

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:

«Мережевий ринок «TradeLink»

Виконав студент групи ПЗк-21-1
КУШНАРЬОВ Олексій Олександрович

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
САВІЦЬКИЙ Роман Святославович

Житомир – 2024

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

«ЗАТВЕРДЖУЮ»

Завідувач кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КУШНАРЬОВ Олексій Олександрович**

Керівник роботи: **ЧИЖМОТРЯ Олексій Володимирович**

Тема роботи: **«Мережевий ринок «TradeLink»,**

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

Вихідні дані для роботи: розробка мережевого ринку «TradeLink»

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	26.02.2024р.	26.02.2024р.
2	Чижмотря О.В.	26.02.2024р.	26.02.2024р.
3	Чижмотря О.В.	26.02.2024р.	26.02.2024р.

РЕФЕРАТ

Випускна робота бакалавра складається з комплексної програмної реалізації онлайн платформи дошки оголошень, а також пояснювальної записки. Пояснювальна записка до дипломної роботи містить 86 сторінок, 35 ілюстрацій, 13 таблиць, 5 додатків та 30 літературних джерел.

Основною метою даної роботи є розробка алгоритмів та реалізація програмного продукту, що надаватиме користувачам широкий спектр інструментів для розміщення, пошуку та керування оголошеннями.

У роботі визначено ключові завдання, пов'язані з розробкою онлайн сервісу дошки оголошень, проведено аналіз існуючих рішень на ринку, а також наведено сценарії роботи програмного продукту. Представлено огляд загальної структури додатку, а також описано алгоритми взаємодії між окремими компонентами додатку.

В рамках випускної роботи проводились експерименти з оптимізації швидкодії системи, в результаті яких були визначені оптимальні налаштування для швидкої та ефективної роботи з оголошеннями.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, ОГОЛОШЕННЯ, ANGULAR, AWS, JAVASCRIPT, NODE.JS, NEST.JS, POSTGRESQL.

					ІПЗ.КР.Б – 121 – 24 – ПЗ							
Змн.	Арк.	№ докум.	Підпис	Дата	Мережевий ринок «TradeLink» Пояснювальна записка			Літ.		Арк.	Аркушів	
Розроб.		О.О. Кушнар'юв										
Керівник		О.В. Чижмотря									3	86
								Житомирська політехніка, група ІПЗк-21-1				
Реценз.		Р.С. Савіцький										
Затверд.		Т.А. Вакалюк										

ABSTRACT

The bachelor's thesis consists of a complex software implementation of an online offers board platform, as well as an explanatory note. The explanatory note to the thesis contains 86 pages, 35 illustrations, 13 tables, 5 appendices and 30 literary sources.

The main goal of this work is the development of algorithms and the implementation of a software product that will provide users with a wide range of tools for placing, searching and managing offers.

The paper defines the key tasks related to the development of an online offers board service, analyzes existing solutions on the market, and also provides scenarios for the software product. An overview of the general structure of the application is presented, and the algorithms of interaction between individual components of the application are also described.

As part of the final work, experiments were conducted to optimize the performance of the system, as a result of which optimal settings were determined for fast and efficient work with offers.

KEY WORDS: ANGULAR, AWS, JAVASCRIPT, NODE.JS, NEST.JS, OFFERS, POSTGRESQL, WEB APPLICATION.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	6
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ.....	9
1.1. Постановка задачі	9
1.2. Аналіз існуючого програмного забезпечення	10
1.3. Обґрунтування вибору інструментальних засобів.....	15
Висновки до першого розділу.....	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	21
2.1. Технічне завдання на розробку системи.....	21
2.2. Аналіз вимог до програмного продукту	22
2.3. Алгоритм роботи та стани програмної системи.....	27
2.4. Проектування бази даних	30
Висновки до другого розділу	39
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ДОДАТКУ	40
3.1. Порядок встановлення та налаштування системи	40
3.2. Розробка основних функцій платформи.....	50
3.3. Опис інструкції користувача	57
3.4. Тестування роботи програмного продукту	62
Висновки до третього розділу.....	66
ВИСНОВКИ	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	68
ДОДАТКИ	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База Даних;
ОС – Операційна Система;
ПЗ – Програмне Забезпечення;
CSS - Cascading Style Sheets;
CSRF - Cross-Site Request Forgery;
DOM – Document Object Model;
HTML – HyperText Markup Language;
PostgreSQL – реляційна БД;
RxJS - Reactive Extensions for JavaScript;
S3 – Simple Storage Service;
SCSS – Sassy Cascading Style Sheets;
SEO – Search Engine Optimization;
SPA – Single Page Application;
SQL – Structured Query Language;
UI – User Interface;
UX – User Experience;

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

У сучасному світі, де інтернет відіграє все більшу роль, оголошення стають невід'ємною частиною нашого життя. Від онлайн-продажів до пошуку роботи та благодійності, оголошення надають зручний та доступний спосіб для людей зв'язатися та обмінюватися інформацією.

Оголошення - це короткі повідомлення, які містять різноманітну інформацію і, зазвичай, мають на меті комерційну діяльність, оренду житла, оголошення про культурні події та робочі пропозиції. Раніше найбільш поширеними засобами розміщення оголошень були газети, які виділяли для них окремі розділи. Однак з плином часу популярність друкованих ЗМІ поступово зменшується. Крім того, застарілі методи, такі як дошки на зупинках транспорту та біля будинків, мають свої недоліки, такі як локальність та перенасиченість неактуальними повідомленнями.

Майданчики для розміщення оголошень відіграють важливу роль у сфері інтернет-торгівлі. Малий бізнес може почати продавати товари та послуги без необхідності великих витрат на оренду приміщень та інші додаткові витрати. У сучасному світі також активно розвивається культура повторного використання, коли люди можуть економити, купуючи вживані товари, та при цьому залишатися свідомими споживачами. Веб-сайти для розміщення оголошень значно спрощують процес пошуку і покупки товарів. Хоча на ринку існує багато аналогічних платформ, кожна з яких має свої переваги та недоліки.

Зростання онлайн-платформ для розміщення оголошень, свідчить про те, що люди шукають більш зручні та ефективні способи пошуку та розміщення оголошень.

Метою даної кваліфікаційної роботи є розробка веб-застосунку дошки оголошень, який забезпечить зручний та ефективний інструмент для розміщення і пошуку оголошень, враховуючи сучасні вимоги та недоліки існуючих рішень.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

Встановлена мета обумовлює наступні завдання:

- аналіз існуючих рішень веб-застосунків для розміщення оголошень;
- визначення вимог до веб-застосунку;
- проектування архітектури власного застосунку;
- реалізація функціонального веб-застосунку дошки оголошень;
- тестування та вдосконалення системи для забезпечення зручності та ефективності використання.

Об'єктом дослідження є процеси та технології, що забезпечують функціонування веб-застосунків для розміщення оголошень, зокрема їх архітектура, користувацький інтерфейс, механізми взаємодії з користувачами та ефективність у виконанні основних завдань.

Предметом дослідження є методи та інструменти розробки веб-застосунків для розміщення оголошень, включаючи аналіз існуючих рішень, визначення вимог, проектування архітектури, реалізацію функціоналу, тестування та вдосконалення системи для забезпечення зручності та ефективності використання.

Оголошення в сучасному інформаційному просторі відіграють важливу роль у зв'язку людей та обміні інформацією. «TradeLink», як одна з ключових онлайн-платформ для розміщення оголошень, має значний потенціал для розвитку та вдосконалення. Ця робота досліджує можливості та перспективи «TradeLink», пропонуючи практичні рекомендації щодо його вдосконалення та розвитку.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Постановка задачі

Проект призначений для того, щоб розробити веб-сайт, з помічу якого користувач міг зручно обрати товар або послугу та швидко замовити або продати його без зайвих кроків. Інтернет-ресурс повинен бути реалізований як веб-сайт, доступний в Інтернеті, як через комп'ютер, так і через смартфон [25]. Додаток повинен складатись з взаємозалежних частин, а функції кожної частини повинні бути чітко розділені.

Програмний продукт повинен мати наступні функціональні характеристики:

- Доступ до системи за допомогою звичних для сучасної людини браузерів;
- Можливість створення облікового запису для покупців та продавців.
- Система підтвердження електронної пошти для забезпечення безпеки реєстрації [22].
- Можливість додавати та редагувати інформацію у профілі користувача (особисті дані, контактна інформація, фотографії тощо).
- Можливість блокування або видалення облікових записів в разі порушення правил.

Сайт повинен складатись з двох частин, а саме адміністративної та користувацької.

Зовнішні користувачі повинні мати такі можливості:

1. Користувач повинен мати можливість швидко та безпечно авторизуватися в системі за допомогою особистого облікового запису або пройти процес реєстрації [9; 11; 12].
2. Можливість перегляду та редагування особистого профілю, включаючи зміну особистої інформації, пароллю та інших деталей.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		9

3. Зручний пошук оголошень та можливість перегляду їхніх характеристик, цін та коментарів.
4. Забезпечення зручного процесу оформлення замовлення.
5. Надання користувачам можливості відстеження статусу свого замовлення.
6. Можливість залишення власних оцінок та відгуків на користувачів або оголошень в системі для надання рекомендацій іншим користувачам.
7. Доступ до контактної інформації служби підтримки для вирішення будь-яких питань чи проблем.

Для досягнення мети потрібно вирішити такі задачі:

- Виконати аналіз сайтів-аналогів
- Розробити вимоги до створюваного web-додатку
- Обрати інструментарій реалізації
- Створити базу даних оголошень
- Реалізувати структуру, функціонал та дизайн веб-сайту
- Виконати тестування роботи web-сайту

1.2. Аналіз існуючого програмного забезпечення

Більшість підприємств, які мають різноманітні масштаби, пристосовуються до сучасних реалій та створюють свою онлайн-присутність. У наш час, коли деякі фізичні заклади не можуть надавати традиційні офлайн-послуги, перехід до онлайн-платформ стає необхідністю. Важливо відзначити, що перехід в онлайн не є лише вимушеним рішенням у зв'язку з обставинами, такими як хвороби чи війна. Вже в мирний час, наявність онлайн-магазину приносить багато переваг: привертає нових клієнтів та полегшує життя існуючих. Зараз, в умовах зростаючої конкуренції та стрімкого розвитку технологій, ефективне управління бізнесом в інтернеті вимагає побудови високоякісних інформаційних систем з використанням сучасних інструментів.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		10

Для аналізу було обрано три сайти зі спорідненою тематикою та призначенням.

1. **OLX** – <https://www.olx.ua/uk/>

OLX працює в тому ж бізнес-сегменті, що і платформа, яка розробляється, і тому є основним конкурентом платформи [24].

OLX визначається як провідна світова платформа оголошень і працює як онлайн-маркетплейс у різних країнах. OLX допомагає об'єднувати людей для купівлі, продажу та обміну товарами та послугами. На OLX будь-хто може розміщувати оголошення зі свого мобільного пристрою або комп'ютера. Щомісяця мільйони людей використовують OLX для торгівлі широким спектром товарів, від електроприладів і меблів до автомобілів і нерухомості. Наразі OLX вважається найпопулярнішим сайтом в Україні для розміщення або пошуку товарів і послуг.

Перевагою OLX порівняно з розглянутою платформою є те, що він має велику кількість користувачів. Однак OLX також має певні недоліки, такі як висока плата за розміщення оголошень, повільний процес розміщення, тривале узгодження оголошень, обмежена кількість оголошень у певних категоріях, реклама, ризики при замовленні, відсутність можливості замовляти в середині сервісу.

Цей аналіз показує, що хоча OLX є важливим конкурентом, він має свої слабкі сторони, які можуть стати точкою зростання для платформи «TradeLink».

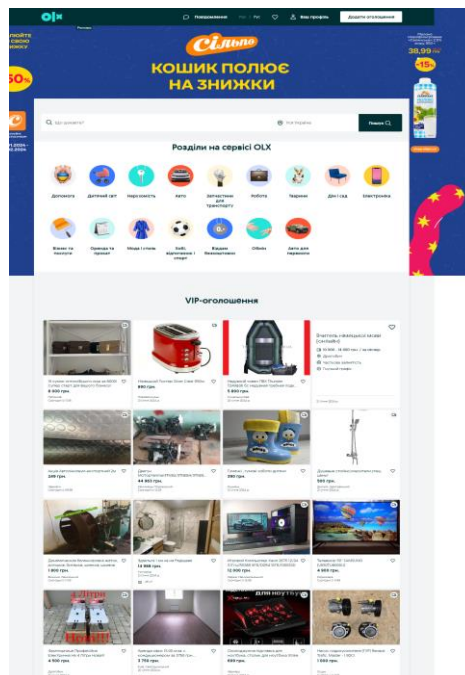


Рисунок 1.1 – Головна сторінка сайту OLX

2. **klubok.com** – <https://klubok.com/>

Klubok представляє собою український інтернет-сервіс, спрямований на функціонування ринку купівлі та продажу різноманітних товарів, зокрема, спеціалізується на галузі одягу та аксесуарів [1].

Серед недоліків сервісу можна виділити деякі аспекти, такі як відсутність можливості авторизації через соціальні мережі, відсутність онлайн-оплати та застарілий дизайн інтерфейсу. Також варто відзначити відсутність мобільних додатків та вузько направлену спеціалізацію на одяг та аксесуари. Значна комісія при відсутності можливості безпечної покупки є ще одним обмеженням для користувачів.

Несприятливі сторони можуть бути компенсовані наявною лояльністю клієнтів, яка визначається вже існуючими покупцями, що залишаються вірними сервісу. Однак важливо врахувати, що розвиток та удосконалення слабких сторін може позитивно вплинути на загальну ефективність та привабливість klubok.com для нових користувачів.

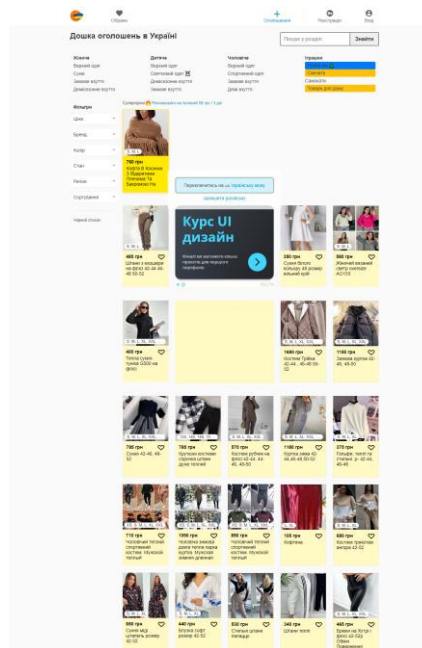


Рисунок 1.2 – Головна сторінка сайту «Klubok»

3. SHAFА – <https://shafa.ua/>

Shafa.ua – це доволі популярний маркетплейс, де користувач має можливість продавати речі, як нові так і ні. Даний сервіс містить користувацький інтерфейс, який забезпечує можливість зручного перегляду товарів та їхньої покупки або продажу, зручний користувацький інтерфейс, який дозволяє фільтрувати та сортувати товари в будьякому порядку [27].

Основними недоліками даного сервісу є те, що акаунти дуже часто блокують з будь-яких причин. Якщо користувача відносно висока кількість проданих товарів і його заблокували, то виникають великі труднощі з відновленням акаунту і часто ці спроби закінчуються невдачами. За кожен проданий товар, користувач повинен сплатити комісію 12% (може збільшуватись в залежності від кількості проданих товарів). Враховуючи волатильність у цій сфері, це не малий відсоток. Крім того відсоток потрібно оплачувати окремо. У чаті з продавцем/покупцем заборонено кидати номер телефону, пошту, посилання в іншому випадку акаунт буде заблоковано.

Рейтинг користувача не впливає на популярність його акаунту, адже хто заплатить за рекламу буде в топі, а людина, яка являється дійсною

перевіреною і має високий показник продуктивності може знаходитись в самому кінці. Ще однією незручністю є те, що заборонено виставляти товари з фото, яке вже існує на іншому товарі, навіть, якщо вони ідентичні, у разі невиконання цієї умови, користувач отримує блокування акаунту. Отже, даний сервіс налаштований суто на набиванні власного гаманця.

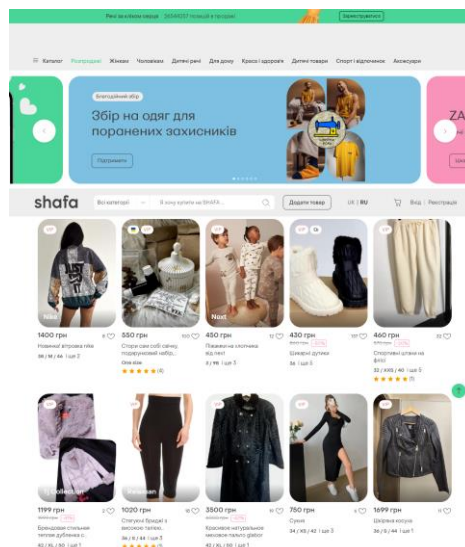


Рисунок 1.3 – Головна сторінка сайту «SHAFI»

Таблиця 1.1

Порівняння існуючих аналогів

Критерії	OLX	klubok.com	SHAFI	TradeLink
Занесення товарів у список обраного	+	+	+	+
Наявність великої кількості категорій	+	-	-	+
Плата за розміщення	+	-	-	-
Перевантаженість інтерфейсу	-	-	+	-
Реклама	+	+	+	-

1.3. Обґрунтування вибору інструментальних засобів

Обґрунтування вибору інструментальних засобів є важливим етапом у розробці будь-якого програмного продукту чи веб-сайту.

Angular — це JavaScript-фреймворк, розроблений компанією Google для створення та керування клієнтською частиною односторінкових додатків (SPA). Розробники мали на меті створити фреймворк на основі нативного JavaScript, який би розділяв логіку додатку та маніпуляції з DOM, а також забезпечував динамічне оновлення сторінок. Особливістю Angular є можливість ререндерінгу лише тієї частини DOM-дерева, що була оновлена, а не всього DOM-дерева, що робить його дуже потужним інструментом, який здобув широку популярність серед розробників [4].

Однією з головних інновацій Angular є концепція двосторонньої прив'язки даних, де представлення автоматично оновлюється при зміні моделі, і навпаки, зміни в представленні оновлюють модель. Також Angular представив концепцію директив, які дозволяють створювати власні HTML-атрибути та теги, що стають динамічними та анімованими. Ще однією важливою новацією є ін'єкція залежностей, яка полегшує зв'язування компонентів додатку та сприяє зменшенню повторного використання коду.

Angular побудований на основі MVC-структури, що робить його особливо зручним для розробки SPA. Основою фреймворку є Typescript, HTML та CSS/SCSS, що дозволяє розробникам уникати ручних маніпуляцій з DOM та полегшує написання та відлагодження коду за допомогою типізації даних [4].

Переваги Angular:

- Підтримка Google: Оскільки фреймворк розроблений та підтримується Google, можна не сумніватися у його подальшому розвитку та підтримці.

- Спрощена прив'язка даних: Angular дозволяє легко прив'язувати дані до HTML за допомогою виразів, а директиви дозволяють розширювати функціонал HTML та створювати нові конструкції.
- TypeScript: Angular використовує строго типізовану мову програмування TypeScript, що допомагає уникнути багатьох помилок та спрощує дебагінг.
- Інтернаціоналізація (i18n): Забезпечує готовність додатку до використання в різних регіонах [21].
- Підтримка відкладеного завантаження: Модулі Angular можуть завантажуватись лише при необхідності, що зменшує час першого завантаження сторінки.
- Ін'єкція залежностей: Полегшує розробку та тестування додатків, зменшуючи кількість зв'язків між компонентами.
- Роутінг: AngularRouter підтримує маршрутизацію та навігацію, забезпечуючи ефективне завантаження елементів у залежності від URL.
- Підтримка Ionic: Фреймворк, що дозволяє розробляти кросплатформені додатки для Android та iOS.
- Велика кількість допоміжних пакетів: Забезпечує додатковий функціонал і розширює можливості розробників.

Недоліки Angular:

- Складність вивчення: Через повноцінний фреймворк та необхідність вивчення специфічних правил, поріг входження в Angular значно вищий, ніж у бібліотек, таких як React.
- Труднощі з SEO: Додатки на Angular, як і інші SPA, погано оптимізовані для індексації пошуковими роботами через динамічний контент.

- Часті оновлення: Часті нові версії можуть створювати труднощі для новачків, які працюють зі старими кодовими базами.
- Зменшення популярності: З появою нових бібліотек і фреймворків, таких як React та Vue, Angular дещо втратив свою популярність.

PrimeNG – це велика колекція UI-компонентів для Angular. Вона розроблена компанією «PrimeTek Informatics», яка має багаторічний досвід розробки бібліотек UI-компонентів з відкритим кодом. PrimeNG також є братом популярної бібліотеки інтерфейсу користувача, такої як JavaServer Faces, PrimeReact і PrimeVue. [17].

RxJS - це бібліотека для створення асинхронних і подієвих програм за допомогою спостережуваних послідовностей. Вона забезпечує один основний тип: «Observable» і три супутні типи: «Observer», «Scheduler» і «Subject» [26].

Node.js – це асинхронний механізм, керований подіями, в якому програма виконує запит, а потім продовжує працювати над іншими корисними завданнями, а не зупиняється, чекаючи на відповідь. Після завершення запитаного завдання програма інформується про результати за допомогою зворотного виклику. Це дозволяє паралельно виконувати велику кількість операцій, що важливо при масштабуванні програм [5].

Nest.js – це прогресивний фреймворк для створення серверної частини застосунків на Node.js за допомогою TypeScript. Він базується на концепціях модульності, декораторів і залежностей, надаючи інструменти для легкої розробки програм, які можна масштабувати та підтримувати [2].

Passport – покращує безпеку завдяки своїм надійним стратегіям, таким як «JWT» і «OAuth», узгоджуючи з найкращими галузевими практиками. Серед іншого, він підтримує архітектуру «plug-and-play», що дозволяє

додавати нові методи автентифікації або замінювати існуючі без істотних змін коду [8].

PostgreSQL – відома своєю надійністю, високою швидкістю та можливістю ефективно працювати з великими обсягами даних. Це особливо корисно для проектів, які вимагають стабільної роботи бази даних та масштабування. PostgreSQL підтримує ряд продвинутих функцій, таких як операції з геоданими, текстовий та графічний пошук, що може бути важливо для різноманітних типів додатків. PostgreSQL може легко інтегруватися з іншими інструментами та технологіями, надаючи гнучкість в роботі з різними екосистемами [16].

Amazon Simple Storage Service (S3) - це веб-сервіс для зберігання та обміну даними в інтернеті. Онлайн-сховище Amazon S3 надає можливість зберігати великі обсяги даних, які будь-де та будь-коли можна отримати. Amazon S3 може зберігати петабайти даних, дає можливість завантажувати будь-яку кількість даних, і S3 автоматично масштабується, щоб забезпечити потрібну продуктивність [10; 23].

S3 гарантує високу доступність. Дані реплікуються на різних серверах та центрах обробки даних для забезпечення неперервного доступу. Amazon S3 пропонує різні методи захисту даних, включаючи шифрування в покої та під час передачі, контроль доступу на рівні об'єктів та ідентифікацію користувачів.

S3 дозволяє зберігати кілька версій об'єктів, щоб надати можливість відновити дані, якщо вони були випадково змінені або видалені. Можно використовувати Amazon S3 для зберігання різних типів даних, таких як зображення, відео, документи тощо. Крім того, можна налаштовувати правила зберігання для автоматичного переміщення або видалення даних відповідно до ваших потреб.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

S3 інтегрується з іншими сервісами Amazon Web Services (AWS), такими як Amazon EC2, Amazon RDS тощо, що робить його ідеальним вибором для зберігання даних для проєкту в області хмарних обчислень.

Враховуючи всі ці переваги, Amazon S3 є потужним і надійним засобом зберігання даних в хмарі, який може використовуватися для різних завдань, від резервного копіювання даних до веб-хостингу.

Висновки до першого розділу

Проєкт передбачає створення веб-сайту для зручного створення та перегляду оголошень. Потрібно провести аналіз аналогів, сформулювати вимоги, обрати інструменти, розробити базу даних та функціонал сайту, протестувати його. Важливі функції для користувачів: авторизація, редагування профілю, пошук товарів, оформлення замовлення, відстеження статусу замовлення та підтримка.

Загальний висновок полягає в тому, що перехід до онлайн-платформ для бізнесу стає необхідністю у сучасних умовах, які включають не лише вимушені обставини, але й стрімкий розвиток технологій та зростаючу конкуренцію. Основні переваги онлайн-присутності для підприємств полягають у привертанні нових клієнтів та полегшенні життя існуючих.

Аналіз трьох веб-сайтів, що мають споріднену тематику, дозволяє виявити як переваги, так і недоліки існуючих рішень на ринку. OLX, незважаючи на свою популярність, має свої недоліки, такі як висока вартість розміщення оголошень та обмежена кількість оголошень у певних категоріях. Klubok.com має обмежену спеціалізацію та деякі технічні обмеження, такі як відсутність можливості онлайн-оплати та застарілий дизайн. Shafa.ua, хоча має зручний інтерфейс, також має свої недоліки, такі як часті блокування акаунтів та високі комісійні відсотки.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		19

Отже, ефективне управління бізнесом в інтернеті вимагає побудови високоякісних інформаційних систем з використанням сучасних інструментів, а також уважного аналізу конкурентного середовища та вивчення недоліків існуючих рішень на ринку.

Angular, RxJS, PrimeNG, Node.js/Nest.js, Passport, PostgreSQL та Amazon S3 - це потужні та ефективні інструменти, які можуть бути використані для різних аспектів проекту, починаючи від розробки клієнтської частини та серверної логіки до зберігання та обміну даними в хмарному середовищі.

Angular, розроблений Google, пропонує потужні можливості для створення односторінкових додатків, забезпечуючи прив'язку даних, директиви та ін'єкцію залежностей, а в поєднанні з допоміжними бібліотеками RxJS та PrimeNG дозволяє швидко та якісно будувати нові застосунки. Node.js та Nest.js надають асинхронний механізм, що дозволяє виконувати багато операцій паралельно та ефективно масштабувати програми, в той час як Passport надає надійність та безпеку в роботі додатку. PostgreSQL забезпечує надійність та високу швидкодію бази даних, а Amazon S3 - надійне та масштабоване сховище даних в хмарі.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		20

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Технічне завдання на розробку системи

1. Назва системи:

- Повне найменування системи: **Мережевий ринок «TradeLink»**.
- Коротке найменування системи: Програма, Програмне забезпечення (ПЗ), Програмний комплекс, Система.

2. Призначення системи:

- Програмний комплекс має на меті надати ефективну та безпечну платформу для онлайн-торгівлі різноманітними товарами та послугами.

3. Вимоги до системи:

- Доступ до системи за допомогою звичних для сучасної людини браузерів;
- Можливість створення облікового запису для покупців та продавців.
- Система підтвердження електронної пошти для забезпечення безпеки реєстрації [22].
- Можливість додавати та редагувати інформацію у профілі користувача (особисті дані, контактна інформація, фотографії тощо).
- Можливість блокування або видалення облікових записів в разі порушення правил.

4. Вимоги до інтерфейсу:

- Вимоги до інтерфейсу мають на меті створити зручний, інтуїтивно зрозумілий та привабливий інтерфейс для ефективної взаємодії користувачів з програмним комплексом «TradeLink».

5. На інтерфейси користувача накладаються наступні вимоги:

- мінімальний час відгуку системи;

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

- повідомлення про критичні ситуації системи;
- отримання відповіді на дії користувача;
- відображення процесу завантаження для ресурсомістких завдань.

2.2. Аналіз вимог до програмного продукту

Давайте розглянемо високорівневі вимоги, які повинна враховувати система «Мережевий ринок «TradeLink», щоб ефективно виконувати свої функції та відповідати встановленим стандартам, специфікаціям і іншим формальним документам. Розглянуті умови, які користувач повинен дотримуватися для взаємодії з системою, та висвітлені ключові бізнес-переваги. Описані можливості системи, обмеження та показники якості також вказані для повноти розуміння її функціоналу.

Бізнес-вимоги:

1. Основні цілі: Розробка та ефективне управління онлайн-сервісом для продажу товарів або послуг з метою забезпечення високоякісного та зручного інтернет-торгівельного досвіду для користувачів.
2. Представлення проєкту: проєкт буде реалізовано у вигляді сайту та серверу, що містить структуровані ресурси з теми пов'язаними за даною тематикою.

Вимоги користувачів

1. Користувач повинен мати можливість швидко та безпечно авторизуватися в системі за допомогою особистого облікового запису або пройти процес реєстрації.
2. Можливість перегляду та редагування особистого профілю, включаючи зміну особистої інформації, фотографії профілю та інших деталей.
3. Зручний пошук оголошень та можливість перегляду їхніх характеристик, власників, цін та коментарів.
4. Забезпечення зручного процесу оформлення замовлення.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		22

5. Надання користувачам можливості відстеження статусу свого замовлення.
6. Можливість залишення коментарів під оголошенням для авторизованих користувачів.
7. Рейтинг користувачів.
8. Мультиязичність застосунку [21].
9. Вибір теми інтерфейсу [28].
10. Доступ до контактної інформації служби підтримки для вирішення будь-яких питань чи проблем.

Характеристика об'єкту комп'ютеризації:

Об'єкт комп'ютеризації представляє собою онлайн-платформу для продажу різноманітних товарів або послуг через Інтернет. Цей сервіс надає можливість користувачам створювати облікові записи, шукати й створювати оголошення та використовувати інші функціональності для зручного та безпечного онлайн-шопінгу.

Функціональні вимоги:

1. Авторизація користувачів в системі: В системі повинна бути представлена можливість реєстрації користувача та присвоєння йому відповідної ролі.
2. Можливість збереження інформації: Система повинна зберігати інформацію і надавати можливість користувачу керувати нею.

Нефункціональні вимоги:

1. Сприйняття
 - Час, потрібний для навчання інструментами роботи з інформаційною системою для звичайних користувачів – 10-15 хвилин, для досвідчених – 5 хвилин;
 - Час відповіді системи для звичайних запитів не повинен перевищувати 5 секунд, а для більш складних запитів – 10 сек.

- Інтерфейс повинен бути інтуїтивно зручним для користувача та не вимагати додаткової підготовки.
2. Продуктивність
- Система повинна підтримувати мінімум 100 користувачів, що одночасно працюють, пов'язаних з спільною базою даних.
3. Можливість експлуатації
- Масштабування – система повинна мати можливість збільшувати потужності (продуктивність), зі збільшенням користувачів таким чином, щоб це негативно не відобразилося на її роботі.

Діаграма варіантів використання

Діаграма використання (Use Case Diagram) є графічним представленням функціональностей системи та їхніх взаємодій з акторами (користувачами чи іншими системами). Вона допомагає визначити основні можливості системи та ролі, які взаємодіють з цими можливостями [3; 6].

Основні елементи діаграми використання:

1. Актори: Представляють ролі, які взаємодіють з системою. Актори можуть бути користувачами, іншими системами чи зовнішніми компонентами.
2. Використання (Use Case): Представляє конкретний функціональний віджет або сервіс, який надає система. Використання описуються як дії, які виконуються системою, щоб задовольнити потреби акторів.
3. Відносини між акторами та використаннями: Використовується для відображення взаємодії між акторами та конкретними використаннями. Наприклад, стрілки можуть вказувати, які використання доступні для конкретного актора.

Діаграма використання допомагає зрозуміти як користувачі або інші системи взаємодіють із системою, які функції вони використовують та які їх

очікування відносно системи. Це допомагає визначити вимоги до системи та створити базу для подальших етапів розробки.

Гість – актор, який може бачити інформацію про оголошення, але не може здійснювати покупки, залишати коментарі або взаємодіяти із особистим обліковим записом.

Користувач – актор, який має обліковий запис та може використовувати розширені функції, такі як додавання оголошень, керування профілем тощо.

Адміністратор – актор, який має права для управління та модерації платформи, включаючи блокування некоректних оголошень та взаємодію з користувачами.

Супер-адміністратор – актор, який має розширені права адміністратора, що дозволяють йому керувати іншими адміністраторами та користувачами.

Діаграма використання допомагає узагальнити та візуалізувати взаємодію між усіма учасниками системи та процеси, які вони виконують.

Діаграма варіантів використання в нотації UML представлена на рисунку 2.1.

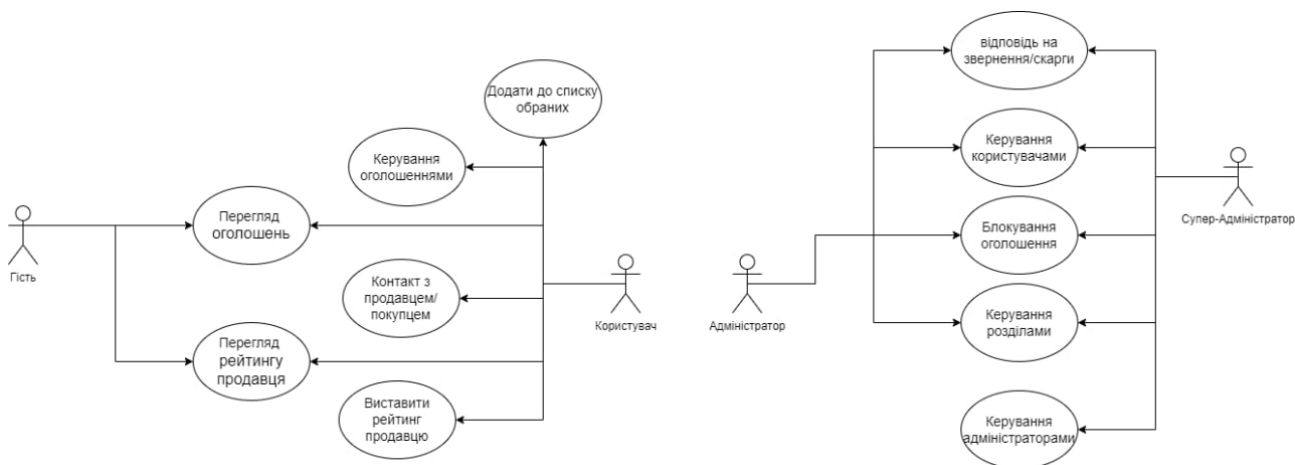


Рисунок 2.1 – Діаграма варіантів використання

Діаграма відображає структуру функціональних можливостей та ролей у системі оголошень. На діаграмі використовуються різні актори (користувачі системи) та дії, які вони можуть виконувати.

1. Гість:

- Перегляд оголошень – гості мають можливість переглядати оголошення на сайті.
- Перегляд рейтингу продавця – гості можуть переглядати рейтинги продавців.

2. Користувач:

- Керування оголошеннями – користувачі можуть створювати, редагувати та видаляти свої оголошення.
- Контакт з продавцем/покупцем – користувачі можуть контактувати з іншими користувачами для здійснення угод.
- Виставлення рейтингу продавцю – користувачі можуть виставляти рейтинги продавцям або споживачам після завершення угоди.
- Додавання до списку обраних – користувачі можуть додавати оголошення до списку обраних для зручного доступу в майбутньому.

3. Адміністратор:

- Відповіді на звернення/скарги – адміністратори відповідають на звернення та скарги користувачів, забезпечуючи підтримку і вирішення проблем.
- Управління користувачами – адміністратори мають можливість керувати обліковими записами користувачів, включаючи активацію, деактивацію та редагування інформації.
- Блокування оголошень – адміністратори можуть блокувати оголошення, які порушують правила платформи.
- Управління розділами – адміністратори відповідають за структурування і управління різними розділами/категоріями сайту.

4. Супер-Адміністратор:

- Всі можливості звичайного адміністратора.
- Управління адміністраторами – супер-адміністратор має право управляти обліковими записами адміністраторів, включаючи призначення нових адміністраторів та редагування їх прав.

Ця діаграма варіантів використання дозволяє візуалізувати основні можливості, які надає система «Мережевий ринок «TradeLink», та взаємодію різних акторів в рамках платформи для ефективної та зручної торгівлі товарами та послугами.

2.3. Алгоритм роботи та стани програмної системи

Діаграми активності, діаграми варіантів використання та діаграми станів вважаються представниками діаграм поведінки, оскільки вони відтворюють процеси та взаємодії, що мають відбуватися в системі, яку вони моделюють.

Діаграма діяльності - це вид UML-діаграми, який використовується для моделювання послідовності дій, процесів або алгоритмів в системі. Вона відображає послідовність активностей та переходи між ними, дозволяючи візуалізувати робочі процеси та їхні взаємодії [3; 6].

Основні елементи діаграми діяльності включають:

1. Активності (Activity): Представляють конкретні дії або операції, які виконуються в системі. Активності зображуються у прямокутниках з заокругленими кутами.
2. Рішення (Decision): Позначають рішення або умовні галузі у виконанні дій. Позначаються ромбами та можуть мати різні шляхи виконання в залежності від умов.
3. Вилучення (Merge): Вказують на об'єднання шляхів виконання після умовного розгалуження. Також позначаються ромбами.

4. Потік управління (Control Flow): Вказує на порядок виконання активностей. Позначається стрілками, які з'єднують активності.
5. Об'єднання та розгалуження (Fork/Join): Використовуються для моделювання паралельних процесів. Об'єднання (Join) позначає об'єднання паралельних шляхів, а розгалуження (Fork) - розділення.
6. Підпрограма (Call Behavior Action): Вказує на виклик іншої діаграми або підпрограми для виконання певної функції.
7. Обмеження часу (Time Constraint): Вказує на обмеження часу виконання певної активності.

Діаграми діяльності допомагають зрозуміти та визначити порядок виконання операцій, логіку процесів та їх взаємодії в системі.

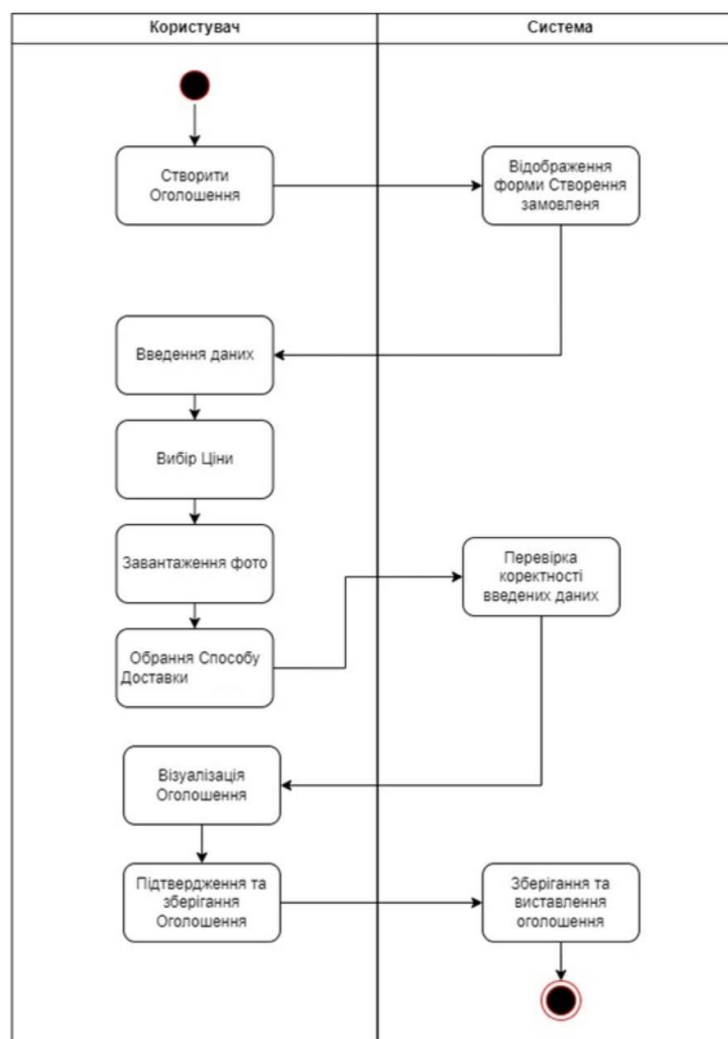


Рисунок 2.2 – Діаграма діяльності «Створення оголошення»

Розглянемо уточнений процес «Створення оголошення» з участю користувача та системи:

Користувач починає процес, обираючи опцію «Створити оголошення» у відповідному розділі системи. Система виводить користувачеві форму створення оголошення з необхідними полями та опціями. Користувач вводить дані про товар чи послугу, такі як назва, опис, фотографії, категорія, ціна та способи доставки.

Система перевіряє введені користувачем дані на відповідність вимогам. Система візуалізує оголошення у користувача, дозволяючи йому переглянути, як виглядатиме оголошення для інших користувачів. Користувач переглядає оголошення, підтверджує його точність та зберігає в системі. Система зберігає створене оголошення, роблячи його доступним для інших користувачів.

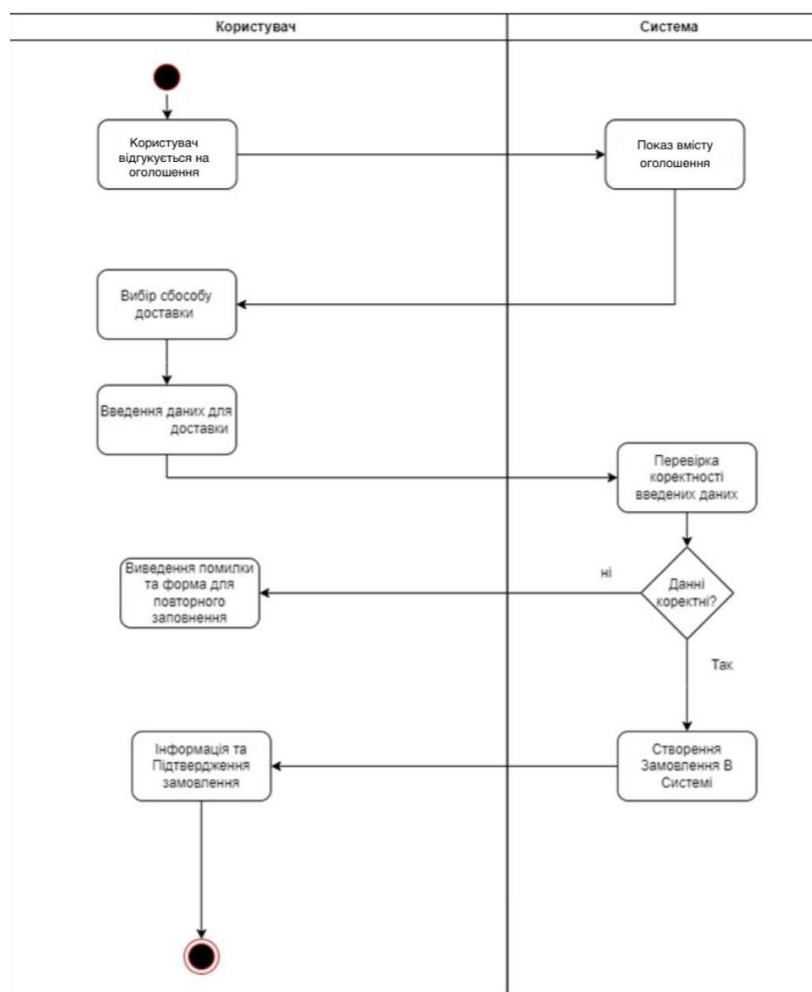


Рисунок 2.3 – Діаграма діяльності оформлення замовлення

Користувач у веб-інтерфейсі вибирає оголошення та приймає його. Веб-система відображає користувачеві вміст прийнятого оголошення та надає можливість обрати спосіб доставки замовлення. Користувач обирає умови доставки та підтверджує оформлення замовлення через веб-інтерфейс. Система перевіряє отримані дані про замовлення на відповідність та доступність, готує відповідь для відображення та інформує користувача.

Цей процес враховує взаємодію користувача та системи на кожному етапі оформлення замовлення, надаючи конкретні дії та учасників на кожному етапі процесу.

2.4. Проектування бази даних

Проектування бази даних є критичним етапом у розробці будь-якої інформаційної системи, оскільки від його правильності залежить ефективність та надійність роботи всієї системи. Цей процес включає аналіз вимог, моделювання структури даних та визначення зв'язків між ними, що дозволяє забезпечити оптимальне зберігання та доступ до інформації.

PostgreSQL надає можливість швидко, ефективно та надійно спроектувати БД для застосунку, враховуючи можливі складності:

- Взаємозв'язки між таблицями;
- Масштабування;
- Локалізація;
- «Повнотекстовий пошук» («full text search»);
- Складні типи даних (JSON, бінарні, координати, геометричні тощо);
- Можливість корегування/оновлення структури таблиць після їх створення.

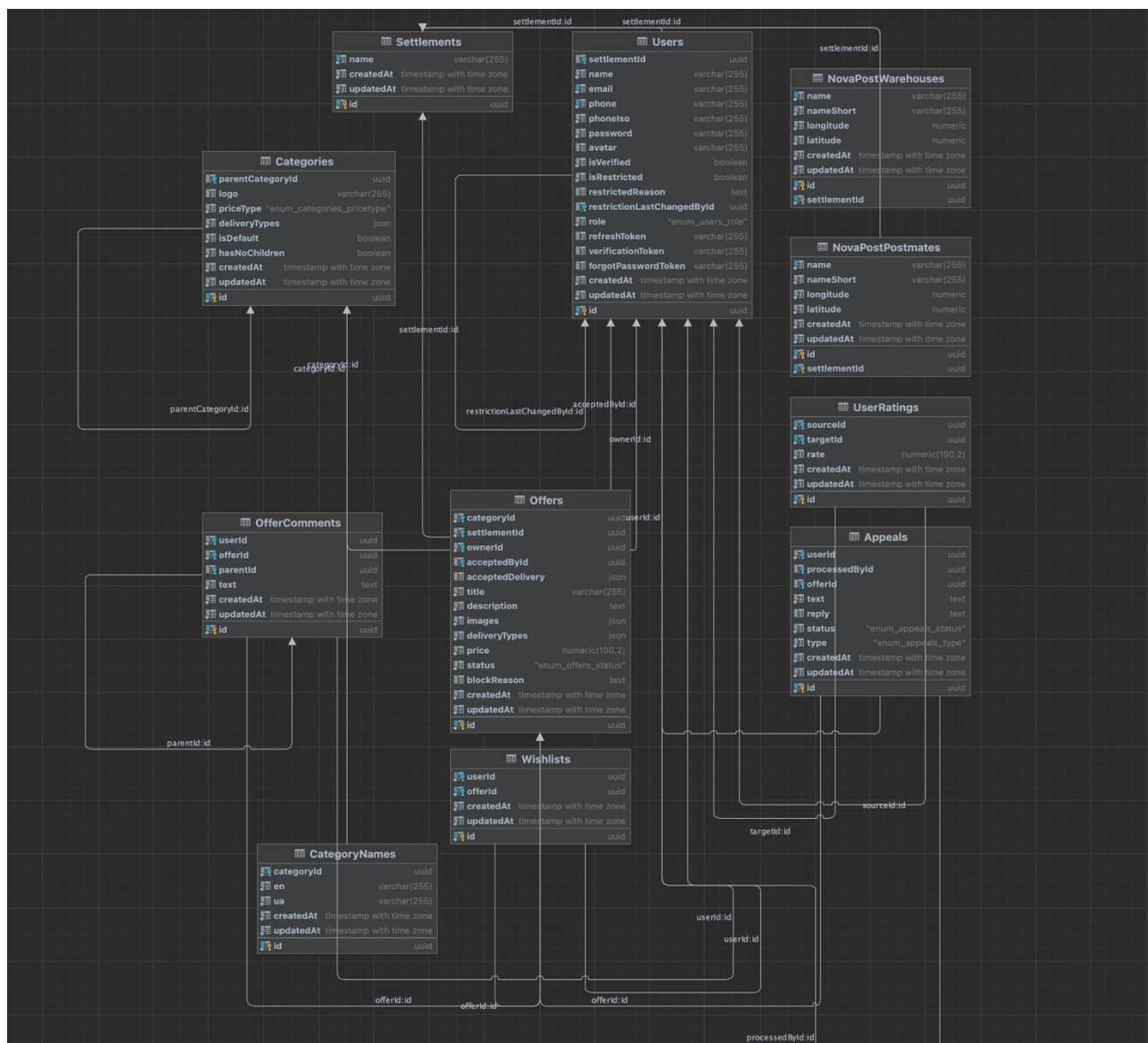


Рисунок 2.6 – Діаграма бази даних

Users: Зберігає інформацію про користувачів, включаючи їхні імена, електронну пошту, паролі, номери телефонів, населених пунктів, ролей тощо, а також включає дати створення та оновлення записів.

Таблиця 2.1

Структура таблиці «Users»

Назва	Тип даних	ПК	ЗК	Опис поля
1.	2.	3.	4.	5.
id	uuid	+	–	ID користувача

Продовження таблиці 2.1

1.	2.	3.	4.	5.
settlementId	uuid	—	+	ID населеного пункту, до якого належить користувач
restrictionLast ChangedById	uuid	—	+	ID користувача, який вніс останні зміни до обмежень
name	varchar	—	—	Ім'я користувача
email	varchar	—	—	Електронна пошта
phone	varchar	—	—	Номер телефону
phoneIso	varchar	—	—	Iso-код країни
password	varchar	—	—	Пароль
avatar	varchar	—	—	Посилання на зображення аватара користувача
isVerified	boolean	—	—	Прапорець, який позначає, чи підтверджено обліковий запис користувача
isRestricted	boolean	—	—	Прапорець, який позначає, чи обмежений обліковий запис користувача
restrictedReason	text	—	—	Причина обмеження облікового запису користувача
role	enum	—	—	Роль користувача
refreshToken	varchar	—	—	Токен оновлення для аутентифікації
verificationToken	varchar	—	—	Токен підтвердження облікового запису
forgotPasswordToken	varchar	—	—	Токен для відновлення пароля

Продовження таблиці 2.1

1.	2.	3.	4.	5.
createdAt	datetime	–	–	Дата створення
updatedAt	datetime	–	–	Дата оновлення

Settlements: містить інформацію про різні населені пункти.

Таблиця 2.2

Структура таблиці «Settlements»

Назва	Тип даних	ПК	ЗК	Опис поля
id	uuid	+	–	ID населеного пункту
name	varchar	–	–	Назва населеного пункту
createdAt	datetime	–	–	Дата створення
updatedAt	datetime	–	–	Дата оновлення

Categories: зберігає інформацію про різні категорії/розділів об'єктів, продуктів або послуг у системі. Категорії можуть бути використані для організації та класифікації даних, спрощення пошуку та фільтрації, а також для створення ієрархічної структури для управління інформацією.

Таблиця 2.3

Структура таблиці «Categories»

Назва	Тип даних	ПК	ЗК	Опис поля
1.	2.	3.	4.	5.
id	uuid	+	–	ID категорії
parentCategoryId	uuid	–	+	ID батьківської категорії
logo	varchar	–	–	Логотип категорії
priceType	enum	–	–	Тип ціни
diliveryTypes	json	–	–	Типи доставки
isDefault	boolean	–	–	Чи є категорія за замовчуванням

Продовження таблиці 2.3

1.	2.	3.	4.	5.
hasNoChildren	boolean	—	—	Чи немає підкатегорій
createdAt	datetime	—	—	Дата створення
updatedAt	datetime	—	—	Дата оновлення

NovaPostWarehouses: містить інформацію про відділення компанії «Нова Пошта». У таблиці зберігаються дані про розташування відділень, їх назви та інші характеристики, необхідні для ефективного управління логістичними процесами та постачанням.

Таблиця 2.4

Структура таблиці «NovaPostWarehouses»

Назва	Тип даних	ПК	ЗК	Опис поля
id	uuid	+	—	ID відділення
settlementId	uuid	—	+	ID населеного пункту
name	varchar	—	—	Назва відділення
nameShort	varchar	—	—	Коротка назва
longitude	numeric	—	—	Довгота
latitude	numeric	—	—	Широта
createdAt	datetime	—	—	Дата створення
updatedAt	datetime	—	—	Дата оновлення

NovaPostPostmates: містить інформацію про поштомати компанії «Нова Пошта». У таблиці зберігаються дані про розташування поштоматів, їх назви та інші характеристики, необхідні для ефективного управління логістичними процесами та постачанням.

Таблиця 2.5

Структура таблиці «NovaPostPostmates»

Назва	Тип даних	ПК	ЗК	Опис поля
id	uuid	+	–	ID поштомату
settlementId	uuid	–	+	ID населеного пункту
name	varchar	–	–	Назва поштомату
nameShort	varchar	–	–	Коротка назва
longitude	numeric	–	–	Довгота
latitude	numeric	–	–	Широта
createdAt	datetime	–	–	Дата створення
updatedAt	datetime	–	–	Дата оновлення

Offers: містить дані про оголошення. Вона включає унікальний ідентифікатор оголошення, інформацію про категорію, населений пункт та власника пропозиції, а також дати прийняття доставки, назву, опис, зображення, типи доставки, ціну, статус та причину блокування.

Таблиця 2.6

Структура таблиці «Offers»

Назва	Тип даних	ПК	ЗК	Опис поля
1.	2.	3.	4.	5.
id	uuid	+	–	ID оголошення
categoryId	uuid	–	+	ID категорії
settlementId	uuid	–	+	ID населеного пункту
ownerId	uuid	–	+	ID власника пропозиції
acceptedById	uuid	–	+	ID користувача, який прийняв пропозицію
acceptedDelivery	json	–	–	Інформація про обраний спосіб доставки

Продовження таблиці 2.6

1.	2.	3.	4.	5.
title	varchar	—	—	Назва оголошення
description	text	—	—	Опис оголошення
images	json	—	—	Зображення
deliveryTypes	json	—	—	Допустимі типи доставки
price	numeric	—	—	Ціна оголошення
status	enum	—	—	Статус оголошення
blockReason	text	—	—	Причина блокування оголошення
createdAt	datetime	—	—	Дата створення
updatedAt	datetime	—	—	Дата оновлення

CategoryNames: призначена для зберігання імен категорій у різних мовах з використанням ідентифікаторів категорій для пов'язання з відповідними категоріями.

Таблиця 2.7

Структура таблиці «CategoryNemes»

Назва	Тип даних	ПК	ЗК	Опис поля
id	uuid	+	—	ID ім'я категорії
categoryld	uuid	—	+	ID категорії
en	varchar	—	—	Назва категорії англійською мовою
ua	varchar	—	—	Назва категорії українською мовою
createdAt	datetime	—	—	Дата створення
updatedAt	datetime	—	—	Дата оновлення

UserRatings: призначена для зберігання інформації про оцінки, які користувачі надають іншим користувачам у системі.

Таблиця 2.8

Структура таблиці «UserRatings»

Назва	Тип даних	ПК	ЗК	Опис поля
id	uuid	+	–	ID оцінки користувача
sourceId	uuid	–	+	ID користувача, який виставляє оцінку
targetId	uuid	–	+	ID користувача, якому виставляється оцінка
rate	numeric	–	–	Оцінка
createdAt	datetime	–	–	Дата створення
updatedAt	datetime	–	–	Дата оновлення

Wishlists: призначена для зберігання інформації про побажання користувачів, які вони додали до свого списку бажань. Кожне побажання має ідентифікатор користувача та оголошення, яке вони хочуть додати до свого списку бажань.

Таблиця 2.9

Структура таблиці «Wishlists»

Назва	Тип даних	ПК	ЗК	Опис поля
id	uuid	+	–	ID побажання
userId	uuid	–	+	ID користувача
offerId	uuid	–	+	ID оголошення
createdAt	datetime	–	–	Дата створення
updatedAt	datetime	–	–	Дата оновлення

OfferComments: призначена для зберігання коментарів користувачів до оголошень. Кожний коментар має ідентифікатор користувача, ідентифікатор оголошення та, в деяких випадках, ідентифікатор батьківського коментаря (якщо цей коментар є відповіддю на інший коментар). Текст коментаря зберігається у вигляді текстового поля.

Таблиця 2.10

Структура таблиці «OfferComments»

Назва	Тип даних	ПК	ЗК	Опис поля
id	uuid	+	–	ID коментаря
userId	uuid	–	+	ID користувача
offerId	uuid	–	+	ID оголошення
parentId	uuid	–	+	ID батьківського коментаря, якщо цей коментар є відповіддю на інший коментар
text	text	–	–	Текст коментаря
createdAt	datetime	–	–	Дата створення
updatedAt	datetime	–	–	Дата оновлення

Appeals: призначена для зберігання інформації про звернення або скарги на оголошення до адміністраторів застосунку. Вона включає дані про користувачів, оголошення, текст та відповідь на апеляцію, а також статус та тип апеляції.

Таблиця 2.11

Структура таблиці «Appeals»

Назва	Тип даних	ПК	ЗК	Опис поля
1.	2.	3.	4.	5.
id	uuid	+	–	ID апеляції

1.	2.	3.	4.	5.
userId	uuid	—	+	ID користувача
processedById	uuid	—	+	ID адміністратора, який обробив цю апеляцію
offerId	uuid	—	+	ID оголошення
text	text	—	—	Текст апеляції
reply	text	—	—	Відповідь на апеляцію
status	enum	—	—	Статус апеляції
type	enum	—	—	Тип апеляції
createdAt	datetime	—	—	Дата створення
updatedAt	datetime	—	—	Дата оновлення

Висновки до другого розділу

У другому розділі ми провели докладний аналіз функціональності нашої системи та вивчили послідовність операцій, які вона виконує. Для більш детального відображення структури та взаємодії компонентів системи були створені діаграми діяльності та послідовності.

Детальний аналіз та використання відповідних інструментів проєктування дозволили нам не лише зрозуміти внутрішню логіку системи, але й розробити оптимальну архітектуру, що враховує вимоги щодо гнучкості та масштабованості.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ДОДАТКУ

3.1. Порядок встановлення та налаштування системи

Архітектура мережевого ринку «TradeLink» базується на трирівневій архітектурі, яка є однією з найпоширеніших у світі. Вона складається з трьох основних рівнів: Клієнт, Сервер та База даних.



Рисунок 3.1 – Архітектура «Клієнт-сервер» мережевого ринку «TradeLink»

В архітектурі клієнт/сервер комп'ютери виконують дві ролі:

- Клієнт запитує певні послуги або ресурси.
- Сервер обробляє ці запити, надаючи запитувані послуги або ресурси.

Особливістю цієї архітектури є те, що сам веб-додаток розміщується і виконується на сервері, тоді як клієнт отримує лише результати його роботи. Функціонування такої програми засноване на отриманні запитів від користувача, їх обробці та поверненні результатів. Передача запитів та оброблених даних здійснюється через Інтернет за допомогою протоколу HTTP(S). Однією з переваг цього підходу є те, що клієнти не залежать від

операційної системи користувача, що робить веб-додатки міжплатформенними сервісами.

Клієнт – комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій.

Сервер – більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних.

Модель такої системи полягає в тому, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат відправляється клієнтові. Сервер може обслуговувати кілька клієнтів одночасно. Наприклад, Node.js має вбудований механізм під назвою «цикл подій» («event loop»), який обробляє різні операції асинхронно не блокуючи основний потік, що й дозволяє виконувати декілька запитів одночасно.

Функції, які реалізуються на сервері:

- зберігання, доступ і захист даних;
- обробка клієнтського запиту;
- відправлення результату (відповіді) клієнту.

Функції, які реалізуються на стороні клієнта:

- надання користувальницького інтерфейсу;
- формулювання запиту до сервера і його відправлення;
- отримання результатів запиту і відправлення додаткових команд (запитів на додавання, оновлення або видалення даних).

Структура «Клієнт-сервер» архітектури мережевого ринку «TradeLink» має наступний вигляд (рис. 3.2).

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		41

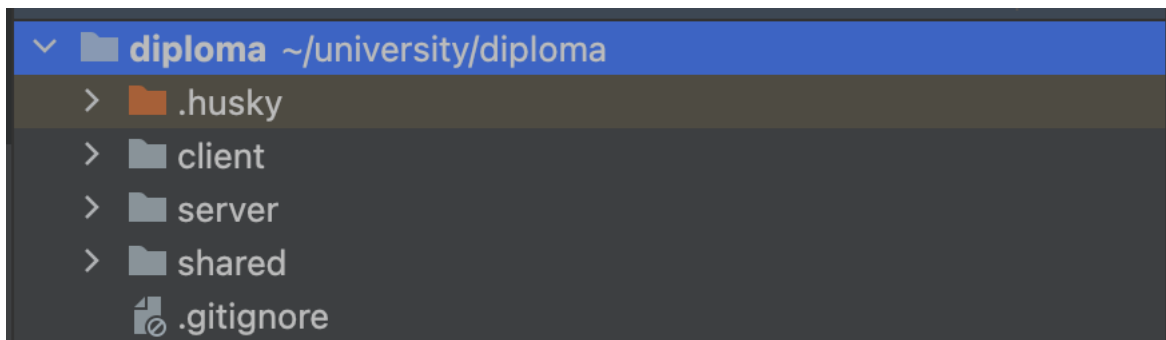


Рисунок 3.2 – Структура «Клієнт-сервер» архітектури мережевого ринку «TradeLink»

Папка «.husky» використовується лише при розробці застосунку й відповідає за виклик sh-скрипта перед комітом в систему контролю версій GIT. Для цього, було створено додатковий файл з назвою «pre-commit», в якому є логіка форматування всіх файлів в проєкті перед виконанням команди «git commit».

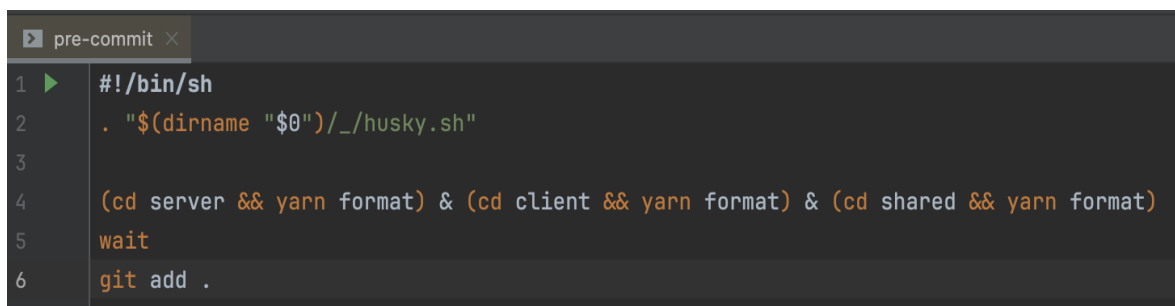


Рисунок 3.3 – Файл «pre-commit»

Папка «client» містить фронтенд-частину застосунку написаного на фреймворці Angular, з використанням допоміжних бібліотек, таких як «rxjs», «primeng», «@ngx-translate» тощо.

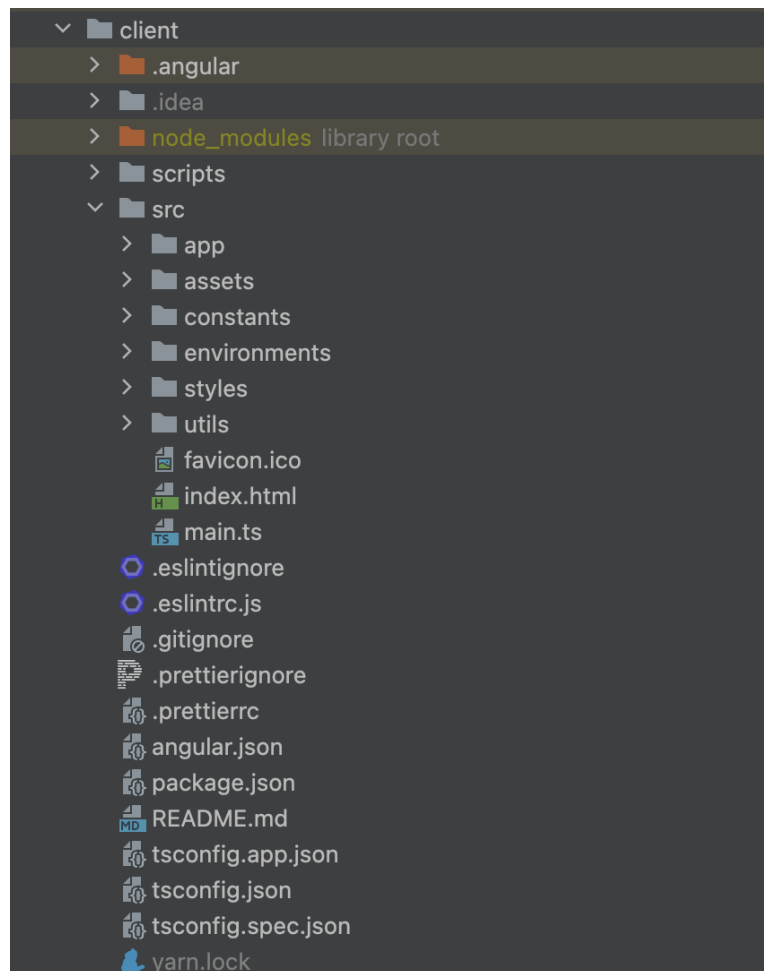


Рисунок 3.4 – Структура фронтенд-частини застосунку

Основні елементи папки «client»:

- папка «scripts» - допоміжні скрипти для розробки, такі як «sync-error-messages.js» та «sync-with-en.js»;
- папка «src» - вихідний код;
- файл «angular.json» - файл конфігурації для фреймворку Angular;
- файл «package.json» - файл конфігурації для застосунків написаних за допомогою Node.js, який містить інформацію про проєкт, зовнішні залежності/модулі та скрипти;
- файл «tsconfig.json» - файл конфігурації для Typescript;

Angular заохочує розробників у використанні модульної структури коду, що знаходиться в папці «app» (рис. 3.5). Клієнтська частина мережевого ринку «TradeLink» має багато модулів, наприклад: «admin», «appeals», «auth»,

«categories» тощо. Кожен з модулів може містити субмодулі, компоненти, сервіси, константи, директиви тощо. Компонент в фреймворці Angular може бути як частиною існуючого модулю, так і автономним («standalone») й поділяється на наступні частини:

- логіка та конфігурація - окремий файл з розширенням «.ts»;
- репрезентація – окремий файл з розширенням «.html» або значення в полі «template» декоратора «Component»;
- стилі - окремий файл з розширенням «.scss» або значення в полі «styles» декоратора «Component»;
- юніт-тести - окремий файл з розширенням «.spec.ts». Цей файл є необов'язковим;

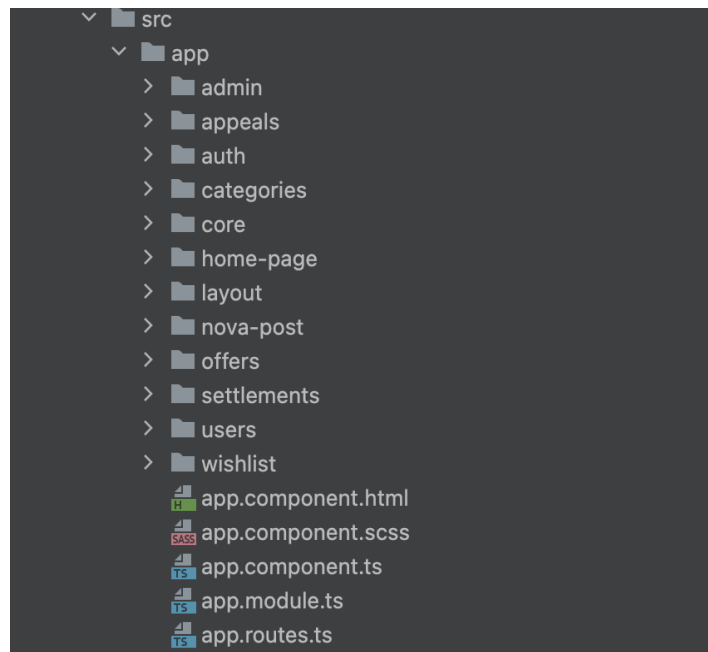


Рисунок 3.5 – Структура папки «app» клієнтської частини мережевого ринку «TradeLink»

Головним компонентом є «AppComponent», що містить логіку ініціалізації дочірніх компонентів (рис. 3.6). Вхідною точкою клієнтської частини застосунку, як і в усіх односторінкових додатків (SPA), є файл «index.html» (рис. 3.7), що знаходиться в папці «src». Головна мета цього

файлу – ініціалізація компонента «AppComponent» за допомогою селектора «app-root».

```

1      <p-toast></p-toast>
2      <p-confirmDialog></p-confirmDialog>
3
4      <ng-container *ngIf="isInitted$ | async; else spinner">
5          <app-header></app-header>
6
7          <app-banners></app-banners>
8
9          <main class="page-content-wrapper">
10             <router-outlet></router-outlet>
11          </main>
12
13          <app-footer></app-footer>
14      </ng-container>
15
16      <ng-template #spinner>
17          <p-progressSpinner class="page-spinner"></p-progressSpinner>
18      </ng-template>
  
```

Рисунок 3.6 – Файл «app.component.html»

```

1      <!doctype html>
2      <html lang="en">
3      <head>
4          <meta charset="utf-8" />
5          <title>Client</title>
6          <base href="/" />
7          <meta name="viewport" content="width=device-width, initial-scale=1" />
8          <link rel="icon" type="image/x-icon" href="favicon.ico" />
9          <link rel="stylesheet" href="light.css" id="app-theme" />
10     </head>
11     <body>
12         <app-root></app-root>
13     </body>
14 </html>
  
```

Рисунок 3.7 – Файл «index.html»

Папка «server» містить бекенд-частину застосунку написаного на фреймворці Nest.js, з використанням допоміжних бібліотек, таких як «sequelize», «sequelize-typescript», «aws-sdk», «passport» тощо.

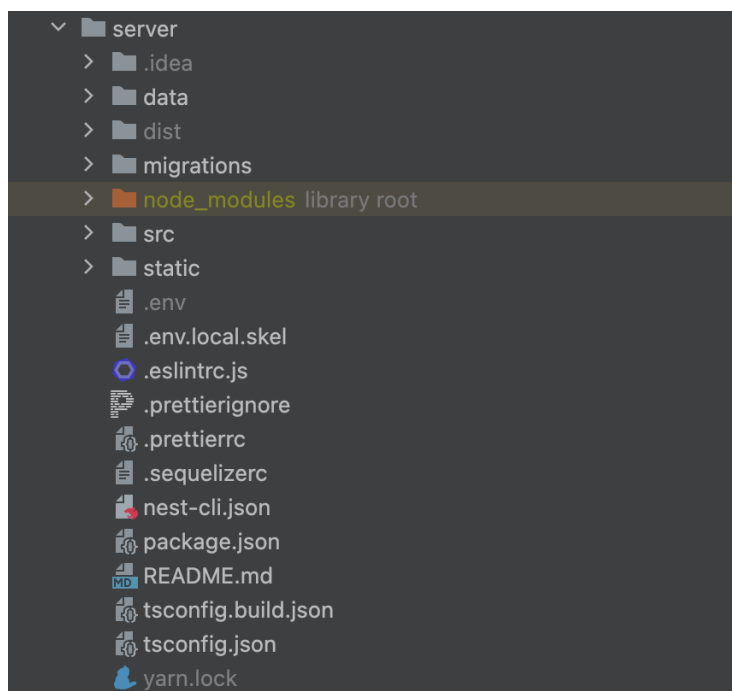


Рисунок 3.8 – Структура бекенд-частини застосунку

Основні елементи папки «server»:

- папка «data» - статичні дані у форматі JSON, які використовуються для наповнення БД;
- папка «migrations» - міграції, що були використані при побудові БД;
- папка «src» - вихідний код;
- файл «.env» - навколишнє середовище, в якому запускається сервер. Цей файл містить параметри підключення до БД, email-серверу, ключі доступу до AWS тощо;
- файл «package.json» - файл конфігурації для застосунків написаних за допомогою Node.js, який містить інформацію про проєкт, зовнішні залежності/модулі та скрипти;
- файл «tsconfig.json» - файл конфігурації для Typescript;

Застосунки побудовані за допомогою фреймворку Nest.js мають схожу структуру з фреймворком Angular. Серверна частина мережевого ринку «TradeLink» має наступні модулі: «app», «auth», «csrf-protection», «database» тощо.

Кожен окремий модуль може містити:

- «об'єкт передачі даних» («Data Transfer Object» / «DTO») – окрема папка «dto», що містить класи з полями, які очікуються при отриманні HTTP-запитів;
- моделі – папка «models» з файлами, які містять типи даних;
- контроллер – файл з розширенням «.controller.ts». Відповідає за HTTP-роутінг, безпечність та виклик відповідних сервісів;
- «конструктор фільтрів» - файл з розширенням «.filter-builder.ts». Цей компонент допомагає обробляти й валідувати інформацію з GET-запиту та отримувати відповідні дані з БД;
- сервіс для валідації - файл з розширенням «.validation.ts»;
- сервіс для обробки файлів - файл з назвою «*-files.service.ts». Відповідає за валідацію та синхронізацію файлів отриманих при HTTP-запиті;
- основний сервіс - файл з розширенням «.service.ts». Кожен метод такого сервісу містить бізнес-логіку для обробки й виконання запиту переданого з контроллера (наприклад створення, оновлення або видалення ресурсу);
- файл-модуль - файл з розширенням «.module.ts»; Цей файл компонує окремі елементи модулю в єдине ціле, що дозволяє іншим компонентам системи їх використовувати;

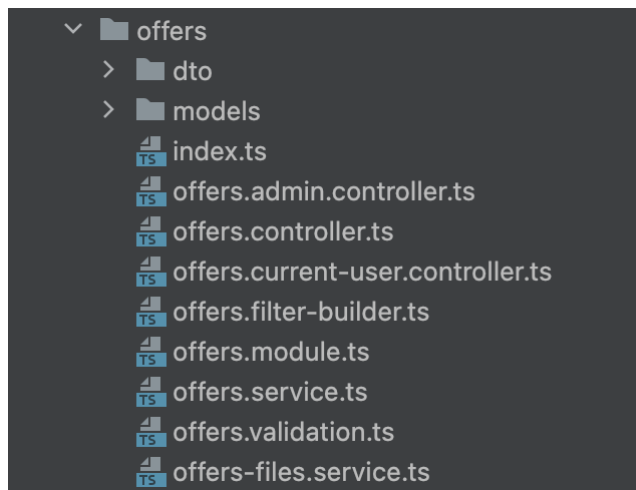


Рисунок 3.9 – Структура модуля «offers»

Вхідною точкою серверної частини застосунку є файл «main.ts» (рис. 3.10), що знаходиться в папці «src». Цей файл відповідає за ініціалізацію кореневого модулю «AppModule», конфігурацію застосунку та запуск HTTP-сервера на відповідному порті.

```

1  import { DocumentBuilder, SwaggerDocumentOptions, SwaggerModule } from "@nestjs/swagger";
2
3  require("dotenv").config();
4  require("dns").setDefaultResultOrder("ipv4first");
5
6  import ...
16
17  const bodyParserOpts: bodyParser.Options = {limit: appConfig.API_BODY_LIMIT...};
20
21  async function bootstrap(): Promise<void> {
22    const app : NestExpressApplication<Server<... = await NestFactory.create<NestExpressApplication>(AppModule);
23
24    app
25      .disable( args: "x-powered-by")
26      .disable( args: "etag")
27      .use(frameguard( options: { action: "sameorigin" } ))
28      .use(cookieParser())
29      .use(requestIp.mw())
30      .use(bodyParser.json(bodyParserOpts))
31      .use(bodyParser.urlencoded( options: { ...bodyParserOpts, extended: true } ))
32      .use(bodyParser.raw(bodyParserOpts));
33
34    app.enableCors({exposedHeaders: "*", Content-Disposition"...});
40
41    const config : Omit<OpenAPIObject, "paths"> = new DocumentBuilder()
42      .setTitle("trade-link-diploma")
43      .setDescription("trade-link-diploma API")
44      .setVersion(appConfig.API_VERSION)
45      .build();
46    const options: SwaggerDocumentOptions = { deepScanRoutes: true };
47    const document : OpenAPIObject = SwaggerModule.createDocument(app, config, options);
48    SwaggerModule.setup( path: "api-docs", app, document);
49
50    await app.listen( port: process.env.PORT || 8080);
51  }
52
53  bootstrap();

```

Рисунок 3.10 – Файл «main.ts»

Кореневий файл «.gitignore» (рис. 3.11) відповідає за перелік елементів, які не повинні потрапляти в систему контролю версій GIT. Це файли та папки з конфіденційними даними (ключі, паролі тощо), логами, скомпільований результат застосунку тощо.

```

1  # compiled output
2  **/dist
3  **/node_modules
4  **/build
5
6  # Logs
7  **/logs
8  **/*.log
9  **/npm-debug.log*
10 **/pnpm-debug.log*
11 **/yarn-debug.log*
12 **/yarn.lock
13 **/yarn-error.log*
14 **/lerna-debug.log*
15
16 # OS
17 **/.DS_Store
18
19 # Tests
20 **/coverage
21 **/.nyc_output
22
23 # IDEs and editors
24 **.idea
25 **.project
26 **.classpath
27 **.c9/
28 **/*.launch
29 **.settings/
30 **/.sublime-workspace
31
32 # dotenv environment variable files
33 **/.env
  
```

Рисунок 3.11 – Файл «.gitignore»

Коренева папка «shared» (рис. 3.12) містить в собі спільний функціонал (рис. 3.13) як для клієнтської частини застосунку, так і для серверної: сервіси, моделі, типи даних, константи, утиліти тощо.

```

  shared
  ├── .idea
  ├── node_modules library root
  └── src
      ├── @types
      ├── auth
      ├── constants
      ├── models
      ├── utils
      ├── .prettierrc
      ├── .prettierrc
      ├── index.ts
      ├── package.json
      ├── tsconfig.json
      └── yarn.lock
  
```

Рисунок 3.12 – Структура папки «shared»

```

1  import { IUser, UserRole } from "../models";
2
3  export abstract class Permissions {
4      private static readonly userRolesHierarchy :UserRole[] = [UserRole.USER, UserRole.ADMIN, UserRole.SUPER_ADMIN];
5
6      static readonly roleAssignmentPermissions: Record<UserRole, UserRole[]> = {
7          [UserRole.USER]: [],
8          [UserRole.ADMIN]: [UserRole.USER],
9          [UserRole.SUPER_ADMIN]: [UserRole.USER, UserRole.ADMIN],
10     };
11
12     static hasPermission(source: IUser | UserRole | undefined, requiredRole: UserRole) :boolean {
13         if (!source || !requiredRole) return false;
14
15         const userRole :UserRole = this.getUserRole(source);
16         return this.userRolesHierarchy.indexOf(userRole) >= this.userRolesHierarchy.indexOf(requiredRole);
17     }
18
19     static canAssignRole(source: IUser | UserRole | undefined, assignmentRole: UserRole) :boolean {
20         if (!source || !assignmentRole) return false;
21
22         const userRole :UserRole = this.getUserRole(source);
23         return (this.roleAssignmentPermissions[userRole] || []).includes(assignmentRole);
24     }
25
26     static hasHigherPermission(source: IUser | UserRole | undefined, targetRole: UserRole) :boolean {
27         if (!source || !targetRole) return false;
28
29         const userRole :UserRole = this.getUserRole(source);
30         return this.userRolesHierarchy.indexOf(userRole) > this.userRolesHierarchy.indexOf(targetRole);
31     }
32
33     private static getUserRole(source: IUser | UserRole) :UserRole {
34         return typeof source === "object" ? source.role : source;
35     }
36 }

```

Рисунок 3.13 – Спільний сервіс «Permissions»

3.2. Розробка основних функцій платформи

Репрезентаційна частина компоненту «OfferFormPageComponent» (рис. 3.14) використовується як для створення нового оголошення, так і для оновлення існуючого. В ній є наступні елементи:

- «app-dropzone» - компонент, що відповідає за отримання, обробку й валідацію файлів від користувача;
- «app-category-select» - випадаючий список з категоріями в системі;
- «app-settlement-select» - випадаючий список з населеними пунктами в системі;
- «app-input» - поле вводу. В цьому компоненті використовується для полів «назва» («title»), «опис» («description») та «ціна» («price»);

- «app-select» - випадаючий список. В цьому компоненті використовується для поля «Типи доставки» («deliveryTypes»);

Поля «Типи доставки» («deliveryTypes») та «ціна» («price») не відображаються на сторінці поки користувач не обере категорію для оголошення, оскільки вона містить в собі необхідну інформацію для цих полів. Наприклад, категорія «Безкоштовно» має всі види доставок (Нова Пошта, Самовивіз, Без доставки) та немає ціни (поле буде неактивне зі значенням «0»), а категорія «Електроніка - Телефони» має лише 2 види доставки (Нова Пошта, Самовивіз) й поле «ціна» є обов'язковим.

```

1 <app-entity-form-page
2   #entityPage
3   title="newOffer"
4   updateTitle="updateOffer"
5   [form]="form"
6   [breadcrumb]="breadcrumb"
7   (formInit)="onFormInit($event)"
8 >
9 <ng-container *ngIf="entityPage.isInitiated" [formGroup]="form">
10   <app-dropzone
11     accept="image/*"
12     [filesValidator]="filesValidator"
13     [form]="form"
14     fieldName="images"
15     [multiple]="true"
16   ></app-dropzone>
17
18   <app-category-select [form]="form" [isOfferSelect]="true"></app-category-select>
19
20   <app-settlement-select></app-settlement-select>
21
22   <app-input fieldName="title"></app-input>
23   <app-input fieldName="description" [isTextarea]="true"></app-input>
24
25   <ng-container *ngIf="!selectedCategory">
26     <app-select
27       fieldName="deliveryTypes"
28       [multi]="true"
29       [enumOptions]="availableDeliveryTypes"
30       enumPrefix="deliveryType"
31     ></app-select>
32
33     <app-input fieldName="price" [isDisabled]="hasNoPrice" keyFilter="money"></app-input>
34   </ng-container>
35 </ng-container>
36 </app-entity-form-page>

```

Рисунок 3.14 – Файл «offer-form-page.component.html»

«EntityFormPageComponent» (в репрезентації – «app-entity-form-page») є батьківським компонентом для багатьох форм у застосунку, в якому міститься основна логіка взаємодії з формами (рис. 3.15): ініціалізація форми, GET-запит

на сервер для отримання існуючого ресурсу (якщо це сторінка редагування), відправка форми та спільна репрезентація форми (рис. 3.16). За допомогою компонентного підходу до розробки, фреймворк Angular дозволяє зберігати спільну логіку, репрезентацію й стилі в окремих компонентах та використовувати їх не повторюючи код двічі або більше разів. Також, компоненти дозволяють декомпонувати складну логіку на окремі елементи, що спрощує читабельність коду та надає можливість ефективного відкладеного завантаження цих елементів за необхідністю, що в свою чергу покращує продуктивність системи в цілому.

```

44 export class EntityFormPageComponent<T extends Entity = any> implements OnInit, OnChanges, OnDestroy {
75   ngOnInit() : void {
76     this.crudService.isItemLoading$
77       .pipe(takeUntilDestroyed(this.destroyRef))
78       .subscribe( observerOrNext: isEntityLoading : boolean => (this.isEntityLoading = isEntityLoading));
79
80     this.crudService.item$.pipe(takeUntilDestroyed(this.destroyRef)).subscribe( observerOrNext: entity : T | undefined => {
81       this.entity = entity;
82       this.error = null;
83     });
84
85     if (this.id) {
86       this.crudService
87         .getOne(this.id)
88         .pipe(
89           takeUntilDestroyed(this.destroyRef),
90           catchError( selector: (error: ApiErrorResponse) => {
91             this.error = ErrorUtils.getErrorMessage(error);
92             this.errorOccurred.emit(error);
93             return EMPTY;
94           },
95         )
96         .subscribe( observerOrNext: entity => this.initForm(entity));
97     } else {
98       this.initForm();
99     }
100   }
101
102   ngOnChanges(changes: SimpleChanges) : void { ... }
103
104   ngOnDestroy() : void { ... }
105
106   onSubmitClick(event: MouseEvent) : void {
107     event.preventDefault();
108
109     if (this.isCustomSubmit) {
110       this.formSubmitted.emit();
111       return;
112     }
113
114     const values = this.form.value;
115     const submit$ : Observable<ApiResult<T>> = this.id
116       ? this.crudService.updateOne(this.id, values, this.entity)

```

Рисунок 3.15 – Фрагмент компонента «EntityFormPageComponent»

```

1 <app-breadcrumb *ngIf="breadcrumb" [breadcrumb]="breadcrumb"></app-breadcrumb>
2
3 <div class="grid h-full align-items-center justify-content-center">
4   <div class="col-12 md:col-8">
5     <p-card [header]="actualTitle ? (actualTitle | translate) : undefined">
6       <form [formGroup]="form || emptyForm" [appForm]="form" [checkDirty]="checkDirty" #authForm="appForm">
7         <p-message
8           *ngIf="error"
9           severity="error"
10          [text]="error | translate"
11          styleClass="w-full justify-content-start mb-2"
12        ></p-message>
13
14        <p-progressSpinner *ngIf="isLoading" class="flex"></p-progressSpinner>
15
16        <ng-container *ngIf="!isLoading && isInitiated">
17          <ng-content></ng-content>
18
19          <p-button
20            *ngIf="!form.disabled"
21            [label]='submit' | translate
22            [disabled]="(authForm.isSubmitDisabled$ | async)!"
23            (onClick)="onSubmitClick($event)"
24          ></p-button>
25        </ng-container>
26      </form>
27    </p-card>
28  </div>
29 </div>

```

Рисунок 3.16 – Файл «entity-form-page.component.html»

Щоб створити нове оголошення на сервері, авторизованому користувачу потрібно виконати відповідний HTTP-запит – «POST /v1/current/offers» разом із даними форми в полі «body», який потрапить до методу «addOne» в контролері «OffersCurrentUserController».

Декоратор «@Controller(«v1/current/offers»)» відповідає за те, щоб фреймворк Nest.js розумів, що цей клас є контролером для всіх HTTP-запитів, які надіслані на відповідну точку на сервері («endpoint»), а «@UseGuards(UserAuthGuard)» в свою чергу забезпечить виконання запитів лише від авторизованого користувача. Так як ці декоратори знаходяться на рівні контролеру, то вони застосовуються для всіх методів цього класу. Далі запит потрапить до методу «addOne», що обробить й отримає всі необхідні дані за допомогою декораторів «@UseInterceptors», «@Body»,

«@UploadedFiles», «@InjectUser» та викличе відповідний метод в сервісі «OffersService».

```

offers.current-user.controller.ts
26 @Controller( prefix: "v1/current/offers")
27 @UseGuards(UserAuthGuard)
28 export class OffersCurrentUserController {
29     constructor(private readonly offersService: OffersService) {}
30
31     @Get()
32     async getAll(@Query() filter: OfferFilterDto, @InjectUser() currentUser: User) : Promise<Offer[]> {
33         return this.offersService.getAll(filter, currentUser);
34     }
35
36     @Get( path: ":id")
37     async getOne(@Param() { id }: IdParams, @Query() filter: OfferFilterDto, @InjectUser() currentUser: User) : Promise<Offer> {
38         return this.offersService.getOne(id, filter, currentUser);
39     }
40
41     @Post()
42     @UseInterceptors(FilesInterceptor( fieldName: "images"), FormDataBodyInterceptor)
43     @UseGuards(UserNotRestrictedGuard)
44     async addOne(
45         @Body() body: AddOfferDto,
46         @UploadedFiles() images: Express.Multer.File[],
47         @InjectUser() currentUser: User,
48     ) : Promise<result: Offer> {
49         const result : Offer = await this.offersService.addOne(body, images, currentUser);
50         return { result };
51     }
52 }

```

Рисунок 3.17 – Метод «addOne» в контролері «OffersCurrentUserController»

«OffersService» робить необхідну валідацію за допомогою сервісів «OffersValidation» та «OffersFilesService», створює нове оголошення в БД за допомогою методу «create» в sequelize-моделі «Offer», завантажує зображення (якщо є) на сервіс «Amazon S3» використовуючи метод «syncImages» в сервісі «OffersFilesService» та повертає створене оголошення (рис 3.18).

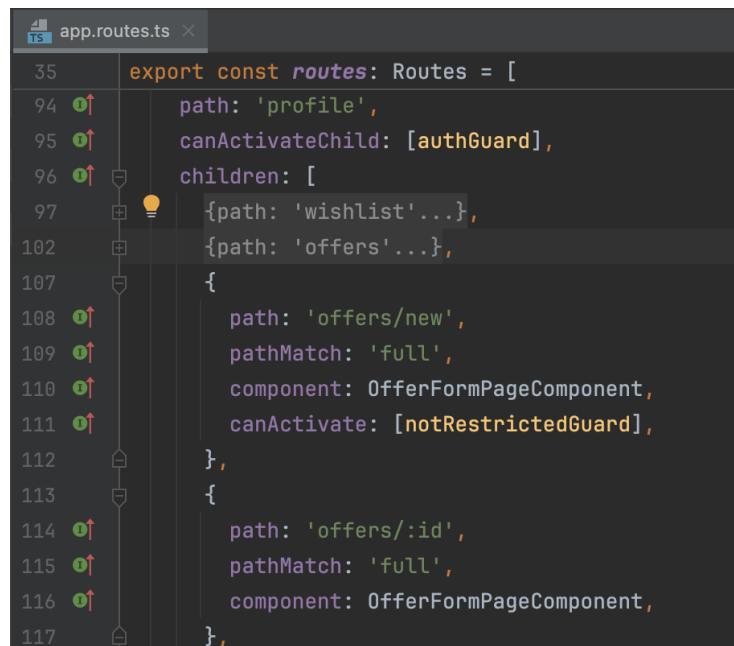
```

110 async addOne(body: AddOfferDto, images: Express.Multer.File[], currentUser: UserModel) : Promise<OfferModel> {
111     await this.offersValidation.validateSubmit(body, { currentUser, instance: { currentUser } });
112     this.offersFilesService.validate(images);
113
114     const instance : OfferModel = await this.Offer.create( values: { ...body, ownerId: currentUser.id });
115     await this.offersFilesService.syncImages(instance, images);
116
117     return instance;
118 }

```

Рисунок 3.18 – Метод «addOne» в сервісі «OffersService»

Оновлення існуючого оголошення схоже на створення нового. В клієнтській частині застосунку різниця в тому, що відповідний компонент «OfferFormPageComponent» для нового оголошення зареєстрований в файлі «app.routes.ts» за шляхом «/profile/offers/new», тоді як оновлення – за шляхом «/profile/offers/:id», де замість «:id» треба зазначити ідентифікатор відповідного оголошення (рис. 3.19).



```

35 export const routes: Routes = [
94   path: 'profile',
95   canActivateChild: [authGuard],
96   children: [
97     {path: 'wishlist'...},
102    {path: 'offers'...},
107    {
108      path: 'offers/new',
109      pathMatch: 'full',
110      component: OfferFormPageComponent,
111      canActivate: [notRestrictedGuard],
112    },
113    {
114      path: 'offers/:id',
115      pathMatch: 'full',
116      component: OfferFormPageComponent,
117    },

```

Рисунок 3.19 – Реєстрація сторінок для створення й оновлення оголошення в файлі «app.routes.ts»

Для оновлення оголошення на сервері, авторизованому користувачу потрібно виконати відповідний HTTP-запит – «PUT /v1/current/offers/:id» разом із даними форми в полі «body», де замість «:id» треба зазначити ідентифікатор відповідного оголошення. Цей запит потрапить до методу «updateOne» в контролері «OffersCurrentUserController» (рис 3.20). Він аналогічний до методу «addOne» цього ж класу, але з додатковим параметром «@Param() { id }: IdParams», що відповідає за отримання ідентифікатора оголошення з запиту.

```

offers.current-user.controller.ts
28 export class OffersCurrentUserController {
58   @Put( path: ":id")
59   @UseInterceptors(FilesInterceptor( fieldName: "images"), FormDataBodyInterceptor)
60   async updateOne(
61     @Param() { id }: IdParams,
62     @Body() body: UpdateOfferDto,
63     @UploadedFiles() images: Express.Multer.File[],
64     @InjectUser() currentUser: User,
65   ) : Promise<...> {
66     const result : Offer = await this.offersService.updateOne(id, body, images, currentUser);
67     return { result };
68   }

```

Рисунок 3.20 – Метод «updateOne» в контролері
«OffersCurrentUserController»

«OffersService» виконує пошук (рис 3.21) оголошення по отриманому ідентифікатору за допомогою приватного методу «findOne» (рис 3.22), який завершиться контрольованою помилкою «NotFoundException» зі значенням константи «ErrorMsg.OFFER_NOT_FOUND», якщо оголошення не існує в БД. Якщо оголошення існує, то продовжує валідацію за допомогою сервісів «OffersValidation» та «OffersFilesService», оновлює оголошення в БД за допомогою методу «update» в екземплярі sequelize-моделі «Offer», синхронізує зображення з сервісом «Amazon S3» використовуючи метод «syncImages» в сервісі «OffersFilesService» та повертає оновлене оголошення.

```

offers.service.ts
21 export class OffersService {
107
118   async addOne(body: AddOfferDto, images: Express.Multer.File[], currentUser: UserModel) : Promise<Offer> {
119
120     async updateOne(
121       id: Id,
122       { filesMeta, ...body }: UpdateOfferDto,
123       images: Express.Multer.File[] | undefined,
124       currentUser: UserModel,
125     ) : Promise<Offer> {
126       const instance : OfferModel = await this.findOne(id, currentUser);
127       await this.offersValidation.validateSubmit(body, {currentUser, instance}: { instance, currentUser });
128       if (images) this.offersFilesService.validate(images, instance, filesMeta);
129
130       await Promise.all( values: [
131         instance.update(body),
132         ...(images?.length || filesMeta?.removed?.length
133           ? [this.offersFilesService.syncImages(instance, images, filesMeta)]
134           : []),
135       ]);
136
137       return instance;
138     }

```

Рисунок 3.21 – Метод «updateOne» в сервісі «OffersService»

```
private async findOne(id: Id, currentUser?: UserModel) : Promise<OfferModel> {
  const instance : OfferModel = await this.Offer.findOne( options: { where: { id, ...(currentUser ? { ownerId: currentUser.id } : {}) } });
  if (!instance) throw new NotFoundException(ErrMsg.OFFER_NOT_FOUND);

  return instance;
}
```

Рисунок 3.22 – Приватний метод «findOne» в сервісі «OffersService»

3.3. Опис інструкції користувача

При відкритті сайту користувач потрапляє на головну сторінку де відразу може побачити розміщені оголошення та розділи.

Головна сторінка мережевого ринку «TradeLink» виконана у темних тонах із синіми акцентами (рис. 3.23). У верхній частині розташовані логотип та назва «TradeLink», перемикач між світлою та темною темами інтерфейсу, опція вибору мови (англійська та українська), кнопка «Профіль» для доступу до профілю користувача та кнопка «Створити оголошення» для створення нового оголошення. Якщо користувач не авторизований, то кнопки «Профіль» та «Створити оголошення» перенаправлять його на сторінку авторизації, де також є опції реєстрації для нових користувачів та відновлення паролю. Під верхнім меню знаходиться пошуковий блок, що включає поле для тексту з можливістю вибору населеного пункту та кнопку «Пошук». Основний контент сторінки поділений на розділи оголошень, які представлені у вигляді іконок з підписами (транспорт, нерухомість, безкоштовно, благодійність, робота, електроніка, різне), та список останніх оголошень. При натисканні на блок з розділом, відобразиться меню з підкатегоріями та посиланням «Переглянути всі оголошення в цьому розділі». Кожне оголошення містить зображення (або позначку «NO IMAGE AVAILABLE»), заголовок, населений пункт, дату публікації та ціну (або маркування «Безкоштовно», якщо послуга або товар є безкоштовними).

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		57

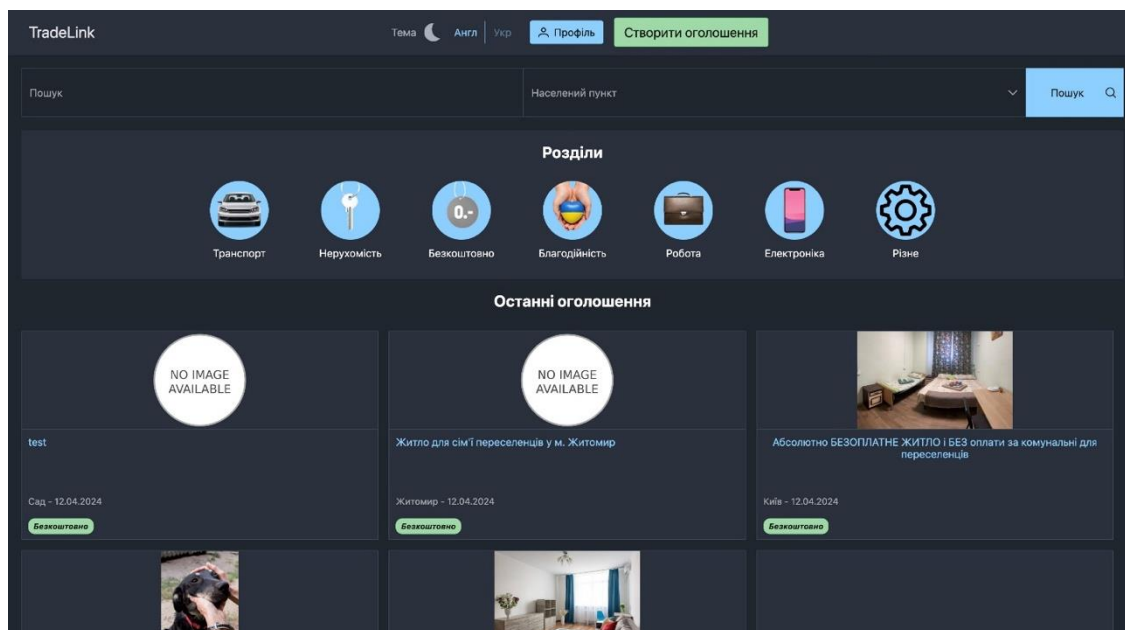


Рисунок 3.23 – Головна сторінка онлайн платформи дошки оголошень

Вгорі сторінки оголошення знаходяться навігаційні елементи, такі як тема, мова, профіль користувача та кнопка для створення оголошення. В оголошенні представлено зображення товару або послуги, поруч із яким вказана дата розміщення оголошення, назва, ціна і кнопки для додавання до улюбленого, відображення номера телефону власника оголошення, кнопка «Прийняти» та коротка інформація про власника оголошення з посиланням на інші його оголошення (рис. 3.24).

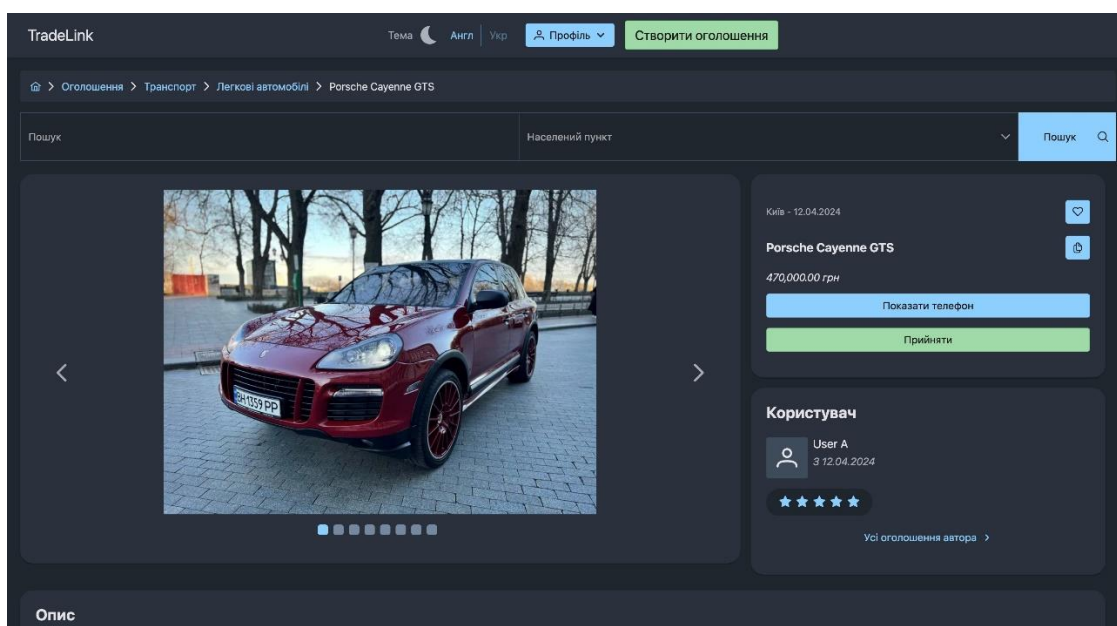


Рисунок 3.24 – Сторінка оголошень онлайн платформи дошки оголошень

Далі йде детальний опис оголошення, його ідентифікатор та кнопка «Поскаржитися». Під описом є розділ коментарів, де користувачі можуть задавати питання та залишати відгуки, а також можливість відповіді на інші коментарі (рис. 3.25).

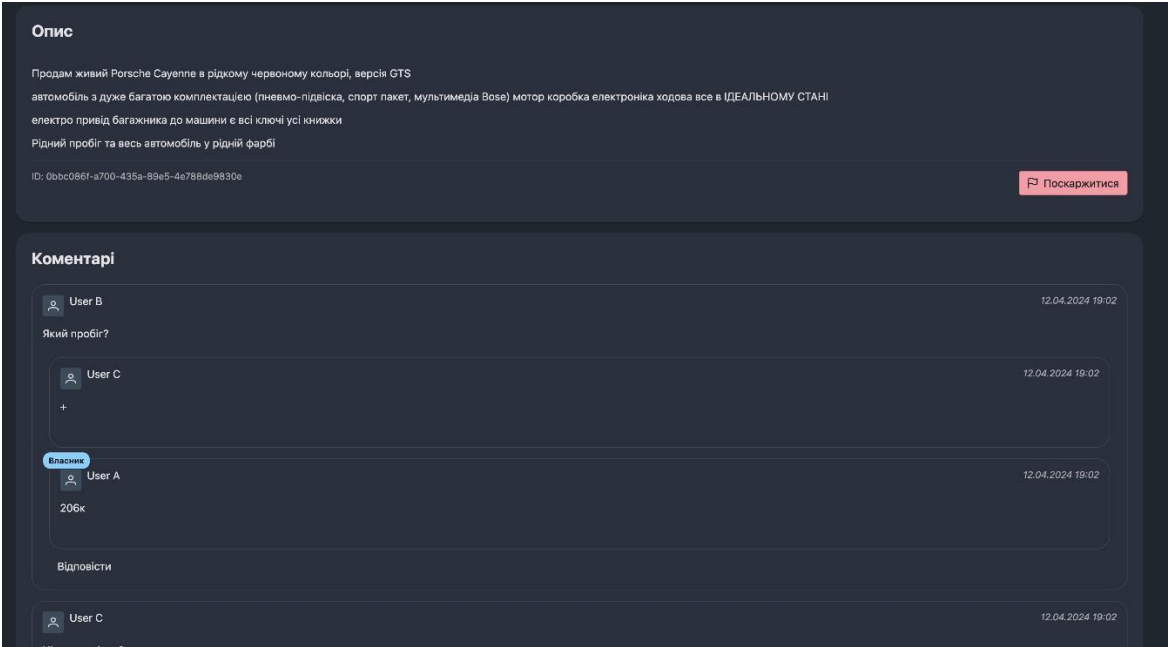


Рисунок 3.25 – Продовження сторінки оголошень онлайн платформи

Верхня частина сторінки створення оголошення (рис. 3.26) містить назву платформи, можливість перемикавання між світлою та темною темами, вибір мови інтерфейсу (українська або англійська), кнопку для доступу до профілю користувача та кнопку «Створити оголошення» для підтвердження створення нового оголошення. Нижче розміщений шлях навігації, що показує поточне місцезнаходження користувача на сайті: «Оголошення > Оголошення». Основна частина сторінки містить форму для створення нового оголошення. Форма включає область для завантаження зображень, випадаючий список для вибору розділу/категорії оголошення, випадаючий список для вибору населеного пункту, текстове поле для введення назви оголошення, текстове поле для введення опису оголошення, випадаючий список типів доставки (Нова Пошта, Самовивіз, Без доставки) та поле для введення ціни в гривнях. У

нижній частині форми знаходиться кнопка «Відправити» для підтвердження та публікації оголошення.

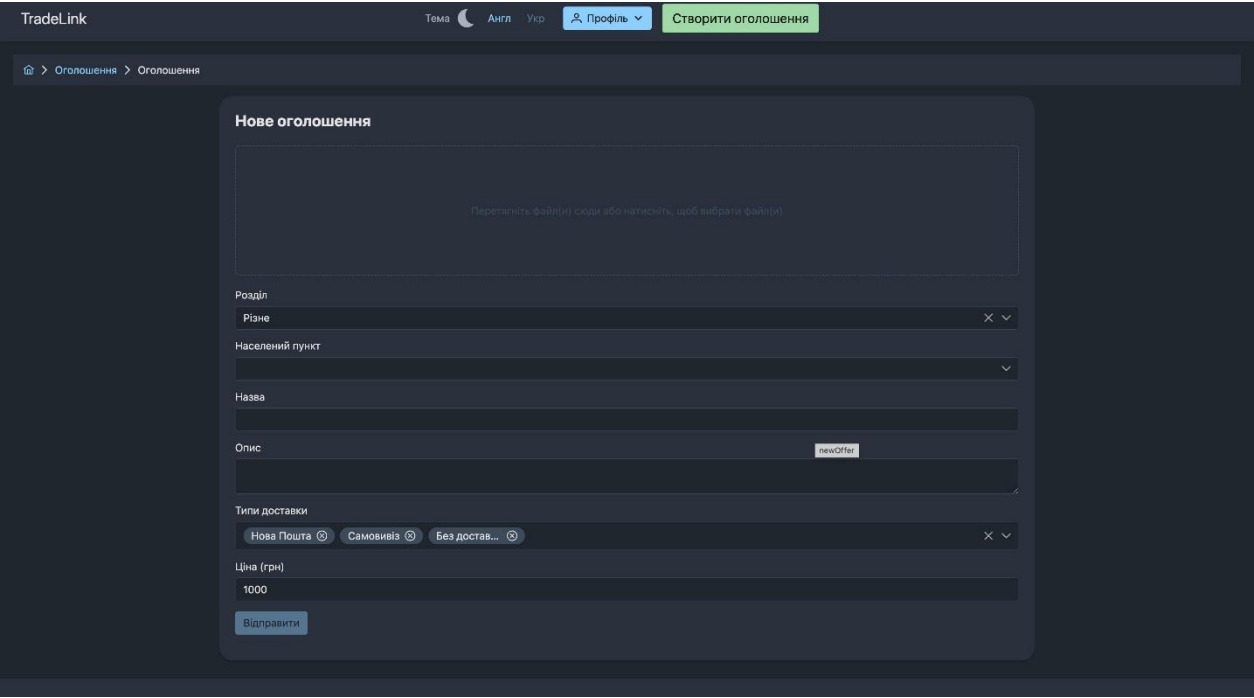


Рисунок 3.26 – Сторінки створення нового оголошення

У верхній частині сторінки «Мої оголошення» (рис. 3.27) знаходяться такі елементи: назва платформи «TradeLink», можливість перемикання між світлою і темною темами, вибір мови інтерфейсу (українська або англійська), кнопка для доступу до профілю користувача, де випадає меню з опціями: «Профіль», «Мої оголошення», «Список бажань», «Мої звернення» і «Вихід», а також кнопка «Створити оголошення» для створення нового оголошення.

Основна частина сторінки містить дві вкладки: «Прийняті оголошення» і «Мої оголошення», з яких активною є вкладка «Мої оголошення». Під вкладками розташований список оголошень користувача, кожне з яких містить зображення, позначку про стан/статус оголошення (наприклад, «Активне»), назву (наприклад, «Bmw e39»), місцезнаходження (наприклад, «Київ») і дату створення оголошення (наприклад, «12.04.2024»). Також відображається ціна оголошення у гривнях (або маркування «Безкоштовно», якщо послуга або товар є безкоштовними) і дві кнопки: «Редагувати» і «Видалити» для

редагування або видалення відповідного оголошення (якщо оголошення активне).

У нижній частині сторінки є елементи навігації для перегортання сторінок зі списком оголошень, що включають кнопки для переходу до першої, попередньої, наступної, останньої сторінки і випадаючий список для вибору кількості відображених оголошень (рис. 3.27).

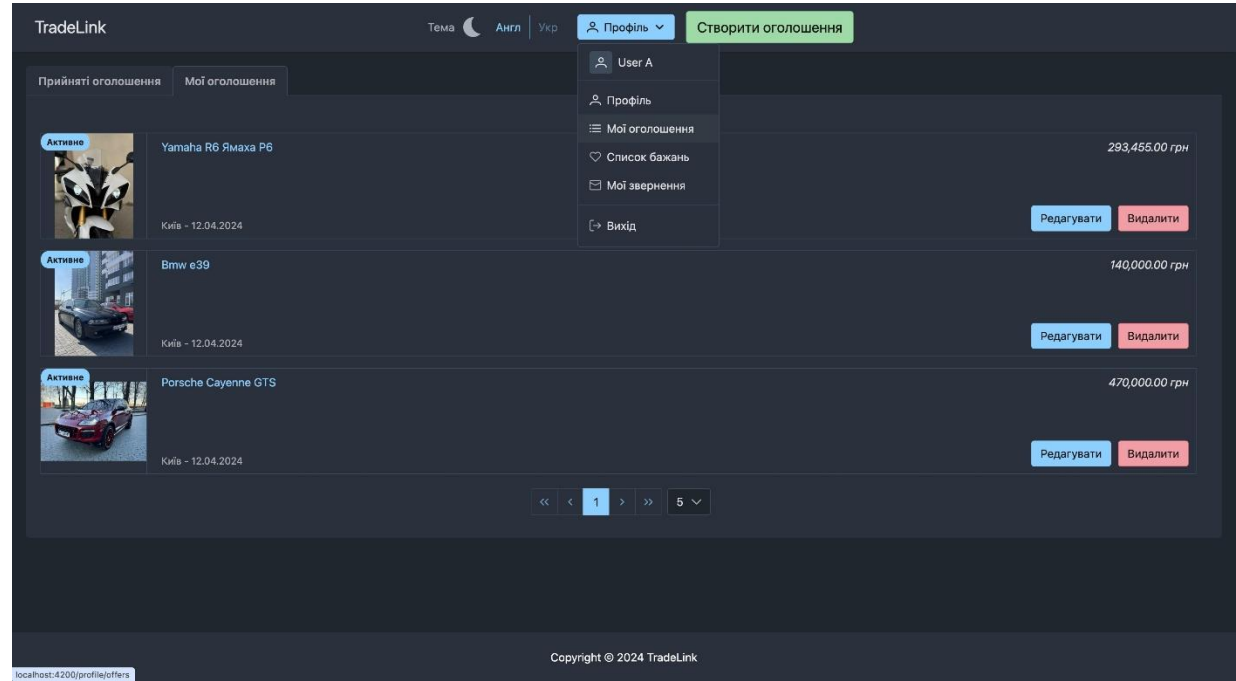


Рисунок 3.27 – Сторінки створених оголошень

Сторінка «Профіль» (рис. 3.28) поділена на дві вкладки: «Профіль» і «Зміна пароля», з яких активною є вкладка «Профіль». Центральна частина сторінки містить форму для редагування профілю. У верхній частині форми розташована область для завантаження аватарки користувача із текстом «Перетягніть файл(и) сюди або натисніть, щоб вибрати файл(и)».

Нижче розміщені текстові поля для введення та редагування наступної інформації:

- Пошта: поле для введення електронної пошти (зараз вказано «admin@gmail.com»).
- Ім'я: поле для введення імені (зараз вказано «Admin»).

- Номер телефону: поле для введення номеру телефону з випадаючим списком для вибору коду країни (зараз вибрано Україну і номер «+380(99)999-99-91»).
- Населений пункт: випадаючий список для вибору населеного пункту.

Рисунок 3.28 – Сторінка редагування даних

3.4. Тестування роботи програмного продукту

Тестування сайту є критично важливим етапом у процесі розробки веб-ресурсу, який забезпечує його якість, стабільність і відповідність вимогам користувачів. Метою цього розділу є ознайомлення з основними принципами, методами і типами тестування, які використовуються для перевірки функціональності, безпеки, продуктивності та зручності використання сайту.

У сучасному світі, де веб-сайти відіграють ключову роль у бізнесі, освіті, розвагах і комунікаціях, якісне тестування стає невід'ємною частиною процесу розробки. Воно допомагає виявити та усунути помилки до того, як вони

вплинуть на кінцевих користувачів, а також забезпечити відповідність сайту технічним і бізнес-вимогам.

Тест-кейси є детальними сценаріями, які використовуються для перевірки функціональності веб-сайту. Вони описують кроки, необхідні для виконання конкретного тесту, очікувані результати і умови, за яких тест вважається успішним [14].

Таблиця 3.1

Test Cases

Test Case Description	Test Steps	Test Data	Expected Results	Actual Results	Pass / Fail
Перевірити можливість реєстрації нового користувача.	1. Перейти на сторінку реєстрації. 2. Заповнити всі обов'язкові поля (ім'я, електронна пошта, пароль, підтвердження пароля). 3. Натиснути кнопку «Зареєструватися».	Email = admin@gmail.com Пароль = 12345678	Користувач успішно реєструється, бачить повідомлення про успішну реєстрацію й отримує посилання для верифікації акаунту на електронну пошту.	Як і очікувалося	Pass
Перевірити можливість входу зареєстрованого користувача.	1. Перейти на сторінку входу. 2. Ввести правильну електронну пошту та пароль. 3. Натиснути кнопку «Увійти».	Email = admin@gmail.com Пароль = 12345678	Користувач успішно входить у систему і перенаправляється на головну сторінку.	Як і очікувалося	Pass

Перевірити можливість створення нового оголошення.	1. Натиснути на кнопку «Створити оголошення». 2. Заповнити всі обов'язкові поля (категорія, населений пункт, назва, опис, типи доставки, ціна). 3. Завантажити зображення. 4. Натиснути кнопку «Відправити».	Категорія = Транспорт – Легкові автомобілі Населений пункт = Київ Назва = Тест назва Опис = Тест опис Типи доставки = Без доставки Ціна = 100000	Оголошення успішно створено і відображається у списку оголошень користувача.	Як і очікувалося	Pass
Перевірити можливість редагування існуючого оголошення.	1. Перейти до розділу «Мої оголошення». 2. Натиснути кнопку «Редагувати» біля одного з оголошень. 3. Внести зміни в поля оголошення. 4. Натиснути кнопку «Зберегти».	Категорія = Транспорт – Легкові автомобілі Населений пункт = Київ Назва = Тест назва 2 Опис = Тест опис 2 Типи доставки = Без доставки Ціна = 101000	Оголошення успішно відредаговано і відображає нові дані.	Як і очікувалося	Pass
Перевірити можливість	1. Перейти до розділу «Мої оголошення».	Користувач повинен мати принаймні	Оголошення успішно видалено зі	Як і очікувалося	Pass

видалення оголошення.	2. Натиснути кнопку «Видалити» біля одного з оголошень. 3. Підтвердити видалення у спливаючому вікні.	одне створене оголошення.	списку оголошень користувача.		
Перевірити можливість виходу з облікового запису.	1. Навести курсор на кнопку профілю. 2. Обрати опцію «Вихід» з випадаючого меню.	Користувач повинен бути зареєстрований і авторизований в системі.	Користувач успішно виходить із системи і перенаправляє на головну сторінку, якщо він знаходиться на сторінці для лише авторизованих користувачів.	Як і очікувалося	Pass
Перевірити можливість зміни пароля користувача.	1. Перейти до розділу «Профіль». 2. Натиснути на вкладку «Зміна пароля». 3. Заповнити всі обов'язкові поля (Попередній пароль, Пароль, Підтвердіть пароль).	Користувач повинен бути зареєстрований і авторизований в системі. Попередній пароль = 12345678 Пароль = 123456789	Пароль успішно змінено, користувач отримує повідомлення на сторінці про успішну зміну пароля.	Як і очікувалося	Pass

	4. Натиснути кнопку «Зберегти».	Підтвердіть пароль = 123456789			
--	---------------------------------	--------------------------------	--	--	--

Висновки до третього розділу

У цьому розділі детально розглянуто інтерфейс та порядок роботи з системою додатку, зосереджуючись на ключових аспектах його встановлення, налаштування та експлуатації. Система побудована на трирівневій архітектурі «клієнт-сервер», що забезпечує чіткий поділ між клієнтською частиною, сервером та базою даних, дозволяючи ефективно керувати ресурсами та забезпечувати міжплатформенність веб-додатків.

Користувацький інтерфейс додатку є інтуїтивно зрозумілим і надає можливість перемикання між темною та світлою темами, вибору мови інтерфейсу (українська або англійська), а також доступу до профілю користувача. Функціонал створення оголошень включає заповнення форми з обов'язковими полями, завантаження зображень, вибір категорій та типів доставки.

Також був проведений детальний опис тест-кейсів для перевірки основних функцій веб-додатку, таких як реєстрація нового користувача, вхід у систему, створення, редагування та видалення оголошень, зміна пароля. Кожен тест-кейс містить конкретні кроки, необхідні дані та/або умови, очікувані результати та фактичні результати, що дозволяє забезпечити якість та стабільність роботи сайту.

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці веб-додатку для мережевого ринку «TradeLink». У ході дослідження була виконана комплексна робота, яка включала аналіз існуючих рішень, проектування системи, розробку користувацького інтерфейсу, а також тестування програмного забезпечення.

Проектування бази даних і алгоритмів роботи системи здійснювалося з урахуванням вимог до гнучкості, масштабованості та безпеки. Створений додаток використовує трирівневу архітектуру «клієнт-сервер», що забезпечує чітке розмежування між клієнтською частиною, сервером та базою даних. Це дозволяє ефективно керувати ресурсами та забезпечувати стабільну роботу системи в умовах високого навантаження.

Користувацький інтерфейс додатку є інтуїтивно зрозумілим, підтримує перемикання між темною та світлою темами, вибір мови інтерфейсу (українська або англійська), а також надає можливість керувати профілем користувача. Основні функції, такі як реєстрація, створення, редагування та видалення оголошень, зміна пароля були детально протестовані за допомогою спеціально розроблених тест-кейсів. Це забезпечує високу якість та стабільність роботи веб-додатку.

В результаті роботи була розроблена система, яка відповідає сучасним вимогам ринку і надає зручний інструмент для користувачів щодо розміщення та перегляду оголошень. Розробка веб-додатку «TradeLink» підтверджує необхідність переходу бізнесу до онлайн-платформ, що сприяє залученню нових клієнтів та спрощенню взаємодії з існуючими.

Загалом, виконана робота демонструє високий рівень технічної підготовки, вміння аналізувати та розробляти програмні рішення, що задовольняють актуальні потреби користувачів та ринку в цілому.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		67

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дошка оголошень в Україні klubok. [Електронний ресурс] – Режим доступу до ресурсу: https://klubok.com/ua?gclid=CjwKCAjwx-CyBhAqEiwAeOcTddSMV-YqsTbiuG3EAxe5tZZHjj7A394JuRuiQiqH5YDPizl-4uL89BoCLy8QAvD_BwE.
2. Про розробку додатків на Nest JS. [Електронний ресурс] – Режим доступу до ресурсу: <https://foxminded.ua/nest-js/>.
3. Розділ 2. Основи UML. [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.kde.org/trunk5/uk/umbrello/umbrello/uml-basics.html>.
4. Що таке Angular? [Електронний ресурс] – Режим доступу до ресурсу: <https://angular.tutorial.in.ua/what-is-angular/>.
5. Що таке Node.js та NPM. [Електронний ресурс] – Режим доступу до ресурсу: <https://angular.tutorial.in.ua/what-are-nodejs-and-npm/>.
6. Як будувати UML-діаграми. [Електронний ресурс] – Режим доступу до ресурсу: <https://dou.ua/forums/topic/40575/>.
7. Addy Osmani. Learning JavaScript Design Patterns / Addy Osmani., 2012. – 254 с.
8. A Step-by-Step Guide to Implement JWT Authentication in NestJS using Passport. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.sipios.com/blog-posts/implementing-authentication-in-nestjs-using-passport-and-jwt>.
9. Authentication and Authorization in Node.js: A Comprehensive Guide. [Електронний ресурс] – Режим доступу до ресурсу: <https://codewithpawan.medium.com/authentication-and-authorization-in-node-js-a-comprehensive-guide-2755b57dce27>.
10. AWS S3 Tutorial: Deep Dive into Amazon Simple Storage Service. [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/edureka/aws-s3-tutorial-d883e5cc2cf5>.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		68

11. Basic Authentication Versus CSRF. [Електронний ресурс] – Режим доступу до ресурсу: <https://trustedsec.com/blog/basic-authentication-versus-csrf>.
12. Building Secure User Registration and Authentication in Node.js. [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.to/aneeqakhan/building-secure-user-authentication-in-nodejs-4cjj>.
13. Cross-site request forgery (CSRF). [Електронний ресурс] – Режим доступу до ресурсу: <https://portswigger.net/web-security/csrf>.
14. How to Write Test Cases: A Step-by-Step QA Guide. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.coursera.org/articles/how-to-write-test-cases>.
15. Implementing Polymorphism in Sequelize TypeScript: Handling Likes, Images, and Video Posts. [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@dineshbyte/implementing-polymorphism-in-sequelize-typescript-handling-likes-images-and-video-posts-3a9bd8e1e231>.
16. Intro to PostgreSQL. [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/codex/intro-to-postgresql-c8da31335c34>.
17. Introduction to PrimeNG: A Rich UI Component Library for Angular. [Електронний ресурс] – Режим доступу до ресурсу: <https://dalenguyen.medium.com/introduction-to-primeng-a-rich-ui-component-library-for-angular-6b15a1e26202>
18. Kyle Simpson. You don't Know JS: ES6 & Beyond / Kyle Simpson., 2015. – 278 с.
19. Kyle Simpson. You dont Know JS: Async & Performance / Kyle Simpson., 2015. – 296 с.
20. Most useful methods from Moment.js. [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@wlodarczyk_j/most-useful-methods-from-moment-js-8b8ff6f6b04d.

21. Multilingual Angular App using @ngx-translate. [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@hujjainwala/multilingual-angular-app-using-ngx-translate-a1087cd07a50>.
22. NestJS – Mailer. [Електронний ресурс] – Режим доступу до ресурсу: <https://nest-modules.github.io/mailer/docs/mailer.html>.
23. Node.js Database Magic: Exploring the Powers of Sequelize ORM. [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@rishu_2701/node-js-database-magic-exploring-the-powers-of-sequelize-orm-a22a521d9d9d/.
24. OLX. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.olx.ua/uk/>.
25. Responsive Design in Angular. [Електронний ресурс] – Режим доступу до ресурсу: <https://bapittu.medium.com/responsive-design-in-angular-48ef1f9db857>.
26. RxJS for Beginner Angular Developers- Part 1: Introduction. [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/ngconf/rxjs-for-beginner-angular-developers-introduction-2d26ffa364af>.
27. SHAFa. [Електронний ресурс] – Режим доступу до ресурсу: https://shafa.ua/?utm_source=google&utm_medium=cpc&utm_campaign=PO_Performance_Max-general-roas&gad_source=1&gclid=CjwKCAjwx-CyBhAqEiwAeOcTdeuP3R87mUZ9oII4kqVKYjm0_WwtncrgAWJk7RsVTPaxBVccW7FIIRoCRIIQAyD_BwE.
28. Theming with PrimeNG CSS Variables. [Електронний ресурс] – Режим доступу до ресурсу: <https://yigitfindikli.medium.com/theming-with-primeng-css-variables-d16ad72ef87c>.
29. What is Amazon S3? [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html>.
30. What is Lodash? [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@saba.205102/what-is-lodash-d4f960c681e>.

ДОДАТКИ

Контроллер «OffersCurrentUserController»

```
import {
  Body,
  Controller,
  Delete,
  Get,
  HttpStatusCode,
  HttpStatus,
  Param,
  Post,
  Put,
  Query,
  UploadedFiles,
  UseGuards,
  UseInterceptors,
} from "@nestjs/common";
import { OffersService } from "../offers.service";
import { UserAuthGuard, UserNotRestrictedGuard } from "@auth";
import { InjectUser } from "@core/decorators";
import { User } from "@database";
import { FilesInterceptor } from "@nestjs/platform-express";
import { FormDataBodyInterceptor } from "@core/interceptors";
import { AddOfferDto, OfferFilterDto, UpdateOfferDto } from "../dto";
import { AcceptOfferDto } from "../dto/accept-offer.dto";
import { IdParams } from "@core/dto";

@Controller("v1/current/offers")
@UseGuards(UserAuthGuard)
export class OffersCurrentUserController {
  constructor(private readonly offersService: OffersService) {}

  @Get()
  async getAll(@Query() filter: OfferFilterDto, @InjectUser() currentUser: User) {
    const { result, count } = await this.offersService.getAll(
      this.offersService.createFilterBuilder({ filter }).byUser(currentUser.id),
    );

    return { result, count };
  }

  @Get(":id")
  async getOne(@Param() { id }: IdParams, @Query() filter: OfferFilterDto, @InjectUser()
currentUser: User) {
    const result = await this.offersService.getOne(id, filter);
    return { result };
  }
}
```

```

@Post()
@UseInterceptors(FilesInterceptor("images"), FormDataBodyInterceptor)
@UseGuards(UserNotRestrictedGuard)
async addOne(
  @Body() body: AddOfferDto,
  @UploadedFiles() images: Express.Multer.File[],
  @InjectUser() currentUser: User,
) {
  const result = await this.offersService.addOne(body, images, currentUser);
return { result };
}

@Put(":id")
@UseInterceptors(FilesInterceptor("images"), FormDataBodyInterceptor)
async updateOne(
  @Param() { id }: IdParams,
  @Body() body: UpdateOfferDto,
  @UploadedFiles() images: Express.Multer.File[],
  @InjectUser() currentUser: User,
) {
  const result = await this.offersService.updateOne(id, body, images, currentUser);
  return { result };
}

@Delete(":id")
@HttpCode(HttpStatus.NO_CONTENT)
async deleteOne(@Param() { id }: IdParams, @InjectUser() currentUser: User) {
  await this.offersService.deleteOne(id, currentUser);
}

@Post(":id/actions/accept")
@HttpCode(HttpStatus.OK)
@UseGuards(UserNotRestrictedGuard)
async acceptOffer(@Param() { id }: IdParams, @Body() body: AcceptOfferDto, @InjectUser()
currentUser: User) {
  const result = await this.offersService.acceptOffer(id, body, currentUser);
  return { result };
}

@Post(":id/actions/in-delivery")
@HttpCode(HttpStatus.OK)
async offerInDelivery(@Param() { id }: IdParams, @InjectUser() currentUser: User) {
  const result = await this.offersService.offerInDelivery(id, currentUser);
  return { result };
}

@Post(":id/actions/cancel")
@HttpCode(HttpStatus.OK)
async cancelOffer(@Param() { id }: IdParams, @InjectUser() currentUser: User) {
  const result = await this.offersService.cancelOffer(id, currentUser);

```

```
    return { result };  
}
```

```
@Post(":id/actions/close")  
@HttpCode(HttpStatus.OK)  
async closeOffer(@Param() { id }: IdParams, @InjectUser() currentUser: User) {  
    const result = await this.offersService.closeOffer(id, currentUser);  
    return { result };  
}  
}
```

Контроллер «AuthController»

```
import {
  Body,
  Controller,
  Delete,
  Get,
  HttpStatusCode,
  HttpStatus,
  Patch,
  Post,
  UploadedFile,
  UseGuards,
  UseInterceptors,
} from "@nestjs/common";
import { ApiTags } from "@nestjs/swagger";
import { Cookies, InjectCookies } from "@core/common";

import { AuthService } from "../auth.service";
import { ForgotPasswordDto, LoginDto, RegistrationDto } from "../dto";
import { RegisterVerifyDto } from "../dto/register-verify.dto";
import { ForgotPasswordVerifyDto } from "../dto/forgot-password-verify.dto";
import { UserAuthGuard, UserNotAuthGuard } from "../guards";
import { InjectUser } from "@core/decorators";
import { User } from "@database";
import { ApiResultResponse } from "@core/dto";
import { UpdateProfileDto } from "../dto/update-profile.dto";
import { ChangePasswordDto } from "@users";
import { FileInterceptor } from "@nestjs/platform-express";
import { FormDataBodyInterceptor } from "@core/interceptors";

@Controller("v1/auth")
@ApiTags("auth")
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Get("current")
  @UseGuards(UserAuthGuard)
  @ApiResultResponse({
    type: User,
    status: 200,
  })
  async getCurrentUser(@InjectUser() user: User) {
    await this.authService.populate(user);
    return { result: user };
  }

  @Patch("current/profile")
```

```

@UseGuards(UserAuthGuard)
@ApiResultResponse({
  type: User,
  status: 200,
})
async updateProfile(@Body() body: UpdateProfileDto, @InjectUser() user: User) {
  const result = await this.authService.updateProfile(body, user);
  return { result };
}

@Patch("current/password")
@UseGuards(UserAuthGuard)
@ApiResultResponse({
  type: User,
  status: 200,
})
async updatePassword(@Body() body: ChangePasswordDto, @InjectUser() user: User) {
  const result = await this.authService.updatePassword(body, user);
  return { result };
}

@Patch("current/avatar")
@UseGuards(UserAuthGuard)
@ApiResultResponse({
  type: User,
  status: 200,
})
@UseInterceptors(FileInterceptor("image"), FormDataBodyInterceptor)
async updateAvatar(@InjectUser() user: User, @UploadedFile() image: Express.Multer.File) {
  const result = await this.authService.updateAvatar(image, user);
  return { result };
}

@Delete("current/avatar")
@UseGuards(UserAuthGuard)
async removeAvatar(@InjectUser() user: User) {
  const result = await this.authService.removeAvatar(user);
  return { result };
}

@Post("logout")
@UseGuards(UserAuthGuard)
@HttpCode(HttpStatus.NO_CONTENT)
async logout(@InjectUser() user: User, @InjectCookies() cookies: Cookies) {
  await this.authService.logout(user, cookies);
}

@Post("tokens/refresh")
@ApiResultResponse({
  type: User,
  status: 200,
}

```

```

    })
    @HttpCode(HttpStatus.NO_CONTENT)
    async refreshTokens(@InjectCookies() cookies: Cookies) {
        await this.authService.refreshTokens(cookies);
    }

    @Post("login")
    @UseGuards(UserNotAuthGuard)
    @ApiResponseResponse({
        type: User,
        status: 200,
    })
    @HttpCode(HttpStatus.OK)
    async login(@Body() body: LoginDto, @InjectCookies() cookies: Cookies) {
        const user = await this.authService.login(body, cookies);
        return { result: user };
    }

    @Post("register")
    @UseGuards(UserNotAuthGuard)
    async register(@Body() body: RegistrationDto) {
        await this.authService.register(body);
    }

    @Post("register/verify")
    @UseGuards(UserNotAuthGuard)
    @HttpCode(HttpStatus.NO_CONTENT)
    async registerVerify(@Body() body: RegisterVerifyDto) {
        await this.authService.registerVerify(body);
    }

    @Post("forgot-password")
    @UseGuards(UserNotAuthGuard)
    @HttpCode(HttpStatus.NO_CONTENT)
    async forgotPassword(@Body() body: ForgotPasswordDto) {
        await this.authService.forgotPassword(body);
    }

    @Post("forgot-password/verify")
    @UseGuards(UserNotAuthGuard)
    @HttpCode(HttpStatus.NO_CONTENT)
    async forgotPasswordVerify(@Body() body: ForgotPasswordVerifyDto) {
        await this.authService.forgotPasswordVerify(body);
    }
}

```

КОМПОНЕНТ «OfferFormPageComponent»

```

import { Component, DestroyRef, OnDestroy, OnInit } from '@angular/core';
import { FormControl, FormGroup, ReactiveFormsModule } from '@angular/forms';
import {
  CoreModule,
  CrudAbstractService,
  CustomValidator,
  DropzoneComponent,
  EntityFormPageComponent,
  FilesValidator,
  InputComponent,
  SelectComponent,
} from '@app/core';
import { OffersCurrentUserService } from '@app/offers';
import { DeliveryType, ICategory, IOffer, offerChangeableWhen, PriceType } from '@shared';
import { FormUtils } from '@utils';
import { Subject } from 'rxjs';
import { MenuItem } from 'primeng/api';
import { CategorySelectComponent, CategoryService } from '@app/categories';
import { SettlementSelectComponent } from '@app/settlements';
import { takeUntilDestroyed } from '@angular/core/rxjs-interop';

@Component({
  selector: 'app-offer-form-page',
  standalone: true,
  imports: [
    CoreModule,
    EntityFormPageComponent,
    ReactiveFormsModule,
    InputComponent,
    SelectComponent,
    DropzoneComponent,
    CategorySelectComponent,
    SettlementSelectComponent,
  ],
  templateUrl: './offer-form-page.component.html',
  styleUrls: ['./offer-form-page.component.scss'],
  providers: [
    {
      provide: CrudAbstractService,
      useExisting: OffersCurrentUserService,
    },
  ],
})
export class OfferFormPageComponent implements OnInit, OnDestroy {
  private readonly formDestroyed$ = new Subject<void>();

```

```
readonly filesValidator = FilesValidator.imagesValidator;
readonly breadcrumb: MenuItem[] = [{ label: 'offers', routerLink: '/offers' }, { label: 'offer' }];
```

```
entity?: IOffer;
form!: FormGroup;
categoriesMapping: Record<string, ICategory> = {};
selectedCategory?: ICategory | null;
availableDeliveryTypes: DeliveryType[] = [];
hasNoPrice = false;
```

```
constructor(
  private readonly destroyRef: DestroyRef,
  private readonly categoryService: CategoryService,
) {}
```

```
ngOnInit() {
  this.categoryService.mapping$
    .pipe(takeUntilDestroyed(this.destroyRef))
    .subscribe(categoriesMapping => (this.categoriesMapping = categoriesMapping));
}
```

```
ngOnDestroy() {
  this.formDestroyed$.next();
  this.formDestroyed$.complete();
}
```

```
onFormInit(entity: IOffer | undefined) {
  this.entity = entity;
  this.form = new FormGroup(
    FormUtils.getFormControls(
      {
        images: new FormControl(FormUtils.getFilesFormValue(entity?.images)),
        title: new FormControl("", CustomValidator.required()),
        description: new FormControl("", CustomValidator.required()),
        categoryId: new FormControl(null, CustomValidator.required()),
        settlementId: new FormControl(null, CustomValidator.required()),
        deliveryTypes: new FormControl([], CustomValidator.required()),
        price: new FormControl('0', CustomValidator.required()),
      },
      {
        initialValues: entity,
        ignoreFields: ['images'],
        getValue: ({ key, value }) => {
          switch (key) {
            case 'price':
              return value?.toString();
            default:
              return value;
          }
        },
      },
    ),
  );
}
```

```

    },
  ),
);

if (this.entity && !offerChangeableWhen.includes(this.entity.status)) this.form.disable();

FormUtils.subscribeFieldChange(this.form, 'categoryId', this.formDestroyed$).subscribe(value
=> {
  this.selectedCategory = (value && this.categoriesMapping[value]) || null;

  this.syncDeliveryTypes();
  this.syncPrice();
});
}
private syncPrice() {
  const field = this.form.get('price') as FormControl<string>;

  this.hasNoPrice = Boolean(!this.selectedCategory || this.selectedCategory.priceType ===
PriceType.NONE);
  if (this.hasNoPrice) field.setValue('0');
}

private syncDeliveryTypes() {
  const field = this.form.get('deliveryTypes') as FormControl<DeliveryType[]>;
  this.availableDeliveryTypes = this.selectedCategory?.deliveryTypes || [];

  const deliveryTypes = (field.value || []).filter(deliveryType =>
    this.availableDeliveryTypes.includes(deliveryType),
  );
  field.setValue(deliveryTypes);
}
}

```

КОМПОНЕНТ «LoginPageComponent»

```

import { Component } from '@angular/core';
import { CoreModule, NavigationService } from '@app/core';

import { AuthFormComponent, AuthFormField, AuthFormSubmittedEvent, AuthFormFooter } from
 '@app/auth/auth-form';
import { AuthService } from '@app/auth/services';
import { finalize } from 'rxjs';
import { RouterLink } from '@angular/router';
import { ButtonModule } from 'primeng/button';
import { Login } from '@app/auth/services/dto';

@Component({
  selector: 'app-login-page',
  standalone: true,
  imports: [CoreModule, AuthFormComponent, ButtonModule, RouterLink],
  templateUrl: './login-page.component.html',
  styleUrls: ['./login-page.component.scss'],
})
export class LoginPageComponent {
  readonly fields = [AuthFormField.Login, AuthFormField.Password];
  readonly footer: AuthFormFooter = {
    label: 'registration',
    routerLink: '/auth/register',
  };

  isLoading = false;

  constructor(
    private readonly authService: AuthService,
    private readonly navigationService: NavigationService,
  ) {}

  onSubmitted(data: AuthFormSubmittedEvent) {
    this.isLoading = true;

    this.authService
      .login(data as Login)
      .pipe(finalize(() => (this.isLoading = false)))
      .subscribe(() => {
        this.navigationService.navigate(['/']);
      });
  }
}

```

Сервіс «CrudAbstractService»

```

import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, combineLatest, finalize, first, map, Observable, of, switchMap, tap } from
'rxjs';

import { ApiListResult, ApiResult, Filter, Forceable } from '../models';
import { SnackBarService } from './snackbar.service';
import { Id, StringUtils, Entity, ArrayUtils } from '@shared';
import { Router } from '@angular/router';
import { ParamsUtils } from '@utils';

export abstract class CrudAbstractService<
  Model extends Entity,
  Item extends Entity = Model,
  SubmitModel = Partial<Model>,
> {
  protected abstract readonly http: HttpClient;
  protected abstract readonly snackBarService: SnackBarService;
  protected abstract readonly router: Router;
  protected abstract readonly baseUrl: string;

  protected readonly _isInitted = new BehaviorSubject(false);
  protected readonly _list = new BehaviorSubject<ApiListResult<Item> | undefined>(undefined);
  protected readonly _item = new BehaviorSubject<Item | undefined>(undefined);
  protected readonly _isListLoading = new BehaviorSubject(false);
  protected readonly _isItemLoading = new BehaviorSubject(false);
  protected readonly _isDeleting = new BehaviorSubject(false);
  protected readonly forceDefault: boolean = false;
  protected readonly saveDefault: boolean = true;

  protected readonly NOTIFICATION_CREATED: string = "";
  protected readonly NOTIFICATION_UPDATED: string = "";
  protected readonly NOTIFICATION_DELETED: string = "";

  protected readonly itemPathTpl: string = "";

  readonly isInitted$ = this._isInitted.asObservable();
  readonly list$ = this._list.asObservable();
  readonly item$ = this._item.asObservable();
  readonly isListLoading$ = this._isListLoading.asObservable();
  readonly isItemLoading$ = this._isItemLoading.asObservable();
  readonly isDeleting$ = this._isDeleting.asObservable();

  readonly isLoading$ = combineLatest([this.isListLoading$, this.isItemLoading$,
this.isDeleting$]).pipe(
  map(() => this.isLoading),
);

```

```

getAll(filter?: Filter<Item> & Forceable) {
  return this.fetch(
    this._list,
    this._isListLoading,
    this.http.get<ApiListResult<Item>>(this.baseUrl, { params: this.getParams(filter) }),
    filter,
  );
}

```

```

getOne(id: Id, filter?: Omit<Filter<Item>, 'id'> & Forceable) {
  return this.fetch(
    this._item,
    this._isItemLoading,
    this.http
      .get<ApiResult<Item>>(`${this.baseUrl}/${id}`, { params: this.getParams(filter) })
      .pipe(map(({ result }) => result)),
    filter,
  );
}

```

```

addOne(body: Omit<SubmitModel, keyof Entity>) {
  this._isItemLoading.next(true);

  return this.processSubmitData(body).pipe(
    switchMap(data => this.http.post<ApiResult<Item>>(this.baseUrl, data)),
    switchMap(response => {
      return this.afterEntityCUD().pipe(map(() => response));
    }),
    finalize(() => this._isItemLoading.next(false)),
    tap(({ result }) => {
      this.showSuccessSnackbar(this.NOTIFICATION_CREATED);
      this.syncResultItems(result);
      const redirectTo = this.getRedirectTo(result);
      if (!redirectTo) return;

      this.router.navigate(['/', ...redirectTo.split('/').filter(Boolean)]);
    }),
  );
}

```

```

updateOne(id: Id, body: Omit<SubmitModel, keyof Entity>, entity?: Item) {
  this._isItemLoading.next(true);

  return this.processSubmitData(body, id, entity).pipe(
    switchMap(data => this.http.put<ApiResult<Item>>(`${this.baseUrl}/${id}`, data)),
    switchMap(response => {
      return this.afterEntityCUD().pipe(map(() => response));
    }),
    finalize(() => this._isItemLoading.next(false)),
  );
}

```

```

    tap(({ result }) => {
      this.showSuccessSnackbar(this.NOTIFICATION_UPDATED);
      this.syncResultItems(result);
    }),
  );
}

```

```

deleteOne(id: Id): Observable<unknown> {
  this._isDeleting.next(true);
  return this.http.delete(`${this.baseUrl}/${id}`).pipe(
    finalize(() => this._isDeleting.next(false)),
    switchMap(response => {
      return this.afterEntityCUD().pipe(map(() => response));
    }),
    tap(() => {
      this.showSuccessSnackbar(this.NOTIFICATION_DELETED);
      this.onEntitiesRemoved(id);
    }),
  );
}

```

```

cleanUp() {
  this._list.next(undefined);
  this._item.next(undefined);
  this._isListLoading.next(false);
  this._isItemLoading.next(false);
  this._isDeleting.next(false);
}

```

```

getRedirectTo(entity: Item) {
  return StringUtils.injectIntoTemplate(this.itemPathTpl, entity as Record<string, unknown>);
}

```

```

// eslint-disable-next-line @typescript-eslint/no-unused-vars
protected processSubmitData(body: Omit<SubmitModel, keyof Entity>, id?: Id, entity?: Item):
Observable<any> {
  return of(body);
}

```

```

protected afterEntityCUD(): Observable<any> {
  return of(null);
}

```

```

protected onEntitiesRemoved(ids: Id | Id[]) {
  if (!this.list) return;

```

```

  const { result, count } = this.list;

```

```

  const idsArr = ArrayUtils.mapToArray(ids);
  const nextResult = result.filter(item => !idsArr.includes(item.id));
  const diff = result.length - nextResult.length;

```

```

    if (diff) this._list.next({ result: nextResult, count: count - diff });
  }

protected syncResultItems(entities: Item | Item[]) {
  const result = [...(this.list?.result || [])];
  let isListChanged = false;

  ArrayUtils.mapToArray(entities).map(entity => {
    const listIdx = result.findIndex(listItem => listItem.id === entity.id);

    if (listIdx !== -1) {
      result[listIdx] = { ...result[listIdx], ...entity };
      isListChanged = true;
    }

    if (this.item?.id === entity.id) {
      this._item.next({ ...this.item, ...entity });
    }
  });

  if (isListChanged) {
    this._list.next({ ...(this.list || {}), result } as ApiListResult<Item>);
  }
}

protected fetch<T>(
  entity$: BehaviorSubject<T | undefined>,
  isLoading$: BehaviorSubject<boolean>,
  fetch$: Observable<T>,
  filter: Forceable | undefined,
): Observable<T> {
  const { force = this.forceDefault, save = this.saveDefault } = filter || {};

  if (!force) {
    if (entity$.getValue()) return of(entity$.getValue() as T);

    if (isLoading$.getValue()) {
      return isLoading$.pipe(
        first(isLoading => !isLoading),
        switchMap(() => {
          const entity = entity$.getValue() as T;
          if (entity) return of(entity);

          return this.fetch(entity$, isLoading$, fetch$, { ...(filter || {}), force: true });
        })
      );
    }
  }
}

```

```

isLoading$.next(true);

return fetch$.pipe(
  finalize(() => isLoading$.next(false)),
  tap(response => save && entity$.next(response)),
);
}

protected getParams<F extends Forceable>(filter: F | undefined) {
  return ParamsUtils.getParams(filter, {
    predicate: key => ![ 'force', 'save' ].includes(key),
  });
}

protected showSuccessSnackbar(detail: string) {
  if (detail) this.snackbarService.add({ severity: 'success', detail });
}

get list() {
  return this._list.getValue();
}

get item() {
  return this._item.getValue();
}

get isListLoading() {
  return this._isListLoading.getValue();
}
get isItemLoading() {
  return this._isItemLoading.getValue();
}

get isDeleting() {
  return this._isDeleting.getValue();
}

get isLoading() {
  return this.isListLoading || this.isItemLoading || this.isDeleting;
}
}

```