЗ відбувалася на рівні API (наприклад, JMS) і була зосереджена на стандартизації взаємодії програміста з різними реалізаціями проміжного ПЗ, а не на забезпеченні взаємодії між кількома реалізаціями.[4] На відміну від JMS, який визначає API і набір поведінки, який має забезпечити реалізація обміну повідомленнями, AMQP є протоколом дротового рівня. Протокол дротового рівня – це опис формату даних, які надсилаються через мережу як потік байтів. Отже, будь-який інструмент, який може створювати та інтерпретувати повідомлення, що відповідають цьому формату даних, може взаємодіяти з будь-яким іншим сумісним інструментом, незалежно від мови реалізації.

Одним з сервісів в якому реалізовано протокол для обміну повідомленнями є RabbitMQ.

Для надсилання повідомлення, модель сервісу відправки відправляє повідомлення на обмін, який схожий на поштове відділення, яке приймає та розподіляє всі повідомлення в потрібному місці. Обмін пов’язаний із багаточисельними чергами через з’єднання, які називаються «прив’язками», які є зв’язуючою ланкою між обміном і чергою. На прив’язки посилаються ключі прив’язки. Кожна черга пов'язана з послугою-споживачем або коротко споживачем. Споживачі записуються в черги.

У сервіс може масштабуватися та має безліч переваг. Однією з великих переваг моделі повідомлень є гнучкість. Ця гнучкість значною мірою пов’язана з різними типами обміну.

Обмін Fanout: коли Checkout надсилає повідомлення на обмін, він дублює це повідомлення та надсилає його до кожної окремої черги, про яку він знає.

Прямий обмін. Під час прямого обміну система оформлення видає повідомлення з ключем маршрутизації. Ключ маршрутизації порівнюється з

ключем прив’язки, і якщо він точно збігається, він відповідно переміститься в систему.

Обмін темами: обмін темами відповідає обом ключам на основі шаблону під назвою «тема». На відміну від прямого обміну, він технічно не може мати довільний ключ маршрутизації — це має бути список слів, розділених крапками.[5]

Під час обміну заголовками ключ маршрутизації повністю ігнорується. Натомість повідомлення переміщується до всієї системи за допомогою заголовка.

Обмін за замовчуванням: обмін за замовчуванням унікальний для RabbitMQ, оскільки він не є частиною моделі AMQP. Кожна створена черга автоматично прив’язується до неї за допомогою ключа маршрутизації, який збігається з назвою черги.[6]

Значним досягенням цієї технології є надзвичайна гнучкість: усі налаштування переміщення повідомлення визначаються, коли адміністратор обміну повідомленнями налаштовує модель повідомлення. У RabbitMQ через те що він переміщується системою, є частина метаданих повідомлень. Значний контроль має програма та розробник, а не адміністратор брокера повідомлень.

Потрібно відзначити зручність обміну у хмарі: сервіс дозволяє розгорнути екземпляр RabbitMQ на Docker або будь-якому програмному забезпеченні контейнеризації.

Сервіс відомий хорошою безпекою: він підтримує FASL, LDAP і TLS для аутентифікації та авторизації.

Підтвердження повідомлень: якщо повідомлення знаходиться в черзі, воно залишається, доки споживач не повідомить брокеру, що він видаляє повідомлення. Це запобігає втраті повідомлень.

Багато плагінів: спільнота RabbitMQ з відкритим кодом створила багато плагінів, які збагачують більшість аспектів RabbitMQ. Він настільки вдосконалений, що тепер RabbitMQ підтримує інші моделі повідомлень, такі як AMQP 1.0, MQTT, STOMP. [7]

1.1.2. Методи побудови веб-додатків

Достатньо популярні підходи до створення можна розділити на два способи – односторінковий (SPA) додаток чи частину монолітного аплікейшену.

Односторінковий додаток — це веб-додаток, що складається з однієї HTML-сторінки, та зміни відбуваються після за допомогою javascript. На відміну від традиціних сторінок, SPA не завантажує нових веб-сторінку— а тільки змінює існуючу сторінку. Таким чином можна створювати односторінкові додатки, які запускаються у браузері, але мають інтерфейс користувача та функціональність, подібну до програм для настільних комп’ютерів.

В основі своїй, SPA засновані з використанням Ajax технології — набору технологічних методів веб-розробки для обробки запитів сервера у в асинхронному режимі без необхідності перезавантажувати сторінку[8]. Ajax надав можливість до складання складні функції в програми, які відображуються на одній сторінці. Різниця в між SPA та традиційним додатком зображена на рисунку 1.1.

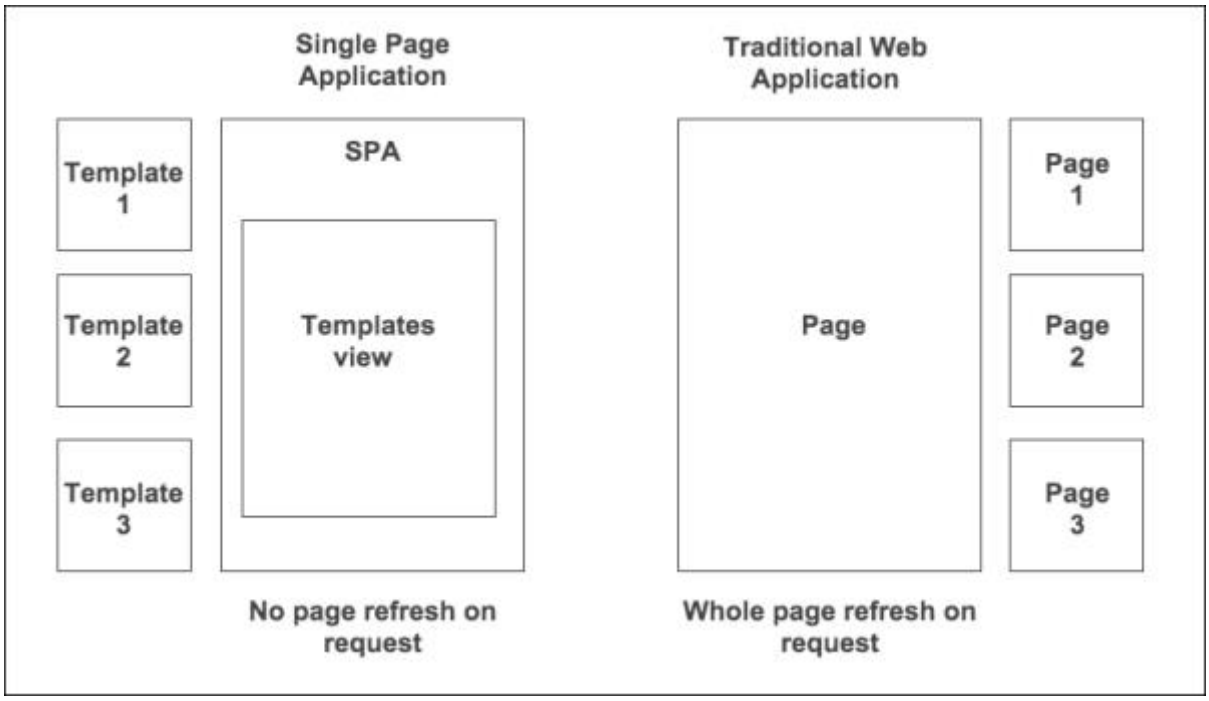

Рисунок 1.1 – Відмінність традиційних та односторінкових веб-додатків

Односторінковий додаток має на меті створення динамічних платформ з великими обсягами даних. Більш того, ця технологія підходить для створення мобільних додатків у майбутньому. Це гарна архітектура для SaaS платформ, інтернет магазинів та клубів за інтересами, там де пошукова оптимізація не має вирішальну роль. Але з розвитком технології односторінкових додатків, проблема SEO оптимізації може бути вирішена.

SPA програми мають широкий спектр використання і можуть створювати як прості сайти так і великі та комплексні платформи. Можливо виконати будь-яке завдання за допомогою SPA, яке може виконувати традиційний багатосторінковий додаток.[9]

До появи односторінкових додатків додатки створювалися та відтворювалися на стороні сервера та доставлялися – повністю сформовані – на клієнтський пристрій: коли користувач переходив у додатку, кожна сторінка створювалася з нуля, тому її потрібно було постійно переглядати. оновлюється, що призводить до мерехтіння екрана. Веб-програми, що

інтенсивно використовують дані, які отримували дані з кількох різних джерел, також можуть завантажуватися повільно.

У односторінкових програмах логіка презентації відображається на стороні клієнта, а візуальна структура веб-програми залишається незмінною протягом сеансу, а нещодавно запитувані дані оновлюються у фоновому режимі. Це забезпечує набагато плавніший досвід для користувача.

SPA мають ряд переваг над традиційними додатками, а саме:

1. Доступність. Можливість завантажити додаток на будь-який девайс який підтримує браузер та не займати багато місця на смартфоні.
2. Універсальність. У випадку якщо додаток розроблявся з урахування екрану, то використовувати SPA зручно як зі стаціонарних комп’ютерів, і зі смартфона.
3. Швидкість. Сторінка з необхідними скриптами не зменшує час на переходи для запиту даних, а й збільшує продуктивність роботи та завантаження.
4. Легкість для розробників. Розробники можуть використовувати наявні рішення в архітектурі та готові платформні рішення які доступні з коробки. Легко тестувати та налагоджувати односторінкові програми за допомогою інструментів розробника, наданих Google у веб-переглядачі Chrome.

Традиційний веб додаток – це веб-додаток, який складається з кількох веб-сторінок, які завантажуються, коли користувач відвідує різні частини сторінки. Це традиційний шлях до розробки веб-додатків, який використовується для веб-сайтів, які мають справу з підвищеною кількістю даних. Запити які надсилаються до серверу, як приклад, введення нової URL-адреси або перехід за посилання, призводить до створення нової сторінки та повернення завантаження[10].

Ідеальними прикладами SPA додатків є такі всесвітньо відомі команії, як Amazon або eBay. Під час роботи з ними відбувається отримання нового

файлу для кожного запиту. Зазвичай, багатосторінкові додатки складні з кількома рівнями, посиланнями та різними інтерфейсами. Контент таких веб-сайтів розділяється на кілька розділів.

1.2. Аналіз наявних рішень

Ebid (https://www.ebid.net/us/) – це веб рішення за допомогою користувачі можуть придбати товар та продати товар. За допомогою цього рішення користувачі можуть зробити свій bid на платформі та налаштувати ціну продажу товару. Товар може бути доступний користувачам для продажу через декілька хвилин.

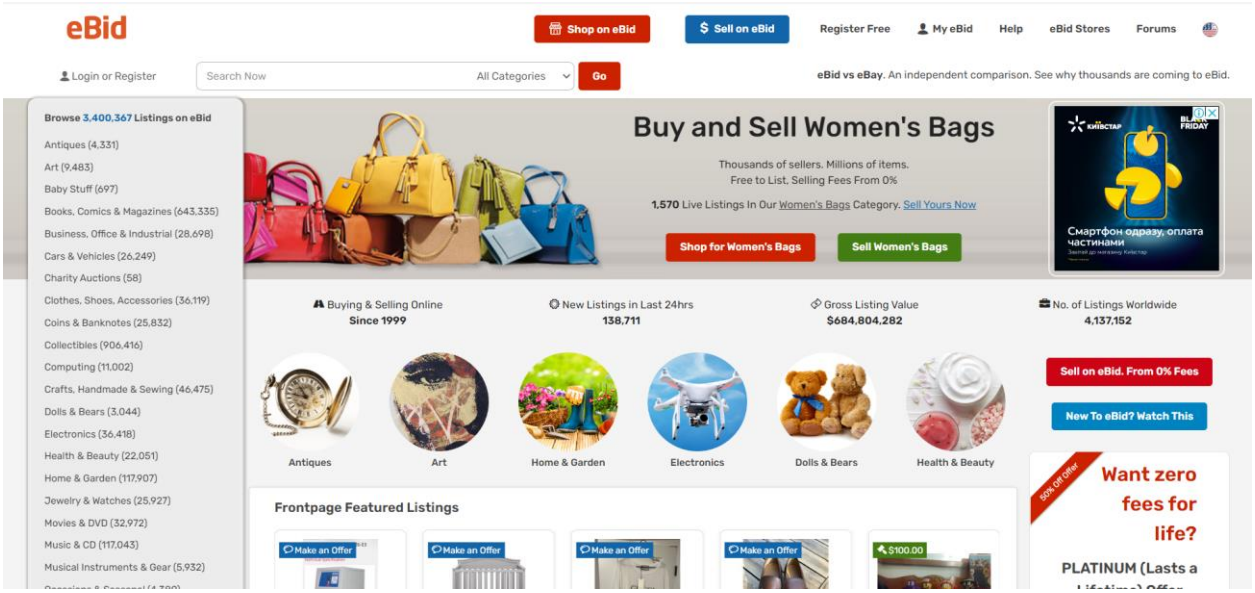

Рисунок 1.2 – Ebid інтерфейс

Аналіз недоліків

Система має громіздкий інтерфейс з великою кількістю параметрів для того, щоб розмістити товар, що робить її більш складною у використані. При використанні сервісу є досить багато рекламних оголошень, що ускладнює його використання.

Ebay (https://www.ebay.com/) – одна з платформ лідерів в онлайн комерції. Забезпечує можливість створювати та продавати товари для інших

користувачів сервісу. Гарний сервіс з огляду на кількість трафіку який до нього приходить та широкий функціонал який він пропонує.

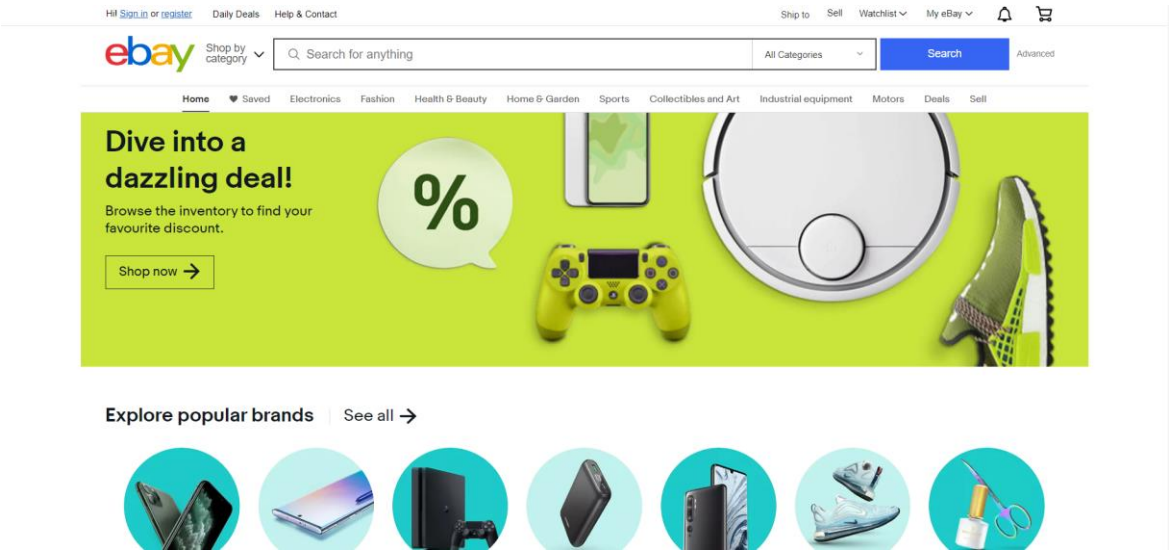

Рисунок 1.3 – Ebay інтерфейс

Аналіз недоліків

Сервіс має складний функціонал під час розміщення товару. Це створює складності в роботі для користувачів які розміщують товар. Сервіс іноді може бути недоступний через проблеми з доступністю з різних країн та може не працювати. Розміщені категорії на сайті обмежені, не можливо додати іншу категорію.

2. ПОСТАНОВКА ЗАДАЧІ ТА ВИБІР МЕТОДІВ РЕЛІЗАЦІЇ

2.1. Постановка задачі та визначення мети

Метою роботи є створення інформаційної технології, яка дозволить вирішити проблему з високим навантаженням та дозволить користувачам робити замовлення на платформі.

Платформа повинна давати можливість:

- Придбати товар;
- Зробити ставку (bid) на товар;
- переглянути придбані товари;
- додавати товари для продажу на платформі в форматі аукціону;
- використовувати Docker як технологію віртуалізації;
- розгортати на власних серверах;

Для досягнення поставленої мети треба вирішити наступні задачі:

- провести аналіз предметної області;
- обрати засоби та методи для вирішення задачі;
- виконати проектування системи;
- здійснити програмну реалізацію.

2.2. Вибір методів реалізації

Для програмування веб-інтерфейсів використовуються достатньо багато мов програмування серед яких: Flutter, Microsoft Silverlight, Flash, Vue, Angular, React та інші. Microsoft Silverlight, Flash має багато проблем пов’язаних з безпекою та не підтримуються компаніями що їх винайшли.

Flutter це молодий фреймворк від компанії Google, має досить коротку історію та не зарекомендував себе в сфері створення веб-інтерфейсів. Має обмежену кількість бібліотек, які можуть не підтримувати створення веб сторінок. Фреймворк не має достатньої кількості проектів написаних під веб та не може розглядатися як технологія створення клієнтської частини сервісу, адже на ньому не створено достатньо готових рішень для використання в реальному проекті.

Angular не було обрано так як він ставить програміста в залежність від свої бібліотек та не дозволяє використовувати інші, більш того розвиток проекту і підтримка на цій технології вимагає багато часу та не має докладної документації. Фреймворк достатньо сильно змінюється в залежності від версії, що може стати проблемою при переході на нову версію та оновленні залежностей.

Vue має обмежену документацію на англійській мові та мало готових прикладів для використання та ознайомлення. Потрібно відзначити що кількість ресурсів написана на vue недостатня - це робить складнішим обмін знань в рамках розробки на данному фреймворку.

Беручи до уваги вище сказане, вибір був зроблений на користь React. В якості мови програмування було обрано веб-технології, які працюють з застосуванням html, css, js. Перевагами цих технологій є їх відкритість та доступність, завдяки чому більшість проектів в вебі використовують саме ці технології. Для розробки клієнтської частини була використана мова програмування javascript та фреймворк React.

React – це бібліотека, яка розроблена на базі JavaScript з відкритим кодом для створення користувацьких інтерфейсів. React був створений компанією Facebook. Вона використовує концепцію створення компонентів, що дозволяє перевикористовувати код у різних проектах. Для створення інтерфейсу React використовую JSX, що дозволяє писати HTML подібний код разом з компонентами. Також, ця бібліотека має високу продуктивність завдяки концепції віртуального DOM. Віртуальний DOM зберігає копію звичайного DOM. При необхідності оновити елемент інтерфейсу спочатку порівнюється та замінюється віртуальний DOM, і при виявленні розбіжностей оновлює основний DOM з мінімальною кількістю затрат. Отже, такий спосіб взаємодії з елементами набагато ефективніший, ніж напряму змінювати DOM через JavaScript. React використовує однонаправлену передачу даних, тобто з батьківських компонентів до компонентів нащадків. Дані, які були передані не можуть бути змінені напряму компонентом нащадком, але при необхідності можна застосувати callback-функції.[11]

Процес отримання даних побудований за архітектурою "клієнт – сервер", де в ролі сервера виступає бекенд написаний на мові програмування node та з застосуванням фреймворку Nestjs. Бекенд був створений в вигляді API, що дозволяє зменшити навантаження на нього.

Nestjs - це Node фреймворк з відкритим кодом та модульною сруктурою, створений Камілом Мислівецем для розробки додатків що наслідують модульності.

Він був створений з використанням ідей Angular, внаслідок чого отримав модульність та імплементував ідею реактивності. В якості ядра Nestjs використовує компоненти іншої платформи – Express або Fastify (робота з запитами, обробка запитів, отримання данних) [12].

За допомогою менеджера пакетів Npm, фреймворк Nestjs дозволяє розширювати вбудований функціонал і підключати інші компоненти (готові рішення) для веб-додатку.

Для роботи з базою даних та полегшення написання запитів для задачі була обрана Prisma ORM. Ця модель управління представляє собою іноваційну систему для роботи з базами даних. ORM складається наступних частин:

- Клієнт Prisma - автоматично згенерований і безпечний конструктор запитів для Node.js і TypeScript
- Prisma Migrate - система міграції
- Prisma Studio - графічний інтерфейс для перегляду та редагування даних у вашій базі даних

Клієнт Prisma можна використовувати в будь-якому серверному додатку Node.js (підтримувані версії) або TypeScript (включаючи безсерверні додатки та мікросервіси). Це може бути REST API, GraphQL API, gRPC API або щось інше, для чого потрібна база даних.

Кожен проект, який використовує інструмент із набору інструментів Prisma, починається з файлу схеми Prisma. Схема Prisma дозволяє розробникам визначати свої моделі додатків на інтуїтивно зрозумілій мові моделювання даних. Він також містить підключення до бази даних і визначає генератор.[13]

Для зменшення помилок кількості помилок, отже якості коду та збільшення швидкості розробки за допомогою мови typescript. Typescript — це безкоштовна мова програмування з відкритим вихідним кодом, розроблена та підтримується Microsoft. Це суворий синтаксичний набір правил JavaScript який додає необов’язкову статичну типізацію до мови. Він створений для розробки великих додатків і транспілюється на JavaScript. Оскільки це надмножина JavaScript, існуючі програми JavaScript також є такими що працюють з програмами TypeScript.[14]

Для розгортання системи та деплою був використаний Docker . Docker — є відкритою платформою для розробки, доставки та запуску програм. Docker робить можливим відокремлення програми від інфраструктури, для швидкої доставки програмного забезпечення. З використанням Docker

можливо керувати інфраструктурою та програмами. Використавши Docker для швидкої доставки, тестування та розгортання коду, значно зменшується затримка між написанням коду та його запуском у виробництві.

Docker надає можливість пакувати та запускати програму в слабко ізольованому середовищі, яке називається контейнером. Ізоляція та безпека надають модливість розробникам запускати багато контейнерів одночасно на певному хості. Легкі контейнери містять необхідний функціонал для запуску програми, тому вам не потрібно покладатися на те, що зараз встановлено на хості. Docker контейнером можливо поділитися під час роботи, і на 100% иожна розрахвувати на те що ті з ким ви ділитеся, отримають той самий контейнер, який працює однаково.

3. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1. Розробка прототипу інтерфейсу користувача

Прототип – визначає основні функції інтерфейсу користувача та оцінює дизайн. Прототип спонукає продумати ідеї без фактичної програмної реалізації. Саме тому можна значно зменшити час розробки системи, зменшити кількість часу на обговорення та зробити її комфортною для користувача.

Інтерфейс системи повинен забезпечувати взаємодію між серверною частиною та клієнтським додатком. Користувач повинен мати змогу переглянути всі товари доступні для продажу. На рисунку 3.1 зображений прототип головної сторінки, яка дозволяє користувачу переглянути список доступних на платформі.

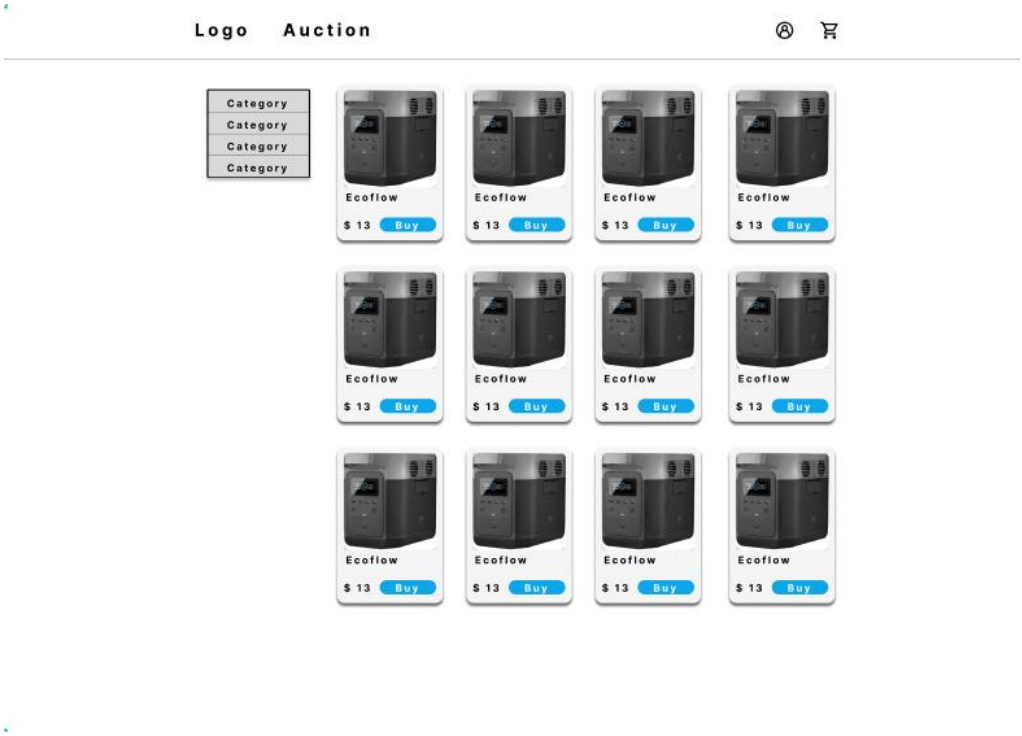

Рисунок 3.1 – Прототип головної сторінки платформи

Користувач повинен переглянути товар та прочитати опис товару. Для доступу до більш детальної інформації користувач повинен натиснути на карточку товару. Даний підхід досить розповсюджений та інтуїтивно зрозумілий користувачам.

На карточні товару користувач має зручний інтерфейс який відповідає основним вимогам сучасних користувачів. Зліва відображується меню переходу по категоріям товарі інтернет магазину. Також на сторінці доступне фото самого товару, його назва та ціна. Короткий опис під товаром дає змогу клієнту ознайомитися з наявним товаром та замовити його використовуючи кнопку покупки.

Рисунок 3.2 – Прототип сторінки звичайного товару

Після натискання кнопки купити, потрібно розробити екран корзини. В ній будуть зберігатися товари які користувач має намір придбати. Вгорі завжди повинен бути відображений бар заголовка за допомогою якого користувач має змогу перейти до найважливіших сторінок платформи. В першому рядку буде назва сторінки та кнопка переглянути попередні товари,

які були придбані раніше користувачем. Нище буде відображена картка товарів з кількістю та ціною товару при збільшенні кількості відбувається зміна в ціні в картці товару, також зміна відбувається в нижньому меню з сумою по всім замовленням.

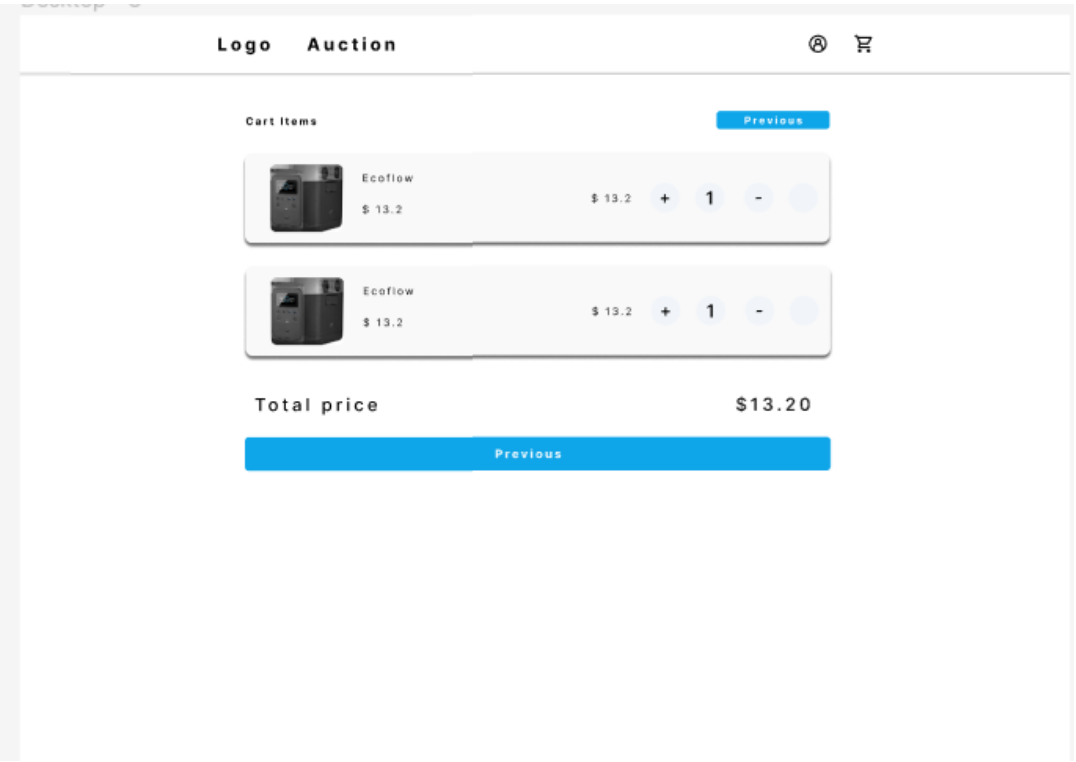

Рисунок 3.3 – Прототип корзини товарів

Товари які доступні для покупки, як товари аукціону повинні мати обмежений час, коли вони будуть доступні для продажу на платформі. Ці данні вказуються користувачами при створенні товару. В цій частині інтерфейс схожий на звичайну карточку товару. На сторінці відображається опис та заголовок товару.

Ключовою відмінністю є відображення таймер відліку який показує час коли ще товар буде доступний до покупки. Поле вводу нової ставки на товар буде відображено рядом з кнопкою створення.

Рисунок 3.4 – Прототип товару аукціону

3.2. Розробка ER-діаграми

Для того, щоб спроектувати базу даних було використано ER-діаграму. Це дозволить визначити предметну область, виокремити головні сутності та встановити зв’язки між ними. На рисунку побудована ER-діаграма, що відповідає потребам системи. Для уникнення потенційних суперечностей між даними модель було приведено до третьої нормальної форми. Було створено діаграму для сервісу користувачів.

Сервіс користувачів відповідає за авторизацію та обробку користувацьких даних. База даних сервісу повинна зберігати дані про коди верифікації та забезпечувати скидання паролю.

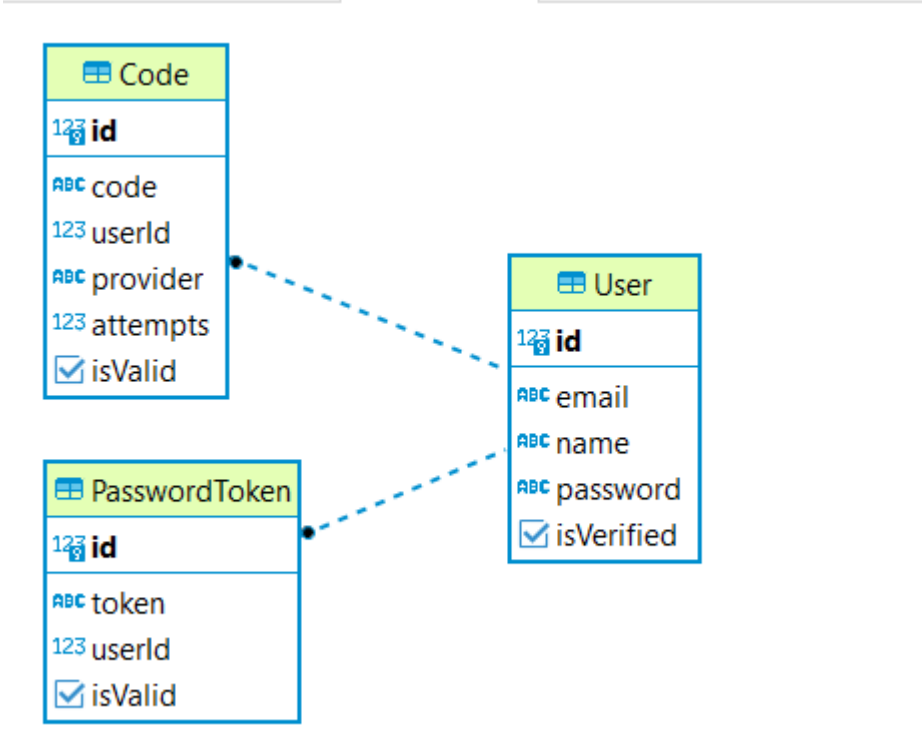

Рисунок 3.5 - ERD діаграма сервісу користувачів

Таблиця 3.1. – Опис сутностей та їх атрибутів.

Таблиця	Поле	Ключі	Зміст
User	id	PK	Унікальний ідентифікатор сутності.
	email		Email користувача.
	name		Ім’я користувача.
	password		Пароль користувача.
	isVerified		Статус верифікації користувача
PasswordToken	id	PK	Унікальний ідентифікатор сутності.
	token		Токен паролю
	userId		Посилання на id користувача
	isValid		Статус валідності токена
Code	id	PK	Унікальний ідентифікатор сутності.
	code		Код поля.
	userId		Посилання на id користувача.
	provider		Призначення токена.
	attemts		Кількість спроб використання.
	isValid		Поле валідністі токена.

Так як проект буде використовувати мікросервісну архітектуру є сенс створити новий сервіс та окрему базу данних. Такий патерн дозволить масштабуватися сервісам оркемо і незалежно та зберігати дані з меншою звязаністю один від одного.

Для сервісу замовлень було обрано базу даних mongodb, так як вона є nosql рішенням та дозвовляє масштабувати більшу кількість даних та отримувати результат швидше в тому числі економлячи час розробників. На рисунку 3.2 спроектована база даних зі зв’язками між моделями.

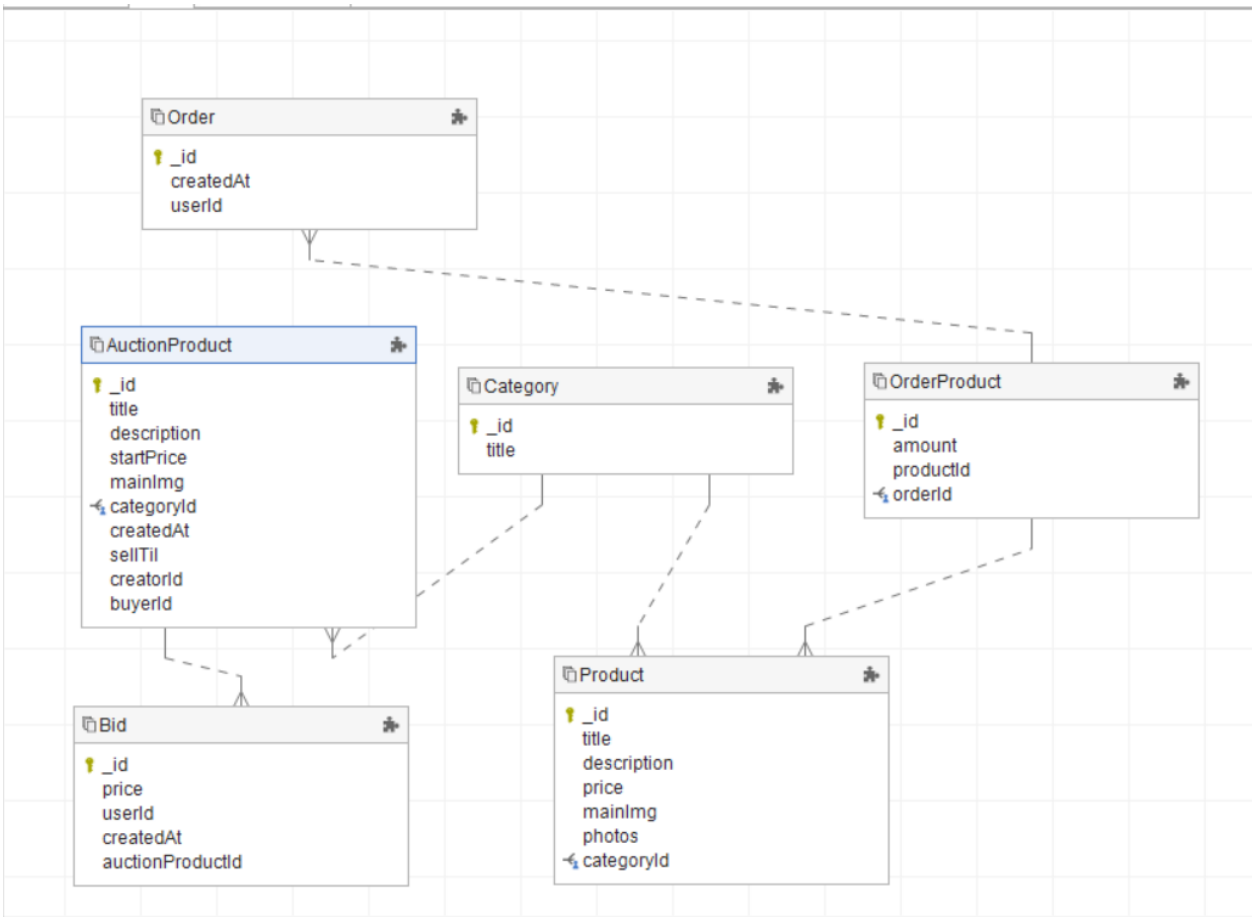

Рисунок 3.6 - ERD діаграма сервісу замовлень

Таблиця 3.1. – Опис сутностей та їх атрибутів.

Таблиця	Поле	Ключі	Зміст
Bid	_id	PK	Унікальний ідентифікатор сутності.
	price		Ціна товару.
	userId		Id користувача.
	createdAt		Дата та час створення.
	auctionProductId		Id продукту аукціону

Таблиця	Поле	Ключі	Зміст
AuctionProduct	_id	PK	Унікальний ідентифікатор сутності.
	title		Назва товару
	description		Опис товару
	startPrice		Початкова ціна
	mainImg		Основне зображення продукту
	categoryId		Посилання на категорію
	createdAt		Дата та час створення.
	sellTil		Дата до якої товар доступний для продажу
	creatorId		Id користувача який продав товар
	buyerId		Id користувача що переміг в аукціоні
Product	_id	PK	Унікальний ідентифікатор сутності.
	title		Назва продукту
	description		Опис продукту
	price		Ціна продукту.
	mainImg		Основне зображення продукту
	photos		Фото продуктів.
	categoryId		Посилання на категорію
Category	_id	PK	Унікальний ідентифікатор сутності.
	title		Назва категорії

Таблиця	Поле	Ключі	Зміст
OrderProduct	_id	PK	Унікальний ідентифікатор сутності.
	Amount		Кількість продуктів в замовленні
	productId		Посилання на продукт
	orderId		Посилання на замовлення
Order	_id	PK	Унікальний ідентифікатор сутності.
	createdAt		Дата та час створення.
	userId		Посилання на користувача

4. ПРАКТИЧНА РЕАЛІЗАЦІЯ

4.1. Програмна реалізація серверної частини

В розробці сервісу був використаний мікросервісний підхід до створення серверної частини. Перед безпосередньою розробкою сервісу було виділено основні компоненти системи. Відбулося розбиття на гетвей сервіс, сервіс замовлень, сервіс користувачів та сервіс повідомлень.

У розробці рішення був використаний docker для розгортання контейнерів та комунікації між ними. Для оркестрації контейнерів використаний docker-compose, який дозволив зручне налаштування та комунікацію між контейнерами.

Запити від клієнтської частини додатку будуть направлені до гетвей сервісу який вирішить до якого сервісу їх відправити надалі. Для брокера повідомлень використовується RabbitMQ. Сервіс реалізую протокол AMQP дає змогу сервісам передавати повідомлення використовуючи згаданий протокол. Таким чином забезпечується асинхронна взаємодія між сервісами.

Всі запити що надходять до сервісу оброблюються контролерами гетвей сервісу, який реалізую REST підхід до проектування API. Технологія включає в себе наступні методи:

- GET
- POST
- PATCH
- PUT
- DELETE

Запити які повертає сервіс реалізують формат json. Наступні шляхи доступні в сервісі та були створені для роботи з клієнтською частиною.

Таблиця 4.1 – Опис розробленого REST-інтерфейсу.

Тип	API інтерфейс	Призначення
POST	/auth/login	Дозволяє отримувати токен авторизації за email та паролем
POST	/auth/register	Дозволяє реєструвати користувача за email та паролем
POST	/auth/confirm	Дозволяє підтвердити аккаунт користувача
POST	/password/verification	Посилає код на пошту для скидання паролю.
POST	/password/token	Віддає токен для зміни паролю
POST	/password/reset	Дозволяє робити новий пароль за токеном
GET	/auction/products/:id/bids	Отримати всі ставки на продукт
POST	/auction/products/:id/bids	Зробити ставку на продукт
GET	/auction/products	Отримати продукти які продаються як товари аукціону

Тип	API інтерфейс	Призначення
GET	/auction/products/:id	Отримати інформацію про один продукт
POST	/auction/products	Додати новий продукт аукціону
GET	/categories	Отримати всі існуючі категорії
GET	/:category/products	Отримати продукти з даної категорії
GET	/orders	Отримати замовлення для користувача
POST	/orders	Зробити замовлення
DELETE	/orders/:id	Видалити замовлення за id
GET	/products	Отримати всі продукти
POST	/products	Створити продукт
GET	/products/:id	Отримати продукт за id
PATCH	/products/:id	Оновити продукт за id
DELETE	/products/:id	Видалити продукт.

Гетвей сервіс передає управління запитом до сервісу користувачів або сервісу замовлень в залежності від запиту. Проуес відбувається завдяки відпаравці подій та виклику віддаленої процедури. Таким чином сервіси працюють незалежно один від одного та мають гарно масштабуватися.

В свою чергу сервіс відправки нотифікацій реалізує підписку на event від сервісу користувачів.

```
@Controller()
export class EmailMessageController {
    constructor(private readonly emailMessageService: EmailMessageService) {}

    @EventPattern(RabbitMqQueues.AccountVerification)
    public async sendAccountVerification(@Payload() body: IEmailMessage, @Ctx() context: RmqContext): Promise<void> {
        await this.emailMessageService.sendAccountVerification(body, context);
    }

    @EventPattern(RabbitMqQueues.ResetPassword)
    public async sendResetPassword(@Payload() body: IEmailMessage, @Ctx() context: RmqContext): Promise<void> {
        await this.emailMessageService.sentResetPassword(body, context);
    }
}
```

Рисунок 4.1 – доступні методи в контролері сервісу повідомлень

Сервіс повідомлень реагує тільки на повідомлення відправлені з патерном який указаний в контролері. Це є аналогом з REST path. Для відправки повідомлень був використаний mailjet сервіс який дозволяє розсилку повідомлень. Таким чином ідеально підходить для відправки нотифікацій на пошту про відновлення паролю чи отримання коду верифікації.

Сервіс користувачів відповідає за логін та реєстрацію нових користувачів. Відповідаючи на події які відбулися в черзі, а отже досягається асинхронна обробка повідомлень та взаємодія мж компонентами системи.

Сервіс виконує всі дії пов’язані с аккаунтом користувача: скидання паролю та підтвердження. Таким чином всі дії користувача будуть ізольовані в рамках одного сервісу та можуть бути самостійно та незалежно виходити до продакшин оточення.

Міжсервісна взаємодія відбувається за рахунок RPC з використанням RabbitMQ, який брокера який і забезпечує асинхронність подій між сервісами.

4.2. Програмна реалізація інтерфейсу користувача

Завершальним етапом розробки системи була реалізація веб-інтерфейсу який надає змогу користувачам отримувати інформацію про товари та створювати нові. Інтерфейс було реалізовано за допомогою бібліотеки React, яка дозволяє створювати Single Page Application з декларативним підходом. Цей підхід робить код передбачуваним та простим для розробки. Елементи інтерфейсу було розбито на компоненти, що дозволяє використовувати їх у різних частинах інтерфейсу. Для роутингу в додатку був використаний вбудований роутинг nextjs бібліотеки, який дозволяє основні дії з . Щоб спростити створення асинхронних запитів на Web API сервер було використано бібліотеку Axios. Також, для пришвидшення розробки веб-інтерфейсу було використано бібліотеку React Bootstrap, що додає адаптивність до сайту з коробки та має набір стилізованих компонентів.

Розробка веб-інтерфейсу почалася зі створення початкової сторінки, на якій виводяться всі звичайні товари. Сторінку можна побачити на рисунку 4.1

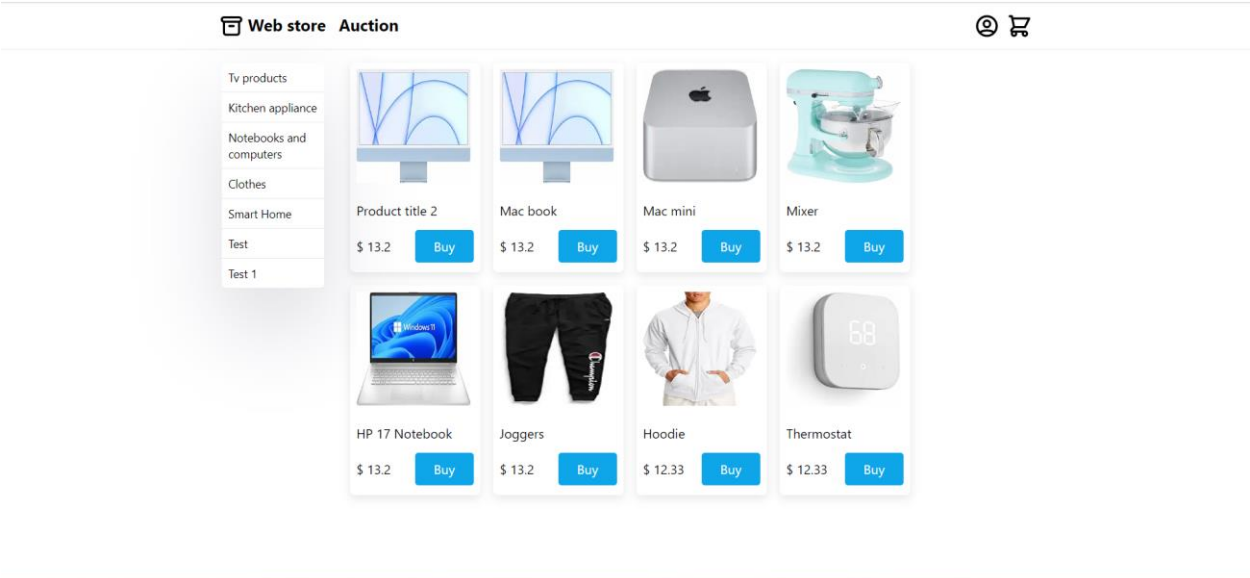

Рисунок 4.2 – Головна сторінка інтерфейсу

Сервіс зможе бути доступний обмежено клієнтам які не пройшли реєстрацію. Для проходження процесу реєстрації та підтвердження номеру створено інтерфейс, який дозволяє користувачам створити або ввійти в існуючий аккаунт.

Вхід відбувається за токеном який отримуємо від серверної частини. Токен зберігається в клієнтському додатку та використовується для створення товарів та додання їх до замовлення.

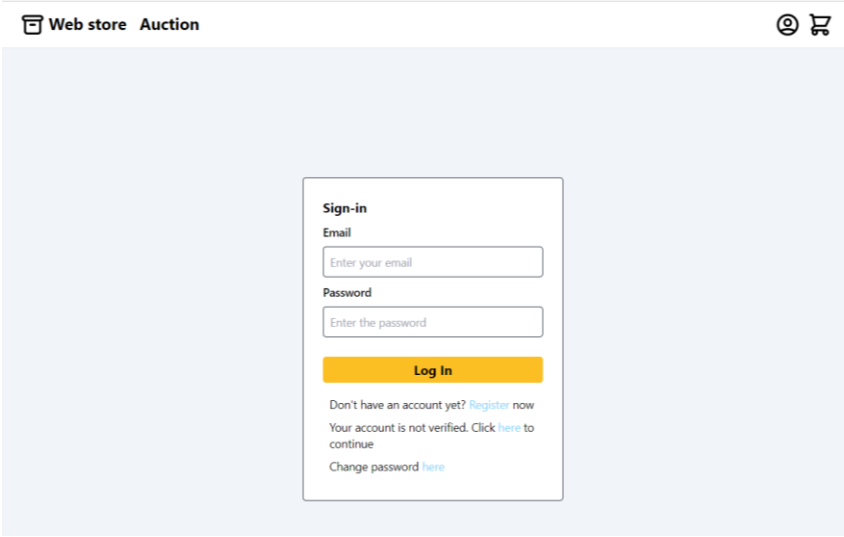

Рисунок 4.3 – Форма авторизації

При виборі категорії відображаються товари доступні для покупки саме в цій категорії. Для відображення використовуються карточка компоненту, яка є одним з основних елементів сервісу.

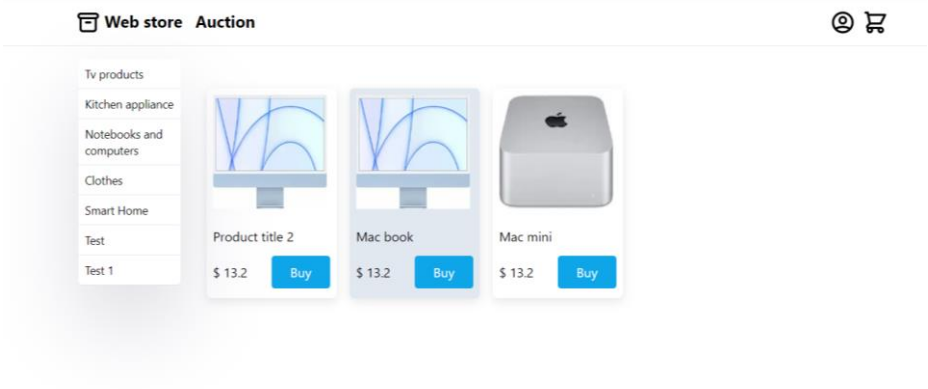

Рисунок 4.4 – Товари відображені за категорією

Відображається звичайний товар який можна придбати за фіксовану ціну та коротка інформація про товар. До інформації входить: назва, ціна, опис продукту, зображення продукту.

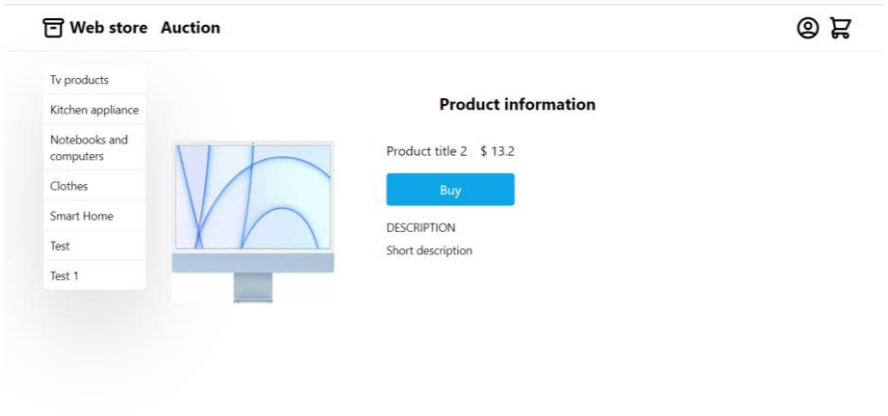

Рисунок 4.5 – Детальна картка товару

Корзина товарів розроблено згідно дизайну та надає основні функції такі як додати товар, зменшити його кількість, повністю видалити його з корзини.

Рисунок 4.6 – Відображення корзини товарів

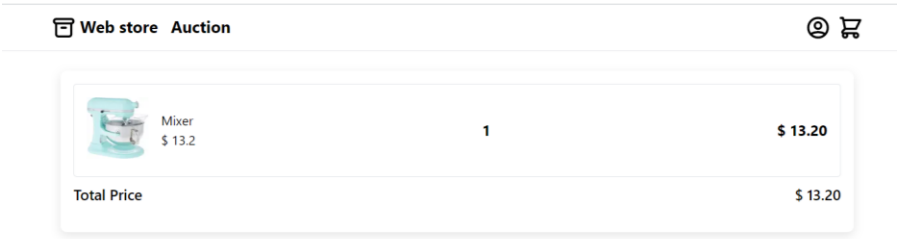

Рисунок 4.7 – Відображення корзини товарів

Для відображення товарів які продаються в вигляді аукціонних товарів створено окрему сторінку на якій є змога ознайомитися з доступними товарами для придбання.

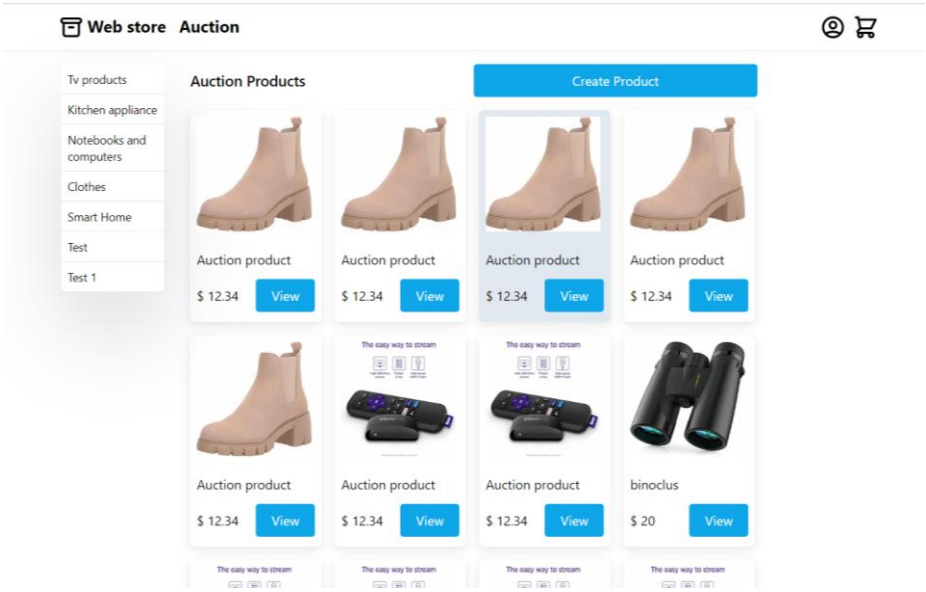

Рисунок 4.8 – Відображення корзини товарів

Користувач повинен розуміти, що товар вже не доступний для продажу та дізнатися ціну за яку товар був проданий. Для інформування користувача відображується текст, що повідомляє користувача що товар був проданий.

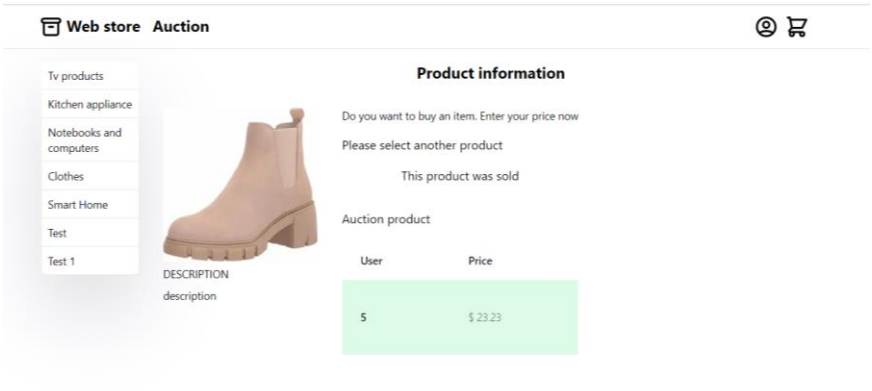

Рисунок 4.9 – Відображення корзини товарів

Для товарів які створені саме користувачем не можна зробити ставку на товар. Текст попередження відображається на екрані. Також надається змога зрозуміти наскільки товар буде доступним для користувачів.

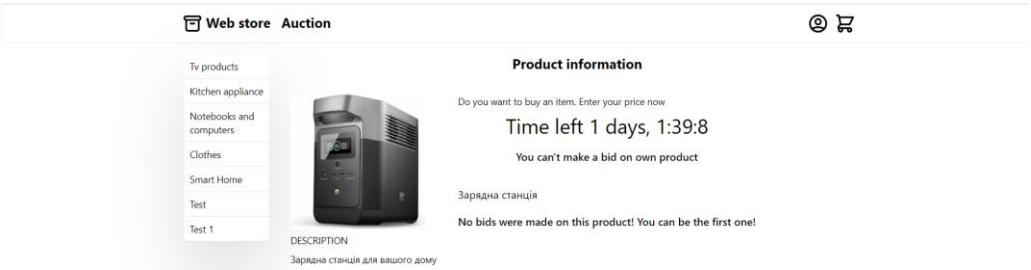

Рисунок 4.10 – Відображення корзини товарів

На вкладці товарів аукціону можна додати чи переглянути наявні товари в магазині. Для отримання інформації про деталі товару потрібно натиснути кнопку “View”. Для створення товару використовується меню в верхній частині.

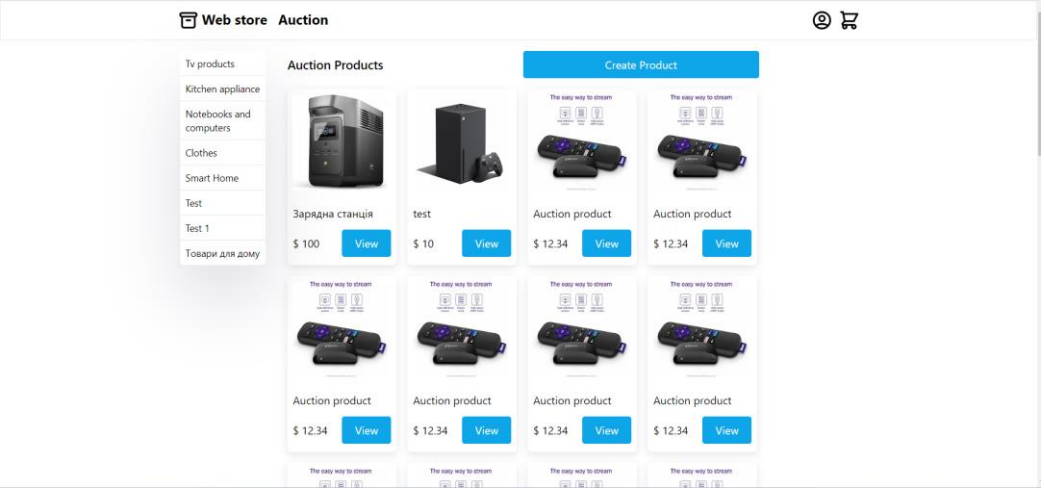

Рисунок 4.11 – Відображення корзини товарів

Для взаємодії з продуктом аукціону є поле вводу ставки на товар, в якому зацікавлена особа може ввести ту суму за яку вона зможе придбати товар. Нижче відображується поле в якому є результати минулих ставок. В правому верхньому куті відображається інформаційне повідомлення в якому вказується, що користувач здійснив ставку на товар.

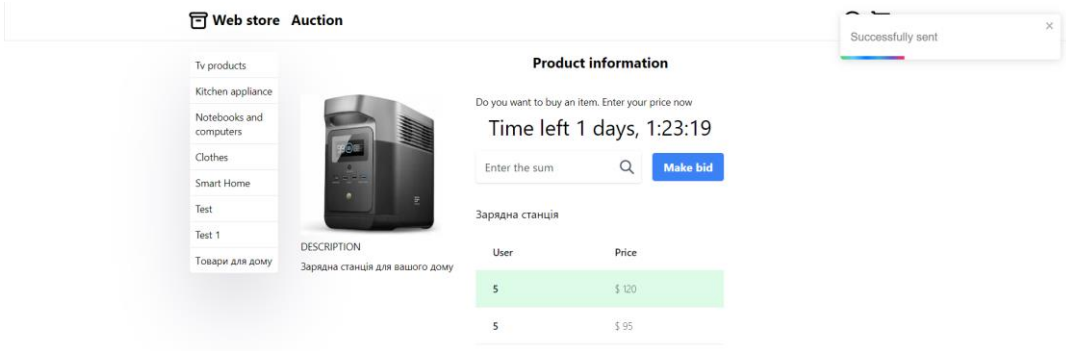

Рисунок 4.12 – Відображення корзини товарів

У випадку, якщо товар був успішно проданий покупцю на стороні серверу відбувається автоматичне встановлення buyerId. Що дозволяє відобразити інформацію про статус товару в даний момент.

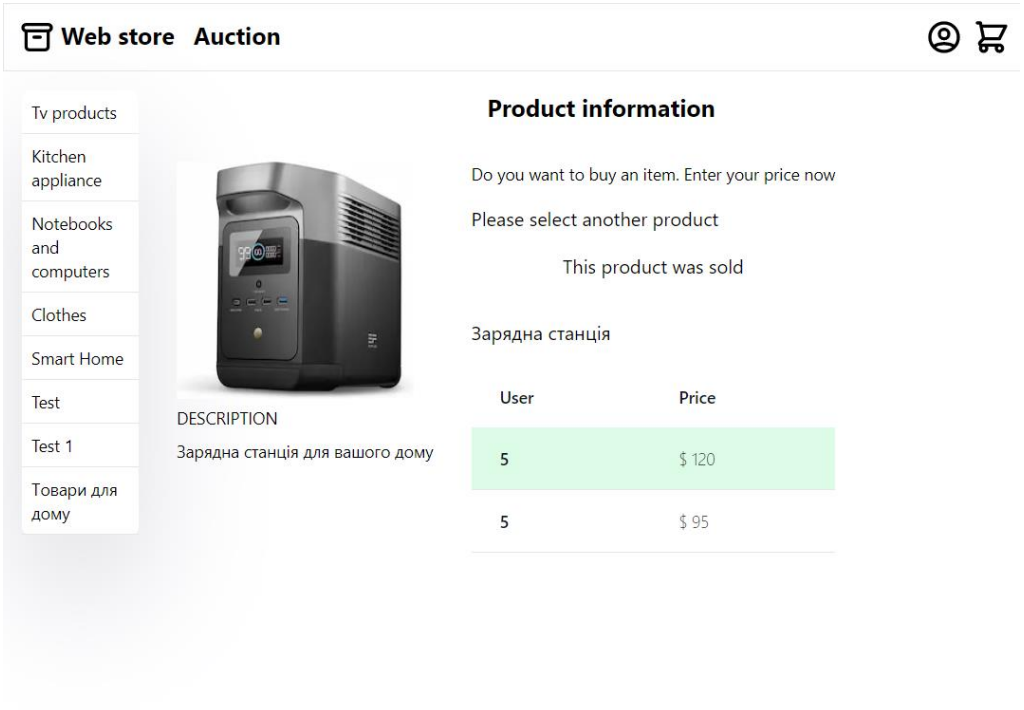

Рисунок 4.13 – Відображення корзини товарів

ВИСНОВОК

У роботі була розроблена інформаційна платформа онлайн аукціону. Створений продукт може використовуватися невеликими компаніями, які займаються електронною комерцією.

Під час роботи було проведено аналіз предметної області, який складався з огляду літератури та аналізу наявних рішень. Досліджено наявні способи забезпечення асинхронної взаємодії, методи побудови веб-додатків та способи взаємодії з веб сервером. Проведено аналіз засобів реалізації програмної реалізації, які дозволять розробити платформу онлайн аукціону. Після аналізу предметної області та засобів реалізації було проведено проектування та моделювання додатку під час якого створено прототип веб-інтерфейсу та спроектована структура бази даних. На завершальному етапі було проведено програмну реалізацію. Результатом роботи є розроблена технологія, яка забезпечує можливість:

- переглядати товари на платформі;
- додавати товари до платформи за допомогою API;
- переглядати товари аукціону;
- відслідковувати історію історію покупок;
- робити ставки на товари аукціону;
- розгортати систему на власних серверах;

СПИСОК ЛІТЕРАТУРИ

1. RFC – Informational [Електронний ресурс] – Режим доступу: https://datatracker.ietf.org/doc/rfc1180/

2. Transmission Control Protocol (TCP) [Електронний ресурс] – Режим доступу: https://www.extrahop.com/resources/protocols/tcp/

3. HTTP (Hypertext Transfer Protocol) [Електронний ресурс] – Режим доступу: https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol

4. Advanced Message Queuing Protocol [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol

5. Клієнт-серверна архітектура [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0

6. Asynchronous message-based communication [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/asynchronous-message-based-communication

7. RabbitMQ 101 [Електронний ресурс] – Режим доступу: https://medium.com/ryanjang-devnotes/rabbitmq-101-d146dd5a7f09

8. Технологія Ajax і сайти в концепції Web 2.0 [Електронний ресурс] – Режим доступу: http://www.znannya.org/?view=AJAX-technology

9. Що таке SPA-додатки [Електронний ресурс] – Режим доступу: https://wezom.com.ua/ua/blog/chto-takoe-spa-prilozheniya

10. Що таке веб додаток? Різниця між сайтом, веб-додатком, spa і pwa [Електронний ресурс] – Режим доступу: https://webcase.com.ua/uk/blog/cho-takoe-web-prilozhenie-vse-vidy/

11. Документація React [Електронний ресурс] – Режим доступу: https://uk.reactjs.org/

12. Документація Nestjs [Електронний ресурс] – Режим доступу: https://docs.nestjs.com/techniques/performance#performance-fastify

13. Документація Prisma [Електронний ресурс] – Режим доступу: https://www.prisma.io/docs/concepts/overview/what-is-prisma

14. Typescript [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/TypeScript

Додаток А

Серверна частина

auth.controller.ts

```
import { Body, Controller, Post, Res, UseFilters } from '@nestjs/common';
import { Response } from 'express';
import { RabbitMqService } from '../rabbit-mq/rabbit-mq.service';
import { RpcExceptionFilterMapping } from
'../utils/filters/RpcExceptionFilter';
import { AuthService } from './auth.service';
import { UserCreateDto } from './dto/UserDto';
import { ConfirmCodeDto } from './dto/ConfirmCodeDto';

@Controller('auth')
@UseFilters(new RpcExceptionFilterMapping())
export class AuthController {
    constructor(private rabbitMQService: RabbitMqService, private
authService: AuthService) {}

    @Post('/login')
    async loginUser(@Body() loginRequest: UserCreateDto, @Res() res):
Promise<Response> {
        return this.authService.login(loginRequest, res);
    }

    @Post('/register')
    async registerUser(@Body() body: UserCreateDto, @Res() res):
Promise<Response> {
        return this.authService.register(body, res);
    }

    @Post('account/confirm')
    async confirmCode(@Body() body: ConfirmCodeDto, @Res() res):
Promise<Response> {
        return this.authService.confirmAccount(body, res);
    }
}
```

auth.service.ts

```
import { Injectable } from '@nestjs/common';
import { Response } from 'express';
import { UserCreateDto } from './dto/UserDto';
import { ConfirmCodeDto } from './dto/ConfirmCodeDto';
import { BaseAuthService } from '../baseAuth/BaseAuth.service';

@Injectable()
export class AuthService extends BaseAuthService {
    async login(body: UserCreateDto, res: Response): Promise<Response> {
        return this.sendResponseTowardToAuthService('login', body, res);
    }

    async register(body: UserCreateDto, res: Response): Promise<Response> {
        return this.sendResponseTowardToAuthService('register', body, res);
    }
```

```
    async confirmAccount(body: ConfirmCodeDto, res: Response):
Promise<Response> {
        return this.sendResponseTowardToAuthService('confirm', body, res);
    }
}
```

auction-products.controller.ts

```
import { Body, Controller, Get, Param, Post, Req, Res, UseGuards } from
'@nestjs/common';
import { Request, Response } from 'express';
import { AuctionProductsService } from './auction-products.service';
import { JwtAuthGuard } from '../jwt/jwt-auth.guard';
import { CreateAuctionProductDto } from './dto/CreateAuctionProductDto';

@Controller('/auction/products')
export class AuctionProductsController {
    constructor(private auctionProduct: AuctionProductsService) {}

    @Get()
    getAll(@Res() res: Response): Promise<Response> {
        return this.auctionProduct.getAll(res);
    }

    @Get('/:id')
    getOneProduct(@Param('id') id, @Res() res: Response): Promise<Response> {
        return this.auctionProduct.getOneProduct(id, res);
    }

    @Post()
    @UseGuards(JwtAuthGuard)
    create(@Body() body: CreateAuctionProductDto, @Req() req: Request, @Res()
res: Response): Promise<Response> {
        return this.auctionProduct.createAuctionProduct(body, req.user, res);
    }
}
```

auction-products.service.ts

```
import { Injectable } from '@nestjs/common';
import { Response } from 'express';
import { BaseOrderService } from '../baseOrder/BaseOrder.service';
import { CreateAuctionProductDto } from './dto/CreateAuctionProductDto';
import { User } from '../utils/dto/User';

@Injectable()
export class AuctionProductsService extends BaseOrderService {
    getAll(res: Response): Promise<Response> {
        return
this.sendResponseTowardToOrderService('/auction/products/getAll', {}, res);
    }

    getOneProduct(id: string, res: Response): Promise<Response> {
        return this.sendResponseTowardToOrderService('/auction/products/one',
{ productId: id }, res);
    }

    createAuctionProduct(body: CreateAuctionProductDto, user: User, res:
Response): Promise<Response> {
```

```
        return
this.sendResponseTowardToOrderService('/auction/products/create', { userId:
user.userId, ...body }, res);
    }
}
```

bids.controller.ts

```
import { Body, Controller, Get, Param, Post, Req, Res, UseGuards } from
'@nestjs/common';
import { Request, Response } from 'express';
import { BidsService } from './bids.service';
import { JwtAuthGuard } from '../jwt/jwt-auth.guard';

@Controller('/auction/products/:id/bids')
export class BidsController {
    constructor(private readonly bidsService: BidsService) {}

    @Get()
    getAll(@Param('id') id, @Res() res): Promise<Response> {
        return this.bidsService.getAllBids(id, res);
    }

    @Post()
    @UseGuards(JwtAuthGuard)
    createBid(@Param('id') id, @Body() data, @Req() req: Request, @Res() res:
Response): Promise<Response> {
        return this.bidsService.createBid({ ...data, userId: req.user.userId,
productId: id }, res);
    }
}
```

bids.service.ts

```
import { HttpStatus, Injectable } from '@nestjs/common';
import { Response } from 'express';
import { RabbitMqService } from '../rabbit-mq/rabbit-mq.service';
import { ErrorHandler } from '../utils/helpers/ErrorHandler';
import { IBareUser, IBid, IBidWithUser } from '../utils/types/types';

@Injectable()
export class BidsService {
    constructor(private readonly rabbitMQService: RabbitMqService) {}

    async getAllBids(id: string, res: Response): Promise<Response> {
        try {
            const allUsersResponse: IBareUser[] = await
this.rabbitMQService.sendAuthService('users-get-all', {});

            const bidsResponse: IBid[] = await
this.rabbitMQService.sendOrderService('/auction/products/bids', { id });

            const bids: IBidWithUser[] = bidsResponse.map((bid: IBid) => {
                return {
                    ...bid,
                    user: allUsersResponse.find((user: IBareUser) =>
bid.userId === user.id),
                };
            });
```

```
            return res.status(HttpStatus.OK).send(bids);
        } catch (error: unknown) {
            return ErrorHandler.handleErrors(res, error as Error);
        }
    }

    async createBid(data: any, res: Response): Promise<Response> {
        try {
            const rabbitResponse: any = await
this.rabbitMQService.sendOrderService(
                '/auction/products/bids/create',
                data,
            );

            return res.status(HttpStatus.OK).send(rabbitResponse);
        } catch (error: unknown) {
            return ErrorHandler.handleErrors(res, error as Error);
        }
    }
}
```

rabbit-mq.module.ts

```
import { Module } from '@nestjs/common';

import { ClientsModule, Transport } from '@nestjs/microservices';

import { ConfigModule, ConfigService } from '@nestjs/config';

import { ClientProvider } from
'@nestjs/microservices/module/interfaces/clients-module.interface';

import { RabbitMqService } from './rabbit-mq.service';


@Module({

    imports: [

        ClientsModule.registerAsync([

            {

                name: 'auth-service',

                imports: [ConfigModule],

                inject: [ConfigService],

                useFactory: async (configService: ConfigService):
Promise<ClientProvider> => {

                    const rabbitmqConnectionUrl: string =
`amqp://${configService.get<string>(

                        'rabbitmq.user',


)}:${configService.get<string>('rabbitmq.password')}@${configService.get<stri
ng>(
```

```
                        'rabbitmq.host',
                    )}/${configService.get<string>('rabbitmq.vhost')}`;

                    return {
                        transport: Transport.RMQ,
                        options: {
                            urls: [rabbitmqConnectionUrl],
                            queue: 'auth-service',
                            queueOptions: {
                                messageTtl: 10_000,
                            },
                        },
                    };
                },
            },
        ]),
        ClientsModule.registerAsync([
            {
                name: 'order-service',
                imports: [ConfigModule],
                inject: [ConfigService],
                useFactory: async (configService: ConfigService):
Promise<ClientProvider> => {
                    const rabbitmqConnectionUrl: string =
`amqp://${configService.get<string>(
                        'rabbitmq.user',

)}:${configService.get<string>('rabbitmq.password')}@${configService.get<stri
ng>(
                        'rabbitmq.host',
                    )}/${configService.get<string>('rabbitmq.vhost')}`;

                    return {
                        transport: Transport.RMQ,
                        options: {
                            urls: [rabbitmqConnectionUrl],
                            queue: 'order-service',
```

```
                            queueOptions: {
                                messageTtl: 10_000,
                            },
                        },
                    };
                },
            },
        ]),
        ConfigModule,
    ],
    providers: [RabbitMqService],
    exports: [RabbitMqService],
})
export class RabbitMqModule {}
```

rabbit-mq.service.ts

```
import { Inject, Injectable } from '@nestjs/common';
import { ClientProxy } from '@nestjs/microservices';
import { firstValueFrom, timeout } from 'rxjs';
import { AuthServiceDto } from '../baseAuth/dto/AuthServiceDTo';
import { OrderServiceMessageType } from '../utils/dto/AuthServiceMessageDto';
import { ProductsMessage } from '../product/Dto/ProductsMessage';
import { CreateAuctionProductDto } from '../auction-
products/dto/CreateAuctionProductDto';
import { GetOneProductInfo } from '../auction-
products/dto/GetOneProductInfo';

const TIMEOUT: number = 10_000;

@Injectable()
export class RabbitMqService {
    constructor(
        @Inject('auth-service') private readonly authClient: ClientProxy,
        @Inject('order-service') private readonly orderClient: ClientProxy,
    ) {}
```

```
    public sendAuthService(pattern: string, data: AuthServiceDto = {}):
Promise<any> {

        return RabbitMqService.send(this.authClient, pattern, data);

    }


    public sendOrderService(

        pattern: string,

        data: OrderServiceMessageType | ProductsMessage |
CreateAuctionProductDto | GetOneProductInfo = {},

    ): Promise<any> {

        return RabbitMqService.send(this.orderClient, pattern, data);

    }


    private static send(client: ClientProxy, pattern: string, data: any):
Promise<any> {

        return firstValueFrom(client.send(pattern,
data).pipe(timeout(TIMEOUT)));

    }

}
```

email-message.controller.ts (notification service)

```
import { Controller } from '@nestjs/common';
import { Ctx, EventPattern, Payload, RmqContext } from
'@nestjs/microservices';
import { RabbitMqQueues } from '../consts/RabbitMqQueues';
import { EmailMessageService } from './email-message.service';
import { IEmailMessage } from '../types/IEmailMessage';

@Controller()
export class EmailMessageController {
    constructor(private readonly emailMessageService: EmailMessageService) {}

    @EventPattern(RabbitMqQueues.AccountVerification)
    public async sendAccountVerification(@Payload() body: IEmailMessage,
@Ctx() context: RmqContext): Promise<void> {
        await this.emailMessageService.sendAccountVerification(body,
context);
    }

    @EventPattern(RabbitMqQueues.ResetPassword)
    public async sendResetPassword(@Payload() body: IEmailMessage, @Ctx()
context: RmqContext): Promise<void> {
        await this.emailMessageService.sentResetPassword(body, context);
    }
}
```

email-message.service.ts(notification service)

```
import { Injectable } from '@nestjs/common';
import { RmqContext } from '@nestjs/microservices';
import { RabbitMqOption } from '../types/RabbitMqOption';
import { EmailMessageBuilder } from '../mail/builder/EmailMessageBuilder';
import { IEmailMessage } from '../types/IEmailMessage';
import { MailService } from '../mail/mail.service';
import { IEmailMessages } from '../types/IEmailMessages';

@Injectable()
export class EmailMessageService {
    constructor(private readonly mailService: MailService) {}

    async sendAccountVerification({ email, code }: IEmailMessage, context:
RmqContext): Promise<void> {
        await this.sendGeneralEmailMessage(new
EmailMessageBuilder().buildVerification([email], code).build(), context);
    }

    async sentResetPassword({ email, code }: IEmailMessage, context:
RmqContext): Promise<void> {
        await this.sendGeneralEmailMessage(new
EmailMessageBuilder().buildReset([email], code).build(), context);
    }

    async sendGeneralEmailMessage(emailMessages: IEmailMessages, context:
RmqContext): Promise<void> {
        const { channel, originalMessage } =
this.getRabbitMqOptions(context);

        await this.mailService.sendUserConfirmation(emailMessages);

        channel.ack(originalMessage);
    }

    getRabbitMqOptions(context: RmqContext): RabbitMqOption {
        const channel: any = context.getChannelRef();
        const originalMessage: Record<string, any> = context.getMessage();

        return { channel, originalMessage };
    }
}
```

Клієнтська частина

```
Index.tsx
import Products from './components/Products/Products';
import CategoryBar from './components/CategoryBar';
import { useFetchProductsQuery } from '../logic/services/services';
import Loader from './components/Loader/Loader';
import ErrorComponent from './components/ErrorComponent';

export default function Home() {
    const { data, isError, isLoading } = useFetchProductsQuery('');


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

```
    if (isError) {
        return <ErrorComponent />;
    }

    return (
        <div>
            <div className={'m-auto grid max-w-5xl grid-cols-6 justify-center
gap-4'}>
                <CategoryBar />
                <div className={'col-span-5 mt-4 self-end'}>
                    <Products products={data} />
                </div>
            </div>
        </div>
    );
}
```

Layout.tsx

```
import Header from './components/Header/Header';
import { store } from '../logic/store';
import { Provider } from 'react-redux';
import HtmlHead from './components/Head/Head';
import { PersistGate } from 'redux-persist/integration/react';
import { persistStore } from 'redux-persist';
import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

const persistor = persistStore(store);


export default function Layout({ children }) {
    return (
        <>
            <Provider store={store}>
                <PersistGate persistor={persistor}>
                    <HtmlHead />
                    <Header />
                    <main>{children}</main>
                    <ToastContainer />
                </PersistGate>
            </Provider>
        </>
    );
}
```

login.tsx

```
import Link from 'next/link';
import { ChangeEvent, useState } from 'react';
import { setCredentials } from '../logic/authSlice';
import { useDispatch } from 'react-redux';
import { useRouter } from 'next/router';
import { useLoginMutation } from '../logic/services/services';
import { Routes } from '../utils/Routes';
import { ErrorHelper } from '../utils/ErrorHelper';
import { toast } from 'react-toastify';
import { Messages } from '../utils/Messages';
import { LoginRequest } from '../utils/types/api';

export default function Login() {
    const dispatch = useDispatch();
    const { push } = useRouter();
    const [formState, setFormState] = useState<LoginRequest>({
```

```
        email: '',
        password: '',
    });

    const [login, { isLoading }] = useLoginMutation();
    const handleChange = ({
                              target: { name, value },
                          }: ChangeEvent<HTMLInputElement>) =>
        setFormState((prev) => ({ ...prev, [name]: value }));


    const loginRequest = async (event: { preventDefault: () => void; }) => {
        event.preventDefault();
        try {
            const result = await login(formState);
            if ('data' in result) {
                dispatch(setCredentials(result.data));
                return await push(Routes.root);
            }
            if ('error' in result) {
                toast(ErrorHelper.getResponseError(result));
            }
        } catch (err) {
            toast(Messages.SomethingWentWrong);
        }
    };

    return (
        <div className={'unauthorized-background'}>

            <form className={'unauthorized-form'}>
                <h3 className={'mb-4 text-3xl font-bold sm:mb-2 sm:text-
base'}>Sign-in</h3>
                <label htmlFor='email' className={'mb-2 flex flex-col text-xl
font-semibold sm:text-sm'}>
                    Email
                </label>
                <input id='email' className={'mb-2 rounded border border-
gray-800 p-2 text-xl sm:text-sm'}
                       placeholder={'Enter your email'}
                       onChange={handleChange}
                       name='email' />

                <label htmlFor='password' className={'mb-2 flex flex-col
text-xl font-semibold sm:text-sm'}>
                    Password
                </label>
                <input className={'mb-10 rounded border border-gray-800 p-2
text-xl sm:mb-6  sm:text-sm'} id='password'
                       name='password'
                       onChange={handleChange}
                       placeholder={'Enter the password'} />

                <button
                    className={'mb-4 w-full rounded bg-amber-400 p-3 text-xl
font-bold hover:bg-amber-300 sm:p-1 sm:text-base '}
                    disabled={isLoading}
                    onClick={loginRequest}>Log In
                </button>

                <p className={'mb-2 pl-2 text-lg sm:text-sm'}>
                    Don't have an account yet?
                    <Link href={Routes.register}>
```

```
                        <a className={'pl-1 text-sky-300'}>Register</a>
                    </Link> now
                </p>

                <p className={'mb-2 inline-block pl-2 text-lg sm:text-sm'}>
                    Your account is not verified. Click
                    <Link href={Routes.confirm}>
                        <a className={'pl-1 text-sky-300'}>here</a>
                    </Link> to continue
                </p>

                <p className={'mb-2 inline-block pl-2 text-lg sm:text-sm'}>
                    Change password
                    <Link href={Routes.reset}>
                        <a className={'pl-1 text-sky-300'}>here</a>
                    </Link>
                </p>
            </form>
        </div>
    );
}
```

product/[id].tsx

```
import CategoryBar from "../../components/CategoryBar";
import Image from "next/image";
import { useRouter } from "next/router";
import { useGetAuctionProductQuery, useMakeBidMutation } from
"../../../logic/services/services";
import Loader from "../../components/Loader/Loader";
import ErrorComponent from "../../components/ErrorComponent";
import Bids from "../../components/Bids";
import { ChangeEventHandler, FormEvent, useEffect, useState } from "react";
import { toast } from "react-toastify";
import { Messages } from "../../../utils/Messages";
import { ErrorHelper } from "../../../utils/ErrorHelper";
import { useSelector } from "react-redux";
import { selectCurrentUser } from "../../../logic/authSlice";
import BidInput from "../../components/BidInput";

export default function AuctionProduct({ id: productId }) {
  const [bidPrice, setBidPrice] = useState("");
  const { id: userId } = useSelector(selectCurrentUser);

  const { data: product, isError, isLoading } =
useGetAuctionProductQuery(productId as string);

  const [makeBidRequest] = useMakeBidMutation();

  const resetStateInput = () => {
    setBidPrice("");
  };
  if (isLoading) {
    return <Loader />;
  }

  if (isError) {
    return <ErrorComponent />;
  }

  const { id, title, mainImg, description, creatorId, buyerId, sellTil, bids
} = product;

  const createdByUserProduct = () => {
```

```
    return creatorId === userId;
  };

  const checkBidPrice = () => {
    const priceNumber = parseFloat(bidPrice);
    const previousBidPrice = bids[0]?.price;

    return priceNumber <= previousBidPrice;
  };

  const makeBid = async () => {
    if (bidPrice === "" || parseFloat(bidPrice) <= 0) {
      return toast("Your price must be a number more than 0");
    }
    if (checkBidPrice()) {
      return toast("Your price should be bigger than previous bid price");
    }


    try {
      const response = await makeBidRequest({
        id,
        price: parseFloat(bidPrice)
      });
      if ("data" in response) {
        resetStateInput();
        toast(Messages.SuccessfullySent);
      }
      if ("error" in response) {
        toast(ErrorHelper.getResponseError(response));
      }
    } catch (e) {
      toast(Messages.SomethingWentWrong);
    }
  };

  const isProductSold: () => boolean = () => {
    return buyerId !== null || Date.now() > Date.parse(sellTil);
  };

  const changeInputType: ChangeEventHandler<HTMLInputElement> = (e:
FormEvent<HTMLInputElement>) => {
    setBidPrice(e.currentTarget.value);
  };

  return (
    <div>
      <div className={"m-auto grid max-w-5xl grid-cols-6 justify-center gap-
4"}>
        <CategoryBar />
        <div className={"col-span-5 mt-4 self-end"}>
          <h2 className={"mb-8 text-center text-xl font-bold"}>Product
information</h2>
          <div className={"flex"}>
            <div>
              <Image width={200} height={200}
                     alt={title}
                     src={mainImg} />
              <div className={""}>
                <h3 className={"mb-2 text-sm uppercase"}>Description</h3>
                <p className={"text-sm"}>{description}</p>
              </div>
            </div>
```

```
            <div className={"flex flex-col gap-8"}>
              <BidInput
                isProductSold={isProductSold()}
                createdByUserProduct={createdByUserProduct()}
                bidPrice={bidPrice}
                changeInputType={changeInputType}
                makeBid={makeBid}
                sellTil={sellTil} />
              <div className={"ml-8 flex flex-col gap-4"}>
                <div className={"flex gap-4"}>
                  <span>{title}</span>
                </div>
                <div className={"inline-block h-96 min-w-full overflow-y-
auto"}>
                  <Bids bids={bids} />
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}

export async function getServerSideProps(ctx) {
  const { id } = ctx.query;

  return {
    props: {
      id
    }
  };
}
```

PrivateRoute.tsx

```
import React from 'react';
import { useSelector } from 'react-redux';
import { selectCurrentUser } from '../../logic/authSlice';
import { Routes } from '../../utils/Routes';
import Link from 'next/link';
import { PrivateRouteProps } from '../../utils/types/props';

export default function PrivateRoute({ children, href }: PrivateRouteProps) {
    const user = useSelector(selectCurrentUser);

    const isVerified = (): boolean => user.isVerified;

    const userExists = (): boolean => user.email !== null && user.token !==
null;

    if (userExists() && !isVerified()) {
        return <Link href={Routes.confirm} passHref>{children}</Link>;
    }

    if (!userExists()) {
        return <Link href={Routes.login} passHref>{children}</Link>;
    }

    return <Link href={href} passHref>{children}</Link>;
}
```

AuctionProducts.tsx

```
import { Routes } from '../../../utils/Routes';
import ProductCard from '../Products/ProductCard';
import CardButton from '../CardButton';
import { AuctionProductsProps } from '../../../utils/types/props';
import { IAuctionProduct } from '../../../utils/types/types';


export default function AuctionProducts({ products }: AuctionProductsProps) {
    return (
        <div className={'flex flex-wrap gap-4'}>
            {
                products.map(({ id, title, mainImg, startPrice: price }:
IAuctionProduct) => {

                    return <ProductCard key={id}
                                        title={title}
                                        mainImg={mainImg}

href={`${Routes.auctionProduct}/${id}`}>
                            <span className={'inline-flex items-center text-
base'}>$ {price}</span>
                            <CardButton
link={`${Routes.auctionProduct}/${id}`} linkText={'View'}>View</CardButton>
                        </ProductCard>;
                },
            )
            }
        </div>
    );
}
Footer.tsx

import Image from 'next/image';

export default function Footer() {
    return (<footer>
        <a
            href='https://vercel.com?utm_source=create-next-
app&utm_medium=default-template&utm_campaign=create-next-app'
            target='_blank'
            rel='noopener noreferrer'
        >
            Powered by{' '}
            <Image src='/vercel.svg' alt='Vercel Logo' width={72} height={16}
/>
        </a>
    </footer>);

}
Header.tsx

import Link from 'next/link';
import { Routes } from '../../../utils/Routes';
import PrivateRoute from '../PrivateRoute';

export default function Header() {
    return (
        <header className={'w-full border'}>
            <nav className={'m-auto my-3 flex max-w-5xl justify-between px-3
'}>
                <div className={'flex'}>
                    <Link href={Routes.root} passHref={true}>
                        <div className={'flex cursor-pointer'}>
```

```
                            <svg className='h-8 w-8'
xmlns='http://www.w3.org/2000/svg' fill='none'
                                 viewBox='0 0 24 24'
                                 stroke='currentColor' strokeWidth={2}>
                                <path strokeLinecap='round'
strokeLinejoin='round'
                                      d='M5 8h14M5 8a2 2 0 110-4h14a2 2 0 110
4M5 8v10a2 2 0 002 2h10a2 2 0 002-2V8m-9 4h4' />
                            </svg>
                            <h2 className={'ml-1 text-xl font-bold'}>Web
store</h2>
                        </div>
                    </Link>
                    <div className={'ml-4 cursor-pointer'}>
                        <Link href={Routes.auction} passHref>
                            <span className={'text-xl font-
bold'}>Auction</span>
                        </Link>

                    </div>
                </div>
                <ul className={'flex'}>
                    <li>
                        <PrivateRoute href={Routes.profile}>
                            <svg className='h-8 w-8 cursor-pointer'
fill='none' stroke='currentColor'
                                 viewBox='0 0 24 24'
                                 xmlns='http://www.w3.org/2000/svg'>
                                <path strokeLinecap='round'
strokeLinejoin='round' strokeWidth='2'
                                      d='M5.121 17.804A13.937 13.937 0 0112
16c2.5 0 4.847.655 6.879 1.804M15 10a3 3 0 11-6 0 3 3 0 016 0zm6 2a9 9 0 11-
18 0 9 9 0 0118 0z'></path>
                            </svg>
                        </PrivateRoute>
                    </li>
                    <li className={'ml-2'}>
                        <Link href={Routes.cart} passHref>
                            <svg className='h-8 w-8 cursor-pointer'
fill='none' stroke='currentColor'
                                 viewBox='0 0 24 24'
                                 xmlns='http://www.w3.org/2000/svg'>
                                <path strokeLinecap={'round'}
strokeLinejoin='round' strokeWidth='2'
                                      d='M3 3h2l.4 2M7 13h10l4-8H5.4M7 13L5.4
5M7 13l-2.293 2.293c-.63.63-.184 1.707.707 1.707H17m0 0a2 2 0 100 4 2 2 0
000-4zm-8 2a2 2 0 11-4 0 2 2 0 014 0z'></path>
                            </svg>
                        </Link>
                    </li>
                </ul>
            </nav>
        </header>);
}
Bids.tsx

import { BidsProps } from '../../utils/types/props';

export default function Bids({ bids }: BidsProps) {
    if (bids.length === 0) {
        return <div className={'font-semibold'}>No bids were made on this
product! You can be the first one!</div>;
    }
```

```
    return <table className={'w-full table-auto'}>
        <thead>
        <tr>
            <th className={'px-6 py-4 text-left text-sm font-medium text-
gray-900'}>User</th>
            <th className={'px-6 py-4 text-left text-sm font-medium text-
gray-900'}>Price</th>
        </tr>
        </thead>
        <tbody className={'h-24'}>
        {
            bids.map(({ id, userId, price }) => {
                return (
                    <tr key={id}
                        data-tooltip-target='tooltip-default'
                        className={`border-b bg-white transition duration-300
ease-in-out first-of-type:bg-green-100 hover:bg-gray-100`}>
                        <td className={'whitespace-nowrap px-6 py-4 text-sm
font-medium text-gray-900'}>{userId}</td>
                        <td className={'whitespace-nowrap px-6 py-4 text-sm
font-light text-gray-900'}>$ {price} </td>
                    </tr>
                );
            })}
        </tbody>
    </table>;
}
```

BidInputs.tsx

```
import CountDownTimer from './auction/timer/CountDownTimer';
import { BidInputProps } from '../../utils/types/props';

export default function BidInput({
                                     isProductSold,
                                     createdByUserProduct,
                                     bidPrice,
                                     changeInputType,
                                     makeBid,
                                     sellTil,
                                 }: BidInputProps) {

    return <div className={'ml-8'}>
        <h2 className={'mb-2 text-sm'}>Do you want to buy an item. Enter your
price now</h2>
        <CountDownTimer dateTime={sellTil} />
        <div
            className={'group relative flex w-full items-center justify-
center'}>
            {
                isProductSold ? <p>This product was sold</p> :
createdByUserProduct ?
                    <div className={'font-semibold'}>You can't make a bid on
own product</div> :
                    <>
                        <div
                            className='group relative flex items-center
justify-center'>
                            <input
```

```
                                placeholder='Enter the sum'
                                type={'number'}
                                value={bidPrice.toString()}
                                onChange={changeInputType}
                                className='flex flex-none appearance-none
rounded border border-transparent p-3 pr-10 leading-tight text-gray-700
shadow outline-none placeholder:text-gray-600 focus:border-gray-400
focus:outline-none'
                                required
                            />
                            <span
                                className='absolute right-0 flex rounded bg-
transparent p-2 text-base text-gray-600'>
                                                                <svg fill='none'
stroke='currentColor' strokeLinecap='round'

strokeLinejoin='round' strokeWidth='2'
                                                                     viewBox='0 0
24 24'
                                                                     className='h-6
w-6'>
                                                                <path d='M21 21l-6-
6m2-5a7 7 0 11-14 0 7 7 0 0114 0z'></path>
                                                                </svg>
                                                            </span>
                        </div>
                        <button
                            className={'ml-4 rounded bg-blue-500 py-2 px-4
font-bold text-white hover:bg-blue-700'}
                            onClick={makeBid}
                        >
                            Make bid
                        </button>
                    </>
            }
        </div>
    </div>;
}
Cart.tsx

import { clearCart, selectCart } from '../logic/orderSlice';
import { useDispatch, useSelector } from 'react-redux';
import ProductRow from './components/ProductRow/ProductRow';
import { useCreateOrderMutation } from '../logic/services/services';
import { Routes } from '../utils/Routes';
import { useRouter } from 'next/router';
import PrivateRoute from './components/PrivateRoute';
import { selectUserExists } from '../logic/authSlice';
import { toast } from 'react-toastify';
import { Messages } from '../utils/Messages';
import { CartItem, CartProductInfo } from '../utils/types/types';

export default function Cart() {
    const cart: CartItem[] = useSelector(selectCart);

    const isUserExists: boolean = useSelector(selectUserExists);

    const dispatch = useDispatch();
    const { push } = useRouter();

    const [createOrder] = useCreateOrderMutation();

    const createOrderRequest = async () => {
```

```
        const products: CartProductInfo[] = cart.map((cartItem: CartItem) =>
{
            return { amount: cartItem.amount, productId: cartItem.product.id
};
        });
        const result = await createOrder({ products });
        if ('data' in result) {
            dispatch(clearCart());
            return await push(Routes.orders);
        }
        if ('error' in result) {
            toast(Messages.SuccessfullySent);
        }
    };

    const totalPrice: string = cart.reduce((acc, val) => {
        return acc + (val.amount * val.product?.price);
    }, 0).toFixed(2);

    return (
        <div className={'m-auto max-w-5xl'}>
            <div className={'m-6'}>
                <div className={'flex justify-between'}>
                    <span className={'p-2 text-lg'}>Cart Items</span>
                    <span className={'flex items-baseline rounded bg-sky-500
p-2 text-white hover:bg-sky-600'}>
                        <PrivateRoute href={Routes.orders}>Previous
orders</PrivateRoute>
                    </span>
                </div>
                {cart.length ? <div>
                        <div className={'rounded'}>
                            {cart.map((product: CartItem) => <ProductRow
key={product.product.id} cartItem={product} />)}
                        </div>
                        <div className={'mb-4 flex justify-between text-lg
font-semibold'}>
                            <div>Total Price</div>
                            <div>$ {totalPrice}</div>
                        </div>
                        <button disabled={!isUserExists}
                                onClick={createOrderRequest}
                                className={'w-full rounded bg-amber-400 p-3
text-xl font-bold hover:bg-amber-300 disabled:bg-slate-300 sm:p-1 sm:text-
base'}>
                            Make an order
                        </button>
                    </div> :
                    <div className={'mt-4 mb-8 text-center text-xl font-
semibold'}>You haven't add any thing to the
                        card</div>}
                <div>
                </div>
            </div>
        </div>
    );
}
```

products/[id].tsx

```
import CategoryBar from '../components/CategoryBar';
import Image from 'next/image';
```

```
import { useGetSingleProductQuery } from '../../logic/services/services';
import { useRouter } from 'next/router';
import Loader from '../components/Loader/Loader';
import CardButton from '../components/CardButton';
import ErrorComponent from '../components/ErrorComponent';
import { Routes } from '../../utils/Routes';
import { useDispatch, useSelector } from 'react-redux';
import { addProduct, selectCart } from '../../logic/orderSlice';
import { CartItem, IProduct } from '../../utils/types/types';


export default function SingleProduct() {
    const router = useRouter();
    const dispatch = useDispatch();

    const cart: CartItem[] = useSelector(selectCart);

    const { data: product, isError, isLoading } =
useGetSingleProductQuery(router.query.id as string);


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

    if (isError) {
        return <ErrorComponent />;
    }

    const addProductToCart = (product: IProduct) => () => {
        dispatch(addProduct(product));
    };
    const { id, title, mainImg, price, description } = product;
    const isAddedToCart = cart.some(({ product: cartProduct }: CartItem) =>
cartProduct.id === id);

    return (
        <div>
            <div className={'m-auto grid max-w-5xl grid-cols-6 justify-center
gap-4'}>
                <CategoryBar />
                <div className={'col-span-5 mt-4 self-end'}>
                    <h2 className={'mb-8 text-center text-xl font-
bold'}>Product information</h2>
                    <div>
                        <div className={'flex gap-8 '}>
                            <Image width={200} height={200}
                                   alt={title}
                                   src={mainImg} />
                            <div className={'ml-8 flex flex-col gap-4'}>
                                <div className={'flex gap-4'}>
                                    <span>{title}</span>
                                    <span>$ {price}</span>
                                </div>
                                <CardButton
                                    displayLink={isAddedToCart}
                                    link={Routes.cart}
                                    linkText={'Check cart'}
                                    handle={addProductToCart(product)}
                                >Buy</CardButton>
                                <div>
                                    <h3 className={'mb-2 text-sm
uppercase'}>Description</h3>
```

```
                                        <p className={'text-
sm'}>{description}</p>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            );
}
services.ts

import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';
import { RootState } from '../store';
import {
    AuctionProductCreateFormRequest,
    ConfirmRequest, CreateBidRequest,
    CreateOrderRequest, IObtainTokenRequest, IResetPasswordRequest,
ISendCodeMessageRequest,
    LoginRequest,
} from '../../utils/types/api';
import {
    CreateBid,
    CreateBidResponse,
    CreateOrder,
    CreateOrderResponse,
    FetchProductsResponse,
    GetAuctionProductsResponse,
    GetCategoriesResponse,
    GetCategoryProductsResponse,
    GetSingleAuctionProductResponse,
    GetSingleProductResponse, IObtainToken, IObtainTokenResponse,
IResetPasswordResponse, ISendCodeMessageResponse,
    OrderHistory,
    OrderHistoryResponse, UserLogin, UserLoginResponse, UserRegister,
UserRegisterResponse,
} from '../../utils/types/api/responses';
import { IAuctionProduct, ICategory, IProduct } from
'../../utils/types/types';


// Define a service using a base URL and expected endpoints
export const shopApi = createApi({
        reducerPath: 'shopApi',
        tagTypes: ['Orders', 'AuctionProducts', 'SingleAuctionProduct'],
        baseQuery: fetchBaseQuery({
            baseUrl: 'http://localhost:3001/',
            prepareHeaders: (headers, { getState }) => {
                // By default, if we have a token in the store, let's use
that for authenticated requests
                const token = (getState() as RootState).auth.token;
                if (token) {
                    const authHeader = 'Authorization';
                    headers.set(authHeader, `Bearer ${token}`);
                }
                return headers;
            },
        }),
        endpoints: (builder) => ({
            fetchProducts: builder.query<IProduct[], string>({
                transformResponse: (response: FetchProductsResponse) =>
response?.payload,
```

```
                query: () => `products`,
            }),
            getCategories: builder.query<ICategory[], string>({
                transformResponse: (response: GetCategoryProductsResponse) =>
response?.payload,
                query: () => `categories`,
            }),
            getCategoryProducts: builder.query<IProduct[], string>({
                transformResponse: (response: GetCategoryProductsResponse) =>
response?.payload,
                query: (id) => `categories/${id}/products`,
            }),
            getSingleProduct: builder.query<IProduct, string>({
                transformResponse: (response: GetSingleProductResponse) =>
response?.payload,
                query: (id) => `products/${id}`,
            }),
            getAuctionProducts: builder.query<IAuctionProduct[], string>({
                transformResponse: (response: GetAuctionProductsResponse) =>
response?.payload,
                query: () => `/auction/products`,
                providesTags: ['AuctionProducts'],
            }),
            getAuctionProduct: builder.query<IAuctionProduct, string>({
                transformResponse: (response:
GetSingleAuctionProductResponse) => response?.payload,
                query: (id) => `auction/products/${id}`,
                providesTags: ['SingleAuctionProduct'],
            }),
            getOrderHistory: builder.query<OrderHistory[], any>({
                transformResponse: (response: OrderHistoryResponse) =>
response?.payload,
                query: () => `orders`,
                providesTags: (result) =>
                    result
                        ? [...result.map(({ id }) => ({ type: 'Orders' as
const, id })), 'Orders']
                        : ['Orders'],
            }),
            login: builder.mutation<UserLogin, LoginRequest>({
                transformResponse: (response: UserLoginResponse) =>
response?.payload,
                query: (credentials) => ({
                    url: 'auth/login',
                    method: 'POST',
                    body: credentials,
                }),
            }),
            confirm: builder.mutation<UserLogin, ConfirmRequest>({
                transformResponse: (response: UserLoginResponse) =>
response?.payload,
                query: (credentials) => ({
                    url: 'auth/account/confirm',
                    method: 'POST',
                    body: credentials,
                }),
            }),
            register: builder.mutation<UserRegister, LoginRequest>({
                transformResponse: (response: UserRegisterResponse) =>
response?.payload,
                query: (credentials) => ({
                    url: 'auth/register',
                    method: 'POST',
```

```
                    body: credentials,
                }),
            }),
            createOrder: builder.mutation<CreateOrder, CreateOrderRequest>({
                transformResponse: (response: CreateOrderResponse) =>
response?.payload,
                query: (data) => ({
                    url: '/orders',
                    method: 'POST',
                    body: data,
                }),
                invalidatesTags: ['Orders'],
            }),
            sendResetPasswordCode: builder.mutation<ISendCodeMessageResponse,
ISendCodeMessageRequest>({
                query: (data) => ({
                    url: '/password/verification',
                    method: 'POST',
                    body: data,
                }),
            }),
            getPasswordResetToken: builder.mutation<IObtainToken,
IObtainTokenRequest>({
                transformResponse: (response: IObtainTokenResponse) =>
response?.payload,
                query: (data) => ({
                    url: '/password/token',
                    method: 'POST',
                    body: data,
                }),
            }),
            resetPassword: builder.mutation<IResetPasswordResponse,
IResetPasswordRequest>({
                // do i need a transform here
                query: (data) => ({
                    url: '/password/reset',
                    method: 'POST',
                    body: data,
                }),
            }),
            makeBid: builder.mutation<CreateBid, CreateBidRequest>({
                transformResponse: (response: CreateBidResponse) =>
response?.payload,
                query: ({ id, ...data }) => ({
                    url: `/auction/products/${id}/bids`,
                    method: 'POST',
                    body: data,
                }),
                invalidatesTags: ['AuctionProducts', 'SingleAuctionProduct'],
            }),
            createAuctionProduct: builder.mutation<IAuctionProduct,
AuctionProductCreateFormRequest>({
                query: (data) => ({
                    url: '/auction/products',
                    method: 'POST',
                    body: data,
                }),
                invalidatesTags: ['AuctionProducts'],
            }),
            protected: builder.mutation({
                query: () => 'protected',
            }),
        }),
```

```
    })
;

// Export hooks for usage in functional components, which are
// auto-generated based on the defined endpoints
export const {
    useLoginMutation,
    useConfirmMutation,
    useCreateOrderMutation,
    useSendResetPasswordCodeMutation,
    useGetPasswordResetTokenMutation,
    useResetPasswordMutation,
    useProtectedMutation,
    useRegisterMutation,
    useGetOrderHistoryQuery,
    useFetchProductsQuery,
    useGetCategoriesQuery,
    useGetCategoryProductsQuery,
    useGetSingleProductQuery,
    useGetAuctionProductQuery,
    useGetAuctionProductsQuery,
    useMakeBidMutation,
    useCreateAuctionProductMutation,
} = shopApi;
```

authSlice.ts

```
import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import { RootState } from './store';


const slice = createSlice({
    name: 'auth',
    initialState: {
        id: null,
        email: null,
        name: null,
        isVerified: null,
        token: null,
    } as {
        id: number | null;
        email: string | null;
        name: string | null;
        isVerified: boolean | null;
        token: string | null;
    },
    reducers: {
        setCredentials: (
            state,
            { payload: { id, email, name, isVerified, accessToken } }:
                PayloadAction<{ id: number; email: string; name: string |
null; isVerified: boolean; accessToken?: string }>,
        ) => {
            state.id = id;
            state.email = email;
            state.name = name;
            state.isVerified = isVerified;
            state.token = accessToken;
        },
        logout: (state) => {
            state.id = null;
            state.email = null;
```

```
            state.name = null;
            state.isVerified = false;
            state.token = null;
        },
    },
    extraReducers: (builder) => {
    },
});

export const { setCredentials, logout } = slice.actions;

export default slice.reducer;

export const selectCurrentUser = (state: RootState) => state.auth;

export const selectUserExists = (state: RootState) => state.auth.email !==
null && state.auth.token !== null;
```

orderSlice.ts

```
import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import { RootState } from './store';
import { CartItem, IProduct } from '../utils/types/types';

const slice = createSlice({
    name: 'cart',
    initialState: {
        cart: [],
    } as {
        cart: CartItem[]
    },
    reducers: {
        deleteFromCart: (
            state,
            { payload: product }:
                PayloadAction<IProduct>,
        ) => {
            state.cart = state.cart.filter((orderProduct: CartItem) =>
orderProduct.product.id !== product.id);
        },
        increaseProductCount: (
            state,
            { payload: product }:
                PayloadAction<IProduct>,
        ) => {
            state.cart = state.cart.map((orderProduct: CartItem) => {
                return orderProduct.product.id === product.id ?
                    { ...orderProduct, amount: orderProduct.amount + 1 } :
orderProduct;
            });
        },
        decreaseProductCount: (
            state,
            { payload: product }:
                PayloadAction<IProduct>,
        ) => {
            state.cart = state.cart.map((orderProduct: CartItem) => {
                return orderProduct.product.id === product.id ?
                    { ...orderProduct, amount: orderProduct.amount - 1 } :
orderProduct;
            }).filter((orderProduct: CartItem) => orderProduct.amount !== 0);
        },
```

```
        addProduct: (
            state,
            { payload: product }:
                PayloadAction<IProduct>,
        ) => {
            if (!state.cart.some((orderProduct: CartItem) =>
orderProduct.product.id === product.id)) {
                state.cart.push({ product, amount: 1 });
            }
        },
        clearCart(state) {
            state.cart = [];
        },
    },
});

export const { addProduct, decreaseProductCount, increaseProductCount,
deleteFromCart, clearCart } = slice.actions;

export default slice.reducer;

export const selectCart = (state: RootState) => state.cart.cart;
```

store.ts

```
import { combineReducers, configureStore } from '@reduxjs/toolkit';
import { setupListeners } from '@reduxjs/toolkit/query';
import { shopApi } from './services/services';
import auth from '../logic/authSlice';
import cart from '../logic/orderSlice';
import storage from 'redux-persist/lib/storage';
import { persistReducer } from 'redux-persist';
import { isRejectedWithValue } from '@reduxjs/toolkit';
import type { MiddlewareAPI, Middleware } from '@reduxjs/toolkit';


const persistConfig = {
    key: 'root',
    storage,
    blacklist: [shopApi.reducerPath],
};

export const rtkQueryErrorLogger: Middleware =
    (api: MiddlewareAPI) => (next) => (action) => {
        // RTK Query uses `createAsyncThunk` from redux-toolkit under the
hood, so we're able to utilize these matchers!
        if (isRejectedWithValue(action)) {
            console.warn('We got a rejected action!');
            //toast.warn({ title: 'Async error!', message:
action.error.data.message });
            //toast(`Some troubles occurred, try again later!`);
        }

        return next(action);
    };


const isNotPersistedReducers = {
    [shopApi.reducerPath]: shopApi.reducer,
};
const reducers = {
    // Add the generated reducer as a specific top-level slice
```

```
    ...isNotPersistedReducers,
    auth,
    cart,
};
const combinedReducers = combineReducers(reducers);
const persistedReducer = persistReducer(persistConfig, combinedReducers);

export const store = configureStore({
    reducer: persistedReducer, //persistedReducer, //persistedReducer,
    // Adding the api middleware enables caching, invalidation, polling,
    // and other useful features of `rtk-query`.
    middleware: (getDefaultMiddleware) =>
        getDefaultMiddleware({ serializableCheck: false
}).concat(shopApi.middleware, rtkQueryErrorLogger),
});

// optional, but required for refetchOnFocus/refetchOnReconnect behaviors
// see `setupListeners` docs - takes an optional callback as the 2nd arg for
customization
setupListeners(store.dispatch);

export type RootState = ReturnType<typeof store.getState>;
```

Routes.ts

```
export const Routes = {
    root: '/',
    cart: '/cart',
    orders: '/orders',
    confirm: '/confirm',
    login: '/login',
    register: '/register',
    singleProduct: '/products',
    categories: '/categories',
    profile: '/profile',
    reset: '/reset/password',
    auction: '/auction',
    auctionProduct: '/auction/product',
    createAuctionProduct: '/auction/product/create',
};
```