МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інтернет-магазин дизайнерських меблів»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Проценко О.Б.**

Студента групи ІН – 73 **Железняк С.О.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи ІН-73 спеціальності "Комп'ютерні науки" денної форми навчання Железняка Сергія Олеговича.

Тема: "**Інтернет-магазин дизайнерських меблів** "

Затверджена наказом по СумДУ

№_____ от _____2021 р.

Зміст пояснювальної записки: 1) огляд проблемної області та подібних рішень; 2) постановка задачі; 3) розробка інформаційного системи та програмної реалізації

Дата видачі завдання "_____"_____2021 р.

Керівник випускної роботи _____ Проценко О.Б.

Завдання приняв до виконання _____ Железняк С.О.

РЕФЕРАТ

Записка: 58 стор., 7 рис., 1 табл., 1 додаток, 7 джерел.

Об’єкт дослідження — інтернет-магазин

Мета роботи — інформаційна і програмна реалізація інтернет-магазину дизайнерських меблів

Методи дослідження — технології створення веб-додатка.

Результати — розроблена клієнтська та серверна частина веб-додатку для продажу меблів, який має усі необхідні функції для його користування. На мові програмування для клієнтської частина React та серверної NodeJS. В ході тестування проблем не було виявлено.

ІНТЕРНЕТ-МАГАЗИН, FULL-STACK, NODEJS, REACT

ЗМІСТ

ВСТУП

В сучасний час стрімко розвиваються засоби масової інформації. Ще не так багато років тому мало хто розумів, що таке Інтернет. Всесвітня павутина займає все більш міцні позиції в сучасному світі. Велика кількість соціальних груп, є одною з основних причин розвитку Інтернету, як засобу для спілкування. З появою Всесвітньої мережі, комунікація між користувачами вийшла за рамки одної країни, міста чи будинку. Кожна людина яка відчула всю незамінність та корисність мережі для своїх потреб приєднується до величезного ком'юніті споживачів інформації. Роль Інтернету в побуті, при просуванні нових послуг та технологій, стає більш зрозумілою. З розвитком технологій HTML, почало з'являтися велика кількість веб-сайтів, на абсолютно різну тематику – від сайтів великих корпорацій, до невеликих фірм. Застосування найновіших технологій, швидкий розвиток Інтернету та підтримка спілкування в комерційній діяльності у повсякденному житті зумовило виникнення нових економічних явищ, таких як електронна комерція. Електронна комерція - це велика сфера економіки, яка містить багато видів діяльності, наприклад, реклама, Інтернет-маркетинг, Інтернет-магазини і так далі. Кожний магазин сьогодні від маленьких вузькоспеціалізованих до крупних торгівельних мереж хочуть мати своє представництво в Інтернеті, а багато хто виключно здійснює свою діяльність в мережі. Інтернет відкриває широкі можливості для реклами та досліджень в маркетингу, надає нові канали збуту продукції. Для реалізації електронної комерції необхідно програмне забезпечення. Сьогодні існує велика кількість технологій за допомогою яких можна створювати ефективні та надійні веб-додатки, які відповідають самим сучасним вимогам, зручні у використанні, як зі сторони продавця, так і зі сторони споживача.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд проблемної області

З розвитком Frontend і Backend технологій електронна комерція має все більш широку масштабованість. Практично у кожної організації на сьогоднішній день є власний веб-сайт. Необхідна умова існування застосування сучасних інформаційних технологій дозволяє розширити можливості рекламної діяльності, тим самим залучаючи додаткових клієнтів. Інформаційні технології тісно пов’язані з інформаційними системами, які для неї є основним середовищем. ІС в діяльності більшості організацій і компаній розглядається як програмне забезпечення, яке реалізовує стратегію організації. Отримання прибутку є основною метою організацій, які займаються діяльністю торгівлі. Основними особливостями такої інформаційної системи є адаптивний та інтуїтивно-зрозумілий інтерфейс, вона повинна інформувати споживача про всіх його видах послуг і мати низьку вартість обслуговування. Під такого роду особливостей веб-технології є найбільш прийнятними. Веб-технології як правило включають в себе мови розмітки і програмування, системи і платформи CMS, бази даних та інші технології які дозволяють створювати веб-сайти, програми та магазини.

Існує безліч різних напрямків по використанню глобальної мережі в комерційних цілях: це може бути і створення сайту-візитки, де користувачі можуть ознайомитись з організацією і виробленою продукцією, це може бути і замовлення рекламних компаній з просування продукції в мережі, в соціальних мережах, зокрема. Однак в разі якщо мова іде про торгове підприємство, що реалізує товари, орієнтовані на широку аудиторію, найбільш доцільним вважається створення власного інтернет-магазину, оскільки даний тип електронного майданчика дозволяє у багато разів збільшити ефективність здійснюваної комерційної діяльності не витрачаючи при цьому грошових коштів

на оплату оренди додаткових площ, як це було б в звичайному магазині. Для сучасних підприємств даний напрямок є одним з пріоритетних векторів розвитку в силу ряду обставин: по-перше, глобальна мережа Інтернет сьогодні є масовим явищем з досить високою часткою його використання в загальній чисельності населення, по-друге, використання електронних майданчиків дозволяє автоматизувати процес продажів і здійснювати реалізацію продукції в будь-який час доби, по-третє, використання мережі Інтернет дозволяє виходити на інші ринки інших регіонів, розширюючи можливості підприємства. Перед початком розробки необхідно сформувати набір даних, на які спиратиметься логіка роботи інтернет-магазину:

1) Вхідні інформаційні потоки:
    a. для адміністратора – кількість товару на складі, асортимент товару у продажу
    б. для клієнта – асортимент товару в каталозі, ціни на товари, новини і пропозиції додаткових послуг, контактна інформація фірми
2) Вихідні інформаційні потоки:
    a. для адміністратора – замовлення від клієнтів, оборот грошових коштів, документація про продажу товару, стан складу, статистика продажів і замовлень
    б. для клієнта – статистика покупок, рахунки до оплати

Наявність поділу між вхідними та вихідними потоками, а також розподіл користувачів магазину на дві різні категорії накладають певні і жорсткі розмежування для різних частин магазину. Те, що може виконувати адміністратор, має бути заборонено і недоступно покупцям. Частина магазину, присвячена покупцеві, повинна бути якомога більш зручною і зрозумілою в застосуванні. Покупцеві необхідно надавати вибір товару на його смак і колір, а також зручність заповнення купівельної корзини та збереження зробленого замовлення. Щоб оцінити загальні витрати, покупцеві треба надавати історію

замовлень. Створення Інтернет-магазину дає фірмі і клієнтам ряд переваг: залучення нових клієнтів за рахунок загальнодоступності, більш просте і швидке оформлення замовлень (зниження кількості менеджерів по обробці замовлень), збільшення товарообігу, а відповідно і прибутку, дешевша реклама через інтернет, розвиток і популярність фірми, з огляду на великі темпи розвитку торгівлі в Інтернеті.

1.2 Огляд подібних рішень

На сьогоднішній день в мережі Інтернет існує дуже велика кількість інтернет-магазинів і з кожним днем їх кількість збільшується. Перед початком розробки інтернет-магазину був проаналізований ринок схожих рішень. За основу прикладів для аналізу були обрані сайти які мають самий високий рейтинг на думку покупців, це сайти такі як:

1. mebelboom.com.ua
2. shopomag.com.ua
3. domoteka.com.ua

Перший магазин який буде розглянутий представлений на рисунку 1.1

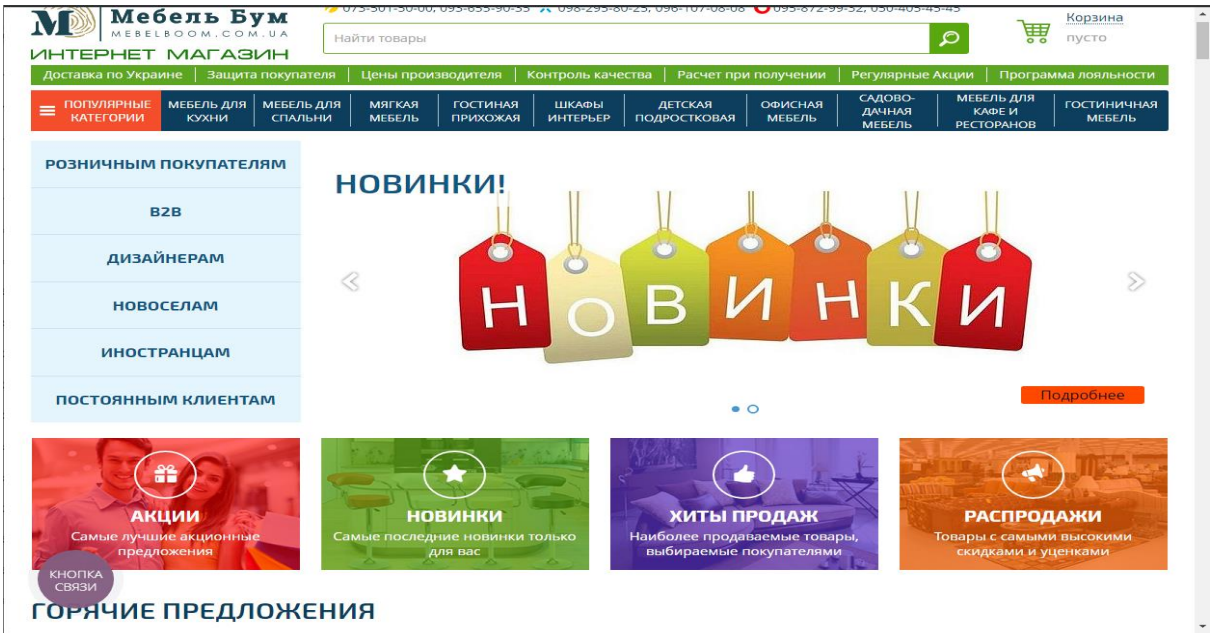

Рисунок 1.1 – Сайт «Мебель бум»

Наступним сайтом розглянемо інтернет-магазин «Шопомаг»

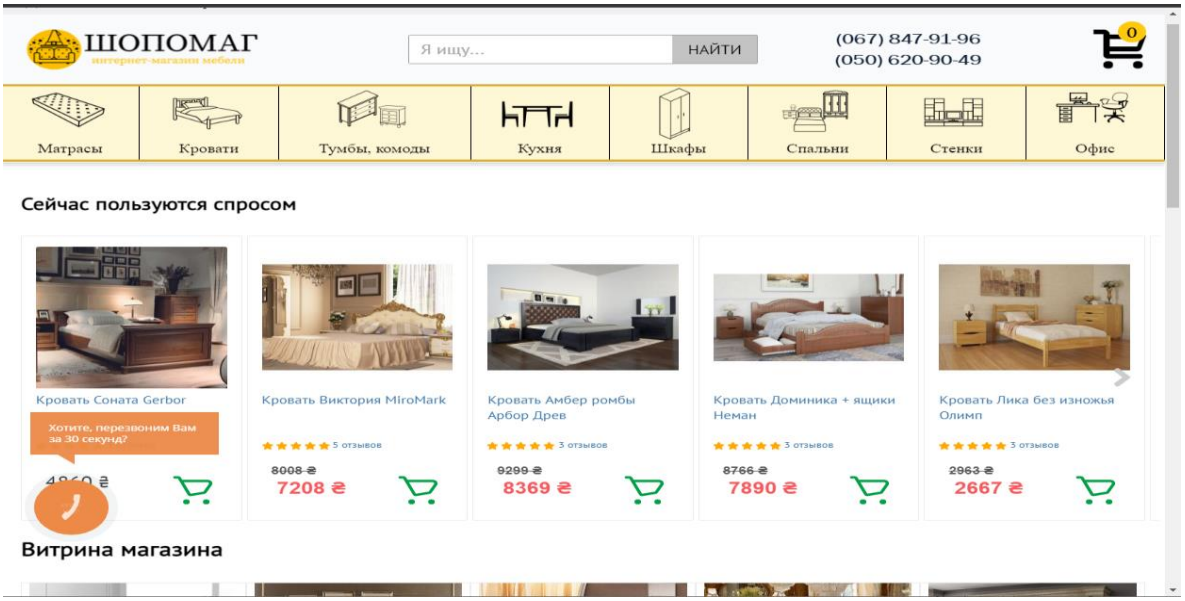

Рисунок 1.2 – Сайт «Шопомаг»

І останній сайт який має самий високий рейтинг представлений на рисунку 1.3

Рисунок 1.3 – Сайт «Домотека»

Представимо аналіз сайтів у вигляді таблиці 1.1

Таблиця 1.1 – Порівняльний аналіз аналогів інтернет-магазинів меблів

Критерії	Сайт «Mebelboom»	Сайт «Shopomag»	Сайт «Domoteka»
Дизайн сайту	+	+	-
Фільтр товару	+	+	+
Зручність у використанні	+	+	-
Адаптивність	-	+	-
Швидкість загрузки	+	+	-
Посилання на соціальні мережі	+	-	+

Згідно з результатами аналізу, які представлені у таблиці, можна зробити деякі висновки:

1. Для того щоб ресурс мав успіх, варто докласти багато зусиль в процесі його виготовлення. Одним із принципових моментів, які варто враховувати в ході розробки, є дизайн. Створення візуального оформлення ресурсу вважається одним з головних етапів при формуванні сайту. Саме від нього залежить те, як його сприймуть користувачі і чи захочуть продовжити роботу на ньому. У веб-сайта «Домотека» відсутній дизайн або він дуже застарілий. Це може відштовхнути потенційного покупця.

2. Крім естетичної привабливості, візуальне оформлення ресурсу несе в собі функціональну особливість - юзабіліті. Дана опція застосовується для максимально точного аналізу сайту, його ефективності та доступності, з точки зору відвідувачів. Якісно виконане юзабіліті зможе збільшити дохід, за рахунок залучення великої кількості клієнтів. На сайті «Домотека» складно зорієнтуватися. Всі елементи які знаходяться на сайті не мають логічної структури. Якщо обрати якусь класифікацію меблів, то можна побачити інформацію у вигляді dropdown меню і одразу в тому ж місці сайт пропонує ознайомитись з другими типами меблів в такому ж стилі. Контент такого роду сприймати дуже складно.
3. Сайти «Мебельбум» та «Домотека» створені лише для десктопної версії. Одна із головних вимог кожного веб-сайту є його адаптивність. За рахунок того, що ресурс однаково добре демонструється на смартфонах, планшетах та екранах комп'ютерів, він гарантує повне охоплення аудиторії. На сьогоднішній день близько 50% відвідувачів користуються сайтом зі смартфонів чи планшетів. Мобільна аудиторія постійно зростає, і ігнорувати її потреби не можна. Не адаптований веб-сайт, неминуче втрачає частину мобільних користувачів, звідки слідує додаткові відмови. Одним із не менш важливих фактів є те, що такий ресурс програє в ранжируванні втрачаючи таким чином охоплення аудиторії та потенційних клієнтів в бізнесі.
4. Сайт «Домотека» має дужу низьку швидкість загрузки сторінок навіть при швидкій швидкості інтернету. Такий критерій істотно впливає на конверсію, показник відмов, відвідуваність і інші важливі для бізнесу KPI. Тому що при інших рівних умовах людина вибере той сайт, який швидше завантажується і не змушує його чекати.

1.3 Постановка задачі

Метою дипломної роботи є створення інтернет-магазину лімітованих дизайнерських меблів. Веб-додаток повинен містити візуальну(Frontent) та серверну частину(Backend) і слідувати наступним вимогам:

- Дизайн сайту повинен бути створений на основі макету, слідувати сучасним стандартам і бути зручним для користувача
- Сайт повинен містити модулі реєстрації, авторизації та можливість додати товар до корзини
- Мати систему адміністрування завдяки якої адміністратор має можливість редагувати інформацію
- Пропонування спеціальних пропозицій та акцій
- Містити основне навігаційне меню в яке входить
  - Головна сторінка – дає представлення користувачу о сайті
  - Про нас – знайомить користувача з магазином
  - Команда – надає інформацію про персонал
  - Каталог – має весь асортимент товару
- На сайті повинні бути текстові, графічні та відео матеріали які надають користувачу всю інформацію про мебель
- Можливість обирати тип та бренд меблів

2 ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

2.1 Вибір стека технологій

Перед початком кожного проекту проходить безліч процесів планування і прийняття рішень. Ця фаза планування має вирішальне значення і може мати значний вплив на майбутнє проекту. Вибір правильного стека технологій може вплинути на час розробки, якість програми, а також на масштабованість, тому так важливо прийняти правильне рішення, навіть якщо потрібно витратити більше часу, аналізуючи всі плюси і мінуси доступних рішень. Стек технологій - це набір інструментів, які застосовуються при роботі в проектах і включають мови програмування, фреймворки, системи управління базами даних і т. д. Від обраного розробником стека технологій залежать продуктивність роботи, вимоги до апаратних ресурсів, надійність роботи програмного забезпечення.

Рано чи пізно кожен веб-розробник замислюється про те, як йому спростити і прискорити процес створення веб-додатку. Вони всіма силами намагаються уникнути виконання однієї і тієї ж роботи, автоматизуючи її та використовуючи вже готові рішення. Таким чином з’явилися бібліотеки та фреймворки. Починаючи розробку нової інформаційної системи, розробники шукають бібліотеки і фреймворки, які вирішують типові завдання, що виникають в процесі створення клієнт-серверних програм. Фреймворки - це програмні продукти, які спрощують створення і підтримку технічно складних або навантажених проектів. Фреймворк, як правило, містить тільки базові програмні модулі, а всі специфічні для проекту компоненти реалізуються розробником на їх основі. Тим самим досягається не тільки висока швидкість розробки, а й велика продуктивність і надійність рішень. Одним з головних переваг фреймворків є те, що він визначає об’єднану структуру для побудованих на його базі додатків.

Тому додатки на фреймворках значно простіше супроводжувати і допрацьовувати, так як стандартизована структура організації компонентів зрозуміла всім розробникам на цій платформі і не потрібно довго розбиратися в архітектурі, щоб зрозуміти принцип роботи програми або знайти місце реалізації того чи іншого функціоналу.

Кожний веб-додаток ділиться на дві частини: серверна(Beckend) та клієнтська(Frontend). Серверна веб-розробка включає в себе створення «мозку» додатка, який бере на себе весь функціонал. Серверної частини програми не видно кінцевому користувачеві, вона забезпечує дані для клієнтської частини. Для її виконання був обраний наступний стек: NodeJS, фреймворк Express, СУБД MySQL, ORM Sequelize.

NodeJS – це середа виконання коду на JavaScript, яка націлена в першу чергу на розробку серверної частини веб-додатка та побудована на основі движка JavaScript Chrome V8. Цей движок використовує JavaScript код і перетворює його в машинний код, який вже розуміє комп’ютер. По-перше віртуальна машина V8 виконує JavaScript код дуже швидко. По-друге підтримує практично всі можливості сучасного JS та самого сучасного стандарту який зараз прийнятий. По-третє дуже економно витрачає пам’ять. Тому NodeJS має таку популярність і успіх серед його альтернатив. При реалізації серверної частини вибір падає на NodeJS із кілька ряду причин:

1. Мова програмування. Використання однієї і тієї ж мови і на клієнті і на сервері
2. Вирішує основні задачі для Web. NodeJS відмінно працює із самими поширеними базами даних. Використовує різного роду протоколи: http, https. В ньому міститься відмінно пророблена колекція модулів
3. Багато з’єднань і задач одночасно. Тобто NodeJS добре себе веде там, де потрібний великий потік користувачів

4. Зручний менеджер пакетів. Пакетна система npm у NodeJS має саму велику екосистему бібліотек з відкритим вихідним кодом. NPM(менеджер пакетів Node) містить пакети які можна використовувати в проектах для того, щоб зробити розробку більш бистрою та більш ефективною.[1]

Express – це фреймворк працюючий всередині середовища виконання NodeJS, який надає гранично простий, але потужний інструмент для створення веб-додатків. Він використовує модуль http, але разом з цим надає ряд готових абстракцій, які спрощують створення сервера та серверної логіки. Основне призначення Express маршрутизація і проміжна обробка з мінімальною власною функціональністю. Додаток являє собою серію викликів функцій тимчасової обробки(middleware).[2]

Більшість сайтів, а зокрема інтернет-магазини, не можуть функціонувати без бази даних. Використання баз даних у веб-додатках дозволяє автоматично оновлювати сайт, розпізнавати користувача та відстежувати дані. Вся інформація зберігається, обробляється та вилучається з баз даних. В базах даних можна добре зберігати структуровану інформацію наприклад таку як:

1. Список користувачів
2. Інформацію о клієнтах
3. Статичні зображення або текст
4. Опис товару
5. Список товару

Бази даних включають в себе сукупність мовних та програмних засобів які в цілому називаються «Система управління базами даних»(СУБД). СУБД - це набір програм, що дозволяє контролювати, організовувати та адмініструвати бази даних. В якості СУБД була обрана MySQL. Вона вважається однією з найпоширеніших. MySQL - реляційна СУБД, головними плюсами якої є її швидкість і гнучкість, яка забезпечена підтримкою великої кількості різних типів

таблиць. Крім того, це надійна безкоштовна система з простим інтерфейсом і можливістю синхронізації з іншими базами даних.

Sequelize – це ORM-бібліотека для додатків на Node.js, яка здійснює зіставлення таблиць в базі даних і відносин між ними з класами. При використанні Sequelize ми можемо не писати SQL-запити, а працювати з даними як зі звичайними об'єктами.[3]

Frontend - це публічна частина web-додатків, з якої користувач може взаємодіяти і контактувати напряму. У Frontend входить відображення функціональних завдань, призначеного для користувальницького інтерфейсу, що виконуються на стороні клієнта, а також обробка запитів користувачів. Frontent - це все, що користувачі можуть бачити на екрані.[] Найважливішими елементами технологічного стека на стороні клієнта є:

1. HTML, який відповідає за відображення вмісту всього контенту який є на сайті через браузер
2. CSS стилізує контент
3. JavaScript відповідає за інтерактивну частину веб-додатка(взаємодія користувача з сайтом)

Ці основоположні технології можна використовувати з корисними фреймворками, такими як Bootstrap або React.js. Сучасні фреймворки зовнішнього інтерфейсу можуть управляти всіма трьома елементами призначеного для користувальницького інтерфейсу. У них є HTML-шаблони, стилі та інтерактивні функції. Для виконання клієнтської частини веб-додатка був обраний наступний стек технологій: HTML, Sass, Bootstrap, ReactJS, Axios, React-router-dom, MobX .

HTML – це стандартизована мова розмітки документів в інтернеті. В першу чергу, необхідно зрозуміти, що HTML - основа кожної веб-сторінки, незалежно

від складності сайту або кількості задіяних технологій. Це важливий навик для будь-якого веб-професіонала і відправна точка для всіх, хто має відношення до створення або редагування контенту в Інтернеті. Всупереч поширеній помилці, HTML - це не мова програмування. Вона використовується для того, щоб всі елементи на сторінці (тексти, малюнки, таблиці ...) були розташовані правильно; за його читання відповідають спеціальні програми, які всім знайомі – браузери.[4]

Sass – це свого роду розширення, створене для спрощення таблиць стилів (CSS). CSS – це формальна мова, яка визначає зовнішній вигляд документа, написаного з використанням HTML. Sass має особливий синтаксис який значно відрізняється від CSS. Цей синтаксис коротше, в ньому відсутні дужки і крапки з комою. Відступи мають логічне значення, тому вкрай важливо стежити за ними – неправильний відступ може зламати таблицю стилів. Використання Sass дає ряд переваг, які значно економлять час і сили. Sass дає можливість призначати змінні. Змінні зручно використовувати, якщо одне значення застосовується кілька разів. В цьому випадку за допомогою змінної всі необхідні значення можна задати на початку коду і далі просто посилатися на них. І якщо буде необхідно змінити значення змінної, то воно зміниться і у всіх інших місцях, де вона проставлена. Sass дає можливість вкладати правила CSS один в одного. Завдяки цьому стає набагато простіше редагувати стилі. Вкладені правила потрібні не тільки для мінімізації коду, але і для структурування коду. Додатки дозволяють слідувати прекрасного правила не дублювати код. Замість того щоб копіювати і множити шматки однакового коду, Sass пропонує зберегти його в змінну, а потім використовувати там, де це необхідно за допомогою міксинів (mixin).[5]

Bootstrap є найпопулярнішим фреймворком і має саме велике співтовариство, тому розробка за допомогою цього фреймворка буде завжди актуальною. Bootstrap дозволяє верстати сайти в кілька разів швидше, ніж це

можна виконати на «чистому» CSS і JavaScript. Крім цього, його популярність ще обумовлена доступністю. Вона полягає в тому, що на ньому навіть розробник початківець може верстати досить якісні макети, які важко було б виконати без глибоких знань веб-технологій і достатньої практики. Фреймворк Bootstrap це набір CSS стилів і JavaScript файлів. Для того щоб його використовувати ці файли необхідно підключити до сторінки. Після цього стануть доступні інструменти даного фреймворка: «колоночна» система (сітка Bootstrap), класи, скидання стандартних браузерних стилів, плагіни та компоненти. Але Bootstrap – це не просто набір готових інструментів, а добре спроектований фронтенд фреймворк, який досить просто можна налаштувати під себе за допомогою редагування Sass змінних і використання міксинів. Переваги Bootstrap при його використанні для frontend розробки сайтів:

1. Висока швидкість створення якісної адаптивної верстки(завдяки «колоночної» системи можна дуже легко створити веб-додаток який буде підлаштовуватись під різні пристрої)
2. Кроссбраузерність і кроссплатформенність [6]
3. Можливість налаштування під свій проект, досягається це за допомогою зміни Sass змінних і використання міксинів [6]
4. Низький поріг входження. Для роботи з фреймворком не обов'язково мати «глибокі» знання з HTML, CSS, JavaScript і jQuery (досить знати тільки основи цих технологій) [6]
5. Наявність величезного співтовариства і навчальних матеріалів. При бажанні це допоможе не тільки добре розібратися в фреймворку, але і знайти відповіді практично на будь-які питання [6]

Незважаючи на всі переваги цей фреймворк як і інші має ряд недоліків. Його використання не підійде для створення проектів з унікальним дизайном. Проект буде мати більш великий розмір css та javascript-файлів ніж вони вийшли,

якби все створювалось самостійно. Це пов’язано з тим, що фрейворк має велику кількість напрацювань, а в проекті може бути використана лише його частина. Таким чином це може вплинути на швидкість завантаження сторінки. Кожна додаткова секунда затримки загрузки сайту підвищує ймовірність переходу користувача до конкурентів. Вирішення цих недоліків можна, якщо виконати самостійну збірку проекту із вихідних кодів та включити в неї тільки ті компоненти, які потрібно. В такому випадку цього фреймворка буде менше і займати місце він буде менше.

ReactJS – це JavaScript бібліотека для створення користувацьких інтерфейсів. Вона дозволяє збирати складний UI з маленьких ізольованих шматочків коду, званих «компонентами». Концепція ReactJS полягає в тому, що можна створювати односторінкові додатки (SPA) які поводяться вкрай чуйне. Вони взаємодіють з користувачем і реагують на його дії зміною зовнішнього вигляду. Основною задачею ReactJS зміна UI без перезавантаження сторінки. ReactJS має багату екосистему, яка складається з самого ReactJS і бібліотеки React-DOM для управління об'єктами DOM. Далі React-Router, який відповідає за маршрутизацію. Крім того, він поставляється з JSX, який є розширенням синтаксису для Javascript для створення шаблонів в ReactJS. Щоб зробити розробку простіше і швидше. JSX має синтаксис HTML розмітки і був створений для більш зручної ієрархії коду. [7]

Axios - це широко відома JavaScript-бібліотека. Вона являє собою HTTP-клієнт, заснований на промісах і призначений для браузерів і для Node.js. У Axios є підтримка запитів, отримання відповідей від сервера, їх трансформація і автоматична конвертація в JSON.[8]

Якщо все що робили раніше було розміщено на одній сторінці і тому в роутері сенсу ніякого не було, то тепер є можливість за допомогою роутера розбивати інформацію на нашій сторінці на деякі логічні шматки. У React є своя

система маршрутизації, яка дозволяє зіставляти запити до додатка з певними компонентами. Ключовою ланкою в роботі маршрутизації є модуль react-router, який містить основний функціонал по роботі з маршрутизацією. Однак робота в браузері, вимагає використовувати модуль react-router-dom. React-router справді має користь і широко використовується в додатках React більше з боку серверної частини ніж з боку клієнтської. Точніше він зазвичай використовується в середовищі NodeJS. React-router дозволяє визначати динамічні URL але відповідно філософії "Single Page Application" (односторінкового додаток). Цей модуль надає два компонента це <BrowserRouter> та <HashRouter>. Ці компоненти відрізняються у вигляді URL, які вони будуть створювати і синхронізувати. <BrowserRouter> більш широко використовується, він використовує History API наявний в HTML5 для моніторингу історії роутера.[9]

У справжніх проектах розробники отримують дані від сервера або користувальницького введення, форматують, валідують, нормалізують і виробляють інші операції над ними. Все це прийнято вважати бізнес логікою і повинне існувати в Model. Так як React – це лише частина MVC (View) моделі, для створення користувацьких інтерфейсів, то буде потрібно ще щось для бізнес логіки. MobX - це автономна бібліотека, для управління фронтенд-станом додатка. MobX дозволяє реалізувати ланцюжок: «Виконання дії» → «Зміна стану» → «Зміна уявлення».[6] При цьому зміни відбуваються атомарно і автоматично - в результаті гарантується, що не буде моменту, коли стан буде несумісний. React в поєднанні з MobX дозволяє досягти постійної відповідності внутрішнього стану з візуальним поданням інтерфейсу.[10]

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка моделі інформаційної системи

При створенні серверної частини програми необхідно враховувати, як інтерфейс буде взаємодіяти з серверної частиною. Однак більш важливим є побудова і дизайн бази даних. Схема бази даних є абстрактним дизайном, який представляє собою зберігання даних в базі даних. Він описує як організацію даних, так і відносини між таблицями в даній базі даних. Розробники заздалегідь планують схему бази даних, щоб знати, які компоненти необхідні і як вони будуть з'єднуватися один з одним. Схема бази даних – це план або архітектура того, як будуть виглядати дані. Він не містить самих даних, а замість цього описує форму даних і те, як вони можуть бути пов'язані з іншими таблицями або моделями. Згідно з вимогами проекту побудуємо діаграму схеми бази даних.

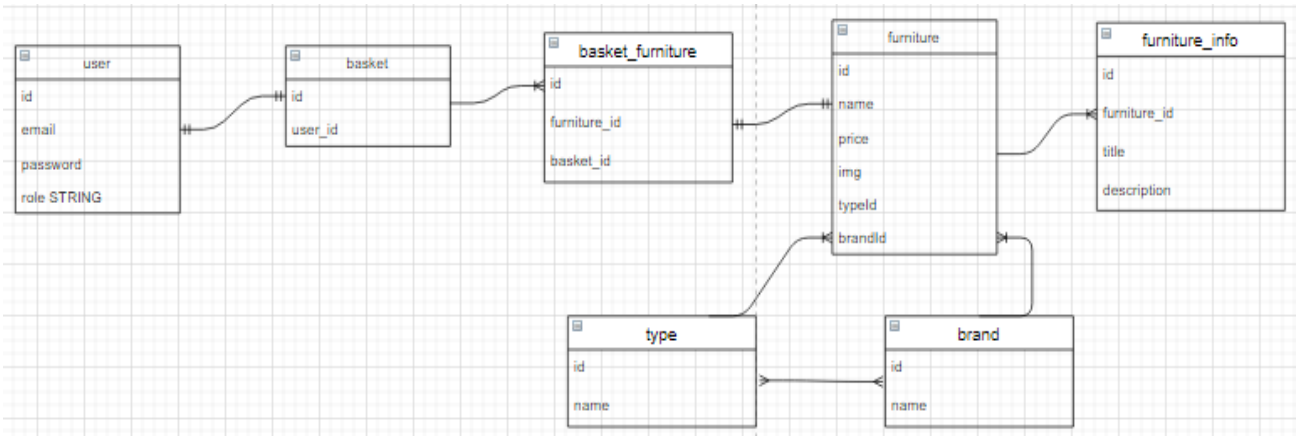

Рисунок 3.1 – Схема бази даних

Перша сутність – це користувач(user). Вона має id, email, password, role(тобто це користувач або адміністратор). Наступна сутність – це корзина(basket) яка має id та зовнішній ключ на користувача user_id для того, щоб розуміти кому вона належить. Зв’язок між цими двома таблицями буде 1 до 1. Одному користувачу може належить лише одна корзина в той час як одна корзина може належить лише одному користувачу.

Наступна сутність – це конкретна мебель(furniture). Вона має id, name, price, img(зображення кожного елемента меблів), та два зовнішніх ключа typeId(тип меблів), brandId(виробник). Типи та бренди мають свої окремі сутності з полями id та name. Зв’язок між типам, брендами та сутністю furniture буде 1 до багатьох. Одному типу та бренду може належить декілька меблів. Зв’язок між типами та брендами теж повинен бути. Оскільки якісь бренди можуть не випускати якийсь тип меблів. Зв’язок між type і brand буде багато до багатьох. Один бренд може належить декільком типів і один тип може належить декільком брендам.

Наступна сутність – це інформація мебелі(furniture_info). Вона має id, зовнішній ключ на мебель furniture_id, title, description. Кожна мебель має якусь індивідуальну інформацію(наприклад у дивана можна описати наповнювач, а у стола ні). Зв’язок між furniture буде 1 до багатьох. Оскільки у одної мебелі може бути багато характеристик.

Наступна сутність – це корзина товарів(basket_furniture). Це буде проміжна таблиця між корзиною та товарами. Вона має id та зовнішні ключі furniture_id, basket_id. Зв’язок між корзиною та basket_furniture буде 1 до багатьох. Оскільки одна корзина може містити в собі велику кількість товарів. А зв’язок між basket_furniture і furniture буде 1 до 1. Оскільки товар який знаходиться в корзині посилається на якийсь товар.

Веб-додаток містить в собі два види користувачів. Адміністратор може керувати даними в базі даних, а користувач взаємодіяти за даними які є на сторінках.

3.2 Підготовка до проекту

В самому початку проекту необхідно зробити ініцілізацію в терміналі за допомогою команди `npm init -y`. Першим етапом підготовки буде встановлення необхідних залежностей в проект. В терміналі за допомогою npm встановлюємо фреймворк express, sequelize, mysql, jsonwebtoken(для генерації токену), bcrypt(для того щоб хешировать паролі і не зберігати їх у відкритому вигляді). Щоб була можливість звертатися з браузера до серверу встановлюємо cors. Для використання змінних оточення встановлюємо dotenv. Також як залежності для розробки встановимо nodemon з ключом D для того щоб при кожній зміні в коді не перезавантажувати сервер, nodemon подбає про це сам. Завершальним етапом напишемо скрипт який буде запускати додаток в режимі розробки. Для цього в файлі package.json полю script прописуємо команду

```
"scripts": {
  "dev": "nodemon index"
}
```

Тепер за допомогою команди `npm run dev` буде здійснюватися запуск сервера

3.3 Програмна реалізація

Головним і кореневим файлом в серверній частині веб-додатка є файл index.js. З нього буде починатись запуск. В самому початку головного файлу імпортуємо необхідні модулі та файли які використовувались у всьому проекті за допомогою required

```
require('dotenv').config()
const express = require('express')
const path = require('path')
const router = require('./routers/index')
const sequelize = require('./db')
const models = require('./models/models')
const errorHandler = require('./middleware/ErrorHandlingMiddleware')
const cors = require('cors')
```

```
const fileUpload = require('express-fileupload')
```

Вказуємо порт на якому буде працювати сервер. Статично вказувати порт не є хорошою практикою. Тому вся конфігурація проекту винесена в змінні оточення .env. Якщо змінна не задана, тоді встановлене значення за замовчуванням 5000

```
const PORT = process.env.PORT || 5000
```

Створюємо об’єкт app викликавши функцію express. З нього буде починатися запуск додатка

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

Створимо асинхронну функцію start (всі операції з базами даних є асинхронними). Весь код обгорнемо в блок try catch для того щоб відловлювати помилки і веб-додаток не падав. У об’єкта sequelize який був імпортований із файла db.js визиваємо метод authenticate. Завдяки їй буде здійснюватися підключення до бази даних. Також у цього об’єкта визиваємо метод sync. Цей метод буде звіряти стан бази даних зі схемою баз даних. У об’єкта app визиваємо функцію listen куди першим параметром передаємо порт, а другим callback функцію яка спрацює при успішному запуску сервера. В консолі виведемо повідомлення про те, на якому порту стартував сервер.

```
async function start(){
    try{
        await sequelize.authenticate()
        await sequelize.sync()
        app.listen(PORT, () => console.log(`Server started on port
${PORT}`))
    } catch (e){
        console.log(e)
```

```
    }
}
start()
```

Наступним не менш важливим файлом є db.js. В ньому буде міститься вся конфігурація підключення до бази даних. Імпортуємо Sequelize та робимо дистуктуризацію оскільки модуль великий, а нам потрібен лише клас. В конструкторі вказуємо змінні із файла .env в яких зберігаються значення із бази даних (тобто назва бази даних, ім’я користувача під яким підключаємося, пароль) та об’єкт де вказуємо діалект, хост та порт

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

Файл ApiErrors.js містить в собі код який відповідає за обробку помилок. Клас ApiErrors розширює клас Error. Задаємо три статичні функції(3 види помилок) в яких повертаємо новий об’єкт в якому вказуємо завдяки конструктору статус помилки та повідомлення яке будемо отримувати параметром.

```
class ApiErrors extends Error{
    constructor(status, messages) {
        super();
        this.status = status
        this.message=messages
    }
    static badRequest(messages){
        return new ApiErrors(404, messages)
    }
    static internal(messages){
        return new ApiErrors(500, messages)
    }
    static forbidden(messages){
```

```
            return new ApiErrors(403, messages)
        }
    }
    module.exports = ApiErrors
```

В файлі UserController.js знаходиться вся логіка яка належить користувачу. Авторизація буде відбуватися по JWT токену. Токен доступу – це рядок розділений точками на три частини. В центральній частині рядка(payload) будуть сховані дані які прийшли від користувача(id, email та роль). Дані такого роду не мають ніякої секретної інформації, але перевірити валідний токен чи ні, можна лише за допомогою секретного ключа який оголошений на сервері. Коли користувач здійснює авторизацію на сайті в першу чергу перевіряємо чи існує такий користувач в базі. Якщо авторизація сталася успішною, тоді генеруємо JWT токен і відправляємо його на клієнт. Після отримання токену на клієнті він зберігається в куках або локалсторедж. Імпортуємо всі необхідні файли за допомогою require

```
    const ApiError = require('../errors/ApiErrors')
    const jwt = require('jsonwebtoken')
    const bcrypt = require('bcryptjs')
    const {User, Basket} = require('../models/models')
```

Функція generateJwt генерує токен доступу. Першим параметром метод sign приймає об’єкт payload, де будуть зберігатись дані. Другим параметром передається секретний ключ який знаходиться в змінних оточення .env. Третій параметр – це опція яка відповідає скільки існує токен

```
    const generateJwt = (id, email, role) => {
        return jwt.sign({id, email ,role},
process.env.JWT_SECRET,{expiresIn: '30d'})
    }

    class UserController {
        async registration (req, res, next){
```

Вилучаємо email, password та role з тілу запита скориставшись диструктуризацією

```
            const {email, password, role} = req.body
             if(!email || !password){
                 return next(ApiError.badRequest('Некорректный email или
password'))
             }
```

Наступним етапом перевіряємо чи існує вже такий користувач в базі даних. Створюємо зміну candidate (оскільки невідомо буде створений користувач чи ні) яка зберігає в собі значення email в разі успіху. У об’єкта User викликаємо метод findOne куди передаємо email. По такому критерію буде здійснюватися пошук. Якщо база даних щось повернула в такому випадку повертаємо помилку оскільки два користувача з однаковим email не може бути.

```
            const candidate = await User.findOne({where: {email}})
             if(candidate){
                 return next(ApiError.badRequest('Пользователь с таким
email уже существует'))
             }
```

Якщо умова не виконалась тоді виконуємо хешування паролю, створення нового користувача і корзини. Зберігати пароль в відкритому вигляді – погана практика. Тому скористаємося методом hash який приймає в якості параметрів сам пароль і степінь його хешуювання.

```
             const hashPassword = await bcrypt.hash(password, 5)
             const user = await User.create({email, password:
hashPassword, role})
             const basket = await Basket.create({userId: user.id})
```

Змінній token присвоюємо згенерований токен і повертаємо його на клієнт

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

Вся логіка логіну обернена в блок try catch оскільки можна отримати потенційні помилки які треба обробити

```
    try{
```

Вилучаємо email та password з тілу запита скориставшись диструктуризацією. Знаходимо користувача в базі даних скориставшись методом findOne. Якщо користувача не було знайдено, повертаємо internal помилку з повідомлення, що такого користувача не існує

```
            const {email, password} = req.body
             const user = await User.findOne({where: {email}})
             if(!user) {
                      return next(ApiError.internal('Пользователь не
найден'))
             }
```

Якщо користувача знайдено перевіряємо паролі які ввів користувач з паролем який знаходиться в базі даних. За допомогою метода compareSync разхешовуємо пароль і порівнюємо. В випадку якщо паролі не збігаються повертаємо помилку

```
            let comparePassword = bcrypt.compareSync(password,
user.password)
             if(!comparePassword){
                 return next(ApiError.internal('Указан неверный
пароль'))
             }
```

У разі успішної авторизації генеруємо токен і повертаємо його на клієнт

```
                const token = generateJwt(user.id, user.email, user.role)
                 return res.json({token})
            } catch (e) {
                console.log(e)
            }
        }

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

За допомогою програми Postman можна переконатися в тому, що код працює і виконує всі функції вірно. Обираємо метод POST, переходимо по URL registration і в форматі json заповнюємо відповідні поля. Якщо повернувся токен, значить користувач був зареєстрований

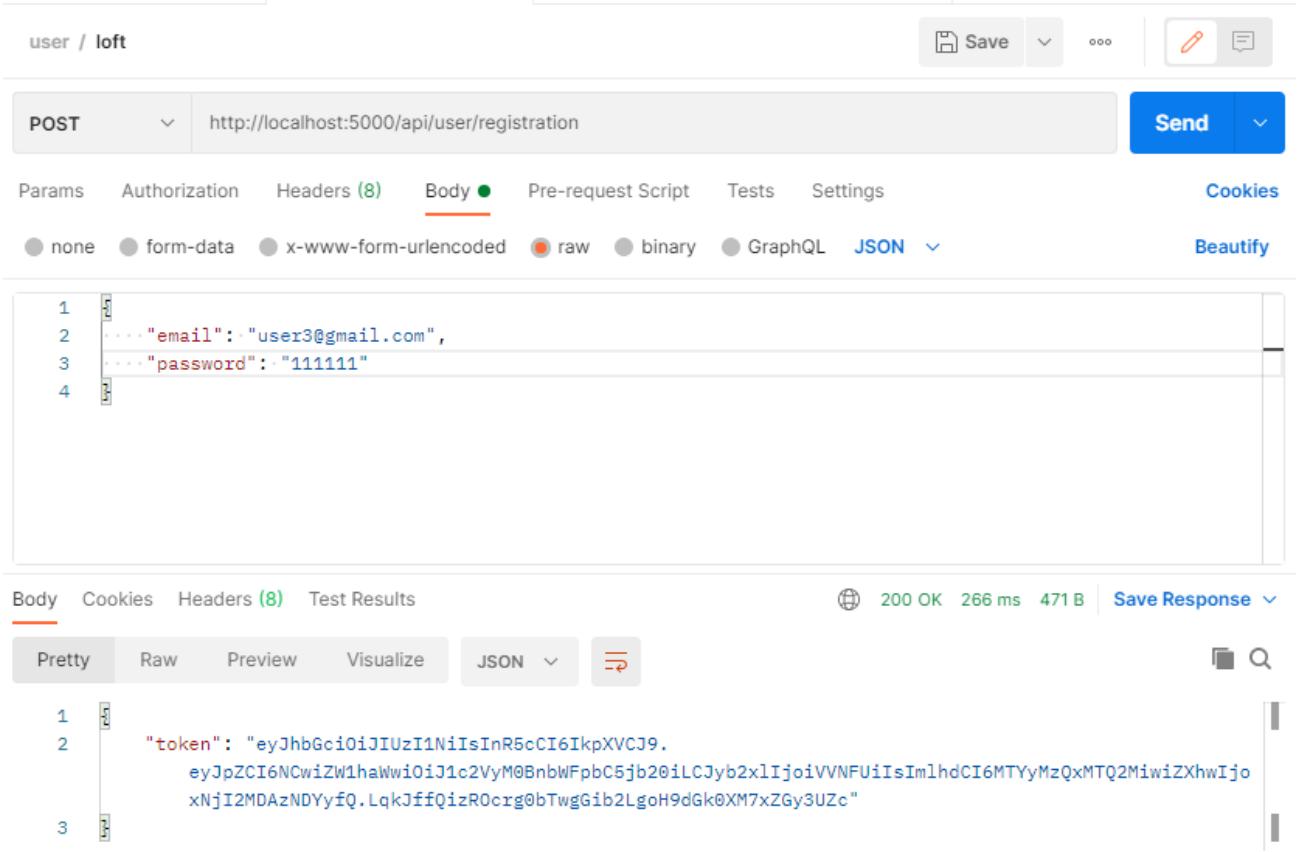

Рисунок 3.1 – Результат реєстрації користувача

Зробимо перевірку логіна. Обираємо метод POST, переходимо по URL login і в форматі json заповнюємо відповідні поля. В полях введемо дані користувача якого тільки що зареєстрували і отримуємо на клієнт токен.

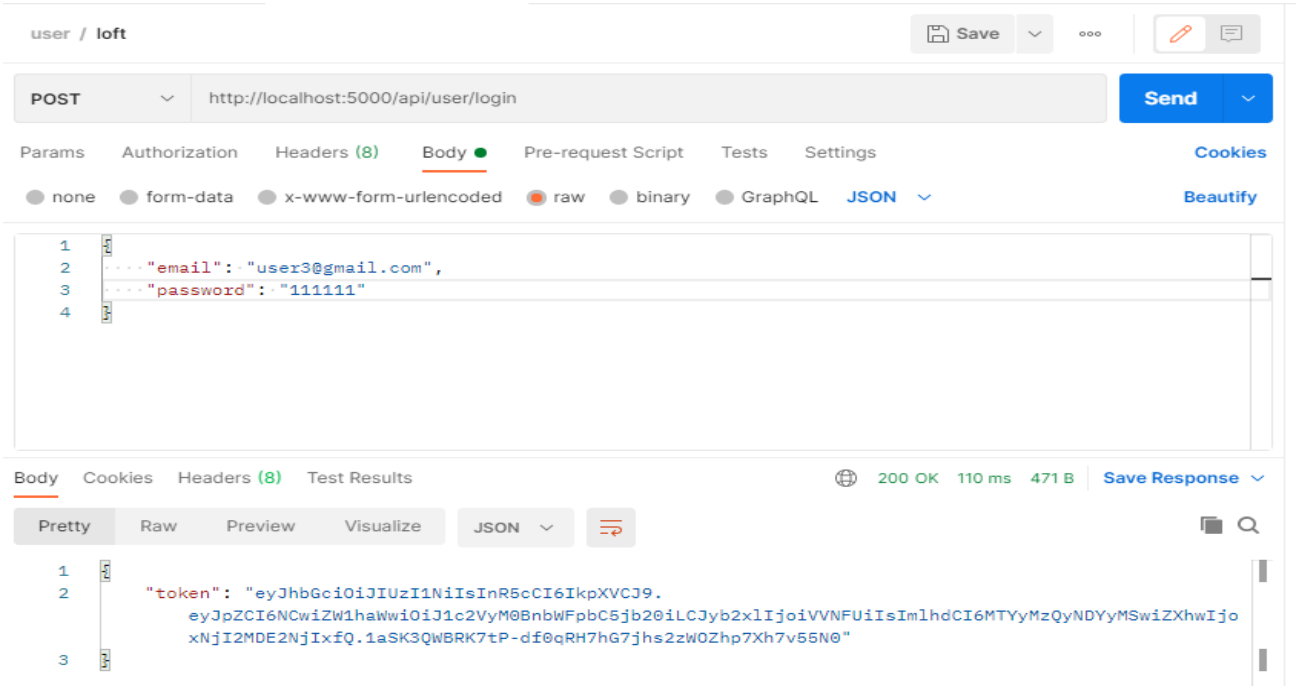

Рисунок 3.2 – Результат успішної авторизації

Спробуємо ввести некоректні дані в поля вводу. Отримаємо помилку

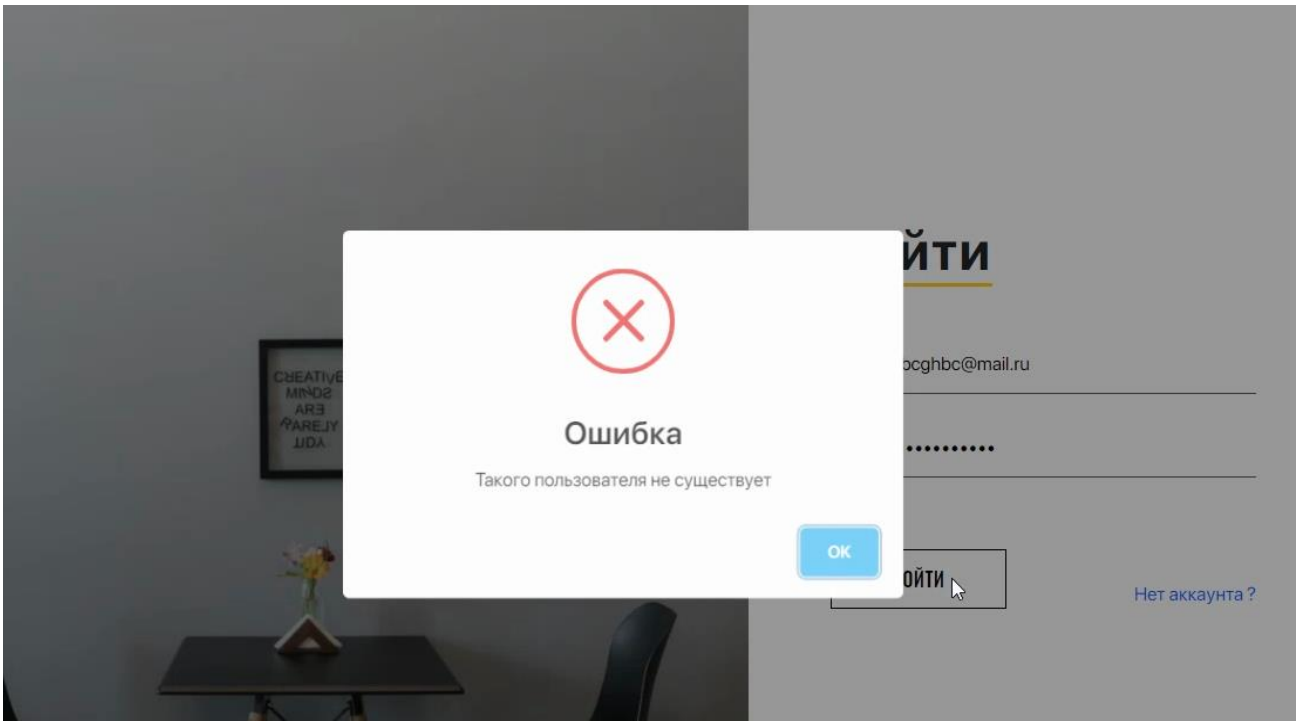

Рисунок 3.3 – Помилка авторизації

Веб-додаток було протестовано завдяки програмі Postman. Postman - зручний HTTP-клієнт для тестування веб-сайтів, входить в число кращих розширень каталогу Chrome Web Store в категорії «інструменти для роботи». Основне призначення програми - створення колекцій із запитами до API. Щоб програми спілкувалися між собою, їх API потрібно побудувати за єдиним стандартом. Одним з них є REST - стандарт архітектури взаємодії додатків і сайтів, що використовує протокол HTTP. Особливість REST в тому, що сервер не запам'ятовує стан користувача між запитами. Іншими словами, ідентифікація користувача (авторизаційний токен) і всі параметри виконання операції передаються в кожному запиті. Цей підхід настільки простий і зручний, що майже витіснив всі інші. Тестування API проводять, ґрунтуючись на бізнес-логіці програмного продукту. Воно відноситься до інтеграційного тестування, а значить в ході нього можна відловити помилки взаємодії між модулями системи або між системами.

ВИСНОВКИ

Вході виконання дипломної роботи були виконані всі поставлені задачі. Проаналізована предметна область та виявлена існуюча проблематика. Були обрані підходящі для виконання роботи технології. Спроектований та реалізований веб-сайт «Loft».

Результатом став програмний продукт – інтернет-магазин дизайнерських меблів за допомогою якого покупці можуть одержати бажаний їм продукт. Надалі планується впровадження веб-сайту і в разі потреби додавання нових функціональних можливостей.

СПИСОК ЛІТЕРАТУРИ

1. NodeJs - https://uk.wikipedia.org/wiki/Node.js
2. ExpressJs - https://metanit.com/web/nodejs/4.1.php
3. Sequelize - https://metanit.com/web/nodejs/9.1.php
4. HTML - https://developer.mozilla.org/ru/docs/Web/HTML
5. Sass - https://sass-scss.ru/guide/
6. Bootstrap - https://incourse.if.ua/web/bootstrap.php
7. ReactJS - https://ru.reactjs.org/
8. Axios - https://inlnk.ru/WGzmK
9. React-router-dom - https://metanit.com/web/react/4.1.php
10. MobX - https://web-creator.ru/technologies/webdev/mobx

ДОДАТОК А

Папка server

Головний файл серверної частини index.js

```
require('dotenv').config()
const express = require('express')
const path = require('path')
const router = require('./routers/index')
const sequelize = require('./db')
const models = require('./models/models')
const errorHandler = require('./middleware/ErrorHandlingMiddleware')
const cors = require('cors')
const fileUpload = require('express-fileupload')



//Порт на котором работает сервер
const PORT = process.env.PORT || 5000

//Запуск приложения
const app = express()

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

//Извлекает всю часть тела входящего потока запросов и предоставляет
его на req.body

//Использование абсолютного пути для предоставления файлов
app.use('/', express.static(path.join(__dirname, 'client')))

//Обработка ошибок последний middleware
app.use(errorHandler)

async function start(){
    try{
        await sequelize.authenticate()
        await sequelize.sync()
        app.listen(PORT, () => console.log(`Server started on port
${PORT}`))
    } catch (e){
        console.log(e)
    }
}
start()
```

ДОДАТОК Б

Папка server

Файл db.js

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

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

ДОДАТОК В

Папка server

Змінні оточення .env

```
PORT=5000
JWT_SECRET = 'random_secret_key45645'
DB_NAME=loft
DB_USER=root
DB_PASSWORD=admin
DB_HOST=localhost
DB_PORT=3306
```

ДОДАТОК Г

Папка server/routers

Файл brandRouter.js

```
const Router = require('express')
const router = new Router()
const brandController = require('../controller/brandController')

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

module.exports = router
```

Файл furnitureRouter.js

```
const Router = require('express')
const router = new Router()
const furnitureController =
require('../controller/furnitureController')

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

module.exports = router
```

Файл typeRouter.js

```
const Router = require('express')
const router = new Router()
const typeController = require('../controller/typeController')
const checkRole = require('../middleware/checkRoleMiddleware')

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

module.exports = router
```

Файл userRouter.js

```
const Router = require('express')
const router = Router()
const userController = require('../controller/userController')
const authMiddleware = require('../middleware/authMiddleware')

//Вход
router.post('/login' ,userController.login)
//Регистрация
router.post('/registration', userController.registration)
//Проверка авторизации
router.get('/auth' ,authMiddleware,userController.check)
```

```
module.exports = router
```

Файл index.js

```
const Router = require('express')
const router = new Router()
const userRouter = require('./userRouter')
const typeRouter = require('./typeRouter')
const brandRouter = require('./brandRouter')
const furnitureRouter = require('./furnitureRouter')
const mainpageRouter = require('./mainpageRouter')

router.use('/user', userRouter)
router.use('/type',typeRouter)
router.use('/brand',brandRouter)
router.use('/furniture',furnitureRouter)


module.exports = router
```

ДОДАТОК Ґ

Папка server/models

Файл models.js

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

const User = sequelize.define('user', {
    id: {type: DataTypes.INTEGER, primaryKey:true, autoIncrement:
true},
    email:{type:DataTypes.STRING, unique:true},
    password:{type:DataTypes.STRING},
    role:{type:DataTypes.STRING, defaultValue:'USER'}
})
const Basket = sequelize.define('basket', {
    id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:
true}
})
const BasketFurniture = sequelize.define('basket_furniture', {
    id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:
true}
})
const Furniture = sequelize.define('furniture', {
    id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:
true},
    name:{type:DataTypes.STRING,unique:true, allowNull:false},
    price:{type:DataTypes.INTEGER, allowNull:false},
    img:{type:DataTypes.STRING  , allowNull:false}
})
const Type = sequelize.define('type', {
    id:{type:DataTypes.INTEGER, primaryKey:true,
autoIncrement:true},
    name:{type:DataTypes.STRING, unique:true,allowNull:false}
})
const Brand = sequelize.define('brand', {
    id:{type:DataTypes.INTEGER, primaryKey:true,
autoIncrement:true},
    name:{type:DataTypes.STRING, unique:true,allowNull:false}
})
const FurnitureInfo = sequelize.define('furniture_info', {
    id:{type:DataTypes.INTEGER, primaryKey:true,
autoIncrement:true},
    title:{type:DataTypes.STRING,allowNull:false},
    description:{type:DataTypes.STRING,allowNull:false}
})
const TypeBrand = sequelize.define('type_brand',{
    id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:true}
})

User.hasOne(Basket)
Basket.belongsTo(User)
```

```
Basket.hasMany(BasketFurniture)
BasketFurniture.belongsTo(Basket)

Type.hasMany(Furniture)
Furniture.belongsTo(Type)

Brand.hasMany(Furniture)
Furniture.belongsTo(Brand)

Furniture.hasMany(BasketFurniture)
BasketFurniture.belongsTo(Furniture)

Furniture.hasMany(FurnitureInfo,{as: 'info'})
FurnitureInfo.belongsTo(Furniture)

Furniture.hasMany(BasketFurniture)
BasketFurniture.belongsTo(Furniture)

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

module.exports = {
    User,
    Basket,
    BasketFurniture,
    Furniture,
    Type,
    Brand,
    TypeBrand,
    FurnitureInfo
}
```

ДОДАТОК Д

Папка server/middleware

Файл authMiddleware.js

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

Файл checkRoleMiddleware.js

```
const jwt = require('jsonwebtoken')
module.exports = function (role){
    return module.exports = function (req, res, next) {
        if(req.method === 'OPTIONS') {
            next()
        }
        try {
            const token = req.headers.authorization.split(' ')[1]
//Bearer
            if(!token){
                return res.status(401).json({message: 'Пользователь
не авторизован'})
            }
            const decoded = jwt.verify(token,
process.env.JWT_SECRET)
            if (decoded.role !== role) {
                return res.status(403).json({message: "Нет
доступа"})
            }
            req.user = decoded
            next()
        } catch (e){
            res.status(401).json({message: 'Пользователь не
авторизован'})
        }
    }
```

```
}
```

Файл checkRoleMiddleware.js

```
const ApiError = require('../errors/ApiErrors')

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

ДОДАТОК Е

Папка server/errors

Файл ApiErrors.js

```
const ApiError = require('../errors/ApiErrors')

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

ДОДАТОК Є

Папка server/controller

Файл brandController.js

```
const {Brand} = require('../models/models')
const ApiError = require('../errors/ApiErrors')
class BrandController{
    async create(req,res){
        const {name} = req.body
        const brand = await Brand.create({name})
        return res.json(brand)
    }
    async getAll(req,res){
        const brands = await Brand.findAll()
        res.json(brands)
    }
}
module.exports = new BrandController()
```

Файл furnitureController.js

```
const uuid = require('uuid')
const path = require('path')
const {Furniture, FurnitureInfo} = require('../models/models')
const ApiError = require('../errors/ApiErrors')

class FurnitureController{
    async create(req,res, next){
        try{
            let {name, price, brandId, typeId, info} = req.body
            const {img} = req.files
            let fileName = uuid.v4() + '.jpg'
            img.mv(path.resolve(__dirname, '..', 'static',
fileName))
            const furniture = await Furniture.create({name, price,
brandId, typeId, img: fileName})
            if(info) {
                info = JSON.parse(info)
                info.forEach(i=>
                    FurnitureInfo.create({
                        title: i.title,
                        description: i.description,
                        furnitureId: furniture.id
                    })
                )
            }

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

```
            }

        }
        async getAll(req,res){
            try{
                let {brandId, typeId, limit, page} = req.query
                page = page || 1
                limit = limit || 12

                let offset = page * limit - limit
                parseInt(offset)
                let furniture;
                if(!brandId && !typeId){
                    furniture = await Furniture.findAndCountAll({limit:
parseInt(limit,10), offset})
                }
                if(brandId && !typeId){
                    furniture = await
Furniture.findAndCountAll({where:{brandId},limit: parseInt(limit,10),
offset})
                }
                if(!brandId && typeId){
                    furniture = await
Furniture.findAndCountAll({where:{typeId},limit: parseInt(limit,10),
offset})
                }
                if(brandId && typeId){
                    furniture = await
Furniture.findAndCountAll({where:{typeId, brandId},limit:
parseInt(limit,10), offset})
                }
                return res.json(furniture)
            } catch (e) {
                console.log(e)
            }

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

    }
    module.exports = new FurnitureController()
```

Файл typeController.js

```
const {Type} = require('../models/models')
const ApiError = require('../errors/ApiErrors')
class TypeController{
    async create(req,res){
        const {name} = req.body
        const type = await Type.create({name})
        return res.json(type)
    }
    async getAll(req,res){
        const types = await Type.findAll()
        return res.json(types)
    }

}
module.exports = new TypeController()
```

Файл typeController.js

```
const ApiError = require('../errors/ApiErrors')
const jwt = require('jsonwebtoken')
const bcrypt = require('bcryptjs')
const {User, Basket} = require('../models/models')

const generateJwt = (id, email, role) => {
    return jwt.sign({id, email ,role},
process.env.JWT_SECRET,{expiresIn: '30d'})
}

class UserController {
    async registration (req, res, next){
        // const errors = validationResult(req)
        // if(!errors.isEmpty()){
        //     return res.status(400).json({message: 'Ошибка при
регистрации', errors})
        // }
        const {email, password, role} = req.body
        if(!email || !password){
            return next(ApiError.badRequest('Некорректный email или
password'))
        }
        const candidate = await User.findOne({where: {email}})
        if(candidate){
            return next(ApiError.badRequest('Пользователь с таким
email уже существует'))
        }
        const hashPassword = await bcrypt.hash(password, 5)
        const user = await User.create({email, password:
hashPassword, role})
        const basket = await Basket.create({userId: user.id})
        const token = generateJwt(user.id, user.email, user.role)
        console.log(token)
        return res.json({token})
```

```
        }
        async login (req, res, next){
            try{
                const {email, password} = req.body
                const user = await User.findOne({where: {email}})
                if(!user) {
                    return next(ApiError.internal('Пользователь не
найден'))
                }
                let comparePassword = bcrypt.compareSync(password,
user.password)
                if(!comparePassword){
                    return next(ApiError.internal('Указан неверный
пароль'))
                }
                const token = generateJwt(user.id, user.email,
user.role)
                return res.json({token})
            } catch (e) {
                console.log(e)
            }
        }


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

ДОДАТОК Ж

Папка client/src/components

Файл AppRouter.js

```
import React, {useContext} from 'react';
import {Switch, Route, Redirect} from 'react-router-dom'
import {authRoutes, publicRoutes} from "../../routes";
import {HOME_ROUTE} from "../../utils/const";
import {Context} from "../../index";

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

export default AppRouter;
```

Файл Header.js

```
import React, {useContext, useState, useEffect} from 'react'
import './style.sass'
import {HOME_ROUTE, BASKET_ROUTE, LOGIN_ROUTE, ABOUT_ROUTE,
CATALOG_ROUTE, TEAM_ROUTE} from '../../utils/const'
import {Context} from "../../index";
import {NavLink, Link} from "react-router-dom";
import {observer} from "mobx-react-lite";
import logo from '../../assets/icons/logo-black.svg'


const menuHamburger = () => {
    const headerMenu = document.querySelector('.header-menu')
    const body = document.querySelector('body')
    headerMenu.classList.toggle('active')
    body.classList.toggle('lock')
}

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

```
        const [links, setlinks] = useState([
            {
                active: false, // true/false
                key: 1,
                route: HOME_ROUTE
            },
            {
                title: "Главная",
                active: false, // true/false
                key: 2,
                route: HOME_ROUTE
            },
            {
                title: "О нас",
                active: false, // true/false
                key: 3,
                route: ABOUT_ROUTE
            },
            {
                title: "Команда",
                active: false, // true/false
                key: 4,
                route: TEAM_ROUTE
            },
            {
                title: "Каталог",
                active: false, // true/false
                key: 5,
                route: CATALOG_ROUTE
            }
        ]);

        const change_state = (key) => {
            const updated = links.map((item) => {
                if (item.key === key) return { ...item, active:
!item.active };
                else return { ...item, active: false };
            });

            setlinks(updated);
        };
        const {user} = useContext(Context)
        return (
            <div className="container">
                <header className="header">
                <div className="header-top">
                    <div className="header-top__logo">
                    {
                        links.map((item, index)=>(
                            index===1 && <Link to={HOME_ROUTE}
onClick={() => change_state(item.key)} key={item.key}><img
src={logo}/></Link>
                        ))
                    }
```

```
                    </div>
                    <div className="header-top__navbar">
                        <div className="header-top__navigation">
                            <nav className="d-flex">
                                <ul className="navbar">
                                {
                                    links.map((item)=>(
                                        <li onClick={() =>
change_state(item.key)} key={item.key} className={item.active ? "active"
: ""}>
                                            <Link to={`${item.route}`}
className="navbar-item">{item.title}</Link>
                                        </li>
                                    ))
                                }
                                </ul>

                            </nav>
                        </div>
                        <hr className="header-top__seperator" />
                    </div>
                </div>
                <div className="header-bottom">
                    <div className="header-bottom__text">
                        Заказать по номеру телефона
                    </div>
                    <a className="header-bottom__phone"
href="tel:380505757575">
                        +380505757575
                    </a>
                </div>
            </header>
            </div>

        );
    });

    export default Header;
```

Файл Footer.js

```
    import React from 'react'
    import {HOME_ROUTE, ABOUT_ROUTE, TEAM_ROUTE} from
'../../utils/const'
    import {Link} from "react-router-dom";
    import './style.sass'
    import footerImg from '../../assets/img/footer-img.jpg'
    import sbmIcon from '../../assets/icons/footer-form.svg'

    const Footer = () => {
        return (
            <footer className="footer" id="footer">
```

```
                <div className="container">
                    <div className="row">
                        <div className="col-lg-6">
                            <div className="footer_img">
                                <img src={footerImg} alt=""/>
                            </div>
                        </div>
                        <div className="footer_wrapper col-12 col-lg-6
d-flex">
                            <div className="row links_width">
                                <div className="footer_links col-12
offset-lg-2 col-lg-10 d-flex">
                                    <div className="footer_links_block">
                                        <div
className="footer_links_title">Помощь</div>
                                        <ul className="footer_menu">
                                            <li><Link to={HOME_ROUTE}
className="footer_links_item">Главная</Link></li>
                                            <li><Link to={ABOUT_ROUTE}
className="footer_links_item">О нас</Link></li>
                                            <li><Link to={TEAM_ROUTE}
className="footer_links_item">Команда</Link></li>
                                            <li><Link to={HOME_ROUTE}
className="footer_links_item">Информация</Link></li>
                                        </ul>
                                    </div>
                                    <div className="footer_links_block">
                                        <div
className="footer_links_title">Соц.сети</div>
                                        <ul className="footer_menu">
                                            <li><Link to={instagram.com}
className="footer_links_item">Instagram</Link></li>
                                            <li><Link to={youtube.com}
className="footer_links_item">YouTube</Link></li>
                                            <li><Link to={snapchat.com}
className="footer_links_item">Snapchat</Link></li>
                                        </ul>
                                    </div>
                                </div>
                                <div className="footer_input_block
offset-lg-2 col-10">
                                    <h2
className="footer_header">Сделать заказ</h2>
                                    <form action="mailer/smart.php"
method="post" className="email_form_wrapper">
                                        <input type="email"
                                               name="email"
                                               autoComplete="off"
                                               placeholder="Введите вашу
почту"
                                               required/>
                                        <button className="submit_btn">
                                            <img src={sbmIcon}
alt="Отправить"/>
```

```
                                        </button>
                                    </form>
                                </div>
                            </div>
                        </div>
                    </div>
                </footer>
    );
};

export default Footer;
```

ДОДАТОК З

Папка client/src/store

Файл UserStore.js

```
import {makeAutoObservable} from "mobx";

export default class UserStore{
    constructor() {
        this._isAuth = false
        this._user = {}
        makeAutoObservable(this)
    }
    setIsAuth(bool) {
        this._isAuth = bool
    }
    setUser(user) {
        this._user = user
    }
    get isAuth(){
        return this._isAuth
    }
    get user(){
        return this._user
    }
}
```

ДОДАТОК И

Папка client/src/utils

Файл const.js

```
export const ADMIN_ROUTE = '/admin'
export const LOGIN_ROUTE = '/login'
export const REGISTRATION_ROUTE = '/registration'
export const ABOUT_ROUTE = '/about'
export const TEAM_ROUTE = '/team'
export const HOME_ROUTE = '/'
export const CATALOG_ROUTE = '/catalog'
export const FURNITURE_ROUTE = '/furniture'
export const BASKET_ROUTE = '/basket'
```

ДОДАТОК І

Папка client/src

Файл App.js

```
import {BrowserRouter} from "react-router-dom";
import AppRouter from "./components/AppRouter/AppRouter";
import Footer from './components/Footer/Footer';
import Header from "./components/Header/Header";
import './App.sass';



function App() {


    return (
    <BrowserRouter>
        <Header />
        <AppRouter/>
        <Footer/>
    </BrowserRouter>
  );
}

export default App;
```

ДОДАТОК Ї

Папка client/src

Файл routes.js

```
import Admin from './pages/Admin/Admin'
import Basket from './pages/Basket/Basket'
import About from './pages/About/About'
import Catalog from './pages/Catalog/Catalog'
import Furniture from './pages/FurniturePage/FurniturePage'
import Home from './pages/Home/Home'
import Auth from './pages/Auth/Auth'
import Team from './pages/Team/Team'
import {ADMIN_ROUTE, BASKET_ROUTE, ABOUT_ROUTE, CATALOG_ROUTE,
FURNITURE_ROUTE, HOME_ROUTE, LOGIN_ROUTE, REGISTRATION_ROUTE, TEAM_ROUTE}
from "./utils/const";

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

export const publicRoutes = [
    {
        path: ABOUT_ROUTE,
        Component: About
    },
    {
        path: CATALOG_ROUTE,
        Component: Catalog
    },
    {
        path: FURNITURE_ROUTE + '/:id',
        Component: Furniture
    },
    {
        path: HOME_ROUTE,
        Component: Home
    },
    {
        path: LOGIN_ROUTE,
        Component: Auth
    },
    {
        path: REGISTRATION_ROUTE,
        Component: Auth
    },
```

```
    {
        path: TEAM_ROUTE,
        Component: Team
    }
]
```

ДОДАТОК Ї

Папка client/src

Файл index.js

```
import React, {createContext} from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import UserStore from './store/UserStore'
import FurnitureStore from './store/FurnitureStore'
import './index.sass'

export const Context = createContext(null)

ReactDOM.render(
    <Context.Provider value={{
        user: new UserStore(),
        furniture: new FurnitureStore()
    }}>
        <App />
    </Context.Provider>,
  document.getElementById('root')
);
```