Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Панченко Антон Андрійович*
(ПІБ)

академічної групи *122-17-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка та адміністрування back-end частини Інтернет магазину з продажу одягу*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Гуліна І.Г.*			
розділів:				
спеціальний	*доц. Гуліна І.Г.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент	*доц. Шедловський І.А.*			
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-17-1* *Панченко Антон Андрійович*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка та адміністрування back-end частини Інтернет магазину з продажу одягу*

затверджена наказом ректора НТУ «ДП» від «07» червня 2021 р. № 317-с

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Гуліна І.Г.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Панченко А.А.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 81 с., 14 рис., 3 дод., 24 джерела.

Об'єкт розробки: система функціонування та адміністрування інтернет-магазину одягу .

Мета кваліфікаційної роботи: розробка та адміністрування back-end частини Інтернет магазину з продажу одягу.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування системи, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження застосунку, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної підсистеми, проведений підрахунок вартості роботи по створенню застосунку та розраховано час на його створення.

Практичне значення полягає у створенні системи, що забезпечує відвідувачам сайту доступ до каталогу товарів за категоріями, можливість редагування заказу та перегляду особистої інформації, що забезпечує комфортне користування інтернет-магазином . Адміністративна панель магазину дає можливість адміністрації магазину надавати актуальну інформацію щодо наявності та вартості товару.

Актуальність інтернет-магазину визначається великим попитом на онлайн послуги, що оптимізують та спрощують обслуговування клієнтів, сприяють створенню позитивного іміджу компанії.

Список ключових слів: ІНТЕРНЕТ-МАГАЗИН, БАЗА ДАНИХ, АДМІНІСТРАТИВНА ПАНЕЛЬ, ВЕБ-САЙТ, АПІ, БЕКЕНД, ЗАПИТИ.

ABSTRACT

Explanatory note: 81 p., 14 figs, 3 apps, 24 sources.

Object of development: the system of functioning and administration of the online clothing store.

The purpose of the diploma project: development and administration of a back-end part of an online clothing store.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the scope of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes the existing solutions, selects the platform for development, performs design and development of the program, describes the algorithm and structure of the system, determines the input and output data, provides characteristics of the parameters of technical means, describes the call and application load, describes the program .

The economic section determines the complexity of the developed information subsystem, calculates the cost of work to create an application and calculates the time for its creation.

The practical significance is to create a system that provides site visitors with access to the catalog of products by category, the ability to edit the order and view personal information, which provides comfortable use of the online store. The administrative panel of the store allows the store administration to provide up-to-date information on the availability and cost of goods.

The relevance of the online store is determined by the high demand for online services that optimize and simplify customer service, contribute to the creation of a positive image of the company.

Keywords: ONLINE STORE, DATABASE, ADMINISTRATIVE PANEL, WEBSITE, API, BACKEND, REQUESTS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД - база даних;

ІС - інформаційна система;

ПЗ - програмне забезпечення;

ПК - персональний комп’ютер;

СУБД - система управління базами даних;

API - Application Programming Interface;

BE - BackEnd;

CMS - система управління контентом;

DOM - Document Object Model;

IT - інформаційні технології;

TS - Type Script;

ORM - Object-Relational Mapping;

HTML - Hypertext Markup Language.

ВСТУП

Кожен з нас хоча б раз за життя замовляв щось у Інтернеті. За останні часи онлайн-продажи сталі ще більшими, адже зараз онлайн можна замовити усе – від їжі до автомобілів.

Онлайн замовлення з-за кордону ростуть, інтернет-магазини стають більш популярними, ринок розвивається і стає важко розробляти щось нове в інтерфейсі для користувачів. Вагомими факторами успішності веб-додатку є його візуальна частина та швидкість завантаження сторінок. Але не менш важливим є наповнення сайту та логічність отримання даних. У інтернет-магазині користувач може роздивитися товари, побачити доступні розміри та інше. Це дає можливість користувачу отримати максимально повну інформацію про передбачувану покупку, як якщо б людина сам побувала в звичайному магазині.

Завдяки тому, що Інтернет є невичерпним джерелом надання різноманітної інформації, створення сайтів дуже актуально, незалежно від обраної тематики. Але варто приділяти увагу не стільки створенню свого сайту, скільки його наповненню, тому що актуальним буде тільки якісний сайт, який проб'ється в топ видачі пошукової системи, зацікавить користувача і принесе певну вигоду своєму власникові.

У даній кваліфікаційній роботі представлена розробка великої інформаційної системи, розробленої для інтернет-магазину одягу. Даний проект описує зв’язки між усіма рівнями інформації.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Все частіше і частіше за допомогою в якомусь питанні ми звертаємося до Інтернету. Набагато швидше у наш час знайти якусь інформацію не в книжках, а саме у світовій павутині. Актуальність створення сайтів полягає також в тому, що якщо ви хочете донести інформацію до великої кількості людей дуже швидко, то кращим способом буде зробити це через власний сайт. Веб-ресурс дозволяє представити інформацію про компанію та її товари або послуги стисло і одночасно повноцінно. Також сайт може повідомляти про новини фірми, про зміни в цінах або режимі роботи, містити відгуки клієнтів, статистику і усю інформацію.

Різниця роботи між одно- і багатосторінковими сайтами полягає в тому, що перші вантажать усю інформацію одразу і у більшості випадків не мають окремої бази даних. У такому випадку завантаження сторінки буде довшим. Другі у свою чергу довантажують дані відповідно до дій користувача, а саме працюють з базою данних, відправляють та обробляють запити. Навантаження на backend менше і дані вантажаться поступово.

Статичні односторінкові сайти створюються, в основному, на мові розмітки HTML без баз даних та бекенду.

Такий спосіб створення сайтів застарілий та використовується рідко, бо такі системи мають свої недоліки, як розподілення прав доступу чи зміна данних про товар.

Для компенсації недоліків чистого HTML до сайту підключаються різні мови програмування, що дають можливість зробити сайт динамічним та допомагає у роботі з базами данних та комфортної навігації між частинами додатку. Динамічної називають сторінку, що формується сервером з декількох частин або отримується шляхом внесення або зміни даних, які зберігається на

сервері, у заготовку сторінки. Тобто динамічна сторінка, на відміну від статичної, збирається якимось чином з даних, що зберігаються на сервері, і тільки після цього показується відвідувачеві. Динамічним слід називати будь-який сайт, на якому є хоча б одна динамічна сторінка.

Додавання товарів в каталог інтернет-магазину і оновлення інформації про товари, наприклад, при зміні ціни, наявності, характеристики, при імпорті даних з файлів прайс-листів, в порівнянні з ручним керуванням товарами, дозволяє значно скоротити час, необхідний для підтримки каталогу в актуальному стані.

У динамічних сайтів також є недоліки, але не такі суттєві. Першим недоліком динамічних сайтів є необхідність використання додаткових програмних засобів для побудови динамічного сторінок. Але у наш час існує багато різних фрейморків та бібліотек, що полегшують процес розробки. Другим недоліком, що випливають з першого, є складність великих структурних змін сайту. В даному випадку все зав'язано на програмному забезпеченні, що об'єднує шматочки дизайну і даних в один повноцінний сайт. Але у будь-якої програми є свої обмеження, і, якщо необхідно отримати щось, що програмно не передбачено, потрібно змінювати саму структуру проекту.

Перше що потрібно зробити у розробці сайту – це вибрати спосіб зберігання інформаційного наповнення (контенту), що відображається на Web-сторінці. В даному проекті для цих цілей використовується база даних в СУБД PostgreSQL, структурує контент по таблицям, що містять поля і записи з даними про магазин. Таблиці мають свої зв’язки, які дають змогу створювати складніші інформаційні системи. СУБД PostgreSQL - відмінний вибір для створення такої бази даних внаслідок простоти у використанні і адмініструванні, вільної адаптації під різні платформи, включаючи Linux і Windows, і швидко зростаючої популярності.

Після створення Backend починають розроблятися сторінки та уся Frontend частина додатку, яка спілкується з БД запитами. Щоб розробити додатки для взаємодії з базою даних і сторінками сайту, був вибран фреймворк Node.JS.

Застосування концепції динамічних Web-сторінок прискорює роботу розробки і роботи сервера і дає можливість швидше завантажувати дані.

1.2. Призначення розробки та галузь застосування

Думаю, що нікого не треба переконувати в тому, що тільки актуальні, зручні і корисні інтернет-ресурси мають можливість поборотися за ТОП-видачу в ранжируванні пошукових систем, а, значить, за увагу цільової аудиторії.

Як об'єкт розробки цього проекту інтернет-магазину одягу розглядається веб-додаток «Closes Shop», в якому користувач може знайти одяг закордонного виробництва за різними категоріями. Магазин має містити сортування за категоріями, розподіл на жіночий, чоловічий та дитячий одяг, підбір виробів по ряду критеріїв. Також користувач повинен мати змогу переглянути своє замовлення, відкоригувати та безпечно відіслати свої данні адміністратору магазину для формування замовлення.

В свою чергу для адміністратора магазину розробляється спеціальна додаткова панель, у якій менеджер може дивитися та редагувати замовлення, бачити дані клієнта, створювати нові позначки для категорій, редагувати контент магазину. Збереження даних у БД дає можливість структурувати усю інформацію про магазин та користувачів.

Розроблена Backend частина додатку призначена для:

– можливості редагування контенту магазину ;

– забезпечення відвідувачам сайту безпечності обробки інформації при створенні замовлення;

– підвищення продажу одягу магазину за рахунок швидкості роботи інтернет-магазину та швидкості обробки замовлень у адміністративній панелі.

1.3. Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студен виконує кваліфікаційну роботу. Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки «виконання кваліфікаційної роботи» є:

– освітня програма 122 «Комп'ютерні науки»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» №317-с від 07.06.2021р.;

– завдання на кваліфікаційну роботу на тему «Розробка backend частини системи відображення і адміністрування інтернет магазину одягу».

1.4. Постановка завдання

Завданням кваліфікаційної роботи є розробка та адміністрування back-end частини Інтернет магазину з продажу одягу.

Програмне забезпечення призначене для надання універсального інструменту для відображення контенту та адміністрування даного інтернет-магазину.

Програма повинна реалізувати наступні функції:

– надання можливості редагування інформації у адмін-панелі;

– формування web-сторінок на основі шаблону з використанням інформації, отриманної з бази даних;

– відображення сторінок для різних девайсів користувача;

– контактування за адміністраторами данного магазину;

– отримання фідбеку від адміністраторів на пошту;

– оформлення замовлень.

Для досягнення поставленої мети необхідно:

– вивчити предметну галузь розв'язуваної задачі;

– створити алгоритм для реалізації поставленого завдання;

– створити базу даних і клієнтську програму та серверну программу, що працює з нею.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Щоб досягти поставлених цілей розроблене програмне забезпечення повинно підтримувати виконання наступних дій:

– зберігання даних підсистеми в реляційній базі даних;

– надання віддаленого доступу до застосунку через веб-браузер на комп'ютері користувача;

– зчитування вхідних даних з пристроїв, хмарних сервісів, за адресними посиланнями;

– обробка отриманної інформації та передача відповіді користувачу.

Для виконання перерахованих вище функцій та гарної роботи магазину у застосунку повинні бути реалізовані:

– можливість отримати доступ до програми через веб-браузер;

– можливість інтеграції в платформу веб-сайту з продажу одягу;

– наявність типової конфігурації, що забезпечує можливість швидкого введення застосунку в експлуатацію;

– програмно-апаратна переносимість.

1.5.2 Вимоги до інформаційної безпеки

Для уникнення некоректної роботи програми необхідно реалізувати:

– контроль вхідних даних;

– обробку виняткових ситуацій;

– виведення повідомлень про помилки;

– можливість повторного введення даних;

– можливість безперервної роботи протягом не менше 120 годин (5 діб);

– платформну незалежність;

– вірогідність виникнення не більше 2 логічних помилок на 1000 операторів за 1 рік експлуатації;

– забезпечення неушкодженого стану даних у випадку відмови.

1.5.3 Вимоги до складу та параметрів технічних засобів

Для нормального та безперебойного функціонування данного інтернет-магазину необхідно, щоб ПК, на якому, буде функціонувати система інтренет магазину, відповідала наступним вимогам:

– процесор класу Intel Pantium з тактовою частотою не менш 2.4 ГГц та двума ядрами;

– доступ до мережі Internet;

– не менше 16 GB оперативної пам'яті;

– 1 Тб вільного місця на жорсткому диску;

– клавіатура;

– маніпулятор "миша".

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний застосунок буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутими замовником.

1.5.4 Вимоги до інформаційної та програмної сумісності

Для нормального функціонування програми необхідно, щоб програмне забезпечення персонального комп’ютера, на якому буде функціонувати веб-орієнтована система, відповідало наступним вимогам:

- операційна система Windows (7+), Linux, MacOs;
- веб-браузер Firefox / Google Chrome / Opera / Microsoft Edge / Safari.

Backend частина додатку має бути реалізована на мові програмування TypeScript з використанням фреймворку NodeJS та СУБД PostgreSQL.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

За завданням кваліфікаційної роботи було реалізовано розробку backend частини системи відображення і адміністрування інтернет магазину одягу.

Призначення розробленої системи:

– підвищити обсяг продажу імпортного одягу;

– забезпечити просте користування магазином;

– надати можливість користувачам легко обрати товари за різними категоріями та рейтингами.

Розроблена програма реалізує наступні функції:

– надання можливості редагування інформації у адміністративній панелі;

– формування web-сторінок на основі шаблону з використанням інформації, отриманної з бази даних;

– відображення сторінок для різних девайсів користувача;

– контактування за адміністраторами данного магазину;

– отримання фідбеку від адміністраторів на пошту;

– оформлення замовлень.

Для досягнення поставленої задачі розроблене програмне забезпечення підтримує виконання таких операцій:

– запезпечення авторизації користувача для перегляду контенту;

– обробка вхідних даних та контактування з хмарними сервісами та базою даних;

– обробка замовлення та відправка у БД;

– коригування контенту через запити до БД.

– зберігання даних системи в реляційній базі даних.

2.2. Опис застосованих математичних методів

Оскільки особливості предметної галузі розв'язуваної задачі не передбачають застосування математичних методів, при розробці системи відображення та управління контенту інтернет-магазину математичні методи не використовувалися.

2.3. Опис використаних технологій та мов програмування

ВЕ частина web-додатку має бути реалізована на мові програмування TypeScript з використанням фреймворку NodeJS та СУБД PostgreSQL.

Вперше TypeScript з’явився в 2012 році, і основною метою було допомогти розробникам працювати над великомасштабними додатками, надаючи їм інструменти для рефакторингу, пошуку посилань тощо.

Популярність мови програмування TypeScript зростає з кожним днем, бо все більше і більше backend розробників починають працювати з цією технологією (рис. 2.1.) [4].

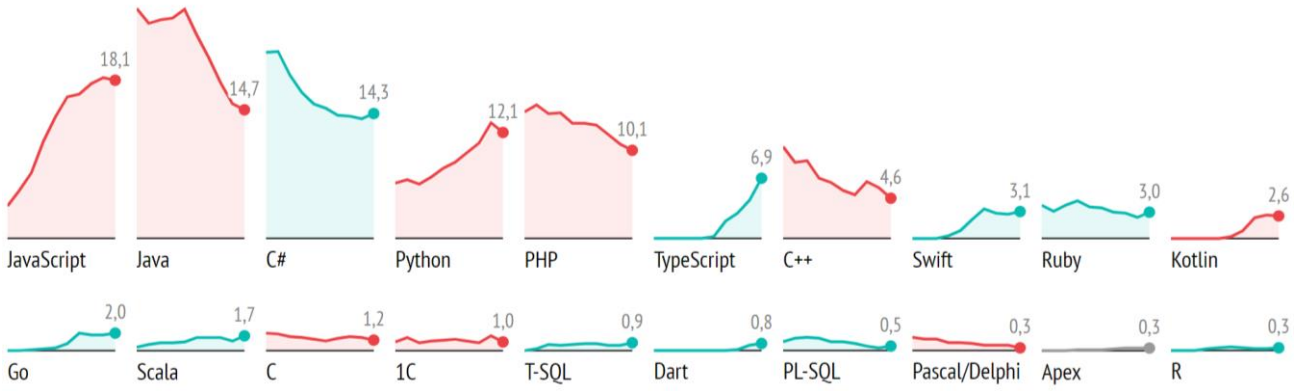

Рис. 2.1. Графіки збільшення чи зменшення використань мов програмування

Тому не дивно що TypeScript вже на шостому місці за популярністю мов програмування, обійшов C++ та й IT спеціалісти прогнозують суперництво з JavaScript чи Python (рис. 2.2.) [4].

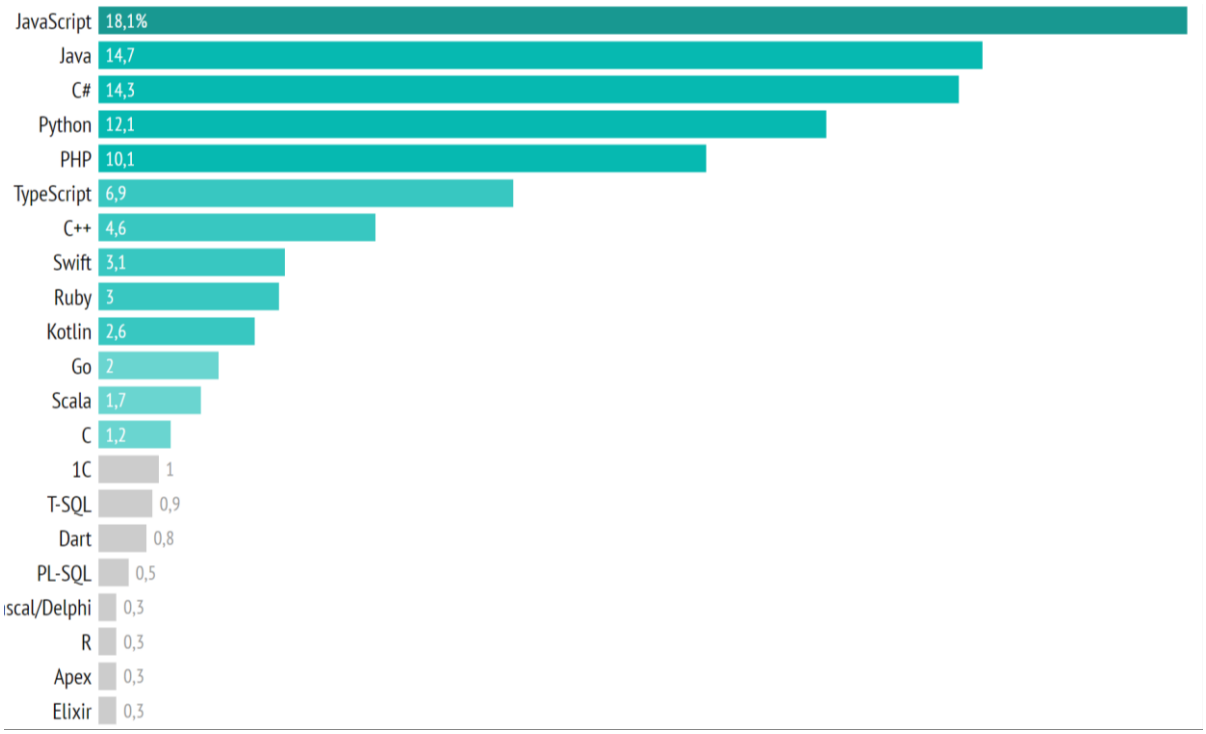

Рис. 2.2 Рейтинг популярності мов програмування на 2021 рік

TypeScript - це мова з відкритим кодом, яка базується на JavaScript, одному з найбільш використовуваних у світі інструментів, додаючи визначення статичного типу. Типи дають спосіб описати форму об’єкта, забезпечуючи кращу документацію та дозволяючи TypeScript перевірити, чи правильно працює ваш код. Типи написання можуть бути необов’язковими в TypeScript, оскільки умовивід типу дозволяє отримати велику потужність без написання додаткового коду [5, 6].

Код TypeScript перетворюється в код JavaScript за допомогою компілятора TypeScript або Babel. Цей JavaScript - це чистий, простий код, який працює в будь-якому місці, де працює JavaScript: у браузері, на Node.JS або у ваших програмах. Наприклад, JavaScript надає мовні примітиви, такі як string і number, але не перевіряє, чи ви їх правильно призначали. TypeScript це робить. Головна

перевага TypeScript полягає в тому, що він може виділити несподівану поведінку у вашому коді, знижуючи ймовірність помилок.

Щоб використовувати TypeScript та отримати всі його переваги на Nodejs, потрібно лише встановити пару пакетів та додати конфігураційні файли:

- npm i -S typescript ts-node @types/node @types/express;
- npm i -D ts-node-dev;
- npx tsc -init.

Ви можете анотувати свій код типами, щоб ловити помилки під час компіляції (час розробки, якщо ви використовуєте редактор / розширення з підтримкою TS), а не ловити помилки під час виконання або навіть у виробництві. Можна використовувати TypeScript без типів, якщо хочете. Якщо ви створили змінну і не присвоїли їй значення одразу, вона виведе тип any, що є чим завгодно.

TS бере на себе багато функцій лінтера (форматує код). Наприклад, використання необ'єднаних глобальних перекладів у TS не дозволяється, і буде видалено помилку на момент компіляції. Тим паче TS не складає жодних обмежень на стиль написання коду. Це питання в багатьох справах справа звички і немає загальноприйнятого правильного підходу, але все приходить до думки, що в проекті повинен бути виконаний єдиний стиль. У цьому допоможе використання TSLint - лінтера для TS. Для його використання треба встановити tslint глобально командою npm instal -g tslint.

TypeScript сам по собі вже приносить масу переваг, але є цікаві речі, які ви отримуєте, використовуючи його в API. TS дозволяє описувати бібліотеки JS за допомогою файлів декларацій (файли декларацій * .d.ts). За цього TS достатньо гібок і підтримує різні типи бібліотеки та режими їх використання (глобальні бібліотеки, модулі, UMP та пр).

Для розробки додатків з TS, як мінімум, потрібно отримати файл декларацій для Node JS:

npm install -D @ types / node

Це встановлення декларацій Node API, таких як глобальні require/process/globals, стандартні модулі fs / path / stream та інші. У цьому пакеті описуються node API останньої версії 7.x, що повинно підійти і для більш старих версій Node JS.

Node.js представляє осередок виконання коду на JavaScript, який побудований на основі движка JavaScript Chrome V8, що дозволяє транслювати виклики на мові JavaScript у машинному коді. Node.js попередньо призначений для створення серверних додатків на мові JavaScript. Хотя також існують проекти за написанням десктопних програм (Electron) і навіть за створенням коду для мікроконтролерів. Але все ж таки ми говоримо про Node.js, як про платформу для створення веб-додатків [7].

Для розробки під Node JS достатньо простейшого текстового редактора, зокрема, Notepad ++. Також можна скористатися більш ізольованими редакторами Atom, Sublime, Visual Studio Code, або середовищами розробки, які підтримують роботу з Node.JS, наприклад, Visual Studio або WebStorm.

Одною з відлічних черт цієї технології є підхід до роботи з вводом-виводом і відсутністю багатопоточності - вживання неблокіруючого, асинхронного API вводу-виводу можна знайти не в кожному визначенні. Щоб поняти, про що сказати, привести простой приклад - виконати HTTP-запрошення:

– з традиційним підходом ви вибираєте функцію / метод, виконуєте запрошення та виконуєте потоки, заблоковані до цих пор, не отримуючи відповіді або не викликаючи помилки. Якщо вам потрібно виконати будь-які дії під час виконання запитів, вам потрібно використовувати ще один потік.

– у вузлі ви вибираєте функцію / метод та помимо параметрів запрошення, переглядаєте ще один аргумент - функцію, яка буде викликана за завершенням запрошення. Визов не блокує виконання, і код продовжує виконуватися.

На цьому принципі в Node побудовано все взаємодія із сетью, базами даних та файловою системою. API для багатопоточності не передбачено, і це

використовує цілий клас помилок, пов’язаних із синхронізацією потоків та стану гонки’амі. «Плата» для цього - сложності з розподіленням вичислень.

npm - це пакетний менеджер для Node, важлива частина екосистеми. Собственно, так и расшифровывается - менеджер пакетів вузлів. На момент написання статті кількість пакетів у хвилину збільшується на 400 тисяч. Любі бібліотеки та фреймворки для Node слід розміщувати через npm. Як правило, пакети ставляться локально, т. е. лише для повідомлення проекту. Однак деякі утиліти, що мають інтерфейс командної строки (CLI), переважно ставлять глобально на рівні системи. Пакети, які виходять із npm локально, повинні бути перераховані у файлі package.json, що знаходяться в корневому проекті. Це потрібно для того, щоб разом з проектом була інформація про всі пакети, які йому потрібні (можна взяти цей файл в аналоговому файлі Podfile).

Для написання простого сервера нам потрібно отримати всього два зовнішніх пакети - це фреймворк Express і модуль body-parser до нього, який розглядає JSON-тело POST-запросів. Незважаючи на те, що Express Express називається фреймворком, жодних складностей з його освоєнням не виникає - в будь-якому випадку, він точно проходить, чим використовується вбудоване в Node HTTP-сервер у чистому вигляді.

Встановити залежність можна наступною командою:

$ npm install --save --exact express body-parser [15].

2.4. Опис структури системи та алгоритмів її функціонування

Структура будується у архітектурі в виді слоїв. Кожен з слоїв відповідає за свою дію на тому чи іншому рівні. В додатку є 4 рівні:

- рівень роутінгу;
- рівень контролю;
- рівень сервісу;
- рівень моделі.

На кожному рівні виконуються свої дії. Роутінг відповідає за розподілення дій на контролер. Контролер в свою чергу відповідає за виклик сервісу та відправку даних на клієнт. Сервіс у свою чергу, викликає модель та задає параметри виклику. Після виклику модель формує запит до бази даних та за допогою ORM здійснює його та отримує запитані дані. Після цього данні передаються в зворотньому напрямку до контроллеру. Дана архітектура допомагає розділити відповідальність на кожному з рівнів, та уникнути складних проблем з програмним застосунком.

При створенні досить великого додатка рекомендується такі папки (особливо якщо використовується якийсь MVC- / ORM-фреймворк типу express або mongoose):

- / Models містить всі ваші моделі ORM (звані Schemas в mongoose);
- / Views містить ваші шаблони уявлень (використовуючи будь-яку мову шаблонів, підтримуваний в express);
- / Public містить весь статичний контент (зображення, таблиці стилів, клієнтська частина JavaScript);
- / Assets / images містить файли зображень;
- / Assets / pdf містить статичні файли pdf;
- / Css містить таблиці стилів (або скомпільовані вихідні дані движком css);
- / Js містить клієнтську сторону JavaScript;
- / Controllers містить всі ваші експрес-маршрути, розділені модулем / областю вашого застосування (Примітка: при використанні функції bootstrapping express ця папка називається / routes).

Файли кожного рівня знаходяться у одній папці (рис 2.3.). Кожен рівень пов’язан з іншим, ці зв’язки виглядають як павутиння.

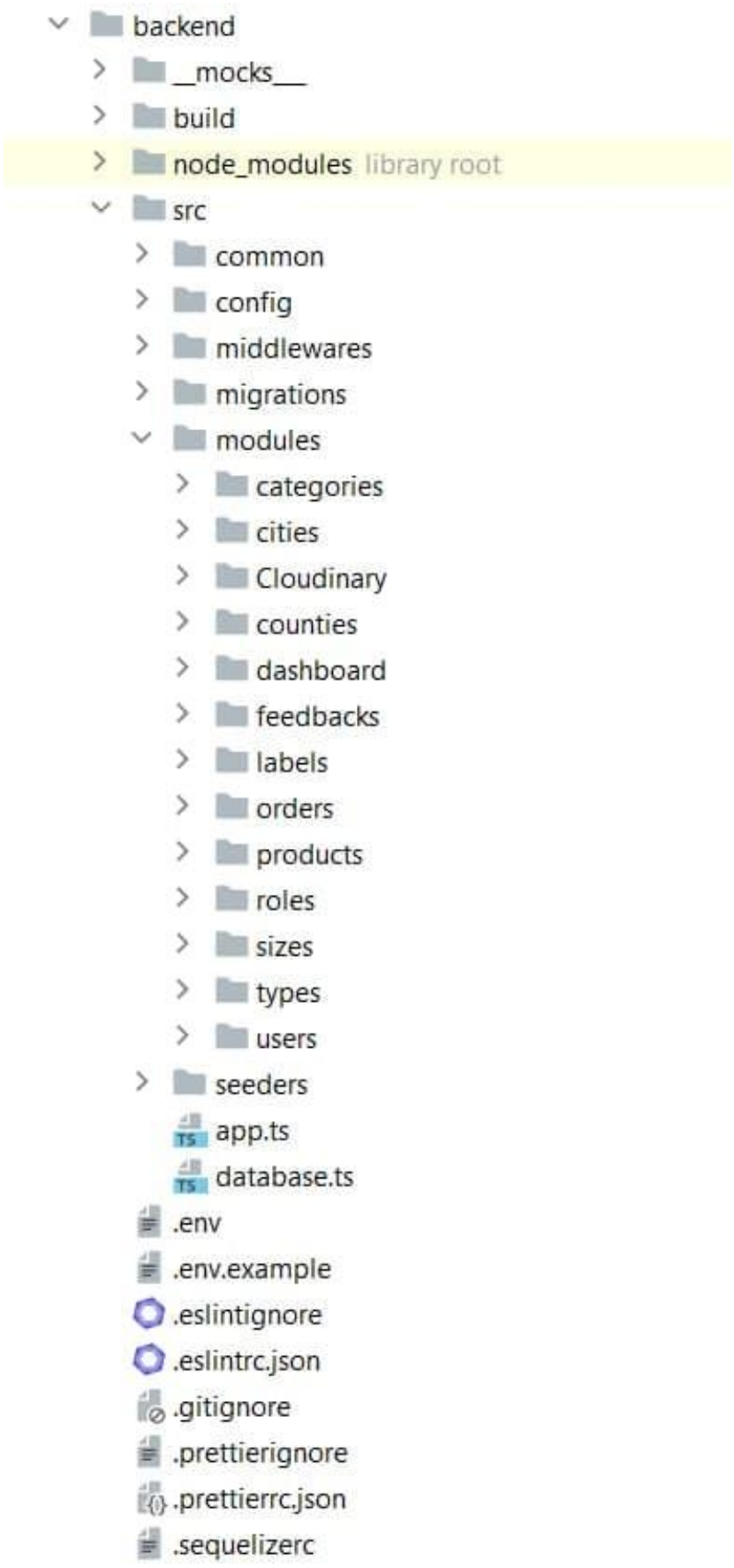

Рис. 2.3. Файли проекту

База даних в свою чергу розробляється для того, щоб процес систематизації відбувався швидше і, відповідно, щоб результативність програми підвищувалася. База буде ключовим складником в загальній системі управління програмним додатком. Тому що в ній будуть зберігатися всі дані, з якими відбувається робота програми та її складових.

Дуже важливо щоб у кожного поля, які будуть надалі вноситися в базу, були унікальні ідентифікатори, які не повторюватимуться.

Також при моделюванні бази даних необхідно врахувати наступне:

– один користувач може замовити різну продукцію і різні користувачі однакову продукцію;

– одне замовлення може бути доставлено в одну країну та в одне місто, але місто та країна може отримувати багато замовлень;

– замовлення може містити багато продуктів та продукт може міститься в багатьох замовленнях.

Для даної концепції бази даних (рис.2.4.) можна виділити такий перелік сутностей:

1. Робітники – містить інформацію про особисті дані робітників фабрики такі як, ПІБ, дату народження, паспортні дані, стать, досвід роботи, посада, місце проживання тощо.
2. Користувачі – містить інформацію про користувачів системи такі як, ім’я та прізвище, мобільний телефон, електрону пошту тощо.
3. Продукція – містить інформацію про продукцію що продає магазин таку як, тип продукції, колір, розмір, категорію тощо.
4. Країни – містить інформацію про країни до яких можлива доставка.
5. Міста – містить інформацію про міста до яких можлива доставка.
6. Розміри – містить інформацію про розміри продукції.
7. Категорії – містить інформацію про категорії продукції.
8. Замовлення – містить інформацію про замовлення таку як, замовник, список товарів, країну та місто доставки тощо.
9. Ролі – містить ролі користувачів системи.

10. Мітки – містить інформацію про мітки продукції.

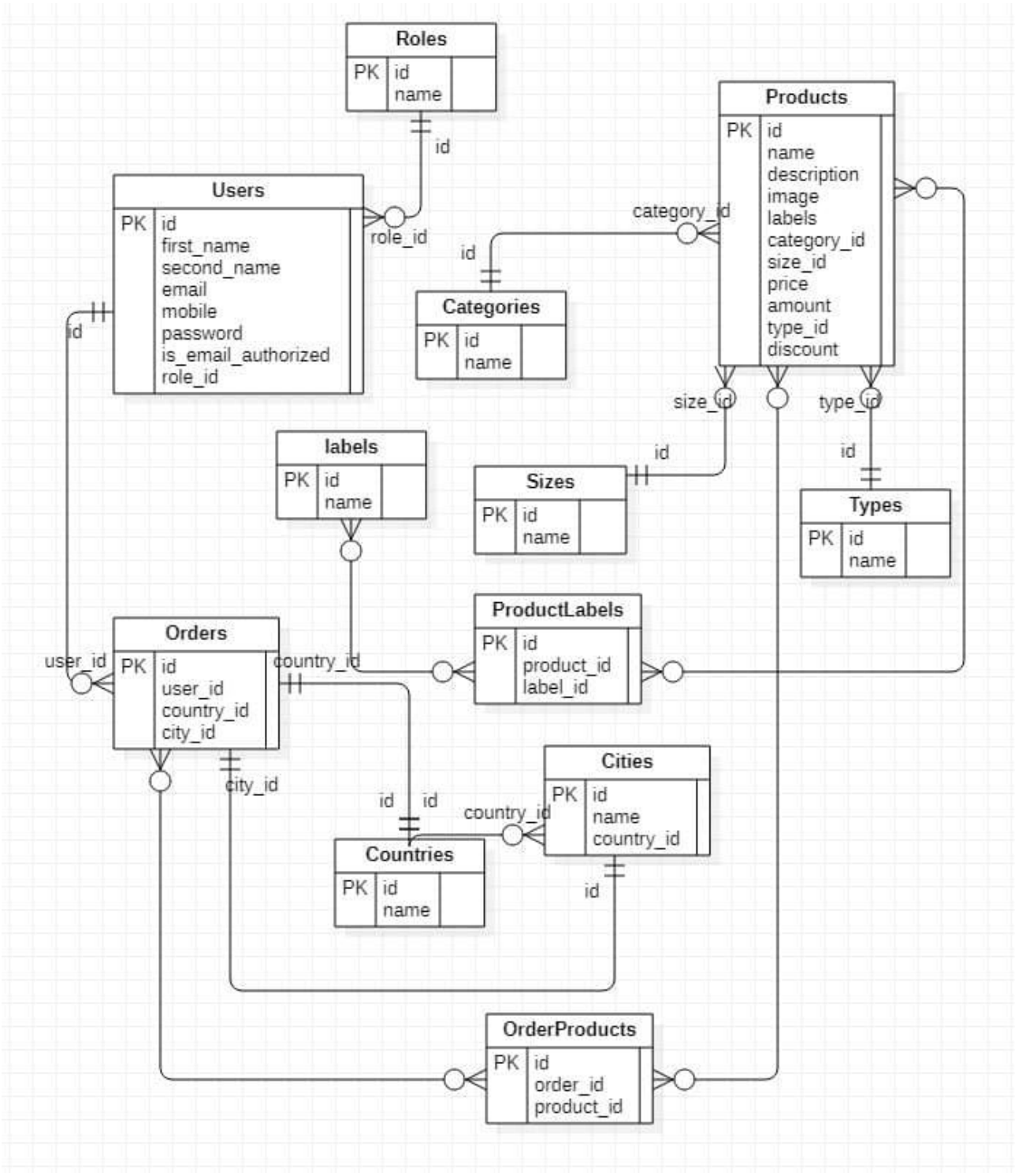

Рис. 2.4. Діаграма відносин

Зв’язки між розписаними віще сутностями будуть такими:

1. Між атрибутами Користувача та Ролі буде зв’язок M:1, так як користувач може мати одну роль, а роль може бути у декількох користувачів.

2. Між атрибутами Користувач та Замовлення буде зв’язок 1:M, так як користувач може мати багато замовлень, а замовлення може бути лише у одного користувача.

3. Між атрибутами Замовлення та Продукція буде зв’язок M:M, так як замовлення може містити багато продукції, та продукція може бути в багатьох замовленнях.

4. Між атрибутами Замовлення та Країна буде зв’язок 1:M, так як замовлення може мати лише одну країну, а країна може отримувати багато замовлень.

5. Між атрибутами Замовлення та Місто буде зв’язок 1:M, так як замовлення може мати лише одне місто, а місто може отримувати багато замовлень.

6. Між атрибутами Продукція та Розмір буде зв’язок 1:M, так як продукція може мати лише один розмір, а розмір може бути у багатьох продуктів.

7. Між атрибутами Продукція та Категорія буде зв’язок 1:M, так як продукція може мати лише одну категорію, а категорія може бути у багатьох продуктів.

8. Між атрибутами Продукція та Типи буде зв’язок 1:M, так як продукція може мати лише один тип, а тип може бути у багатьох продуктів.

9. Між атрибутами Продукція та Мітка буде зв’язок M:M, так як продукція може містити багато міток, та мітка може бути у багатьох продуктів.

2.5. Обгрунтування та організація вхідних та вихідних даних програми

Вхідними даними для backend частини додатку є сформовані SQL запити, що приходять з інтерфейсу користувача, коли той хоче отримати ту чи іншу інформацію про магазин. Вихідними даними є JSON файл з інформацією про товари, який передається інтерфейсу для відображення.

Обов’язковими відомостями, які будуть зберігатися в базі є:

– список товарів з даними про кожен з них;

– список користувачів системи з даними на кожного з них;

– список замовлень, що були зареєстровані з інформацією про кожен з них.

– список країн та їх міст до яких можлива доставка.

На даному етапі розробки бази даних можна скласти такі таблиці сутностей (табл. 2.1. - 2.13.), в яких прописано назви їх атрибутів, типи даних цих атрибутів, розмір поля, а також дано певним полям обмеження на введення значень.

Таблиця 2.1.

Таблиця «Roles»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
Name	VARCHAR(256), Not null	

Таблиця 2.2.

Таблиця «Users»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
First_name	VARCHAR(256), Not null	
Second_name	VARCHAR(256)	
Email	VARCHAR(256), Not null	
Mobile_phone	VARCHAR(256)	
Password	VARCHAR(256)	
Is_email_authorized	BOOLEAN, Default Value: false	
Role_id	INT, Not null, Default Value: 1	FK

Таблиця 2.3.

Таблиця «Orders»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
User_id	INT, Not null	FK
Country_id	INT, Not null	FK
City_id	INT, Not null	FK

Таблиця 2.4.

Таблиця «Countries»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
Name	VARCHAR(256), Not null	

Таблиця 2.5.

Таблиця «Cities»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
Name	VARCHAR(256), Not Null	
Country_id	INT, Not null	FK

Таблиця 2.6.

Таблиця «OrderProducts»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
Order_id	INT, Not null	FK
Product_id	INT, Not null	FK

Таблиця 2.7.

Таблиця «Products»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
Name	VARCHAR(256), Not Null	
Description	VARCHAR(2048)	
Image	VARCHAR(512)	
Category_id	INT, Not null	FK
Size_id	INT, Not null	FK
Price	FLOAT, Not null	
Discount	FLOAT	
Amount	INT, Not null	
Type_id	INT, Not null	FK

Таблиця 2.8.

Таблиця «Types»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
Name	VARCHAR(256), Not Null	

Таблиця 2.9.

Таблиця «Sizes»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
Name	VARCHAR(256), Not Null	

Таблиця 2.10.

Таблиця «Categories»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
Name	VARCHAR(256), Not Null	

Таблиця 2.11.

Таблиця «ProductLabels»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
Product_id	INT, Not null	FK
Label_id	INT, Not null	FK

Таблиця 2.12.

Таблиця «Labels»

Назва	Тип даних, обмеження	Тип поля
Id	INT, Not null, Auto Increment	PK
Name	VARCHAR(256), Not null	
Color	VARCHAR(256), Not null	

Батьківськими таблицями в базі даних є таблиця «Users», «Orders» та «Products».

У даній базі даних представлені наступні запити:

1. Уся продукція – повертає усю доступну продукцію з заданими фільтрами.
2. Усі користувачі – повертає усіх користувачів системи та інформацію про них.
3. Усі замовлення – повертає усі замовлення в системі.
4. Усі країни – повертає список країн.
5. Усі міста – повертає список міст.

6. Усі розміри – повертає список усіх розмірів продукції.
7. Усі категорії – повертає список усіх категорій продукції.
8. Усі типи – повертає список усіх типів товарів.
9. Усі ролі – повертає список ролів користувачів системи.

А також запит на додавання продукції та замовлення. Запити на додавання нових країн та міст, категорій та типів. Про додаванні нового замовлення буде змінюватися загальна статистика проданих товарів.

Також можливе редагування кожної сутності окремо.

Перед тим як використовувати базу даних користувачі будуть зобов'язані пройти авторизацію (вказати свої login і password).

Найбільшими правами користується адміністратор. Він може: переглядати вже занесені до бази даних дані; брати інформацію з бази; вносити принципові зміни до побудованої бази даних; встановлювати оновлення бази даних; додавати нові поля та таблиці, тощо.

Адміністратор також контролює те, в якому зараз стані знаходиться база даних, при необхідності архівувати дані, а також за необхідності здійснювати модернізацію.

Працівники магазину (оператори) можуть переглядати інформацію щодо товарів, створювати нові записи, корегувати їх, видаляти при необхідності та формувати нові записи що до інформації з продажу продукції з магазину.

Адміністративно – управляючий персонал має доступ до перегляду бази та формування нової продукції, яку продає магазин, зміни інформації що до замовлень.

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

Для серверних технічних засобів рекомендована конфігурація, що забезпечує цілодобову роботу програми з резервуванням даних:

– процесор класу AMD Ryzen 9 5900X 3.7(4.8)GHz 64MB sAM4 Tray (100-000000061);

– материнська плата Gigabyte X570 GAMING X (sAM4, AMD X570);

– модулі пам’яті HyperX DDR4 16GB (2x8GB) 3200Mhz Fury Black (HX432C16FB3K2/16);

– блок живлення Gigabyte AORUS P750W 750W (GP-AP750GM);

– система охолодження Be Quiet Dark Rock Pro 4 (BK022);

– SSD диск Kingston A2000 3D NAND 500GB M.2 (2280 PCI-E) NVMe x4 (SA2000M8/500G);

– рідкокристалічний монітор з діагоналлю не менше 17 ";

– доступ до мережі Internet;

– маніпулятор "миша";

– клавіатура.

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний виріб буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутими замовником.

2.6.2. Використані програмні засоби

Для написання та редагування коду був обраний WebStorm. WebStorm - це IDE, створена на базі платформи IntelliJ, розробленої JetBrains і випущеної під відкритою ліцензією, і оснащена безліччю можливостей, які зроблять розробку більш приємною і продуктивною.

Проект реалізований на мові програмування TypeScript з використанням бібліотеки Node JS, до якої були завантажені npm пакети. Для створення API також використовувалася бібліотека Express - це мінімалістичний і гнучкий веб-фреймворк для додатків Node.js, що надає великий набір функцій для мобільних і веб-додатків. В якості СУБД була обрана PostgreSQL - реляційна СУБД з вільною ліцензією.

Для збереження зображень товару був обраний хмарний сервіс Cloudinary. Він робить автоматизацію робочих процесів зображень, відео та мультимедійних даних простими, пропонуючи різноманітні функції. Від завантаження засобів масової інформації до оптимізації зображень, перекодування відео та маніпуляцій через доставку через глобальні CDN, все пропонується в плагіні.

2.6.3. Виклик та завантаження програми

Для виклику та завантаженяя необхідно виконаті наступні дії:

1. Орендувати або придбати VPS сервер.
2. Налаштувати сервер та його середу.
3. Налаштувати скомпільовані фали під сервер.
4. Перемістити файли за допомогою FTP клієнта на сервер у папку src.
5. Встановити на сервері програмне забезпечення Node.JS.
6. Встановити базу даних на окремий сервер.
7. Додати базу даних до додатку.
8. За допомогою Node.JS додати до активних процесів головний вайл додатку.
9. За допомогою міграцій та сидів генерувати базу даних та її начальні данні.

2.6.4. Опис інтерфейсу користувача

При відкритті інтернет-магазину одягу «Closes Shop» користувач потрапляє на Головну сторінку (рис.2.5.). Верхня частина головної сторінки складається з головного меню і списку розділів сайту. У верхньому правому кутку користувач з роллю «Адміністратор» чи «Менеджер» під відповідними логінами та паролями можуть потрапити до адміністративної панелі, в якій відображається уся інформація про магазин.

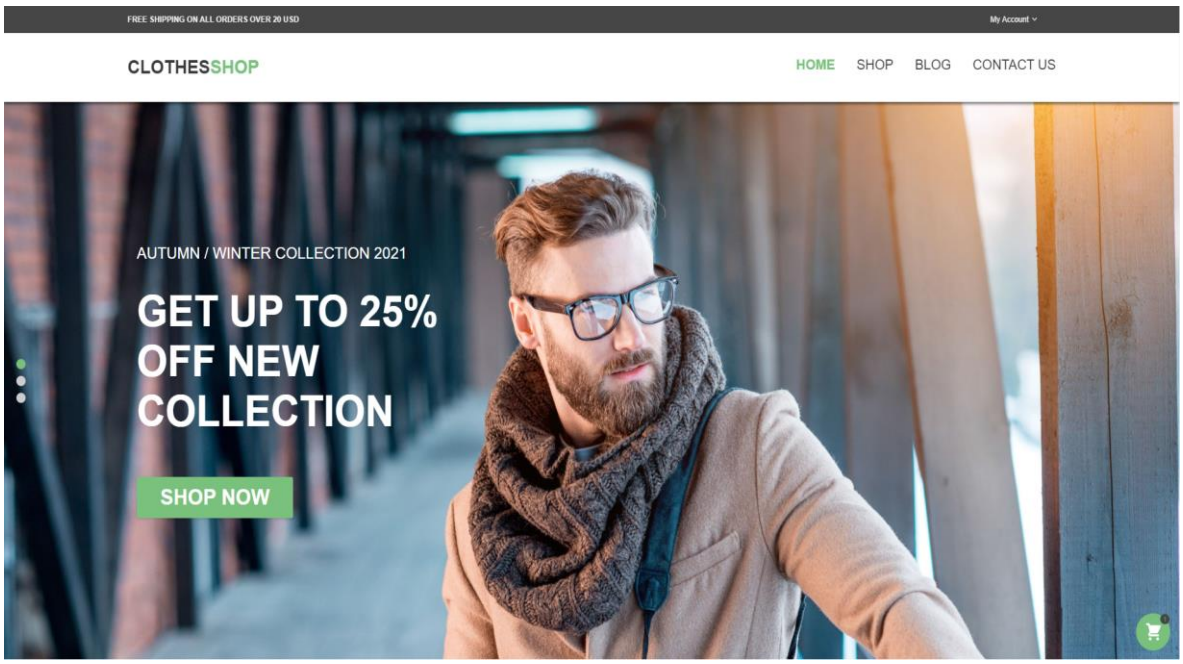

Рис. 2.5. Головна сторінка

Якщо користувач з роллю адміністратора перейде на адміністративну панель, то він потрапляє на статистику (рис.2.6.) усього додатку. На цій сторінці відображено прогрес замовлень, кількість відгуків від користувачів, з детальною статистикою. Також відображено загальну статистику додатку.

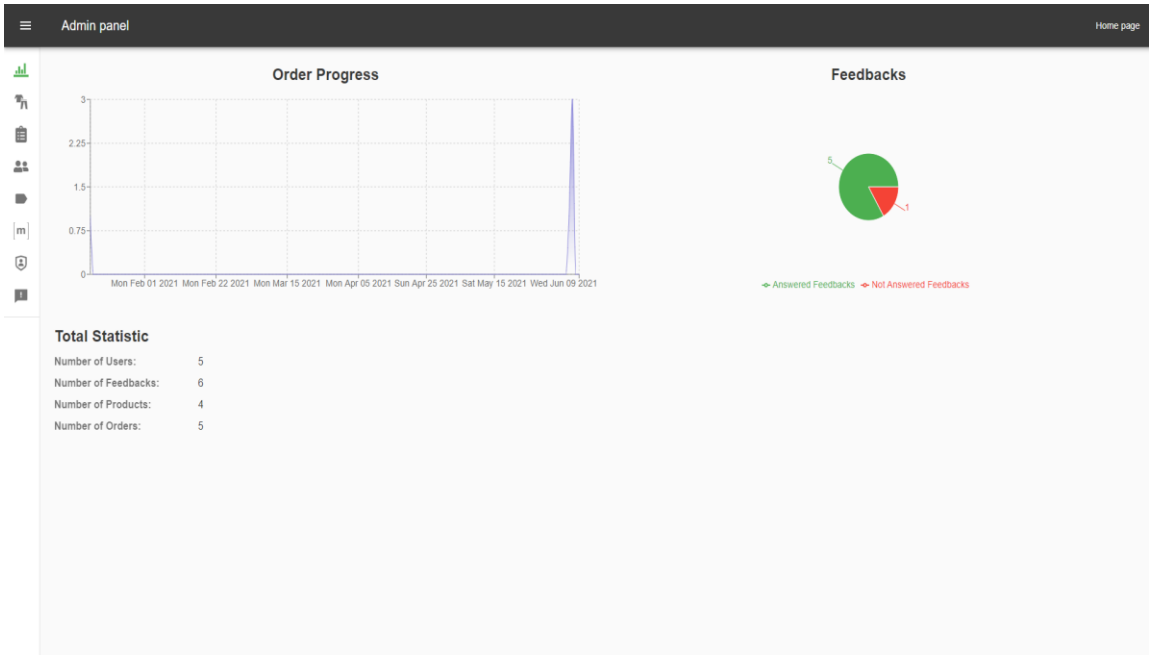

Рис. 2.6. Статистика у адміністративній панелі

Якщо адміністратор перейде на сторінку товарів (рис.2.7.), що відображена в лівому навігаційному меню він потрапить на сторінку, на якій з бази даних витягуються уся інформація про товари. Товари можна редагувати та видаляти. Усі коригування інформації про товари також відобразяться у БД.

Адміністратор може коригувати назву, опис, розмір, категорію товару, додавати чи видаляти лейбли, оновлювати фотографію. При натисканні на кнопку олівця інтерфейс викликає модальне вікно, при зміні та збереженні інформації в якому зміни відобразяться у базі даних.

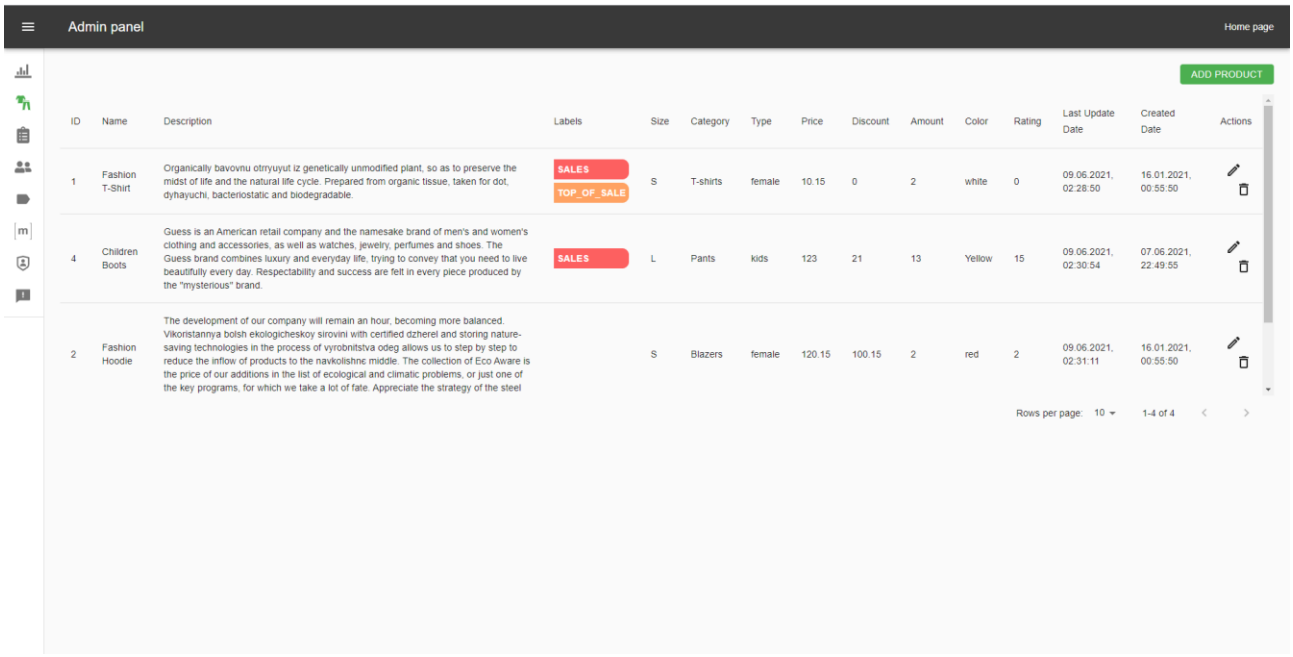

Рис. 2.7. Сторінка товарів у адміністративній панелі

Якщо менеджер перейде на сторінку замовлень, що також відображена в лівому навігаційному меню він потрапить на сторінку замовлень. На цій сторінці (рис.2.8.) відображено список усіх замовлень сайту з інформацією про них. Також можливо редагувати, додавати, переглядати інформацію та видаляти кожен з них.

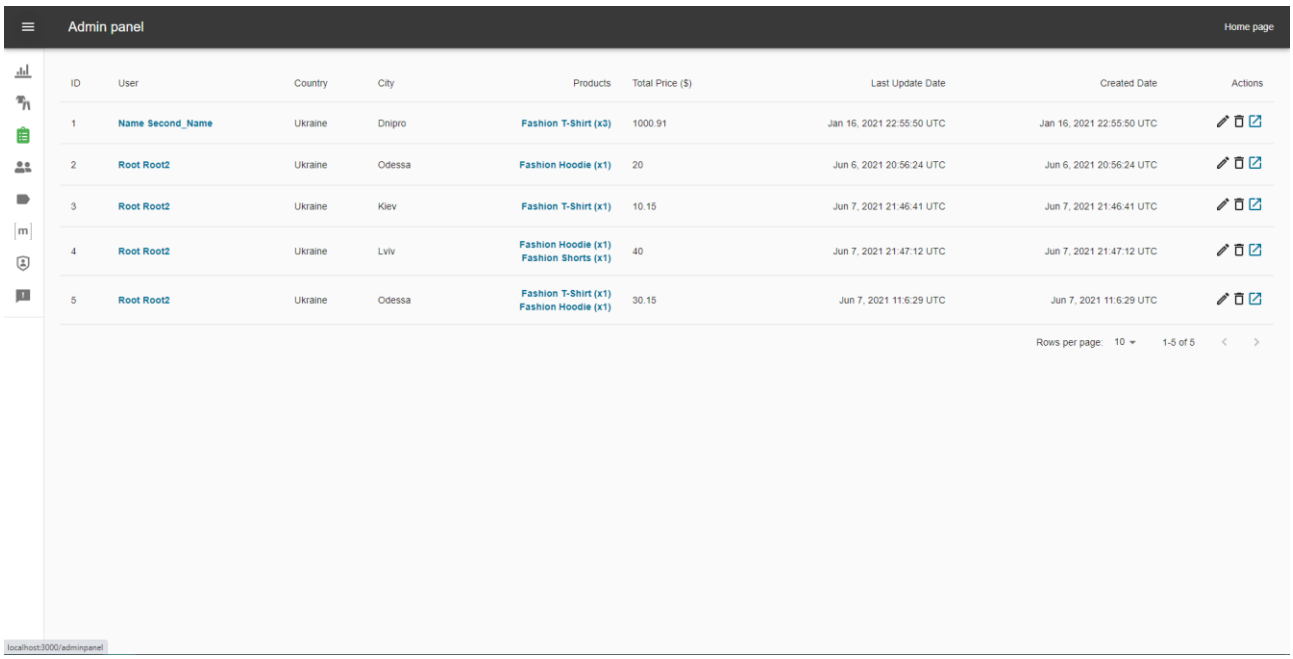

Рис. 2.8. Сторінка замовлень у адміністративній панелі

Якщо адміністратор відкриє інформацію о замовленні, він побачить сторінку з детальною інформацією (рис.2.9.). На сторінці показано дані доставки, товари що містяться в замовлені та інформацію про замовника. На цій вкладці будуть оновлятися і додаватися дані тільки, як користувач створить замовлення, сервер та база даних обробить його та відправить оновлені дані на інтерфейс.

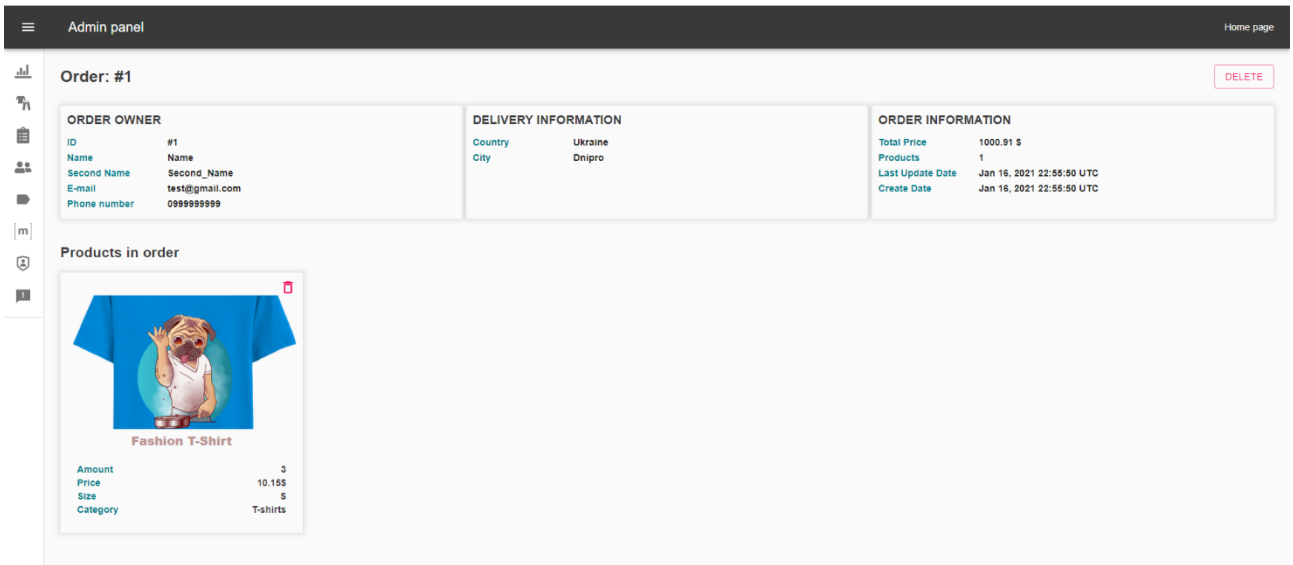

Рис. 2.9. Деталі замовлення у адміністративній панелі

Якщо адміністратор перейде на сторінку користувачів (рис.2.10.), що відображена в лівому навігаційному меню він потрапить на сторінку з даними усіх користувачів сайту з інформацією про них.

Якщо користувачів буде більше, ніж розташовано на 1 сторінці, то запитами будуть формуватися списки за сторінками навігації внизу сторінки.

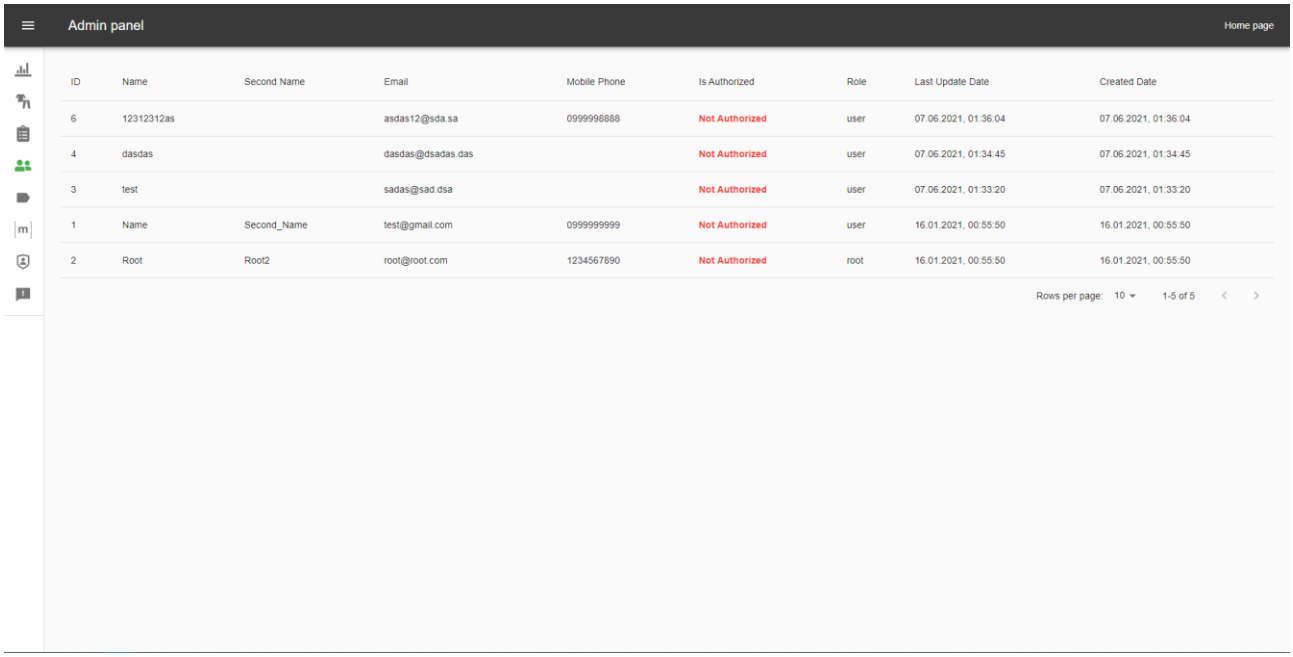

Рис. 2.10. Список користувачів у адміністративній панелі

Якщо користувач перейде на сторінку міток товарів (рис.2.11.), що відображена в лівому навігаційному меню він потрапить на сторінку з різними лейблами, що відображають якусь відмінність товару. Списки лейблів для кожного товару зберігається окремою таблицею, тому кожен товар має змогу отримати будь-який лейбл, хоч усі. Також можливо редагувати, додавати та видаляти кожну з них.

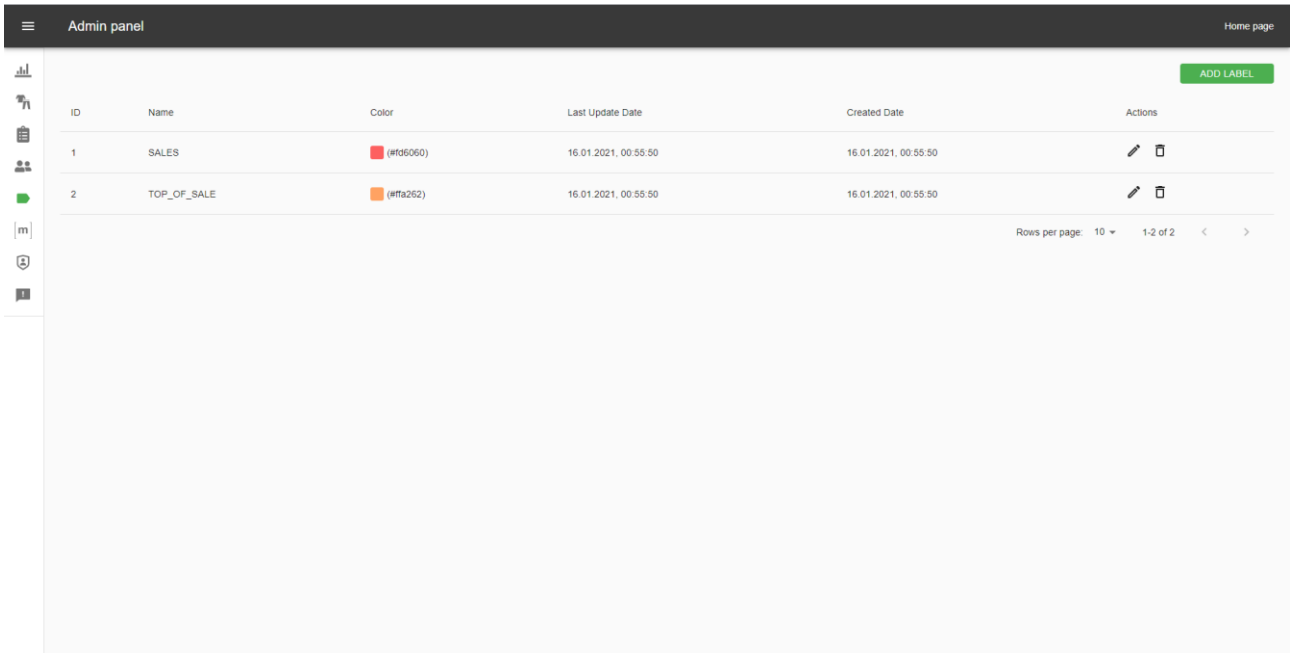

Рис. 2.11. Список лейблів у адміністративній панелі

Якщо адміністратор перейде на сторінку розмірів товарів (рис.2.12.), що відображена в лівому навігаційному меню він потрапить на сторінку де лише видно інформацію який розмір коли був створений та відредагований. На даному етапі дата редагування співпажає з датою створення, бо поки що немає потреби змінювати стандартні розміри.

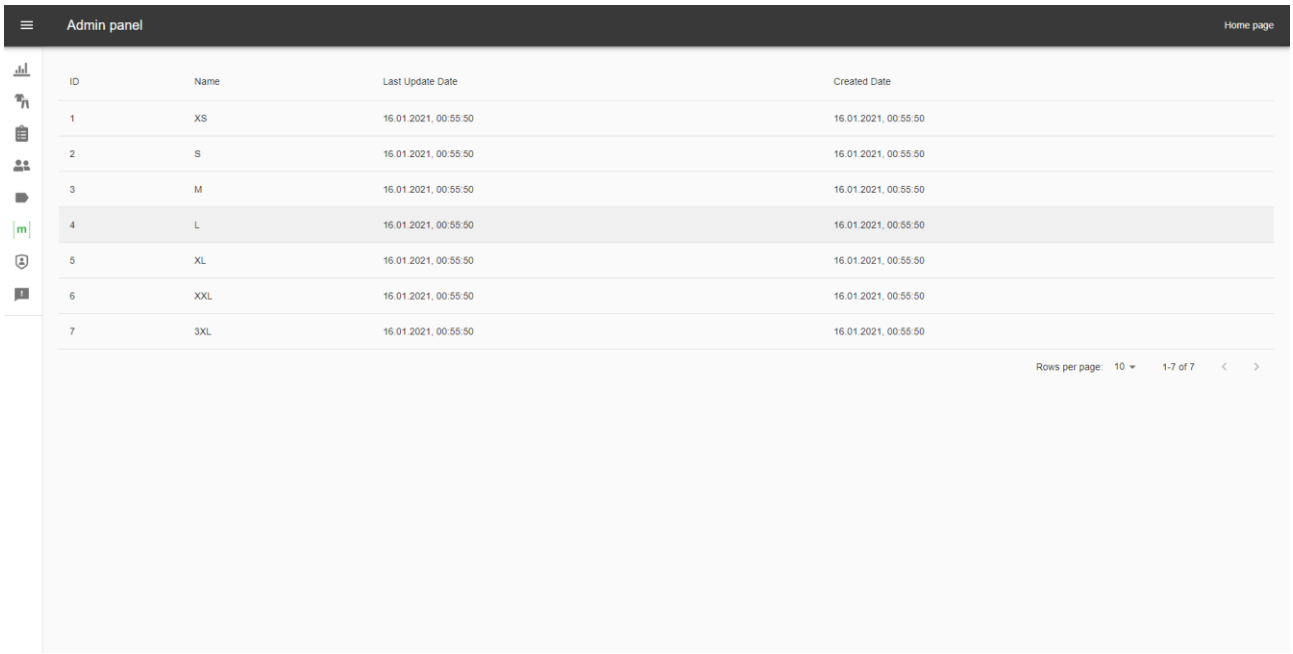

Рис. 2.12. Таблиця розмірів у адміністративній панелі

Якщо адміністратор перейде на сторінку ролів користувачів (рис.2.13.) , то побачить 4 різні ролі, що відображають розділення прав доступу до цього додатку.

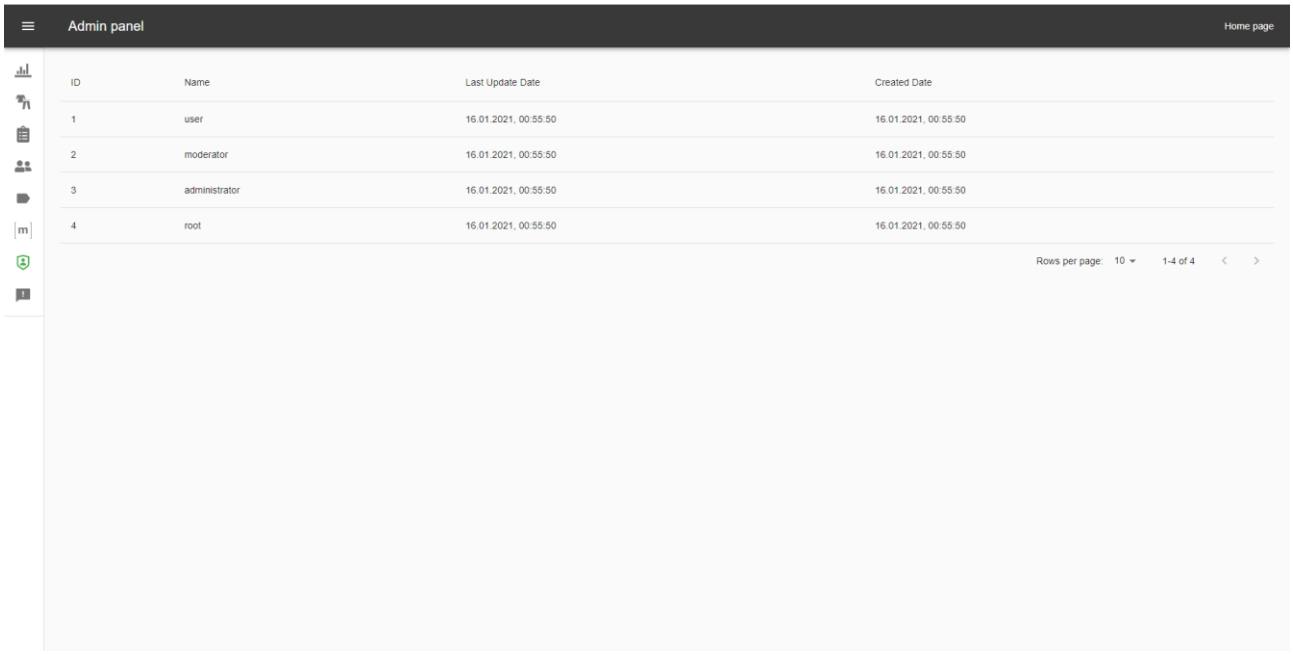

Рис. 2.13. Ролі користувачів у адміністративній панелі

При переході на сторінку зворотного зв’язку (рис. 2.14.) , що відображена в лівому навігаційному меню він потрапить на сторінку відгуків що були отримані від користувачів. На сторінці зворотного зв’язку відображено список усіх відгуків що приходять з сайту від користувачі. Також є можливість відповісти на кожен з них. Кожен клієнт отримає відповідь на свою особисту пошту, яку він вказав при відправленні запиту. У таблиці гарно видно на які звернення вже відповів адміністратор, на які ні. При відповіді на звернення статус також змінюється та зберігається у базі даних.

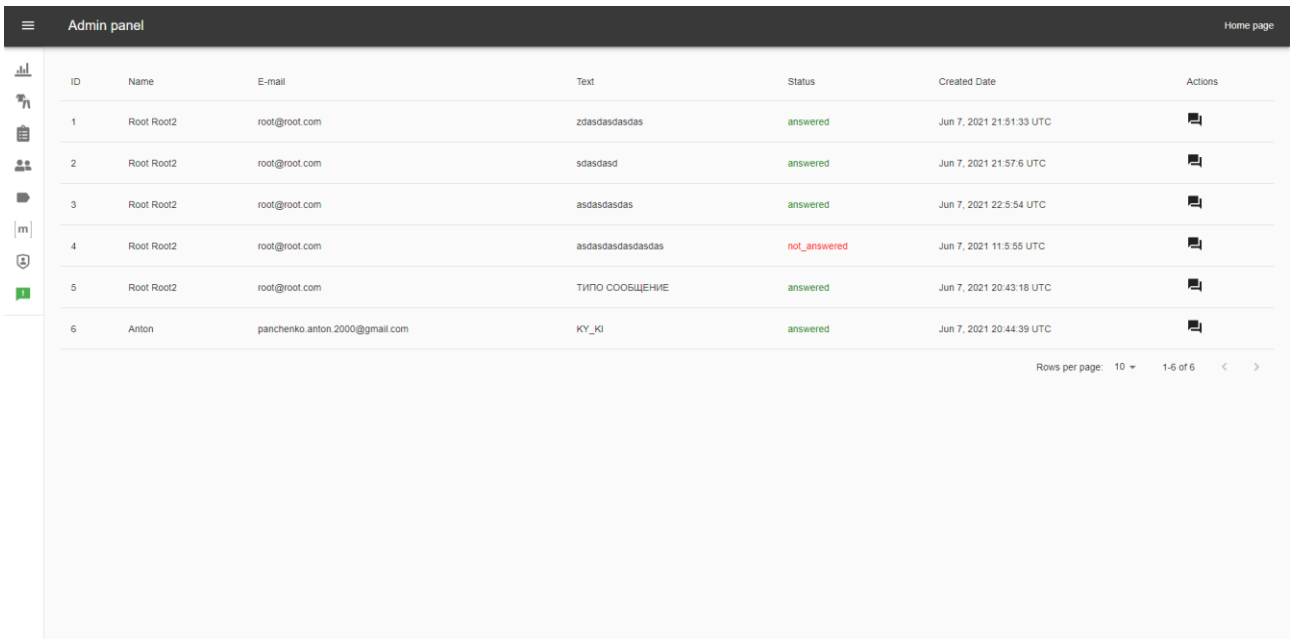

Рис. 2.14. Сторінка відгуків у адміністративній панелі

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 2120;
2. коефіцієнт корекції програми в ході її розробки – 0,05;
3. коефіцієнт складності програми – 1,7;
4. годинна заробітна плата програміста– 130 грн/год;

Середня годинна зарплата програміста була вирахувати виходячи з даних «Української спільноти програмістів (DOU)». Станом на кінець 2020 року зарплата Junior Web UI розробника простягається від 550$ до 1120$. Виходячи с цього середня заробітна плата програміста буде 836,8$ у місяць. При курсі валют НБУ на початок червня 2021 року один американський долар дорівнює 27,34 грн, тому середня зарплата в гривнях дорівнює 22878 грн. При стандартному графіку (176 годин/місяць) зарплата за годину буде становити близько 130 грн [21].

5. коефіцієнт збільшення витрат праці в наслідок недостатнього опису задачі – 1,4;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,4;
7. вартість машино-години ЕОМ –21 грн/год.

Оскільки для цього проекту протрібна велика потужність ПК для бекенду та підняття великої кількості даних на локальному сервері, гарним рішенням буде аренда комп’ютера на час розробки додатку. Вартість аренди комп’ютера на місяць 1300 грн (монітор) та 2400 грн (системний блок). Загалом на місяць оренда коштуватиме 3700 грн. При стандартному графіку (176 годин/місяць) вартість машино-години ЕОМ за годину роботи буде становити 21 грн. В цю

вартість входить ремонт за гарантією та базовий комплект гарнітури, такі як клавіатура та миша [22].

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин, (3.1)}$$

де t_o- витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a- витрати праці на розробку блок-схеми алгоритму;

t_n-витрати праці на програмування по готовій блок-схемі;

$t_{отл}$-витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмногму забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p),$$

де q - передбачуване число операторів (2120);

C - коефіцієнт складності програми (1,7);

p - коефіцієнт корекції програми в ході її розробки (0,05).

Звідси умовне число операторів в програмі:

$$Q = 1{,}7 \cdot 2120 \cdot (1 + 0{,}05) = 3784{,}2$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k} \text{ , людино-годин,}$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 5 до 8 років він складає 1,4.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1,2). З урахуванням коефіцієнта кваліфікації k = 1,4, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (3784,2 \cdot 1,2) / (75 \cdot 1,4) = 43,25 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25) \cdot k} \text{ , людино-годин, (3.2)}$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.2), отримаємо:

$$t_a = 3784,2 / (20 \cdot 1,4) = 135,15 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин.}$$

t_n = 3784,2/ (25 · 1,4) = 108,12 людино-годин.

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}\ \text{, людино-годин.}$$

$t_{отл}$= 3784,2/ (5 · 1,4) = 540,6 чел.-ч.

- за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5\cdot t_{отл}\ \text{, людино-годин.}$$

$t^k_{отл}$= 1,5 · 540,6 = 810,9 людино-годин.

Витрати праці на підготовку документації визначаються за формулою:

$$t_д = t_{др} + t_{до}\ \text{, людино-годин,}$$

де $t_{др}$-трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20)\cdot k}\ \text{, людино-годин,}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др}$$, людино-годин.

Підставляючи відповідні значення, отримаємо:

$t_{др}$ = 3784,2/ (18 · 1,4) = 170,5 людино-годин.

$t_{до}$ = 0,75 · 170,5 = 127,9 людино-годин.

$t_{д}$ = 170,5 + 127,9 = 298,4 людино-годин.

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

t = 50 + 43,25 + 135,15 + 108,12 + 540,6 + 298,4 = 1175,52 людино-годин.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}$$, грн.

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}$$, грн,

де: t - загальна трудомісткість, людино-годин;

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 130 грн / год, отримуємо:

$$З_{ЗП} = 1175{,}52 \cdot 130 = 150818 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot C_{мч}, \text{ грн, (3.3)}$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (21 грн/год).

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 540{,}6 \cdot 21 = 11\,352{,}6 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 150\,818 + 11\,352{,}6 = 162\,170{,}6 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.}$$

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Звідси витрати на створення програмного продукту:

$$T = 1175{,}52 / 1 \cdot 176 \approx 6{,}7 \text{ міс.}$$

Висновок: програмне забезпечення розроблено для забезпечення доступу користувачів до основних програм та пропозицій компанії продажу одягу, ефективної взаємодії між потенційними споживачами та компанією, створення позитивного іміджу компанії, підвищення продажу товарів. Вартість даного програмного забезпечення близько 162 170,6 грн і не вимагає додаткових витрат при розробці програми. Очікуваний час розробки становить 6,7 місяця. Цей термін пов'язаний зі значним числом операторів, і включає час на дослідження і розробку алгоритму вирішення поставленого завдання, програмування по готовому алгоритму, налагодження програми і підготовку документації.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було поставлено завдання розробити backend частину системи відображення та адміністрування інтернет-магазину одягу.

Це програмне забезпечення призначене для надання універсального інструменту для відображення та управлінням контенту інтернет-магазину, управління магазину з адміністративної панелі. Практичне призначення даної системи полягає в забезпеченні відвідувачам сайту простого і комфортного доступу до каталогу товарів за рахунок оптимальних параметрів візуалізації його вмісту, що зробить процес покупки швидшим і зручнішим, і підвищить ефективність роботи інтернет-магазину.

Під час виконання даного проекту були виконані наступні задачі:

– вивчено предметну галузь розв'язуваної задачі;

– створено алгоритм для реалізації поставленого завдання;

– створено базу даних і клієнтську та серверну програму, що працює разом.

Розроблене програмне забезпечення дозволяє:

– легке та доступне адміністрування інтернет магазину;

– формування web-сторінок на основі шаблону з використанням контенту полученого з серверу;

– формування web-сторінок на основі шаблону з використанням контенту з бази даних;

– контактування за адміністратором данного магазину;

– оформлення замовлень з данного магазину.

Програма реалізована на базі фреймворка NodeJS з використанням мови програмування TypeScript і СУБД PostgreSQL.

Також у кваліфікаційній роботі було визначено трудоміскість розробленої системи, проведений підрахунок вартості роботи по створенню програми у 162 170,6 грн та розраховано час на його створення - 6,7 місяців.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Хокинс С. Администрирование Web-сервера Apache / С. Хокинс — М.: Вильямс, 2020. — 336 с.
2. MySQL. Довідник. MySQL AB. — М: «Вільямс», 2016 — 521 с
3. Прамодкумар Дж. Садаладж, Мартин Фаулер. NoSQL: новая методология разработки нереляционных баз данных. Диалектика-Вильямс. 2017. -192 с. ISBN978-5-8459-1920-5
4. Ланг К., Чоу Дж. Публикация баз данных в Интернете - СПб.: Символ-Плюс, 2019р. - 206 с.
5. TypeScript – популярна мова програмування [Електронний ресурс] - Режим доступу: https://www.typescriptlang.org/
6. Використання TypeScript з платформою Node JS [Електронний ресурс] - Режим доступу: https://medium.com/ae-studio/why-you-should-use-typescript-on-the-backend-too-22fa94efb768
7. Node JS [Електронний ресурс] - Режим доступу: https://metanit.com/web/nodejs/
8. Зростання популярності мови програмування TypeScript [Електронний ресурс] - Режим доступу: https://habr.com/ru/post/543346/
9. Проектування баз даних - [Електронний ресурс] /IMF. – Режим доступу: https://ru.wikipedia.org/wiki/Проектирование_баз_данных
10. Этапы проектирования данных [Електронний ресурс] - Режим доступу: http://www.mstu.edu.ru/study/materials/zelenkov/ch_5_1.html
11. Сравнение реляционных и не реляционных (NOSQL) баз данных [Електронний ресурс] - Режим доступу: https://sibac.info/studconf/science/xliv/106548
12. SQL или NoSQL [Електронний ресурс] - Режим доступу: https://habr.com/ru/company/ruvds/blog/324936/
13. А.Аллан. Клієнтська розробка для професіоналів. Node.js – СПБ.:2017. – 220с.

14. Комисаров Д.А., Станкевич А.Г. Персональный учитель по персональному компьютеру. – М, 2000

15. Angular и TypeScript. Сайтостроение для профессионалов, 2018 р, 2020. - 752 с. Яков Файн, Антон Моисеев.

16. Хэррон, Д. Node.js Разработка серверных веб-приложений на JavaScript / Д. Хэррон. - М.: ДМК, 2018. - 144 с.

17. Белоногов, Г.Г. Автоматизация процессов накопления, поиска и обобщения информации / Г.Г. Белоногов, А.П. Новоселов. - М.: Наука, 2017. - 256 с.

18. Website и Web Application: в чем разница? [Електронний ресурс] - Режим доступу: https://dinarys.com/ru/blog/websitevs.-webapplication

19. Дейт, К.Дж. Введение в системы баз данных / К.Дж. Дейт. - К.: Диалектика; Издание 6-е, 2004. - 784 с.

20. Інькова Н. А. Створення Web-сайтів: Навчально-методичний посібник [Електронний ресурс] / Інькова Н.А., Зайцева Е.А., Кузьміна Н.В., Толстих С.Г. / / Режим доступу: http://club-edu.tambov.ru/methodic/fio/p5.doc

21. Середня заробітна плата програміста у Дніпрі станом на початок 2021 року. [Електронний ресурс] - Режим доступу: https://dou.ua/lenta/articles/salary-report-devs-june-2020/

22. Вартість аренди ноутбуку почасово [Електронний ресурс] - Режим доступу: https://notebooksbu.com/garantiya-3-goda/

23. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності «Комп'ютерні системи» / О.Г. Вагонова, О.Б. Нікітіна, Н.Н. Романюк; М-во освіти і науки України, ДВНЗ «Нац. гірн. ун- т». – Д.: НГУ, 2013. – 11 с.

24. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 6.050101 «Комп'ютерні науки / І.М. Удовик, Л.М. Коротенко, О.С. Шевцова. Нац. гірн. ун-т. – Д : НТУ «Дніпровська політехніка» . - 2018. – 65 с.

ДОДАТОК А

КОД ПРОГРАМИ

Index.ts // for helpers folder

```
/* eslint-disable */
import { hashSync } from 'bcrypt';
import { Strategy, ExtractJwt } from 'passport-jwt';
import { autCfg } from '../../config/auth.config';
import { Unauthorized } from '../exeptions';
import { IStrategyOptions } from '../interfaces';
import UsersService from '../../modules/users/users.service';
import { UsersModel } from '../../modules/users/users.model';
import { ProductsModel } from '../../modules/products/products.model';
export const hashPassword = (password: string | undefined): string => {
  return hashSync(password, 10);
};
const options: IStrategyOptions = {
  jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
  secretOrKey: autCfg.secretKey,
  expiresIn: autCfg.expiresIn
};
export const strategy: Strategy = new Strategy(
  options,
  async (jwtPayload, done) => {
    try {
      const user: UsersModel | null = await UsersService.getOneByEmail(
        jwtPayload.email
      );
      user ? done(null, user) : done(null, false);
    } catch (error) {
      done(new Unauthorized(error.message), false);
    }
  }
);
export const mapSingleProduct = (product: ProductsModel | any) => ({
  id: product.id,
  name: product.name,
  description: product.description,
  image: product.image,
  category: {
    id: product.category_id,
    name: product.category.name
  },
  size: {
    id: product.size_id,
    name: product.size.name
  },
  type: product.type,
```

```
  price: product.price,
  discount: product.discount,
  rating: product.rating,
  amount: product.amount,
  color: product.color,
  labels: product.labels.map((label: any) => label.label),
  updatedAt: product.updatedAt,
  createdAt: product.createdAt
});
export const mapProductsData = (products: Array<ProductsModel | any>) =>
  products.map((product) => mapSingleProduct(product));
export const mapOrder = (order: any) => ({
  id: order.id,
  city: order.city.name,
  country: order.country.name,
  user: order.user,
  totalPrice: order.total_price,
  products: order.products.map((orderPosition: any) => ({
    amount: orderPosition.amount,
    product: {
      id: orderPosition.product.id,
      name: orderPosition.product.name,
      description: orderPosition.product.description,
      image: orderPosition.product.image,
      price: orderPosition.product.price,
      discount: orderPosition.product.discount,
      amount: orderPosition.product.amount,
      category: orderPosition.product.category.name,
      size: orderPosition.product.size.name
    }
  })),
  createdAt: order.createdAt,
  updatedAt: order.updatedAt
});
```

Seguelize.config.ts // for connect with DB

```
import dotEnv from 'dotenv';
dotEnv.config();
module.exports = {
  development: {
    username: process.env.DB_USER,
    password: process.env.DB_PASS,
    database: process.env.DB_NAME,
    host: process.env.DB_HOST,
    dialect: process.env.DB_DIALECT,
    operatorsAliases: false,
  },
  test: {
    username: process.env.DB_USER,
    password: process.env.DB_PASS,
```

```
    database: process.env.DB_NAME,
    host: process.env.DB_HOST,
    dialect: process.env.DB_DIALECT,
    operatorsAliases: false,
  },
  production: {
    username: process.env.DB_USER,
    password: process.env.DB_PASS,
    database: process.env.DB_NAME,
    host: process.env.DB_HOST,
    dialect: process.env.DB_DIALECT,
    operatorsAliases: false,
  },
};
```

AdminCheck.ts // for midlewares

```
import { Request, Response, NextFunction } from 'express';
import { UsersModel } from '../modules/users/users.model';
import { Unauthorized } from '../common/exeptions';
interface RequestWithUser extends Request {
  user: UsersModel;
}
const adminCheck = (
  req: RequestWithUser | Request,
  res: Response,
  next: NextFunction
) => {
  // eslint-disable-next-line @typescript-eslint/ban-ts-comment
  // @ts-ignore
  if (req.user.role_id === 3 || req.user.role_id === 4) {
    return next();
  }
  next(new Unauthorized('You dont have access to do this.'));
};
export default adminCheck;
```

Create-users-table.js// for create user table

```
import { QueryInterface, DataTypes } from 'sequelize';
import { UsersModel } from '../modules/users/users.model';
import { RolesModel } from '../modules/roles/roles.model';
export async function up(query: QueryInterface) {
  return query.createTable(UsersModel.tableName, {
    id: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true,
      allowNull: false,
    },
    name: {
```

```
      type: DataTypes.STRING(256),
      allowNull: false,
    },
    second_name: {
      type: DataTypes.STRING(256),
      allowNull: true,
    },
    email: {
      type: DataTypes.STRING(256),
      allowNull: false,
      unique: true,
    },
    mobile_phone: {
      type: DataTypes.STRING(20),
      allowNull: true,
      unique: true,
    },
    password: {
      type: DataTypes.STRING(2048),
      allowNull: false,
    },
    is_email_authorized: {
      type: DataTypes.BOOLEAN,
      allowNull: false,
      defaultValue: false,
    },
    role_id: {
      type: DataTypes.INTEGER,
      defaultValue: 1,
      references: {
        model: RolesModel.tableName,
        key: 'id',
      },
      onDelete: 'SET NULL',
      onUpdate: 'CASCADE',
    },
    createdAt: {
      type: DataTypes.DATE,
      allowNull: false,
    },
    updatedAt: {
      type: DataTypes.DATE,
      allowNull: false,
    },
  });
}
export async function down(query: QueryInterface) {
  return query.dropTable(UsersModel.tableName);
}
```

Cloudinary.service.ts // for work with pictures

```
import cloudinary from 'cloudinary';
import isBase64 from 'is-base64';
import UnsupportedMedia from '../../common/exeptions/unsupportedMedia';
import PayloadToLarge from '../../common/exeptions/payloadToLarge';
class CloudinaryService {
  static Cloud = cloudinary.v2;
  public static signedUpload = async (base64_file: string) => {
    if (base64_file === '') {
      return null;
    }
    if (!isBase64(base64_file, { mimeRequired: true })) {
      throw new UnsupportedMedia('Unsupported file type');
    }
    CloudinaryService.Cloud.config({
      cloud_name: process.env.CLOUD_NAME,
      api_key: process.env.CLOUD_API_KEY,
      api_secret: process.env.CLOUD_API_SECRET
    });
    try {
      const image = await CloudinaryService.Cloud.uploader.upload(
        base64_file,
        (err, res) => err || res
      );
      return image;
    } catch (err) {
      throw new PayloadToLarge(
        'Image size is to large. Image should not be biggest than 100mb.'
      );
    }
  };
}
export default CloudinaryService;
```

Products.controller.js // module

```
import { NextFunction, Request, Response } from 'express';
import ProductsService from './products.service';
import { OrderItem } from 'sequelize';
class ProductsController {
  private static SORTS: { [key: string]: Array<string> } = {
    ratingUp: ['rating', 'ASC'],
    ratingDown: ['rating', 'DESC'],
    priceUp: ['price', 'ASC'],
    priceDown: ['price', 'DESC'],
  };

  private static getSort = (sort: any) => {
    if (!sort || typeof sort !== 'string') return undefined;
    const sortArr = sort.split('+');
    const result: OrderItem[] = [];
```

```
    sortArr.forEach((s) => {
      if (ProductsController.SORTS[s]) {
        // eslint-disable-next-line @typescript-eslint/ban-ts-comment
        // @ts-ignore
        result.push(ProductsController.SORTS[s]);
      }
    });
    return result;
  };
  public getAll = async (
    req: Request,
    res: Response,
    next: NextFunction

  ): Promise<void> => {
    try {
      console.log(req.query);
      const products = await ProductsService.getAll(
        Number(req.query.limit) || undefined,
        ProductsController.getSort(req.query.sort_by),
        Number(req.query.price_start) || 0,
        Number(req.query.price_end) || 9999999999,
        // eslint-disable-next-line @typescript-eslint/ban-ts-comment
        // @ts-ignore
        req.query.categories ? req.query.categories?.split(' ') :
undefined,
        // eslint-disable-next-line @typescript-eslint/ban-ts-comment
        // @ts-ignore
        req.query.types ? req.query.types?.split(' ') : undefined,
        // eslint-disable-next-line @typescript-eslint/ban-ts-comment
        // @ts-ignore
        req.query.search ? req.query.search : undefined,
      );
      res.statusCode = 200;
      res.send(products);
    } catch (error) {
      next(error);
    }
  };
  public getCartProducts = async (req: Request, res: Response, next:
NextFunction) => {
    try {
      const products = await ProductsService.getCartProducts(req.body);
      res.statusCode = 200;
      res.send(products);
    } catch (error) {
      next(error);
    }
  }
  public getOnById = async (
    req: Request,
```

```
    res: Response,
    next: NextFunction
  ): Promise<void> => {
    try {
      const product = await
ProductsService.getOneById(Number(req.params.id));
      res.statusCode = 200;
      res.send(product);
    } catch (error) {
      next(error);
    }
  };

  public createOne = async (
    req: Request,
    res: Response,
    next: NextFunction
  ): Promise<void> => {
    try {
      const product = await ProductsService.createOne(req.body);
      res.statusCode = 201;
      res.send(product);
    } catch (error) {
      next(error);
    }
  };

  public updateOneById = async (
    req: Request,
    res: Response,
    next: NextFunction
  ): Promise<void> => {
    try {
      const product = await ProductsService.updateOneById(
        Number(req.params.id),
        req.body
      );
      res.statusCode = 200;
      res.send(product);
    } catch (error) {
      next(error);
    }
  }

  public deleteOneById = async (
    req: Request,
    res: Response,
    next: NextFunction
  ): Promise<void> => {
    try {
      await ProductsService.deleteOneByID(Number(req.params.id));
```

```
      res.statusCode = 200;
      res.send();
    } catch (error) {
      next(error);
    }
  };
}
const productsController: ProductsController = new ProductsController();
export default productsController;
```

ProductRoutes.ts // module

```
import { Router } from 'express';
import ProductsController from './products.controller';
import withAuth from '../../middlewares/withAuth';
import adminCheck from '../../middlewares/adminCheck';
import createValidator from '../../middlewares/create-validator';
import { updateProductDto, createProductDto, getCartProductsDto } from
'./products.dto';
export const router: Router = Router();
router.get('/', ProductsController.getAll);
router.get('/:id', ProductsController.getOnById);
router.post(
  '/',
  createValidator(createProductDto),
  withAuth,
  adminCheck,
  ProductsController.createOne
);
router.post('/cart', createValidator(getCartProductsDto),
ProductsController.getCartProducts)
router.put(
  '/:id',
  createValidator(updateProductDto),
  withAuth,
  adminCheck,
  ProductsController.updateOneById
);
router.delete('/:id', withAuth, adminCheck,
ProductsController.deleteOneById);
```

Products.service.ts // module

```
import { Op, OrderItem, Transaction } from 'sequelize';
// eslint-disable-next-line @typescript-eslint/no-var-
requires,@typescript-eslint/no-unsafe-assignment
import isURL from 'is-url';
import isBase64 from 'is-base64';
import { ProductsModel } from './products.model';
import { ProductLabelModel } from './product-label.model';
import { LabelsModel } from '../labels/labels.model';
import { mapProductsData, mapSingleProduct } from '../../common/helpers';
```

```
import { CategoriesModel } from '../categories/categories.model';
import { SizesModel } from '../sizes/sizes.model';
import { sequelize } from '../../database';
import CloudinaryService from '../Cloudinary/cloudinary.service';
import UnsupportedMedia from '../../common/exeptions/unsupportedMedia';
import { TypesModel } from '../types/types.model';

interface ProductDataWithLabels extends ProductsModel {
  labels: Array<number>;
}
class ProductsService {
  public async getAll(
    limit: number | undefined = undefined,
    order: OrderItem[] | undefined = undefined,
    price_start: number | any | undefined = undefined,
    price_end: number | any | undefined = undefined,
    category: string[] | number[] | undefined = undefined,
    type: string[] | number[] | undefined = undefined,
    search: string | undefined = undefined,
  ) {
    const whereSearch = (() => {
      let result = {
        price: {
          [Op.gte]: price_start,
          [Op.lte]: price_end,
        }
      };
      if (category && category.length) {
        result = {
          ...result,
          // eslint-disable-next-line @typescript-eslint/ban-ts-comment
          // @ts-ignore
          category_id: category,
        };
      }
      if (type && type.length) {
        result = {
          ...result,
          // eslint-disable-next-line @typescript-eslint/ban-ts-comment
          // @ts-ignore
          type_id: type,
        };
      }

      if (search && search !== '') {
        result = {
          ...result,
          // eslint-disable-next-line @typescript-eslint/ban-ts-comment
          // @ts-ignore
          name: {
            [Op.iLike]: `%${search}%`,
```

```
          },
        };
      }
      return result;
    })();
    console.log(whereSearch)
    const products = await ProductsModel.findAll({
      limit,
      include: [
        {
          model: ProductLabelModel,
          attributes: ['id'],
          as: 'labels',
          include: [
            {
              model: LabelsModel,
              attributes: ['id', 'name', 'color'],
              as: 'label'
            }
          ]
        },

        { model: CategoriesModel, as: 'category', attributes: ['name'] },
        { model: TypesModel, as: 'type', attributes: ['id', 'name'] },
        { model: SizesModel, as: 'size', attributes: ['name'] }
      ],
      where: {
        ...whereSearch,
      },
      order
    });
    const mappedProducts = mapProductsData(products);
    return mappedProducts;
  }
  public async getCartProducts(data: Array<number>) {
    const products = await ProductsModel.findAll({ where: { id: data },
include: [
        { model: SizesModel, as: 'size', attributes: ['name'] },
        { model: TypesModel, as: 'type', attributes: ['id', 'name'] },
      ]});
    return products;
  }
  public async getOneById(productId: number, transaction?: Transaction) {
    const product = await ProductsModel.findOne({
      where: { id: productId },
      include: [
        {
          model: ProductLabelModel,
          attributes: ['id'],
          as: 'labels',
          include: [
```

```
          {
            model: LabelsModel,
            attributes: ['id', 'name', 'color'],
            as: 'label'
          }
        ]
      },
      { model: CategoriesModel, as: 'category', attributes: ['name'] },
      { model: TypesModel, as: 'type', attributes: ['id', 'name'] },
      { model: SizesModel, as: 'size', attributes: ['name'] }
    ],
    transaction
  });
  if (!product) {
    throw new Error('Product not found');
  }
  const mappedProduct = mapSingleProduct(product);
  return mappedProduct;
}
public async createOne(productData: ProductDataWithLabels) {
  const { labels } = productData;
  // eslint-disable-next-line @typescript-eslint/ban-ts-comment
  // @ts-ignore
  delete productData.labels;
  const transaction = await sequelize.transaction();
  try {
    const image = await CloudinaryService.signedUpload(
      productData.image || ''
    );

    const product = await ProductsModel.create(
      { ...productData, image: image?.url },
      { transaction }
    );

    for (const labelId of labels) {
      await ProductLabelModel.create(
        {
          product_id: product.id,
          label_id: labelId
        },
        { transaction }
      );
    }
    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-ignore
    const newProduct = await this.getOneById(product.id, transaction);
    await transaction.commit();
    return newProduct;
  } catch (error) {
    await transaction.rollback();
```

```
      throw error;
    }
  }
  public async updateOneById(
    productId: number,
    productData: ProductDataWithLabels
  ) {
    const { labels, image } = productData;
    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-ignore
    delete productData.labels;
    // eslint-disable-next-line @typescript-eslint/no-unsafe-call
    if (
      !isURL(image || '') &&
      !isBase64(image || '', { mimeRequired: true }) &&
      image !== ''
    ) {
      throw new UnsupportedMedia('Unsupported file type');
    }

    const transaction = await sequelize.transaction();
    try {
      let newImage = productData.image;
      if (isBase64(image || '', { mimeRequired: true })) {
        const uploadedImage = await CloudinaryService.signedUpload(image
|| '');
        newImage = uploadedImage?.url;
      }
      await ProductLabelModel.destroy({
        where: { product_id: productId },
        transaction
      });
      for (const labelId of labels) {
        await ProductLabelModel.create(
          {
            product_id: productId,
            label_id: labelId
          },
          { transaction }
        );
      }
      await ProductsModel.update(
        { ...productData, image: newImage },
        {
          where: { id: productId },
          transaction
        }
      );
      const product = await this.getOneById(productId);
      await transaction.commit();
      return product;
```

```
    } catch (err) {
      await transaction.rollback();
      throw err;
    }
  }
  public async deleteOneByID(productId: number | string) {
    const transaction = await sequelize.transaction();
    try {
      await ProductLabelModel.destroy({
        where: { product_id: productId },
        transaction
      });
      await ProductsModel.destroy({ where: { id: productId }, transaction
});
      await transaction.commit();
      return true;
    } catch (err) {
      await transaction.rollback();
      throw err;
    }
  }
}
const productsService = new ProductsService();
export default productsService;
```

User.controller.ts // module

```
import { NextFunction, Request, Response } from 'express';
import UsersService from './users.service';
import { USER_CREATE_SUCCESS, USER_LOGIN_SUCCESS } from './constants';
import { UsersModel } from './users.model';
class UsersController {

  public registration = async (
    req: Request,
    res: Response,
    next: NextFunction
  ): Promise<void> => {
    try {
      await UsersService.registration(req.body);
      res.statusCode = 201;
      res.statusMessage = USER_CREATE_SUCCESS;
      res.send({
        status: 201,
        message: USER_CREATE_SUCCESS
      });
    } catch (error) {
      next(error);
    }
  };
  public login = async (
```

```
    req: Request,
    res: Response,
    next: NextFunction
  ): Promise<void> => {
    try {
      const token = await UsersService.login(req.body);
      res.statusCode = 201;
      res.statusMessage = USER_LOGIN_SUCCESS;
      res.send(`Bearer ${token}`);
    } catch (error) {
      next(error);
    }
  };
  public getAllUsers = async (
    req: Request,
    res: Response,
    next: NextFunction
  ): Promise<void> => {
    try {
      const users: Array<UsersModel> = await UsersService.getAll();
      res.statusCode = 200;
      res.send(users);
    } catch (error) {
      next(error);
    }
  };
  public getUserByToken = async (
    req: Request,
    res: Response,
    next: NextFunction
  ): Promise<void> => {
    try {
      // eslint-disable-next-line @typescript-eslint/ban-ts-comment
      // @ts-ignore
      const user: UsersModel = await
UsersService.getOneByToken(req.user.id);
      res.statusCode = 200;
      res.send(user);
    } catch (error) {
      next(error);
    }
  };
  public updateUser = async (
    req: Request,
    res: Response,
    next: NextFunction
  ): Promise<void> => {
    try {
      // eslint-disable-next-line @typescript-eslint/ban-ts-comment
      // @ts-ignore
      const user: UsersModel = await UsersService.updateUser(req.user.id,
```

```
req.body);
      res.statusCode = 200;
      res.send(user);
    } catch (error) {
      next(error);
    }
  }
}
const userController: UsersController = new UsersController();
export default userController;
```

Users.servise.ts // module

```
import { compareSync } from 'bcrypt';
import { sign } from 'jsonwebtoken';
import { autCfg } from '../../config/auth.config';
import { UsersModel } from './users.model';
import { BadRequest } from '../../common/exeptions';
import {
  USER_ALREADY_EXISTS,
  USER_NOT_FOUND
} from '../../common/exeptions/errorMessages';
import { hashPassword } from '../../common/helpers';

import Unauthorized from '../../common/exeptions/unauthorized';
import { RolesModel } from '../roles/roles.model';
class UsersService {

  public async registration(userData: User) {
    const { email } = userData;
    const user = await UsersModel.findOne({ where: { email } });
    if (user) {
      throw new BadRequest(USER_ALREADY_EXISTS);
    }
    const newUser = new UsersModel(userData);
    newUser.password = hashPassword(newUser.password);
    return newUser.save();
  }
  public async login({ email, password }: Login) {
    const user: UsersModel | null = await UsersModel.findOne({
      where: { email }
    });
    if (!user || !compareSync(password, user.password || '')) {
      throw new Unauthorized(USER_NOT_FOUND);
    }
    const token: string = sign(
      { id: user.id, email: user.email },
      autCfg.secretKey,
      { expiresIn: autCfg.expiresIn }
    );
    return token;
```

```
  }
  public async getOneByEmail(userEmail: string) {
    const user = await UsersModel.findOne({ where: { email: userEmail }
});
    return user;
  }
  public async getAll() {
    const users = await UsersModel.findAll({
      attributes: [
        'id',
        'name',
        'second_name',
        'email',
        'mobile_phone',
        'is_email_authorized',
        'createdAt',
        'updatedAt'
      ],
      include: [{ model: RolesModel, as: 'role', attributes: ['name'] }]
    });
    return users;
  }
  public async getOneByToken(userId: number) {
    const user = await UsersModel.findOne({
      where: { id: userId },
      attributes: [
        'id',
        'name',
        'second_name',
        'email',
        'mobile_phone',
        'is_email_authorized'
      ],
      include: [{ model: RolesModel, as: 'role', attributes: ['id',
'name'] }]
    });
    return user;
  }
  public async updateUser(userId: number, data: UsersModel) {
    const user = await UsersModel.update(data, { where: { id: userId }});
    return user;
  }
}
const usersService: UsersService = new UsersService();
export default usersService;
```

users.model.ts // module

```
import { DataTypes, Model, Sequelize } from 'sequelize';
import { sequelize } from '../../database';
```

```
import { RolesModel } from '../roles/roles.model';
export class UsersModel extends Model {
  public static readonly tableName: string = 'users';
  public id: number | undefined;
  public name: string | undefined;
  public second_name: string | undefined;
  public email: string | undefined;
  public mobile_phone: string | undefined;
  public password: string | undefined;
  public is_email_authorized: boolean | undefined;
  public role_id: number | undefined;
  public static prepareInit(seq: Sequelize) {
    this.init(
      {
        id: {
          type: DataTypes.INTEGER,
          primaryKey: true,
          allowNull: false,
          autoIncrement: true,
        },

        name: {
          type: DataTypes.STRING(256),
          allowNull: false,
        },

        second_name: {
          type: DataTypes.STRING(256),
          allowNull: true,
        },

        email: {
          type: DataTypes.STRING(256),
          allowNull: false,
          unique: true,
        },

        mobile_phone: {
          type: DataTypes.STRING(20),
          allowNull: true,
          unique: true,
        },

        password: {
          type: DataTypes.STRING(2048),
          allowNull: false,
        },

        is_email_authorized: {
          type: DataTypes.BOOLEAN,
```

```
          allowNull: false,
          defaultValue: false,
        },

        role_id: {
          type: DataTypes.INTEGER,
          defaultValue: 1,
        }

      },
      {
        sequelize,
        tableName: this.tableName,
      },
    );
  }
}
UsersModel.prepareInit(sequelize);
UsersModel.belongsTo(RolesModel, {
  foreignKey: 'role_id',
  as: 'role'
});
RolesModel.hasMany(UsersModel, {
  foreignKey: 'role_id',
  as: 'users'
});
```

Orders.controller.ts // module

```
import { NextFunction, Request, Response } from 'express';
import OrdersService from './orders.service';
class OrdersController {
  public getAll = async (req: Request, res: Response, next: NextFunction)
=> {
    try {
      const orders = await OrdersService.getALl();
      res.statusCode = 200;
      res.send(orders);
    } catch (err) {
      next(err);
    }
  };
  public getOneById = async (
    req: Request,
    res: Response,
    next: NextFunction
  ) => {
    try {
      const order = await
OrdersService.getOneById(Number(req.params.id));
      res.statusCode = 200;
```

```
      res.send(order);
    } catch (err) {
      next(err);
    }
  };

  public getAllByUser = async (
    req: Request,
    res: Response,
    next: NextFunction
  ) => {
    try {
      // eslint-disable-next-line @typescript-eslint/ban-ts-comment
      // @ts-ignore
      const orders = await
OrdersService.getAllByUser(Number(req.user.id));
      res.statusCode = 200;
      res.send(orders);
    } catch (err) {
      next(err);
    }
  };
  public createOne = async (
    req: Request,
    res: Response,
    next: NextFunction
  ) => {
    try {
      const order = await OrdersService.createOne(req.body);
      res.statusCode = 201;
      res.send(order);
    } catch (err) {
      next(err);
    }
  }
  public deleteOneById = async (
    req: Request,
    res: Response,
    next: NextFunction
  ) => {
    try {
      await OrdersService.deleteOneById(Number(req.params.id));
      res.statusCode = 200;
      res.send();
    } catch (err) {
      next(err);
    }
  };
  public deleteProductFormOrder = (
    req: Request,
    res: Response,
```

```
    next: NextFunction
  ) => {
    try {
      res.statusCode = 200;
      res.send(`${req.params.orderId}:${req.params.productId}`)
    } catch (err) {
      next(err);
    }
  };
}
const ordersController: OrdersController = new OrdersController();
export default ordersController;
```

orders.model.ts // module

```
import { DataTypes, Model, Sequelize } from 'sequelize';
import { sequelize } from '../../database';
import { OrderProductsModel } from './order-products.model';
import { ProductsModel } from '../products/products.model';
import { UsersModel } from '../users/users.model';

import { CountriesModel } from '../counties/countries.model';
import { CitiesModel } from '../cities/cities.model';
export class OrdersModel extends Model {
  public static readonly tableName: string = 'orders';
  public id: number | undefined;
  public user_id: number | undefined;
  public country_id: number | undefined;
  public city_id: number | undefined;
  public total_price: string | undefined;
  public static prepareInit(seq: Sequelize) {
    this.init(
      {
        id: {
          type: DataTypes.INTEGER,
          primaryKey: true,
          allowNull: false,
          autoIncrement: true,
        },
        user_id: {
          type: DataTypes.INTEGER,
          allowNull: false,
        },

        country_id: {
          type: DataTypes.INTEGER,
          allowNull: false,
        },

        city_id: {
          type: DataTypes.INTEGER,
```

```
          allowNull: false,
        },

        total_price: {
          type: DataTypes.STRING(256),
          allowNull: false,
        },
      },
      {
        sequelize,
        tableName: this.tableName,
      },
    );
  }
}
OrdersModel.prepareInit(sequelize);
OrdersModel.belongsTo(UsersModel, {
  foreignKey: 'user_id',
  as: 'user',
});

UsersModel.hasMany(OrdersModel, {
  foreignKey: 'user_id',
  as: 'orders',
});


OrdersModel.belongsTo(CountriesModel, {
  foreignKey: 'country_id',
  as: 'country',
});
CountriesModel.hasMany(OrdersModel, {
  foreignKey: 'country_id',
  as: 'orders',
});

OrdersModel.belongsTo(CitiesModel, {
  foreignKey: 'city_id',
  as: 'city',
});

CitiesModel.hasMany(OrdersModel, {
  foreignKey: 'country_id',
  as: 'orders',
});


OrdersModel.hasMany(OrderProductsModel, {
  foreignKey: 'order_id',
  as: 'products',
});
```

```
// eslint-disable-next-line @typescript-eslint/ban-ts-comment
// @ts-ignore
OrdersModel.product = OrderProductsModel.belongsTo(ProductsModel, {
  foreignKeyConstraint: true,
  foreignKey: 'product_id',
  targetKey: 'id',
  as: 'product',
});
```

Labels.service.ts // module

```
import { LabelsModel } from './labels.model';
import { BadRequest } from '../../common/exeptions';
class LabelsService {
  public async getAll() {
    const labels = await LabelsModel.findAll();
    return labels;
  }
  public async getOneById(labelId: number) {
    const label = await LabelsModel.findOne({ where: { id: labelId } });
    return label;
  }
  public async createOne(labelData: LabelsModel) {
    const labelsWithColors = await LabelsModel.findAll({
      where: { color: labelData.color }
    });
    if (labelsWithColors.length) {
      throw new BadRequest('Color should be unique');
    }
    const label = await LabelsModel.create(labelData);
    return label;
  }
  public async updateOneById(labelData: LabelsModel, labelId: number) {
    const labelsWithColors = await LabelsModel.findAll({
      where: { color: labelData.color }
    });

    if (labelsWithColors.length) {
      const sortedLabels = labelsWithColors.filter(
        (label) => Number(label.id) !== Number(labelId)
      );
      if (sortedLabels.length) {
        throw new BadRequest('Color should be unique');
      }
    }
    const label = await LabelsModel.update(labelData, {
      where: { id: labelId }
    });
    return label;
  }
  public async deleteOneById(labelId: number) {
```

```
    await LabelsModel.destroy({ where: { id: labelId } });
    return true;
  }
}
const labelsService: LabelsService = new LabelsService();
export default labelsService;
```

feedbacks.controller.ts// module

```
import { NextFunction, Request, Response } from 'express';
import FeedbacksService from './feedbacks.service';

class FeedbacksController {
  public getAll = async (req: Request, res: Response, next: NextFunction)
=> {
    try {
      const feedbacks = await FeedbacksService.getAll();
      res.statusCode = 200;
      res.send(feedbacks);
    } catch (err) {
      next(err);
    }
  };

  public createOne = async (
    req: Request,
    res: Response,
    next: NextFunction
  ) => {
    try {
      const feedback = await FeedbacksService.createOne(req.body);
      res.statusCode = 201;
      res.send(feedback);
    } catch (err) {
      next(err);
    }
  };

  public sendAnswer = async (
    req: Request,
    res: Response,
    next: NextFunction
  ) => {
    try {
      const feedback = await FeedbacksService.sendResponse(
        req.params.id,
        req.body
      );
      res.statusCode = feedback ? 200 : 500;
      res.send(feedback);
    } catch (err) {
```

```
      next(err);
    }
  };
}

const feedbacksController = new FeedbacksController();

export default feedbacksController;
```

feedbacks.model.ts // module

```
import { DataTypes, Model, Sequelize } from 'sequelize';
import { sequelize } from '../../database';

export class FeedbacksModel extends Model {
  public static readonly tableName: string = 'feedbacks';

  public id: number | undefined;
  public name: string | undefined;
  public email: string | undefined;
  public comment: string | undefined;

  public static prepareInit(seq: Sequelize) {
    this.init(
      {
        id: {
          type: DataTypes.INTEGER,
          primaryKey: true,
          allowNull: false,
          autoIncrement: true,
        },
        name: {
          type: DataTypes.STRING(256),
          allowNull: false,
        },
        email: {
          type: DataTypes.STRING(256),
          allowNull: false,
        },
        comment: {
          type: DataTypes.TEXT(),
          allowNull: false,
        },
        status: {
          type: DataTypes.STRING(24),
          allowNull: false,
          defaultValue: 'not_answered'
        }
      },
      {
```

```
        sequelize,
        tableName: this.tableName,
      },
    );
  }
}

FeedbacksModel.prepareInit(sequelize);
```

feedbacks.service.ts // module

```
import { FeedbacksModel } from './feedbacks.model';
import Mailer from '../../common/mailer';
import { sequelize } from '../../database';

class FeedbacksService {
  public async getAll() {
    const feedbacks = await FeedbacksModel.findAll();
    return feedbacks;
  }

  public async createOne(data: FeedbacksModel) {
    const feedback = await FeedbacksModel.create({
      ...data,
      status: 'not_answered'
    });
    return feedback;
  }

  // eslint-disable-next-line
  public async sendResponse(id: string | number, data: any) {
    // eslint-disable-next-line
    const transaction = await sequelize.transaction();
    try {
      const feedback = await FeedbacksModel.findOne({
        where: { id },
        transaction
      });
      if (feedback) {
        // eslint-disable-next-line
        Mailer.fireMessage(feedback.email, data.message, data.title);
        await FeedbacksModel.update(
          { status: 'answered' },
          { where: { id }, transaction }
        );
        await transaction.commit();
        return true;
      } else {
        return false;
      }
    } catch (err) {
```

```
      await transaction.rollback();
      return false;
    }
  }
}

const feedbackService = new FeedbacksService();

export default feedbackService;
```

package.json

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "NKN Project rest API",
  "main": "build/app.js",
  "scripts": {
    "pretter": "prettier --check . --ignore-path ./.prettierignore",
    "pretter:fix": "prettier . --write --ignore-path ./.prettierignore",
    "eslint": "eslint . --ext .js,.jsx,.ts,.tsx",
    "start": "npm run build && node build/app.js",
    "start:linux": "npm run build:linux && node build/app.js",
    "start:watch": "nodemon",
    "build": "npm run pretter && npm run eslint && rd /s /q \"./build\"
&& tsc",
    "build:fast": "rd /s /q \"./build\" && tsc && node build/app.js",
    "build:linux": "rm -rf \"./build\" && tsc && node build/app.js",
    "migrate": "npx sequelize-cli db:migrate",
    "migrate:undo": "npx sequelize-cli db:migrate:undo",
    "migrate:undo:all": "npx sequelize-cli db:migrate:undo:all",
    "seed": "npx sequelize-cli db:seed:all",
    "seed:undo:all": "npx sequelize-cli db:seed:undo:all"
  },
  "repository": {
    "type": "git",
    "url": "git+ssh://git@gitlab.com/nkn.inc/backend.git"
  },
  "author": "NKN Co.",
  "license": "ISC",
  "bugs": {
    "url": "https://gitlab.com/nkn.inc/backend/issues"
  },
  "homepage": "https://gitlab.com/nkn.inc/backend#readme",
  "devDependencies": {
    "@types/bcrypt": "^3.0.0",
    "@types/cors": "^2.8.8",
    "@types/express": "^4.17.8",
    "@types/is-base64": "^1.1.0",
    "@types/is-url": "^1.2.29",
```

```
    "@types/jsonwebtoken": "^8.5.0",
    "@types/node": "^14.14.6",
    "@types/nodemailer": "^6.4.1",
    "@types/passport": "^1.0.4",
    "@types/passport-jwt": "^3.0.3",
    "@types/yup": "^0.29.9",
    "@typescript-eslint/eslint-plugin": "^4.6.1",
    "@typescript-eslint/parser": "^4.6.1",
    "eslint": "^7.12.1",
    "eslint-plugin-import": "^2.22.1",
    "nodemon": "^2.0.6",
    "prettier": "2.1.2",
    "sequelize-cli": "^6.2.0",
    "ts-node": "^9.0.0",
    "typescript": "^4.0.5"
  },
  "dependencies": {
    "bcrypt": "^5.0.0",
    "body-parser": "^1.19.0",
    "cloudinary": "^1.25.0",
    "cors": "^2.8.5",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "is-base64": "^1.1.0",
    "is-url": "^1.2.4",
    "isurl": "^4.0.2",
    "jsonwebtoken": "^8.5.1",
    "nodemailer": "^6.6.0",
    "passport": "^0.4.1",
    "passport-jwt": "^4.0.0",
    "pg": "^8.4.2",
    "sequelize": "^6.3.5",
    "yup": "^0.29.3"
  },
  "nodemonConfig": {
    "ignoe": [
      "**/*.test.ts",
      "**/*.spec.ts",
      ".git",
      "node_modules",
      ".idea",
      ".vscode"
    ],
    "watch": [
      "src"
    ],
    "exec": "npm run fast-build",
    "ext": "ts"
  }
}
```

app.ts

```
import express, { Application } from 'express';
import bodyParser from 'body-parser';
import cors from 'cors';
import dotEnv from 'dotenv';
import passport from 'passport';
import './database';

import { router as UsersRouter } from
'./modules/users/users.routes';
import { router as RolesRouter } from
'./modules/roles/roles.routes';
import { router as ProductsRouter } from
'./modules/products/products.routes';
import { router as LabelsRouter } from
'./modules/labels/labels.routes';
import { router as SizesRouter } from
'./modules/sizes/sizes.routes';
import { router as CountriesRouter } from
'./modules/counties/countries.routes';
import { router as CitiesRouter } from
'./modules/cities/cities.routes';
import { router as CategoriesRouter } from
'./modules/categories/categories.router';
import { router as OrdersRouter } from
'./modules/orders/orders.routes';
import { router as FeedbacksRouter } from
'./modules/feedbacks/feedbacks.routes';
import { router as TypesRouter } from
'./modules/types/types.routes';
import { router as DashboardRouter } from
'./modules/dashboard/dashboard.routes';
import { errorHandler } from
'./middlewares/error-handler';
import { strategy } from './common/helpers';

dotEnv.config();
passport.use(strategy);

const App: Application = express();

App.use(bodyParser.json({ limit: '100mb' }));
App.use(cors());
App.use(passport.initialize());

App.use('/api-v1/users', UsersRouter);
App.use('/api-v1/roles', RolesRouter);
App.use('/api-v1/categories', CategoriesRouter);
App.use('/api-v1/products', ProductsRouter);
App.use('/api-v1/labels', LabelsRouter);
```

```
App.use('/api-v1/sizes', SizesRouter);
App.use('/api-v1/countries', CountriesRouter);
App.use('/api-v1/cities', CitiesRouter);
App.use('/api-v1/orders', OrdersRouter);
App.use('/api-v1/feedbacks', FeedbacksRouter);
App.use('/api-v1/types', TypesRouter);
App.use('/api-v1/dashboard', DashboardRouter);

App.use(errorHandler);
const PORT = process.env.API_PORT || 3000;
App.listen(PORT, () => {
  console.log(`Server is running in http://localhost:${PORT}`);
});
```

…

Інші файли знаходяться на електронному носії.

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
«Розробка та адміністрування back-end частини Інтернет магазину з продажу одягу»
студента групи 122-17-1 Панченко Антона Андрійовича

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н **Л. В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом Панченко.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом Панченко.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
diplom.zip	Архів. Містить коди програми.
Презентація	
Презентація Панченко.ppt	Презентація кваліфікаційної роботи.