

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Вебдодаток для продажу ветеринарних товарів з CMS Strapi:
користувачі, адмін-панель, розподіл прав»**

виконав студент групи ІПЗ-20-2

БЕРЛІНОВ Андрій Олександрович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:

ЄФРЕМОВ Юрій Миколайович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БЕРЛІНОВ Андрій Олександрович**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Вебдодаток для продажу ветеринарних товарів з CMS Strapi: користувачі, адмін-панель, розподіл прав»,**

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**

Вихідні дані для роботи: **методи та засоби розробки веб-додатків, аналіз проблематики додатку для продажу ветеринарних товарів, використання для розробки веб-додатку CMS Strapi та NestJS.**

РЕФЕРАТ

Випускна робота бакалавра включає розробку вебдодатку для продажу ветеринарних товарів та пояснювальну записку. Пояснювальна записка містить 66 сторінки, 17 ілюстрацій та 12 таблиць.

Метою роботи є створення веб-додатку з використанням Nest.js та CMS Strapi, що покращує доступність адміністратора до управління сайтом. Додаток забезпечує зручність користування та підвищує ефективність процесу продажу товарів, дозволяючи власникам легко знаходити і купувати необхідні товари.

В роботі визначено основні завдання розробки системи, проаналізовано аналоги програмного продукту та обрано засоби реалізації. Проведено розробку структури бази даних, вибрано сучасний стек технологій та розроблено концепцію програмного продукту, проведено деплоймент. Описано програмну реалізацію та результати тестування.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, ТВАРИНИ, NEST JS, STRAPI, DOCKER, TRAEFIK, ЗРУЧНІСТЬ, АДМІН-ПАНЕЛЬ, КОРИСТУВАЧІ.

					ІПЗ.КР.Б-121-24-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка веб додатку з продажу ветеринарних товарів	Літ.	Арк.	Аркушів
Розроб.		Берлінов А.О.						
Керівник		Власенко О.В.					3	65
						Житомирська політехніка група ІПЗ-20-2		
Н. контр.		Гичина О.П.						
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's thesis involves the development of a web application for selling veterinary products and includes an explanatory note. The explanatory note comprises 66 pages, 17 illustrations, and 12 tables. The objective of the project is to create a web application using Nest.js and CMS Strapi, enhancing administrative management of the site. The application ensures user convenience and improves the efficiency of product sales, enabling owners to easily find and purchase necessary items.

The work outlines the primary development tasks, analyzes comparable products, and selects implementation tools. It covers database structure development, modern technology stack selection, and product concept creation, followed by deployment. The thesis also describes the software implementation and testing results.

KEYWORDS: WEB APPLICATION, PETS, NEST JS, STRAPI, DOCKER, TRAEFIK, CONVENIENCE, ADMIN PANEL, USERS.

					ИПЗ.КР.Б-121-24-ПЗ	Арк.
						4

ЗМІСТ

ABSTRACT.....	4
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	9
1.1 Постановка задачі.....	9
1.2 Аналіз аналогів програмного продукту	11
1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	16
Висновки до першого розділу.	20
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	22
2.1 Визначення варіантів використання та структури системи	22
2.2 Розробка бази даних системи	31
2.3 Проектування та реалізація алгоритмів роботи додатку	34
2.4 Реалізація функціоналу додатку	35
Висновки до 2 розділу:	41
РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ.....	43
3.1 Порядок встановлення та налаштування параметрів додатку	43
3.2 Структура інтерфейсу веб-додатку та порядок роботи.....	47
3.3 Тестування роботи веб-додатку.....	52
Висновки до третього розділу	54
ВИСНОВКИ	55
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	56
ДОДАТКИ	58
Додаток А	59

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

CMS – система управління контентом сайту

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						6

ВСТУП

Актуальність теми полягає в необхідності створення ефективного вебдодатку для продажу ветеринарних товарів у зв'язку із зростаючим попитом на ветеринарні послуги та товари для домашніх улюбленців. Останніми роками спостерігається значне зростання кількості власників домашніх тварин, які шукають зручні та надійні способи придбання ветеринарних товарів. Традиційні методи продажу товарів через фізичні магазини поступаються місцем електронній комерції, яка надає можливість швидкого та зручного доступу до широкого асортименту продукції. Розробка спеціалізованого вебдодатку дозволяє не тільки поліпшити обслуговування клієнтів, але й розширити ринок збуту та забезпечити більш ефективне управління запасами і логістикою.

Метою розробки випускної кваліфікаційної роботи бакалавра є розробка вебдодатку для продажу ветеринарних товарів. Для реалізації поставленої мети потрібно виконати наступні завдання:

1. Проаналізувати існуючі рішення, зробити додаток зручнішим для користувача.
2. Визначити вимоги щодо функціоналу додатку.
3. Визначити засоби розробки та проектування проєкту.
4. Розробити додаток, його основний функціонал.
5. Протестувати додаток, визначити його ефективність.

Об'єктом дослідження є електронний магазин у сфері продажу ветеринарних товарів, що включає в себе управління товарами, можливість зробити замовлення і адміністрування магазином.

Предмет дослідження: Предметом дослідження є функціональність вебдодатку для продажу ветеринарних товарів, його структура, методи інтеграції з платіжними системами, засоби забезпечення безпеки даних користувачів та оптимізація процесу покупки і доставки товарів. Дослідження також включає аналіз переваг та недоліків використання вебдодатку у порівнянні з традиційними методами продажу ветеринарних товарів.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						7

Для розробки вебдодатку було використано вебфреймворки NextJS та NestJS, а також для створення швидких і легко керованих API JavaScript використовувався Strapi, з об'єктно-реляційною системою керування базами даних – PostgreSQL.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						8

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Постановка задачі

Метою кваліфікаційної роботи бакалавра є створення вебдодатку для продажу ветеринарних товарів, що забезпечить функціональність, фільтрацію, сортування товарів та надасть допомогу користувачам у процесі здійснення покупок.

Вебдодаток має на меті підвищення доступності ветеринарних товарів для власників домашніх улюбленців через інтернет, забезпечення зручності користування та підвищення ефективності процесу продажу продукції. Це дозволить власникам домашніх тварин без зусиль знаходити та купувати необхідні товари, враховуючи їхні потреби.

Основні завдання, що потребують вирішення для досягнення мети, включають:

1. Аналіз ринку та вимог до вебдодатку:

- Провести дослідження наявних рішень для електронної комерції у сфері ветеринарних товарів.
- Визначити ключові вимоги до функціональності, фільтрації, сортування товарів та підтримки користувачів.

2. Проектування архітектури вебдодатку:

- Розробити архітектуру клієнтської та серверної частин вебдодатку.
- Визначити структуру бази даних для зберігання інформації про товари, користувачів та замовлення.

3. Розробка основного функціоналу:

- Розробити контролери для реалізації CRUD для кожної сутності.
- Імплементувати систему фільтрації та сортування товарів за різними критеріями (ціна, категорія, популярність).

4. Інтеграція з платіжними системами:

- Забезпечити можливість здійснення безпечних платежів через популярні платіжні системи.
- Реалізувати систему обробки замовлень та відстеження їх статусу.

Основними етапами розробки вебдодатку є побудова архітектури серверної частини додатку з використанням NestJS, яка включає визначення бібліотек для реалізації серверної частини додатку, розробку API для взаємодії з клієнтською частиною та створення модулів для роботи з товарами, користувачами та замовленнями. Наступним етапом є розробка системи авторизації та ролей користувачів, яка включає реалізацію механізму реєстрації та авторизації користувачів і впровадження системи ролей для адміністраторів та користувачів. Використання Strapi для управління контентом передбачає налаштування Strapi для створення та керування швидкими і гнучкими API, а також інтеграцію Strapi з базою даних PostgreSQL. Інтеграція з платіжними системами включає підключення до популярних платіжних систем і реалізацію механізму обробки платежів та замовлень. Розробка основного функціоналу охоплює створення логіки роботи з товарами (додавання, редагування, видалення), реалізацію системи фільтрації та сортування товарів, а також впровадження механізму для надання допомоги користувачам (онлайн-чат). Завершальним етапом є розгортання проекту, яке включає вибір платформи для хостингу (наприклад, Heroku, AWS, DigitalOcean), налаштування середовища для розгортання, розгортання додатку на обраному сервері та забезпечення безперервної інтеграції та доставки (CI/CD) для автоматизації процесу оновлення додатку. Результатом виконання плану дій є створення вебдодатку, який дозволить користувачам зручно та швидко знаходити, фільтрувати, сортувати та купувати ветеринарні товари, отримуючи при цьому необхідну допомогу та підтримку. Підсумковим продуктом буде завантажений на сервер застосунок, розроблений з використанням NestJS для серверної частини, з адміністраторським інтерфейсом в Strapi для керування контентом та з клієнтським інтерфейсом, який забезпечить зручне та ефективне користування для кінцевих користувачів.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						10

- **Консультації спеціалістів:** Можливість отримання професійних консультацій.
- **Швидке замовлення і доставка:** Оперативне оброблення замовлень та доставка по всій Україні.
- **Акції та знижки:** Регулярні пропозиції та спеціальні умови для клієнтів.
- **Підтримка оптових клієнтів:** Спеціальні умови для оптових покупців та ветеринарних клінік.

Недоліки:

- **Дизайн і навігація:** Деякі елементи дизайну можуть виглядати застарілими, а навігація не завжди інтуїтивно зрозуміла.
- **Швидкість завантаження:** Можливі затримки при завантаженні сторінок, що може вплинути на користувацький досвід.
- **Мобільна версія:** Мобільна версія сайту може мати обмежену функціональність або виглядати менш зручною у порівнянні з настільною.
- **Інформаційне наповнення:** Не завжди достатньо детальної інформації про товари та їх характеристики

Рудий Кіт – це інтернет-магазин зоотоварів, який пропонує широкий асортимент продукції для домашніх тварин, включаючи корм, вітаміни, аксесуари, ветеринарні препарати та іграшки для собак, котів, птахів, гризунів і риби (рис. 1.2). Сайт має зручну навігацію, детальні описи товарів, систему фільтрації за категоріями та брендами, а також можливість замовлення товарів з доставкою по всій Україні. Присутня інформація про акції, знижки та програми лояльності.

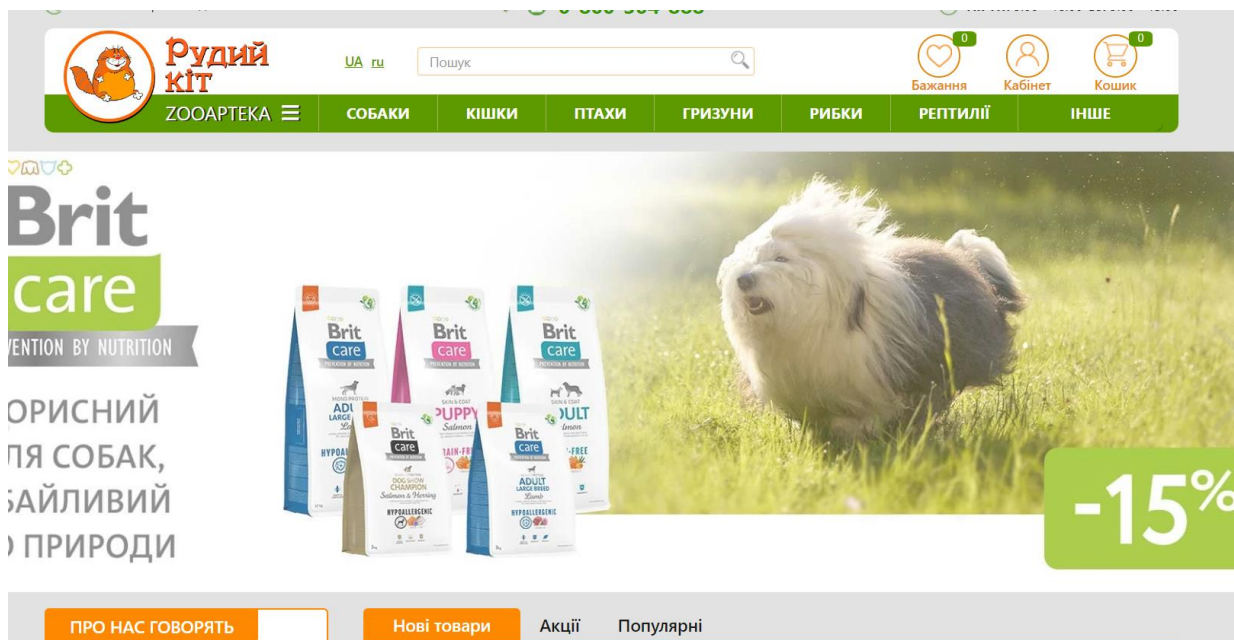


Рис. 1.2 Інтерфейс додатку «Рудий Кіт»

Переваги:

1. **Широкий асортимент:** Пропонує великий вибір зоотоварів для різних видів домашніх тварин.
2. **Зручна навігація:** Інтуїтивно зрозумілий інтерфейс та система фільтрації товарів за категоріями і брендами.
3. **Детальні описи товарів:** Надання вичерпної інформації про продукцію.
4. **Акції та знижки:** Регулярні пропозиції та програми лояльності для клієнтів.
5. **Швидка доставка:** Можливість замовлення товарів з доставкою по всій Україні.
6. **Підтримка клієнтів:** Наявність консультацій та допомоги від спеціалістів

Недоліки:

1. **Швидкість завантаження:** Можливі затримки при завантаженні сторінок, що впливає на користувацький досвід.
2. **Мобільна версія:** Може бути менш зручною у порівнянні з настільною версією.

3. **Інформаційне наповнення:** Не завжди достатньо детальної інформації про товари та їх характеристики.
4. **Відгуки та рейтинги:** Відсутність або обмежена система відгуків і рейтингів від користувачів, що ускладнює вибір продукції.
5. **Підтримка клієнтів:** Можливі труднощі з оперативністю та якістю відповідей на запити клієнтів.

Shinshilka.ua – це інтернет-магазин зоотоварів, що пропонує широкий асортимент кормів, ветеринарних препаратів, аксесуарів для собак, котів, птахів та гризунів (рис. 1.3). Магазин надає знижки на популярні товари, зручну навігацію та фільтрацію продукції. Є можливість консультацій та допомоги при виборі товарів. Сайт забезпечує швидку доставку по всій Україні. Доступні відеоогляди товарів та інформація про акції і спеціальні пропозиції.

Переваги:

1. **Широкий асортимент:** Пропонує великий вибір кормів, ветеринарних препаратів та аксесуарів для собак, котів, птахів та гризунів.
2. **Регулярні акції та знижки:** Часті спеціальні пропозиції та знижки на популярні товари.
3. **Швидка доставка:** Оперативна доставка товарів по всій Україні.
4. **Детальні описи товарів:** Надання вичерпної інформації про кожен товар.
5. **Зручна навігація:** Інтуїтивно зрозумілий інтерфейс та система фільтрації товарів.
6. **Консультації спеціалістів:** Можливість отримання професійних порад та допомоги при виборі товарів.
7. **Відеоогляди товарів:** Додаткова інформація про продукцію через відеоогляди.

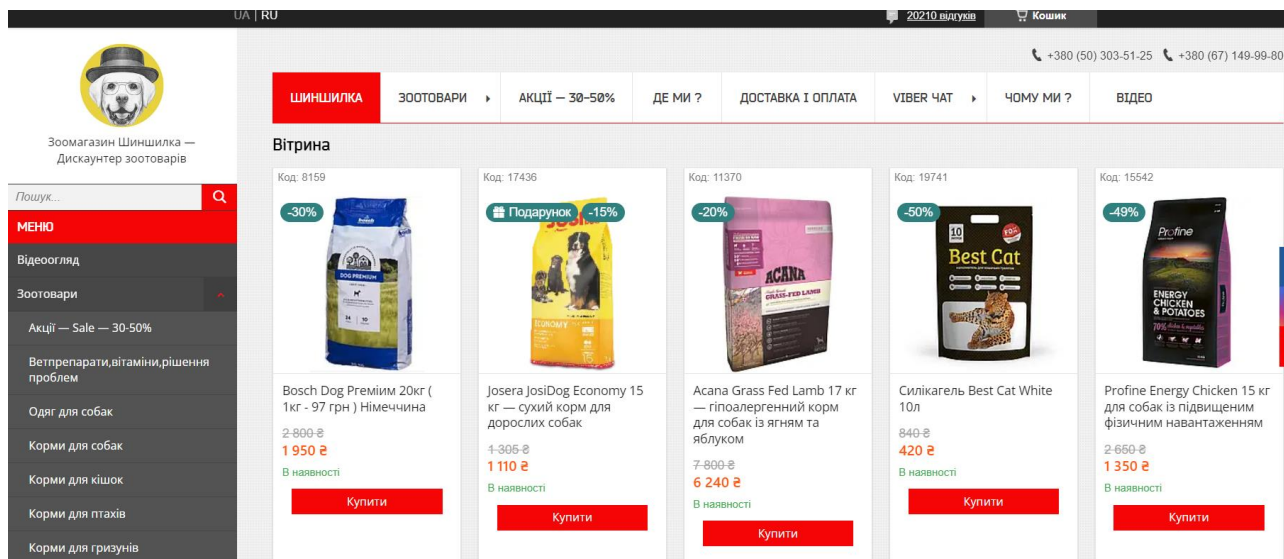


Рис. 1.3 Інтерфейс додатку «Шиншилка»

Недоліки:

1. **Швидкість завантаження:** Можливі затримки при завантаженні сторінок, що може негативно вплинути на користувацький досвід.
2. **Мобільна версія:** Може бути менш зручною у порівнянні з настільною версією, що ускладнює користування сайтом на мобільних пристроях.
3. **Відгуки та рейтинги:** Обмежена система відгуків від користувачів, що ускладнює вибір продукції.
4. **Інформаційне наповнення:** Не завжди достатньо детальної інформації про товари та їх характеристики, що може вплинути на рішення про покупку.
5. **Підтримка клієнтів:** Можливі труднощі з оперативністю та якістю відповідей на запити клієнтів.

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

Таблиця 1.1

Порівняльна таблиця аналогів

Характеристика	33 Корови	Шиншилка	Рудий кіт
Перегляд товарів	Зручний	Зручний	Зручний

Продовження таблиці 1.1

Пошук та фільтрація	Зручний пошук	Погано працює пошук	Є пошук фільтрація
Адаптивність інтерфейсу	Адаптивний	Адаптивний	Адаптивний
Зручність використання	Зручний	Зручний	Зручний
Наявність специфічних товарів	Деякі специфічні відсутні	Надають консультації	Деякі специфічні відсутні
Дизайн	Чудовий	Чудовий	Чудовий

Провівши аналіз готових рішень, стало зрозуміло, що існуючі інтернет-магазини ветеринарних товарів в Україні, такі як 33 корови, Рудий кіт та Shinshilka, мають низку переваг, включаючи широкий асортимент товарів, зручний інтерфейс, програми лояльності, швидку доставку та професійні консультації. Однак, їхні недоліки, такі як високі ціни, обмежена кількість фізичних магазинів і відсутність деяких специфічних товарів, створюють можливості для поліпшення.

Було прийнято рішення створити власний програмний продукт для продажу ветеринарних товарів, щоб запропонувати унікальні функції та інноваційні рішення, забезпечити оптимізацію і швидкість роботи, гнучкість і масштабованість, а також побудувати власний бренд і контроль якості. Це дозволить краще задовольняти потреби клієнтів та забезпечити конкурентоспроможність на ринку.

1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Веб-додаток буде реалізований за клієнт-серверною архітектурою. Серверна частина, що функціонуватиме у вигляді API, буде розроблена з

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						16

використанням сучасного серверного фреймворку NestJS. Для клієнтської частини буде застосовано фреймворк NextJS, який базується на бібліотеці ReactJS. В якості сховища даних використовуватиметься база даних PostgreSQL.1.3.1 Визначення архітектури серверної частини веб-додатку.

Для розробки клієнтської частини вебдодатку було обрано фреймворк **NextJS**. Це сучасний інструмент, побудований на основі ReactJS, який дозволяє створювати високопродуктивні та SEO-оптимізовані вебдодатки завдяки підтримці серверного рендерингу (SSR) та статичної генерації (SSG).

Основні переваги NextJS включають можливість відмальовувати сторінки на сервері завдяки Server-Side Rendering (SSR), що збільшує швидкість завантаження сторінок та покращує їх SEO-оптимізацію. Також NextJS підтримує Static Site Generation (SSG), що дозволяє генерувати статичний контент під час збірки додатку, зменшуючи навантаження на сервер та збільшуючи швидкість завантаження сторінок. Крім цього, підтримка клієнтського відмальовування (Client-Side Rendering, CSR) дозволяє створювати динамічний контент та інтерактивні елементи. Важливою перевагою є вбудована підтримка TypeScript, яка забезпечує більш надійну та зручну розробку завдяки строго типізованому коду. Нарешті, NextJS має широкі можливості розширення завдяки підтримці багатьох різних плагінів та модулів, що забезпечує більшу гнучкість розробки.

Для розробки серверної частини веб-додатку був обраний фреймворк NestJS через його строго модульну архітектуру, яку зручно розширювати та вдосконалювати. Разом з NestJS використовується TypeORM, яка є зручною ORM для взаємодії з базою даних PostgreSQL, маючи функцію синхронізації описаних таблиць з реальною базою. Для системи авторизації використовується бібліотека JWT, вбудована в NestJS, а для взаємодії серверного додатку зі Strapi застосовується HttpModule.

Основні переваги NestJS включають використання TypeScript, що забезпечує надійність та зручність розробки завдяки строго типізованому коду. Модульність NestJS сприяє легкому розширенню та вдосконаленню коду,

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						17

забезпечуючи чітку структуру проекту, що спрощує підтримку. Вбудований механізм впровадження залежностей (Dependency Injection) забезпечує високу розширюваність додатків. HttpModule дозволяє просто взаємодіяти серверному додатку з іншими сервісами, такими як Strapi, та підтримує асинхронні запити та обробку даних. Спрощена система модульного тестування полегшує написання та підтримку тестів, забезпечуючи високу якість коду та надійність застосунку. Вбудований Swagger дозволяє легко взаємодіяти з серверною частиною, з можливістю детального опису кожного запиту.

Для розробки був використаний CMS Strapi, оскільки це зручний інструмент для адміністрування інтернет-магазину. Переваги Strapi включають наявність готової UI адмін-панелі, яка забезпечує зручне адміністрування без додаткових налаштувань. Strapi є open-source продуктом, що дозволяє написання власних контролерів та налаштування функціоналу відповідно до потреб. Крім того, велика кількість плагінів розширює функціонал Strapi, додаючи нові можливості для адміністрування та управління контентом.

Для взаємодії з базою даних використовується PostgreSQL разом з ORM TypeORM. PostgreSQL є потужною об'єктно-реляційною системою керування базами даних, яка забезпечує надійність, високу продуктивність та підтримку складних запитів. Використання PostgreSQL гарантує стабільну роботу додатку навіть при великих обсягах даних та високих навантаженнях. Переваги PostgreSQL включають високу надійність і стабільність, забезпечену роками розвитку та використання у промислових масштабах. Висока продуктивність забезпечується підтримкою складних запитів та оптимізацією для високих навантажень, що дозволяє швидко працювати з великими обсягами даних. Підтримка транзакцій ACID гарантує збереження цілісності даних навіть у випадку збою системи. Розширюваність PostgreSQL дозволяє додавати нові функції і типи даних завдяки модульній архітектурі. Широка підтримка різноманітних програмних мов і середовищ робить інтеграцію легкою та зручною. Вбудовані інструменти для реплікації та масштабування дозволяють

налаштовувати реплікацію даних і масштабувати базу даних відповідно до зростання потреб бізнесу.

TypeORM у свою чергу, є ефективним інструментом для роботи з базою даних, що дозволяє розробникам писати чистий та зрозумілий код для взаємодії з даними. Завдяки можливості синхронізації описаних таблиць з реальною базою, TypeORM забезпечує зручність в управлінні базою даних, мінімізуючи ризики помилок і підвищуючи продуктивність розробки.

TypeORM, у свою чергу, є ефективним інструментом для роботи з базою даних, що дозволяє розробникам писати чистий та зрозумілий код для взаємодії з даними. Завдяки можливості синхронізації описаних таблиць з реальною базою, TypeORM забезпечує зручність в управлінні базою даних, мінімізуючи ризики помилок і підвищуючи продуктивність розробки.

Переваги TypeORM включають сумісність з різними базами даних, такими як PostgreSQL, MySQL, MariaDB, SQLite, Microsoft SQL Server, Oracle та MongoDB. Декларативний синтаксис з використанням декораторів для оголошення моделей і зв'язків між ними робить код чистим і зрозумілим. Вбудований механізм міграцій дозволяє створювати та виконувати міграції бази даних, спрощуючи управління схемою. Розширюваність забезпечується модульною архітектурою, яка дозволяє додавати нові функції і типи даних. Повна підтримка TypeScript забезпечує безпеку типів та інтелектуальне завершення коду. Можливість автоматичної синхронізації схем бази даних з описаними моделями зменшує кількість помилок і прискорює розробку. Крім того, TypeORM дозволяє створювати запити за допомогою зручного та читабельного синтаксису, що полегшує роботу з даними.

Вимоги до апаратного забезпечення

Для забезпечення стабільної та швидкої роботи вебдодатку, необхідно врахувати наступні вимоги до апаратного забезпечення:

1. Віддалений сервер:

- **Процесор:** Xeon 3x2.60 GHz.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						19

- **Оперативна пам'ять:** Рекомендовано 4 ГБ оперативної пам'яті.
- **Накопичувач:** SSD-диск з 40 ГБ вільного простору.
- **Мережа:** Високошвидкісне підключення до інтернету з мінімальною затримкою.

2. Клієнтські пристрої:

- **Процесор:** Сучасний багатоядерний процесор.
- **Оперативна пам'ять:** Мінімум 4 ГБ оперативної пам'яті.
- **Браузер:** Останні версії браузерів, таких як Google Chrome, Mozilla Firefox, Microsoft Edge.
- **Мережа:** Стабільне підключення до інтернету.

Використання цих інструментальних засобів та відповідного апаратного забезпечення забезпечить високу продуктивність, надійність та зручність використання вебдодатку для продажу ветеринарних товарів.

Висновки до першого розділу.

Після проведення ретельного аналізу предметної області, було визначено основні вимоги до платформи для продажу ветеринарних товарів. Основна мета платформи – забезпечити користувачам зручний спосіб пошуку та придбання необхідних товарів.

Під час дослідження аналогічних продуктів були розглянуті різні платформи з різними підходами до роботи, функціонал яких включав пошук, фільтрацію та сортування товарів, а також підтримку користувачів. На основі цього аналізу були сформовані ключові вимоги до власної платформи.

З огляду на ці вимоги та критерії, було визначено набір основних технологій для розробки вебдодатку. NestJS був обраний як серверний фреймворк завдяки своїй строгій модульній архітектурі, що спрощує розширення та вдосконалення коду. TypeORM використовується для ефективної взаємодії з базою даних PostgreSQL, яка забезпечує надійність і високу продуктивність. Крім того, Strapi був обраний для адміністрування

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						20

контенту завдяки своїй зручній UI адмін-панелі, можливості створення власних контролерів та великій кількості плагінів, що розширюють функціональність.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						21

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Визначення варіантів використання та структури системи

Для забезпечення успішної реалізації дипломного проекту з розробки вебсайту для продажу ветеринарних товарів, мої основні обов'язки зосереджені на налаштуванні та інтеграції адміністративної панелі з використанням CMS Strapi, а також на зв'язуванні даних між Strapi та серверним додатком, розробленим на основі NestJS.

1. Налаштування CMS Strapi:

- Встановлення та конфігурація Strapi для створення зручної адміністративної панелі.
- Створення моделей даних для товарів, категорій, користувачів та замовлень.
- Налаштування ролей та дозволів для адміністраторів і користувачів системи.

2. Інтеграція Strapi з NestJS:

- Розробка API для отримання даних з Strapi та передачі їх до NestJS.
- Забезпечення безперебійної взаємодії між Strapi та серверним додатком, включаючи обробку запитів та відповіді.
- Використання HttpModule в NestJS для асинхронної взаємодії з Strapi.

3. Забезпечення безпеки та авторизації:

- Реалізація системи авторизації за допомогою JWT, інтегрованої в NestJS, для безпечного керування сесіями користувачів.
- Налаштування ролей і дозволів у Strapi для забезпечення доступу до чутливих даних лише авторизованим користувачам.

4. Тестування та оптимізація:

- Проведення функціонального тестування інтеграції між Strapi та NestJS.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						22

- Оптимізація продуктивності взаємодії між адміністративною панеллю та серверним додатком для забезпечення швидкої роботи додатку навіть при високих навантаженнях.

Ця робота дозволить створити надійну і ефективну систему для продажу ветеринарних товарів, яка буде легко керуватися та масштабуватися, забезпечуючи зручність і безпеку для кінцевих користувачів.

Вимоги користувачів

Внутрішній користувач – адміністратор

1. Управління каталогом ветеринарних товарів.
2. Підтвердження та модерація нових товарів і оголошень.
3. Спілкування з клієнтами та постачальниками.
4. Підтвердження та модерація нових замовлень

Внутрішній користувач – клієнт

1. Реєстрація та авторизація.
2. Перегляд каталогу товарів.
3. Пошук товарів за категоріями та фільтрами.
4. Додавання товарів до кошика.
5. Оформлення та відстеження замовлень.
6. Перегляд історії покупок.
7. Редагування особистої інформації

Внутрішній користувач – гість

1. Перегляд каталогу товарів.
2. Пошук товарів за категоріями та фільтрами.
3. Перегляд детальної інформації про товари.
4. Реєстрація для оформлення замовлень.
5. Зв'язок зі службою підтримки
6. Спілкування з консультантом для отримання допомоги.

Функціональні вимоги

- ## 1. Авторизація адміністратора:
- Реєстрація адмінського акаунту.

					ИПЗ.КР.Б-121-24-ПЗ	Арк.
						23

- Авторизація за допомогою логіна та пароля.

2. Управління замовленнями та їх статусом:

- Можливість перегляду замовлень від користувачів, можливість змінювати статус.
- Фільтрація та сортування замовлень по статусу, користувачам.

3. Обробка фото товарів та субкатегорій:

- Інтеграція з AWS.
- Можливість завантаження та видалення фото.

Нефункціональні вимоги

1. Сприйняття:

- Час, потрібний для освоєння вебсайту – від 3 до 5 хвилин, для досвідчених користувачів менше 2 хвилин.
- Час відповіді вебсайту на дії користувача не повинен перевищувати 2 секунд.
- Інтерфейс вебсайту повинен бути інтуїтивно зрозумілим та зручним для користувача, та не вимагати додаткової підготовки.

2. Надійність:

- Вебсайт має бути доступний цілодобово.
- Система повинна забезпечувати збереження даних навіть у випадку збою.

3. Продуктивність:

- Сайт повинен витримувати одночасне використання великою кількістю користувачів.
- Система повинна забезпечувати швидке завантаження сторінок та обробку запитів.

4. Безпека:

- Захист персональних даних користувачів.
- Використання SSL для шифрування даних.
- Регулярне оновлення системи для захисту від вразливостей.

Після аналізу вимог користувачів і для наглядної демонстрації взаємодії між елементами додатку було визначено варіанти використання додатку для продажу ветеринарних товарів. На рис. 2.1 наведено діаграму використання програмної системи.



Рис. 2.1 Діаграма варіантів використання

Опис діаграми варіантів використання наведено в таблицях 2.1 – 2.10.

Таблиця 2.1

Опис прецеденту «Оформлення замовлень»

Назва варіанту використання	Оформлення замовлень
Основні актори	Авторизований користувач
Короткий опис	Авторизований користувач може оформити замовлення, вказуючи адресу доставки та обираючи спосіб оплати.

Таблиця 2.2

Опис прецеденту «Перегляд інформації про магазини»

Назва варіанту використання	Перегляд інформації про магазини
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть переглядати інформацію про фізичні магазини, включаючи адреси, години роботи та контактні дані.

Таблиця 2.3

Опис прецеденту «Управління каталогом ветеринарних товарів»

Назва варіанту використання	Управління каталогом ветеринарних товарів
Основні актори	Адміністратор
Короткий опис	Адміністратор може додавати, редагувати та видаляти товари в каталозі, оновлювати їх описи, ціни та наявність.

Таблиця 2.4

Опис прецеденту «Підтвердження та модерація замовлень»

Назва варіанту використання	Підтвердження та модерація замовлень
Основні актори	Адміністратор
Короткий опис	Адміністратор перевіряє та підтверджує нові замовлення, контролює процес їх виконання та вирішує можливі проблеми з замовленнями.

Таблиця 2.5

Опис прецеденту «Додавання нових магазинів на мапу»

Назва варіанту використання	Додавання нових магазинів на мапи
Основні актори	Адміністратор

Продовження таблиці 2.5

Короткий опис	Адміністратор додає нові магазини на карту, оновлює інформацію про існуючі магазини та видаляє закриті точки.
---------------	---

Таблиця 2.6

Опис прецеденту «Перегляд повідомлень від клієнтів»

Назва варіанту використання	Перегляд повідомлень від клієнтів
Основні актори	Адміністратор
Короткий опис	Адміністратор переглядає та обробляє повідомлення від клієнтів, надає відповіді на запити та вирішує виниклі проблеми.

Таблиця 2.7

Опис прецеденту «Логін для адміністратора»

Назва варіанту використання	Логін для адміністратора
Основні актори	Адміністратор
Короткий опис	Адміністратор може потрапити в систему для редагування товарів, управління замовленнями та даними сайту

Таблиця 2.8

Опис прецеденту «Створення нового товару»

Назва варіанту використання	Створення нового товару
Основні актори	Адміністратор
Короткий опис	Адміністратор може створити новий товар

Таблиця 2.9

Опис прецеденту «Створення нової категорії»

Назва варіанту використання	Створення нової категорії
Основні актори	Адміністратор
Короткий опис	Адміністратор може створити нову категорію товарів

Таблиця 2.10

Опис прецеденту «Створення нової підкатегорії»

Назва варіанту використання	Створення нової підкатегорії
Основні актори	Адміністратор
Короткий опис	Адміністратор може створити нову підкатегорію

Аналіз функціональних вимог дозволив визначити такі сутності, які забезпечать програмну реалізацію додатку а саме Order, User, Basket, Authorization, Product, SubCategory, Category, ShopAddresses, Payment . Було створено діаграму класів (рис. 2.2.) та описано модулі що відповідають за функціонал пов'язаний з продуктами, фільтрацією та сортуванням товарів, категоріями, користувачами та корзиною користувача.

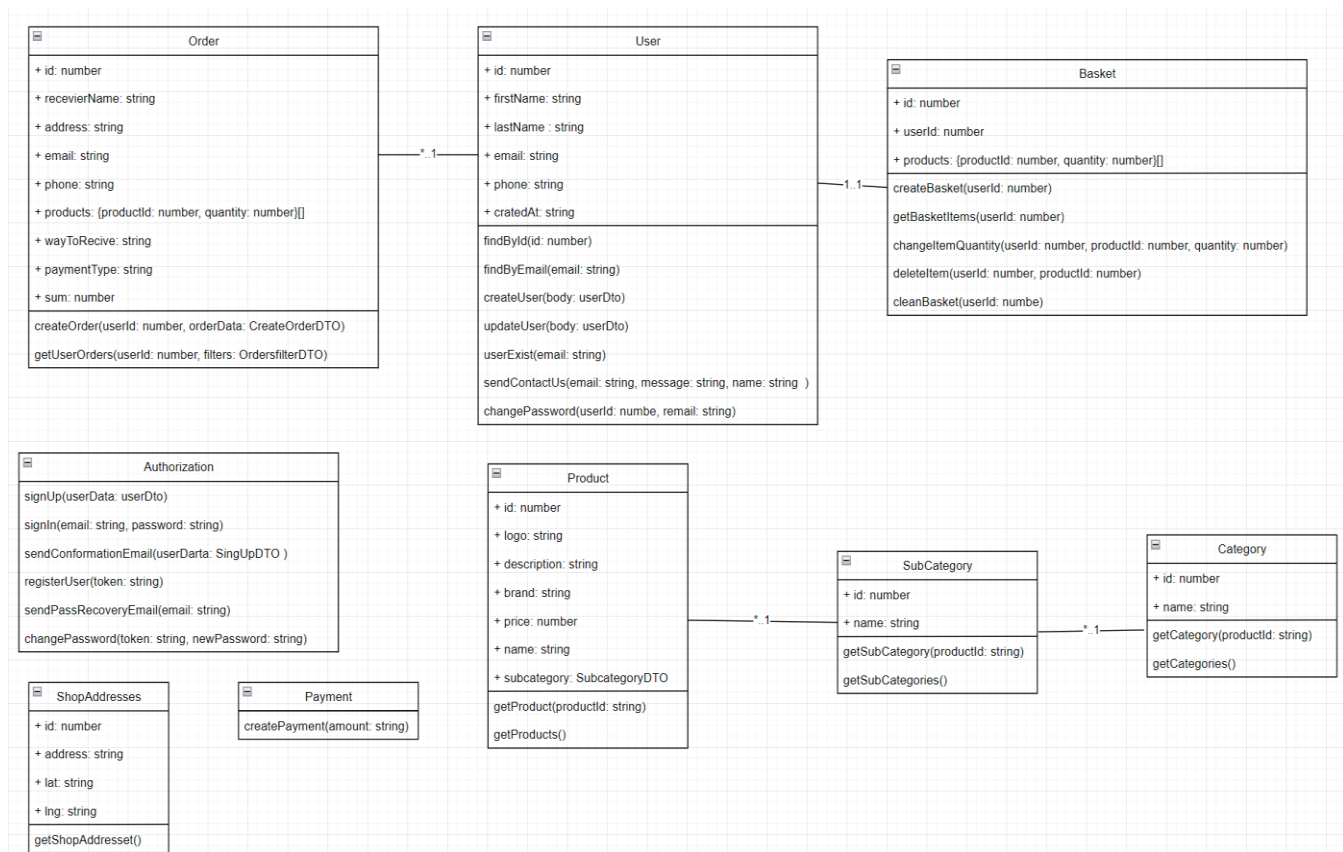


Рис. 2.2 Діаграма класів

Розглянемо кожен клас, його поля та методи.

User – клас який відповідає за користувача, має наступні методи:

- `createUser()` – метод що відповідає за створення і збереження користувача в базі даних.
- `updateUser()` – метод що відповідає за оновлення даних існуючого користувача.
- `sendContactUs()` – метод що відповідає за оновлення даних існуючого користувача.
- `userExist()` – метод що перевіряє чи є такий користувач в базі даних.
- `changePassword()` – метод що відповідає за зміну пароля в існуючого користувача.
- `findById()` – метод що відповідає за пошук користувача за його ID.
- `findByEmail()` – метод що відповідає за пошук користувача за його електронною поштою.

Basket – клас який відповідає за роботу з корзиною користувача, має наступні методи:

- createBasket() – метод що відповідає за створення нової корзини та збереження її в базі даних на етапі реєстрації нового користувача.
- getBasketItems() – метод що відповідає за отримання товарів які були додані до корзини.
- changeItemQuantity() – метод що відповідає за додавання нових товарів до корзини і зміну кількості товарів що вже були додані в корзину.
- deleteItem() – метод що відповідає за видалення раніше доданого товару з корзини.
- cleanBasket () – метод що відповідає за видалення всіх товарів з корзини що були додані раніше.

Product – клас який відповідає за роботу з товарами, має наступні методи:

- getProduct() – метод що відповідає за отримання детальної інформації про обраний товар.
- getProducts() – метод що відповідає за отримання товарів, відповідно до фільтрів які обрав користувач.
- Аналогічно адресам, створення видалення редагування товарів відбувається в панелі адміністратора створеній з використанням StrapiCMS, тому потреби в окремих методах немає.

SubCategory – клас який відповідає за роботу з підкатегоріями:

- getSubCategory() – метод що відповідає за отримання детальної інформації про одну підкатегорію.
- getSubCategories() – метод що відповідає за отримання всіх підкатегорій.
- Аналогічно товарам, створення видалення редагування підкатегорій відбувається в адмін панелі, тому потреби в окремих методах немає.

2.2 Розробка бази даних системи

Обидві системи мають свої специфічні підходи до управління даними, і розуміння цих підходів є ключовим для ефективної інтеграції.

Strapi забезпечує гнучку та зручну адмін-панель для управління контентом, що дозволяє створювати та налаштовувати таблиці безпосередньо через інтерфейс користувача. У той час, як NestJS відповідає за реалізацію бізнес-логіки та обробку запитів, використовуючи свої власні моделі та структури даних.

Далі ми розглянемо основні таблиці, які використовуються в обох системах, та проаналізуємо, як вони взаємодіють між собою для забезпечення цілісності даних і узгодженості інформації між Strapi та серверним додатком на NestJS. Це включає в себе опис схем баз даних, зв'язків між таблицями, а також механізми синхронізації та обробки даних.

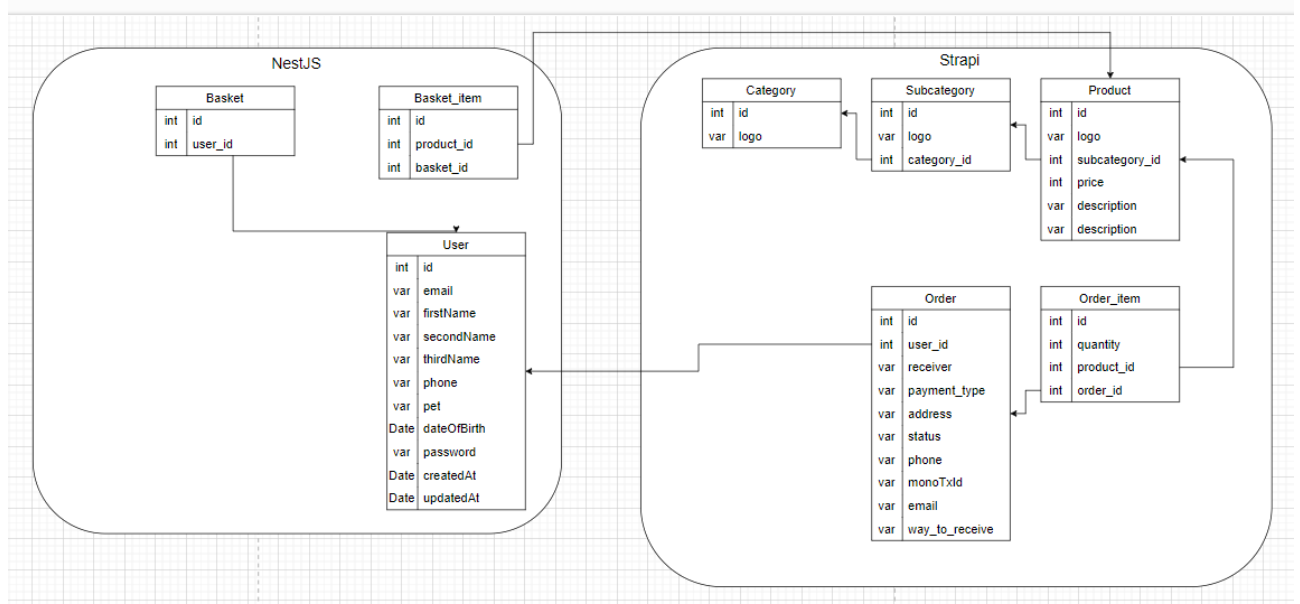


Рис 2.3 Діаграма, що описує взаємодію таблиць в частинах додатку

Опис схеми взаємодії між NestJS та Strapi

На діаграмі представлена структура таблиць, що використовуються в системі, яка поєднує CMS Strapi для управління контентом та бекенд на NestJS для обробки запитів і взаємодії з клієнтським додатком на Next.js.

Структура таблиць у Strapi

Category:

id (int): Унікальний ідентифікатор категорії.

logo (var): Логотип категорії.

Категорія має зв'язок один-до-багатьох з підкатегоріями (Subcategory).

Subcategory:

id (int): Унікальний ідентифікатор підкатегорії.

logo (var): Логотип підкатегорії.

category_id (int): Зовнішній ключ до таблиці категорій (Category).

Підкатегорія має зв'язок один-до-багатьох з продуктами (Product).

Product:

id (int): Унікальний ідентифікатор продукту.

logo (var): Логотип продукту.

subcategory_id (int): Зовнішній ключ до таблиці підкатегорій (Subcategory).

price (int): Ціна продукту.

description (var): Опис продукту.

Продукти пов'язані з підкатегоріями.

Order:

id (int): Унікальний ідентифікатор замовлення.

user_id (int): Ідентифікатор користувача, який зробив замовлення.

receiver (var): Отримувач замовлення.

payment_type (var): Тип оплати.

address (var): Адреса доставки.

status (var): Статус замовлення.

phone (var): Телефон отримувача.

monoTxId (var): Ідентифікатор транзакції Monobank.

email (var): Електронна адреса отримувача.

way_to_receive (var): Спосіб отримання замовлення.

Замовлення має зв'язок один-до-багатьох з позиціями замовлення (Order_item).

Order_item:

id (int): Унікальний ідентифікатор позиції замовлення.

quantity (int): Кількість товарів.

product_id (int): Зовнішній ключ до таблиці продуктів (Product).

order_id (int): Зовнішній ключ до таблиці замовлень (Order).

Позиції замовлення пов'язані з замовленнями та продуктами.

Структура таблиць у NestJS

User:

id (int): Унікальний ідентифікатор користувача.

email (var): Електронна адреса користувача.

firstName (var): Ім'я користувача.

secondName (var): Прізвище користувача.

thirdName (var): По батькові користувача.

phone (var): Номер телефону користувача.

pet (var): Домашня тварина користувача.

dateOfBirth (Date): Дата народження користувача.

password (var): Пароль користувача.

createdAt (Date): Дата створення запису.

updatedAt (Date): Дата останнього оновлення запису.

Basket:

id (int): Унікальний ідентифікатор кошика.

user_id (int): Зовнішній ключ до таблиці користувачів (User).

Кошик пов'язаний з користувачами.

Basket_item:

id (int): Унікальний ідентифікатор позиції в кошику.

product_id (int): Ідентифікатор продукту.

basket_id (int): Ідентифікатор кошика.

Позиції в кошику пов'язані з кошиками та продуктами.

Взаємодія між NestJS та Strapi

У цій архітектурі клієнтський додаток взаємодіє виключно з серверним додатком на NestJS. Коли клієнтська частина надсилає запит на отримання або зміну даних, серверна частина обробляє цей запит та робить запит до Strapi для отримання або зміни необхідної інформації. Оскільки Strapi має власний специфічний формат даних на виході, серверна частина виконує аналіз отриманих даних, форматуючи їх у зручний для клієнтського додатку вигляд.

Цей підхід дозволяє зосередитися на бізнес-логіці у серверному додатку, використовуючи Strapi для зручного управління контентом. Це розділення відповідальностей забезпечує гнучкість, масштабованість та зручність в управлінні даними.

2.3 Проектування та реалізація алгоритмів роботи додатку

Використання CMS Strapi як адміністративної панелі разом з серверним додатком на NestJS в проекті забезпечує кілька важливих переваг, роблячи таку інтеграцію ефективною та гнучкою. Strapi надає готовий інтерфейс адміністратора, що дозволяє зручно керувати контентом без необхідності розробляти власну адмін-панель з нуля, що значно знижує час та витрати на розробку. Крім того, Strapi є open-source продуктом, що дозволяє розробникам модифікувати його під специфічні потреби проекту, легко створювати кастомні контролери та інтегрувати додаткові плагіни для розширення функціональності. Strapi підтримує роботу з різними базами даних, такими як MongoDB, MySQL, PostgreSQL, надаючи гнучкість у виборі сховища даних відповідно до вимог проекту.

Використання NestJS у проекті має кілька важливих переваг. Модульна архітектура NestJS забезпечує зручність у підтримці та розширенні додатка, оскільки кожен модуль відповідає за конкретну частину функціоналу, що сприяє кращій організації коду та його повторному використанню. Повна підтримка TypeScript в NestJS забезпечує статичну типізацію та інструменти

для більш безпечного та передбачуваного коду. Крім того, NestJS легко інтегрується з іншими популярними бібліотеками та фреймворками, такими як TypeORM для роботи з базами даних, що дозволяє створювати високопродуктивні додатки.

Переваги інтеграції Strapi та NestJS

Розділення відповідальностей:

Використання Strapi як адміністративної панелі дозволяє зосередитися на розробці основного бізнес-логіки в NestJS. Це розділення відповідальностей забезпечує більш чітке розмежування ролей і функцій в системі.

Швидкий старт:

Завдяки готовій адмін-панелі Strapi, можна швидко почати керувати контентом і даними без додаткових витрат часу на розробку. Це особливо важливо на початкових етапах проекту, коли час має критичне значення.

Безпека і масштабованість:

NestJS забезпечує високий рівень безпеки та масштабованості, що дозволяє створювати надійні та стабільні додатки. Використання Strapi додає ще один рівень безпеки, оскільки він має вбудовані механізми для управління правами доступу та аутентифікації користувачів.

У висновку можна сказати, що інтеграція CMS Strapi як адмін-панелі з серверним додатком на NestJS створює потужну та гнучку архітектуру для сучасних веб-додатків. Це рішення дозволяє оптимізувати процеси розробки, забезпечити високу продуктивність та гнучкість системи, а також швидко реагувати на змінні потреби проекту. Така комбінація інструментів є чудовим вибором для розробки масштабованих, надійних та ефективних веб-додатків.

2.4 Реалізація функціоналу додатку

Для реалізації функціоналу завантаження файлів у Strapi з використанням AWS S3 необхідно виконати кілька кроків, що включають налаштування

самого хмарного сховища на AWS, створення облікових даних для підключення та конфігурацію Strapi для роботи з цим сервісом.

Amazon S3 (Simple Storage Service) – це об'єктне сховище, яке забезпечує високу масштабованість, доступність та безпеку для зберігання будь-яких обсягів даних. AWS S3 широко використовується для зберігання та резервного копіювання даних, медіафайлів, і як сховище для статичних файлів веб-додатків.

Переваги використання AWS S3 для зберігання зображень

AWS S3 забезпечує безмежну масштабованість, дозволяючи зберігати будь-які обсяги зображень без потреби в управлінні фізичною інфраструктурою. Це особливо корисно для проектів, які швидко ростуть або мають змінні обсяги даних. S3 розроблено з акцентом на високу доступність та надійність, дані автоматично реплікуються в межах декількох центрів обробки даних в обраному регіоні, що гарантує їх збереження. Крім того, AWS S3 надає потужні інструменти для захисту даних, включаючи шифрування у спокої та під час передачі, політики контролю доступу та інтеграцію з AWS IAM для управління правами доступу користувачів.

Економічна вигода S3 полягає у моделі оплати за використання, що робить його ефективним рішенням для зберігання великих обсягів зображень. Можливість автоматичного переміщення даних між різними класами зберігання допомагає оптимізувати витрати. Інтеграція S3 з іншими сервісами AWS та популярними системами керування контентом, такими як Strapi, забезпечує безшовну взаємодію і обробку даних, роблячи його гнучким і сумісним рішенням для зберігання.

Конфігурація Strapi для роботи з AWS S3

Для налаштування Strapi для роботи з AWS S3 необхідно змінити конфігураційні файли проекту. Створення правильного конфігураційного файлу дозволить Strapi завантажувати файли безпосередньо у ваш S3 хмарне сховище.

Налаштування параметрів з'єднання:

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						36

```

module.exports = ({ env }) => ({
  upload: {
    config: {
      provider: 'aws-s3',
      providerOptions: {
        accessKeyId: env('AWS_ACCESS_KEY_ID'),
        secretAccessKey: env('AWS_ACCESS_SECRET'),
        region: env('AWS_REGION'),
        params: {
          Bucket: env('AWS_BUCKET'),
          ACL: env('AWS_ACL', 'public-read'),
        },
      },
      actionOptions: {
        upload: {},
        uploadStream: {},
        delete: {},
      },
    },
  },
});

```

Безпечне спілкування між NestJS та Strapi за допомогою приватного API ключа

Створення та налаштування приватного API ключа у Strapi

Для забезпечення безпечного обміну даними між NestJS і Strapi, використовується приватний API ключ. Приватний API ключ є унікальним токеном, що генерується в Strapi та використовується для аутентифікації запитів від серверного додатку NestJS. Це гарантує, що тільки авторизовані запити можуть отримати доступ до даних у Strapi.

У Strapi цей ключ створюється через адмін-панель. Після створення ключа задаються права доступу, які дозволяють виконувати різні операції з даними. Збереження згенерованого ключа в безпечному місці є критично важливим для підтримки безпеки.

Використання приватного API ключа у NestJS

Збереження приватного API ключа у конфігураційних файлах або як змінної середовища забезпечує його безпеку та зручність використання. Під час надсилання запитів з NestJS до Strapi, ключ включається в заголовки запитів (headers), що дозволяє Strapi перевірити автентичність запиту.

Переваги використання приватного API ключа

Використання приватного API ключа забезпечує підвищену безпеку, оскільки тільки авторизовані запити можуть отримати доступ до даних Strapi.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						37

Крім того, контроль доступу дозволяє налаштувати різні рівні прав для різних ключів, що забезпечує гнучкість у керуванні доступом до даних. Інтеграція та використання API ключа є відносно простими та не вимагають значних змін в існуючій архітектурі додатку.

Реалізація безпечного обміну даними

Конфігурація API ключа в NestJS передбачає додавання його до файлу конфігурації. Ключ включається у заголовки кожного запиту до Strapi для аутентифікації. Для забезпечення додаткового рівня безпеки, дані під час передачі між сервісами шифруються за допомогою HTTPS.

Регулярний моніторинг і аудит журналів доступу у Strapi дозволяє відстежувати використання API ключів та виявляти можливі спроби несанкціонованого доступу. Це забезпечує додатковий рівень захисту та гарантує надійність системи.

Таким чином, використання приватного API ключа для забезпечення безпечного обміну даними між NestJS і Strapi є надійним і ефективним рішенням, що дозволяє захистити дані, забезпечити контроль доступу та гарантувати надійність системи.

Авторизація користувача у системі реалізована на стороні серверного додатку з використанням JSON Web Token (JWT). JWT є стандартом для створення токенів, які використовуються для передачі інформації про користувача між клієнтом та сервером без необхідності зберігання цієї інформації на сервері:

```
async sendConfirmationEmail(userData: SignUpDTO) {
  const token = this.encodeToken(userData, 3);
  const appConfig = this.configService.get('app');
  const url = `${appConfig.url}/${appConfig.confirmSignUpRoute}?token=${token}`;

  const mailResponse = await this.mailServiceProvider.sendConfirmationEmail(
    userData.email,
    url,
  );

  if (!mailResponse) {
    this.logger.error("Mail Service wasn't able to sent mail");
    throw new InternalServerErrorException();
  }

  return true;
}
```

```

async registerUser(token: string) {
  try {
    const userData = this.decodeToken<SignUpDTO>(token);
    return this.userService.createUser(userData);
  } catch (e) {
    this.logger.error(e);
    throw new ConflictException();
  }
}

```

Реєстрація користувача:

1. Користувач вводить свої дані для реєстрації (ім'я, електронна пошта, пароль, ім'я).
2. Створення JWT: Після успішної реєстрації, серверний додаток генерує JWT, який містить зашифровану інформацію про користувача.
3. Відправка токена: JWT відправляється користувачеві та зберігається на клієнтській.

Верифікація електронної пошти через Mailjet

Для верифікації електронної пошти користувачів використовується сервіс Mailjet. Після реєстрації користувача, на його електронну адресу надсилається лист з посиланням для підтвердження.

Після реєстрації, серверний додаток створює посилання для верифікації, яке містить зашифровані дані користувача. Це посилання надсилається користувачеві через сервіс Mailjet. Користувач переходить за посиланням у листі, що автоматично надсилає запит на підтвердження на серверний додаток. Серверний додаток отримує запит і розшифровує дані, щоб підтвердити електронну пошту користувача. Після успішної верифікації, статус користувача оновлюється, і реєстрація вважається завершеною.

Переваги використання JWT та Mailjet

- Безпека: JWT забезпечує надійне зберігання даних користувача та їх передачу без необхідності зберігання інформації на сервері. Верифікація через Mailjet гарантує, що електронна адреса користувача є дійсною.

- **Ефективність:** Використання токенів та зовнішніх сервісів для верифікації значно спрощує процес авторизації та підвищує його ефективність.
- **Зручність для користувачів:** Процес реєстрації та верифікації є простим та зручним, що покращує користувацький досвід.

Ця схема забезпечує надійну та безпечну авторизацію користувачів у системі, гарантуючи, що тільки верифіковані користувачі можуть отримати доступ до сервісу.

Скидання пароля через Mailjet

Функція скидання пароля також реалізована за допомогою сервісу Mailjet.

Якщо користувач забуває свій пароль, він може запросити скидання пароля.

```
async sendPasswordRecoveryEmail(email: string) {
  const token = this.encodeToken(email);

  const appConfig = this.configService.get('app');
  const url =
`${appConfig.url}/${appConfig.changePasswordRoute}?token=${token}`;

  const mailResponse = await this.mailServiceProvider.sendConfirmationEmail(
    email,
    url,
  );

  if (!mailResponse) {
    this.logger.error("Mail Service wasn't able to sent mail");
    throw new InternalServerErrorException();
  }

  return true;
}

async changePassword(token: string, newPassword: string) {
  try {
    const email = this.decodeToken<string>(token);
    const user = await this.userService.findByEmail(email);
    await this.userService.changePassword(user.id, newPassword);
    return true;
  } catch (e) {
    this.logger.error(e);
    throw new ConflictException();
  }
}
```

1. **Запит на скидання пароля:** Користувач вводить свою електронну адресу на формі запиту скидання пароля.

2. **Створення посилання для скидання:** Серверний додаток генерує посилання для скидання пароля, яке містить зашифровані дані користувача.
3. **Відправка листа:** Це посилання надсилається користувачеві через Mailjet.
4. **Скидання пароля:** Користувач переходить за посиланням, яке веде на сторінку для введення нового пароля. Після введення нового пароля, Серверний додаток оновлює інформацію про користувача.

Висновки до 2 розділу

У цьому розділі було детально розглянуто різні аспекти забезпечення безпеки та авторизації користувачів у системі, яка поєднує CMS Strapi та серверний додаток на NestJS.

Безпечне спілкування між NestJS та Strapi: Використання приватного API ключа забезпечує високий рівень безпеки при обміні даними між цими сервісами. Приватний API ключ дозволяє тільки авторизованим запитам отримувати доступ до даних Strapi, що гарантує захист інформації від несанкціонованого доступу.

Авторизація за допомогою JWT: JWT забезпечує надійну авторизацію користувачів без необхідності зберігання їх даних на сервері. Це підвищує безпеку та зручність роботи з системою.

Верифікація електронної пошти через Mailjet: Верифікація електронної пошти користувачів через Mailjet забезпечує додатковий рівень безпеки, підтверджуючи дійсність електронних адрес. Процес включає відправку листа з посиланням для підтвердження, яке користувач використовує для завершення реєстрації.

Скидання пароля: Функція скидання пароля реалізована за допомогою Mailjet, що дозволяє користувачам безпечно змінювати свої паролі у разі потреби. Це підвищує зручність користування системою та забезпечує додатковий рівень безпеки.

Ці заходи разом створюють надійну та безпечну систему, яка забезпечує захист даних користувачів, ефективну авторизацію та зручність у використанні, гарантує безпеку при передачі даних між компонентами системи та надає користувачам простий механізм для верифікації електронної пошти і скидання паролів.

РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ.

3.1 Порядок встановлення та налаштування параметрів додатку

Розглянемо процес розгортання додатку, що складається з кількох сервісів, використовуючи Docker Compose. Docker Compose дозволяє визначити та запустити багатоконтейнерні додатки з усіма їхніми залежностями за допомогою єдиного конфігураційного файлу.

Структура Docker Compose файлу

Файл docker-compose.yml версії "3.5" містить визначення кількох сервісів, необхідних для роботи додатку: backend, backend-db, strapi, cms-db, nextjs-app і reverse-proxy. Кожен з цих сервісів має свої налаштування та залежності.

Опис сервісів

1. Backend:

- Контейнер: backend
- Автоматичний перезапуск: unless-stopped
- Контекст збірки: ./backend
- Порти: "3001:3001"
- Змінні середовища: включають конфігурації для бази даних, Mailjet, Strapi тощо.
- Налаштування Traefik для керування маршрутизацією та сертифікацією HTTPS.

2. Backend-db:

- Контейнер: backend-db
- Образ: postgres:14-alpine
- Змінні середовища: параметри для створення бази даних, користувача та пароля.

3. Strapi:

- Контейнер: cms

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						43

- Автоматичний перезапуск: unless-stopped
- Контекст збірки: ./cms
- Порти: "1337:1337"
- Змінні середовища: конфігурації для підключення до бази даних, безпека та інтеграція з AWS S3.
- Налаштування Traefik для адміністрування через HTTPS.

4. Cms-db:

- Контейнер: cms-db
- Образ: postgres:14-alpine
- Змінні середовища: параметри для створення бази даних, користувача та пароля.

5. Nextjs-app:

- Контейнер: frontend
- Контекст збірки: ./vet-store
- Порти: "3000:3000"
- Томі: маппінг локальних файлів і директорій
- Змінні середовища: параметри для інтеграції з Nova Poshta API, MapTiller, серверною частиною та OpenAI.
- Налаштування Traefik для керування маршрутизацією та сертифікацією HTTPS.

6. Reverse-proxy:

- Образ: traefik:v2.4
- Контейнер: traefik
- Команди для налаштування: керування API, налаштування entrypoints та провайдерів, сертифікація через ACME.
- Порти: "443:443", "80:80", "8080:8080"
- Томі: маппінг локальних файлів для зберігання сертифікатів і доступу до Docker сокету.

Опис змінних середовища

- **NODE_ENV**: Вказує режим роботи додатку (production, development).
- **PORT**: Порт, на якому працює веб-сервер.
- **DB_HOST**: Адреса хосту бази даних PostgreSQL.
- **DB_PORT**: Порт, на якому база даних PostgreSQL прослуховує з'єднання.
- **DB_NAME**: Ім'я бази даних, яку використовуватиме додаток.
- **DB_USERNAME**: Ім'я користувача для підключення до бази даних.
- **DB_PASSWORD**: Пароль для підключення до бази даних.
- **DB_LOGGING**: Вказує, чи буде логування запитів до бази даних увімкнено (true або false).
- **MAILER_USER**: Ім'я користувача для сервісу надсилання електронної пошти.
- **MAILER_PASSWORD**: Пароль для сервісу надсилання електронної пошти.
- **MAILJET_API_KEY**: Ключ для інтеграції з Mailjet, сервісом для надсилання електронних листів.
- **MAILJET_API_SECRET**: Секретний ключ для доступу до API Mailjet.
- **STRAPI_URL**: URL-адреса для підключення до Strapi.
- **STRAPI_ACCESS_KEY**: Ключ доступу для взаємодії з Strapi API.
- **MONOBANK_API_TOKEN**: Токен доступу для інтеграції з MONOBANK API.
- **FRONTEND_SUCCESS_LINK**: URL-адреса, на яку буде перенаправлено користувача після успішної оплати.
- **WEBHOOK_URL**: Адреса для отримання вебхуків для оновлення статусу замовлень.
- **NEXT_PUBLIC_NP_API_KEY**: Ключ для доступу до Nova Poshta API.

- **NEXT_PUBLIC_NP_API_URL:** URL для підключення до Nova Poshta API.
- **NEXT_PUBLIC_MAPTILLER_API_KEY:** Ключ для доступу до MapTiller API.
- **NEXT_PUBLIC_API_URL:** URL серверної частини.
- **NEXT_PUBLIC_CHAT_API_KEY:** Ключ для доступу до OpenAI.

Переваги використання Traefik

Для розгортання додатку я використовував Traefik разом з Docker та Docker Compose. Traefik інтегрується з Docker, автоматично виявляє нові сервіси та налаштовує для них маршрути, що значно спрощує управління мережею. Підтримка автоматичного отримання та оновлення SSL сертифікатів через Let's Encrypt забезпечує безпечний доступ до сервісів, а функціональність балансувальника навантаження дозволяє рівномірно розподіляти трафік між сервісами.

Процес розгортання починався з підготовки середовища на віддаленому сервері Ubuntu, де були встановлені Docker і Docker Compose. Я створив необхідні директорії для контекстів збірки та конфігураційних файлів. Потім конфігурував Docker Compose, створивши файл `docker-compose.yml` з визначеннями сервісів і перевіривши правильність налаштувань і змінних середовища. Після цього запустив сервіси за допомогою команди `docker-compose up -d`, що дозволило запустити їх у фоновому режимі. Логи і статус сервісів перевіряв за допомогою команд `docker-compose logs` і `docker-compose ps`.

Наступним кроком було налаштування DNS та сертифікації. Я налаштував DNS записи для піддоменів (наприклад, `api.codlo.pp.ua`, `admin.codlo.pp.ua`, `app.codlo.pp.ua`) та перевіряв оновлення SSL сертифікатів через Traefik. Останнім етапом було тестування та моніторинг: перевіряв доступність всіх сервісів через відповідні URL та здійснював моніторинг стану контейнерів та логів для виявлення можливих проблем.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						46

Розгортання додатку за допомогою Docker Compose забезпечило гнучкість та масштабованість системи. Використання контейнерів дозволило ізолювати середовища, спрощуючи керування залежностями та конфігураціями. Traefik забезпечив керування маршрутизацією та сертифікацією HTTPS, підвищуючи безпеку та надійність додатку. Завдяки детальним налаштуванням і використанню змінних середовища, додаток легко адаптувався до різних умов розгортання та вимог проекту. Центральне управління та доступність додатку через домен codlo.pp.ua забезпечили його стабільну роботу та легке масштабування.

3.2 Структура інтерфейсу веб-додатку та порядок роботи

Для використання додатку користувачам потрібний пристій зі встановленим інтернет браузером. Інтерфейс додатку адаптивний і гарно працює на рідних пристроях.

Для авторизації у ролі адміністратора нам потрібно перейти по субдомену admin. Після переходу ми побачимо екран, який пропонує нам авторизуватись як адміністратор.

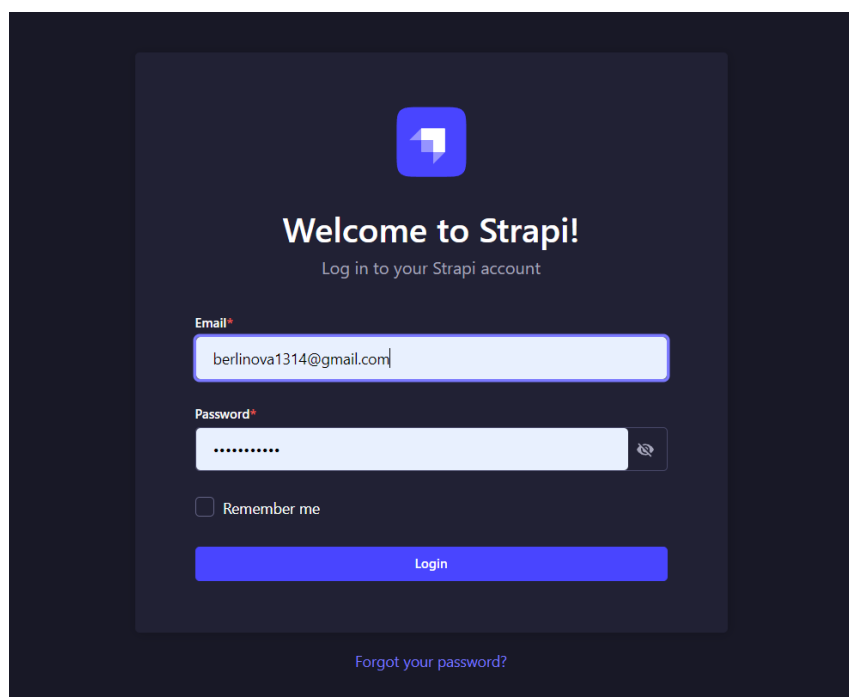


Рисунок 3.1 Логін для адміністратора

По замовчуванню в адмін-панель потрібно буде додати всі дані про позиції та товари, але вони вже додані, тому розглянемо як виглядає екран з продуктами, які вже частково створені.

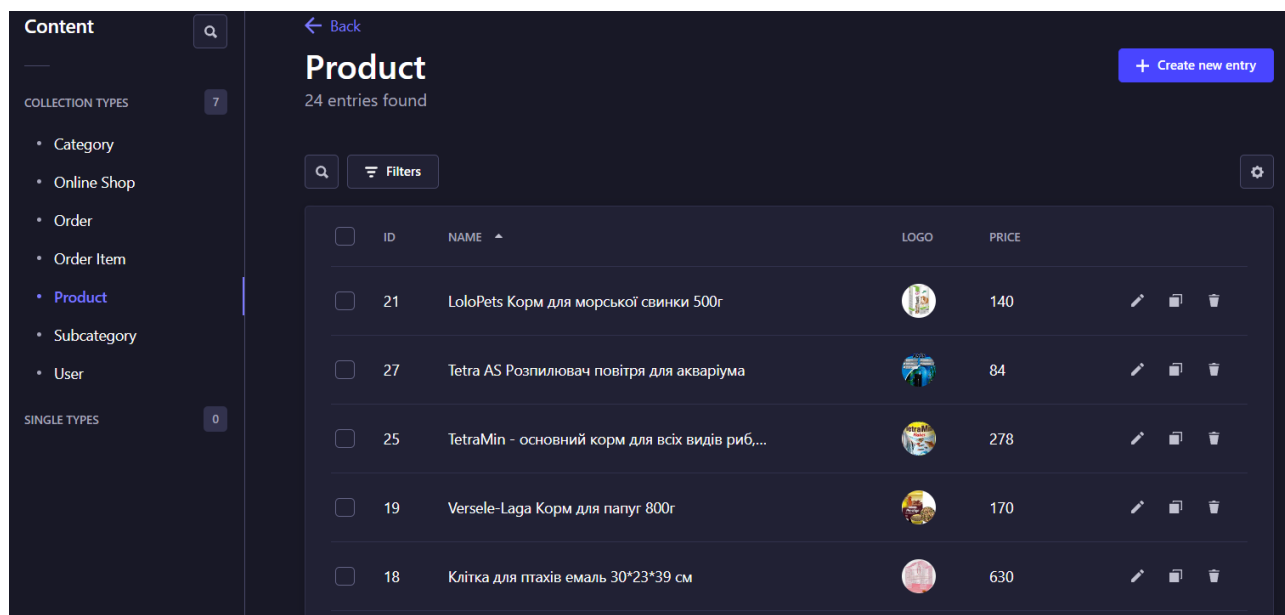


Рисунок 3.2 Список продуктів зі сторони адміна

CMS Strapi надає зручні засоби для створення, редагування та видалення записів з колекцій. Розглянемо як виглядає екран створення нового продукту.

Create an entry
API ID: product

name*

logo*

Click to add an asset or drag and drop one in this area

price*

description*

brand

subcategory

INFORMATION

Created now

By -

Last update now

By -

[Edit the model](#)

[Configure the view](#)

Рисунок 3.3 Створення нового продукту

Крім цього CMS Strapi надає зручні засоби роботи з файлами, по замовчуванню файли зберігаються локально, але проєкт інтегрований з AWS S3, тому всі файли, які завантажуються потрапляють туди.

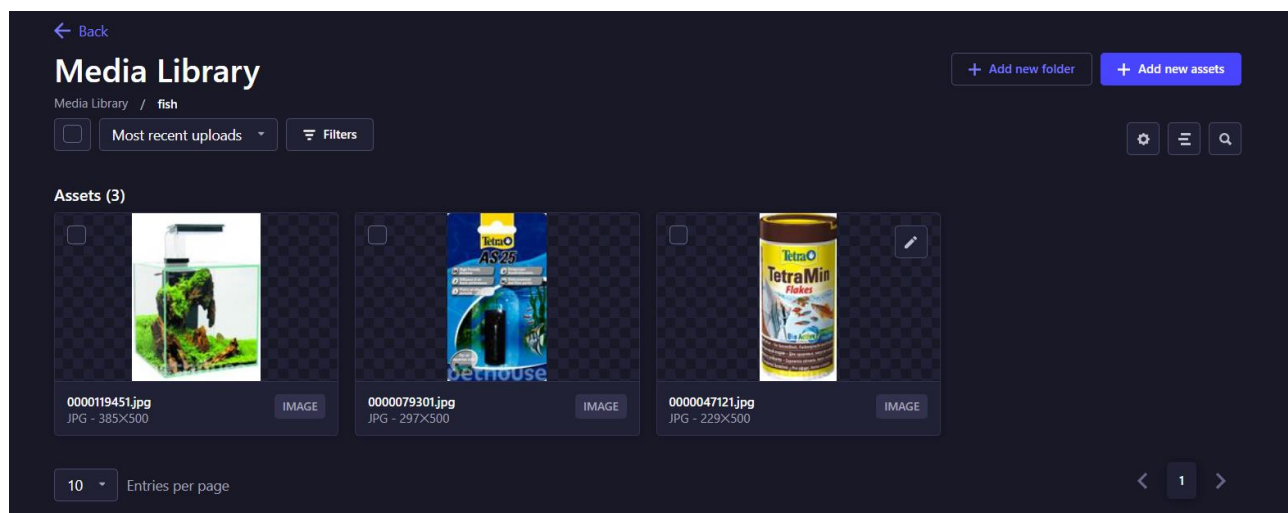


Рисунок 3.4 Медіа галерея

Також CMS Strapi надає можливість використовувати приватний API ключ для отримання, оновлення та видалення даних через Strapi API, маємо згенерований ключ, яким користується серверний додаток для отримання та збереження даних за допомогою Strapi.

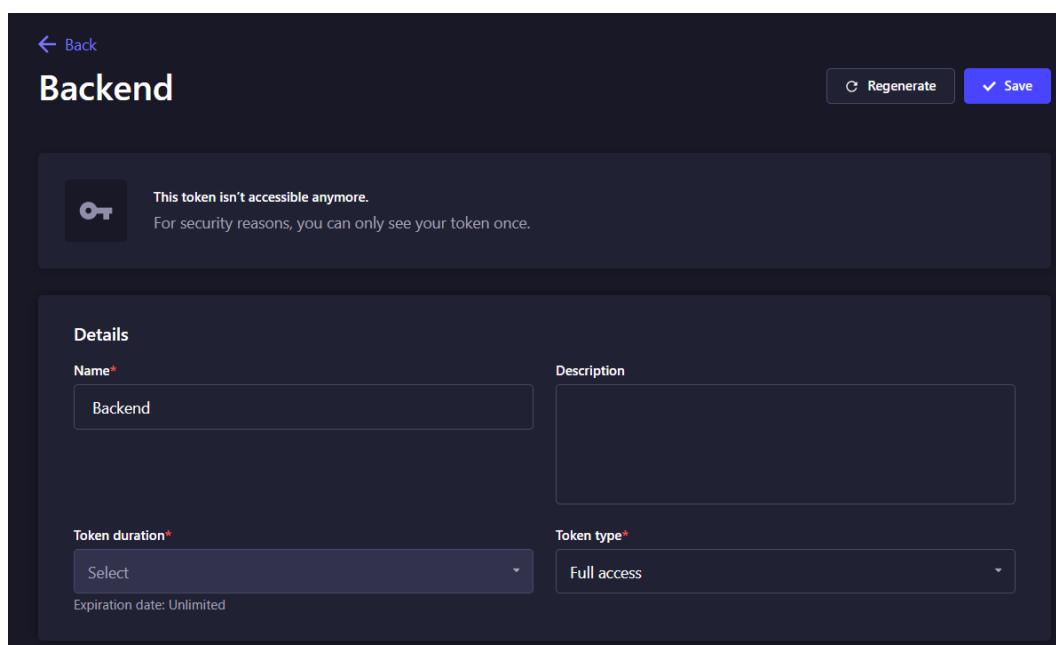


Рисунок 3.5 Приватный ключ

Для користувача є екран реєстрації, де він може ввести свої дані для реєстрації в сервісі.

КаталогМагазиниДоставкаПро нас

Реєстрація

Вже зареєстровані? [Увійти?](#)

Як вас звати?

Електронна пошта

Номер телефону

Пароль

Повторіть пароль

* Пароль повинен мати мінімум 8 символів, літери верхнього і нижнього регістрів і хоча б один спецсимвол

Рисунок 3.6 Реєстрація користувача

Після введення всіх даних користувач отримає повідомлення про успішну відправку листа верифікації на пошту.

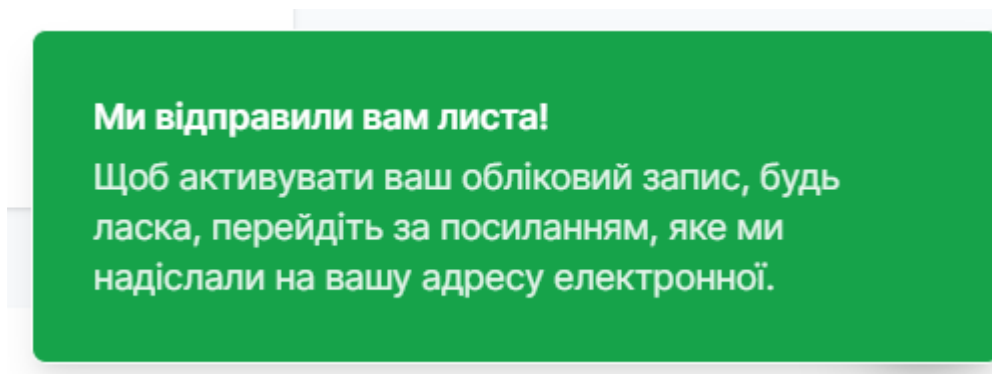


Рисунок 3.7 Успішно відправлений лист



noreply@codlo.pp.ua via bnc3.mailjet.com
to me ▾

2:05 AM (1 minute ago)



Translate to English



Активация вашего аккаунта

Шановний клієнт,

Дякуємо за реєстрацію на Codlo! Ви ще один крок від завершення реєстрації та початку користування всіма нашими послугами.

Будь ласка, активуйте ваш аккаунт, натиснувши на кнопку нижче:

Активация

Активация гарантирует, что ваша электронная адреса введена правильно, та дозволяє нам надати вам кращий досвід користування.

Дякуємо, що обрали нас!

З повагою,

Команда ветеринарного магазину

Рисунок 3.8 Лист з підтвердженням

В листі користувач бачить текст про те, що реєстрація майже закінчена, залишається лише перейти за посиланням для завершення активації.



Каталог

Магазини

Доставка

Про нас



Вхід

Вітаємо, ви успішно зареєструвались, гарних покупок.

PetPals

Залишилися питання? Зв'яжіться з нами



Політика конфіденційності



Рисунок 3.9 Екран успішної реєстрації

ІПЗ.КР.Б-121-24-ПЗ

Арк.

51

3.3 Тестування роботи веб-додатку

1. Перевірка моделей даних:

- **Створення нових записів:** Заходжу в адмін-панель Strapi і додаю нові записи для різних моделей (наприклад, користувачів, товарів).
- **Редагування існуючих записів:** Редакую існуючі записи, щоб переконатися, що всі поля оновлюються правильно.
- **Видалення записів:** Видаляю записи і перевіряю, чи зникають вони з бази даних.

2. Перевірка API:

- **GET-запити:** Відправляю GET-запити до кінцевих точок API Strapi, щоб отримати дані і перевірити їх коректність.
- **POST-запити:** Використовую інструменти, такі як Postman, щоб відправити POST-запити для створення нових записів і перевірити відповіді.
- **PUT-запити:** Відправляю PUT-запити для оновлення записів і перевіряю, чи змінюються дані в базі даних.
- **DELETE-запити:** Перевіряю, чи видаляються записи після відправки DELETE-запитів.

1. Реєстрація нового користувача:

- **Форма реєстрації:** Заповнюю форму реєстрації на сайті, вводячи коректні дані (ім'я, електронну адресу, пароль).
- **Перевірка електронної пошти:** Після реєстрації перевіряю свою електронну пошту на отримання листа з підтвердженням.
- **Підтвердження електронної пошти:** Клацаю на посилання в листі для підтвердження електронної пошти. Перевіряю, чи отримую повідомлення про успішне підтвердження.

2. Вхід до системи:

- **Форма логіну:** Вводжу дані для входу (електронну адресу та пароль) і перевіряю, чи можу успішно увійти до системи.

- **Помилкові дані:** Пробую увійти з неправильними даними (невірний пароль або електронна адреса) і перевіряю, чи отримую відповідні повідомлення про помилку.

3. Скидання пароля:

- **Запит на скидання пароля:** Використовую форму для скидання пароля, вводячи електронну адресу.
- **Отримання листа з інструкціями:** Перевіряю електронну пошту на отримання листа з інструкціями для скидання пароля.
- **Скидання пароля:** Виконую інструкції з листа, щоб скинути пароль, і перевіряю, чи можу увійти з новим паролем.

1. Перевірка навігації:

- **Меню та посилання:** Переконаюсь, що всі посилання на сайті працюють і ведуть до відповідних сторінок.

2. Валідація форм:

- **Правильні дані:** Вводжу правильні дані у всі форми (реєстрація, логін, скидання пароля) і перевіряю, чи приймаються вони системою.
- **Неправильні дані:** Вводжу неправильні дані (некоректний формат електронної пошти, занадто короткий пароль тощо) і перевіряю, чи система видає відповідні повідомлення про помилку.

3. Взаємодія з інтерфейсом:

- **Кнопки та елементи управління:** Клацаю на всі кнопки та елементи управління, щоб переконатися, що вони працюють правильно.
- **Модальні вікна та сповіщення:** Перевіряю, чи з'являються всі необхідні модальні вікна та сповіщення після відповідних дій.

Такий підхід до тестування вручну дозволяє переконатися, що всі основні функції додатку працюють коректно, ідентифікувати можливі проблеми та вчасно їх усунути.

Висновки до третього розділу

У цьому розділі було детально розглянуто функціонал Strapi, процес тестування та розгортання додатку за допомогою Docker.

Розглянуто всі аспекти налаштування та роботи з CMS Strapi, включаючи створення, редагування та видалення записів через адмін-панель, а також взаємодію з API для керування даними.

Також проведено тестування основних функцій додатку, таких як реєстрація користувачів, скидання пароля та функціональна перевірка користувацького інтерфейсу. Це включало ручне тестування форм, валідацію даних і перевірку роботи електронних листів.

Розглянуто покроковий процес розгортання додатку за допомогою Docker, включаючи налаштування сервісів через Docker Compose, конфігурацію змінних середовища, налаштування Traefik для керування маршрутизацією та сертифікацією HTTPS, а також перевірку і моніторинг запущених контейнерів.

Ці заходи забезпечили високу стабільність, продуктивність і безпеку додатку, що дозволяє ефективно використовувати його для управління контентом та обслуговування користувачів.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було спроектовано та розроблено веб-додаток для продажу ветеринарних товарів. Проведено всебічний аналіз проблематики, методів і засобів вирішення задачі, а також здійснено порівняння аналогічних програмних продуктів. На основі цього аналізу визначено ключові вимоги до веб-додатку та обґрунтовано вибір інструментальних засобів і апаратного забезпечення для реалізації проекту.

Під час проектування веб-додатку було визначено сценарії використання та структурні елементи системи. Основну увагу приділено розробці бази даних і реалізації основних алгоритмів функціонування додатку. Особливий акцент зроблено на забезпеченні безпеки та захисту даних, що є критично важливим для електронної комерції. Створений функціонал націлений на забезпечення зручності використання як для адміністраторів, так і для кінцевих користувачів, що сприяє покращенню користувацького досвіду.

У розділі, присвяченому інтерфейсу та роботі з веб-додатком, представлено детальну інструкцію з встановлення та налаштування додатку. Описано структуру інтерфейсу та порядок його використання, що допомагає користувачам ефективно орієнтуватися в системі та використовувати її функціональні можливості. Проведено тестування веб-додатку, яке підтвердило його стабільність та високу продуктивність на мобільних пристроях, забезпечуючи якісний сервіс для кінцевих користувачів.

Загалом, розроблений веб-додаток для продажу ветеринарних товарів повністю відповідає встановленим вимогам, забезпечуючи необхідний функціонал. Додаток демонструє високу адаптивність та продуктивність, що дозволяє ефективно вирішувати задачі, пов'язані з онлайн-продажем ветеринарних товарів. Завдяки цьому, він надає користувачам зручний інструмент для здійснення покупок, що підвищує ефективність і якість обслуговування.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						55

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 33 корови [Електронний ресурс] - Режим доступу до ресурсу:
<https://33korovy.in.ua/uk/>
2. Шиншилка [Електронний ресурс] - Режим доступу до ресурсу:
<https://shinshilka.ua>
3. Рудий кіт [Електронний ресурс] - Режим доступу до ресурсу:
<https://www.zooapteka.kiev.ua/ua/>
4. NestJS [Електронний ресурс] - Режим доступу до ресурсу:
<https://nestjs.com>
5. NextJS [Електронний ресурс] - Режим доступу до ресурсу:
<https://nextjs.org>
6. NestJS first steps [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.nestjs.com/first-steps>
7. NestJS modules [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.nestjs.com/modules>
8. NestJS middleware [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.nestjs.com/middleware>
9. TypeScript [Електронний ресурс] - Режим доступу до ресурсу:
<https://www.typescriptlang.org/>
10. Docker compose [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.docker.com/compose/>
11. Traefik [Електронний ресурс] - Режим доступу до ресурсу:
<https://doc.traefik.io/traefik/>
12. Strapi [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.strapi.io>
13. Class validator [Електронний ресурс] - Режим доступу до ресурсу:
<https://github.com/typestack/class-validator>

14. Strapi with S3_[Електронний ресурс] - Режим доступу до ресурсу:
<https://strapi.io/blog/how-to-set-up-amazon-s3-upload-provider-plugin-for-our-strapi-app>
15. Monobank API [Електронний ресурс] - Режим доступу до ресурсу:
<https://api.monobank.ua/docs/index.html>
16. Strapi API key_[Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.strapi.io/dev-docs/configurations/api-tokens>
17. Typeorm [Електронний ресурс] - Режим доступу до ресурсу:
<https://orkhan.gitbook.io/typeorm/docs/find-options>
18. Ubuntu docs_[Електронний ресурс] - Режим доступу до ресурсу:
<https://ubuntu.com>
19. Mailjet Docs [Електронний ресурс] - Режим доступу до ресурсу:
<https://dev.mailjet.com>
20. OpenAI Integration API Documentation [Електронний ресурс] - Режим доступу до ресурсу: <https://platform.openai.com/docs/api-reference>
21. Нова пошта API - [Електронний ресурс] - Режим доступу до ресурсу:
<https://developers.novaposhta.ua/documentation>
22. PostgreSQL - [Електронний ресурс] - Режим доступу до ресурсу:
<https://www.postgresql.org/>

ДОДАТКИ

Лістинг файлу docker-compose.yml

```
version: "3.5"

services:
  backend:
    container_name: backend
    restart: unless-stopped
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "3001:3001"
    environment:
      - NODE_ENV=production
      - PORT=3001
      - DB_HOST=backend-db
      - DB_PORT=5432
      - DB_NAME=db_name
      - DB_USERNAME=db_user
      - DB_PASSWORD=db_password
      - DB_LOGGING=false
      - MAILER_USER=2fb66ac256c4a5
      - MAILER_PASSWORD=86332b29bed996
      - MAILJET_API_KEY=2a055e1774494bdc0d2f5dfd89f80506
      - MAILJET_API_SECRET=2ef53c4502cb352972181c930e1c2b19
      - STRAPI_URL=http://cms:1337/api
      - STRAPI_ACCESS_KEY=f92076cb5072e39fcb9dc4380e6c7e5af99b684c0947c89b7156150fe4e2dda15aea435c011582b29055b3a34b5ea50cb7724c8e5df03d5296fa64a090101404a872e740275ca5e71cec7f621574f6fbb49cb0bcd8f67d23d76870c268b16a9b2cefca542eb3e011ed949be6f6b3c517c070e96e4dc50493ee003341a9436e8
      - MONOBANK_API_TOKEN=uhch_B_zzq52_DnpKx77oiEHa3Pk8Fs-FrapmpB9wtYc
      - FRONTEND_SUCCESS_LINK=https://successurl.com
      - WEBHOOK_URL=http://webhook.com
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.backend-api.rule=Host(`api.codlo.pp.ua`)"
      - "traefik.http.routers.backend-api.entrypoints=websecure"
      - "traefik.http.routers.backend-api.tls.certresolver=myresolver"
      - "traefik.http.middlewares.backend-https-redirect.redirectscheme.scheme=https"
      - "traefik.http.middlewares.backend-secure-redirect.redirectscheme.permanent=true"
      - "traefik.http.routers.backend-redirs.rule=hostregex(`{host:.+}`)"
      - "traefik.http.routers.backend-redirs.entrypoints=web"
      - "traefik.http.routers.backend-redirs.middlewares=backend-https-redirect"

  backend-db:
    container_name: backend-db
```

					ИПЗ.КР.Б-121-24-ПЗ	Арк.
						59

```
image: postgres:14-alpine
environment:
  - POSTGRES_DB=db_name
  - POSTGRES_USER=db_user
  - POSTGRES_PASSWORD=db_password
```

```
strapi:
  container_name: cms
  restart: unless-stopped
  build:
    context: ./cms
    dockerfile: Dockerfile
  ports:
    - "1337:1337"
  environment:
    - HOST=0.0.0.0
    - PORT=1337
    -
```

```
APP_KEYS=gw7ay/yV0beGBVRd6BUrAw==,DH0XHLsJd820503/CxJUFg==,V3DPn1CTxuKH1wbqqiVvCw==,Eqc0xBwcZFHvgRwSUNEmtA==
```

```
  - API_TOKEN_SALT=AjUiux2ayWglaR2gem5ucQ==
  - ADMIN_JWT_SECRET=j5atn0KA7HAjTAG00j04Tg==
  - TRANSFER_TOKEN_SALT=NMzY2GF1kvvKLhNiu5u31A==
  - DATABASE_CLIENT=postgres
  - DATABASE_HOST=cms-db
  - DATABASE_PORT=5432
  - DATABASE_NAME=db_name
  - DATABASE_USERNAME=db_user
  - DATABASE_PASSWORD=db_password
  - DATABASE_SSL=false
  - JWT_SECRET=Im9ALmMbJfhIDrt9VeAnew==
  - AWS_ACCESS_KEY_ID=AKIAU6GDVPAP6JVBOLNH
  - AWS_ACCESS_SECRET=f250+WHKdadcGMr3uufEMUag9ygBG8v6E0z7nzq
  - AWS_REGION=eu-north-1
  - AWS_BUCKET=online-shop-bucket-diplom
```

```
labels:
  - "traefik.enable=true"
  - "traefik.http.routers.strapi-admin.rule=Host(`admin.codlo.pp.ua`)"
  - "traefik.http.routers.strapi-admin.entrypoints=websecure"
  - "traefik.http.routers.strapi-admin.tls.certresolver=myresolver"
  - "traefik.http.middlewares.strapi-https-
```

```
redirect.redirectscheme.scheme=https"
```

```
  - "traefik.http.middlewares.strapi-secure-
```

```
redirect.redirectscheme.permanent=true"
```

```
  - "traefik.http.routers.strapi-redirs.rule=hostregex(`{host:.+}`)"
  - "traefik.http.routers.strapi-redirs.entrypoints=web"
  - "traefik.http.routers.strapi-redirs.middlewares=strapi-https-redirect"
```

```
cms-db:
  container_name: cms-db
  image: postgres:14-alpine
```


```

environment:
  - POSTGRES_DB=db_name
  - POSTGRES_USER=db_user
  - POSTGRES_PASSWORD=db_password

nextjs-app:
  container_name: frontend
  build:
    context: ./vet-store
    dockerfile: Dockerfile
  ports:
    - "3000:3000"
  volumes:
    - ./usr/src/app
    - /usr/src/app/node_modules
  environment:
    - NEXT_PUBLIC_NP_API_KEY=3b6585558625da6ceb2369d832d7e3d5
    - NEXT_PUBLIC_NP_API_URL=https://api.novaposhta.ua/v2.0/json/
    - NEXT_PUBLIC_MAPTILLER_API_KEY=7x6eHL3F1paySox3bXeH
    - NEXT_PUBLIC_API_URL=https://api.codlo.pp.ua
    - NEXT_PUBLIC_CHAT_API_KEY=sk-proj-
XNFZJLQU6QK7wCvKl4UQT3BlbkFJDcIlXd8Dvhdf7b7fTysy
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.client-server.rule=Host(`app.codlo.pp.ua`)"
    - "traefik.http.routers.client-server.entrypoints=websecure"
    - "traefik.http.routers.client-server.tls.certresolver=myresolver"
    - "traefik.http.middlewares.redirect-to-https.redirectscheme.scheme=https"
    - "traefik.http.middlewares.secure-redirect.redirectscheme.permanent=true"
    - "traefik.http.routers.redirs.rule=hostregexp(`{host:.+}`)"
    - "traefik.http.routers.redirs.entrypoints=web"
    - "traefik.http.routers.redirs.middlewares=redirect-to-https"

reverse-proxy:
  image: traefik:v2.4
  container_name: "traefik"
  command:
    - "--api.insecure=true"
    - "--api.dashboard=true"
    - "--providers.docker=true"
    - "--entryPoints.web.address=:80"
    - "--entryPoints.websecure.address=:443"
    - "--entrypoints.web.http.redirects.entryPoint.to=websecure"
    - "--entrypoints.web.http.redirects.entryPoint.scheme=https"
    - "--entrypoints.web.http.redirects.entrypoint.permanent=true"
    - "--providers.docker.exposedbydefault=false"
    - "--certificatesresolvers.myresolver.acme.httpchallenge=true"
    - "--certificatesresolvers.myresolver.acme.httpchallenge.entrypoint=web"
    - "--certificatesresolvers.myresolver.acme.email=hello@paularah.com"
    - "--certificatesresolvers.myresolver.acme.storage=/letsencrypt/acme.json"

```


```

ports:
  - "443:443"
  - "80:80"
  - "8080:8080"
volumes:
  - "./letsencrypt:/letsencrypt"
  - "/var/run/docker.sock:/var/run/docker.sock:ro"

```

volumes:

build:

Лістинг файлу mailjet-mallet.ts

```

import { Injectable, Logger } from '@nestjs/common';
import { MailService } from '../common/interfaces/mailService';
import { MailjetService } from '@app/mailjet';

@Injectable()
export class MailjetMailerService implements MailService {
  // private readonly logger = new Logger(MailjetMailerService.name);
  senderEmail = 'noreply@codlo.pp.ua';
  constructor(private mailjetService: MailjetService) {}

  public async sendPasswordRestoreEmail(
    email: string,
    url: string,
  ): Promise<boolean> {
    try {
      return !!this.mailjetService
        .sendEmail({
          Messages: [
            {
              From: {
                Email: this.senderEmail,
                Name: '',
              },
              To: [
                {
                  Email: email,
                  Name: 'Client',
                },
              ],
              TemplateID: 1,
              TemplateLanguage: true,
              Subject: '',
              Variables: {
                url,
              },
            },
          ],
        })
        .then((res) => {
          return res;
        })
        .catch((e) => {
          Logger.error(e);
        });
    } catch (error) {
      Logger.error(error);
      return false;
    }
  }
}

```

```

public async sendConfirmationEmail(
    email: string,
    url: string,
): Promise<boolean> {
    try {
        return !!this.mailjetService
            .sendEmail({
                Messages: [
                    {
                        From: {
                            Email: this.senderEmail,
                            Name: '',
                        },
                        To: [
                            {
                                Email: email,
                                Name: 'Client',
                            },
                        ],
                        TemplateID: 4818734,
                        TemplateLanguage: true,
                        Subject: '',
                        Variables: {
                            url,
                        },
                    },
                ],
            })
            .then((res) => {
                return res;
            })
            .catch((e) => {
                Logger.error(e);
            });
    } catch (error) {
        Logger.error(error);
        return false;
    }
}

public async sendContactUsForm(
    email: string = 'ipz202_bao@student.ztu.edu.ua',
    message: string,
    name: string,
): Promise<boolean> {
    try {
        return !!this.mailjetService
            .sendEmail({
                Messages: [
                    {
                        From: {
                            Email: this.senderEmail,
                            Name: 'No reply',
                        },
                        To: [
                            {
                                Email: 'ipz202_bao@student.ztu.edu.ua',
                                Name: 'Administration',
                            },
                        ],
                        TemplateID: 5740928,
                        TemplateLanguage: true,
                        Subject: "Нове повідомлення з форми зворотного зв'язку",
                        Variables: {

```

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						63

```

        message,
        email,
        name,
    },
    },
],
})
.then((res) => {
    return res;
})
.catch((e) => {
    Logger.error(e);
});
} catch (error) {
    Logger.error(error);
    return false;
}
}
}
}

```

Лістинг файлу authorization.service.ts

```

import {
    Injectable,
    UnauthorizedException,
    Logger,
    ConflictException,
    InternalServerErrorException,
    NotFoundException,
} from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { UserService } from '../user/user.service';
import { UserEntity } from '../user/user.entity';
import { SignUpDTO } from './authorization.dto';
import { ConfigService } from '@nestjs/config';
import { MailService } from '../common/interfaces/mailService';

@Injectable()
export class AuthorizationService {
    private readonly logger = new Logger(AuthorizationService.name);
    constructor(
        private userService: UserService,
        private jwtService: JwtService,
        private configService: ConfigService,
        private readonly mailServiceProvider: MailService,
    ) {}

    signUp(userData: Partial<UserEntity>) {
        return this.userService.createUser(userData);
    }

    async signIn(email: string, password: string) {
        const user = await this.userService.findByEmail(email);
        if (!user) {
            throw new NotFoundException();
        }
        if (!user.verifyPassword(password)) {
            throw new UnauthorizedException();
        }
        const payload = { userId: user.id };
        return {
            access_token: await this.jwtService.signAsync(payload),
        };
    }
}

```

```

async sendConfirmationEmail(userData: SignUpDTO) {
    const token = this.encodeToken(userData, 3);
    const appConfig = this.configService.get('app');
    const url =
`${appConfig.url}/${appConfig.confirmSignUpRoute}?token=${token}`;

    const mailResponse = await this.mailServiceProvider.sendConfirmationEmail(
        userData.email,
        url,
    );

    if (!mailResponse) {
        this.logger.error("Mail Service wasn't able to send mail");
        throw new InternalServerErrorException();
    }

    return true;
}

async registerUser(token: string) {
    try {
        const userData = this.decodeToken<SignUpDTO>(token);
        return this.userService.createUser(userData);
    } catch (e) {
        this.logger.error(e);
        throw new ConflictException();
    }
}

async sendPasswordRecoveryEmail(email: string) {
    const token = this.encodeToken(email);

    const appConfig = this.configService.get('app');
    const url =
`${appConfig.url}/${appConfig.changePasswordRoute}?token=${token}`;

    const mailResponse = await this.mailServiceProvider.sendConfirmationEmail(
        email,
        url,
    );

    if (!mailResponse) {
        this.logger.error("Mail Service wasn't able to send mail");
        throw new InternalServerErrorException();
    }

    return true;
}

async changePassword(token: string, newPassword: string) {
    try {
        const email = this.decodeToken<string>(token);
        const user = await this.userService.findByEmail(email);
        await this.userService.changePassword(user.id, newPassword);
        return true;
    } catch (e) {
        this.logger.error(e);
        throw new ConflictException();
    }
}

encodeToken<T>(data: T, expiresIn = 1 /* days */) {
    return this.jwtService.sign({
        data,

```

					ИПЗ.КР.Б-121-24-ПЗ	Арк.
						65

```

        iat: Math.floor(Date.now() / 1000),
        exp: Math.floor(Date.now() / 1000) + 60 * 60 * 24 * expiresIn,
    });
}

decodeToken<T>(token: string) {
    return this.jwtService.verify(token).data as T;
}
}

```

Лістинг файлу authorization.controller.ts

```

import {
    Body,
    Controller,
    Get,
    Post,
    Query,
    ConflictException,
    NotFoundException,
} from '@nestjs/common';
import { ApiOkResponse, ApiTags } from '@nestjs/swagger';
import { AuthorizationService } from '../authorization.service';
import {
    SignUpDTO,
    SignInDTO,
    ConfirmSignUpDTO,
    PasswordRecoveryRequestDTO,
    ChangePasswordDTO,
} from '../authorization.dto';
import { UserService } from '../user/user.service';
import { ConfigService } from '@nestjs/config';
import { BasketService } from '../basket/basket.service';

@Controller('auth')
@ApiTags('Authorization')
export class AuthorizationController {
    constructor(
        private readonly authService: AuthorizationService,
        private readonly userService: UserService,
        private readonly configService: ConfigService,
        private readonly basketService: BasketService,
    ) {}

    @ApiOkResponse()
    @Post('signup')
    async signUp(@Body() signUpDTO: SignUpDTO) {
        const userExists = await this.userService.userExists(signUpDTO.email);
        if (userExists) {
            throw new ConflictException('Email already exists, please log in');
        }

        return this.authService.sendConfirmationEmail(signUpDTO);
    }

    @ApiOkResponse()
    @Get('confirmSignUp')
    async confirmSignUp(@Query() { token }: ConfirmSignUpDTO) {
        const createdUser = await this.authService.registerUser(token);

        await this.basketService.createBasket(createdUser.id);

        return createdUser;
    }
}

```

```

@ApiOkResponse()
@Post('signin')
signin(@Body() signInDTO: SignInDTO) {
    const { email, password } = signInDTO;

    return this.authService.signIn(email, password);
}

@ApiOkResponse()
@Post('requestPasswordRecovery')
async requestPasswordRecovery(@Body() { email }: PasswordRecoveryRequestDTO) {
    const user = await this.userService.findByEmail(email);
    if (!user) {
        throw new NotFoundException('User not found');
    }
    return this.authService.sendPasswordRecoveryEmail(email);
}

@ApiOkResponse()
@Post('changePassword')
changePassword(@Body() { token, password }: ChangePasswordDTO) {
    return this.authService.changePassword(token, password);
}
}

```

Лістинг файлу plugins.ts

```

module.exports = ({ env }) => ({
  upload: {
    config: {
      provider: 'aws-s3',
      providerOptions: {
        accessKeyId: env('AWS_ACCESS_KEY_ID'),
        secretAccessKey: env('AWS_ACCESS_SECRET'),
        region: env('AWS_REGION'),
        params: {
          Bucket: env('AWS_BUCKET'),
          ACL: env('AWS_ACL', 'public-read'),
        },
      },
      actionOptions: {
        upload: {},
        uploadStream: {},
        delete: {},
      },
    },
  },
});

```

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						67