ДИПЛОМНА РОБОТА

ДР. ПІс – 24.00.000 ПЗ

Група ПІс-2015

Шіман М. І.

2019

ПВНЗ Університет Короля Данила

Кафедра Інформаційних технологій та програмної інженерії

УДК 004

ДИПЛОМНА РОБОТА

Тема ***Розробка інформаційного веб-сервісу робочих місць користувачів роздрібної торгівлі.***

Напрям підготовки ***6.050103 – «Програмна інженерія»***
(код і назва спеціальності)

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДР.ПІс – 24.00.000 ПЗ
(позначення)

Студент
Шіман М.І.
(підпис) (дата) (розшифрування підпису)

Керівник проекту
д.т.н.,доц. Мельничук С.І.
(посада) (підпис) (дата) (розшифрування підпису)

Нормоконтроль
к.т.н. Мануляк І.З.
(посада) (підпис) (дата) (розшифрування підпису)

Допускається до захисту
Завідувач кафедри
д.т.н.,доц. Мельничук С.І.
(посада) (підпис) (дата) (розшифрування підпису)

2019

ПВНЗ УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет *Інформаційних технологій*
Кафедра *Інформаційних технологій та програмної інженерії*
Напрям підготовки *6.050103 – «Програмна інженерія»*

ЗАТВЕРДЖУЮ:
Завідувач кафедри ІТПІ
_____ *С.І. Мельничук*
"____" _____ 2019 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Студенту *Шіману Маріусові Івановичу*

1. Тема проекту (роботи) *Розробка інформаційного веб-сервісу робочих місць користувачів роздрібної торгівлі.*

Затверджена наказом ректора Університету Короля Данила від *15.11.2018 р. № 20/4*

2. Термін здачі студентом закінченого проекту (роботи) *10.06.2019 р.*

3. Вихідні дані до проекту (роботи) *Java, Spring Framework, MySQL, JavaScript, Vue.js*

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити) *1.Огляд наявних аналогів та їх недоліків обґрунтування використаних технологій. 2. Налаштування конфігурації сервера та визначення типу зв'язку з клієнтом, проектування структури бази даних та розробка структури сайту. 3. Розробка основних компонентів та огляд інтерфейсу користувача.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *Постановка завдання, огляд сервісу аналогів: OLX та prom.ua, функції доступні користувачам, структура сайту, огляд сторінки каталогу товарів, огляд системи замовлень, висновки.*

6. Консультанти з проекту (роботи), із зазначенням розділів проекту, що стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 30.10.2018 р.

Керівник _____ Мельничук С. І.

Завдання прийняв до виконання _____ Шіман М.І.

КАЛЕНДАРНИЙ ПЛАН

Пор №	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1.	Опис типових реалізацій та сервісів аналогів	04.12.2018	
2.	Розробка структури бази даних, визначення типів даних та зв’язків між таблицями	12.12.2018	
3.	Розробка користувацької частини та інтерфейсу користувача	24.01.2019	
4.	Розробка серверу додатку	05.03.2019	
5.	Тестування серверу та користувацької частини сервісу	12.04.2019	
6.	Оформлення пояснювальної записки	02.05.2019	
7.	Оформлення графічного матеріалу та підготовка до захисту роботи	23.05.2019	

Студент-дипломник _____ *Шіман М. І.*
(підпис) (розшифровка підпису)

Керівник проекту _____ *Мельничук С. І.*
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Підчас виконання дипломної роботи було розроблено back end частину а також інтерфейс користувача інформаційного веб-сервісу робочих місць користувачів роздрібної торгівлі. За допомогою сервіс можна здійснювати роздрібний продаж та купівлю товарів у інших користувачів за допомогою функціоналу проекту. Сервіс повністю готовий до розгортання на хостингу або локальному комп’ютері.

SUMMARY

During the implementation of the thesis, the back end part as well as the user interface of the information service web site of the retail user's workplaces was designed and developed. With the help of the service you can retake and buy goods from other users through the project's function. The service is fully ready for deployment on a hosting or local computer.

РЕФЕРАТ

Розрахунково-пояснювальна записка: 83 сторінок, 44 рисунки.

Завданням на дипломну роботу є розробка інформаційного веб-сервісу робочих місць користувачів роздрібної торгівлі.

Відповідно до поставленого завдання у дипломній роботі проводиться та описується процес розробки та тестування інформаційного веб-сервісу, метаю якого є зпрощення процесу купівлі/продажу роздрібних товарів. У процесі виконання буде розроблено сервер додатку, який буде виконувати всі команди з клієнту за допомогою API, яке дозволить гнучко використовувати всі розроблені функції. Сервер може бути використаний також окремо від користувацької частини, так як всі функції будуть розроблені таким чином, щоб їх можна було використати на різних реалізаціях клієнту. Окрім сервера, буде розроблено клієнтську частину, яка буде включати весь функціонал сервера, та буде мати користувацький інтерфейс для зручного користування. Клієнтську частину буде розроблено та спроектовано для використаня на будь-яких типах пристроїв для того, щоб веб-сервісом можна було скористатись з на будь-якому розмірі екрану телефону, планшету чи ПК.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

CSS — Cascading Style Sheets;

HTML — HyperText Markup Language;

JS – JavaScript;

JSON – JavaScript Object Notation;

БД – база даних;

СКБД – система керування базами даних;

JWT – JSON Web Token;

API –Application Programming Interface.

ВСТУП

Актуальність теми. В сучасному світі, кожного дня мільйони людей роблять покупки, торгуються, та виконують різні операції пов’язані з торгівлею. Кожна людина кожного дня приймає участь у цій сфері. На відміну від того, щ обуло колись, сучасна торгівля сильно відрізняється від тієї, що була навіть сто років тому. Появилось безліч видів роздрібної торгівлі, які доступні всім бажаючим, в число цих нових способів входять інтернет магазини і веб-сервіси, які забезпечують умови торгівлі.

Мета дослідження полягає у вивченні та огляді сервісів-аналогів, та виявлення всіх їхніх недоліків а також типових функцій, для того щоб після цього можна було створити сервіс, у якому зосереджені самі популярні серед аналогів функції і при цьому відсутні типові недоліки.

Об’єкт досліджень – інформаційні веб-сервіси робочих місць користувачів роздрібної торгівлі.

Предмет дослідження – функції для користувача, які підвищують зручність використання на продуктивність сервісу.

Задачі досліджень:

- розглянути типові реалізації веб-сервісів для забезпечення роздрібної торгівлі;
- розглянути та визначити недоліки сучасних веб-сервісів для забезпечення роздрібної торгівлі;
- на основі зібраних даних розробити веб-сервіс, який буде реалізувати найбільш корисний функціонал, а також не буде мати недоліків типових реалізацій сервісів даного типу.

Методи дослідження – методи порівняння, які допоможуть з усіх наявних аналогів вибрати найбільш потрібний та корисний функціонал, який буде вибрано за допомогою порівняння, за допомогою методів експеременту буде

виявлено додаткові функції відсутні в аналогах при цьому які будуть не менш корисними.

Наукова новизна полягає у створенні унікального веб-сервісу для забезпечення роздрібної торгівлі, який буде вбирати в себе всі кращі функції аналогів, без типових недоліків.

Практична значимість розробки полягає у зпрощенні процесу роздрібної торгівлі в інтернеті, автоматизації та оптимізації даного процесу.

Ключові слова: інформаційні веб-сервіси, послідовність замовлення товарів, користувачі, адміністратори, товари, універсальність, гнучкість.

1 ОПИС АНАЛОГІВ ТА ПРИНЦИП РОБОТИ САЙТУ

1.1 Огляд типових реалізацій інформаційних веб-сервісів для забезпечення роздрібної торгівлі

Сайти для роздрібної та оптової торгівлі в наш час організовуються за досить стандартизованою схемою та структурою. Основним призначенням таких сайтів є надання користувачам (клієнтам) ознайомитись з наявним асортиментом товарів, а також замовити все що їх цікавить. Це звільняє продавця від складних процедур винаймання приміщень, розміщення товарів на місці. А покупця звільняє від обов'язкових походів чи поїздок до продавця товару, який цікавить, особливо якщо потрібно просто переглянути асортимент. Причина, по якій немає сенсу – це те, що клієнтам важко орієнтуватись на сервісі, тому зазвичай структура таких сайтів складається з наступних компонентів:

- головна сторінка – зазвичай на цій сторінці розміщені новини проекту, після яких йдуть популярні товари (або так званий топ товарів, місце в якому потрібно купити);
- каталог товарів – основний інструмент взаємодії із вмістом сервісу, на будь якому сервісі є така сторінка. Реалізація каталогу може бути різною, його можуть реалізовувати як список, або ж за допомогою карточок товарів, на яких виводиться фото та коротка інформація про позицію;
- функціонал та сторінка кошику – ні один інтернет-магазин чи будь-який сайт пов'язаний з торгівлею не може обійтись без кошика, так як це є одним з основних інструментів;
- форма створення замовлення та перегляд замовлень – як і у випадку з кошиком, ці два компоненти не можуть існувати окремо, так як в них не буде ніякого сенсу.

Таким чином формується розуміння того, що може бути присутнє на таких сайтах, а що є додатковим функціоналом. Також може бути присутній додатковий функціонал, який спрощує пошук чи користування основним функціоналом. Наприклад для пошуку товарів у каталозі часто присутня велика кількість фільтрів. У випадку з сервісом оголошень OLX велику роль має місто чи область продажу, так як на сервісі присутні оголошення з всієї України, і якщо потрібний товар знаходиться у іншій частині країни, чи навіть у сусідньому місті – доставка передбачає додаткові витрати на перевезення. Таким чином, іноді користування каталогом доступних оголошень чи товарів, у якому немає фільтрів – просто неможливе.

1.2 Огляд веб-сервісів аналогів робочих місць користувачів роздрібної торгівлі

В Україні основним сервісом для продажу та покупки нових та бувших у використанні товарів, є OLX, але в першу чергу, сервіс є майданчиком оголошень, будь-якого типу, від оголошень купівлі та продажу, до вакансій.

OLX — платформа онлайн-оголошень в Україні, яка об'єднує людей для покупки, продажу або обміну товарами та послугами [1].

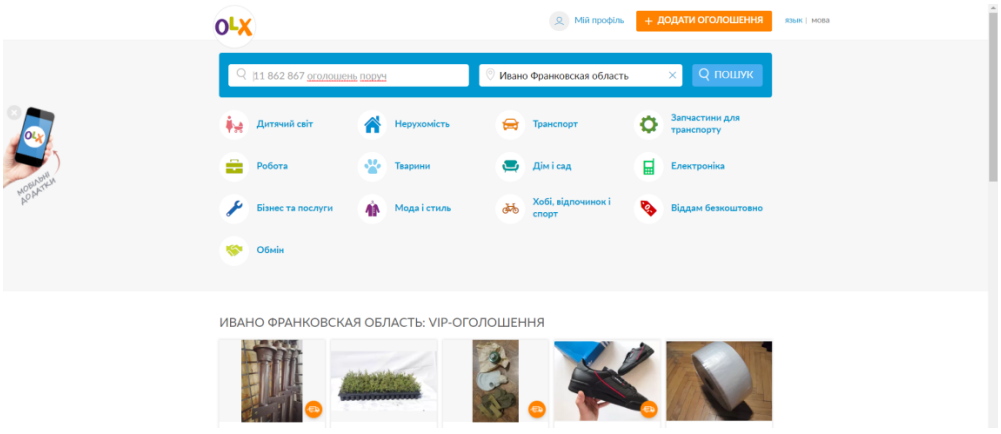

Рисунок 1.1 – Головна сторінка сервісу OLX

Головна сторінка сервісу зображена на рис. 1.1. Станом на час написання даного порівняння на сервісі присутньо понад 12 мільйонів оголошень. Станом на 2018 рік на сервісі зареєстровано 1,5 млн продавців, розміщено понад 11 млн оголошень і кожну хвилину додається близько 100 нових. OLX є найбільш відвідуваним сервісом оголошень в Україні.

Кожен другий інтернет-користувач з України відвідує ресурс мінімум один раз на місяць. Але на жаль навіть такий великий сервіс як OLX задля універсальності та різнонаправленості опустив безліч моментів, що призводить до досить таки розпливчатих оголошень, особливо коли діло доходить до продажу багатьом покупцям за один раз.

Можна змоделювати досить просту ситуацію, коли користувач продає будь який товар оптом на OLX. З самого початку не буде ніяких проблем, сервіс дозволяє вказати багато даних про товар, опис та заголовок оголошення, але проблеми починаються тоді, коли багатьох користувачів зацікавить це оголошення, і вони почнуть писати, та дзвонити по вказаним в оголошенні номерах. В цій ситуації є декілька проблем, одна з них – це часті відволікання продавця дзвінками, можливо навіть в не дуже зручний для нього момент. Друга проблема яка витікає з першої – це поширення номера по мережі, кожен хто відвідує OLX може дізнатись номер і ім’я продавця що часто приводить до спаму, розсилок смс та дзвінків задля обману людей. Третя проблема це неможливість якимось чином зафіксувати хто і що замовив і тим більше скільки замовив. Ця проблема частково вирішується якщо покупці не телефонують а пишуть повідомлення які на сервісі також присутні, але якщо покупців багато – це все разом перемішується, і важко якимось чином відслідкувати і тим більше перевірити стутуси цих замовлень. Ця проблема частково була вирішена з впровадженням функції «OLX доставка», але це стосується лише частини замовлень з інших міст, а також не завжди і не кожен може підключити цю послугу, так як в неї є свої певні правила. Також у сервісу відсутні будь які позначення одиниць товару який продається, наприклад літри, кілограми тощо, що призводить до частих путаниць та проблем з кількістю товарів, та ціною за

цю кількість – це сильно розмиває мету оголошення і веде за собою лишні запитання та уточнення, які тільки шкодять швидкості оброблення замовлень.

Іншим прикладом торгового сервісу є Prom.ua [2] (головна сторінка сервісу зображена на рис 1.2) – досить великий проект, який також надає можливість користувачам розміщувати свої товари, також на сервісі присутня більша адаптація для оптових продаж ніж в попереднього сервісу, але свою не дуже хорошу репутацію Prom заслужив тим, що він не зовсім безкоштовний, присутній ліміт в 10 оголошень, якого часто не достатньо продавцям, в яких є багато товарів, що заставляє їх купляти додаткові оголошення.

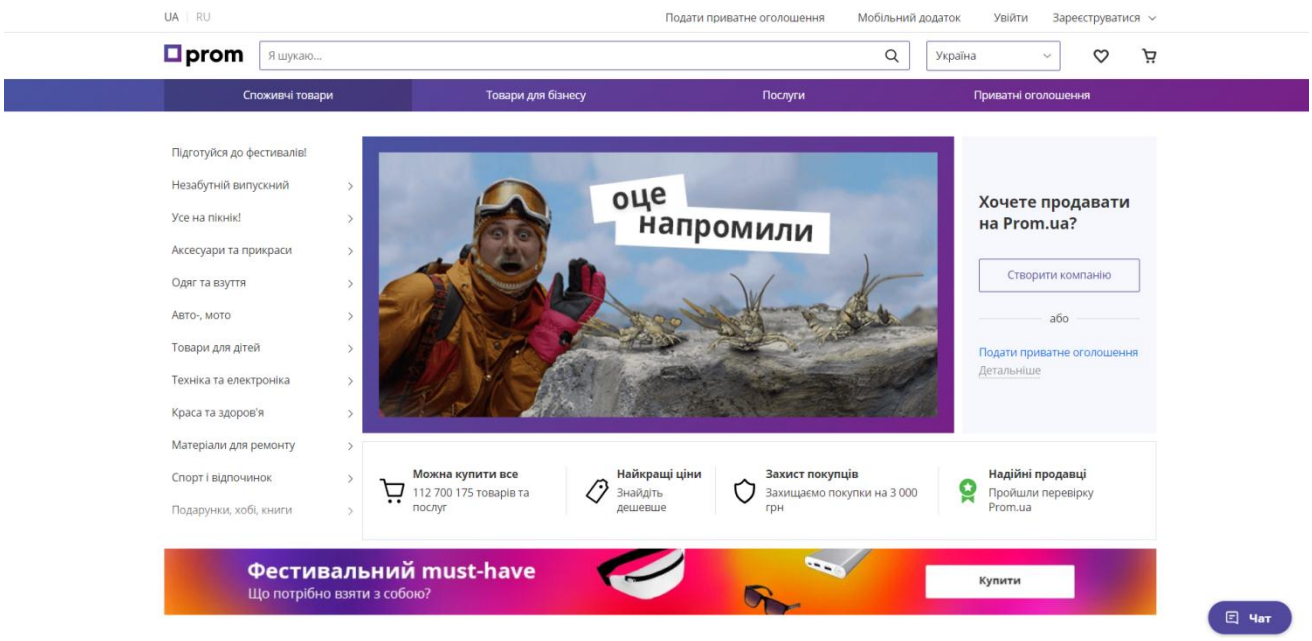

Рисунок 1.2 – Головна сторінка Prom.ua

В результаті чого сервіс для розміщення оголошень перетворюється в місце на території ринку за яке потрібно платити оренду. В OLX також обмежена кількість замовлень, але сервіс і не позиціонується як місце для оптової торгівлі, все ж таки OLX це більше сервіс для оголошень, кількості яких зазвичай достатньо для людей яким потрібно продати будь-які речі.

На відміну від вище перечислених сервісів, мій проект адаптований саме для оптового продажу, і містить в собі зручні і чіткі розділи і обмеження, щоб всі товари були інтуітивно посортовані і доступні. Також присутні одиниці

вимірювання товарів. Наприклад, при додаванні товару продавець може вибрати в чому саме вимірюється товар, в кілограмах, грамах, літрах тощо.

Це набагато зменшує путаницю, так як покупець бачить що, скільки і чим вимірюється позиція на сервісі.

Присутня система керування замовленнями, з статусами та коментування статусів. Це означає що кожен, хто замовив будь-який товар, зможе відслідкувати його статус, а також зможе побачити прикріплений до статусу коментар. Це може бути зручно наприклад якщо продавець хоче передати покупцю якусь інформацію, наприклад номер відділення, час прибуття та номер кур'єра тощо.

Для покупця доступні статуси замовлень, по яким він може зрозуміти стан замовлення, перевірити чи воно відправлено, або ж скасовано/відхилено продавцем. В замовленні також вказані всі товари, та їх кількість, це надає можливість зручного обліку проданих товарів, та дозволяє продавцю уникнути путаниць в повідомленнях, дзвінках і зменшує кількість неуточнених даних.

Також додатковою зручною функцією для клієнтів, є збереження адрес доставки, один раз заповнивши форму адреси, можна в один клік додавати її до замовлень. Це позбавляє покупців лишній раз не заповнювати додаткових форм і ще більше скорочує час оформлення замовлення, що дозволяє зосередитись на покупках, а не на процесі комунікації з продавцем, та заповненню форм які щоразу повторюються. Продавець в свою чергу має доступ до всіх потрібних йому даних, таких як ім'я, номер мобільного та адреси доставки покупця.

1.3 Постановка задачі на дипломну роботу

Основним призначенням проекту, є надання торговим агентам, та іншим людям можливість просто та швидко продати будь-які продовольчі товари. Також для іншої сторони (покупців) створено зручну систему відслідковування замовлень та керування ними. Сервіс надає можливість будь-кому продати

власні товари та знайти нових клієнтів. Всі організаційні моменти по доставці та комунікації з клієнтами лягають на плечі продавців, так як проект представляє з себе лише зручний та якісний інструмент, а як ним роспоряджатися – вибирають продавці.

Першочерговою метою проекту є спрощення та зосередження багатьох торгових агентів, та інших людей які зацікавлені в покупці або продажу, в одному місці де буде зосереджена основна маса продовольчих товарів, робиться це для того, щоб кожен магазин, або мережа магазинів, будь-яке підприємство або ж людина яка захотіла придбати будь-який товар на площадці, могла швидко і зручно знайти, додати у кошик та оформити замовлення на позиції які її цікавлять.

Основними можливостями користувача на проекті є:

- вільний перегляд всіх товарів без реєстрації – кожен відвідувач сайту може переглянути каталог товарів по категоріях;
- реєстрація та вхід – реєстрація нових облікових записів, яка дозволяє користувачеві спробувати всі доступні функції;
- зміна базових параметрів облікового запису користувача – зміна імені, емейлу та паролю;
- додавання товарів на продаж – додавання власного товару на сервіс, встановлення цін, вибір одиниць вимірювання товару, категорію та завантаження зображення товару;
- керування товарами – зміна даних товару, опису, зображення;
- створення та керування замовленнями – замовлення товарів у інших користувачів сервісу, закріплення статусів за замовленнями, скасування та додавання коментарів до статусів;
- сортування та фільтрування товарів – сортування товарів у каталозі, глобальне групування всіх товарів сервісу по категоріях;
- кошик – єдиний кошик користувача, дані якого зберігаються у базі даних, в результаті все що додано у кошик на будь-якому пристрої синхронізується на всіх пристроях з одним і тим ж обліковим записом.

Весь функціонал сервісу є визначеним за допомогою аналізу предметної області та сервісів аналогів. Деякі функції було дадоно у список обов’язкового функціоналу, тому, що він відсутній на одних сервісах, але є на інших. Я старався вибрати самі корисні та зручні функції, щоб користування сервісом було зручним, а результат продуктивним.

1.4 Обґрунтування використаних технологій

Для користувацької частини сайту (front-end) було використано як стандартні для любого сайту технології, так додаткові можливості сучасних фреймворків. Це було зроблено для оптимізації як самого процесу користування, так і для зменшення навантаження на сервер.

Як і всі сайти в мережі, абсолютно всі сторінки були написані за допомогою мови розмітки HTML та оформлені за допомогою каскадних таблиць стилів CSS. Зазвичай, при написанні простих сайтів, які несуть виключно інформативну функцію цього було б достатньо. Так як сервіс для оптової торгівлі далеко не схожий на статичний сайт – було використано додаткові технології, системи збирання проекту, та фреймворки, такі як: Webpack, Vue.js, SCSS.

Webpack – дуже поширена система збирання проектів, відома своєю гнучкістю, великою кількістю налаштувань та параметрів. Перш за все Webpack здобув свою популярність завдяки не менш відомому фреймворку під назвою React та Redux. Для свого проекту, я використав цю систему збирання в парі з Vue CLI – це спеціально налаштована бібліотека (модуль), для збирання додатків, які написані за допомогою Vue.js.

Vue.js — JavaScript-фреймворк що використовує шаблон MVVM для створення інтерфейсів користувача на основі моделей даних, через реактивне зв'язування даних.

Model-View-ViewModel — це шаблон проектування, що застосовується під час проектування архітектури веб додатків. Публічно вперше був представлений у 2005 році як модифікація шаблону Presentation Model. MVVM орієнтований на такі сучасні платформи розробки, як Windows Presentation Foundation та Silverlight від компанії Microsoft.

Не дивлячись на те, що Vue.js є фреймворком мови javascript [3-8], він кардинально міняє сприйняття цієї мови, а завдяки шаблонам – міняє ще й структуру. Завдяки цьому стає можливим створення великих комплексних веб додатків, з розділеним функціоналом і модульною структурою, саме по цій причині я і вибрав Vue.js в якості фреймворку для створення користувацького інтерфейсу, адже швидка, зрозуміла і інтуітивна взаємодія користувача з системою – це одна з найважливіших частин мого проекту.

Основою серверної частини стала мова програмування Java з фреймворком Spring, який дозволяє зробити сервер для будь-чого в досить короткі терміни, а саме головне – ця робота буде набагато зручнішою і якіснішою навіть якби сервер був написаний на чистій мові Java, це вже не говорячи про те, що більшу частину речей можна робити тільки у даному фреймворку, так як він поєднує в собі дуже велику кількість бібліотек які дозволяють робити різні види веб серверів, і додатків, API тощо. Детальніше про кожен використаний компонент, я описав нижче, для того, щоб було зрозуміло що і для чого було використано.

Java є мовою програмування, за допомогою якої розробники програмного забезпечення створюють різни прикладні додатки для комп’ютерів, смартфонів, планшетів та інших інтелектуальних пристроїв. Особливістю програм на Java є те, що вони можуть запускатись на будь-яких комп’ютеризованих пристроях, які працюють під різними операційними системами, причому без повторної компіляції коду [15-19].

Spring Framework - універсальний фреймворк з відкритим вихідним кодом для Java-платформи. Також існує форк для платформи .NET Framework, названий Spring.NET [20].

Перша версія була написана Родом Джонсоном, який вперше опублікував її разом з виданням своєї книги «Expert One-on-One Java EE Design and Development» у жовтні 2002 року.

Незважаючи на те, що Spring не забезпечував якусь конкретну модель програмування, він став широко поширеним в Java-співтоваристві головним чином як альтернатива і заміна моделі Enterprise JavaBeans. Spring надає велику свободу Java-розробникам в проектуванні, крім того, він надає добре документовані і легкі у використанні засоби вирішення проблем, що виникають при створенні додатків корпоративного масштабу [20-24].

Але як вище згадувалось, Spring це не просто фреймворк, це набір потужних інструментів для розробки, який включає в себе безліч компонентів. Зроблено це для того, щоб кожен програміст, який використовує Spring, міг підключити лише те, що планує використовувати, що зразу ж позитивно впливає на оптимізацію, та розмір додатку, а також сильно звужує коло пошуку помилок.

Одним з таких компонентів є Spring Boot який дозволяє легко створювати автономні, Spring додатки корпоративного класу, які можна «просто запустити». Іншими словами – додатки написані за допомогою Spring Boot можна скомпілювати в звичайний jar файл [25].

Jar файли – це архіви, створені для вміщення в собі програм, написаних на мові Java, ці файли, призначені для віртуальної машини JRE, та містять в собі файли класів програми, маніфести і т.д. За допомогою цих файлів можуть бути підключені бібліотеки, або ж запущені звичайні програми, які до речі ще й ніяким чином не прив’язана до ОС, а це означає що вони можуть бути запущені на любому комп’ютері або системі, яка підтримує JRE.

Не дивлячись на те, що здавалося б всі бібліотеки для розробки вже описані, залишилась ще одна, в даному випадку як і у користувацькій частині сайту, система збирання проекту. В якості такої системи був вибраний Maven – система для збирання проектів, яка до того ж надає функцію майже автоматичного підключення будь-яких бібліотек [26]. Дана система збирання має

свій власний репозиторій, де зібрана більшість (якщо не всі) бібліотек для Java, система дозволяє додати будь-яку бібліотеку за допомогою додавання спеціальної залежності (рис. 2.1) у файлі pom.xml, який являє собою файл, у якому перечислені всі залежності в проекті, також він містить налаштування системи збирання, плагінів тощо. Все що потрібно зробити програмісту – додати залежності які його цікавлять у файл, все інше зробить за нього Maven. Основних залежностей файлу pom.xml, які я використав у своєму проекті:

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <optional>true</optional>
</dependency>
```

У кожній із залежностей є два основних параметра, це groupId – назва бібліотеки яка буде додана, також назва конкретного компоненту яка має назву artifactId. У деяких залежностей є своя версія, і при розробці можна легко змінити версію тієї чи іншої бібліотеки, не затрачаючи на це багато зусиль.

2 РОЗРОБКА АЛГОРИТМІВ ТА ВЗАЄМОЗВ'ЯЗКІВ МІЖ ФУНКЦІОНАЛЬНИМИ ОБ'ЄКТАМИ

2.1 Налаштування конфігурації сервера додатку

У дипломній роботі я використав систему керування базами даних MySQL. Можливостей даної СКБД досить навіть для дуже складних проектів з не менш складними базами даних, тому функціональних обмежень не виникало [27-29].

Spring Boot додатки використовують спеціальний файл конфігурації, який дозволяє швидко та просто налаштувати всі параметри, це стосується навіть параметрів бібліотек які входять до Spring. Крім цього, цей файл дозволяє конфігурувати параметри навіть коли сервер зкомпільовано в jar файл, що дуже спрощує розгортання проекту на хостингу або на локальному комп'ютері.

Файл конфігурації містить в собі такі параметри як:

- посилання на сервер баз даних;
- ім'я користувача для підключення до БД;
- пароль користувача для підключення до БД;
- налаштування діалекту двигуна який буде використовуватись у БД;
- параметри автоматичної генерації DDL для ORM системи;
- порт сервера по якому він буде доступний;
- налаштування мінімальних та максимальних розмірів даних які будуть поступати на сервер (розміри картинок у даному випадку);
- мінімальний та максимальний розмір файлів, які можуть бути завантажені на сервер;
- максимальний розмір запиту до сервера;
- параметри оптимізації даних.

Фрагмент коду файлу конфігурації:

```
spring.datasource.url=jdbc:mysql://localhost:3306/product_manager_db?useUnicode=yes&characterEncoding=UTF-8
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.generate-ddl=false
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=false
server.port = 80
spring.servlet.multipart.file-size-threshold=2KB
spring.servlet.multipart.max-file-size=5MB
spring.servlet.multipart.max-request-size=50MB
```

Файл містить всі вище перечислені параметри, а також додаткові, наприклад в який каталог будуть завантажуватись зображенн.

2.2 Проектування структури бази даних

Для проекту я використав СУБД MySQL, за допомогою якої здійснювалось зберігання та упорядкування даних.

MySQL — вільна система керування реляційними базами даних. Ця система керування базами даних з відкритим кодом була створена як альтернатива комерційним системам. В дипломному проекті MySQL використовувалась в парі з Hibernate ORM, яка значно пришвидшує процес створення програми, так як дозволяє швидко створити всі таблиці бази даних майже не використовуючи будь-які можливості середовища проектування баз даних MySQL. Це зменшило час розробки структури бази даних в рази, а також дозволило набагато краще зпроектувати БД, за рахунок продумування зв’язків між об’єктами напряму в коді, а все інше виконує ORM.

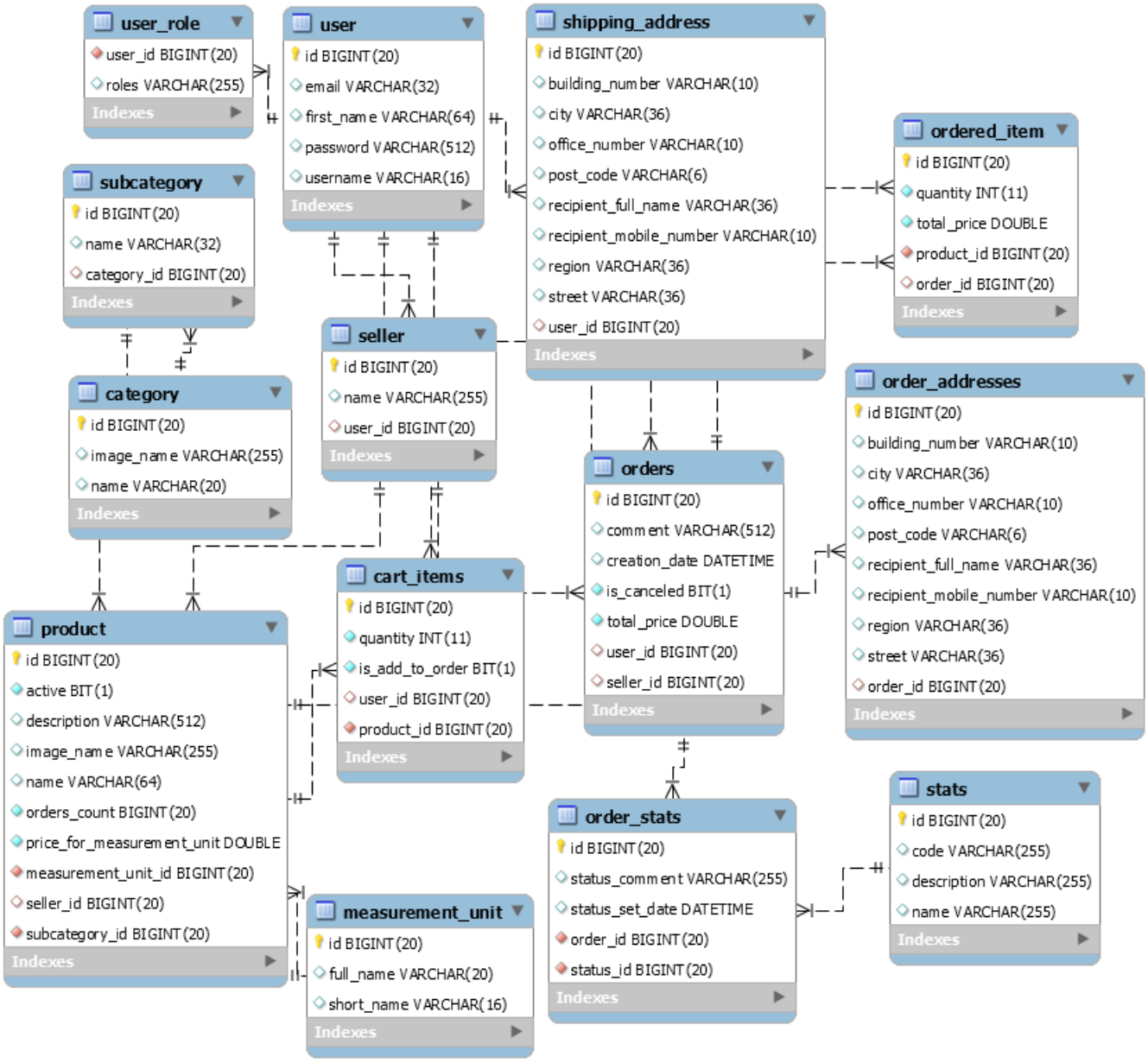

Рисунок 2.1 – Структура бази даних

ORM (Object-Relational Mapping) - технологія програмування, яка зв'язує базу даних з концепціями об'єктно-орієнтованих мов програмування, в даному випадку мовою програмування є Java, створюючи віртуальну об'єктну базу даних. Схема на рис. 2.1 фактично результат роботи даної системи керування зв’язками між об’ктами, так як всі параметри, кількість символів, та все інше було прописано напряму в коді. Також всі зв’язки між таблицями, унікальні значення та автоінкрементація ідентифікаторів у таблицях теж генеруються за допомогою ORM у коді сервера.

Всі таблиці бази даних, були створені виключно з метою оптимізації та збільшення швидкості роботи бази даних, в базі даних не існує таблиць, які б дублювали інші таблиці і це не було б логічно використано.

Прикладом створення таблиці через ORM є фрагмент коду нижче.

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "user_id")
@JsonIgnore
private User user;

@OneToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "seller_id")
@JsonIgnore
private Seller seller;

@Column(name = "user_id", insertable = false, updatable = false)
private Long userId;

@NotNull
@OneToMany(fetch = FetchType.EAGER, mappedBy = "order", cascade =
CascadeType.ALL)
@JsonManagedReference
private List<OrderedItem> orderedItems;

@OneToMany(fetch = FetchType.EAGER, mappedBy = "order", cascade =
CascadeType.ALL)
@OrderBy("statusSetDate DESC")
@JsonManagedReference
private Set<OrderStatus> statuses = new HashSet<>();

@Temporal(TemporalType.TIMESTAMP)
private Date creationDate;

@OneToOne(mappedBy = "order", cascade = CascadeType.ALL,
    fetch = FetchType.EAGER, optional = false)
private OrderAddress address;
```

Наведений код, це фрагмент класу написаний на мові програмування Java. Спершу оголошується ідентифікатор поля таблиці за допомогою анотації @id, також в наступному рядку вказується що дане значення є автоматично генерованим бібліотекою Hibernate. Значення @id генеруються глобально, це означає що інкрементація поля проходить глобально, наприклад, якщо в одній таблиці додалось нове поле, у колонці id буде значення 1, і після додавання даних в іншу таблицю, в доданому полі в колонці id буде значення 2. Так як для віртуальної машини, і для мови Java в цілому, це звичайний клас, далі поле оголошується за стандартизованим виглядом, як і будь-яка змінна у мові. І навіть типи полів не ігноруються, а записуються як аналогічні типи у БД, це дозволяє навіть не задумуватись про сумісність написаного коду, а точніше типів даних, з базою даних.

В фрагменті наведеному вище, також прописані реляційні моделі (ключі), в даному випадку присутня анотація:

```
@ManyToOne(fetch = FetchType.LAZY)
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "user_id")
@JsonIgnore
private User user;
```

Це означає, що запис поточної таблиці має зовнішній ключ у таблиці користувачів (user), цей ключ автоматично створюється у «віртуальній базі даних» і автоматично відслідковується бібліотекою Hibernate. Додатковим параметром анотації виступає параметр fetch, він відповідає за те, як буде проходити вибірка даних які мають прив’язку до поточної таблиці. Параметр має два значення:

- Lazy – дані вибираються «лінивим» способом, а саме, коли вкладений об’єкт стає потрібен, він автоматично завантажується з БД. У великих проектах, це сильно може розвантажити систему, за рахунок оптимізації таким чином, такий спосіб споживає менше оперативної пам’яті, але вимагає більше ресурсів процесора, так як в будь-який момент може знадобитись завантаження додаткових даних;

- Eager – так зване «жадібне» завантаження даних. Це означає що всі дані автоматично завантажуються при завантаженні батьківського класу. Функція дуже корисна у випадку коли потрібно отримати все і зразу без додаткових запитів до БД, але платою за це є використання додаткової оперативної пам'яті, яка в деякі моменти буває дуже цінною, тому не варто зловживати даною можливістю.

У базі даних в загалному було створено 16 таблиць, деякі з них сервісні (їх використовують бібліотеки для своїх службових функцій, та для обслуговування інших таблиць), а всі інші, використовуються як основні, на них побудована структура проекту.

Одною з найважливіших таблиць у проекті, є таблиця «user», в якій зберігаються основні дані, такі як: логін, пароль (який зберігається у зашифрованому вигляді), ел. адреса, та ім’я. Нижче наведений SQL скрипт створення таблиці.

```
create table user (
  id     bigint not null,
  active   INTEGER,
  email   varchar(32),
  first_name varchar(64),
  password  varchar(512),
  username  varchar(16),
  primary key (id)
) engine = InnoDB;
```

SQL скрипт – написаний на мові SQL набір команд, в даному випадку для створення таблиці у базі даних, частіше всього використовується для створення міграцій баз даних, а також для автоматичного введення стартових даних.

В скрипті описане створення таблиці user з полями:

- id – ідентифікатор, по якому буде здійснюватись вибірка полів;
- active – булеве значення, в якому зберігається статус активності облікового запису;
- email – електронна адреса користувача;
- first_name – ім’я користувача;

- password – пароль;
- username – логін користувача, який використовується для входу на сайт.

Довжина колонки паролю має максимальний розмір 512 символів, тому що паролі зберігаються у базі даних у зашифрованому вигляді. Для шифрування використовується алгоритм bCrypt, що дозволяє забезпечити високий рівень безпеки, пароль з цим шифрування неможливо розшифрувати звичайними способами.

Не менш важливою таблицею у БД є таблиця user_role:

```
create table user_role (
  user_id bigint not null,
  roles  varchar(255)
) engine = InnoDB;
```

Таблиця містить в собі лише дві колонки:

- user_id – id користувача з таблиці «user»;
- role – роль користувача. Потреба зберігати ролі в БД відсутня, так як всі ролі, які потрібні для функціонування сервісу записані в спеціальному класі, це допомагає максимально зберегти сумісність і цілісність всіх компонентів.

Таблиця, за допомогою якої виконується групування по категоріях всіх підкатегорій, використовується вона тільки для цього, так як самі товари в свою чергу посилаються на таблицю з підкатегоріями, і визначити категорію товару можна лише знаючи в якій підкатегорії його можна знайти. Таблиця доволі скромна по кількості полів, а також по загальній кількості записів в ній не буде записано багато даних.

Скрипт створення таблиці category:

```
create table category(
  id     bigint not null,
  image_name varchar(255),
  name    varchar(20),
  primary key (id)
) engine = InnoDB;
```

У таблиці окрім стандартних текстових полів присутнє одне досить незвичайне поле, яке є посиланням, але не на запис у таблиці, а на файл у файловій системі. Опис колонок таблиці:

- id – ідентифікатор, автоінкрементується при додаванні нового запису;
- image_name – колонка в якій зберігається назва зображення, так як кожна категорія має іконку. Сама іконка зберігається у файловій системі, а її назва генерується за допомогою UUID, тому максимальний розмір колонки саме 255 символів;
- name – ім’я категорії, має не великий розмір так як мета категорії коротко та чітко визначити призначення групи товарів, а ще на користувацькій частині повинно бути достатньо місця для імені.

Таблиця subcategory напряму залежить від попередньої таблиці, так як кожен запис в цій таблиці посилається на запис у таблиці category, при цьому сама таблиця має додаткові зв’язки з іншими таблицями. За допомогою даної таблиці групуються товари, тому досить важливо було правильно визначити її вміст, та зв’язки. Скрипт створення таблиці subcategory:

```
create table subcategory(
  id     bigint not null,
  name    varchar(32),
  category_id bigint,
  primary key (id)
) engine = InnoDB;
```

Опис колонок таблиці:

- id – ідентифікатор категорії. Використовується у таблиці товарів, для групування по підкатегоріях;
- name – назва підкатегорії. Повинна відповідати логічному вмісту категорії до якої і відноситься;
- category_id – зовнішній ключ, посилання на таблицю category, посилається на один запис у таблиці.

Другою по значенню в базі даних, після таблиці користувачів, є таблиця product – основний об’єкт для якого і існують більшість таблиць, це як таблиці для групування записів з цієї таблиці, так і замовлення, кошик та інші таблиці.

Найбільш завантажена в плані зв‘язків таблиця, так як містить досить багато специфічних посилань, які використовуються у всіх станах товару, будь то товар у кошику, чи замовлений товар, всі вони посилаються на цю таблицю, тому цілісність даних тут дуже важлива. Скрипт створення даної таблиці:

```
create table product (
  id              bigint       not null,
  active            bit        not null,
  description         varchar(512),
  image_name          varchar(255),
  name              varchar(64),
  orders_count         bigint      not null,
  price_for_measurement_unit double precision not null,
  measurement_unit_id    bigint      not null,
  seller_id          bigint,
  subcategory_id       bigint      not null,
  primary key (id)
) engine = InnoDB;
```

Вище наведені колонки, досить важливі для багатьох процесів у проектів, і кожна колонка має своє не тільки інформаційне, а й логічне призначення. Опис колонок таблиці:

- id – ідентифікатор товару, використовується частіше для вибірки одного запису з таблиці;
- active – індикатор активності товару, має лише два значення. Використовується для можливості деактивувати товар або зупинити продаж;
- description – поле вміщує в себе опис товару, але не є обов’язковим, так як оптовій торгівлі часто продаються товари, які не вимагають додаткового опису, і робити поле обов’язковим немає сенсу, так як це збільшило б кількість інформації яка в багатьох випадках не принесла б користі, але при цьому займала б місце у базі даних, хоч і не багато;

- image_name – назва зображення розташованого у каталозі на диску;
- name – ім’я товару, основне, за чим користувачі будуть розрізняти товари на сайті;
- orders_count – кількість замовлень товару. Поле за допомогою якого продавець може дізнатись скільки разів був замовлений його товар. Також дана колонка використовується для сортування товарів по популярності, чим більша кількість замовлень у товару, тим він популярніший;
- price_for_measurement_unit – в даній колонки міститься ціна за одну одиницю вимірювання товару, наприклад 10 (грн) за один кілограм. Таке точне визначення ціни за одну одиницю допомагає створити зручну та гнучку систему продажу та купівлі товарів, так як користувач точно знає скільки і в яких одиницях вимірювання він продав, так і покупець при виборі товару бачить за що і скільки він платить;
- measurement_unit_id – ідентифікатор з таблиці measurement_unit, посилається на один запис в таблиці, який відображає тип одиниці вимірювання товару, її назву та скорочення (кг, гр, шт. тощо);
- seller_id – ідентифікатор до запису у таблиці seller. Посилається на користувача який має статус продавця який продає даний товар. При цьому, не кожен користувач є продавцем. Це розділення зроблено для того, щоб розгрузити таблицю продавців, та користувачів, і зменшити час пошуку товарів від одного продавця, так як не всі користувачі на сайті будуть продавцями, відповідно і записів у таблиці seller буде значно менше ніж у таблиці user;
- subcategory_id – ідентифікатор категорії за якою закріплений товар.

Таблиця cart_items являє собою кошик користувача, в якому зберігаються всі замовлені товари та їх кількість. Скрипт створення таблиці:

```
create table cart_items
(
  id       bigint not null,
  quantity    integer not null,
  is_add_to_order bit   not null,
  user_id     bigint,
  product_id   bigint not null,
  primary key (id)
) engine = InnoDB;
```

Опис полів таблиці:

- id – ідентифікатор запису (товару у кошику);
- quantity – кількість вибраного товару у кошику;
- is_add_to_order – булеве значення, яке дозволяє вибрати який товар з кошика додавати у замовлення, а який ні, використовується для вибору у користувацькому інтерфейсі;
- user_id – ідентифікатор користувача. За допомогою цього поля здійснюється вибірка товарів у кошику для кожного користувача;
- product_id – товар, який користувач додає у кошик, його кількість, яка визначається у колонці quantity.

Таблиця з одиницями вимірювання measurement_unit, в якій зберігаються всі одиниці вимірювання товарів, у кожному товарі присутнє посилання на дану таблицю, так як кожен товар вимірюється в якихось одиниця (кілограми, грами, літри, штуки тощо). Таблиця створюється за допомогою наступного SQL скрипта:

```
create table measurement_unit (
     id     bigint not null,
     full_name varchar(20),
     short_name varchar(16),
     primary key (id)
) engine = InnoDB;
```

Опис колонок таблиці:

- id – ідентифікатор запису в таблиці, автоінкрементується при додаванні запису;
- full_name – повне ім’я одиниці вимірювання, наприклад: кілограм, літр;
- short_name – скорочене ім’я одиниці вимірювання, наприклад: кг.(кілограм) , л.(літр).

Таблиця seller. Використовується для оптимізації пошуку товарів по продавцях, за рахунок того що відділена від таблиці користувачів, має менше записів і відповідно при великій кількості користувачів які продають товари, скорочує час пошуку, і дозволяє в подальшому вести облік та статистику продажу у продавців а також додати інший функціонал, SQL скрипт створення таблиці:

```
create table seller (
       id   bigint not null,
       name  varchar(255),
       user_id bigint,
       primary key (id)
) engine = InnoDB;
```

Опис колонок таблиці:

- id – ідентифікатор продавця;
- name – ім’я продавця, в даний момент воно таке ж як і ім’я користувача, використовується для розпізнавання продавця по імені, допомагає користувачам краще орієнтуватись в тому, що і у кого він купив;
- user_id – ідентифікатор користувача який продає товари (користувач стає продавцем коли виставляє хоча б один товар);

Таблиця orders, у якій зберігаються замовлення, та детальна інформація про замовлення. Призначення даної таблиці – надання можливості залишити замовлення на вибрані товари, прикріпивши до цих даних адресу доставки, з можливістю відслідковування статусів замовлення. SQL скрипт створення таблиці orders:

```
create table orders (
  id      bigint      not null,
  comment    varchar(512),
```

```
  creation_date datetime,
  is_canceled  bit       not null,
  total_price  double precision not null,
  user_id    bigint,
  seller_id   bigint,
  primary key (id)
) engine = InnoDB;
```

Опис колонок таблиці:

- id – ідентифікатор, який виступає номером замовлення, також може використовуватись для орієнтування що і до якого замовлення належить;
- comment – коментар до замовлення. Може містить будь-яку додаткову інформацію до замовлення, яку клієнт вважав за потрібне вказати;
- creation_date – дата та час створення замовлення (у форматі timestamp);
- is_canceled – визначає чи скасовано замовлення, визначається в дані таблиці так як при скасуванні замовлень не можливо додати ніякі статуси до замовлення;
- total_price – загальна сума замовлення;
- user_id – ідентифікатор користувача який розмістив замовлення;
- seller_id – ідентифікатор продавця, який обробляє створене замовлення.

У замовленні окрім загальної інформації присутні не менш важливі дані, такі як замовлені товари, ці записи зберігаються у таблиці ordered_items, в таблиці зберігається інформація про замовлені товари, їх кількість та ціна. SQL скрипт таблиці:

```
create table ordered_item
(
  id     bigint      not null,
  quantity  integer     not null,
  total_price double precision not null,
  product_id bigint      not null,
  order_id  bigint,
  primary key (id)
) engine = InnoDB;
```

Опис колонок таблиці:

- id – ідентифікатор замовленого товару;
- quantity – кількість замовленого товару;
- total_price – кінцева ціна за вказану кількість товару. Дане значення потрібно зберігати у цій таблиці, так як з часом ціна товару може мінятись, і цілісність даних може бути порушена, тому ціна повинна бути зафіксованою;
- product_id – ідентифікатор замовленого товару;
- order_id – ідентифікатор замовлення, до якого відноситься замовлений товар.

Адреса доставки також присутня у кожному замовлені, але специфіка адрес користувачі така, що дозволяє змінювати дані адреси, і єдиним виходом в даній ситуації є дублювання даних адреси користувача у момент створення замовлення. Для користувача процедура створення адреси виконується лише раз для однієї адреси, і йому не потрібно кожного разу переписувати всі дані. Кожен користувач може мати декілька адрес, і при створення замовлення може як створити нову, так і вибрати адресу з існуючих, це дозволяє зменшити час на оформлення замовлення в декілька разів. SQL скрипт створення таблиці з адресами користувачів наведений нижче. Таблиця містить всі адресні дані, необхідні для відправлення товарів навіть у інші міста.

```
create table shipping_address (
  id           bigint not null,
  building_number     varchar(10),
  city          varchar(36),
  office_number      varchar(10),
  post_code        varchar(6),
  recipient_full_name   varchar(36),
  recipient_mobile_number varchar(10),
  region         varchar(36),
  street         varchar(36),
  user_id         bigint,
  primary key (id)
) engine = InnoDB;
```

Опис полів таблиці shipping_address:

- id – ідентифікатор адреси;
- building_numbe – номер будівлі куди потрібно доставити замовлення;
- city – місто доставки;
- office_number – не дивлячись на назву, може бути як номером офісу так і номером квартири доставки товару;
- post_code – поштовий індекс. Потрібен для відправки поштою, або для простішого орієнтування в регіоні, в який потрібно відправити посилку;
- recipient_full_name – П. І. Б. отримувача замовлення. Хоч адреса і прив’язана до облікового запису користувача, він може вказати отримувача замовлення, це корисно якщо користувач розміщує замовлення на когось іншого, і після підтвердження, продавець зможе відправити замовлення і вказати отримувача по цим даних;
- recipient_mobile_number – номер мобільного отримувача. По даному номеру продавцем буде здійснюватись підтвердження замовлення;
- region – область куди буде відправлено замовлення;
- street – вулиця на якій розміщений вказаний вище будинок;
- user_id – ідентифікатор користувача який додав адресу.

Таблиця яка виконує ту ж функцію, тільки замість користувача прив’язана до замовлення має назву order_addresses, SQL скрипт створення якої наведений нижче:

```
create table order_addresses (
  id           bigint not null,
  building_number     varchar(10),
  city          varchar(36),
  office_number      varchar(10),
  post_code        varchar(6),
  recipient_full_name   varchar(36),
  recipient_mobile_number varchar(10),
  region         varchar(36),
  street         varchar(36),
  order_id        bigint,
  primary key (id)
) engine = InnoDB;
```

Єдина відмінність цієї таблиці це те, що в даній таблиці поле user_id замінено на order_id, тому що записи в даній таблиці вже не відносяться до користувача, а відносяться до замовлення, і не підлягають зміні, і можу бути видалені лише у випадку видалення замовлення;

Кожне замовлення від створення до виконання має певні статуси, які відображають його поточний стан (не підтверджено, відправлено і т. д.), та дозволяють як продавцю так і покупцеві орієнтуватись на якій стадії замовлення, і при необхідності чинити якісь дії по відношенню до стану замовлення (підтвердити або відхилити замовлення, у випадку продавця, або підтвердити отримання у випадку покупця).

Статуси замовлень зберігаються у таблиці stats, SQL скрипт створення якої наведений нижче:

```
create table stats(
  id     bigint not null,
  code    varchar(255),
  description varchar(255),
  name    varchar(255),
  primary key (id)
) engine = InnoDB;
```

Опис колонок таблиці stats:

- id – ідентифікатор статусу;
- code – код статусу, по якому у коді програми здійснюється орієнтування по поточному статусу, та який статус потрібно присвоїти;
- description – опис статусу. У даній колонці може бути детально описано що означає статус;
- name – назва статусу, яка буде відображатись користувачеві.

Всі статуси пов’язані між собою логічним чином, і мають послідовність яка визначена в коді програми, той чи інший статус може бути призначений

тільки в певному випадку (наприклад тільки після того як в замовлення буде статус, який повинен бути лише перед статусом який потрібно присвоїти).

Для відслідковування статусу замовлень, упорядкування цих статусів, та додавання додаткової інформації використовується таблиця order_stats, в ній статуси присвоюються до конкретних замовлень, а також є можливість додати супутню інформацію (наприклад номер накладної після відправки замовлення). SQL скрипт створення таблиці:

```
create table order_stats (
  id       bigint not null,
  status_comment varchar(255),
  status_set_date datetime,
  order_id    bigint not null,
  status_id    bigint not null,
  primary key (id)
) engine = InnoDB;
```

Опис колонок таблиці:

- id – ідентифікатор статусу замовлення;
- status_comment – коментар до статусу. Продавець або покупець, підчас присвоєння певних статусів до замовлень, можуть додати коментар (наприкла продавець може вказати причину чому було відхилено замовлення);
- status_set_date – дата встановлення статусу. Дає можливість користувачам зрозуміти коли сталась та чи інша подія по відношенню до конкретного замовлення;
- order_id – ідентифікатор замовлення, до якого додається статус;
- status_id – ідентифікатор статусу з таблиці stats.

Всі таблиці у БД тісно пов’язані між собою, хоча деякі з них і можна було б оптимізувати об’єднавши, але це б привело до некоректної і непередбачуваної роботи додатку, я вважаю, що всі рішення, прийняті по відношенню формування та проектування структури бази даних прийняті правильно.

2.3 Визначення типу зв’язку між сервером та клієнтом додатку

Сучасні сайти усіх типів, будь то платіжна система, чи сайт візитка з мінімальним функціоналом, мають декілька типів зв’язку фронтенду з бекендом, самий простий з них це коли обробкою та формуванням сторінки займається сервер, а точніше шаблонізатор. Цей спосіб є доволі поширеним але втрачає популярність тому, що сучасні комп’ютери досить потужні, і можуть обробити інформацію яку їм потрібно відобразити самостійно, на відміну від комп’ютерів які були приблизно 18 років назад. Мінусом способу є те, що все бере на себе сервер, і це вимагає потужніших серверів яким потрібно робити те, що може робити користувацький пристрій.

Виходячи з мінусів першого способу, виник інший – REST, основним принципом якого є передача даних у невеликій кількості у стандартизованих форматах (наприклад JSON).

Мій проект розроблений по принципу RESTFull додатку. Це означає, що клієнтська і серверна частина є незалежними і нічим не пов’язані, це дозволяє розмістити їх на різних серверах а також гнучко масштабувати.

У дипломному проекті для автентифікації та авторизації було використано технологію, яка базується на використанні JWT токенів для авторизації, та автентифікації користувачів за допомогою спеціального символьного ключа (токена) який генерується на певний період, і містить в собі зашифровані дані [30]. Принцип роботи полягає в тому, що при вході в систему, користувачеві надсилається спеціальний HTTP заголовок, в якому міститься згенерований сервером код, після чого на клієнтській частині записується у локальне сховище браузера, і при надсиланні будь-якого запиту до серверу, прикріплюється в спеціальне поле із заголовком «Authorization». Якщо термін дії токену не вийшов – сервер авторизує користувача, і дає доступ до потрібних даних якщо у користувача достатньо прав.

Приклад реалізації запиту реєстрації на сервері:

```
@PostMapping("/sign-up")
public void signUp(@Valid @RequestBody User user) throws
```

```
UserRegistrationFailedException {
          userService.createUser(user, passwordEncoder);
        }
```

У коді вище наведено посилання, та метод який приймає об’єкт User, при цьому валідний (всі поля об’єкту повинні відповідати налаштованій валідації). Анотацією «@PostMapping» позначається те, що метод доступний лише для типу HTTP методу POST. Типи HTTP методів розроблені для того, щоб по одній і тій ж адресі, можна було відправляти запити з різними призначеннями, наприклад: delete (видалення), put (редагування).

Після того, як метод буде успішно виконано, сервер поверне статус з кодом HTTP кодом 200 – це означає що запит успішний і ніяких помилок не виникло. Таким чином працюють HTTM запити до сервера, в результаті яких повертаються дані, або ж нічого, і у випадку виникнення помилок сервер поверне код і повідомлення що привело до помилки.

2.4 Розробка структури інтерфейсу користувача

Структура сайту є важливим елементом оформлення, так як напряму впливає на зручність користування сайтом, а також на сприйняття користувачем вмісту. У моєму проекті сприйняття контенту сторінки є важливим, для цього потрібно забезпечити логічне розміщення кнопок, елементів, і зробити перехід між сторінками інтуітивним. Користувач повинен без підказок і схем розуміти що, і на якій сторінці він зможе знайти, це зробить навігацію швидкою і зменшить шанс заплутатись між сторінками.

Структура сторінок сайту розробляється для того, щоб зпланувати яким чином розмістити сторінки і переходи між ними, а також в процесі проектування схеми, придумуються ці самі сторінки та їх призначення.

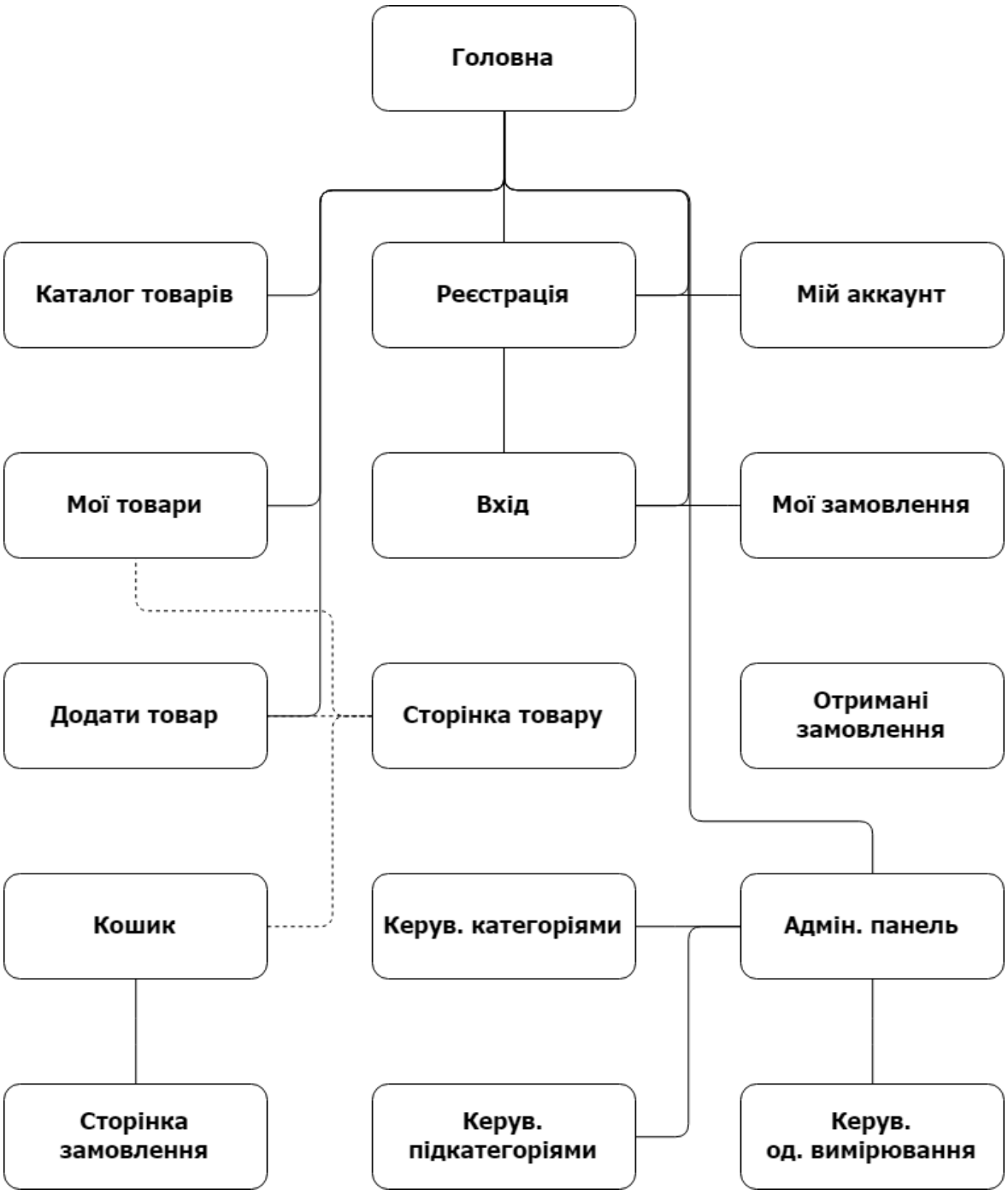

Рисунок 2.2 – Структура сайту

На рис. 2.2 зображена структура сайту, на якій зображено більшість сторінок сайту, а також лінії, які означають можливість переходу між цими сторінками. На деякі сторінки можна перейти майже з будь-якої сторінки, на інші ж можна перейти тільки з однієї сторінки.

Наприклад на сторінку замовлення можна перейти лише у випадку, якщо користувач знаходиться у кошику.

3 РОЗРОБКА ОСНОВНИХ КОМПОНЕНТІВ, МЕТОДІВ ТА ОГЛЯД ІНТЕРФЕЙСУ КОРИСТУВАЧА

3.1 Програмні та апаратні вимоги

Дипломний проект розділено на дві частини, які взаємодіють між собою, але при цьому знаходяться на різних системах і пристроях, тому і мінімальні вимоги у них різні, так як у клієнтської частини мінімальна вибагливість до ресурсів системи.

Для роботи серверу додатку, на комп'ютері (або на іншій машині), де буде розгортатись сервер перш за все повинна бути встановлена віртуальна машина Java (JVM), за допомогою якої і буде виконуватись файл серверу. Також робота сервера неможлива без СУБД, так як я використав MySQL, на системі де буде розгортатись сервер потрібно встановити MySQL Server 5.7. Інших програмних засобів сервер не вимагає, спеціальної ОС для сервера не потрібно, так як додатки, написані на мові Java працюють на будь-якій платформі де підтримується JVM.

Як і будь-який сервер додатку – мій сервер потребує апаратних ресурсів, але їх важко точно визначити, тому що вони залежать від кількості клієнтів, кількості одночасних замовлень, пошукових запитів та інших факторів. Мінімальною конфігурацією сервера є як мінімум 2-х ядерний процесор, 1 гігабайт оперативної пам'яті та хоча б 20 гігабайт жорсткого диску або SSD диску, який є більш бажаним так як значно пришвидшує роботу з дрібними файлами. Вказана конфігурація не є самим мінімумом, так як сервер запрацює і на одноядерному процесорі з 128 мегабайтами оперативної пам'яті, але при цьому можуть бути великі затримки. Частота процесору не є параметром, так як по частоті не можна сказати наскільки продуктивний процесор.

Для користувацької частини апаратні вимоги не є суттєвими, так як потужності будь-якого смартфону навіть 5-річності давності достатньо для

відобразити будь-яку сторінку сайту, але при цьому все гострим питанням є програмна підтримка технологій, які були використані, так як деякі старі смартфони перестали підтримувати вже дуже давно, але технології міняються не так часто, тому для використання сайту буде достатньо будь-якого пристрою який підтримує хоча б ECMAScript 5 (Javascript версії 5, представлена у 2009 році), а також HTML5. Весь JavaScript код, а точніше деякі його вбудовані функції, перетворюються у код сумісний з ECMAScript, так як деякі браузери досі не підтримують частину функціонала ECMAScript 6, і єдиним виходом з ситуації є конвертування нових можливостей мови у старий формат, який використовується все менше і менше.

3.2 Розробка способу авторизації та аутентифікації користувача

Дипломний проект, а саме його серверна частина, та її зв'язок з клієнтською, побудований на принципі REST що передбачає обмін даними з клієнта у форматі коротких HTTP запитів. Одним з основних компонентів серверу є система безпеки, яка забезпечує доступність даних тільки тим користувачам, для яких вона призначена. У проекті в якості базової системи безпеки використовується Spring Security. Всі запити та дії перевіряються, кожен запит повинен пройти всі фільтри системи безпеки.

Для авторизації та аутентифікації користувача на сервері, я розробив декілька нових, та вдосконалив існуючі методи, для того щоб реалізувати систему безпеки за допомогою JWT токенів.

Для реалізації аутентифікації в обхід стандартним засобам Spring Security, я створив новий клас, та перевизначив базовий фільтр, який використовується для входу у систему:

```
public class JWTAuthenticationFilter extends UsernamePasswordAuthenticationFilter
```

Фільтр відповідає за вхід у систему за допомогою імені користувача та паролю. Моїм завданням було розширити його функціонал з простого входу, до входу у систему та подальшому відправленні токену користувачеві.

Для додавання токену у запит після успішного входу я використав метод successfulAuthentication, який викликається у випадку успішного входу у систему. Для генерації токену була використана спеціальна бібліотека, для роботи з JWT токенами. Також я створив додатковий клас, для розмежування процедури створення токену з процедурою його отримання та перевірки. Метод successfulAuthentication при виклику створює об’єкт класу JWTUtil, призначений для роботи з токеном:

```
JWTUtil jwtUtil = new JWTUtil();
```

Після цього викликається метод generateToken який присутній у вище створеному класі:

```
String token = jwtUtil.generateToken(auth);
```

Результат роботи методу записується в рядок. В результаті роботи, метод повинен повернути готовий JWT токен. Код методу jwtUtil.generateToken():

```
public String generateToken(Authentication auth) {
      final String authorities = auth.getAuthorities().stream()
            .map(GrantedAuthority::getAuthority)
         .collect(Collectors.joining(","));
      return JWT.create()
            .withSubject(((User) auth.getPrincipal()).getUsername())
            .withClaim("roles", authorities)
            .withExpiresAt(new Date(System.currentTimeMillis() + EXPIRATION_TIME))
         .withIssuedAt(new Date(System.currentTimeMillis()))
         .sign(HMAC512(SECRET.getBytes()));
}
```

Спочатку для того щоб розмістити та зашифрувати ролі користувача, вони збираються в один рядок (константа authorities). Потім за допомогою бібліотеки auth0 та класу jwt створюється токен з такими параметрами (перелічені в порядку оголошення):

- withSubject – у метод передається основний об’єкт, в даному випадку ім’я користувача;

- withClaim – додаю додаткову інформацію у токен, яка буде необхідна на користувацькій частині проекту;
- withExpiresAt – вказую час через який токен буде не дійсним (в мілісекундах). В зразку коду береться поточна дата та до неї додається значення з константи EXPIRATION_TIME яка містить час в мілісекундах на який користувачеві надається доступ;
- withIssuedAt – встановлюється час надання доступу (або створення токену);
- sign – вказую сигнатуру, по якій буде зашифрована інформація, являється конфіденційною інформацією, яка не повинна виходити за межі сервера.

Після створення, токен передається у попередній метод, де був викликаний, після цього додається у заголовок «Authorization»:

```
res.addHeader(HEADER_STRING, TOKEN_PREFIX + token);
```

Назва заголовку, префікс токену та інша інформація яка потрібна для глобального поширення у класах додатку, додана у вигляді констант. Я використав саме такий підхід, тому що глобальні константи легко використовувати в будь-якій частині коду, без оголошення класу та інших змінних. Перелік констант які використовуються у всьому циклі створення та перевірки токену:

```
public static final String SECRET = "йцвйца213124цйвйцайцв";
public static final long EXPIRATION_TIME = 1_284_012_568;
static final String TOKEN_PREFIX = "Bearer ";
static final String HEADER_STRING = "Authorization";
```

Кожна константа має своє призначення, і якщо змінити будь-яку з них, система безпеки вже не буде працювати з попередньо згенерованими токенами. У проекті існують такі константи для забезпечення безпеки:

- SECRET – секретний ключ шифрування, який використовується для шифрування даних у токені;
- EXPIRATION_TIME – час дії токену. Вказується у мілісекундах і визначає як довго буде діяти токен. Якщо час валідності токену вийде,

користувач втратить доступ до облікового запису і йому потрібно буде щераз увійти в систему;

- TOKEN_PREFIX – префікс токену (слово яке буде вставлено безпосередньо перед самим шифром токену);
- HEADER_STRING – назва HTTP заголовку, в який буде вставлений токен. По замовчуванню для цього виділений спеціальний заголовок «Authorization»;

Також не менш важливо було розробити спеціальний фільтр для перевірки токену при кожному запиті. Для цього я використав ще один базовий фільтр Spring Security який має назву BasicAuthenticationFilter. Він відповідає за підтвердження даних користувача при будь-якій дії у системі.

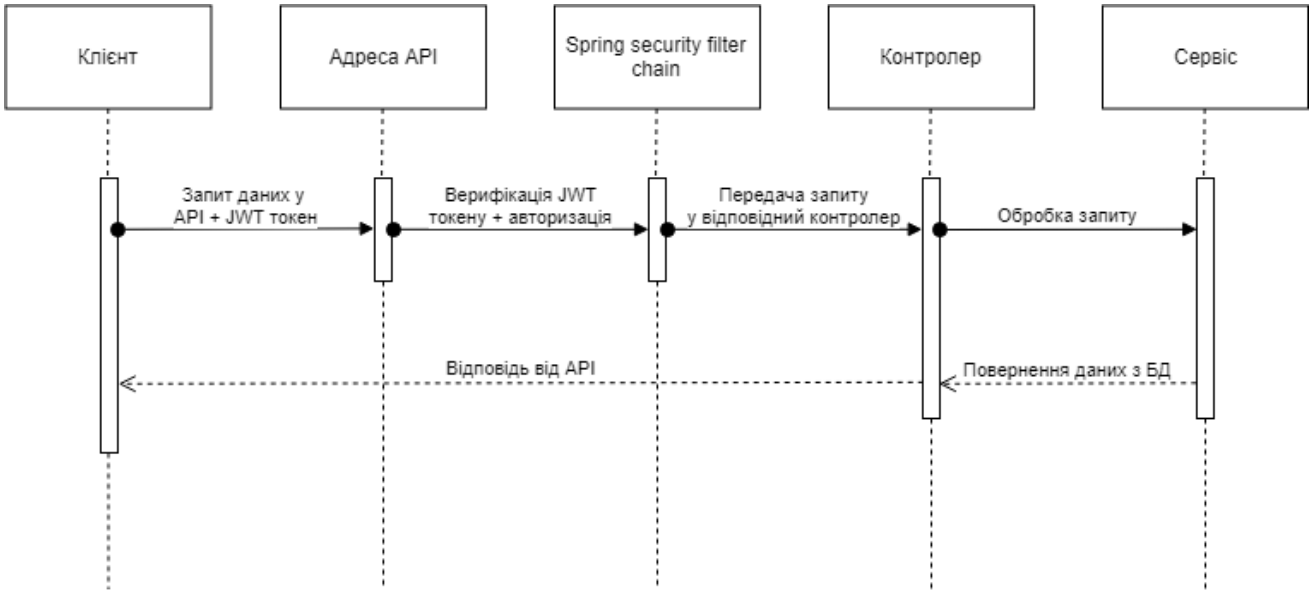

Рисунок 3.1 – UML діаграма процесу авторизації

На діаграмі, зображеній на рис. 3.1 відображений принцип роботи з JWT токенами, а також процедуру верифікації токену у кожному запиті до сервера. В процесі надсилання кожного запиту до API перед обробкою, запит проходить всі фільтри безпеки, після чого запит передається на контролер, після чого виконується обробка, вибірка даних з БД та відправка відповіді клієнту.

Я розширив даний фільтр, та перевизначив його основний метод doFilterInternal. Метод повинен виконати завдання фільтра, після чого продовжити ланцюг фільтрів.

Я переписав даний метод таким чином, щоб при кожному запиті який буде проходити через фільтр, із запиту брався заголовок Authorization:

```
String header = req.getHeader(HEADER_STRING);
```

Після спроби отримання заголовку йде перевірка чи заголовок існує:

```
if (header == null || !header.startsWith(TOKEN_PREFIX)) {
      chain.doFilter(req, res);
      return;}
```

Якщо заголовку знайдено не буде (у цьому випадку рядкова змінна буде мати значення null) фільтр передасть запит наступному фільтрі у ланцюгу, але при цьому користувач залишиться на авторизованим, і якщо будуть потрібні додаткові привілеї – користувачеві буде відмовлено у доступі.

Наступним кроком при вдалій перевірці заголовку, є перевірка токену та отримання даних користувача:

```
UsernamePasswordAuthenticationToken authentication = getAuthentication(req);
```

Перевірку виконує метод getAuthentication, який виконує перевірку сигнатури, та дати валідності токену:

```
DecodedJWT decoded = JWT.require(Algorithm.HMAC512(SECRET.getBytes()))
      .build()
   .verify(token.replace(TOKEN_PREFIX, ""));
```

Токен розшифровується за допомогою раніше використаного секретного ключа. Також відбувається перевірка дати. Якщо не виникає ніяких помилок, то з токену беруться всі потрібні дані, такі як ім’я користувача та ролі:

```
String username = decoded.getSubject();
Claim roles = decoded.getClaim("roles");
```

Це є обов’язковою умовою для правильного розприділення ролей між користувачами на сервері, при цьому не прибігати до перевірки ролей у базі даних. Отримані дані перевіряються на присутність:

```
if (username != null && roles.asString() != null) {
      return      new      UsernamePasswordAuthenticationToken(username,      null,
parseRolesFromTokem(roles.asString()));
} return null;
```

При присутності всіх даних повертається спеціальний об’єкт Spring Security які містить всі дані користувача в системі, такі як: логін, пароль та ролі.

Якщо якісь дані відсутні чи пошкоджені і не можуть бути прочитані, метод повертає null, якщо дане значення буде вставлено в контекст безпеки:

```
SecurityContextHolder.getContext().setAuthentication(authentication);
```

Тоді користувач буде вважатись як неавторизованим. Якщо ж в контекст безпеки буде вставлено користувача з валідними даними та ролями, користувач буде розцінюватись як авторизований, і йому будуть надані всі дозволи, які прописані в його ролях.

```
chain.doFilter(req, res);
```

Після всіх маніпуляцій фільтр передає запит до наступного фільтру та завершує свою роботу.

3.3 Розробка методу реєстрації користувача

Користувачі у проекті є основним керуючим механізмом, більшість дій які виконуються над товарами, замовленням і на сервісі в цілому. Саме тому реєстрація користувачів на проекті є дуже важливою процедурою, яку я старався розробити максимально простою та при цьому надійною і ефективною, без перегружених методів які б вимагали багато ресурсів сервера додатку.

Реєстрація користувача на серверній частині починається у класі AuthController, з методу signUp:

```
@PostMapping("/sign-up")
public void signUp(@Valid @RequestBody User user) throws UserRegistrationFailedException {
    userService.createUser(user, passwordEncoder);}
```

Метод виконується коли з клієнтської частини приходить запит по адресі [адреса до сервера]/api/auth/sign-up з HTTP методом POST, також в тілі запиту повинен бути присутнім об’єкт користувача з усіма заповненими даними. Після цього, об’єкт передається у метод createUser(), який знаходиться у спеціальному

класі-сервісі з назвою UserService. У даному сервісі виконуються всі дії над об’єктами, робиться це для того, щоб розмежувати функціонал між контролером і сервером.

Метод createUser приймає об’єкт класу User а також passwordEncoder (спеціальна утиліта від Spring, яка дозволяє шифрувати та перевіряти паролі). Код методу виглядає наступним чином:

```
public void createUser(
            User user,
            PasswordEncoder passwordEncoder
            ) throws UserRegistrationFailedException {
      if              (!userRepository.existsByUsernameOrEmail(user.getUsername(),
user.getEmail())) {
            if (user.getPassword().length() <= 20) {
          user.setPassword(passwordEncoder.encode(user.getPassword()));
             user.setActive(true);
             user.setRoles(Collections.singleton(Role.USER));
                  userRepository.save(user);
         } else throw new IllegalArgumentException("Maximum password length is 20
characters");
      } else throw new UserRegistrationFailedException("Failed to register new
user, because username or email already exist."); }
```

Отримавши вхідні дані, метод за допомогою репозиторія даних перевіряє чи існує вже такий користувач з таким іменем та емейлом:

```
userRepository.existsByUsernameOrEmail(user.getUsername(), user.getEmail())
```

Якщо існує користувач хоча б з одним схожим параметром – тоді реєстрація неможлива, і на клієнтську частину відправляється відповідне повідомлення. Якщо ж користувача з такими даними немає, користувачеві додається пароль який перед цим шифрується за допомогою раніше вказаного passwordEncoder, за допомогою методу encode():

```
user.setPassword(passwordEncoder.encode(user.getPassword()));
```

Далі користувачеві присвоюється id null, та статус active:

```
user.setId(null);
user.setActive(true);
```

Це робиться для того, щоб захистити цілісність даних, адже метод save() виконує не тільки вставку даних а ще й оновлює, і якщо в запиті прийдуть дані

користувача з вказаним id – можлива втрата даних та перезапис, тому що якщо передати в метод save() об’єкт з вказаним ідентифікатором, Hibernate спробує знайти і оновити вказаний запис, а при реєстрації це не потрібно.

Після того як всі дані вставлено і відредаговано, виконується метод userRepository.save(user) який виконує збереження користувача у базу даних, після чого на клієнтську частину відправляється повідомлення про успішне виконання запиту.

3.4 Розробка методу вибірки та фільтрування товарів

Як і у будь-якому інтернет-магазині, чи сервісі, у моєму проекті для зручного відображення товарів у бажаній послідовності, наприклад по зростанню чи спаданню ціни, по популярності, мінімальній чи максимальній ціни я розробив спеціальний метод який в залежності від параметрів повертає ту чи іншу послідовність товарів. Метод написаний за допомогою query параметрів, що допомогло досягти бажаного результату, і при цьому зберегти зручність та зрозумілість реалізації.

Query параметри – це параметри, які перелічуються безпосередньо в URL. В параметрів є стандартизований вигляд. Оголошення параметрів починається із знаку питання і всі наступні параметри розділюються за допомогою знаку амперсанта у форматі:

```
[адреса]/[шлях]?параметр1=значення1&параметр2=значення2
```

Параметри бувають як обов’язковими так і ні. Це дозволяє гнучко реалізувати сортування, групування та інші можливості за допомогою query параметрів. Також ще одним не менш великим плюсом цих параметрів є те, що скопіювавши адресу сторінки з такими параметрами, і вставивши її на інакшому – відкриється та ж сама сторінка з тими ж параметрами пошуку, сортування та всім що буде записано як URL параметр.

Метод який відповідає за обробку запитів, які відносяться до отримання списку товарів з певними параметрами:

```
@GetMapping
public Page<Product> getProducts(
            @RequestParam(name = "p") int page,
            @RequestParam(name = "s") int size,
            @RequestParam(name = "sc") Long subcategoryId,
            @RequestParam(name = "minPrice", required = false) Double minPrice,
            @RequestParam(name = "maxPrice", required = false) Double maxPrice,
            @RequestParam(name = "sortType", required = false) String sortType
        ) throws NotFoundException {
          return  productService.getProducts(page,  size,subcategoryId,  minPrice,
maxPrice, sortType);
 }
```

У методі перераховані вхідні параметри, деякі з них є обов'язковими, а деякі можна не вказувати. Оголошення параметру починається із спеціальної анотації @RequestParam, це означає що вхідний параметр є параметром який розміщений в URL, це допомагає Spring зрозуміти звідки потрібно взяти значення цього параметру. Наступним йде параметр анотації «name», який вказує назву параметру в URL, в параметру можуть відрізнятись імена в коді і в URL, але якщо не вказати цей параметр, Spring буде шукати параметр з таким ж іменем як оголошено в коді. Також можна вказати чи параметр є обов'язковим, якщо параметр не обов'язковий то не знайшовши його в URL запиту, Spring проігнорує його і продовжить роботу. Якщо ж параметр обов'язковим і його немає в рядку запиту, буде видане виключення і метод завершить роботу повернувши помилку.

Перелік вхідних параметрів у метод getProducts:

- page ("p") – цілочисельний тип, який визначає номер сторінки яку потрібно повернути (сторінка – це кількість товарів розділена на кількість яка відображається за один раз). Э обов'язковим параметром і не може бути не вказаним;
- size ("s") – кількість елементів на сторінку, є обов'язковим параметром по замовчуванню;

- subcategoryId (“sc”) – ідентифікатор підкатегорії товари з якої потрібно відобразити;
- minPrice – мінімальна ціна товару у списку;
- maxPrice – максимальна ціна товару у списку;
- sortType – тип сортування товарів у списку;

Після зчитування всіх параметрів викликається метод сервісу який обслуговує товари, і має назву ProductService, в даному випадку використовується метод під назвою getProducts() (називається так само як метод контролеру). Основним призначенням методу є повернути вказану сторінку з вказаною кількістю елементів з параметрами сортування. Метод не поверне нічого у випадку відсутності категорії, сторінки або ж кількості елементів.

Вигляд логічних умов для не обов’язкових параметрів:

```
if (minPrice != null) {
        endSpecification = endSpecification.and(productsByMinPrice(minPrice));
}
if (maxPrice != null) {
        endSpecification = endSpecification.and(productsByMaxPrice(maxPrice));
}
```

Наведені вище дві умови вибору спрацьовують лише тоді, коли URL параметр присутній, тоді в спеціальний об’єкт специфікації, який визначає товар з якими параметрами підходить під вибірку, додається спеціальний в який прописується значення параметру. В результаті вибірки – будуть вибрані тільки товари які підходять під вказану специфікацію.

Вибір типу сортування виконується за допомогою оператора switch в який передається значення з яким потрібно порівняти. В проекті існує декілька типів сортування товарів, які можна переключати в будь-який момент у каталозі товарів на клієнтській частині сайту. Типом параметра є звичайний рядок, таким чином можна реалізувати будь який тип сортування і дати йому зрозумілу назву.

Вибір типу сортування в залежності від вхідного параметра:

```
Sort sort;
if (sortType != null) {
      switch (sortType.toUpperCase()) {
```

```
            case "PRICE_ASC":
                  sort = Sort.by("priceForMeasurementUnit").ascending();
                  break;
            case "PRICE_DESC":
                  sort = Sort.by("priceForMeasurementUnit").descending();
                  break;
            case "POPULAR":
                  sort = Sort.by("ordersCount").descending();
                  break;
         default:
            sort = Sort.by("ordersCount").descending();
      }
} else {sort = Sort.by("ordersCount").descending();}
```

Типи сортування:

- PRICE_ASC – сортування по збільшенню ціни (від дешевих до дорогих);
- PRICE_DESC – сортування по зпаданню ціни (від дорогих до дешевих);
- POPULAR – популярні товари, сортуються по принципу – чим більше замовлень тим популярніший товар.

Після того як всі параметри було визначено, створюється спеціальний об’єкт класу Specification<Product>, в який додаються всі параметри вибірки, після чого за допомогою репозиторія проводиться вибірка даних та повертається у форматі JSON.

За допомогою такого підходу можна реалізувати системи сортування будь-якої складності так як критеріїв може бути безліч, і придумати різну послідовність вибірки в залежності від параметрів цілком реально. Це значно зпрощує процес розробки користувацької частини так як дані приходять вже у потрібній послідовності.

3.5 Реалізація алгоритмів створення та керуванням замовленнями

На будь якому проекті, де хоч якось присутні замовлення, алгоритми керування замовленнями підрозумівають чіткість та надійність цих алгоритмів, так як будь-яке замовлення пов’язано з фінансами, обліком та обов’язками перед клієнтами.

У дипломному проекті я реалізував систему створення замовлень, яка відрізняється від класичних систем, в яких замовником є користувач сайту, а виконавцем є вже безпосередньо організація яка займається продажу. В проекті система замовлень працює по принципу користувач-користувач, це означає що замовником може бути будь-який користувач, при цьому виконавцем замовлення може бути інший користувач сервісу. Причиною цієї реалізації є те, що продавцем на сайті може бути будь-хто, і створивши замовлення на певний товар, або групу товарів, покупець оформляє замовлення, яке повинен виконати користувач (продавець), який додав товари на сервіс.

Метод який виконує створення замовлення знаходиться у класі OrderService, та приймає в себе майже повністю сформоване замовлення. Метод має назву createOrder, і після виклику та передачі в нього параметрів першим, що робить метод це отримує всю інформацію про користувача:

```
User user = (User) userService.loadUserByUsername(getPrincipal().getName());
```

За допомогою сервісу userService, метод отримує всі дані користувача з бази даних, за допомогою імені користувача. Після отримання даних користувача, починається спроба знайти у базі даних адресу, ідентифікатор якої був вказаний при створенні замовлення. Отримання адреси відбувається за допомогою ідентифікатора користувача, це зроблено для запобігання встановлення адреси, яку додав інший користувач. Користувач може додати в замовлення лише адресу, яку додав перед цим він сам.

```
Optional<ShippingAddress> opAddress =
shippingAddressRepo.findByUserIdAndId(user.getId(), order.getUserAddressId());
```

Тип змінної Optional означає що змінна може бути не присвоєною, і в даному випадку адреси може не існувати. Після вибірки даних з БД, проводиться перевірка чи присутня вказана адреса у списку адрес користувача, і чи в отриманому об’єкті замовлення присутні товари, які користувач хоче замовити:

```
if (!order.getOrderedItems().isEmpty() && opAddress.isPresent())
```

Така перевірка робиться для того, щоб запобігти створенню пустого замовлення, та замовлень з неіснуючими адресами.

Перед продовження перевірок створюються три змінні:

```
double orderTotalPrice = 0;
Long firstSellerId = null;
int index = 0;
```

Змінна orderTotalPrice, яка має тип double (дробове значення), призначення для акумуляці загальної вартості замовлення, вона ініціюється з початковим значенням 0. Наступною змінною є firstSellerId, яка відповідає за зберігання ідентифікатора першого продавця у списку товарів замовлення. Потрібно це для тому, що одною з умов створення замовлення, є те, що всі товари які присутні замовленні повинні буди подані одним й тим ж продавцем. Змінна ініціалізується із значенням null, так як значення їй буде присвоєно підчас першого проходу по товарах замовлення. Також у змінну index записується індекс проходу по товарах.

Перелічення товарів у замовленні відбувається за допомогою циклу for:

```
for (OrderedItem item : order.getOrderedItems())
```

Підчас кожної ітерації по товарам які додані до замовлення, відбувається перевірка кожного товару, так опиратись на отримані дані небезпечно, так як прислати під видом структури товару можна будь що. Тому кожен товар перевіряється по id:

```
Optional<Product> opCurrentProduct =
productRepo.findById(item.getProduct().getId());
```

Якщо товар присутній, то продовжується перевірка та додавання товар у замовлення. Але якщо товару не існує, і дані отримані сервером – хибні, то користувачеві виведеться відповідне повідомлення. А в результаті перевірки станеться помилка через розбіжності у кількості товарів у замовленні і кінцевій

кількості підтверджених товарів. Після підтвердження ідентифікатору товару виконується присвоєння товару у звичайну зміну класу Product (перед цим товар був опціональним):

```
Product currentProduct = opCurrentProduct.get();
```

Після присвоєння даних у змінну виконується функція логічного вибору if, яка перевіряє чи дана ітерація є першою:

```
if (index == 0) {
      firstSellerId = currentProduct.getSeller().getId();
      order.setSeller(currentProduct.getSeller());
      index++;
 }
```

Якщо ітерація перша, то змінній firstSellerId присвоюється id першого продавця першого товару у списку. В подальших ітераціях дані змінної будуть порівнюватись із продавцями інших товарів у списку.

Наступна перевірка порівнює ідентифікатор першого продавця з ідентифікаторами продавців у списку:

```
if (Objects.equals(firstSellerId, currentProduct.getSeller().getId())) {
      if (item.getQuantity() > 0) {
            item.setId(null);
            double itemTotalPrice = currentProduct.getPriceForMeasurementUnit() *
      item.getQuantity();
         item.setTotalPrice(itemTotalPrice);
         orderTotalPrice += itemTotalPrice;
            incrementProductOrderCount(currentProduct);
}
```

При співпадінні ідентифікаторів продавців проходить чи кількість товару більше ніж нуль. Я вирішив розділити дві умови вибору, щоб розділити помилки які можуть виникнути, зроблено це для зручності написання та тестування користувацької частини.

Так як перебір елементів списку йде не по звичайним товарам а по спеціальним об’єктам класу OrderedItem, які окрім об’єкту Product містять ще кількість замовлених елементів, для додавання цих даних в БД, потрібно анулювати ідентифікатор об’єкту, якщо він якимось чином присутній в отриманих даних:

```
item.setId(null);
```

Наступним кроком є підрахунок ціни за кількість товару, так як дані товару вже завантажені з БД, сенс перевіряти правильність та коректність даних відсутній. Підрахунок та запис у змінну ціни за кількість товару:

```
double itemTotalPrice = currentProduct.getPriceForMeasurementUnit() *
item.getQuantity();
```

Я використав для цього конкретну змінну, тому що дані будуть потрібні два рази, першого разу для запису кінцевої ціни товару за кількість у об’єкт item:

```
item.setTotalPrice(itemTotalPrice);
```

Другого разу дані знадобляться при додаванні суми за кількість товару до загальної суми замовлення:

```
orderTotalPrice += itemTotalPrice;
```

Після всіх корегування та перевірок останнім кроком у ітерації циклу є додавання кількості замовлення у товару, за допомогою методу incrementProductOrderCount, який викликається у кожній ітерації:

```
incrementProductOrderCount(currentProduct);
```

Код методу виглядає наступним чином:

```
public void incrementProductOrderCount(Product product) {
      product.setOrdersCount(product.getOrdersCount() + 1);
      productRepo.save(product);
}
```

Метод призначений тільки для використання у методі createOrder, так як передбачає отримання валідного (вибраного з бази даних) об’єкту класу Product інкрементації кількості замовлень товару, та подальше збереження екземпляру у базу даних.

Після повної перевірки списку товарів, задаються базо параметри замовлення, такі як зкидання id щоб запобігти перезапису, встановлення поля canceled у значення false, встановлення дати створення замовлення та кінцеву ціну замовлень, яка обчислена при переборі списку товарів:

```
order.setId(null);
order.setCanceled(false;
order.setCreationDate(new Date());
order.setTotalPrice(orderTotalPrice);
```

Для закріплення замовлення за користувачем, за ним закріплюється користувач, який був знайдений на самому початку методу, а також адреса яка була вибрана з бази даних та перевірені:

```
order.setUser(user);
order.setAddress(new OrderAddress(opAddress.get(), order));
```

Так як у кожного замовлення є набір статусів, замовленню додається початковий статус:

```
Optional<Status> opStatus = statusRepo.findByCode("CREATED");
OrderStatus orderStatus = new OrderStatus();
orderStatus.setOrder(order);
orderStatus.setStatusSetDate(new Date());
orderStatus.setStatus(opStatus.orElse(null));
```

З бази даних вибирається статус по спеціально вибраному коду. Я придумав розприділяти та міняти статуси по кодах, тому що так простіше орієнтуватись який статус у замовлення на даний момент та який може бути присвоєним.

Статуси є статичними і записуються в БД за допомогою міграці, на відміну від статусів замовлень (OrderStatus) як крім статусу містять в собі дату додавання статусу, сам статус замовлення та коментар. Всі ці параметри визначаються у взірці коду вище. Після створення статусу, та присвоєння об’єкту всіх потрібних даних, статус додається до замовлення і замовлення зберігається у базі даних:

```
Set<OrderStatus> statuses = new HashSet<>();
statuses.add(orderStatus);
order.setStatuses(statuses);
orderRepo.save(order);
removeOrderedItemsFromCart(order.getOrderedItems());
```

Останнім кроком у створенні замовлення (після безпосереднього збереження замовлення у базу даних) є видалення замовлених товарів з кошика користувача за допомогою методу removeOrderedItemsFromCart:

```
@Transactional
public void removeOrderedItemsFromCart(List<OrderedItem> orderedItems) {
      for (OrderedItem orderedItem : orderedItems) {
      cartItemRepo.deleteByProductId(orderedItem.getProduct().getId());}}
```

Призначення методу – перебору пройтись по списку отриманому як параметр, та видалити з кошика по ідентифікатору.

Не менш важливим є те, що всі методи зв’язані з замовленнями, проходили під анотацією @Transactional, яка підчас виконання методу створює транзакцію, яка позначає даний метод як транзакцію, і при будь яких помилках пов’язаних з виконанням коду чи запитів у базу даних буде здійснено так званий rollback – відміна транзакції. Я вважаю це дуже важливим так як настільки великий метод, який робить багато змін у БД повинен бути максимально захищеним від збоїв через некоректність даних, а при виникненні якихось помилок дані не повинні попасти до бази даних. За відсутності транзакції можлива фрагментація і незв’язність даних, що погано відобразиться на цілісності даних.

Керування замовленнями також передбачає відслідковування статусу замовлень. Кожне замовлення має перелік спеціальних статусів, які присвоюються на різних етапах існування замовлення, від етапу оформлення та збереження замовлення у БД, до етапу отримання товарів клієнтом.

За додавання нових статусів до створеного замовлення відповідає метод addOrderStatus, який також позначений як транзакція, за допомогою анотації @Transactional. Метод приймає як параметр екземпляр класу OrderStatus, який містить в собі статус, дату зміни, коментар та номер замовлення для якого присвоюється статус.

Тіло методу виглядає наступний чином:

```
Optional<Status> opStatus =
statusRepo.findByCode(orderStatus.getStatus().getCode());
```

Перед будь якими діями над статусом замовлення потрібно перевірити чи статус який потрібно встановити існує, що і виконує код вище. Якщо статус існує:

```
if (opStatus.isPresent()) {
```

Якщо статус існує, використовую той самий спосіб щоб перевірити чи існує замовлення для якого потрібно додати статус:

```
Optional<Order> opOrder = orderRepo.findById(orderStatus.getOrder().getId());
      if (opOrder.isPresent()) {
```

Якщо замовлення існує, створюється новий екземпляр класу Order, в який присвоюється посилання на об’єкт отриманий з бази даних. Наступним кроком додавання статусу є отримання поточного користувача з БД та перевірка чи користувач може додати статус:

```
User user = (User) userService.loadUserByUsername(getPrincipal().getName());
boolean isUserCanAddStatus = user.getId().equals(currentOrder.getUserId()) ||
user.getId().equals(currentOrder.getSeller().getUser().getId());
```

Перевірка потрібна для того, щоб змінити статус замовлення міг тільки продавець та отримувач замовлення, тому перевіряється ідентифікатор як продавця так і замовника. Після запису даних у змінну, йде запис іншої змінної, в якій міститься поточний статус замовлення:

```
OrderStatus currentStatus =
currentOrder.getStatuses().stream().findFirst().orElseThrow(() -> new
NotFoundException("Cannot find first status"));
```

Дані про поточний статус потрібні для того, щоб перевірити та порівняти та визначити який статус до замовлення можна додати.

Якщо користувач може додати новий статус до замовленння, потрібно точно визначити хто саме користувач, продавець чи покупець:

```
boolean isUserSeller =
user.getId().equals(currentOrder.getSeller().getUser().getId());
```

Визначення типу користувача який додає статус я зробив за максимально простою схемою: якщо продавець не покупець, то він продавець і навпаки.

Після цього, створюється новий єкземпляр Status та новий екземпляр класу OrderStatus:

```
Status newStatus = opStatus.get();
OrderStatus newOrderStatus = new OrderStatus();
```

Визначення доступних статусів для додавання до замовлення визначається за допомогою конструкції switch. Якщо користувач продавець по відношенню до замовлення:

```
switch (currentStatus.getStatus().getCode()) {
      case ("CREATED"):
            if          (newStatus.getCode().equals("CONFIRMED")          ||
      newStatus.getCode().equals("REJECTED")) {
                  newOrderStatus.setStatus(newStatus);
```

```
                    break;
              }
             case ("CONFIRMED"):
                    if (newStatus.getCode().equals("SHIPPED_OUT")) {
                          newOrderStatus.setStatus(newStatus);
                  break;        }
                    }
      }
```

Як видно на конструкції нижче, якщо користувач продавець, він може міняти обмежену кількість статусів. Продавець може перевести замовлення з кодом статусу «CREATED» в наступні статуси: «CONFIRMED» - який буде означати що продавець приймає замовлення і підтверджує його, «REJECTED» - статус означає що продавець відхилив замовлення.

Наступним варіантом є статус замовлення «CONFIRMED» - з якого продавець може змінити статус лише на «SHIPPED_OUT» - означає що продавець підготував та відправив замовлення. При цьому продавець вже не може скасувати замовлення, так як підтвердивши замовлення він зобов’язується його виконати.

Для покупця доступний інший набір статусів які можна додати до замовлення, він також вибирається за допомогою оператора switch:

```
switch (currentStatus.getStatus().getCode()) {
      case ("SHIPPED_OUT"):
            if (newStatus.getCode().equals("COMPLETED")) {
                  newOrderStatus.setStatus(newStatus);
            break;
              }
      case ("CREATED"):
            if (newStatus.getCode().equals("CANCELED")) {
                  newOrderStatus.setStatus(newStatus);
            break;
              }
}
```

Покупець має набір статусів, який дозволяє йому в разі потреби відмовитись від замовлення, або ж підтвердити отримання.

Продавець може підтвердити отримання за допомогою статусу «COMPLETED» тільки у випадку якщо продавець відправив замовлення і вказав

статус «SHIPPED_OUT». Іншою можливістю є скасування замовлення але лише у випадку коли замовлення ще не підтверджено і має статус «CREATED».

3.6 Розробка та огляд інтерфейсу користувача

Користувацький інтерфейс – перше з чим зтикається користувач потрапивши на будь-який сайт, по ньому навіть іноді можна судити чи варто продовжувати користуватись можливостями того чи іншого сайту, чи все ж варто пошукати аналоги.

У своєму дипломному проекті я старався досягти максимальної зручності користування сервісом як на комп'ютерах, так і на мобільних пристроях, так як в наш час смартфони є у більшості людей – реалізація доступу до всіх можливостей було одним з основних моїх завдань при проектуванні та створенні інтерфейсу користувача.

Перше що бачить відвідувач, який переходить по посиланню на сайт, це головна сторінка, на якій має бути виведена основна інформація, яка може зацікавити користувача. Я постарався зробити все таким чином, щоб відвідувач (можливий майбутній користувач сайту), зайшовши на сервіс, побачив саме те, що йому може бути цікавим. Наприклад на головній сторінці перш за все розміщено досить масивний по своїх розмірах слайдер, зображений на рис. 3.2 (своєрідна «карусель» фотографій, які перегортаються з вказаною періодичністю), а також самі популярні товари у категоріях. Користувач сам може вибрати категорію, популярні товари в якій він хоче переглянути. Діаграма, яка відображає всі можливості різних типів користувачів зображена на рис. 3.2, на ній зображені дії, які доступні гостям, користувачам, та адміністраторам. Відокремлення однієї групи людей від іншої робиться для того, щоб можна було чітко розмежувати права та функції, які доступні тій чи іншій

групі роле. За допомогою цього і створюється система, яка базується на ролях (role-based system).

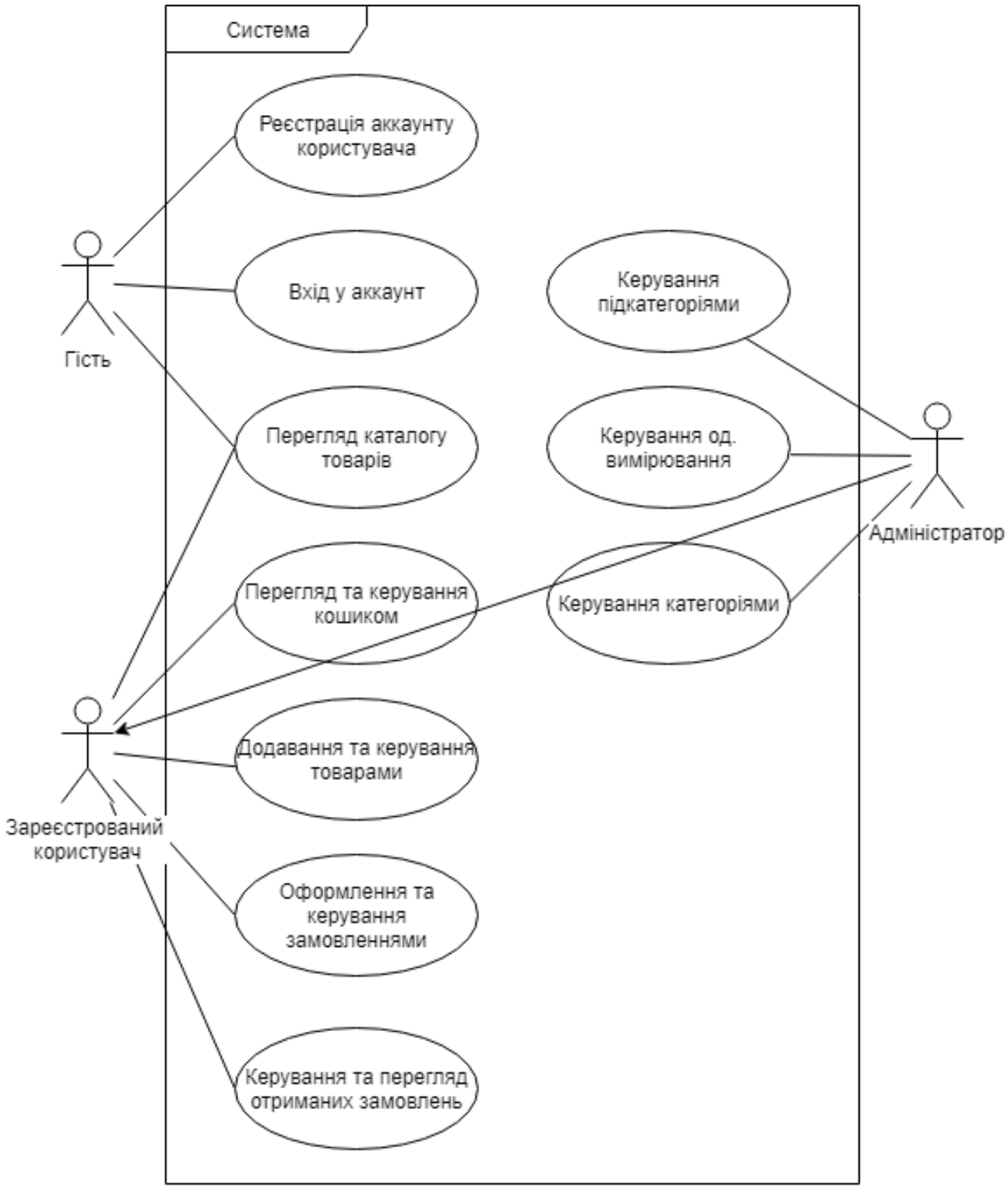

Рисунок 3.2 – Use case діаграма користувачів у системі

На діаграмі (рис. 3.2) користувач, який є адміністратором має додаткові привілеї а також більше прав, але також, адміністратор має доступ і до привілеїв, доступних користувачам, на відміну від гостя, який має обмежений набір функціоналу, так як обліковий запис користувача у нього відсутній. Обмеження у функціоналі є і у користувача, який не може повторно зареєструватись, так як він вже присутній в системі, але також він не може користуватись функціоналом для адміністратора, таким чином у системі реалізовані розмежування (ролі) користувачів.

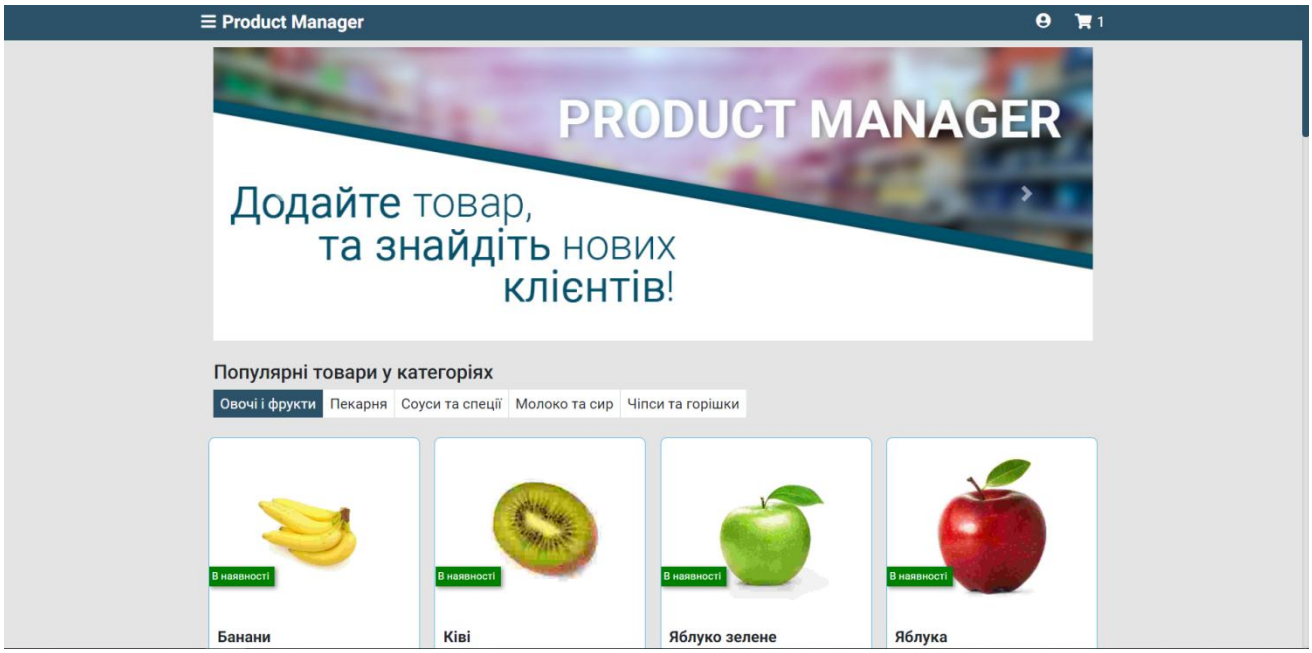

Рисунок 3.3 – Головна сторінка проекту

Також у верхньому правому куті, кожен відвідувач може зареєструватись, або увійти, якщо він раніше був зареєстрований.

Самим основним способом взаємодії клієнта з сервісом є реєстрації та створення облікового запису, так як всі основні функції прив’язані до профілю користувача. Реєстрація. У формі реєстрації (рис. 3.4) вводяться дані про обліковий запис користувача.

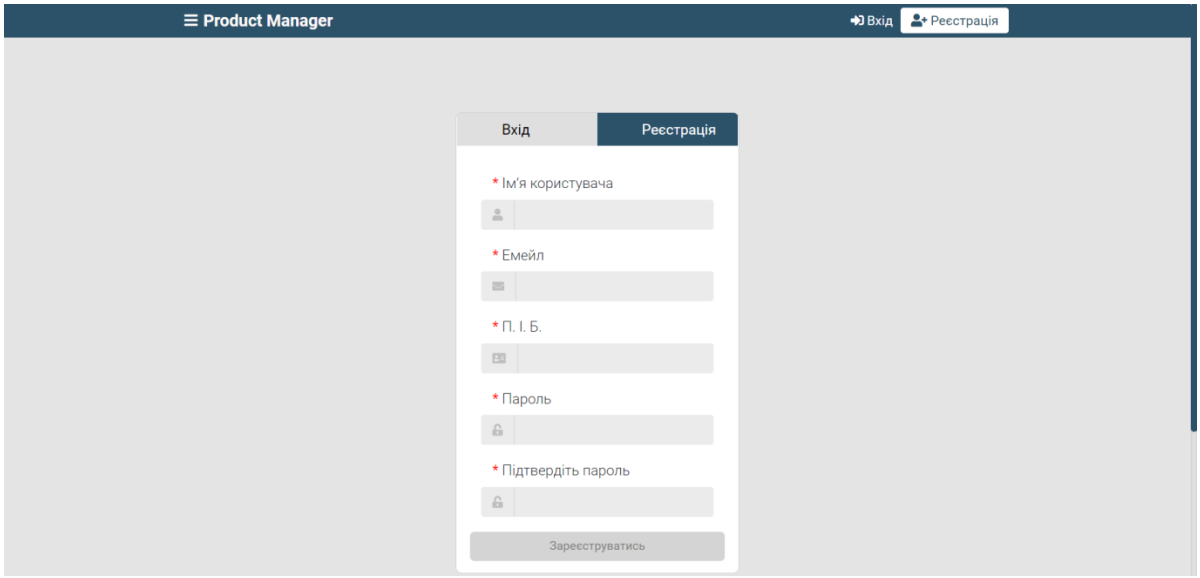

Рисунок 3.4 Форма реєстрації користувача

Форма містить наступні поля:

- ім‘я користувача (логін) – використовується для авторизації у системі;
- емейл – електронна адреса користувача, за допомогою якої буде відбуватись зв’язок із користувачем у разі потреби;
- П. І. Б. – використовується для звертань до клієнта, а також може бути використано у електронних листах в якості імені отримувача;
- пароль – використовується для входу у систему і є конфіденційним. Паролі всіх користувачів не тримаються у відкритому вигляді і не відображаються на сторінках та запитах сервісу;
- Підтвердження паролю – дає користувачеві перевірити правильність введення даних.

Форма має всі потрібні перевірки даних, це як перевірки на існування користувачів з таким іменем чи емейлом, так і самі прості перевірки на кількість символів. На рис. 3.5 зображено декілька помилок форми, які дозволяють користувачеві зрозуміти що введено не вірно. Таким чином можна легко орієнтуватись у полях форми.

Також про некоректність форми сигналізує колір кнопки «зареєструватись» яка при введенні неправильних даних відключається та стає доступною лише у випадку введення коректних даних.

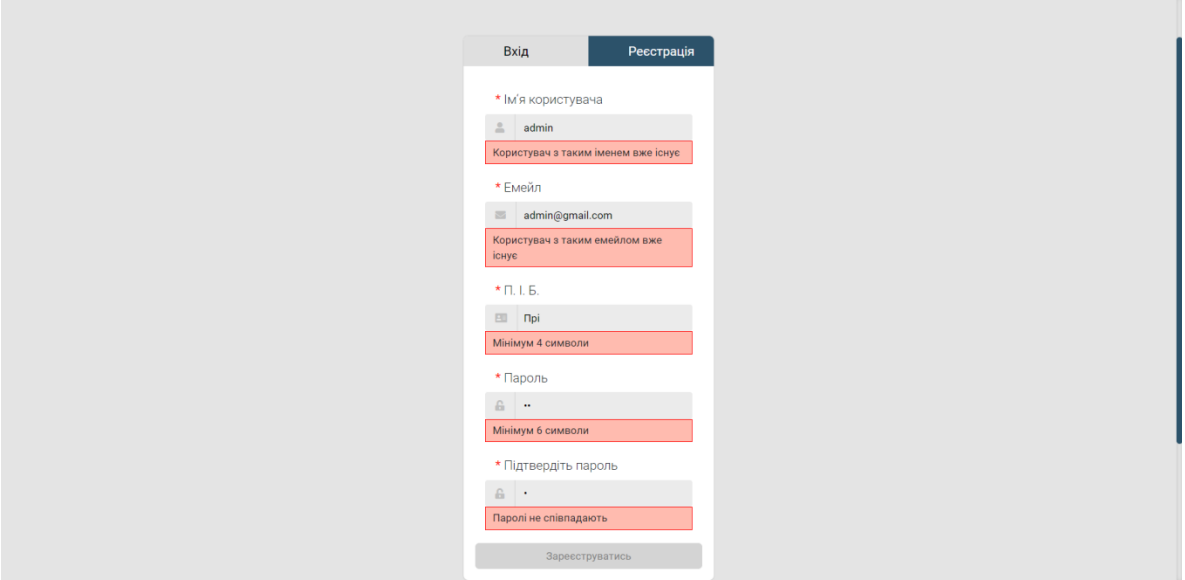

Рисунок 3.5 – Відображення помилок форми входу

Реєструючись, користувач отримує доступ до всіх функцій, дозволених для користувачів сервісу, йому одразу надаються всі права на публікацію своїх товарів, а також можливість здійснювати покупки.

Якщо у користувача вже існує обліковий запис, і він бажає увійти з іншого пристрою або ж здійснити повторний вхід, він може скористатись формою реєстрації (рис. 3.6) яка дозволяє увійти в систему а також при кожному вході генерується новий JWT токен. Все що потрібно для входу це: логін та пароль.

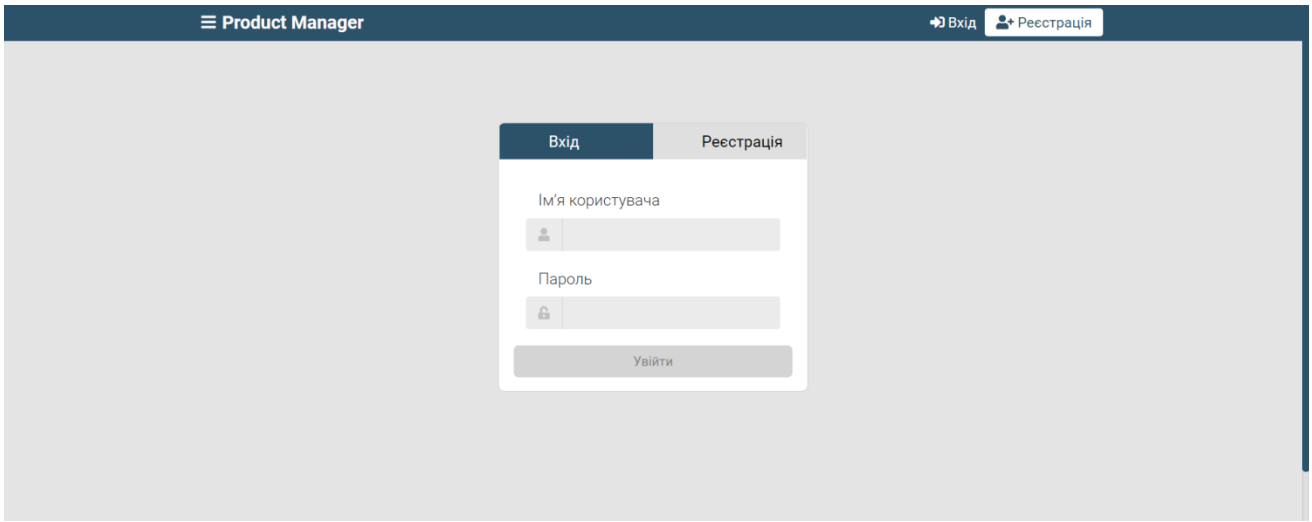

Рисунок 3.6 – Форма входу

Після реєстрації та входу, користувач отримує доступ до налаштувань облікового запису, в яких він може змінити пароль, емейл чи ім’я. Також можлива зміна паролю, але зміни вступлять в силу тільки після повторного входу, а на пристроях де був здійснений вхід – до моменту поки не завершиться термін дії токену. На сторінці інформації про обліковий запис (рис 3.7) користувач може переглянути поточні дані, такі як: ім’я користувача (логін) та адрес електронної пошти. Вся інформація доступна лише власнику, і ніхто інший не може дізнатись її.

Всі дані розділені на блоки, кожен з яких вміщує специфічні дані. Наприклад «безпека» - розділ в якому доступні всі дії пов’язані з безпекою облікового запису, а також «основна інформація» - тут зібрані всі основні дані користувача.

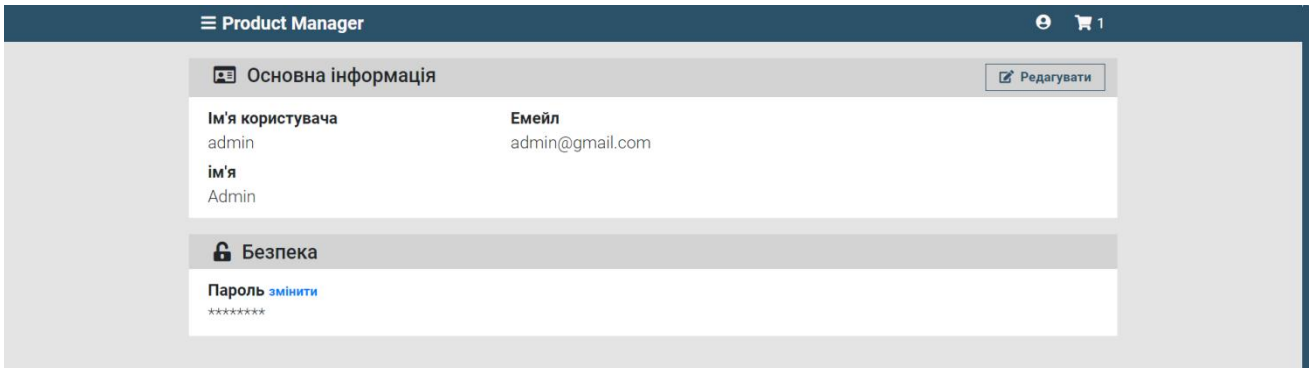

Рисунок 3.7 – Розділ «Мій аккаунт»

Дії над даними (редагування) також розділені по блокам, щоб користувачеві було зрозуміло до якого саме розділу чи поля відноситься та чи інша кнопка, або посилання.

Користувач може змінити не всі дані обл. запису. Наприклад логін користувача має постійне значення, та закріплений за користувачем. Вікно редагування даних зображено на рис. 3.8.

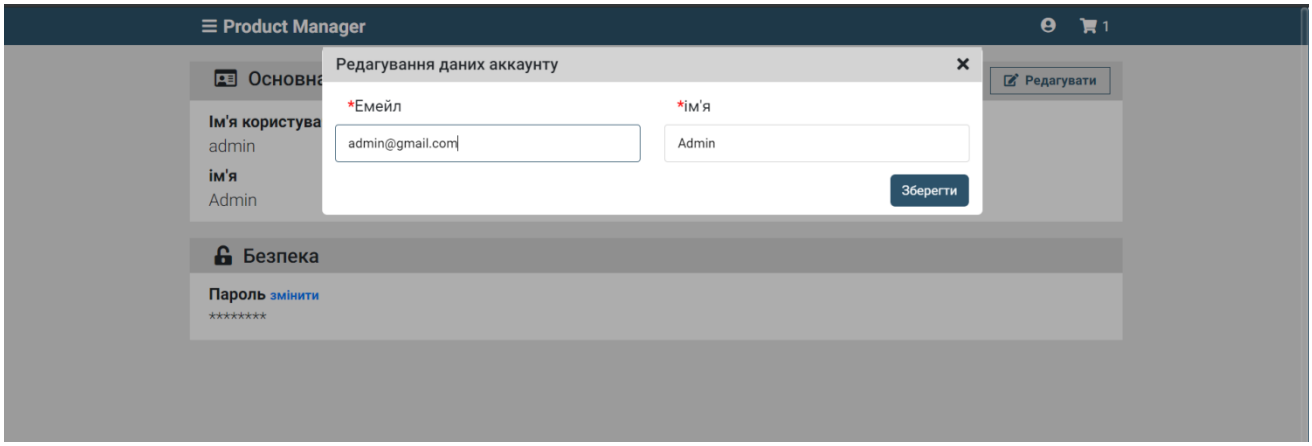

Рисунок 3.8 – Вікно редагування основних даних облікового запису

Користувач може змінити емейл та ім’я, при введенні даних потрібно клікнути на кнопку «Зберегти» і дані будуть завантажені на сервер. Також форма редагування валідується, і не дасть завантажити на сервер некоректні дані з неправильною кількістю символів. Також емейл проходить валідацію емейлу, це означає що значення, вписане в поле повинно містити саме емей, не просто набір символів.

Для зміни паролю, користувачеві потрібно знати свій попередній пароль, тільки тоді він зможе ввести новий пароль та успішно змінити його. Форма зміни паролю зображена на рис. 3.9. Для заповнення форми також потрібно підтвердити новий пароль.

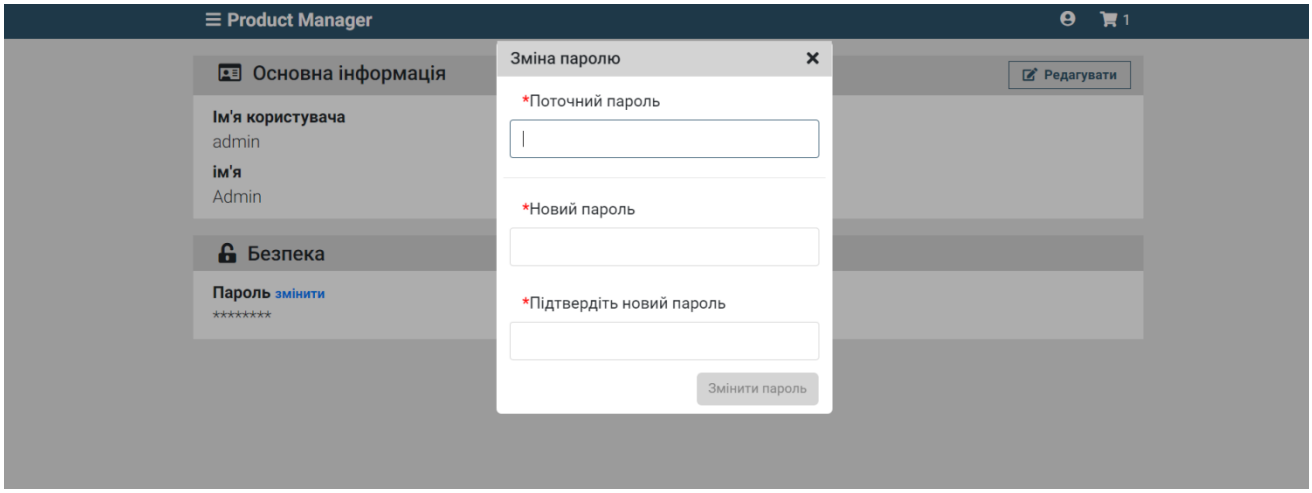

Рисунок 3.9 – Вікно форми зміни паролю

За допомогою повідомлень, користувач зможе побачити, що саме він ввів невірно, і на що потрібно звернути увагу та виправити. Про наявність помилок також сигналізує колір кнопки «Змінити пароль».

Я намагався зробити кожну форму, в яку потрібно вводити дані, максимально зрозумілою та простою для заповнення, за рахунок цього вдалось уникнути довгих розписів до кожного поля (що і для чого потрібно), у більшості випадків все зрозуміло і наглядно.

Одною з основних можливостей користувача у моєму сервісі є можливість торгівлі товарами. Для додавання та маніпулюванням товарами є спеціальні функції та можливості. Кожному зареєстрованому користувачеві доступна функція додавання нового товару. Відповідно кожен користувач, зареєструвавшись, може розмістити на сервісі свій товар. Після чого він стає доступним для покупки іншим користувачам. Форма публікації товару повинна бути максимально захищеною від некоректних даних (повинна бути зроблена так, щоб жоден користувач не зміг ввести в неї щось не так як має бути), також я постарався зробити форму максимально простою для розуміння, щоб користувачеві було одразу зрозуміло, які дані і куди потрібно ввести, форма зображена на рис. 3.10.

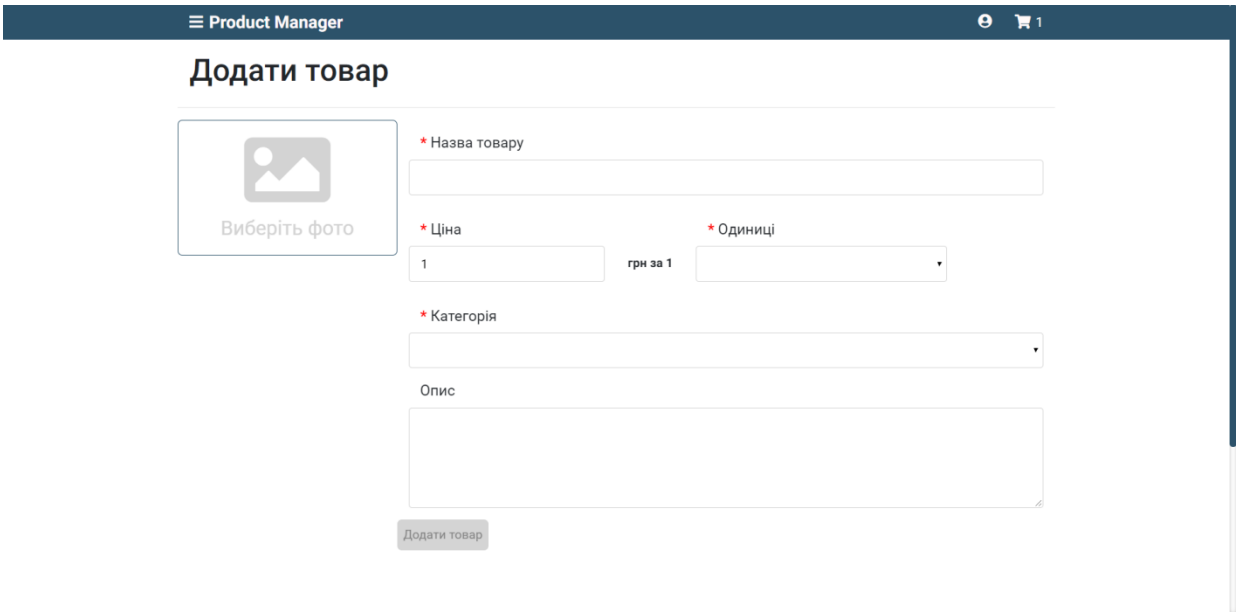

Рисунок 3.10 – Сторінка додавання нового товару на сервіс

Поля форми розташовані в певній послідовності, щоб користувач по мірі заповнення, вказував дані у їх логічній послідовності. Форма складається з наступних полів:

- фото товару – у кожного товару, який публікується повинно бути фото;
- назва товару – назва товару, яку будуть бачити користувачі у каталозі товарів;
- ціна – значення у гривнях. Ціну бачать всі користувачі у каталозі товарів, та на сторінці товару;
- одиниці – одиниці вимірювання товару (рис. 3.11), користувач може вибрати у якій одиниці вимірювання він буде продавати товар;
- категорія – категорія до якої відноситься товар;
- підкатегорія – у каталозі якої, буде відображатись товар;
- опис – продавець може вказати опис до товару. Поле є не обов'язковим але при наявності даних про товар – покупцеві буде простіше орієнтуватись.

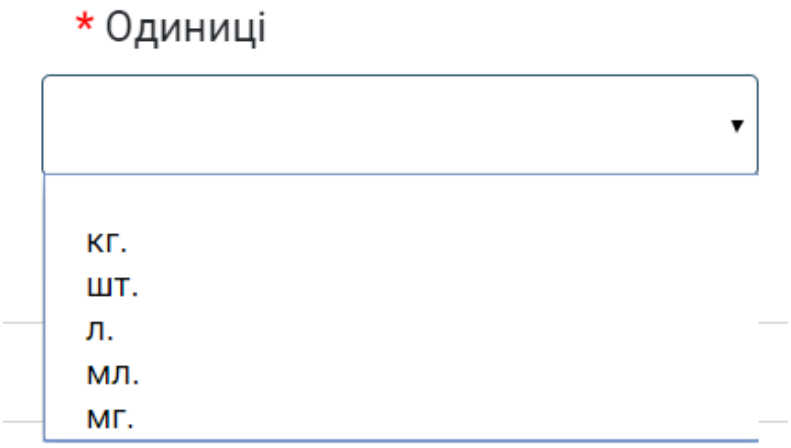

Рисунок 3.11 – Меню вибору одиниці вимірювання товару

Вибір одиниць вимірювання є важливою частиною публікації товару, так як це одна з переваг мого сервісу над іншими сервісами для розміщення товарів на продаж. Дані одиниці вимірювання, приведені на рис. 3.11, відображаються

на протязі всіх етапів взаємодії з товарами, від відображення у каталозі, до відображення в таблиці замовлених товарів.

Не менш важливим є і розміщення товару у потрібній категорії товарів, що є не менш важливим, так як правильне розміщення товарів – запорука зручного пошуку їх у тій категорії, яка цікавить користувача. Приклад вибору категорій та підкатегорій зображено на рис. 3.12.

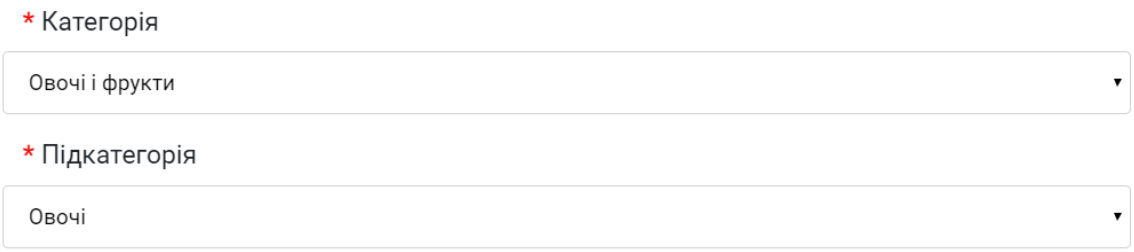

Рисунок 3.12 – Приклад вибору категорії та підкатегорії товару

Так як у кожної категорії свій набір підкатегорій – меню вибору підкатегорій міняється в залежності від вибраної категорії.

Як і у всіх форм на сайті, у даної форми присутня валідація, присутні певні вимоги до вмісту полів. Так само присутні обов'язкові поля, які позначені символом «*», також є зображення товару, яке даним символом не позначено, але є обов'язковим. Приклад роботи валідації зображений на рис. 3.13.

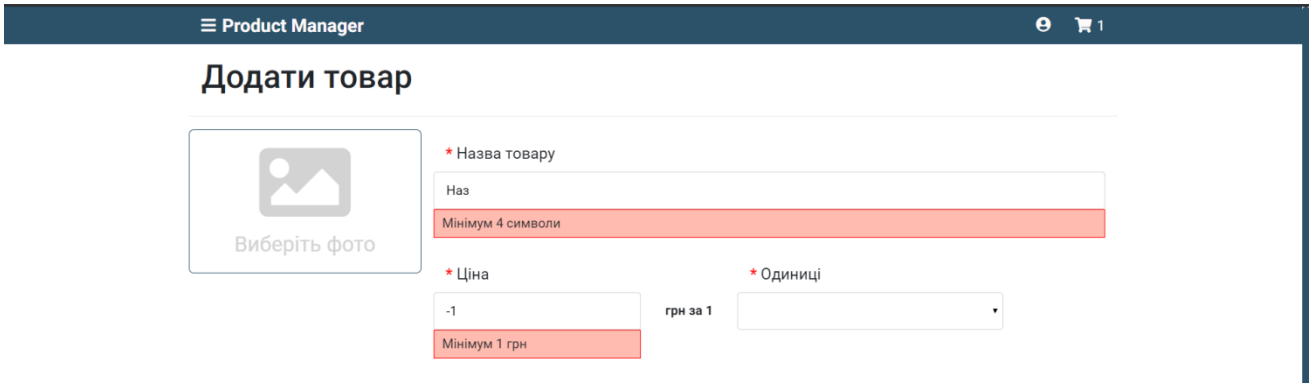

Рисунок 3.13 – Приклад валідації даних на формі публікації товару

У всіх компонентів у формі, присутні певні вимоги, від того, що поле повинно мати не більше ніж якусь кількість символів, до просто обов’язковості заповнення поля, хоча б одним символом.

Повністю заповнена форма додавання товару зображена на рис. 3.14. У формі окрім всіх даних, вибрано зображення товару. Яке автоматично завантажилось у спеціальне місце зліва. Таким чином, користувач зможе приблизно дізнатись як буде виглядати зображення його товару після завантаження.

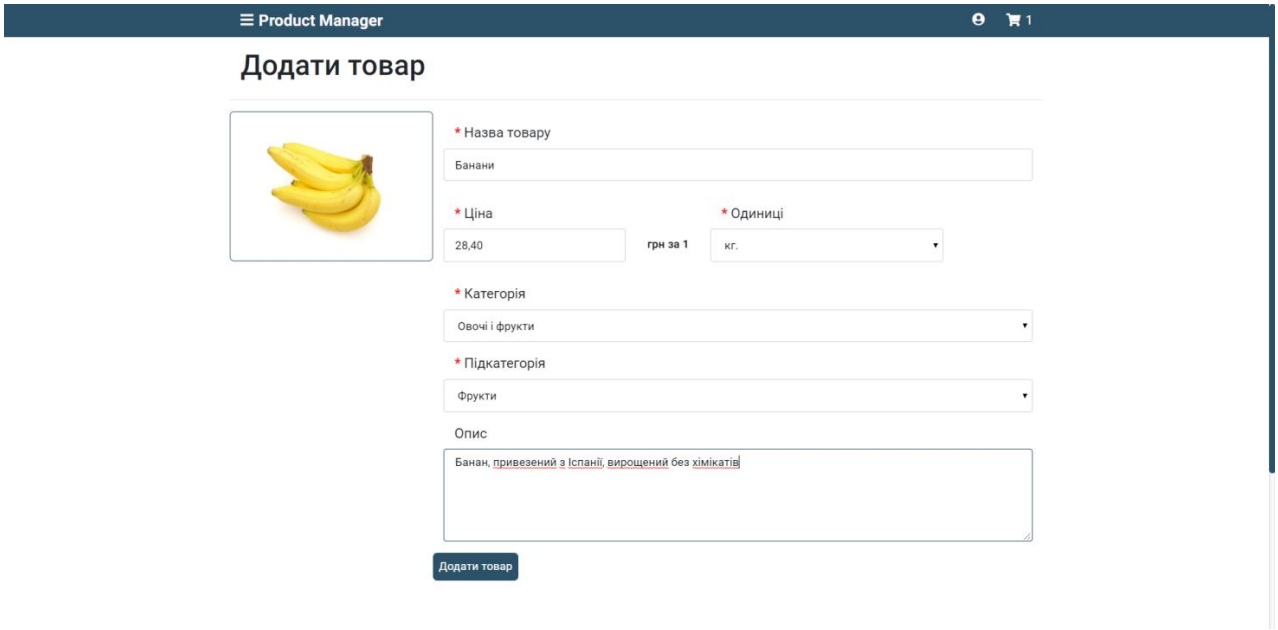

Рисунок 3.14 – Приклад заповненої форми публікації товару

При повному заповненні форми даними, про можливість додати товар, сигналізує колі кнопки, а також відсутність помилок і повідомлень про необхідність внести правки у введені дані.

Після додавання товару, користувач буде перенаправлений у розділ «Мої товари». Сторінка містить всі товари, які додав користувач (рис. 3.15), а також додаткову інформацію до них. Список організований таким чином, що користувач бачить товари у спаданні порядку їх додавання (самі нові товари зверху). Товари також розміщуються на різних сторінках, перехід по яким, можливий по номеру.

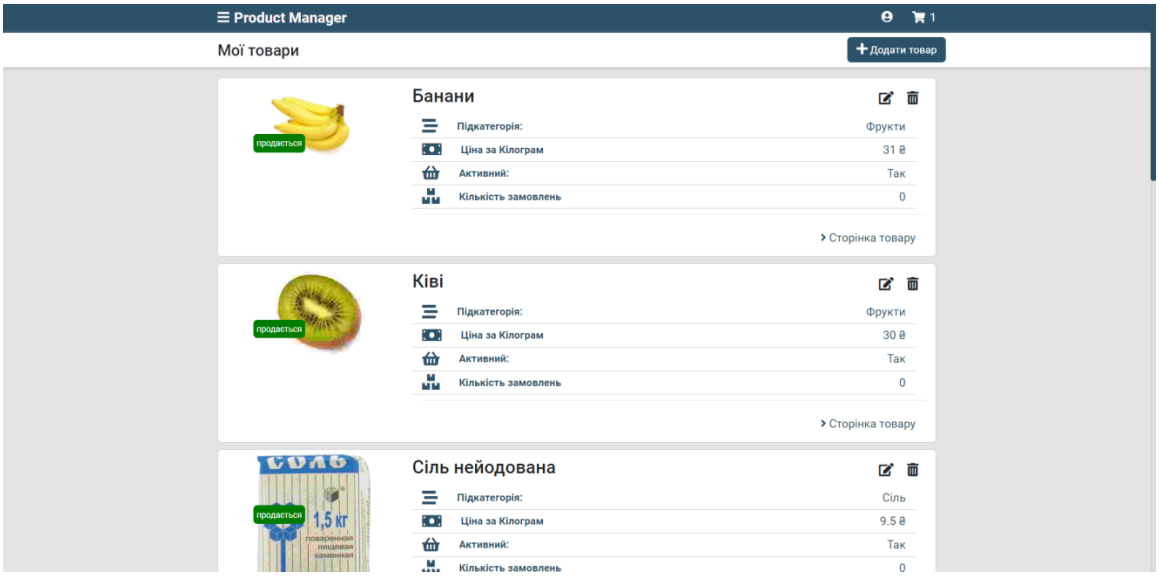

Рисунок 3.15 – Приклад заповненої форми публікації товару

Користувач бачить основні дані доданих ним товарів, такі як: ціну за одиницю вимірювання, чи товар активний, кількість замовлень. Також користувач може перейти на сторінку то вару по посиланню біля кожного товару. Кожен товар можна редагувати (рис 3.15),

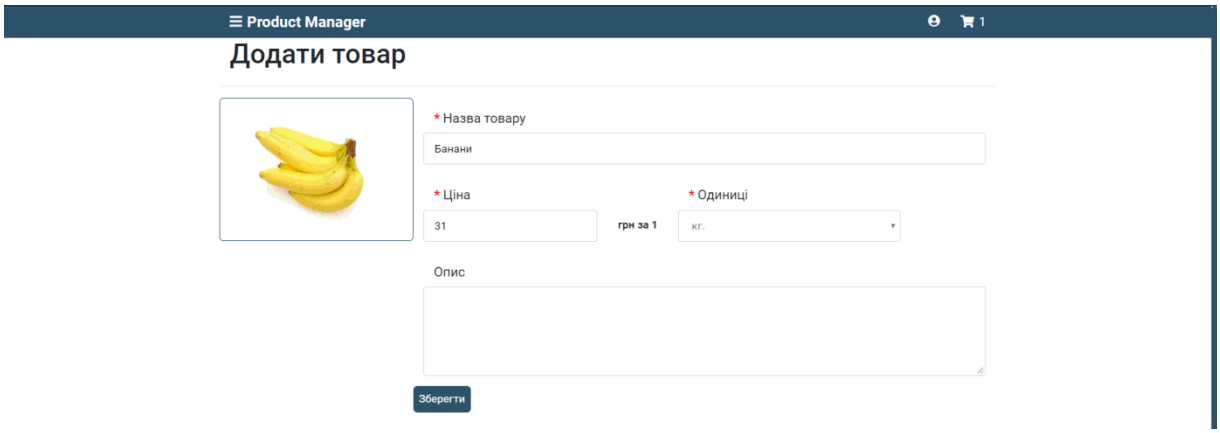

Рисунок 3.16 – Форма редагування товару

У формі редагування товару можна змінювати не всі дані, так як при створені товару, він закріплюєтьс за певною підкатегорією – ні категорію, ні підкатегорію змінити неможливо. Всі інші дані, включаючи зображення, можуть бути змінені в будь-який момент.

Для покупки товарів у інших користувачі можна скористатись меню категорій (рис. 3.17), або ж вибрати товар, з популярних, на головній сторінці.

Для того щоб відобразити меню категорії, достатньо просто навести курсор на відповідний знак у лівій частині екрану, поряд з назвою проекту.

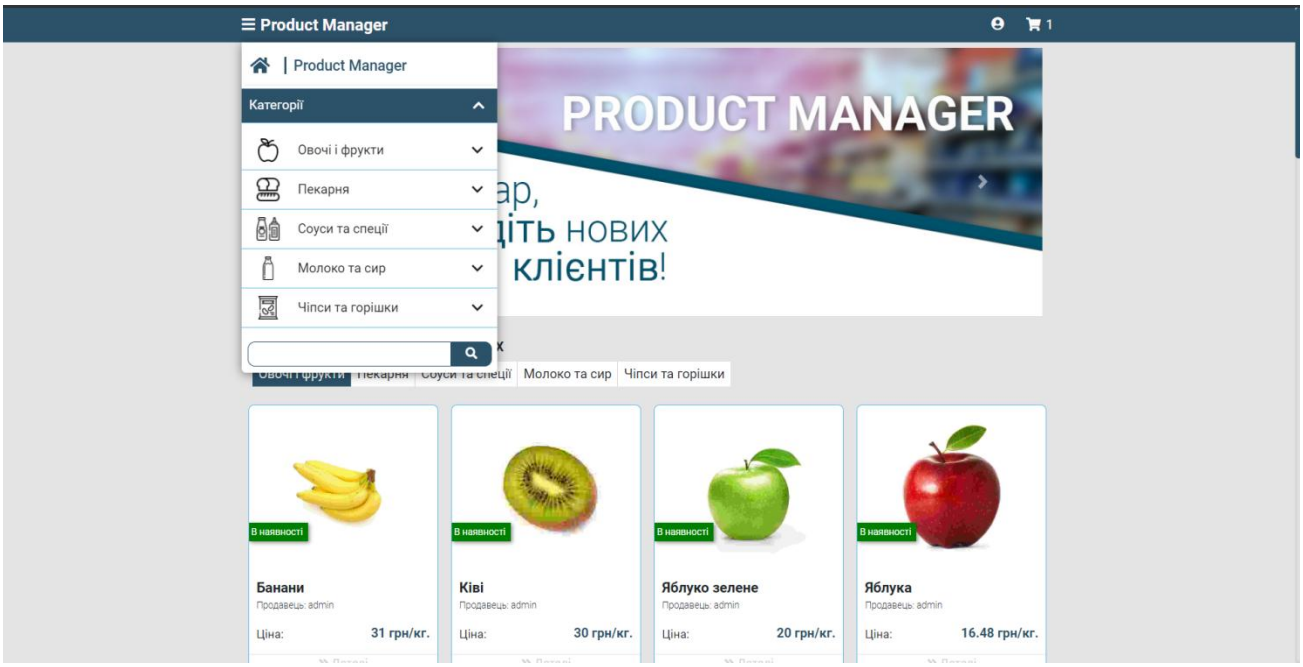

Рисунок 3.17 – Меню категорій

Користувачеві доступні всі категорії, і саме мені можна відкрити з будь-якої сторінки сайту. Дане меню являє собою точку швидкого переходу до категорій та підкатегорій.

На мобільних пристроях дане меню виглядає трохи по-іншому (рис. 3.18). Меню, як і весь сайт адаптовано для екранів смартфонів, планшетів, і в загальному для екранів будь-яких пристроїв.

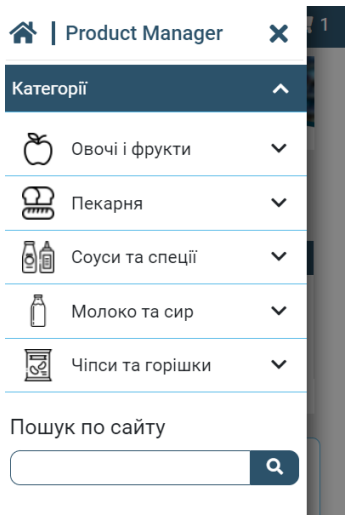

Рисунок 3.18 – Вигляд меню категорій на мобільних пристроях

На мобільних пристроях меню зроблено таким чином, щоб користувачеві не потрібно було щось масштабувати чи шукати, все завжди під рукою.

При кліку на категорію розгортається список підкатегорій. По кліку на підкатегорію, користувач перейде до каталогу підкатегорій (рис. 3.19), у якому по сторінках виводяться всі товари підкатегорії.

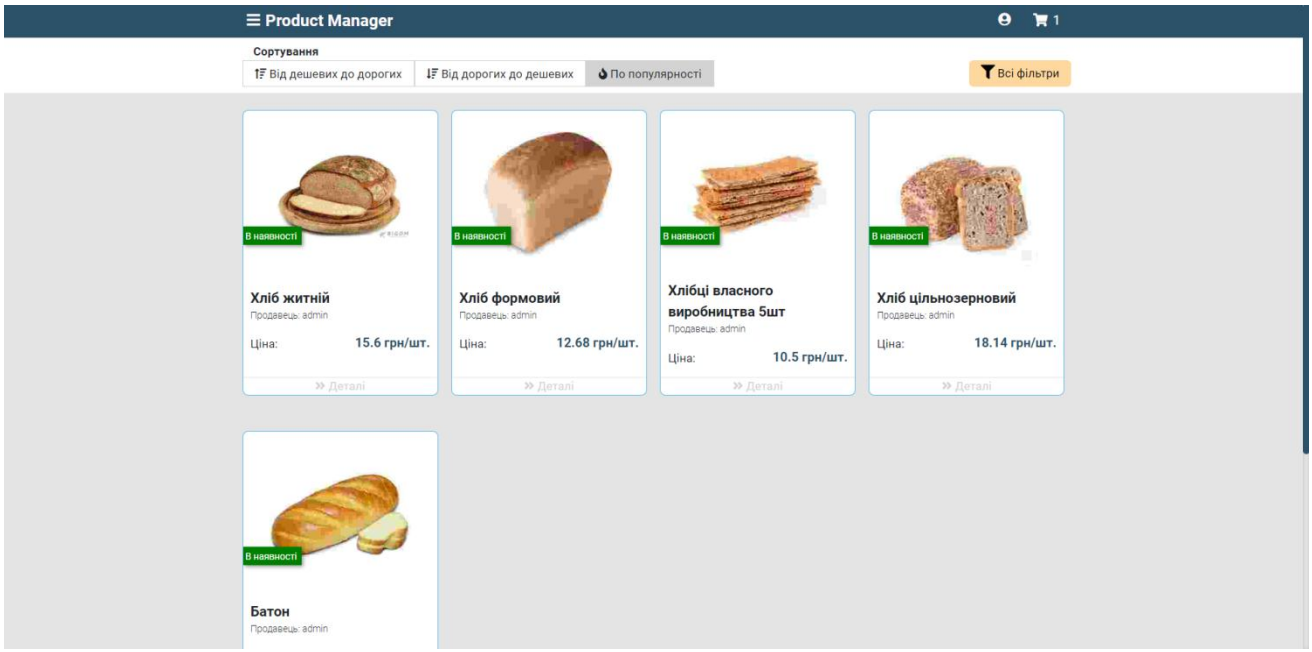

Рисунок 3.19 – Сторінка каталогу товарів по підкатегорії

Каталог передбачає сортування товарі по зростанню та спаданню ціни а також по кількості замовлень. Також можливе сортування від конкретної ціни до конкретної. Сама концепція сервісу передбачає додавання великої кількості однакових товарів від різних людей і щоб якось орієнтуватись в цих товарах їм пригодяться фільтри по яких вони зможуть вибрати що купити, а фільтр по популярності допоможе знайти такі собі «загально признані» товари, які люди часто купляють.

Знайшовши потрібний товар, користувач може перейти на сторінку товару (рис. 3.20), клікнувши по карточці товару. Про те що по кліку на карточку присутня дія сигналізує полоса з нижньої частини карточки, яка при наведенні стає синьою.

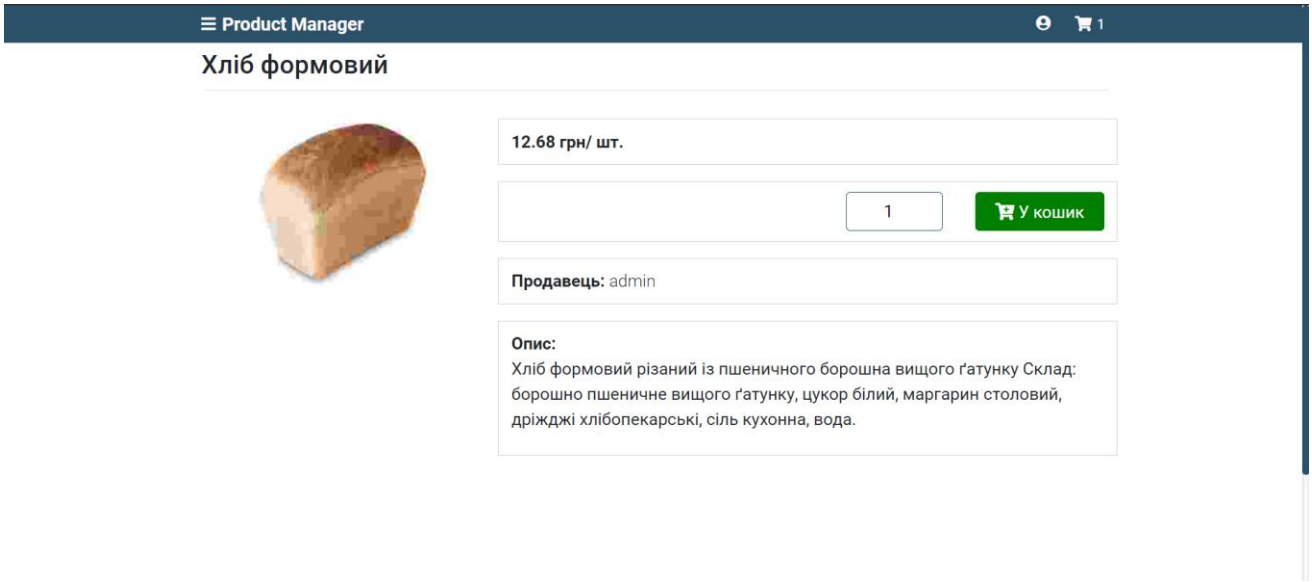

Рисунок 3.20 – Сторінка товару

Іншим способом перейти на сторінку товару, є пряме посилання на неї. Так як у кожного товару є унікальний ідентифікатор – його легко можна відрізнити від інших.

Щоб додати товар у кошик, достатньо вибрати кількість товару, та клікнути на кнопку «У кошик», після чого появиться спеціальне повідомлення (рис. 3.21) яке буде означати що товар додано у кошик користувача. Кошик користувача є єдиним для всіх пристроїв і ніяким чином не залежить від браузера, пристрою, чи місця звідки користувач заходить на сервіс. Всі дані кошика зберігаються на сервері у базі даних. Також одразу після додавання товару у кошик, користувач побачить що з права зверху, біля іконки кошика, з’явиться кількість товарів, які на даний момент присутні у кошику, це можна спостерігати на рис. 3.21.

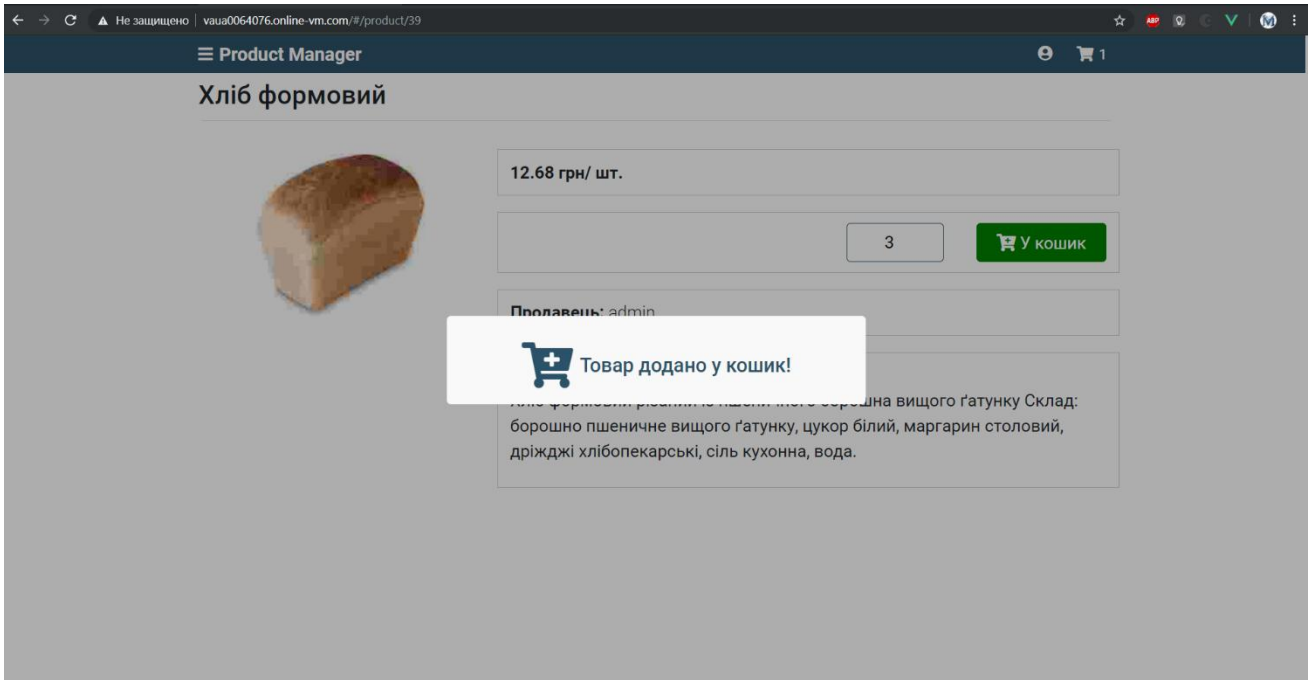

Рисунок 3.21 – Повідомлення про успішне додавання товару у кошик

Повідомлення означає, що запит на сервері успішно виконався і товар додано у кошик, тепер він доступний на всіх пристроях і користувач може продовжити покупки з іншого пристрою.

Переглянути товари у кошику, загальну ціну та назву товару у кошику можна просто якщо навести курсор іконку кошика, після цього появиться відповідне меню в якому буде показана вся інформація (рис. 3.22).

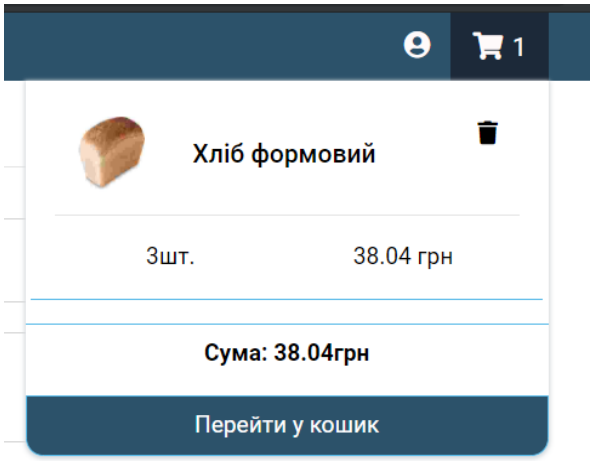

Рисунок 3.22 – Меню кошика

Також у меню можна швидко видалити товар із кошика. Функції меню призначені лише для швидкого орієнтування у вмісті кошику, для повної інформації потрібно натиснути на кнопку «Перейти у кошик», після чого користувач перейде на сторінку кошика (рис. 3.23), на які зможе отримати повну інформацію про товари, вибрати кількість та оформити замовлення.

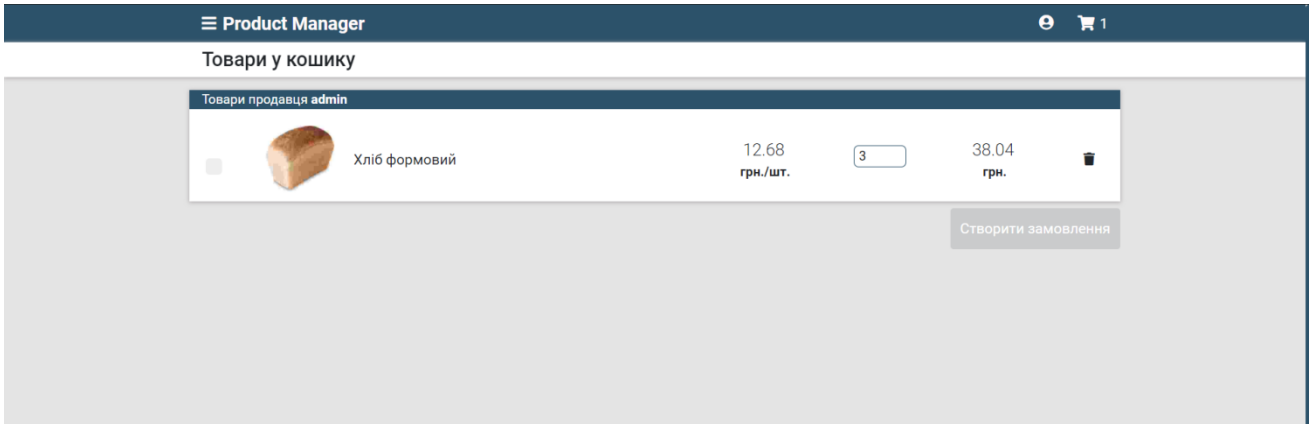

Рисунок 3.23 – Сторінка кошику

На сторінці кошику, користувач отримує доступ до всіх функцій, таких як вибір товару який потрібно додати до замовлення, чи зміна кількості товару. Також сторінка дублює можливості мінімізованої версії кошика такі як: видалення, кількість та ціна за кількість.

На сторінці товари групуються по продавцях. Замовлення можна оформити тільки з товарами від одного й того ж продавця. Для оформлення товарів від інших продавців, потрібно вибрати їхні товари та повторити процедуру. У кошик можна додати товари від будь-якої кількості користувачів, вони всі будуть погруповані, щоб користувачеві було простіше вибрати товари одного продавця навіть не дивлячись на його ім’я. Про некоректність вибору, як і на багатьох інших сторінках, сигналізує колір кнопки «Створити замовлення», вона відключається якщо товари у поточній конфігурації не підходять для створення замовлення, наприклад як вище на рис. 3.23 вони просто не вибрані. Стан товару (вибраний до замовлення чи ні) зберігається на сервері, це означає, що користувач може відмітити потрібні товари а замовлення створити з іншого пристрою або пізніше.

Користувач може замовити не всі товари продавця, для цього була реалізована система вибору за допомогою чекбоксів (галочок), які дозволяють легко вибрати тільки потрібне, а все інше не буде замовлено. Як і всі інші сторінки, сторінка адаптована під любі типи екранів, і підстроюється в залежності від ширини екрану пристрою.

Після вибору товару користувачеві потрібно клікнути на кнопку «Створити замовлення», після чого він перейде на сторінку оформлення замовлення, яка зображена на рис. 3.24.

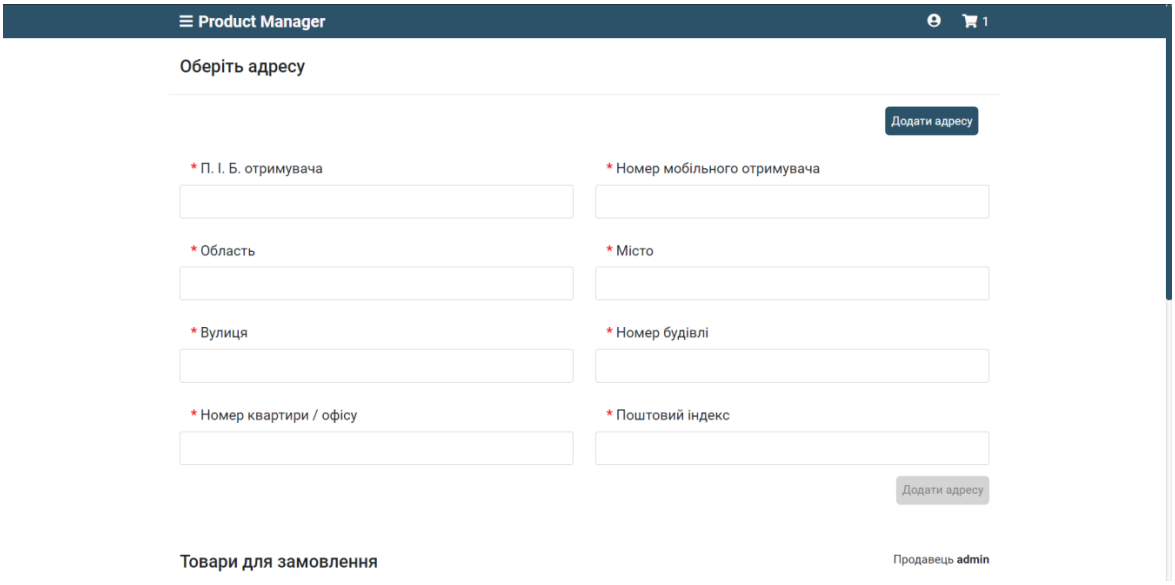

Рисунок 3.24 – Сторінка оформлення замовлення, сегмент вибору адреси

Якщо це перше замовлення користувача, він повинен додати свою адресу, або адресу куди потрібно доставити товар. Одноразово ввівши даних, користувач зможе використовувати адреси повторно і просто вибирати їх у замовленні (рис. 3.25), це дозволить економити час і нерви користувача при оформленні замовлень, і додавши декілька адрес можна ніколи більше не перейматись цим питанням. Як видно на рисунку 3.25, одного разу додавши адресу, вона переходить у формат карточки, яку можна як відредагувати, так і видалити. При видаленні або редагуванні адреси, в самому замовленні (яке було створено перед редагуванням) адреса не зміниться. Це зроблено спеціально, щоб уникнути втрати даних замовлення, а також достовірності даних. Також якщо у користувача в списку адрес немає потрібної – він завжди може додати ще одну

адресу просто клікнувши на кнопку «Додати адресу», після чого перед користувачем появиться та ж сама форма що і при першому додаванні адреси. У формі також присутні певні обмеження по кількості символів, та цифр номеру телефону. Всі дані повинні бути максимально достовірними, так як продавець буде відправляти товари орієнтуючись саме на ці дані.

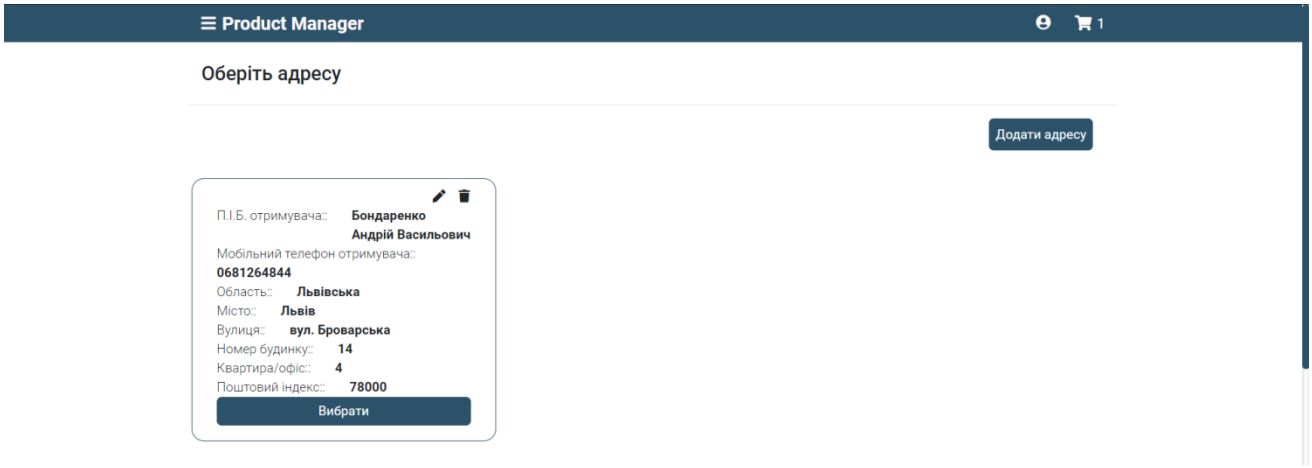

Рисунок 3.25 – Карточки доданих адрес

Наступним сегментом на сторінці замовлень є сегмент товарів які буле замовлено. Користувач може переглянути товари які він замовив, та звірити дані, щоб в разі розбіжності даних можна було скасувати замовлення. У таблиці товарів (рис. 3.26) перелічені всі додані товари, їхня назва, замовлена кількість та ціна за цю кількість.

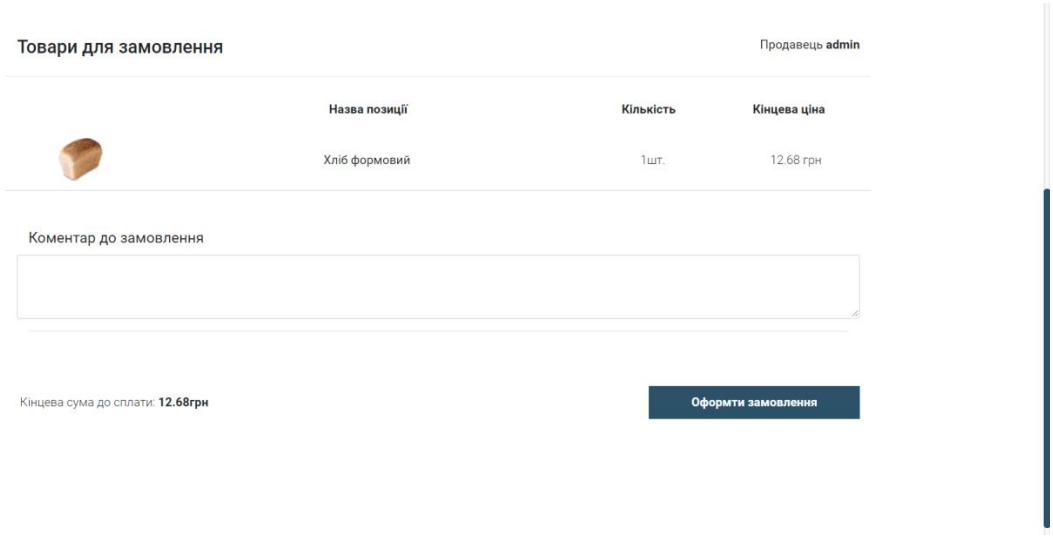

Рисунок 3.26 – Сегмент товарів для замовлення

На даному етапі, користувач вже не може видаляти товари, для цього йому потрібно повернутись до кошика і зробити ці дії там.

До кожного замовлення може бути додано коментар, який продавець побачить коли буде обробляти замовлення. У коментарі можуть бути якісь побажання щодо відправки товару, упаковки, чи способу/служби доставки товару до покупця.

Після того як всі дані будуть введені і перевірені покупцем на правильність, потрібно клікнути на кнопку «Оформити замовлення». Після натискання на кнопку, користувачеві буде показано повідомлення про успішність створення замовлення, та перенаправлено на сторінку «Мої замовлення» (рис. 3.27).

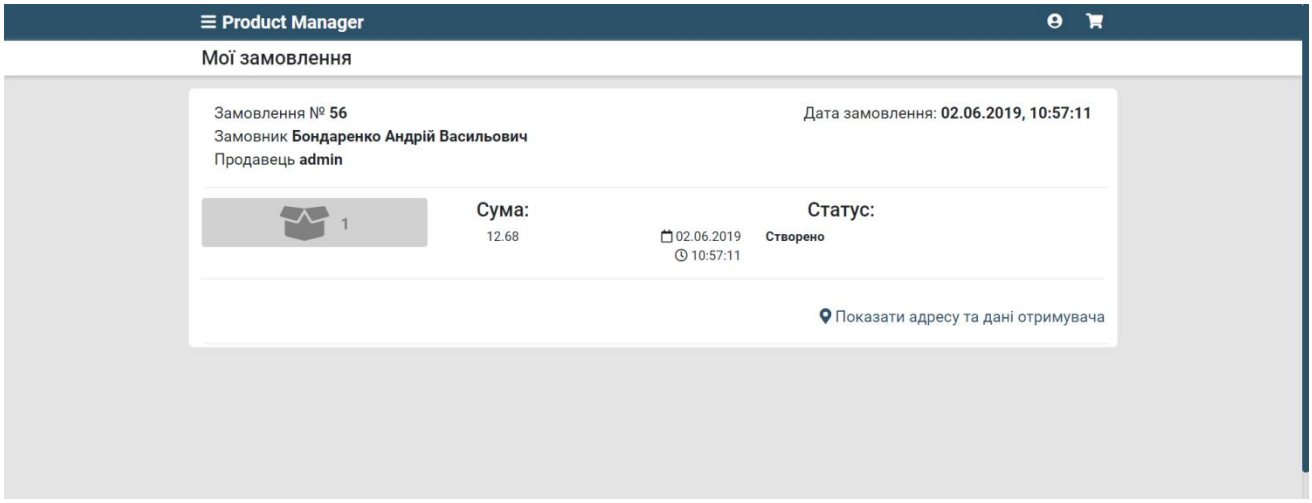

Рисунок 3.27 – Сторінка «Мої замовлення»

На сторінку виводиться список всіх замовлень користувача у скороченому вигляді, при необхідності інформацію можна розгорнути. С укороченому варіанті замовлення користувач бачить наступні дані: номер замовлення, дата замовлення, замовника та продавця, кількість замовлених товарів які позначені цифрою, суму замовлення, сегмент статусів.

Для перегляду товарів замовлення достатньо клікнути на кнопку з кількістю товарів, при наведенні курсору на яку, появляється надпис який сигналізує про можливість показати товари.

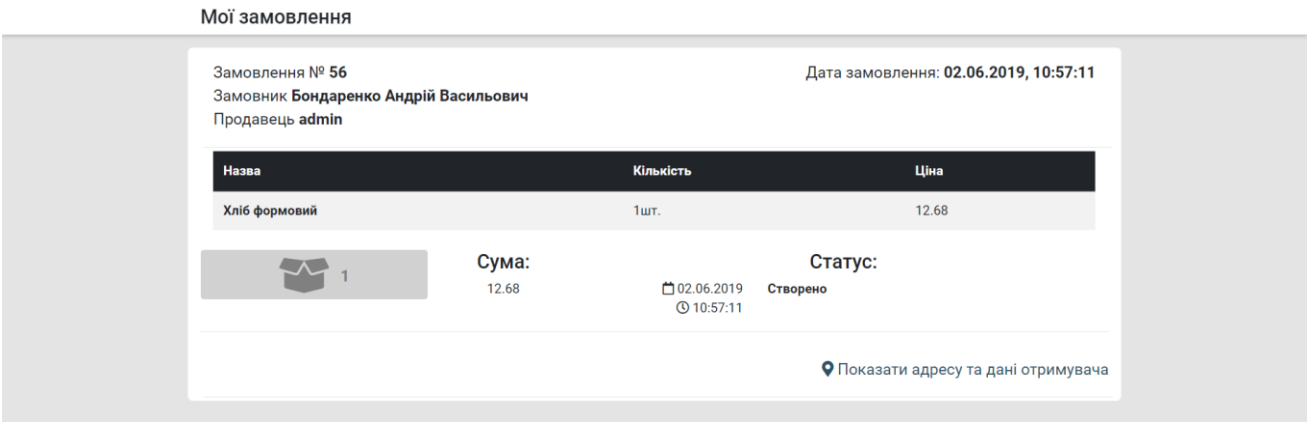

Рисунок 3.28 – Розгорнутий список замовлених товарів

У списку товарів до замовлення користувач може переглянути всі товари, їх кількість та ціну за них у замовленні (рис. 3.28).

У кожному замовленні також доступні статуси, які дають зрозуміти на якому етапі зараз замовлення. У кожного статусу є дата, час та назва статусу. Також кожен статус може мати коментар.

Зі сторони продавця, у якого дане замовлення відображається у спеціальному розділі під назвою «Отримані замовлення», замовлення окрім стандартних даних отримувача, містить інструменти керування замовленням які зображені на рис. 3.29.

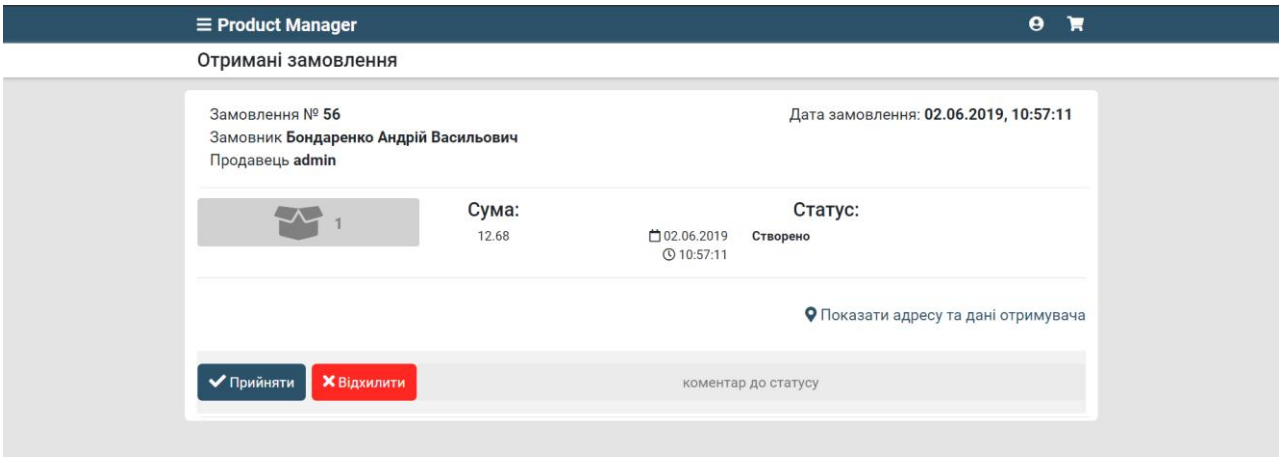

Рисунок 3.29 – Вигляд замовлення зі сторони продавця

Продавець має можливість прийняти, або ж відхилити замовлення. В незалежності від прийнятого рішення статус буде доданий у замовлення. Також продавець має змогу додати коментар до статусу, якщо хоче надати додаткову

інформацію покупцеві. Приклад статусу замовлення із коментарем наведено на рис. 3.30.

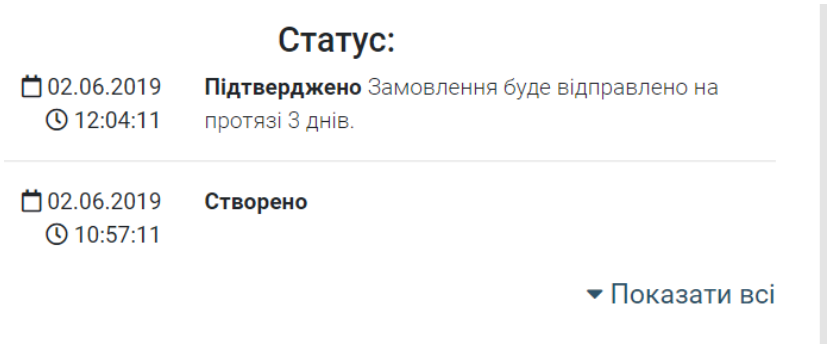

Рисунок 3.30 – Список статусів замовлення

Після того, як продавець прийняв замовлення, він зобов'язується виконати замовлення. Після підтвердження, на панелі керування замовленням змінюються доступні дії над замовленням (рис. 3.31), продавець вже не може відхилити замовлення, так як всі підтверджені замовлення повинні бути виконані. Продавцеві доступна лише одна єдина кнопка «Помітити як відправлене».

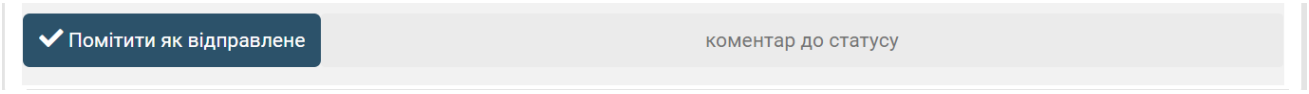

Рисунок 3.31 – Вигляд панелі керування замовленням після підтвердження

Як і раніше, продавець може додати коментар, у випадку з відправленням, можна вказати наприклад відділення пошти, або ж додаткову інформацію. Прикладом статусу відправки замовлення є статус на рис. 3.32, на якому зображені статуси мають додаткову інформацію.

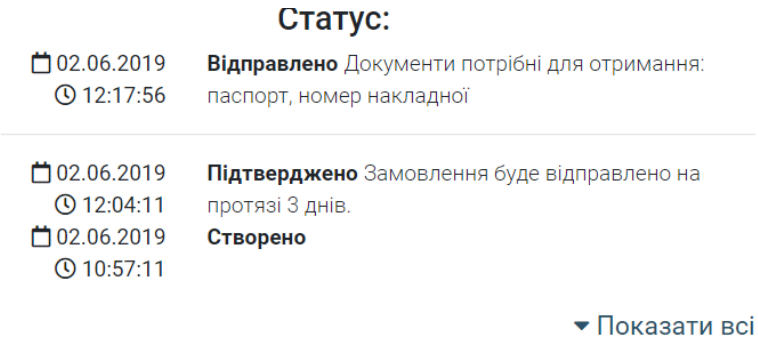

Рисунок 3.32 – Вигляд списку статусів замовлення після відправки

Як видно на рисунку вище, покупець може бачити всі статуси, а також додаткову інформацію до статусів. Також всі статуси посортовані за датою додавання, і покупець може бачити дату, та час додавання статусу.

Як покупцеві, так продавцю доступна адреса відправки. У випадку покупця він може в будь-який момент переглянути на яку адресу, та на кого було оформлено замовлення, а для продавця це інформація, по якій буде здійснюватись відправка товарів. Всі адреси в замовленнях закріплені за замовленнями, та не можуть бути зміненими або ж видаленими. За рахунок такого рішення, замовлення завжди буде цілісним та без втрачених даних.

Щоб переглянути адресу, яка прикріплена до замовлення (рис. 3.33), потрібно клікнути на призначене для цього посилання з назвою «Показати адресу та дані отримувача».

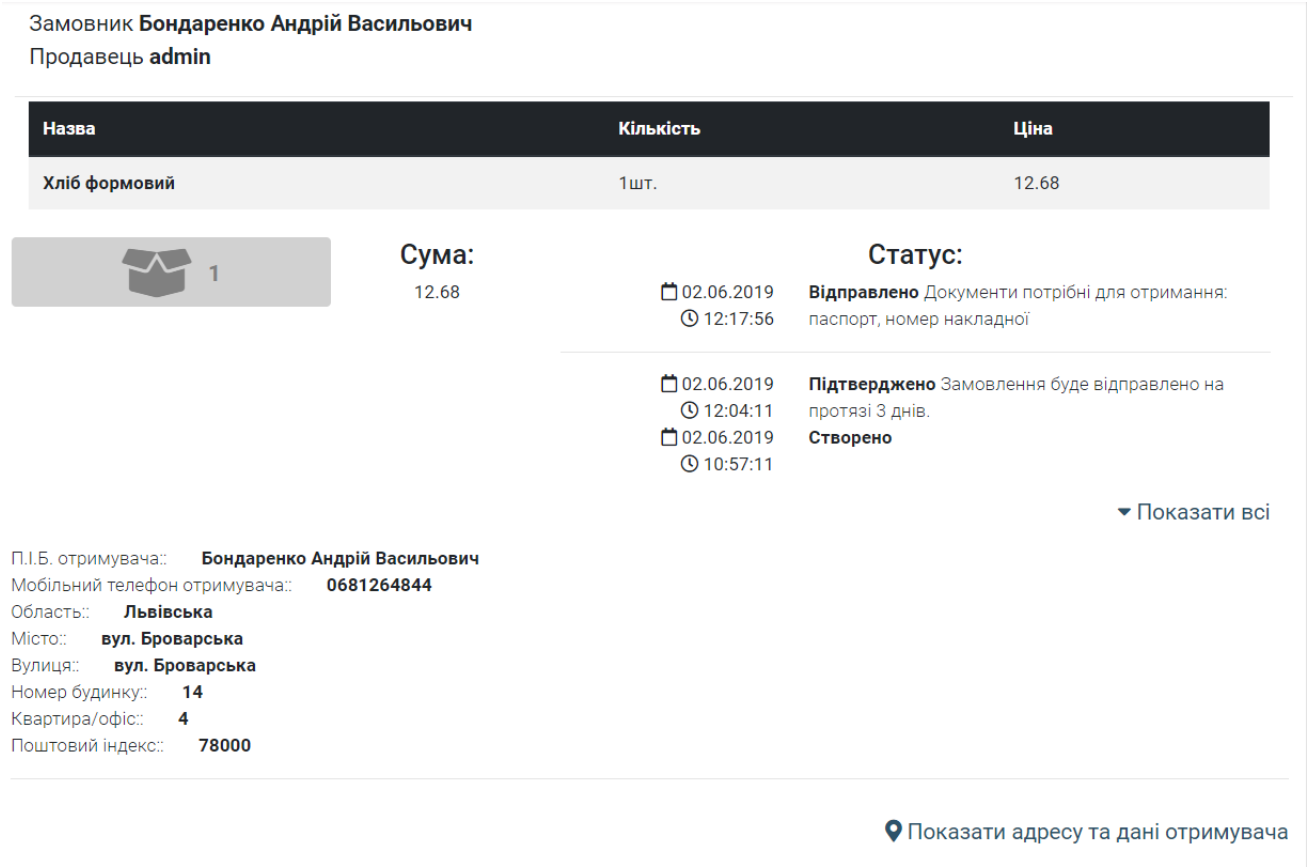

Рисунок 3.33 – Повністю розгорнуте замовлення, зі всіма можливими даними

Кожне замовлення можна розгорнути, і отримати всю інформацію про нього. Можна як відслідкувати, так і перевірити адресу відправки та список замовлених товарів.

Також у проекті реалізований функціонал адміністратора, який дозволяє керувати всіма базовими даними. Адмін. панель знаходиться у меню користувача, яке присутнє при наведенні на іконку користувача, яка знаходиться зліва від кошика (рис. 3.34). Меню користувача присутнє лише у зареєстрованих користувачів. А пункт меню «Адмін. Панель» присутній лише у користувача з правами адміністратора, і звичайні користувачі не буде бачити даного пункту. І не зможе перейти по посиланню, яке ведуть на адмін. Панель.

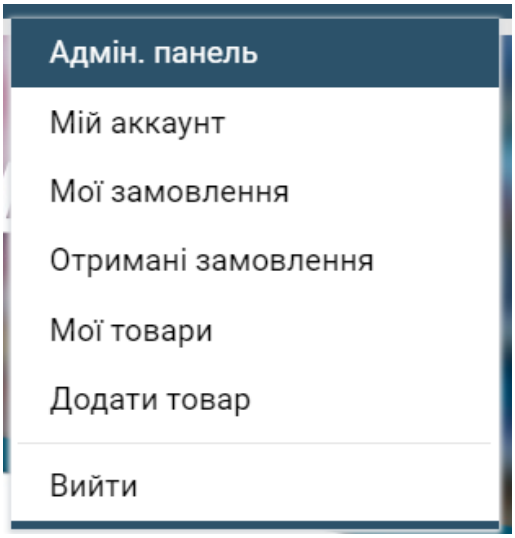

Рисунок 3.34 – Меню користувача

Також у даному меню доступні всі інші сторінки, які були описані вище. Також в меню присутній пункт «Вийти», який дозволяє будь-якому користувачеві вийти з облікового запису.

При переході до панелі адміністратора, користувач потрапляє на сторінку керування категоріями (рис. 3.35), де він може додати нову категорію.

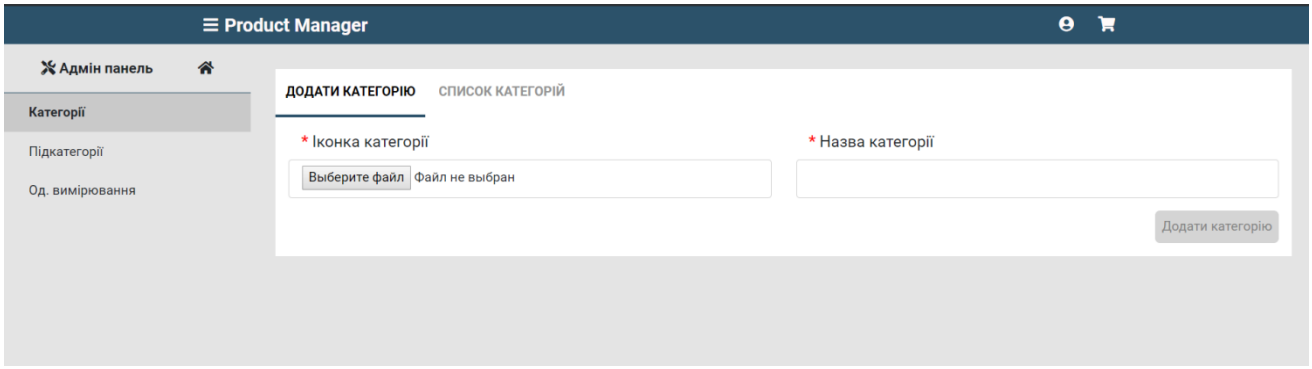

Рисунок 3.35 – Меню користувача

На панелі керування категорія у першій вкладці адміністратор може додати нову категорію.

Форма складається з двох полів, в одному потрібно вибрати фото, а в іншому вказати назву категорії. Назви категорій повинні бути унікальними. Переглянути список категорій можна на вкладці «Список категорій», вміст якої зображений на рис. 3.36. також на тій ж вкладці можна видалити категорію. Видалення можливе лише у випадку, якщо у категорії відсутні підкатегорії, це реалізовано для того, щоб при видаленні категорії, доступ до підкатегорії не було втрачено, так як це приведе до фрагментації даних які не будуть ні до чого прив’язані, що в свою чергу приведе до пошкодження цілісності даних у сервісі, а структури таблиць та зв’язків у базі даних.

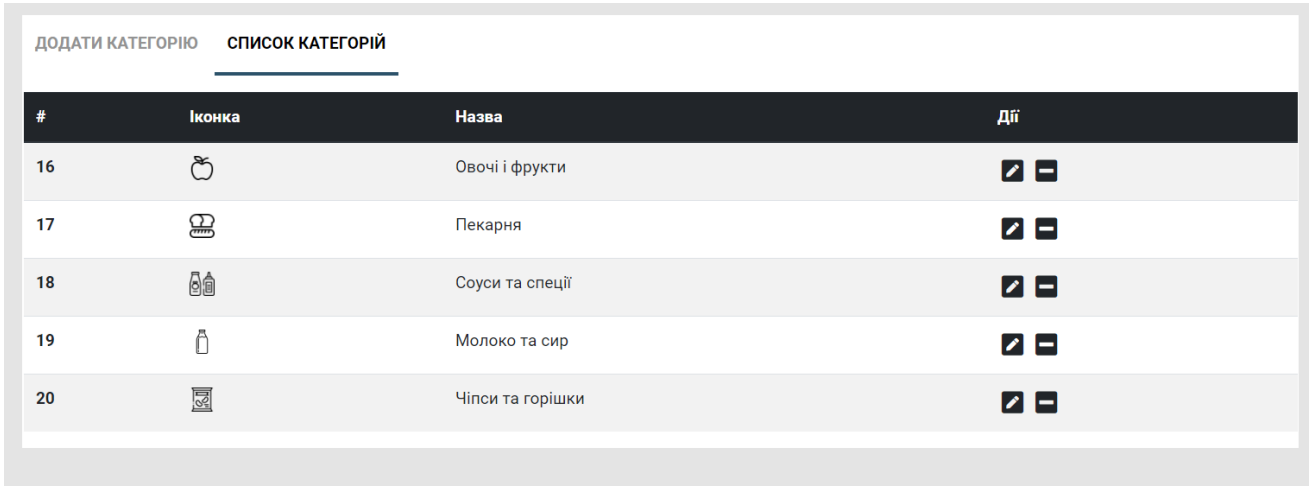

Рисунок 3.36 – Вкладка «Список категорій» у адмін. Панелі

На вкладці, за допомогою таблиці, відображено всі категорії на сайті, та всі дії які над ними можуть бути виконані. Так як категорій буде не багато, вони виводяться на одну сторінку, так як записів у даній таблиці буде не багато.

Наступним пунктом у панелі адміністратора є «Підкатегорії», у пункті першою вкладкою йде додавання нової підкатегорії (рис. 3.37).

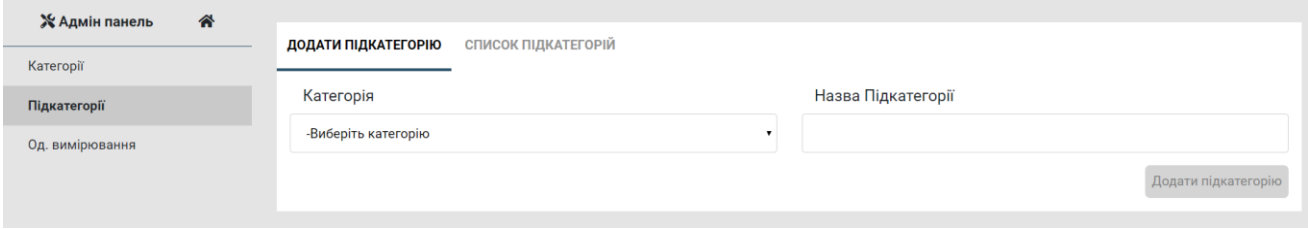

Рисунок 3.37 – Владка «Додати підкатегорію» у пункті «Категорії»

Для додавання нової підкатегорії потрібно вибрати категорію (одну з раніше доданих категорій) а також ввести назву підкатегорії. Призначення даного пункту меню, як і пункту меню з категоріями полягає в тому, щоб адміністратор міг додати категорії і підкатегорії, сама ця дія буде відбуватись не так часто, але можливість робити це – є обов’язковою.

У вкладці «Список підкатегорій» можна переглянути всі підкатегорії (рис. 3.38). Підкатегорію можна видалити лише у випадку, якщо в ній відсутні товари, в іншому випадку це б могло призвести до втрати користувачами даних, товарів. А можливо і замовлень – що є не допустимим. Тому всі підкатегорії можуть бути видаленими лише у випадку відсутності товарів. Редагування ж, доступно в любому випадку, адміністратор може в будь-який момент виправити допущену помилку в імені підкатегорії, або у випадку з категоріями ще й змінити зображення.

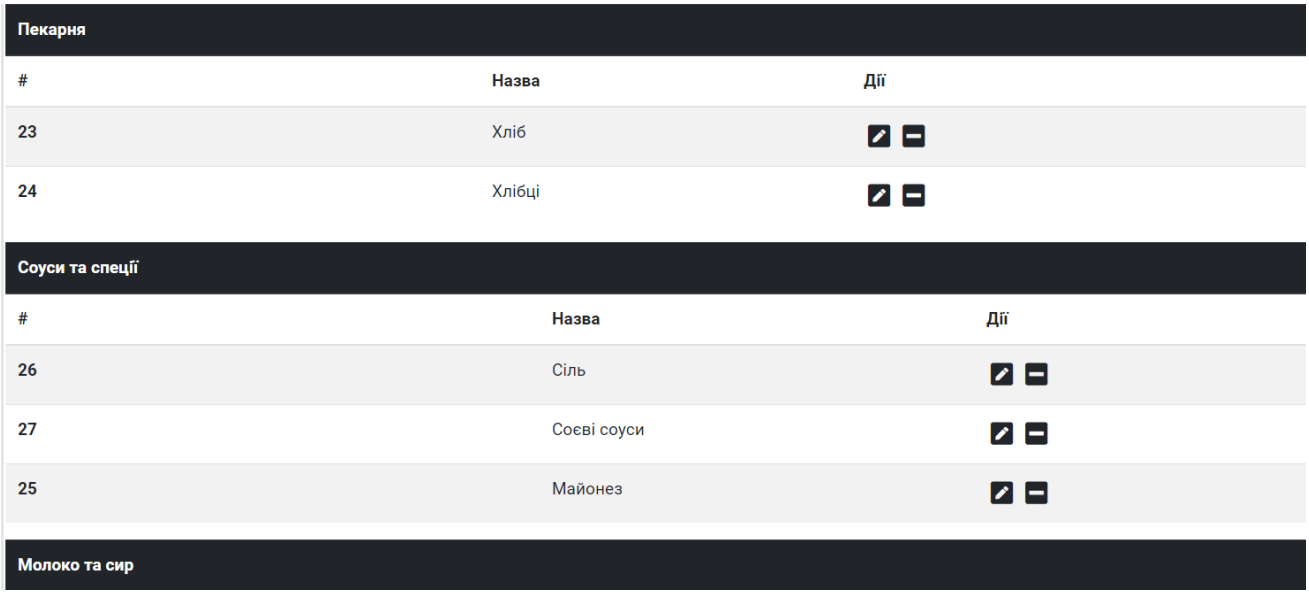

Рисунок 3.38 – Владка «Список підкатегорій» у пункті «Підкатегорії»

Підкатегорії посортовано і згруповано по категоріях, що дозволить швидко знайти потрібну підкатегорію. Розміри колонок таблиць вибираються автоматично, тому можуть відрізнятись у різних групах.

Останнім пунктом у панелі адміністратора є «Од. вимірювання», як зрозуміло по назві, в даному пункті меню виконується керування та додавання одиниць вимірювання (рис. 3.39).

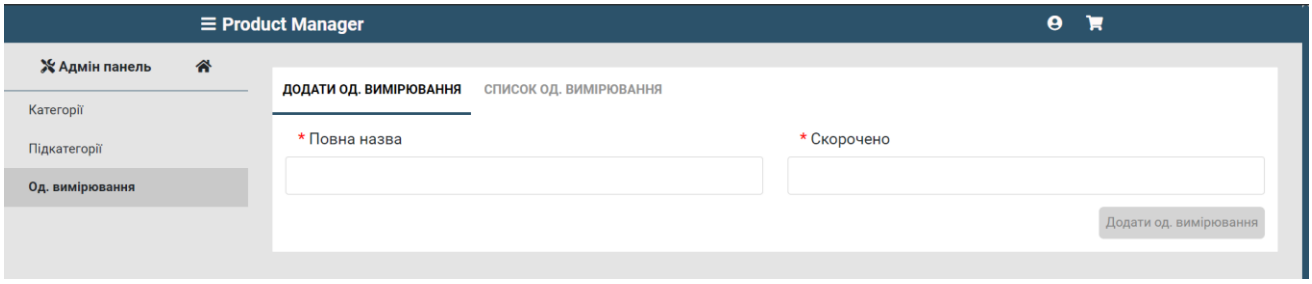

Рисунок 3.39 – Вкладка «Додати од. вимірювання» у пункті «Од. вимірювання»

У кожної одиниці вимірювання є повна назва, і скорочення. У більшості випадків достатньо скорочення, але іноді потрібно вставити повну назву, і саме для таких випадків існує дане поле.

Користувачі бачать список одиниць вимірювання кожного разу, коли додають товар, і дуже важливо щоб на сайті був базовий набір самих популярних

одиниць вимірювання, щоб користувач, який публікує товар знайшов там саме ту одиницю, в якій бажає продавати товар.

Всі одиниці вимірювання можна переглянути на вкладці «Список од. вимірювання» який зображений на рис. 3.40.

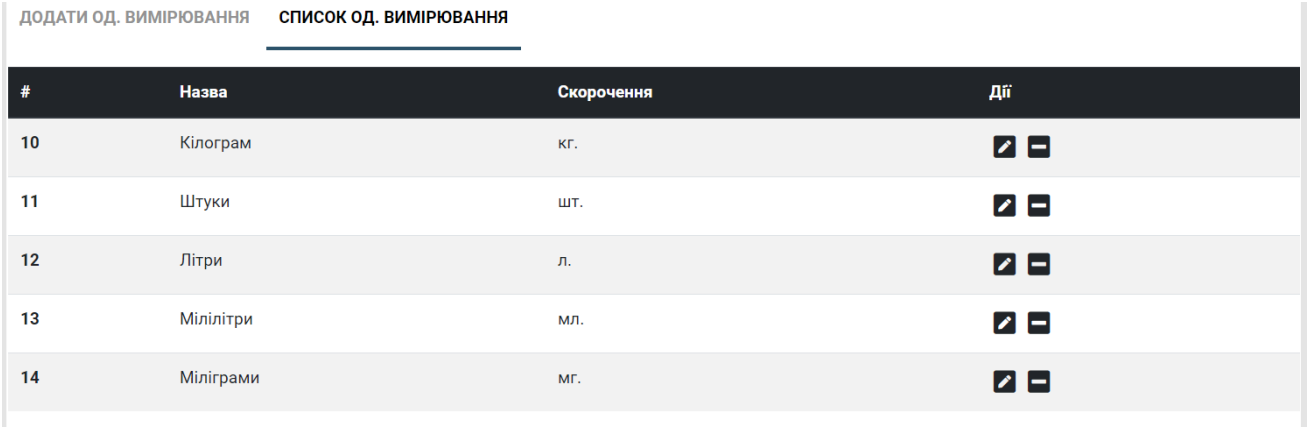

Рисунок 3.40 – Таблиця одиниць вимірювання товарів

Кожен запис у таблиці можна відредагувати або видалити. Чим більше різноманіття одиниць вимірювання присутнє на сервісі, тим більша гнучкість по відношенню до продавців товарів.

Всі вище описані сторінки та сценарії також повністю доступні на будь-яких розмірах екранів, всі дії можна виконати і з смартфону і з комп’ютера, кожна сторінка сайту є адаптивною і її елементи автоматично міняють розміщення в залежності від ширини екрану. Я старався досягти максимальної зручності на всіх пристроях, щоб користувач не був обмеженим у функціональності на будь-якому пристрої.

Багато уваги було приділено обробці помилок та розміру сторінки в цілому, всі компоненти (як видимі так і невидимі) були оптимізовані під високу нагруженість та високу швидкість завантаження сторінок.

ВИСНОВКИ

Підчас виконання дипломного проекту було спроектовано та розроблено back-end частину а також інтерфейс користувача інформаційного веб-сервісу робочих місць користувачів роздрібної торгівлі.

Веб-сервіс реалізований за допомогою мови програмування Java та фреймворку Spring, який сильно спрощує та розширює функціонал мови, в якості бази даних використовувалась MySQL з одноіменною СУБД. В якості основи для інтерфейсу користувача був використаний JavaScript фреймворк Vue.js, який дозволяє проектувати та розробляти динамічний інтерфейс користувача. Для стилізації було використано CSS в парі з препроцесором SCSS, який дозволяє компактно записувати CSS класи з наступним їх перетворенням в чистий CSS код. Для розмітки сторінки був використаний HTML5. Результатом виконання дипломної роботи є повністю функціональний сервер з клієнтською частиною, яку без будь-яких проблем можна розгорнути на хостингу чи локальному сервері.

Тестування додатку проводилось в декілька етапів, та відбувалось від створення перших функціонуючих алгоритмів до повного написання та відладки користувацького інтерфейсу. Підчас написання кожної функції, вона тестувалась на правильність роботи. Після написання функціональної частини (наприклад контролеру авторизації та автентифікації) проводилось тестування всіх методів які входять до даної частини. Наступним проходом по всім функціях було тестування користувацького інтерфейсу, під яке підпадає і тестування функцій сервера. Кінцевим етапом тестування було розміщення веб-сервісу на VPS сервері, та тестування його в реальних задачах.

Дипломний проект виконано у відповідності до завдання і всіма вимогами. Додаток дозволяє адміністратору керувати (додавати, редагувати, видаляти) всіма даними, які відображаються в програмі, реєструватися новим користувачам і відвідувати особистий кабінет, сторінки доданих товарів, та сторінки керування замовленнями.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. OLX. Матеріал з Вікіпедії - вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/OLX (дата звернення: 14.10.2018)
2. Prom.ua. Матеріал з Вікіпедії - вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Prom.ua (дата звернення: 15.10.2018)
3. Липпман С., Лажойе Ж. Язык программирования JavaScript : Вводный курс. С-Пб: Невский проспект, 2006. 1406 с.
4. Marijn Haverbeke-Вячеслав Голованов. Выразительный JavaScript: Введение. Пер. с англ. «Питер», 2016. 433 с.
5. Дэвид Сойер Макфарланд. JavaScript и jQuery: Исчерпывающее руководство. Пер. с англ. М.: Издательский дом «Вильямс», 2017. 450 с.
6. Дэвид Флэнаган. JavaScript : Карманный справочник. Пер. з англ. «Питер», 2015. 600с.
7. Дэвид Флэнаган. JavaScript : Подробное руководство 6-ое издание. Пер. с англ. М.: Издательский дом «Вильямс», 2015. 300 с.
8. Морето С. JavaScript и jQuery: Исчерпывающее руководство. Пер. с англ. Издательский дом «Вильямс», 2017. 880 с.
9. Vue Js. Матеріал з Вікіпедії - вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Vue.js (дата звернення: 25.01.2019).
10. Введение Vue Router. URL: https://router.vuejs.org/ru/ (дата звернення: 28.01.2019).
11. Введение Vuex. URL: https://router.vuejs.org/ru/ (дата звернення: 30.01.2019)
12. Используем Axios для доступа к API - Vue.js. URL: https://ru.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html (дата звернення: 30.01.2019).
13. HTTP-методи запиту - HTTP MDN. URL: https://developer.mozilla.org/uk/docs/Web/HTTP/Methods (дата звернення: 06.02.2019).

14. Основы компонентов - Vue.js. URL: https://uk.wikipedia.org/wiki/JetBrains (дата звернення: 010.02.2019).
15. Java. Матеріал з Вікіпедії - вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Apache_Maven – Назва з титул. екрану. (дата звернення: 05.03.2019).
16. Эккель Б. Философия Java. 4-е вид., СПб : Питер, 2009. 640 с.
17. Хорстманн К. С., Корнелл Г. Библиотека профессионала. Java 2. В 2т. / Том 1. Основы, 7-е вид., Пер. с англ. М.: Издательский дом «Вильямс», 2007. Т.1. 896 с.
18. Хорстманн, К., С., Корнелл, Г. Библиотека профессионала. Java 2. В 2т. / Тонкости программирования, 7-е изд., Пер. с англ. Москва: Издательский дом «Вильямс», 2007. Т.2. 1168 с.
19. Шилдт Г. Полный справочнык по Java : Java SE 6 Edition, 7-е вид. Пер. с англ. М.: Издательский дом «Вильямс», 2007. 1034 с.
20. Spring Framework. Матеріал з Вікіпедії - вільної енциклопедії. URL: https://ru.wikipedia.org/wiki/Spring_Framework - Назва з титул. екрану. (дата звернення: 10.03.2019).
21. Spring Data JPA. URL: https://habr.com/ru/post/435114/ (дата звернення: 11.03.2019).
22. Spring Data JPA - Reference Documentation. URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/. (дата звернення: 13.03.2019).
23. Many-To-Many Relationship in JPA. URL: https://www.baeldung.com/jpa-many-to-many (дата звернення: 13.03.2019).
24. Security with Spring. Матеріал з baeldung. URL: https://www.baeldung.com/security-spring (дата звернення: 20.03.2019).
25. Building an Application with Spring Boot. URL: https://spring.io/guides/gs/spring-boot/ (дата звернення: 21.03.2019).

26. Apache Maven. Матеріал з Вікіпедії - вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Apache_Maven - Назва з титул. екрану. (дата звернення: 21.03.2019).
27. MySQL. Матеріал з Вікіпедії - вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/MySQL - Назва з титул. екрану. (дата звернення: 20.11.2018).
28. Юрчишин В.М., Клим Б.В., Кропивницька В.Б. Організація баз даних: навч. посіб. Івано-Франківськ: Факел, 2009. 224 с.
29. Пасічник В.В., Резніченко В.А. Організація баз даних та знань. Київ: Видавнича група BHV, 2006. 384 с.
30. Implementing JWT Authentication on Spring Boot APIs. Матеріал з Auth0. URL: https://auth0.com/blog/implementing-jwt-authentication-on-spring-boot/– Назва з титул. екрану. (дата звернення: 25.03.2019)

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: Розробка інформаційного веб-сервісу робочих місць користувачів роздрібної торгівлі.

Обсяг пояснювальної записки:. ___83___аркуші

Перелік графічних матеріалів:

- таблиць_____-_____
- рисунків_____44_____
- додатків_____-_____аркушів.

Дата закінчення дипломного проекту: “__”_____2019 р.

Студент – дипломник _____
(підпис) (розшифровка підпису)