МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційний ресурс для компанії «Фасад» з використанням фреймворку React»

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

Керівник роботи **Проценко О.Б.**

Студента групи ІН – 63 **Бурмака І.О.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедри Довбиш А.С.

"_____" _____2020 г.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-63 спеціальності "Інформатика" денної форми навчання Бурмаки Іллі Олександровича.

Тема: "**Інформаційний ресурс для компанії «Фасад» з використанням фреймворку React**"

Затверджена наказом по СумДУ

№_____ от _____2020 г.

Зміст пояснювальної записки: 1) аналітичний огляд моделі веб-додатку; 2) постановка завдання й формування етапів створення; 3) створення дизайну; 4) розробка клієнтської та серверної частин веб додатку; 5)Тестування веб-додатку; 6) аналіз та висновки роботи веб-додатку.

Дата видачі завдання "_____"_____2020 г.

Керівник випускної роботи _____ Проценко О.Б.

Завдання приняв до виконання _____ Бурмака І.О.

РЕФЕРАТ

Записка: 49 стор., 19 рис., 1 табл., 6 додатків, 21 джерело.

Об’єкт дослідження – інформаційний веб-ресурс.

Мета роботи — розробка швидкого та гнучкого веб-додаток з поставленими вимогами від замовника, виконаний на сучасних фреймворках та бібліотеках JavaScript (ES6).

Результати — створено сучасний інформаційний веб-додаток для ПП «FASAD». Реалізовано всі поставлені задачі. Розроблений веб-додаткок, який реалізовано за допомогою React та Node.js

РОЗРОБКА ВЕБ-ДОДАТУ, ІНФОРМАЦІЙНИЙ РЕСУРС НА БАЗІ REACT, ВЕБ СЕРВЕР NODE.JS.

ЗМІСТ

ВСТУП 5
1 ІНФОРМАЦІЙНИЙ ОГЛЯД 7
1.1 Огляд веб-додатку та його переваги 7
1.2 Клієнт-серверна технологія для веб-додатку 8
1.3 Огляд існуючих програмних рішень 10
1.4 Постановка задачі 11
2 ВИБІР МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ 12
2.1 Вибір методів розроблення 12
2.2 Вибір засобів програмування та тестування 13
3 ПРОГРАМНА РЕАЛІЗАЦІЯ 21
3.1 Проектування структури сайту 21
3.2 Логічна реалізація бази даних 23
3.3 Інтерфейс веб-додатку 28
3.4 Тестування веб-додатку 32
3.5 Розгортання додатка 34
ВИСНОВКИ 36
СПИСОК ЛІТЕРАТУРИ 37
ДОДАТКИ 39
Додаток А 39
Додаток Б 39
Додаток В 41
Додаток Г 41
Додаток Д 41
Додаток Е 42
Додаток Ж 46

ВСТУП

Впроваджування сучасних технологій в майже всі русла діяльності обумовлено швидким, а також прогресивним розвитком науки, якісним стрибком в можливостях для людини, стрімко зростаючими обсягами інформації в світі, складністю певних процесів, що відбуваються. Не обійшла стороною інформатизація та продаж товарів. Розробка сайтів для всіх компаній є актуальною і затребуваною сферою діяльності, тому що сайт фірми на веб-ресурсах являє собою досить дешевий і масовий спосіб реклами певного товару: або ж і самої компанії, дає можливість потенційним та існуючим споживачам легко отримувати інформацію про товари чи послуги компанії, інтереси та напрямки компанії, що може допомогти знайти нових замовників і партнерів по бізнесу, а отже, сприяє збільшенню обсягу продажів і рентабельності компанії.

Продаж товару через інтернет - це складний Інтернет - господарський комплекс з численними зовнішніми та внутрішніми зв'язками. І управління Інтернет магазину, його інформаційними потоками, процесом продажів, документообігом та іншими процесами являє собою складну систему, дрібні і великі завдання якої тісно пов'язані один з одним.

Управління діяльністю організації торгівлі, її підрозділів здійснюється за рахунок управлінських рішень, що приймаються керівництвом на основі даних про стан підрозділів, їх показників і даних із зовнішнього середовища. При цьому важливо розподілити між підрозділами не тільки функції, що випливають з призначення тієї чи іншої служби в роботі організації торгівлі, але і наявні трудові, фінансові та матеріальні ресурси.

Для реалізації мети поставлені наступні задачі:

- аналіз методів та засобів розробки серверної частини;
- створення структури бази даних;
- створення серверної частини;
- створення клієнтської частини;
- тестування продукту.

Для виконання задачі взяте замовлення у ФОП «ФАСАД» для створення інформаційного веб-ресурсу з використанням сучасного фреймворку React, Nest.js, а також Node.js.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд веб-додатку та його переваги

Все частіше компанії вдаються до послуг розробки веб-додатків, щоб ефективно вирішувати широкий спектр бізнес-завдань. Клієнт-серверний додаток, основна частина якої міститься на вилученому сервері, а призначений для користувача інтерфейс відображається в браузері у вигляді веб-сторінок. Для запуску веб-додатку користувачеві не потрібно встановлювати ніяких додаткових програм, воно запускається на будь-якому пристрої з браузером і з доступом в інтернет. Робота клієнта не залежить від операційної системи, що стоїть на комп'ютері користувача, тому при розробці веб-додатків немає необхідності писати окремі версії для Windows, Linux, Mac OS і інших операційних систем.

Архітектура веб-додатку - це структура, яка підтримує взаємодію між компонентами додатків. Веб-додаток поєднує сценарії як на стороні сервера, так і на стороні клієнта. Сценарії на стороні сервера відповідають за зберігання даних, а сторона клієнта представляє їх клієнтам. Всі мови веб-додатків можна класифікувати на клієнтські і серверні. Як випливає з назви, клієнтські мови використовуються для написання програм, які виконуються на стороні клієнта (web-браузер), а серверні - для програм, які виконуються на сервері. Це поділ графічно продемонстровано на рисунку 1.1.

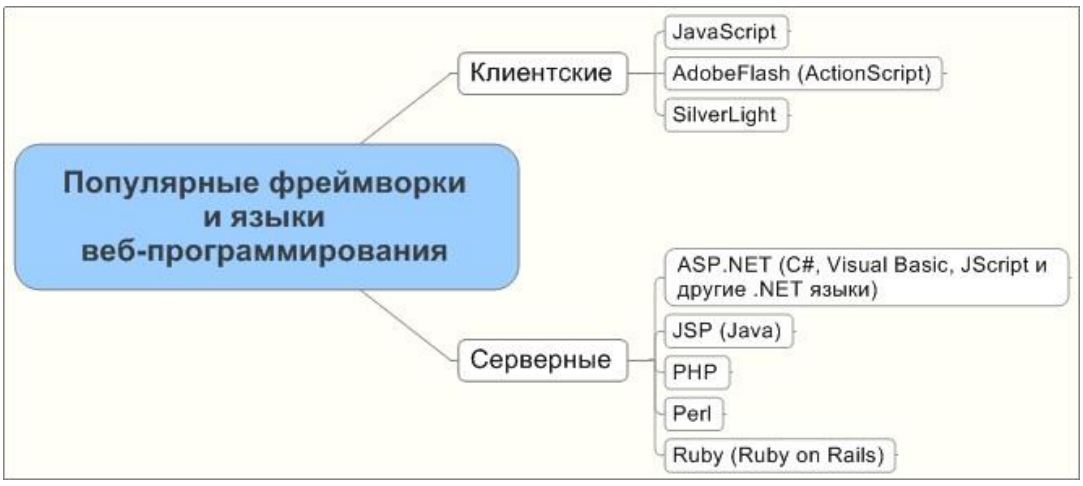

Рисунок 1.1 – Список мов для розробки веб-додатку

Для створення серверної частини веб-додатків використовуються такі мови програмування, як: PHP, ASP, ASP.NET, Perl, C / C ++, Java, Python, Ruby, NodeJS. Для реалізації клієнтської частини – HTML, CSS, JS, AJAX.

В переваги можемо додати адаптивність, відсутність клієнтського ПЗ, мережеву безпеку, тому що веб-система має єдину точку входу, захистити і налаштувати безпеку якої можна централізовано. А також масштабованість. З ростом навантаження на систему не треба нарощувати потужність клієнтських місць. Веб-додаток дозволяє обробляти більшу кількість даних, як правило, тільки силами апаратних ресурсів, без переписування коду і зміни архітектури.

1.2 Клієнт-серверна технологія для веб-додатку

Клієнт і сервер взаємодіють один з одним в мережі Інтернет або в будь-якій іншій комп'ютерній мережі за допомогою різних мережевих протоколів, наприклад, IP протокол, HTTP протокол, FTP та інші. Протоколів насправді дуже багато і кожен протокол дозволяє надавати ту чи іншу послугу. Завдяки HTTP протоколу браузер відправляє спеціальне HTTP повідомлення, в якому зазначено якусь інформацію і в якому вигляді він хоче отримати від сервера, сервер, отримавши таке повідомлення, відсилає браузеру у відповідь схоже по структурі повідомлення (або кілька повідомлень), в якому міститься потрібна інформація, як правило це HTML документ.

Повідомлення, які посилають клієнти отримали назви HTTP запити. Запити мають спеціальні методи, які говорять сервера про те, як обробляти повідомлення. А повідомлення, які посилає сервер отримали назву HTTP відповіді, вони містять крім корисної інформації ще й спеціальні коди стану, які дозволяють браузеру дізнатися те, як сервер зрозумів його запит.

Це був опис, як взаємодіють клієнт і сервер на сьомому рівні моделі OSI, але, насправді це взаємодія відбувається на всіх семи рівнях. Коли клієнт відправляє запит, повідомлення упаковується, можна уявити, що повідомлення загортається в сім обгорток (хоча їх може бути набагато більше або ж менше), а коли повідомлення отримує сервер, він починає ці обгортки розгортати.

Також варто зауважити, що в основі взаємодії клієнт-сервер лежить принцип того, що таку взаємодію починає клієнт, сервер лише відповідає клієнту і повідомляє про те чи може він надати послугу клієнтові і якщо може, то на яких умовах. Клієнтське програмне забезпечення та серверне програмне забезпечення зазвичай встановлено на різних машинах, але також вони можуть працювати і на одному комп'ютері. Дана концепція взаємодії була розроблена в першу чергу для того, щоб розділити навантаження між учасниками процесу обміну інформацією, а також для того, щоб розділити програмний код постачальника і замовника. На рисунку 1.2 можна побачити спрощену схему взаємодії клієнт-сервер.

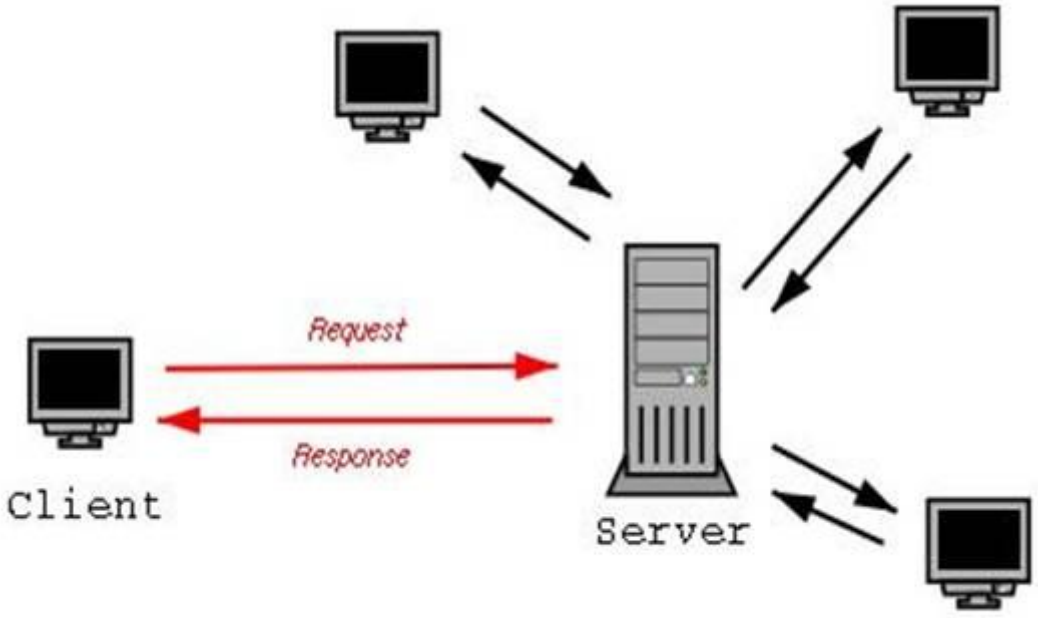

Рисунок 1.2 – Схема зв’язку клієнта та сервера

До одного сервера може звертатися відразу кілька клієнтів. Також варто зауважити, що кількість клієнтів, які можуть одночасно взаємодіяти з сервером залежить від потужності сервера і від того, що хоче отримати клієнт від сервера. Багато мережевих протоколів побудовані на архітектурі клієнт-сервер, тому в їх основі зазвичай лежать однакові або схожі принципи взаємодії, а різницю можливо побачити о лише в деталях, які обумовлені особливостями і специфікою області, для якої розроблявся той чи інший мережевий протокол.

HTTPS(Hypertext Transfer Protocol Secure) - це «апгрейднутий» HTTP, з покращеною безпекою. Проблема звичайного протоколу HTTP полягає в тому, що інформація, що надходить з сервера на браузер, не шифрується, а значить, її можна легко вкрасти. Протоколи HTTPS виправляють це за допомогою сертифіката SSL, який допомагає створити захищене зашифроване з'єднання між сервером та браузером, тим самим захищаючи потенційно важливу інформацію від крадіжки під час передачі між сервером та браузером.

Найважливішою відмінністю між двома протоколами є сертифікат SSL. Ця додаткова безпека може бути надзвичайно важливою, особливо для веб-сайтів, які приймають конфіденційні дані від своїх користувачів, такі як інформація про кредитну карту та паролі. SSL шифрує інформацію, яку користувачі надають на сайт. Навіть якщо комусь вдасться викрасти дані, що передаються між відправником та одержувачем, вони не зможуть зрозуміти це завдяки цьому шифруванню. Але крім додавання додаткового рівня безпеки, HTTPS також захищений за допомогою протоколу TLS. Він допомагає забезпечити цілісність даних, що запобігає зміні або пошкодженню передачі даних, а також автентифікацію, яка доводить вашим користувачам, що вони спілкуються із призначеним веб-сайтом. Користувачі можуть визначити, чи веб-сайт використовує протокол HTTPS за веб-адресою. Перша частина веб-адреси вказує, чи веб-сайт використовує протоколи HTTP або HTTPS. Отже, різниця між HTTP та HTTPS - просто наявність сертифіката SSL та TLS протокол, тому надійність та безпека на стороні HTTPS.

1.3 Огляд існуючих програмних рішень

Додатки для мебельних фабрик можна умовно поділити на три наступні групи:

- сайт-візитка;
- ІМ(інтернет-магазин);
- інтерактивний ІМ з використанням 3D моделей.

Було розглянуто конкурентів (ІМ), велика кількість переглянутих веб-додатків – написані на CMS, мають малий рівень безпеки, та повільно завантажується та роутинг між сторінками працює також повільно. Одним із таких сайтів є https://konstanta.ua. Та самий головний мінус, це те, що більшість додатків, не підтримують декілька мов, а це в свою чергу великий мінус, при роботі з «західними» споживачами.

1.4 Постановка задачі

У даній роботі необхідно розробити зручний веб-додаток для ПП «ФАСАД» - меблевої фабрики, а саме з такими можливостями:

- можливість створення категорії;
- можливість створення підкатегорії;
- створювати та переглядати продукцію;
- можливість перегляду усіх товарів;
- можливість сортувати продукцію;
- залишати відгуки.

Провівши аналіз – можливостей CMS, а також вимог заказника, вирішено було створити на додаток на основі React. Результатом роботи має стати швидкий веб-додаток, який повинен бути безпечним, підтримувати декілька мов, адаптивним. Та саме головне бути «user friendly».

2 ВИБІР МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ

2.1 Вибір методів розроблення

Перш за все потрібно для себе вирішити що краще підходить під вимоги нашого веб-додатку. І саме перше: з чим ми стикаємося з таким питанням: «CMS vs Framework». CMS - це програма, що використовується для веб-розробки, яка дозволяє розробникам вмісту створювати, редагувати та публікувати текст чи зображення чи товар на веб-сайті. Програмне забезпечення CMS дуже поширене в Інтернеті, оскільки дозволяє «нетехнічним» людям писати та керувати вмістом, не турбуючись про код. Однак існує тенденція, використовувати фреймворки для сучасних веб-додатків, які обіцяють більшу гнучкість за рахунок більш тривалої розробки додатку. Основна відмінність від CMS полягає в тому, що він не має «клієнтської частини», його можна використовувати для зберігання вмісту, але він не відображає його візуально. CMS - це програмне забезпечення, яке використовується для управління веб-контентом (наприклад, WordPress та Squarespace, Joomla). У них є графічні інтерфейси користувачів (GUI), які дозволяють розробникам створювати їх вміст, вибираючи з існуючих шаблонів або завантажуючи плагіни для більшої надстройки. Вміст зберігається в базі даних і відображається користувачеві на основі шаблону. Оскільки засоби CMS призначені для зручності користування, люди можуть вносити зміни не торкаючись коду.

Framework – це абстракція, в якій програмне забезпечення, що надає функціональність додатку, може бути вибірково змінено додатковим користувацьким кодом, специфічне для програми, це універсальне програмне середовище багаторазового використання, яке надає особливу функціональність як частина великої програмної платформи для полегшення розробки програмних продуктів, продуктів і рішень. Фреймворки можуть включати в себе програми підтримки, компілятори, бібліотеки, набори інструментів і інтерфейси прикладного програмування (API), які об'єднують всі різні компоненти для розробки проекту або системи.

Фреймворки мають ключові відмінні ознаки, які відокремлюють їх від звичайних бібліотек:

- розширюваність: користувач може розширити фреймворк – зазвичай шляхом написанням нового коду, щоб забезпечити певну функціональність;
- на відміну від бібліотек або стандартних користувальницьких додатків, загальний потік програми диктується за правилами фреймворку;
- можливість розширяти фреймворк, при цьому не змінюючи його код.

За допомогою CMS можна отримати заздалегідь визначений набір функцій, можна вибрати вже стилізовану тему для сайту, і можна додавати нові функції, легко встановлюючи плагіни. Проте можуть виникнути ускладнення, коли потрібно веб-сайт для реалізації конкретних потреб. У фреймворках потрібно побудувати все з нуля, але можна створити відмінні та унікальні функції. Фреймворки також не винаходитимуть колеса, оскільки використовуваний код надає базу таких функцій, як система входу або привілеї користувача.

Фреймворки дуже легко налаштовуються, а CMS, як правило, має обмеження. Наприклад, ви не можна реально змінити базові функціональні можливості CMS, або вони не будуть оновлені належним чином, натомість фреймворки не мають обмежень.

Ураховуючи усі обставини, а саме поставлені задачі, можна зробити висновок, що фреймворк краще підійде для створення такого додатку, адже він унікальний та специфічний.

2.2 Вибір засобів програмування та тестування

Варто відразу наголосити, що не існує однієї мови програмування, яка перевершувала б усі інші. Перевага будь-якої мови програмування може виявлятися тільки в контексті певної задачі, але це зовсім необов'язково. Багато

задач можуть бути ефективно вирішені за допомогою будь-якої сучасної популярної мови програмування. Під час дослідження було обрано наступні технології: Node.js, PostgreSQL, TypeScript, Docker, JavaScript.

Node.js – це платформа на стороні сервера, побудована на Google Engine Engine (V8 Engine). Node.js був розроблений Райном Далем у 2009 році.

Node.js – це платформа, побудована на середовищі виконання JavaScript Chrome для легкого створення швидких і масштабованих мережевих програм. Node.js використовує модуль вводу-виводу, керовану подіями, що робить його легким і ефективним, ідеально підходить для інтенсивних додатків в реальному часі, які працюють у розподілених пристроях.

Node.js є відкритим вихідним кодом, крос-платформним середовищем виконання для розробки серверних і мережевих додатків. Програми Node.js написані на JavaScript, і можуть бути запущені в середовищі виконання Node.js на OSX, Microsoft Windows і Linux.

Node.js також надає багату бібліотеку різних модулів JavaScript, що значною мірою спрощує розробку веб-додатків, що використовують Node.js.

За результатами досліджень, нижче наведено деякі важливі функції, які роблять Node.js гарним вибором для реалізації поставлених задач:

- асинхронне управління – всі API бібліотеки Node.js є асинхронними, тобто неблокуючими. Це означає, що сервер на основі Node.js ніколи не чекає на повернення даних. Сервер переходить до наступного API після виклику, а механізм сповіщень Node.js допомагає серверу отримувати відповідь від попереднього виклику API;
- Будується на Google Chrome Engine V8, бібліотека Node.js дуже швидко виконує код;
- один потік, але високо масштабована Node.js використовує одну потокову модель з циклом подій. Механізм подій допомагає серверу реагувати неблокуючим способом і робить сервер дуже масштабованим, на відміну від

традиційних серверів, які створюють обмежені потоки для обробки запитів. Node.js використовує одну потокову програму, і одна й та сама програма може надавати послуги набагато більшу кількість запитів, ніж традиційні сервери;

- програми Node.js ніколи не буферують будь-які дані. Ці програми просто виводять результат;
- Node.js випускається під ліцензією MIT.

PostgreSQL – не просто реляційна, а об'єктно-реляційна база. Це дає їй певні переваги перед іншими базами даних з відкритим вихідним кодом, наприклад MySQL, MariaDB і Firebird.

Основною характеристикою об'єктно-реляційної бази даних є підтримка визначених користувачем об'єктів та їх поведінки, включаючи типи даних, функції, оператори, домени та індекси. Це робить PostgreSQL надзвичайно гнучким і надійним рішенням. Серед іншого, складні структури даних можна створювати, зберігати і читати. За результатами досліджень, нижче описані складні структури, які не підтримують стандартну СУБД.

Існує великий список типів даних, які підтримує PostgreSQL. Окрім числових, масивів, логічних типів і типів дат, які очікується, PostgreSQL може представити uuid, грошовий тип, перерахований тип, геометричний тип, двійковий тип, тип мережевого адресу, тип рядку бітів, а також текстовий пошук, xml, json , масиви, складові та типи діапазонів, а також деякі внутрішні типи для ідентифікації об'єкта. За результатами досліджень, кожна з цих MySQL, MariaDB і Firebird має деякі з них, але лише PostgreSQL підтримує їх усі.

Оскільки PostgreSQL є об'єктно-реляційною базою даних, масиви значень можна зберігати для більшості існуючих типів даних. Зробити це можна, додавши квадратні дужки до специфікації типу даних для стовпця або за допомогою виразу ARRAY. Розмір масиву можна вказати, але не обов'язково. MySQL, MariaDB і Firebird не мають такої можливості. Щоб зберегти масив таких значень у традиційній реляційній базі даних, потрібно створити окрему таблицю з рядком для кожного з значень

Підтримка JSON у PostgreSQL дає змогу переходити до схеми з базами даних SQL. Це може бути корисним, коли структура даних вимагає певної гнучкості, оскільки вона все ще змінюється в процесі розробки або коли невідомо, які поля даних об'єкт даних буде містити.

Тип даних JSON забезпечує дійсний JSON, який дозволяє використовувати спеціалізовані JSON-оператори та функції, вбудовані в PostgreSQL для запитів і маніпулювання даними. Також доступний тип JSONB – двійкова форма JSON, де порядок об'єктів не зберігається, але зберігається оптимально, і зберігається тільки останнє значення для дубльованих ключів. JSONB, як правило, є кращим форматом, оскільки він вимагає менше місця на об'єкт, може бути проіндексований і може оброблятися швидше, оскільки не вимагає повторного аналізу.

PostgreSQL має багато можливостей. Побудований з використанням об'єктно-реляційної моделі, вона підтримує складні структури і широту вбудованих і визначених користувачем типів даних. База надає велику ємність даних і довіряється його цілісності даних. Можливо, не всі переваги на даний момент використані, але, оскільки потреби в даних можуть швидко розвиватися, безсумнівно, очевидні переваги.

TypeScript – це мова програмування з відкритим вихідним кодом, розроблена та підтримувана корпорацією Майкрософт. Це синтаксична надмножина JavaScript. Яка додає до мови додаткову статичну типізацію.

TypeScript призначений для розробки великих додатків і транскомпілюється на JavaScript. Оскільки TypeScript є набором JavaScript, існуючі програми JavaScript також є дійсними програмами TypeScript.

TypeScript може використовуватися для розробки програм JavaScript для виконання на стороні клієнта та на стороні сервера (Node.js). Існує декілька варіантів для транскомпіляції. Можна використовувати за замовчуванням TypeScript Checker, або можна викликати компілятор Babel для перетворення TypeScript у JavaScript.

Docker – це інструмент, призначений для спрощення створення, розгортання та запуску додатків за допомогою контейнерів. Контейнери дозволяють розробнику упаковувати додатки з усіма необхідними частинами, такими як бібліотеки та інші залежності, та розсилати їх як один пакет. Роблячи це, завдяки контейнеру, розробник може бути впевнений, що програма буде працювати на будь-якій іншій машині Linux, незалежно від будь-яких налаштованих налаштувань, які може мати машина, яка може відрізнятися від машини, що використовується для написання та тестування коду.

У певному сенсі Docker схожий на віртуальну машину. Але, на відміну від віртуальної машини, Docker, замість того, щоб створювати цілу віртуальну операційну систему, дозволяє програмам використовувати те саме ядро Linux, що й система, на якій вони працюють, і лише вимагає, щоб програми були доставлені з речами, які вже не виконуються на комп'ютері. Це дає значний приріст продуктивності і зменшує розмір програми.

Docker є відкритим кодом. Це означає, що будь-хто може внести свій внесок у Docker і розширити його для задоволення власних потреб, якщо їм потрібні додаткові функції, які не доступні за замовчуванням.

Docker – це інструмент, призначений для використання як розробникам, так і системним адміністраторам. Для розробників це означає, що вони можуть зосередитися на написанні коду, не турбуючись про систему, на якій він буде працювати. Вона також дозволяє їм одержати перевагу, використовуючи одну з тисяч програм, вже розроблених для роботи в контейнері Docker, як частину їх застосування. Для операційного персоналу, Docker надає гнучкість і потенційно зменшує кількість систем, необхідних через невеликий розмір і менші накладні витрати.

Стосовно клієнтської частини, то буде використовуватись React разом з гарним додатком Next.js. React - це декларативна, ефективна та гнучка бібліотека JavaScript для створення інтерфейсів користувача. React може бути використаний як основа при розробці односторінкових або мобільних додатків.

Однак React стосується лише надання даних у DOM, і тому створення додатків React зазвичай вимагає використання додаткових бібліотек для управління стейту та маршрутизації Redux та React Router або Next є відповідними прикладами таких бібліотек. На відміну від багатьох своїх попередників, React оперує не безпосередньо на моделі об'єкта документа (DOM) браузера, а на віртуальному DOM. Тобто, замість того, щоб маніпулювати документом у браузері після зміни наших даних (що може бути досить повільним), він вирішує зміни в DOM певні елементи. Після того як віртуальний DOM був оновлений, React інтелектуально визначає, які зміни потрібно внести до фактичної DOM-адреси браузера.

React Virtual DOM існує повністю в пам'яті і є репрезентацією DOM веб-браузера. Через це, коли пишемо компонент React, то не пишемо безпосередньо до DOM, але пишемо віртуальний компонент, який React перетвориться на DOM. В основі всіх програм React лежать компоненти.

Компонент - це автономний модуль, який дає деякий вихід. То можемо записати елементи інтерфейсу, як кнопка або поле введення, як компонент React. Компоненти є композиційними. Він може включати в свій вихід один або більше інших компонентів. В цілому, для створення додатків на React, пишемо компоненти React, які відповідають різним елементам інтерфейсу. Потім організовуємо ці компоненти всередині компонентів вищого рівня, які визначають структуру нашої програми.

Наприклад, візьміть форму. Форма може складатися з багатьох елементів інтерфейсу, таких як поля введення, мітки, чекбокси або кнопки. Кожен елемент всередині форми може бути записаний як компонент React. Після чого запишемо компонент вищого рівня, сам компонент форми. Компонент форми вказував би структуру форми і включав би кожен із цих елементів інтерфейсу всередині неї. Компонент в React дотримується суворих принципів управління даними. Складні, інтерактивні користувацькі інтерфейси часто включають складні дані та стан програми. Площа поверхні React обмежена і спрямована на те, щоб дати

нам інструменти, щоб можна було передбачити, як буде виглядати наша клієнтська частина за певного набору обставин.

До React було вирішено підключити Next.js. Тому що, щоб створити повний веб-додаток з React з нуля, є багато важливих деталей, які потрібно врахувати:

- код потрібно поєднувати за допомогою такого пакета, як webpack, і трансформувати за допомогою компілятора типу Babel;
- потрібно зробити оптимізацію виробництва, наприклад розбиття коду;
- ви можете статично заздалегідь надати деякі сторінки для продуктивності та оптимізації. Ви також можете використовувати візуалізацію на стороні сервера або візуалізацію на стороні клієнта;
- можливо, вам доведеться написати якийсь код на стороні сервера, щоб підключити додаток React до сховища даних(API).

React може вирішити ці проблеми. Але таки підхід повинний мати правильний рівень абстракції - інакше це буде не дуже корисно. Він також повинен мати чудовий "досвід для розробників", забезпечуючи досвід під час написання коду.

Next.js пропонує вирішення всіх перерахованих вище проблем. Але що ще важливіше, це ставить вас та вашу команду на вершину успіху під час створення веб-додатку React. Next.js має найкращий у своєму класі «Досвід розробників» та багато вбудованих функцій; зразок з них:

- інтуїтивна система маршрутизації на основі сторінки (з підтримкою динамічних маршрутів);
- попереднє візуалізація, як статична генерація (SSG), так і серверне візуалізація (SSR) підтримуються на основі сторінки;
- автоматичне розділення коду для швидшого завантаження сторінки;

- маршрутизація на стороні клієнта з оптимізованим попереднім завантаженням;
- вбудована підтримка CSS та Sass та підтримка будь-якої бібліотеки CSS в JS;
- середовище розробки, яке підтримує заміну гарячого модуля;
- маршрути API для створення кінцевих точок API з функціями без сервера;
- повністю розширюється.

Next.js використовується в десятках тисяч виробничих веб-сайтів та веб-додатків, включаючи багато найбільших світових брендів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проектування структури сайту

Для початку потрібно реалізувати API(серверну частину). На етапі проектування потрібно знайти сутності та створити таблиці, а саме:

- таблиця користувачів– user;
- таблиця товарів – product;
- таблиця категорій – category;
- таблиця підкатегорій – sub_category;
- таблиця відгуків – feedback;
- таблиця зображень – assets;
- суміжна таблиця між товаром та категорією - product_subcategories_sub_category.

ERD схему зображено на рисунку 3.1. На якій можемо побачити відношення між таблицями.

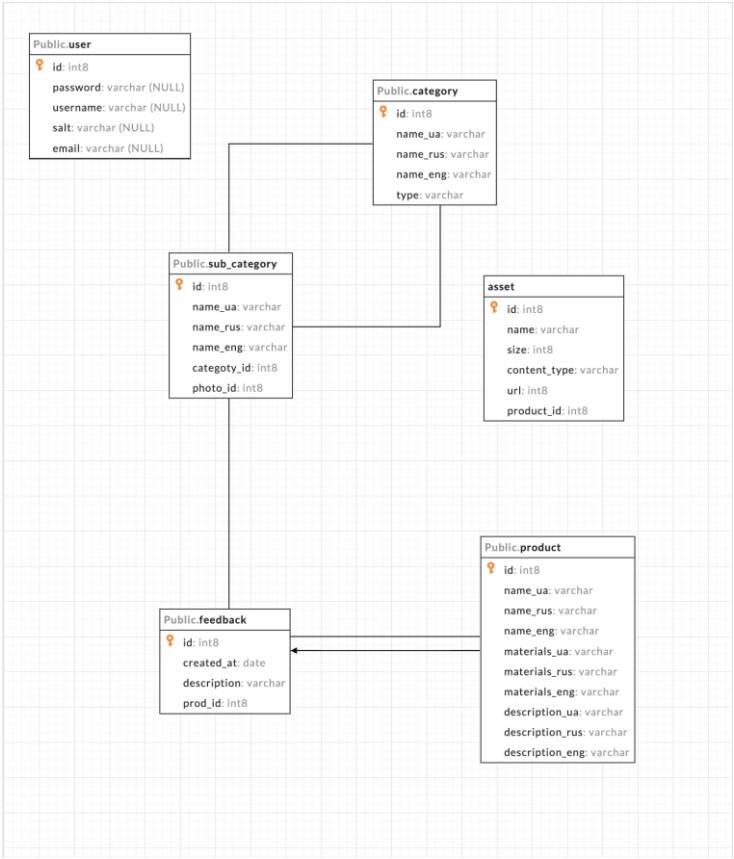

Рисунок 3.1 – ERD схема серверної частини

Проект реалізовано за допомогою однієї з найпопулярніших фреймворків Nest.js та ORM(TypeORM).

Nest - це основа для створення ефективних масштабованих додатків на сервері Node.js. Він використовує прогресивний JavaScript, створений і повністю підтримує TypeScript і поєднує елементи OOP (об'єктно-орієнтоване програмування), FP (функціональне програмування) та FRP (функціональне реактивне програмування). Під капотом Nest використовує надійні рамки HTTP-сервера, такі як Express (за замовчуванням).

Nest забезпечує рівень абстракції над цими загальними рамками Node.js (Express / Fastify), але також піддає свої API безпосередньо розробнику. Це дозволяє розробникам свободу використовувати безліч сторонніх модулів, доступних для базової платформи. Тестування запитів проводилося через Postman. ORM(Object-Relational Mapping) - технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних». Існують як пропрієтарні, так і вільні реалізації цієї технології.

Структура проекту має наступний вигляд – рисунок 3.2:

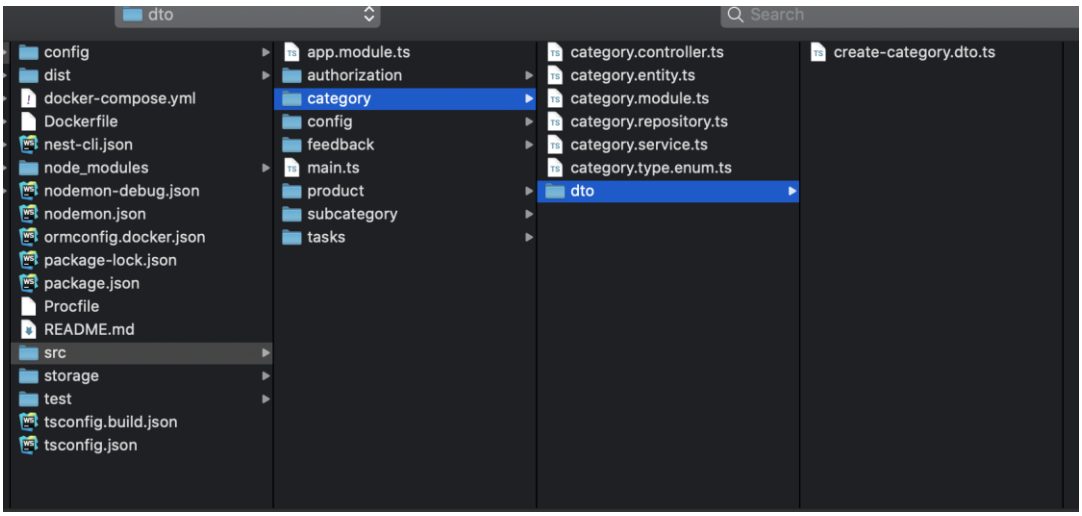

Рисунок 3.2 – Структура проекту

3.2 Логічна реалізація бази даних

Проаналізувавши сутності, перейдемо до реалізації структури БД. Для цього представимо імена необхідних таблиць, атрибутів, типів, їх призначення та обмеження - таблиця. 3.1.

Таблиця 3.1 Структура бази даних

Ім'я таблиці	Поле	Зміст	Тип	Ключі	Обмеження
user	id	ідентифікатор користувача	int	PK	NOT NULL
	username	логін користувача	varchar		NOT NULL
	password	пароль	varchar		NOT NULL
	salt	хеш для пароля	varchar		NOT NULL
	email	електрона адреса	varchar		NOT NULL
category	id	ідентифікатор категорії	int	PK	NOT NULL
	name_ua	ім'я категорії	varchar		NOT NULL
	name_rus	ім'я категорії	varchar		NOT NULL
	name_eng	ім'я категорії	varchar		NOT NULL
	subcategoty_id	ідентифікатор підкатегорії	int	FK	NULL
sub_category	id	ідентифікатор підкатегорії	int	PK	NOT NULL
	name_ua	ім'я підкатегорії	varchar		NOT NULL
	name_rus	ім'я підкатегорії	varchar		NOT NULL
	name_eng	ім'я підкатегорії	varchar		NOT NULL
	category_id	ідентифікатор категорії	int	FK	NOT NULL
product	id	ідентифікатор продукту	int	PK	NOT NULL
	name_ua	ім'я продукту	varchar		NOT NULL
	name_rus	ім'я продукту	varchar		NOT NULL
	name_eng	ім'я продукту	varchar		NOT NULL
	materials_ua	склад продукту	varchar		NULL
	materials_rus	склад продукту	varchar		NULL
	materials_eng	склад продукту	varchar		NULL
	description_ua	опис продукту	varchar		NULL
	description_rus	опис продукту	varchar		NULL
	description_eng	опис продукту	varchar		NULL
	subcategory_id	ідентифікатор підкатегорії	varchar	FK	NOT NULL
feedback	id	ідентифікатор відгуку	int	PK	NOT NULL
	created_at	дата створення	date		NOT NULL
	description	відгук	varchar		NOT NULL
	prod_id	ідентифікатор продукту	int	FK	NULL

Для створення таблиці, необхідно створити модель, та задати параметри. Відповідно до цього реалізована структура сайту. На прикладі створення таблиці для категорій – існує файл category.entity.ts – рисунок 3.3, у якому є всі данні. В ньому описано кількість, типи та можливі відношення до інших таблиць. А також в майбутньому завдяки цьому файлу отримуємо типізацію для TypeScript.

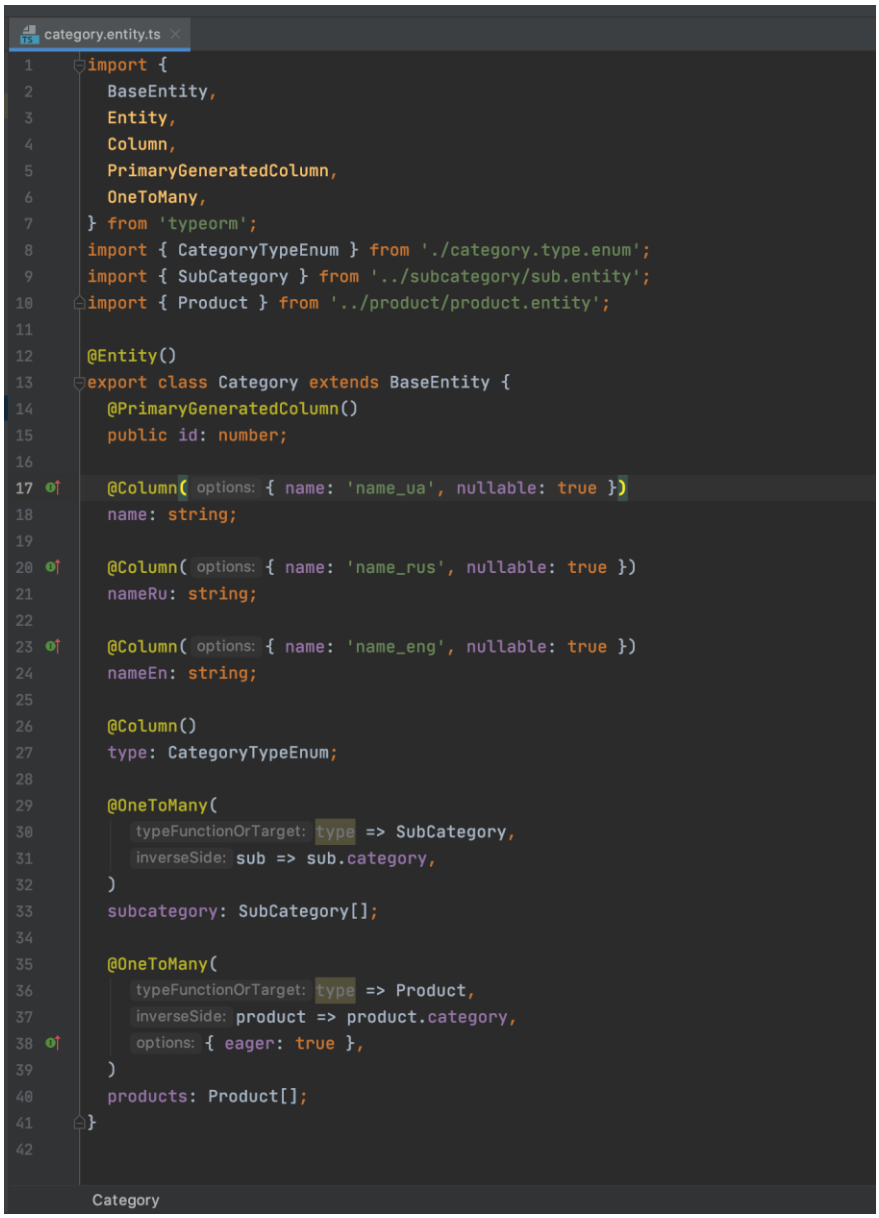

Рисунок 3.3 – Зміст файлу category.entity.ts

Поряд з цим файлом розташований файл - category.controller.ts(рисунок 3.4). Він слугує для контролю всіх запитів таблиці «Категорії». В ньому реалізовані всі роутинги завдяки яким він орієнтується, стосовно категорій – створення категорії, оновлення категорії, видалення, можливі асоціації с іншими таблицями, а також маємо можливість всі ці функції потім експортувати в будь які інші контролери. Наприклад, коли створюємо підкатегорію, то можемо відправивши запит на контролер «Підкатегорій» створити там і категорії(експортувавши контролер), а потім, якщо «категорія» успішно збереглася, то взяти її ідентифікатор(id) і створити вже «Підкатегорію» котра зв’язана з нашою новою «Категорією».

```
@Controller( prefix: 'category')
export class CategoryController {
  constructor(private categoryService: CategoryService) {}

  @Get()
  getCategories(): Promise<Category[]> {
    return this.categoryService.getCategories();
  }

  @Get( path: '/:id')
  getCategory(@Param( property: 'id', ParseIntPipe) id: number): Promise<Category> {
    return this.categoryService.getCategory(id);
  }

  @Post()
  @UsePipes(ValidationPipe)
  createCategory(@Body() categoryValues: CreateCategoryDto): Promise<Category> {
    return this.categoryService.categoryCreate(categoryValues);
  }

  @Post( path: '/update/:id')
  @UsePipes(ValidationPipe)
  updateCategory(
    @Param( property: 'id', ParseIntPipe) id: number,
    @Body() categoryValues: CreateCategoryDto,
  ): Promise<Category> {
    return this.categoryService.categoryUpdate(categoryValues, id);
  }

  @Delete( path: '/:id')
  categoryDelete(@Param( property: 'id', ParseIntPipe) id: number): Promise<boolean> {
    return this.categoryService.categoryDelete(id);
  }
}
```

Рисунок 3.4 – Зміст файлу category.controller.ts

Останній важливий файл – це category.repository.ts(рисунок 3.5), у ньому знаходяться всі резолвери, які стосуються тільки категорій. В ньому проходять всі важливі процеси, наприклад при оновленні категорії, то робимо перевірку на її наявність та валідацію отриманих даних. Після чого повертається помилка, або ж виконується необхідна функція, після чого повертається позитивна відповідь з серверної частини веб-додатку.

```ts
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { CategoryRepository } from './category.repository';
import { Category } from './category.entity';
import { CreateCategoryDto } from './dto/create-category.dto';

@Injectable()
export class CategoryService {
  constructor(
    @InjectRepository(CategoryRepository)
    private readonly categoryRepository: CategoryRepository,
  ) {}

  async getCategories(): Promise<Category[]> {...}

  async getCategory(id: number): Promise<Category> {...}

  async categoryCreate(categoryValues: CreateCategoryDto): Promise<Category> {...}

  async categoryUpdate(
    categoryValues: CreateCategoryDto,
    id: number,
  ): Promise<Category> {...}

  async categoryDelete(id: number): Promise<boolean> {...}
}
```

Рисунок 3.5 – Вміст файлу - category.repository.ts

На наступних рисунках рисунок 3.6 та рисунок 3.7 можна побачити реакцію системи на оновлення категорії з правильно введеними даними та неправильно введеними даними.

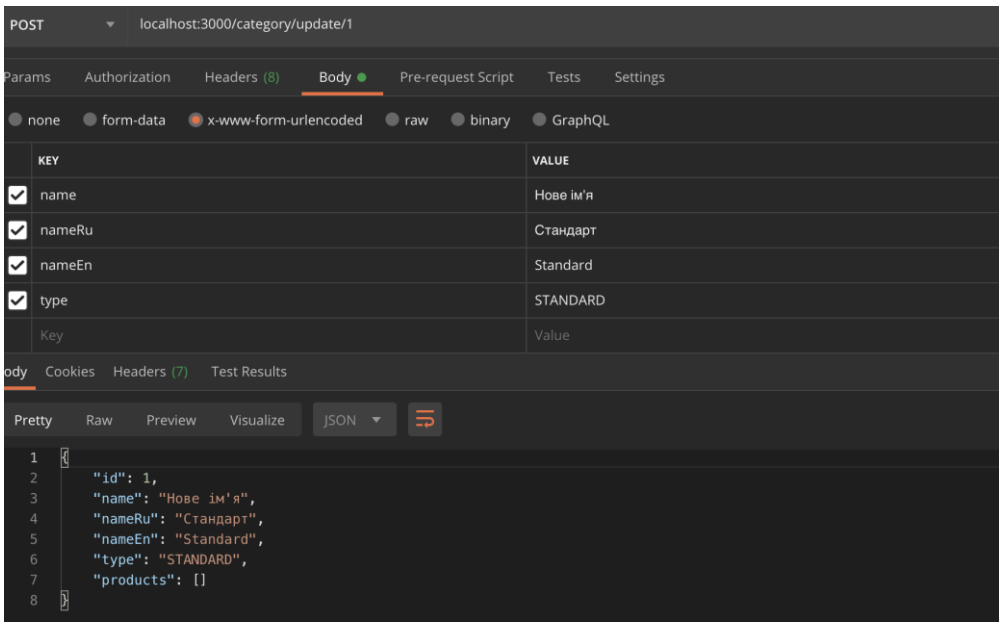

Рисунок 3.6 – Запит та відповідь сервера на вірні данні

Наприклад передаємо id, котрого немає в нашій системі, тобто в базі даних, серверна частина: проаналізувавши, що передали хибний id, повертає нам помилку, в котрій описує де допущено помилку. Також важливим є “Access Token”, без нього більшість запитів не буде виконуватись, за для безпеки важливих даних.

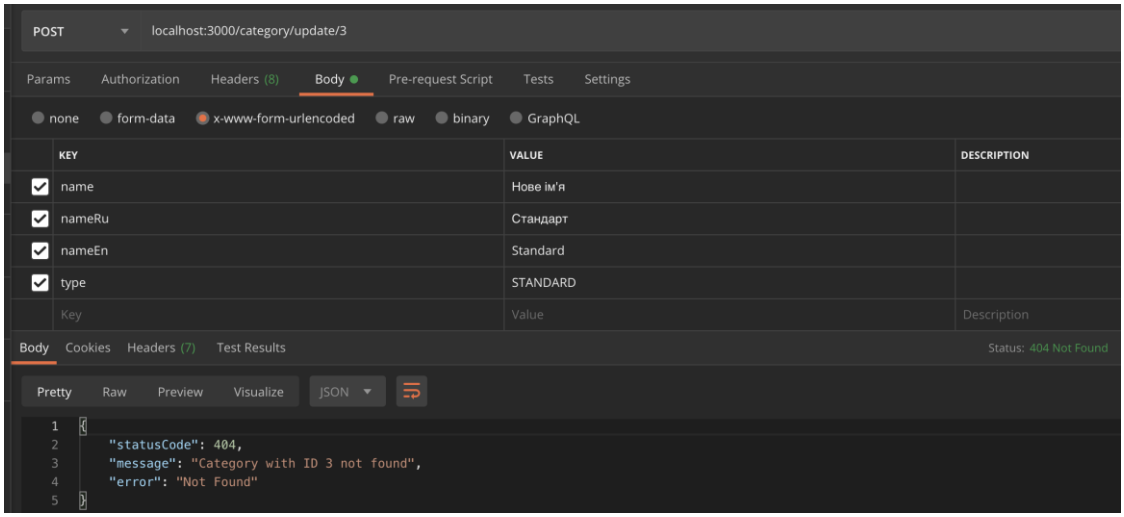

Рисунок 3.7 – Запит та відповідь сервера на невірні данні

Для того, щоб система вважала користувача авторизованим, потрібно додати токен доступу до заголовку запиту у форматі, зображеному на рисунку 3.8

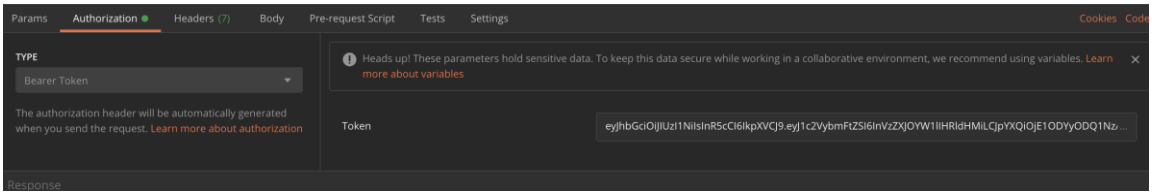

Рисунок 3.8 - Формат заголовку запиту

Також в додатках А-Д – реалізовано завантаження однієї або декілька зображень на сервер сайту. Додаток Е-Ж – вміщує в себе реалізацію завантаження на клієнтській стороні. Реалізовано процес створення продукту, та завантаження малюнків, а також асоціація між категорією та підкатегорією.

3.3 Інтерфейс веб-додатку

Одним із важливих пунктів розробки є також дизайн веб-додатку. Головна ціль, це створення сучасного, зручного інтерфейсу користувача. А ще, можемо додати такій пункт як адаптивність. Якщо раніше для виходу в Інтернет використовувалися в 99% тільки ПК або ноутбуки, то в наш час пристроїв з можливістю вийти в Інтернет стало більше: смартфони, планшети і навіть певні смарт-годиники. Щоб користувачам цих пристроїв було комфортно переглядати сайт, варто подумати про застосування адаптивної верстки. Щоб розробити дизайн сайту недостатньо просто встановити на своєму персональному комп'ютері потужний графічний редактор. Потрібно ще й освоїти основи, а щоб отримати дійсно привабливий дизайн, необхідно також освоїти основи веб-розробки, інтернет-маркетингу та психології. Розробити дизайн сайту - це непросте завдання, адже сайт повинен бути не тільки красивим, але і функціональним, повинен залишати приємне враження у відвідувачів. Цьому сприяє правильний вибір фону сайту, шрифтів, колірної гами, декоративних елементів, грамотне розміщення всіх елементів сторінок.

Так, наприклад, вибір фону сайту багато в чому залежить від того, чи буде дизайн сайту «гумовим» або ж «жорстким». При «жорсткому» дизайні контент сайту поміщається в колонки фіксованої ширини. Якщо ширина екрану монітора перевищують встановлену ширину таких фіксованих колонок, то з'являється багато «порожнього простору». Про те, як буде виглядати такий простір, слід подумати заздалегідь.

Щоб розробити дизайн сайту слід пам'ятати також, що у переважної більшості інтернет-користувачів встановлені лише стандартні шрифти. Шрифти ж, розроблені, наприклад, компанією Adobe, є платними і не всі користувачі можуть дозволити їх собі. Використання подібних платних шрифтів у дизайні сайту недоцільно. Дизайн для велихик екранів інформаційного веб-ресурсу «FASAD» та його адмін-панелі зображено на рисунках 3.9 - 3.13.

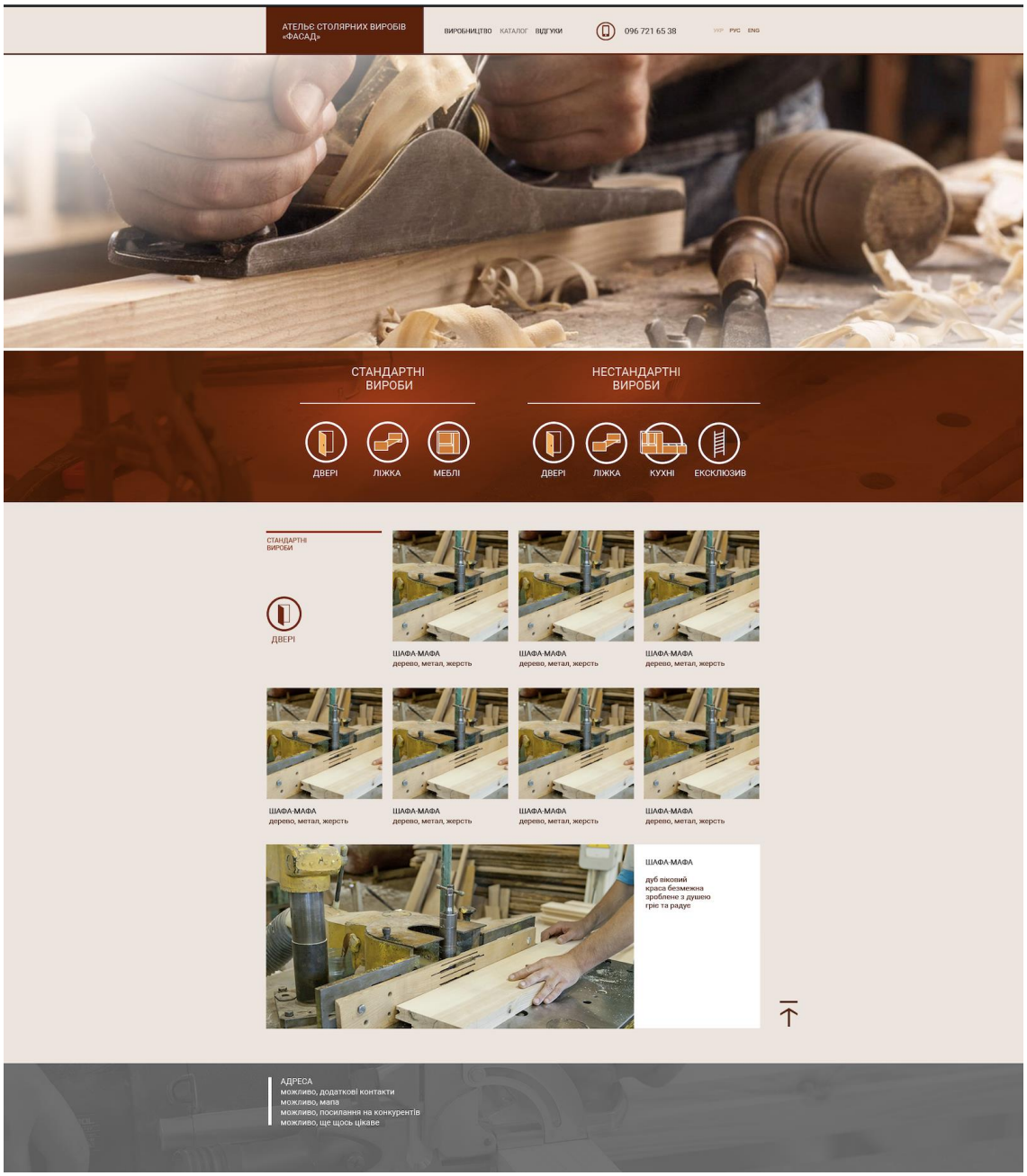

Рисунок 3.9 – Каталог товарів

Дизайн виконаний в сучасному форматі, а саме головне інтуїтивно зрозумілий для більшості. Має гарно виражені картинки(товари), фільтри, а також при великій кількості товарів, є стрілка «піднятись вгору».

Відгуки – це також важлива сторінка, на ній користувач може ознайомитись з якістю товару, оцінити рівень обслуговування магазину.

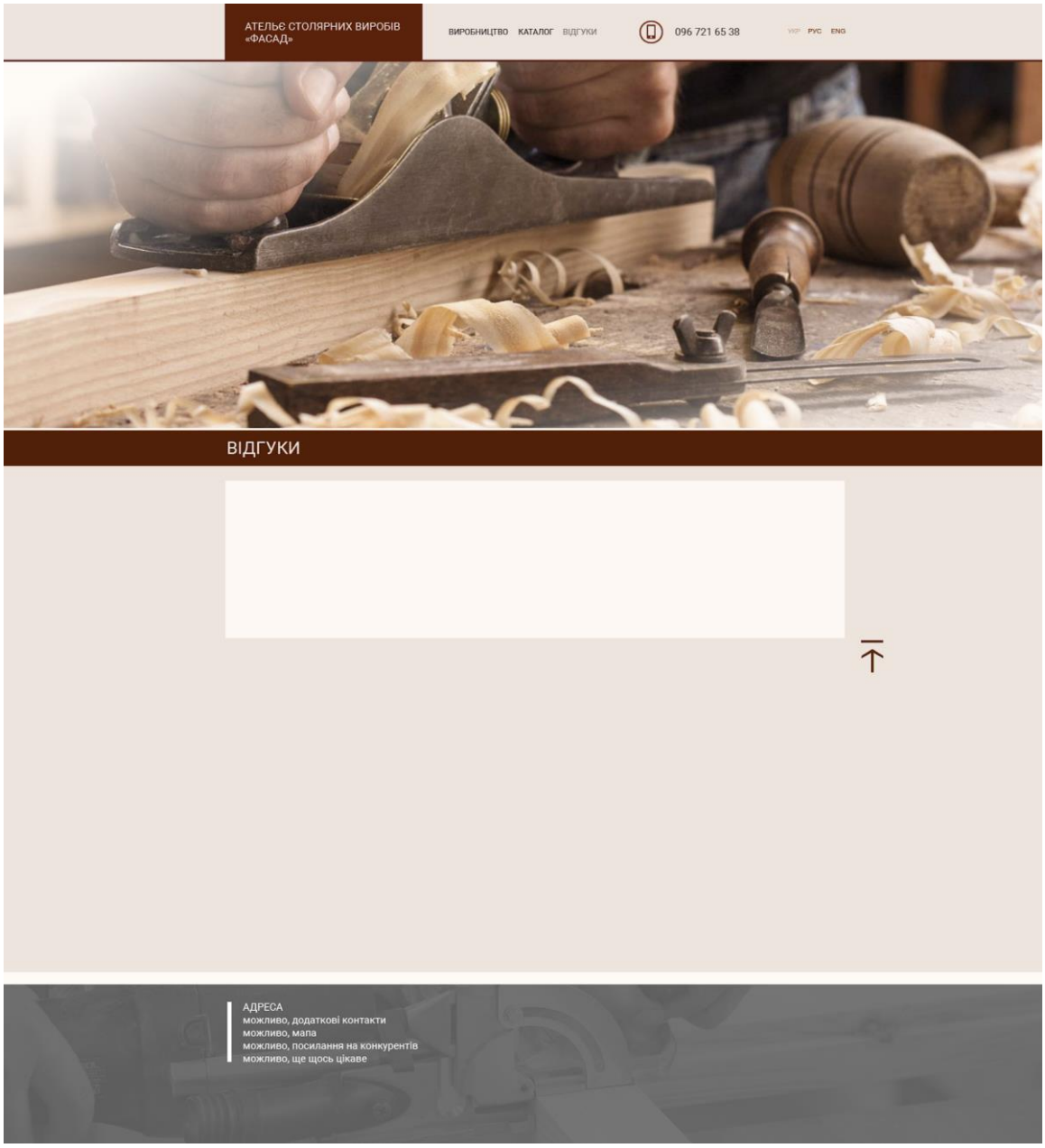

Рисунок 3.10 – Сторінка з відгуками

Головна сторінка, має гарний дизайн. В розділі «Майстерня» - є певний опис товарів, матеріалів, терміну виготовлення. Також чотири зображення їхніх робіт. В розділі «Галерея» буде також опис товарів, матеріали, та певна кількість готових робіт компанії «FASAD». Також внизу буде розміщена адреса компанії.

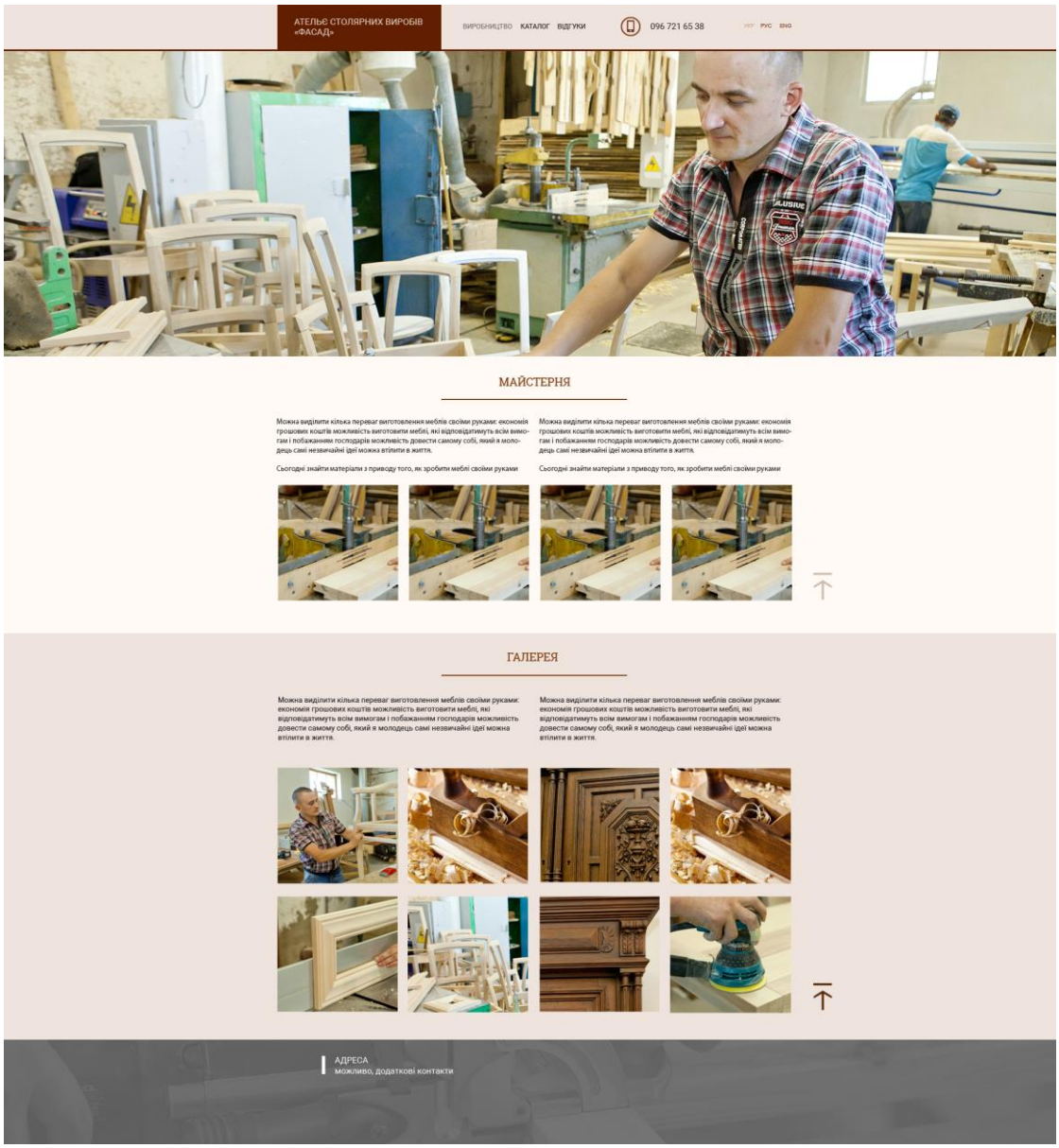

Рисунок 3.11 – Головна сторінка

Тепер, переходимо до адмін-панелі, рисунок 3.12. В ній не потрібно «привабливий» дизайн. Саме головне це простота та зрозумілість. Вгорі, в правому боці, знаходиться зміна мови, зліва – меню «бургер», а також назва сторінки, на якій користувач розташований. Кожна таблиця з серверної частини, наприклад категорія, буде виглядати в форматі таблиці, в якій зможемо редагувати, видалити, або ж подивитись детально.

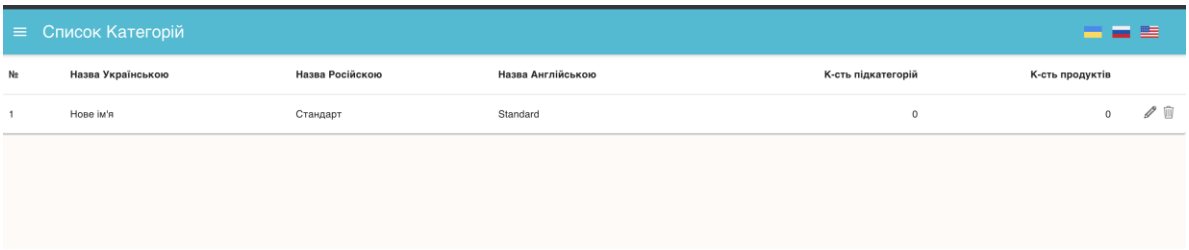

Рисунок 3.12 – Вигляд адмін-панелі, сторінка «Категорії»

На рисунку 3.13 – зображено всі сторінки адмін-панелі та кнопку «вихід» - тобто вийти з аккаунту адмін-панелі, та потім автоматичний перехід на сайт.

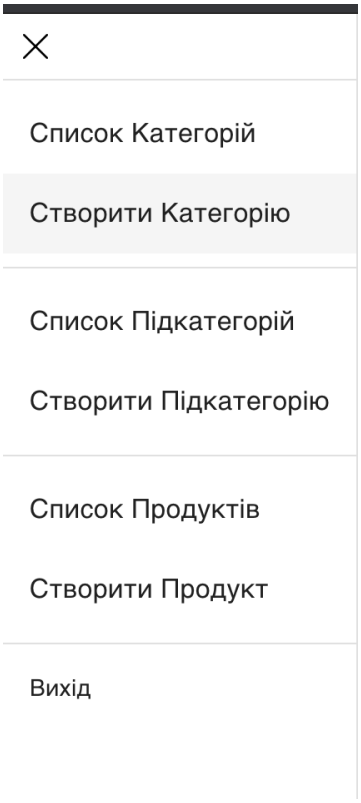

Рисунок 3.13 – Меню адмін-панелі

3.4 Тестування веб-додатку

Тестування сайту - один з важливих життєвих етапів, після якого, надається замовнику готовий проект без помилок, з хорошою читабельністю, яка сприймається легкістю, зручністю і надійністю. Тестування - це відхилення фактичного результату від очікуваного, іншими словами - це процес пошуку помилок. Основні правила тестування веб-сайтів - це кроки, які показують

користувачеві, наскільки зручний і логічний буде проект, наскільки просто і можливо знайти ту чи іншу інформацію. Чи добре сприймається людському погляду і чи правильно працює весь функціонал даного сайту, який був поставлений по ТЗ.

Було протестовано сайт - на «Usability» сайту. Тобто, було перевірено всі кнопки, сторінки, поля введення. Також перевірка на правильність тексту, тобто перевірити, чи немає сайт граматичні чи орфографічні помилки. Після чого була перевірка також на сумісность. Сайт перевіряли з різних пристроїв, та різних ОС.

Тестування навантаження проводилось, для розуміння на який потік можна розраховувати. Програм для тестування досить багато, я вирішив використовувати Apache Benchmark. Тому що, він легкий в використанні та виконує всі необхідні функції. Виконалось 250 000 запитів за 0.215 секунди. Така кількіть запитів була взята, для того, щоб розуміти як сайт себе поведе при DDoS нападі. Запит був на отримання списку категорій згідно дослідження можна зробити висновок, що сервер здатний на 2315 відповідей за секунду. Симулювалося тестування 500 користувачами, які відправляли 500 запитів одночасно. Результати зображені на рисунку 3.14.

```
Server Software:
Server Hostname:        localhost
Server Port:            3000

Document Path:          /graphql
Document Length:        18 bytes

Concurrency Level:      500
Time taken for tests:   0.216 seconds
Complete requests:      500
Failed requests:        0
Non-2xx responses:      500
Keep-Alive requests:    0
Total transferred:      78500 bytes
HTML transferred:       9000 bytes
Requests per second:    2317.51 [#/sec] (mean)
Time per request:       215.749 [ms] (mean)
Time per request:       0.431 [ms] (mean, across all concurrent requests)
Transfer rate:          355.32 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0   17   7.3     17      31
Processing:    15  157  48.7    174     178
Waiting:        1   82  48.0     83     163
Total:         32  175  44.5    190     199

Percentage of the requests served within a certain time (ms)
  50%    190
  66%    192
  75%    193
  80%    193
  90%    194
  95%    194
  98%    197
  99%    198
 100%    199 (longest request)
```

Рисунок 3.14 – Результат тестування навантаженням

3.5 Розгортання додатка

Розгортання відбувається за допомогою Docker – бази даних PostgreSQL, Nodemon - серверної частини та NPM та Express – для клієнтської частини. Вивід консолі можна побачити на рисунку 3.15, 3.16, 3.17. На рисунку 3.15 зображено, що виконалось розгортання БД в контейнері Docker і тепер маємо доступ через порт 5432.

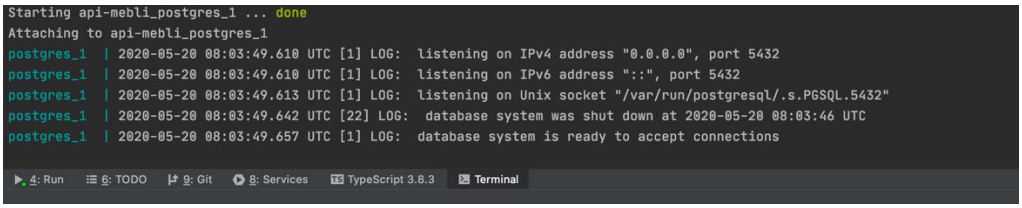

Рисунок 3.15 – Вивід при розгортанні БД

При запуску клієнтської частини, Express буде звертатися до файлу .babelrc в корені проекту, в якому зберігається конфігурація і список використовуваних налаштувань та плагінів. Також важливим файлом на проекті є server.js, він слугує для сполучення клієнстької частини та серверної частини. Завдяки чому, клієнтська частина буде в доступі через порт 4000, а всі запити з сайту будуть переадресовані на 3000 порт і після чого повернеться відповідь на певний запит. У самому кінці виконається певний «build» проекту, і якщо не було помилок то сайт стане доступний на localhost:4000.

Рисунок 3.16 – Вивід при розгортанні клієнтської частини додатка

На рисунку 3.17 зображено розгортання серверної частини(API) та те що воно виконалось успішно, воно завдяки Nodemon проаналізувало всі папки в проекті, дістало всі контроллери, та відкрилось для нас на 3000 порті, через який маємо змогу виконувати всі REST запити, та отримувати відповіді з серверу.

Nodemon - це утиліта, яка буде стежити за будь-якими змінами у вашому джерелі. І при кожній зміні файлу(-ів), вона автоматично буде виконувати перезавантаження всього додатку(сервера).

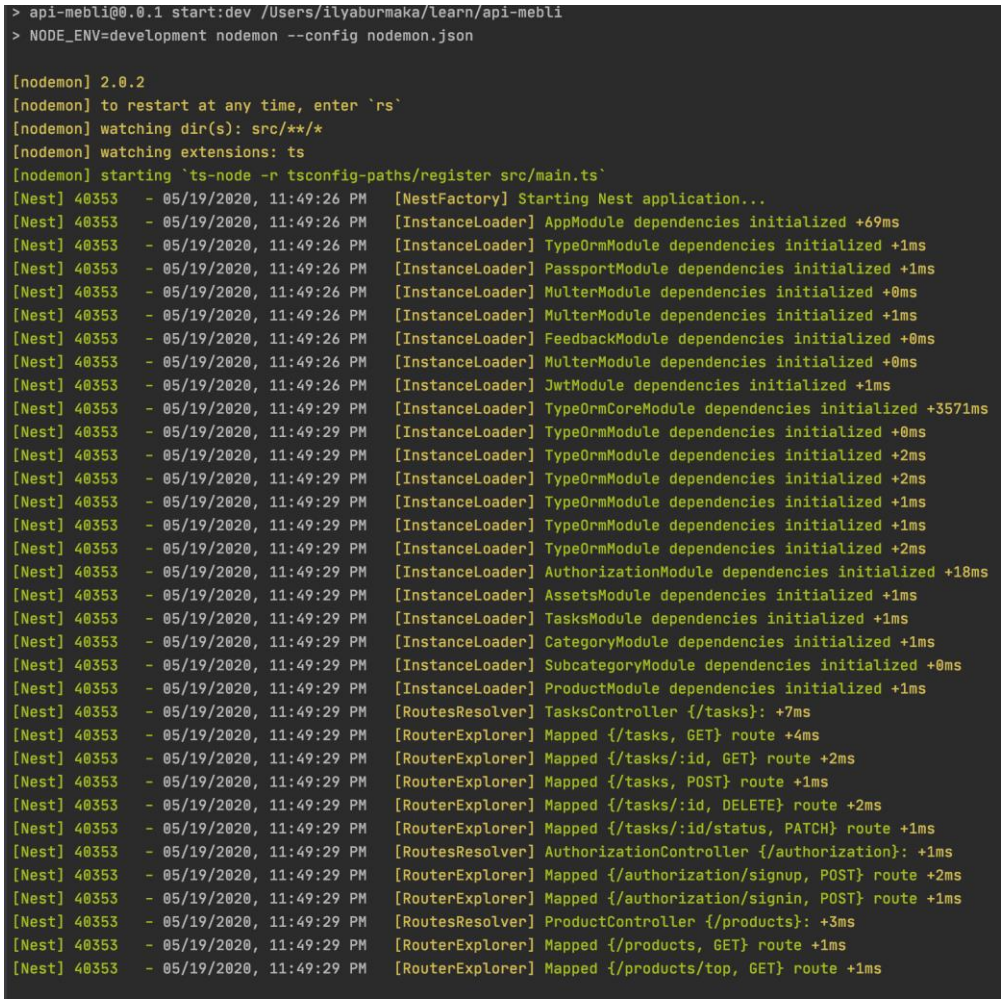

Рисунок 3.17– Вивід при розгортанні серверної частини додатка

ВИСНОВКИ

За результатом виконаної роботи було реалізовано поставлене завдання - розробка інформаційного веб-ресурс «FASAD». Розглянуто можливі варіанти реалізації та приклади подібних рішень. Обґрунтовано вибір програмної реалізації. Створено та реалізовано прототип дизайну веб-ресурсу. У ході розробки та аналізу використовувались сучасні технології та методи проектування програмного забезпечення. Завдяки сучасним технологіям, програмний додаток буде простіше підтримувати. Актуальність даної роботи полягає в покращенні та полегшенні для ФОП «FASAD» - реалізацію їхньої продукції.

СПИСОК ЛІТЕРАТУРИ

1 Хавербеке М. Выразительный JavaScript. Современное веб-программирование. 3-е издание. - Питер:Издательский дом «Питер», 2019. - 480 с.

2 Кара-Ушанов В. Ю. SQL — язык реляционных баз данных. – Екатеринбург:Издательство Уральского университета, 2016. - 156 с.

3 Розенталс Н. Изучаем TypeScript 3. – Москва:ДМК-Пресс, 2019. - 624 с.

4 Фаулер М. Рефакторинг: улучшение существующего кода. – Питер:Диалектика, 2019. – 464 с.

5 Швец А. Погружение в паттерны проектирования. - Интернет-издание, 2018. – 406 с.

6 Азат М. React быстро. – Питер:Издательский дом «Питер», 2016. – 560 с.

7 Чиннатамби К. Изучаем React. - Москва:ЭКСМО, 2019. – 368 с.

8 Сухов К. К. Node.js. Путеводитель по технологии. – Москва:ДМК-Пресс, 2015. – 416 с.

9 Морето С. Bootstrap в примерах. – Москва:ДМК-Пресс, 2017. – 314 с.

10 Уэйншенк С. 100 новых главных принципов дизайна. Как удержать внимание. – Питер:Издательский дом «Питер», 2016. – 288 с.

11 Янг А., Мек Б., Кантелон М. Node.js в действии. – Питер: Издательский дом «Питер», 2018. - 432 с.

12 Васильев А. JavaScript в примерах и задачах. - Москва:ЭКСМО, 2018. – 720 с.

13 Браун И. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript. – Питер:Издательский дом «Питер», 2016. – 336 с.

14 Лучано М. Шаблоны проектирования Node.JS. - Москва:ДМК-Пресс, 2017 – 396 с.

15 Лузанов П., Рогов Е., Лёвшин И. PostgreSQL для начинающих. – Питер:БХВ-Петербург, 2018. – 116с.

16 Rozentals N. Mastering TypeScript 3 – Third Edition. - Birmingham:Packt Publishing, 2019. – 694c.

17 Чёрный Б. Programming TypeScript: Making Your JavaScript Applications Scale. - Sebastopol:O'Reilly Media, 2019. – 324 с.

18 Стефанов С. Книга Javascript.Шаблоны. - Москва:Символ-плюс, 2018. – 272 с.

19 Офіційний опис та документація фреймворку React – https://reactjs.org

20 Офіційний опис та документація TypeORM – https://typeorm.io

21 Офіційний опис та документація Node.js. – https://v8.dev

ДОДАТКИ

Додаток А

```
import {
  BaseEntity,
  Entity,
  PrimaryGeneratedColumn,
  Column,
  ManyToOne,
  OneToOne,
} from 'typeorm';
import { Product } from '../product/product.entity';
import { SubCategory } from '../subcategory/sub.entity';

@Entity()
export class Asset extends BaseEntity {
  @PrimaryGeneratedColumn('uuid')
  public id: string;

  @Column({ type: 'varchar', nullable: false })
  public name: string;

  @Column({ type: 'int', nullable: false })
  public size: number;

  @Column({ name: 'content_type', type: 'varchar', nullable: false })
  public contentType: string;

  @Column({ type: 'varchar', nullable: false })
  public url: string;

  @OneToOne(
    type => SubCategory,
    subcategory => subcategory.photo,
  )
  subcategory: SubCategory;

  @ManyToOne(
    type => Product,
    product => product.photos,
    { eager: true, onDelete: 'CASCADE', onUpdate: 'CASCADE' },
  )
  product: Product;
}
```

Додаток Б

```
import {
  Controller,
  Get,
  InternalServerErrorException,
  Param,
  Post,
  Res,
  UploadedFile,
  UploadedFiles,
  UseInterceptors,
} from '@nestjs/common';
import { FileInterceptor, FilesInterceptor } from '@nestjs/platform-express';
import { extname } from 'path';
```

```
import { diskStorage } from 'multer';
import { AssetsService } from './assets.service';

export const imageFileFilter = (req, file, callback) => {
 if (!file.originalname.match(/\.(jpg|jpeg|png|gif)$/)) {
  return callback(new Error('Only image files are allowed!'), false);
 }
 callback(null, true);
};

export const editFileName = (req, file, callback) => {
 const name = file.originalname.split('.')[0];
 const fileExtName = extname(file.originalname);
 const randomName = Array(4)
  .fill(null)
  .map(() => Math.round(Math.random() * 16).toString(16))
  .join('');
 callback(null, `${name}-${randomName}${fileExtName}`);
};
@Controller('assets')
export class AssetsController {
 constructor(private assetsService: AssetsService) {}

 @Post('/file')
 @UseInterceptors(
  FileInterceptor('image', {
   storage: diskStorage({
    destination: './files',
    filename: (req, file, cb) => {
     const randomName = Array(32)
      .fill(null)
      .map(() => Math.round(Math.random() * 16).toString(16))
      .join('');
     return cb(null, `${randomName}${extname(file.originalname)}`);
    },
   }),
  }),
 )
 async uploadedFile(@UploadedFile() image) {
  return this.assetsService.saveFile(image);
 }

 @Post('files')
 @UseInterceptors(
  FilesInterceptor('image', 5, {
   storage: diskStorage({
    destination: './files',
    filename: editFileName,
   }),
   fileFilter: imageFileFilter,
  }),
 )
 uploadFile(@UploadedFiles() files): any {
  return this.assetsService.saveFiles(files);
 }

 @Get('/:imgpath')
 seeUploadedFile(@Param('imgpath') image, @Res() res) {
  return res.sendFile(image, { root: './files' });
 }
}
```

Додаток В

```
import { Module } from '@nestjs/common';
import { AssetsController } from './assets.controller';
import { AssetsService } from './assets.service';
import { MulterModule } from '@nestjs/platform-express';
import { TypeOrmModule } from '@nestjs/typeorm';
import { AuthorizationModule } from '../authorization/authorization.module';
import { AssetsRepository } from './assets.repository';

@Module({
 imports: [
  MulterModule.register({
   dest: './files',
  }),
  TypeOrmModule.forFeature([AssetsRepository]),
  AuthorizationModule,
 ],
 controllers: [AssetsController],
 providers: [AssetsService],
})
export class AssetsModule {}
```

Додаток Г

```
import { EntityRepository, Repository } from 'typeorm';
import { InternalServerErrorException } from '@nestjs/common';
import { Asset } from './asset.entity';

@EntityRepository(Asset)
export class AssetsRepository extends Repository<Asset> {
 async saveAsset(image: any): Promise<any> {
  const asset = new Asset();
  asset.contentType = image.mimetype;
  asset.name = image.filename;
  asset.size = image.size;
  asset.url = image.path;

  try {
   await asset.save();
   return asset;
  } catch (e) {
   throw new InternalServerErrorException();
  }
 }

 async saveFiles(images: any[]): Promise<any> {
  let data = [];
  for (let i = 0; i < images.length; i++) {
   const res = await this.saveAsset(images[i]);
   data = [...data, res.id];
  }

  return await data;
 }
```

Додаток Д

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { AssetsRepository } from './assets.repository';
```

```
@Injectable()
export class AssetsService {
 constructor(
  @InjectRepository(AssetsRepository)
  readonly assetsRepository: AssetsRepository,
 ) {}

 async saveFile(image: any): Promise<any> {
  return await this.assetsRepository.saveAsset(image);
 }

 async saveFiles(image: any): Promise<any> {
  return await this.assetsRepository.saveFiles(image);
 }

 async getAsset(imageId: any): Promise<any> {
  const found = await this.assetsRepository.findOne(imageId);

  return found;
 }
}
```

Додаток Е

```
import * as React from 'react'
import RaisedButton from 'material-ui/RaisedButton'
import styled from 'styled-components'
import { useFieldArray, useForm } from 'react-hook-form'
import axios from 'axios'
import { CurrentContext } from '../../../contexts/currentContext'

const Row = styled.div`
 display: flex;
 flex-direction: column;
`

const Label = styled.div`
 font-family: 'MyriadPro';
 text-transform: uppercase;
 color: #535353;
`

const CustomInput = styled.input`
 width: 280px;
 border: 1px solid darkgray;
 border-radius: 8px;
 height: 35px;
 padding: 3px;
 font-size: 16px;

 :focus {
  outline: none;
 }
`
const CustomSelect = styled.select`
 width: 280px;
 border: 1px solid darkgray;
 border-radius: 8px;
 height: 35px;
 padding: 3px;
 font-size: 16px;
```

```
  :focus {
    outline: none;
  }
`
const Form = styled.form`
  display: flex;
  flex-direction: column;
  height: calc(100vh - 65px);
  justify-content: center;
`

const tableLanguage = {
  ua: {
    name: 'Назва Українською',
    nameSecond: 'Назва Російскою',
    nameThird: 'Назва Англійською',
    description: 'Опис Українською',
    descriptionSecond: 'Опис Російською',
    descriptionThird: 'Опис Англійською',
    countProd: 'К-сть продуктів',
    reqMessage: "Це поле є обов'язковим",
    type: 'Розділ',
    btnName: 'Зберегти'
  },
  ru: {
    name: 'Имя по Украински',
    nameSecond: 'Имя по Русском',
    nameThird: 'Имя по Английскому',
    description: 'Описание по Украински',
    descriptionSecond: 'Описание по Русском',
    descriptionThird: 'Описание по Английскому',
    countProd: 'К-во товаров',
    reqMessage: 'Поле долдно быть заполненым',
    type: 'Тип',
    btnName: 'Сохранить'
  },
  en: {
    name: 'Name in Ukrainian',
    nameSecond: 'Name in Russian',
    nameThird: 'Name in English',
    description: 'Description in Ukrainian',
    descriptionSecond: 'Description in Russian',
    descriptionThird: 'Description in English',
    countProd: 'Count of product',
    reqMessage: 'This field is required',
    type: 'Type',
    btnName: 'Submit'
  }
}

const CreateCategoryForm = ({ onSubmit, defaultValues }) => {
  const { uploadFiles, createProduct, categories } = React.useContext(CurrentContext)
  const [data, setData] = React.useState([])
  const [catId, setId] = React.useState(null)
  const { register, handleSubmit, errors, setValue, control, getValues } = useForm({ defaultValues })
  const { currentLang } = React.useContext(CurrentContext)

  const getData = async () => {
    const response = await categories()
    setData(response)
    setId(response[0].id)
  }
```

```
const handleFg = async () => {
  const dat1 = getValues()
  console.log(dat1)
  setId(dat1.categoryId)
}

React.useEffect(() => {
  getData()
  handleFg()
}, [])

const handleChange = event => {
  if (event && !!event.target) {
    setValue('file', event.target.files[0])
  }
}
const { fields, append, remove } = useFieldArray({
  control,
  name: 'files'
})

return (
  <Form onSubmit={handleSubmit(onSubmit)}>
    <Row>
      <Label>{tableLanguage[`${currentLang}`]?.name}</Label>
      <CustomInput name='name' type='text' ref={register({ required: true })} />
      {errors.name && <span>{tableLanguage[`${currentLang}`]?.reqMessage}</span>}
    </Row>
    <Row>
      <Label>{tableLanguage[`${currentLang}`]?.nameSecond}</Label>
      <CustomInput name='nameRu' type='text' ref={register({ required: true })} />
      {errors.nameRu && <span>{tableLanguage[`${currentLang}`]?.reqMessage}</span>}
    </Row>
    <Row>
      <Label>{tableLanguage[`${currentLang}`]?.nameThird}</Label>
      <CustomInput name='nameEn' type='text' ref={register({ required: true })} />
      {errors.nameEn && <span>{tableLanguage[`${currentLang}`]?.reqMessage}</span>}
    </Row>
    <Row>
      <Label>{tableLanguage[`${currentLang}`]?.description}</Label>
      <CustomInput name='description' type='text' ref={register({ required: true })} />
      {errors.nameEn && <span>{tableLanguage[`${currentLang}`]?.reqMessage}</span>}
    </Row>
    <Row>
      <Label>{tableLanguage[`${currentLang}`]?.descriptionSecond}</Label>
      <CustomInput name='descriptionRus' type='text' ref={register({ required: true })} />
      {errors.nameEn && <span>{tableLanguage[`${currentLang}`]?.reqMessage}</span>}
    </Row>
    <Row>
      <Label>{tableLanguage[`${currentLang}`]?.descriptionThird}</Label>
      <CustomInput name='descriptionEng' type='text' ref={register({ required: true })} />
      {errors.nameEn && <span>{tableLanguage[`${currentLang}`]?.reqMessage}</span>}
    </Row>
    <Row>
      <Label>Materials</Label>
      <CustomInput name='materials' type='text' ref={register({ required: true })} />
      {errors.nameEn && <span>{tableLanguage[`${currentLang}`]?.reqMessage}</span>}
    </Row>
    <Row>
      <Label>Materials Rus</Label>
      <CustomInput name='materialsRus' type='text' ref={register({ required: true })} />
```

```
      {errors.nameEn && <span>{tableLanguage[`${currentLang}`]?.reqMessage}</span>}
    </Row>
    <Row>
      <Label>Materials Eng</Label>
      <CustomInput name='materialsEng' type='text' ref={register({ required: true })} />
      {errors.nameEn && <span>{tableLanguage[`${currentLang}`]?.reqMessage}</span>}
    </Row>
    <Row>
      <Label>Category</Label>
      <CustomSelect name='categoryId' ref={register({ required: true })} onChange={handleFg}>
        {data.map((category, index) => (
          <option value={category.id}>{category.name}</option>
        ))}
      </CustomSelect>
      {errors.type && <span>{tableLanguage[`${currentLang}`]?.reqMessage}</span>}
    </Row>
    <Row>
      <Label>Subcategory</Label>
      <CustomSelect name='subcategoryId' ref={register({ required: true })}>
        {data.map((category, index) => {
          if (category.id.toString() === catId.toString()) {
            console.log(catId, category, '111')

            return (
              <>
                {category.subcategory.map(subdata => (
                  <option value={subdata.id}>{subdata.name}</option>
                ))}
              </>
            )
          }
        })}
      </CustomSelect>
      {errors.type && <span>{tableLanguage[`${currentLang}`]?.reqMessage}</span>}
    </Row>
    <Row>
      <Label>Files</Label>
      {fields.map((field, index) => (
        <div key={field.id}>
          <input name={`files[${index}]`} type='file' onChange={handleChange} ref={register()} />
          <button onClick={() => remove(index)}>Delete</button>
        </div>
      ))}
    </Row>
    <section>
      <button type='button' onClick={() => append({ name: 'files' })} disabled={fields.length >= 5}>
        Add photo
      </button>
    </section>
    <RaisedButton label={tableLanguage[`${currentLang}`]?.btnName} type='submit' primary={true}
style={style} />
   </Form>
  )
}
const style = {
  margin: '15px 0px',
  width: '100%'
}

export default CreateCategoryForm
```

Додаток Ж

```
import * as React from 'react'
import Cookies from 'js-cookie'
import axios, { get, post } from 'axios'
import omit from 'lodash/omit'
import Router from 'next/router'

export const CurrentContext = React.createContext(null)

const CurrentProvider = ({ children }) => {
 const [currentLang, setLang] = React.useState(null)
 const token = localStorage?.getItem('accessToken')
 const token = {}
 const headerContent = {
  headers: {
   'Content-Type': 'multipart/form-data'
  }
 }

 const link = 'https://afternoon-scrubland-24663.herokuapp.com/'
 React.useEffect(() => {
  setLang(Cookies.get('lang') || 'ua')
 }, [Cookies.get('lang')])

 const onLangClick = value => {
  Cookies.set('lang', value, { expires: 100 })
  setLang(value)
 }

 const categories = async token => {
  try {
   const { data } = await axios.get(link + `category`, {}, { token })
   return data
  } catch (e) {
   console.log('CurrentProvider.categories', e)
  }
 }

 const getOneCategory = async id => {
  try {
   const { data } = await axios.get(link + `category/${id}`, {}, { token })
   return data
  } catch (e) {
   console.log('CurrentProvider.categories', e)
  }
 }

 const createCategory = async variables => {
  try {
   await post(link + 'category', { ...variables })
   Router.push('/super-admin/categories')
  } catch (error) {
   console.log('CurrentProvider.createCategory: ', error)
  }
 }

 const updateCategory = async (id, variables) => {
  try {
   await post(link + `category/update/${id}`, { ...variables })
   Router.push('/super-admin/categories')
  } catch (error) {
```

```
    console.log('error', error)
  }
}

const removeCategory = async (id, token) => {
  try {
    await axios.delete(link + `category/${id}`, {}, { token })
  } catch (e) {
    console.log('CurrentProvider.removeCategory', e)
  }
}

const subcategories = async () => {
  try {
    const { data } = await axios.get(link + `subcategory`, {}, { token })
    return data
  } catch (e) {
    console.log('CurrentProvider.subcategories', e)
  }
}

const createSubcategory = async (variables, photoId) => {
  try {
    await post(link + `subcategory`, {
      ...omit(variables, ['photo']),
      photoId
    })
    Router.push('/super-admin/subcategories')
  } catch (e) {
    console.log('CurrentProvider.createSubcategory', e)
  }
}

const getOneSubCategory = async id => {
  try {
    const { data } = await get(link + `subcategory/${id}`)
    return data
  } catch (e) {
    console.log('CurrentProvider.getOneSubCategory', e)
  }
}

const updateSubcategory = async (id, variables) => {
  try {
    await post(link + `subcategory/update/${id}`, { ...variables })
    Router.push('/super-admin/subcategories')
  } catch (error) {
    console.log('CurrentProvider.updateSubcategory', error)
  }
}

const removeSubcategory = async id => {
  try {
    return await axios.delete(link + `subcategory/${id}`, {}, { token })
  } catch (e) {
    console.log('CurrentProvider.removeSubcategory', e)
  }
}

const products = async token => {
  try {
    const { data } = await axios.get(link + `products`, {}, { token })
```

```
    return data
  } catch (e) {
    console.log('CurrentProvider.products', e)
  }
}

const createProduct = async (assetIds, data) => {
  try {
    await post(link + `products/create`, {
      ...omit(data, 'files'),
      assetIds
    })
    Router.push('/super-admin/products')
  } catch (e) {
    console.log('CurrentProvider.createProduct', e)
  }
}

const removeProduct = async (id, token) => {
  try {
    await axios.delete(link + `products/${id}`, {}, { token })
  } catch (e) {
    console.log('CurrentProvider.removeProduct', e)
  }
}

const uploadFile = async file => {
  try {
    const uploadedPhoto = await post(link + 'assets/file', file, headerContent)
    return uploadedPhoto
  } catch (e) {
    console.log('CurrentProvider.uploadFile', e)
  }
}

const uploadFiles = async files => {
  try {
    const { data } = await post('http://localhost:3000/assets/files', files, headerContent)

    return data
  } catch (e) {
    console.log('CurrentProvider.uploadFiles', e)
  }
}

const onLogout = () => {
  localStorage.removeItem('accessToken')
  Router.push('/')
}

const value = {
  onLangClick,
  uploadFiles,
  currentLang,
  categories,
  subcategories,
  products,
  createCategory,
  updateCategory,
  removeCategory,
  getOneCategory,
  createSubcategory,
```

```
    getOneSubCategory,
    updateSubcategory,
    removeSubcategory,
    createProduct,
    removeProduct,
    onLogout,
    link,
    uploadFile
  }

  return <CurrentContext.Provider value={value}>{children}</CurrentContext.Provider>
}

export default CurrentProvider
```