Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Сідельник Євген Олександрович
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Інтернет-магазин з продажу молочної продукції дистриб'юторам
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2101103.01.04.ПЗ

Виконав студент ІІІ курсу, група ІПЗс-21-1 _____ Євген СІДЕЛЬНИК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. пед. наук, доцент _____ Оксана ОНИШКО
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

11 червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
02 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сідельник Євген Олександрович
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Інтернет-магазин з продажу молочної продукції дистриб'юторам

Керівник кваліфікаційної роботи Онишко Оксана Григорівна, канд. пед. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 08.01.2024 р. № 6
2. Строк подання студентом роботи на кафедру 01.06.2024 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування системи
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Праворська Н. І. канд. пед. наук, доц.	06.06.24	06.06.24
Антиплагіат	Форкун Юрій Вікторович, доцент	06.06.24	06.06.24

7. Дата видачі завдання «02» січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проектування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12– 31.12.2023	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2024	
3 Проектування програмного забезпечення	21.02 – 20.03 2024	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2024	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2024	
6 Попередній захист КвР	травень 2024	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2024	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2024	

Студент _____ Є.О. СІДЕЛЬНИК
Підпис Ініціали, ПРІЗВИЩЕ

Керівник роботи _____ О.Г. ОНИШКО
Підпис Ініціали, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема дипломного проєкту: «Інтернет-магазин з продажу молочної продукції дистриб'юторам».

Автор роботи: Сідельник Євген Олександрович.

Керівник роботи: Онишко Оксана Григорівна.

Пояснювальна записка 81 с., 30 рис., 3 дод., 30 джерел.

Графічна частина: 20 презентаційних слайдів.

B2B, ІНТЕРНЕТ-МАГАЗИН, PHP, SYMFONY, ORO, POSTGRESQL.

Метою роботи є створення Інтернет-магазину з оптового продажу молочної продукції дистриб'юторам з функціоналом повного циклу створення замовлень з врахуванням специфічних вимог B2B ринку.

В рамках кваліфікаційної роботи було визначено специфіку B2B взаємодій; проаналізовано існуючі рішення даної області; сформовано технічне завдання; спроектовано структуру застосунку у вигляді модульної архітектури, об'єднаної в рамках одного монолітного ядра; обрано оптимальний стек технологій для розробки, враховуючи їх переваги й недоліки.

Для розробки програмної системи використано мову програмування PHP на базі фреймворку Symfony, а також сторонніх пакетів платформи ORO. Для збереження постійних даних використано базу PostgreSQL.

За результатами проектування, виконано програмну реалізацію системи інтернет-магазину відповідно до поставленого технічного завдання; проведено тестування застосунку; написано керівництво користувача.

05 червня 24
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2101103.01.04.ПЗ	Пояснювальна записка	81		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
4	А4		Код програми	11		
5	А4		Презентаційні матеріали	20		
			Графічна частина			
6	А3	КвРІПЗ. 2101103.01.04.Е8	Діаграма варіантів	1		
			використання			
7	А3	КвРІПЗ. 2101103.01.04.Е8	Діаграма розгортання	1		
8	А3	КвРІПЗ. 2101103.01.04.Е8	Діаграма послідовності	1		
			пошуку товарів			

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 2101103.01.04.ВД		
Виконав		Сідельник Є. О.		05.06	Інтернет-магазин з продажу молочної продукції дистриб'юторам Відомість документів	Літ.	Арк. 1 / Аркушів 1
Керівник		Онишко О. Г.		05.06			
Н. контр.		Праворська Н. І.		05.06		ХНУ, ІПЗс-21-1	
Зав. каф.		Бедратюк Л.П.		05.06			

ЗМІСТ

Вступ 5

1 Дослідження предметної області та постановка задачі 7

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей 7

1.2 Аналіз програмно-технічного забезпечення предметної області 11

1.3 Аналіз вимог до програмного забепечення 15

2 Проектування програмного забезпечення 19

2.1 Аналіз та вибір архітектури системи 19

2.2 Проектування архітектури та структури системи 23

2.3 Визначення архітектурних границь системи 27

2.4 Аналіз ти вибір технологій для реалізації програмної системи 29

3 Програмна реалізація та тестування 40

3.1 Втілення архітектури в програмних модулях 40

3.2 Розробка функціоналу промо-цін 45

3.3 Розробка інтеграції з платіжною системою 55

3.4 Технічні характеристики застосунку 60

3.5 Тестування програмного продукту 61

3.6 Інструкція користувача 70

Висновки 78

Перелік джерел посилання 80

Додаток А – Технічне завдання 83

Додаток Б – Лістинг програми 89

Додаток В – Презентаційні матеріали 100

ВСТУП

Оптові онлайн-покупки завжди були актуальним питанням серед підприємців і компаній B2B-сектору. Сучасні технології дозволяють створювати платформи, які значно спрощують процес замовлення великих партій товарів, роблячи його зручним і доступним у будь-який час. Це особливо актуально для ринку молочної продукції, де важливо мати змогу швидко і легко замовити необхідну кількість продуктів, враховуючи всі вимоги та особливості до їх зберігання та транспортування. Завдяки поширенню інформаційних технологій за останні десятиріччя, тепер закупівлю товарів можна здійснювати не виходячи з офісу. Маючи доступ до Інтернет-магазину постачальника, оптові покупці можуть ефективно планувати свої закупівлі, керувати замовленнями та оптимізувати процеси постачання, що сприяє підвищенню ефективності як їх бізнесу, так і бізнесу самого постачальника.

Актуальність теми дипломного проекту полягає в тому, що сучасний бізнес вже активно використовує цифрові технології для оптимізації роботи та розширення своїх можливостей. Враховуючи, що електронна комерція бізнесу до бізнесу (B2B) є однією з найбільших за об’ємом продажів форм електронної комерції, створення Інтернет-магазину для оптового збуту продукції є надзвичайно актуальним.

Згідно прогнозів аналітиків на період з 2024 до 2029 року в Європі відбудеться зростання ринку молочної продукції на 4,15%, досягаючи вартості в 253,08 мільярда доларів, що відображає значний потенціал для економічного розвитку. Особливої уваги заслуговує італійський ринок, де високі обсяги виробництва молочних продуктів можуть бути розширені на експортні ринки. Автоматизація процесів замовлення та управління запасами через онлайн-платформу дозволить бізнесу значно покращити ефективність своєї роботи, забезпечуючи швидкий доступ до потрібних товарів.

Мета проекту полягає в розробці інтернет-магазину для продажу молочної продукції дистриб'юторам з функціоналом повного циклу створення замовлень. Цей магазин дозволятиме користувачам здійснювати оптову закупівлю

молочної продукції, надаючи можливість використання персоналізованих цінових пропозицій, а також функціонал оплати замовлень платіжною системою.

Для досягнення цієї мети були визначені такі завдання:

– провести змістовний аналіз B2B-сегменту ринку для визначення специфік даної області;

– проаналізувати наявне програмно-технічне забезпечення для виявлення його можливостей, переваг та недоліків;

– сформувати перелік вимог до розроблюваної системи;

– ознайомитись з доступними архітектурними моделями та прийняти рішення щодо використовуваної архітектури;

– спроектувати архітектуру інтернет-магазину;

– визначити та обрати оптимальний стек технологій для реалізації програмного продукту;

– розробити систему відповідно до технічних вимог;

– провести тестування системи для виявлення і виправлення помилок;

– підготувати керівництво користувача для забезпечення легкості використання інтернет-магазину.

Результатом даної кваліфікаційної роботи стане функціонуючий інтернет-магазин, який забезпечить оптовим користувачам можливість здійснювати покупку товарів, переглядаючи доступні продукти та оформлюючи замовлення з використанням зручного веб-інтерфейсу.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Аналіз предметної області спрямований на ідентифікацію існуючих проблем і викликів, пов'язаних із впровадженням інформаційних технологій та автоматизацією процесів. Цей процес допомагає зрозуміти, які аспекти можуть бути покращені або вирішені за допомогою розробки нового програмного забезпечення.

Бізнес активно використовує інформаційні технології для оптимізації своїх внутрішніх процесів, а також для забезпечення кращого обслуговування клієнтів. Електронна комерція бізнесу до бізнесу [1] (B2B) є однією з найбільших по об’єму продажів форм електронної комерції, де компанії зосереджені на продажі продуктів та послуг іншим підприємствам. Це включає в себе широкий спектр застосувань, від автоматизації роздрібних магазинів до впровадження програмних реєстраторів розрахункових операцій (ПРРО), які допомагають підприємствам збирати та організовувати дані, а також керувати ключовими аспектами діяльності в єдиній інформаційній системі.

Особливу увагу B2B бізнеси приділяють автоматизації, що дозволяє переходити від ручних операцій до використання сучасних інформаційних технологій у їх оперативній діяльності. Це включає в себе впровадження технологій у сферах, де вони раніше не застосовувалися, що сприяє підвищенню ефективності роботи та залученню нових клієнтів.

Інформаційні технології використовуються B2B бізнесами для створення мереж, де вузли цих мереж є компаніями, а з'єднання між ними представляють партнерства. Важливим є те, що ці взаємовідносини посилюють або навіть визначають цінність продукту компанії. Ці мережі створюють стійкі екосистеми, які підтримують експоненційне зростання, що можна відстежувати через аналіз щільності мережі [2], діаграма якої зображена на рисунку 1.1.

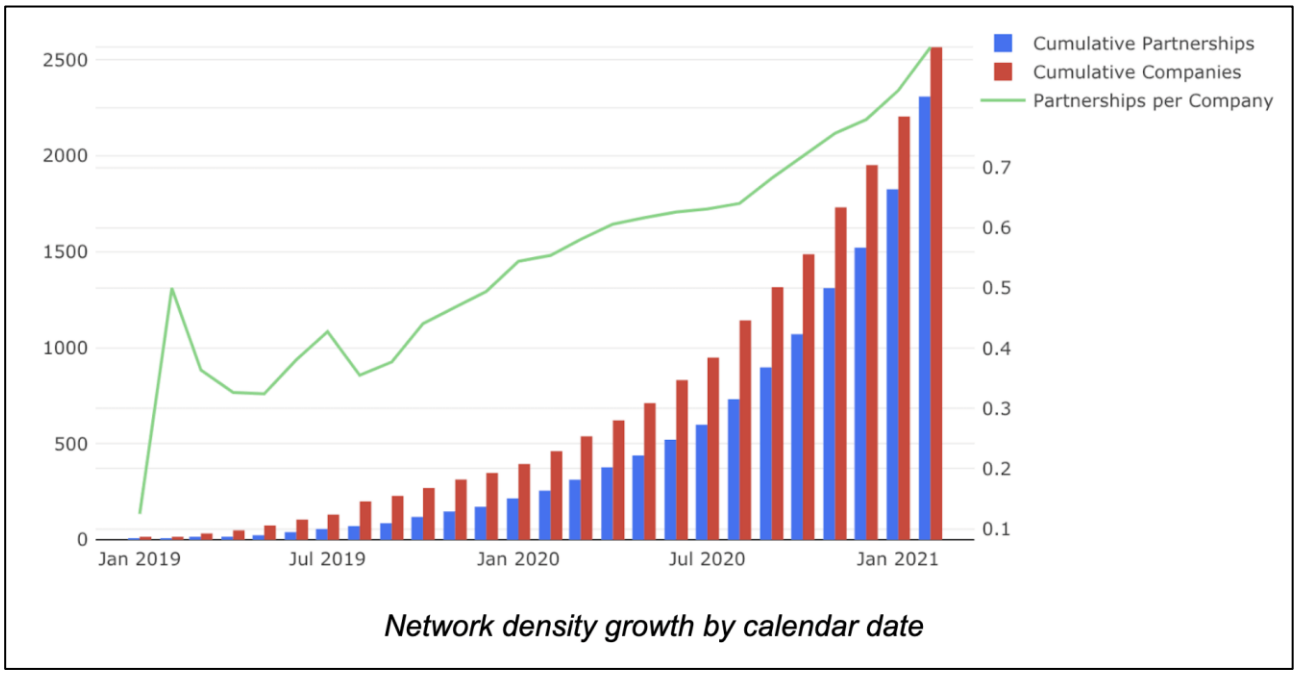


Рисунок 1.1 – Ріст динаміки B2B партнерств в останні роки

Розробка інтернет-магазину, орієнтованого на B2B-сектор набуває особливої актуальності у сучасних умовах ринку, де динаміка взаємодій компаній та потреби оптовиків постійно зростають. Автоматизація процесів замовлення через онлайн-платформу дозволить бізнесу значно покращити ефективність своєї роботи через швидкий доступ до необхідної продукції, сприяючи зміцненню ділових відносин з існуючими партнерами, а також відкриваючи нові можливості для розширення бази клієнтів.

Особливої уваги заслуговує сектор збуту молочної продукції, де тенденції свідчать про зростання попиту на молочні продукти й розширення асортименту, який включає уже не тільки традиційне молоко і сири, але й альтернативні молочні продукти, такі як пробіотики, безлактозне молоко, тощо.

Загалом, в Європі очікується ріст молочного ринку на 4,15% в період з 2024 до 2029 року, досягаючи вартості в 253,08 мільярда доларів до 2029 року [3]. На рисунку 1.2 наглядно зображено порівняння поточної вартості ринку та прогнозу на найближчих п’ять років. Ринок включає широкий асортимент продуктів, включаючи як прямі продукти, такі як молоко, йогурт та морозиво, так і продукти для кулінарії, такі як масло і сир.

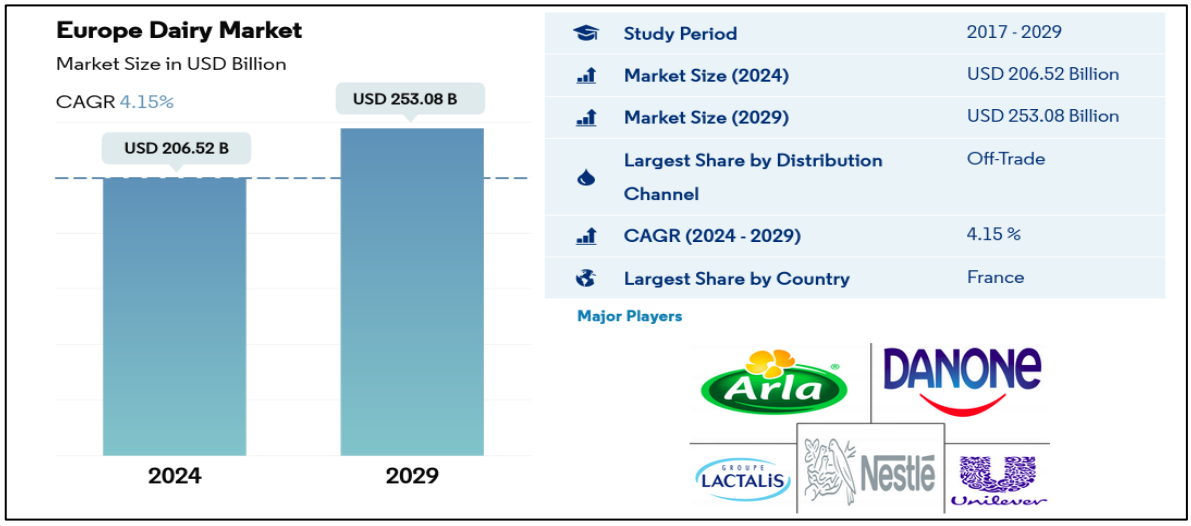


Рисунок 1.2 – Прогнози росту ринку на 2029 рік

Актуальність розробки особливо виражена в контексті італійського ринку, який характеризується високими обсягами виробництва та споживання молочних продуктів, що є результатом зростаючого населення та підвищення рівня доходів. Саме в Італії наразі спостерігається значний потенціал для розширення експортних можливостей. На діаграмі, що зображена на рисунку 1.3 показано, що 64% збуту молочної продукції припадає на внутрішній ринок, а експорт становить лише 36% [4]. Очевидно, що існує великий невикористаний потенціал для збільшення зовнішньої торгівлі через B2B канали.

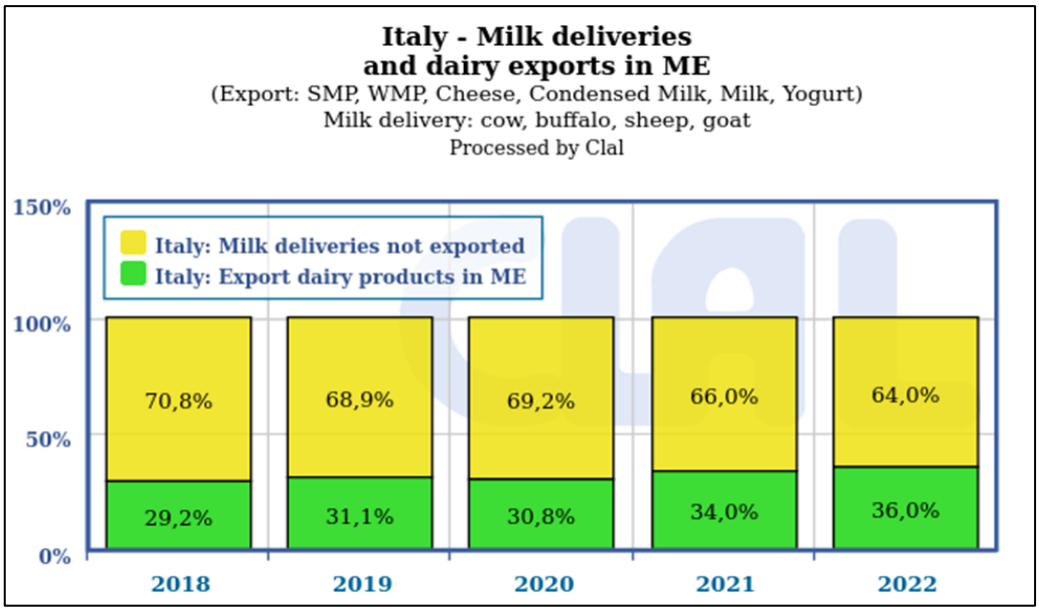


Рисунок 1.3 – Стан експорту в Італії за останні роки

З огляду на статистику за останні роки, розробка B2B-магазину, спрямованого на оптовий збут молочних продуктів з Італії через партнерства з-за кордону, виглядає стратегічно привабливим кроком для італійського бізнесу, який має перспективи для залучення нових ринків та розширення ділової присутності на міжнародному рівні.

Для розробки інтернет-магазину важливим початковим кроком є аналіз предметної області, який охоплює глибоке дослідження ринку молочної продукції, включаючи особливості асортименту та потреби оптових покупців. Визначення викликів, таких як управління запасами та ціноутворення, є ключовим етапом для розробки ефективної системи, яка задовольняла б вимоги даної галузі. Такий аналіз створює міцний фундамент для вирішення проблем якості продуктів та вдосконалення процесів взаємодії з клієнтами через інтернет-магазин, забезпечуючи зручний доступ до інформації про товари та їхнє замовлення.

У процесі аналізу предметної області найпершим кроком є визначення ключових доменних сутностей інтернет-магазину, якими оперує бізнес. Визначення цих основоположних елементів допомагає у формуванні чіткої структури проекту, що спрямована на задоволення потреб оптових покупців і забезпечення ефективної взаємодії між усіма учасниками процесу. Опис ключових сутностей [5] наведено текстом нижче.

Продукт – це основна одиниця асортименту інтернет-магазину, яка представляє молочну продукцію. Кожен продукт може мати свої специфічні характеристики, такі як об’єм, вага, термін придатності, виробник та нормативні показники якості, що важливо для оптових покупців при виборі постачальника.

Категорія – це система класифікації, що дозволяє групувати продукти за певними критеріями, як-от за типом продукції (сир, молоко, йогурт та ін.), виробником або іншими характеристиками. Це спрощує навігацію по сайту та допомагає покупцям легко знаходити потрібні товари.

Список покупок – сутність, що використовуються для створення та управління множинними списками товарів, які зареєстровані користувачі

можуть формувати для закупівлі різного роду товарів. Це дозволяє оптовим покупцям гнучко планувати свої закупівлі, додаючи товари в різні списки залежно від їх потреб, з можливістю подальшого переведення списків покупок в замовлення.

Замовлення – представляє собою фіналізований список товарів, обраних покупцем для купівлі. Процес оформлення замовлення включає в себе вибір всіх товарів зі списку покупок, їх перевірку, внесення інформації про доставку та оплату, а також підтвердження умов обробки замовлення.

Клієнти – компанії або підприємства, які додаються до системи менеджером з продажу через адміністративну панель. Такий підхід дозволяє адміністраторам магазину контролювати реєстрацію нових клієнтів та налаштовувати індивідуальні параметри для кожного з них, включаючи цінові листи, умови оплати та інші специфічні вимоги.

Користувач клієнта – сутність, що моделює представників клієнтських компаній, які мають доступ до інтернет-магазину для управління списками покупок, замовленнями та можливість перегляду історії покупок. Користувачі клієнта мають обліковий запис, за допомогою якого можуть автентифікуватись на сайті та здійснювати оформлення замовлень.

Завершивши структурний та функціональний аналіз предметної області та визначивши ключові особливості, необхідні для розробки інтернет-магазину, можна перейти до наступного етапу, де буде проведено програмно-технічний аналіз предметної області.

1.2 Аналіз програмно-технічного забезпечення предметної області

Вибір програмно-технічного забезпечення є дуже важливим етапом у процесі розробки інтернет-магазину, особливо коли мова ведеться про ринок B2B. На сьогоднішній день, ринок електронної комерції B2B та B2C пропонує широкий вибір програмних рішень, кожне з яких надає унікальні можливості та функції. Вибір найбільш підходящої платформи для розробки інтернет-магазину

безпосередньо впливає на успіх проекту. У цьому контексті особливу увагу заслуговують такі платформи як Sylius, Magento, OpenCart та OroCommerce, кожна з яких має свої переваги та недоліки.

Sylius [6] є сучасною платформою для електронної комерції, розробленою на основі фреймворку Symfony, маючи акцент на гнучкості та легкості у користуванні. Платформа призначена для розробки переважно B2C рішень, маючи потенціал для B2B сектору, адже вирізняється своїм API-орієнтованим підходом, який сприяє легкій інтеграції з іншими системами та сервісами. Для розробки B2B магазину Sylius може запропонувати ефективні інструменти для управління продуктовим каталогом та замовленнями, підтримкою віджетів з статистикою продажів, легкою можливістю інтеграції платіжних систем, що потенційно може бути корисним елементмо для оптових продажів. Однак, як і будь-яка спеціалізована платформа, Sylius має свої недоліки, такі як відсутність вбудованої підтримки повнотекстового пошуку, недостатня гнучкість в функціональності ціноутворення (неможливість налаштовувати ціни під конкретних клієнтів), а це є критичними елеменами для B2B ринку. Панель керування Sylius зображено на рисунку 1.4.

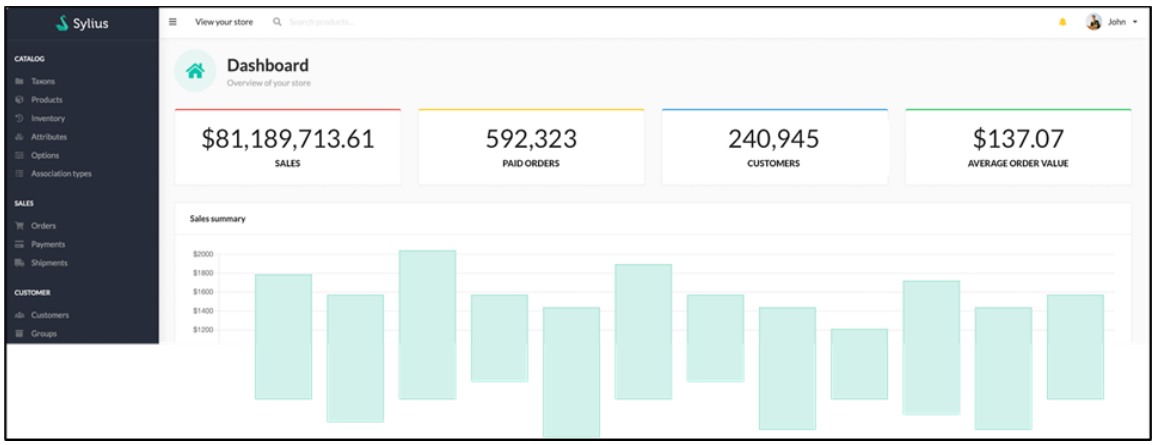


Рисунок 1.4 – Панель керування Sylius

Magento [7], створена компанією Adobe, є одним з лідерів у сфері створення інтернет-магазинів завдяки своїй адаптивності та широкому спектру налаштувань. Платформа ефективно обслуговує як B2C, так і B2B сегменти,

надаючи різноманітні інструменти для керування продукцією, персоналізацію цін та каталогів, а також розширені можливості оплати. Дані функціональні можливості роблять платформу хорошим кандидатом для реалізації магазину, хоча вона і може вимагати значних ресурсів для розробки та обслуговування.

Magento пропонує систему переговорів, яка полегшує взаємодію з вибагливими оптовиками, пропонуючи персоналізоване ціноутворення в ручному режимі та ефективне обслуговування великих замовлень, тим самим підвищуючи рівень задоволеності клієнтів та сприяючи зростанню продажів. На рисунку 1.5 зображено панель керування Magento.

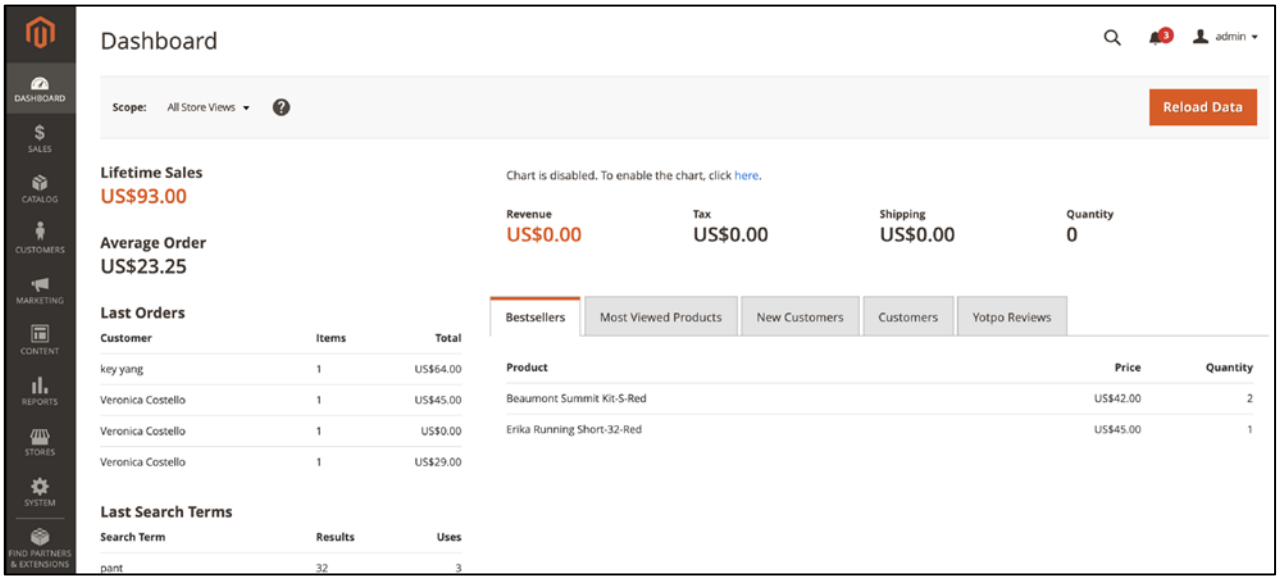


Рисунок 1.5 – Панель керування Magento

OpenCart [8], як безкоштовна та відкрита платформа для електронної комерції, вирізняється своєю простотою у використанні та гнучкістю, що робить її доступною для бізнесів будь-якого розміру. Ця платформа підтримує багатомовність та надає розширений набір звітів та метрик, що дозволяє легко відстежувати стан бізнесу. Однак, вона надає лише базовий функціонал магазину, не враховуючи специфіку B2B взаємодії, а також може створювати певні виклики в питанні розширення функціональності. На рисунку 1.6 зображено панель керування OpenCart.

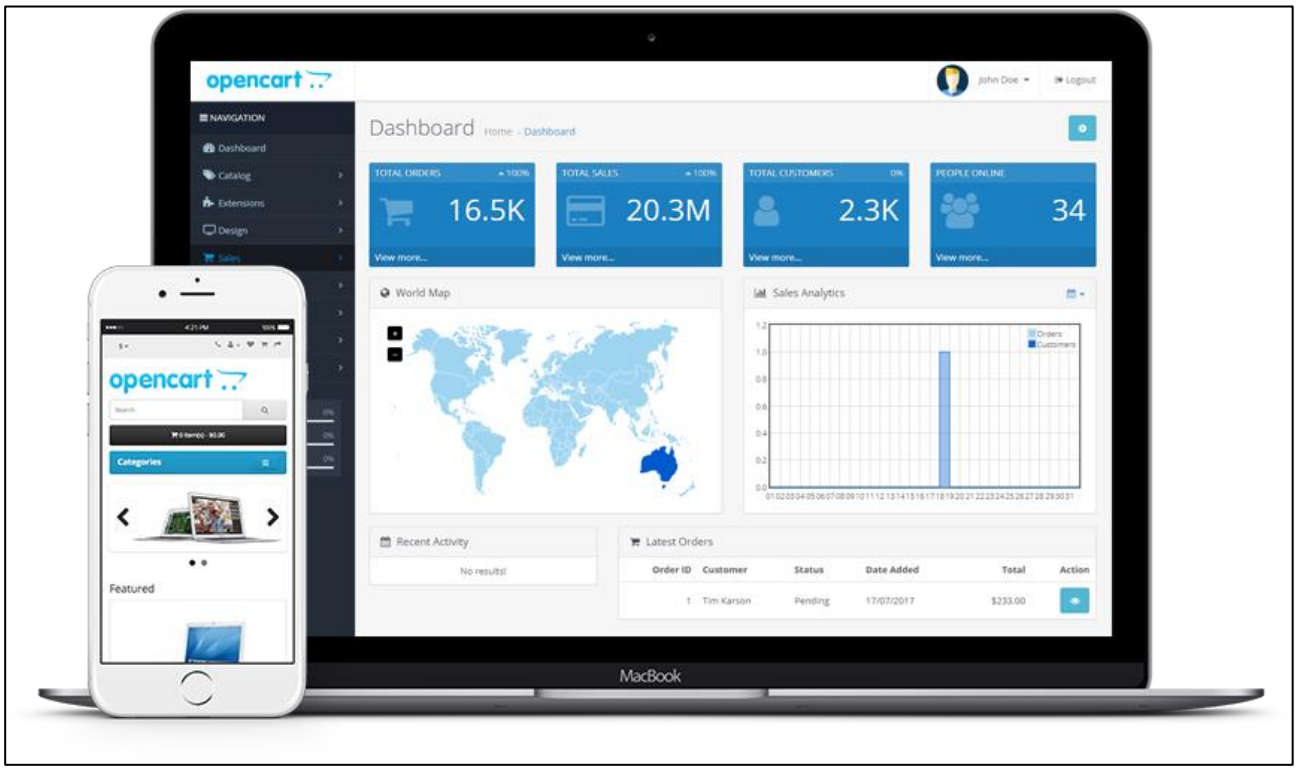


Рисунок 1.6 – Панель керування OpenCart

Одним з ключових рішень, які можуть бути використані для розробки магазину, є платформа OROCommerce [9], розроблена компанією ORO Inc на основі потужного фреймворку Symfony та похідних компонентів, вирізняється своєю гнучкістю та масштабованістю. Платформа славиться своєю спеціалізацією на B2B рішеннях для електронної комерції та CRM системах, специфічних для даного ринку. ORO пропонує інструменти для управління складними процесами ціноутворення та великих замовлень, що є ключовим для оптової торгівлі на зарубіжний ринок.

Спеціалізовані B2B функції включають персоналізовані каталоги для різних груп клієнтів з можливістю повнотекстового пошуку, а також надзвичайно гнучку систему налаштування цін [10]. Оскільки архітектура плафторми є модульною, то інтегрувати специфічний функціонал досить легко.

Крім того, платформа забезпечує високий рівень інтеграції з платіжними системами, надаючи розробникам уніфікований API-інтерфейс, а також реалізує можливість інтеграції інших систем, таких як ERP та CRM, що може бути корисним для створення єдиної інформаційної платформи управління бізнесом.

З недоліків, використання OROCommerce може вимагати значних фінансових та людських ресурсів для налаштування, розгортання та інтеграцій, а також потребує спеціалізованих знань для повноцінного використання всіх її можливостей.

Адміністративну панель ORO зображено на рисунку 1.7.

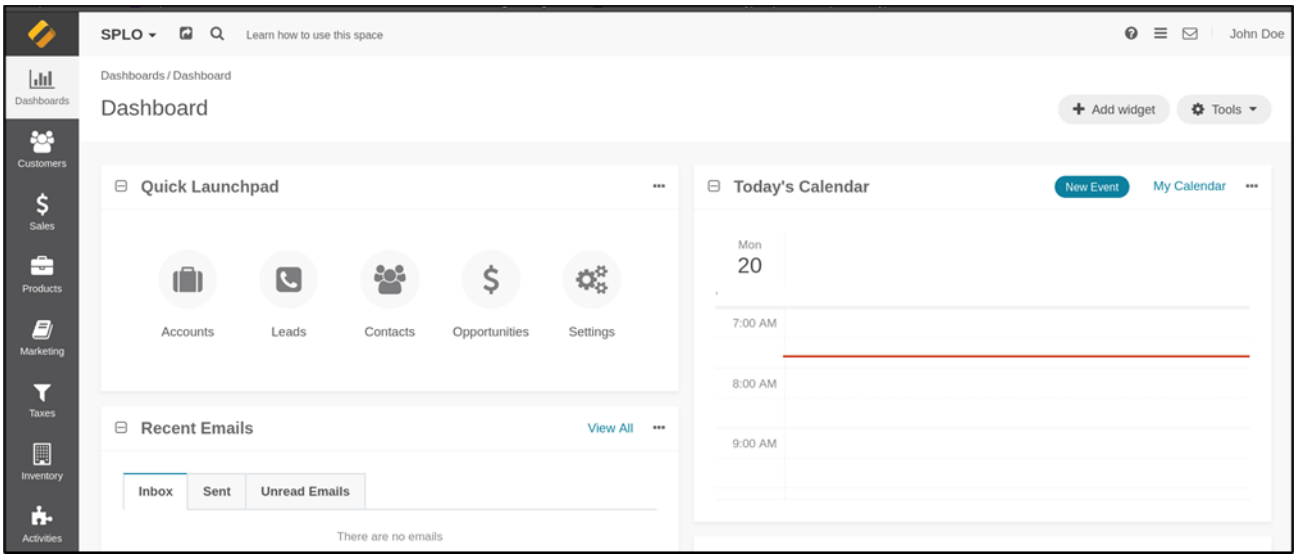


Рисунок 1.7 – Адміністративна панель ORO

Враховуючи спеціалізацію на B2B ринку, гнучкість, можливості інтеграції, високий рівень підтримки зі сторони розробників, а також легкість кастомізації, OROCommerce було обрано як оптимальне рішення для розробки інтернет-магазину оптового продажу молочної продукції. Цей вибір дасть змогу суттєво оптимізувати бізнес-процеси, дозволяючи компанії розвиватися та масштабуватися в майбутньому.

1.3 Аналіз вимог до програмного забепечення

В процесі аналізу вимог до розробки інтернет-магазину оптового продажу молочних продуктів, потрібно сформувати чіткий перелік очікуваних функціональних можливостей для розроблюваної системи. Для формування такого списку варто спершу розглянути можливі сценарії використання системи.

На рисунку 1.8 зображено діаграму варіантів використання функціоналу інтернет-магазину покупцем. Робота з магазином починається з аутентифікації корстувача для доступу до персонального кабінету. Після входу в систему, оптовик має змогу переглянути каталог товарів, скористатися пошуковою системою для вибору потрібних продуктів, а також створити та управляти списком покупок, бачачи проміжну ціну кошика. Фінальнм етапом є оформлення замовлення, під час якого можна оплатити продукти за допомогою спеціалізованої платіжної системи, забезпечуючи безпечний переказ коштів.

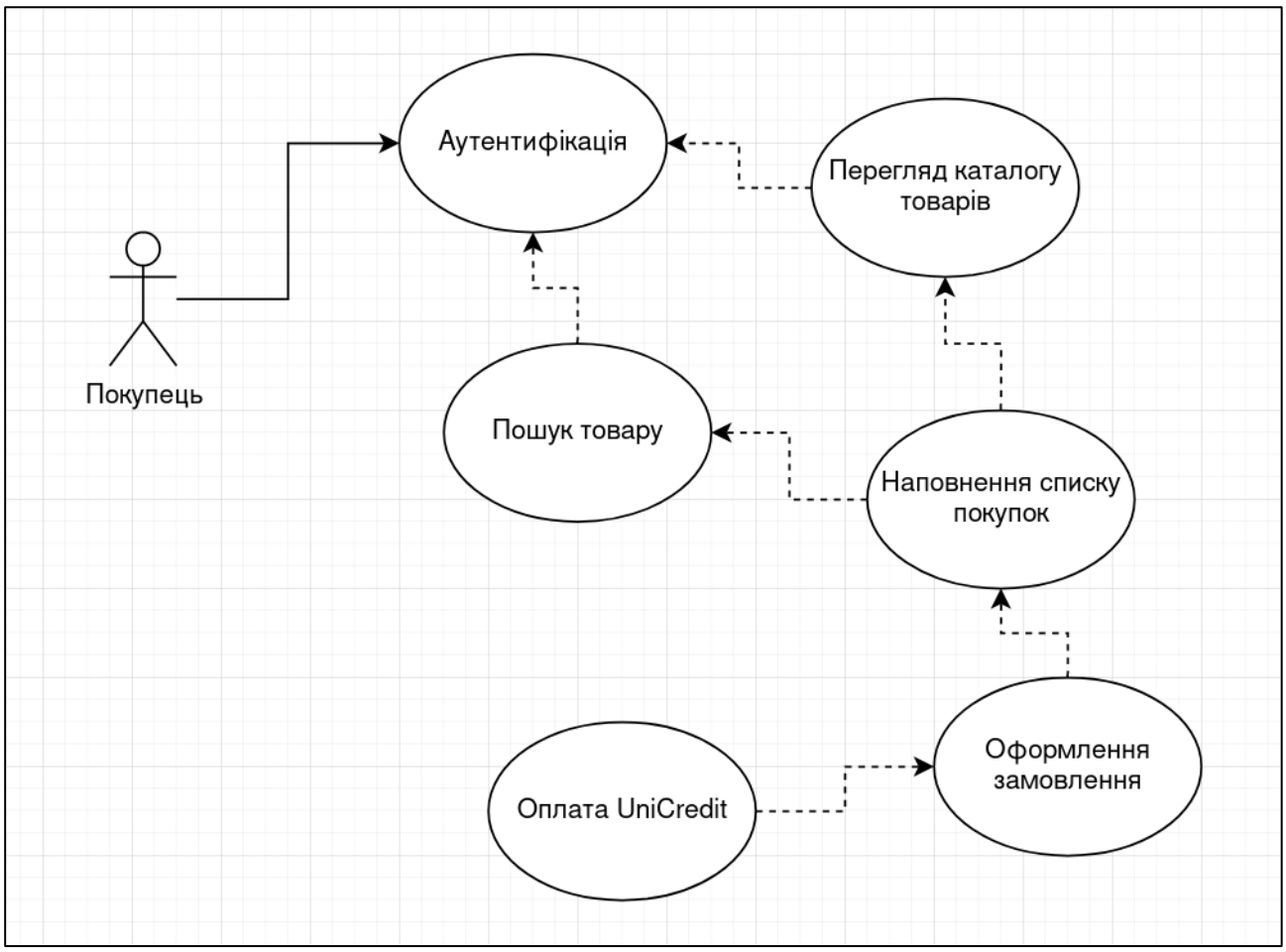


Рисунок 1.8 – Діаграма варіантів використання для оптового покупця

З точки зору користувача, оскільки взаємодія з системою починається з моменту автентифікації, то даний користувач вже повинен мати створений акканут. Для цього передбачається інший адміністративний функціонал системи, де менеджери з продажу можуть додавати й редагувати інформацію про

користувачів системи. Діаграма варіантів використання адміністратором системи зображена на рисунку 1.9, де схематично зображено можливі опції взаємодії з системою, які надаються менеджеру з продажу. Після автентифікації, адмін може реєструвати нових користувачів, управляти каталогом продуктів, задавати персоналізовані ціові пропозиції, а також займатись обробкою оформлених замовлень.

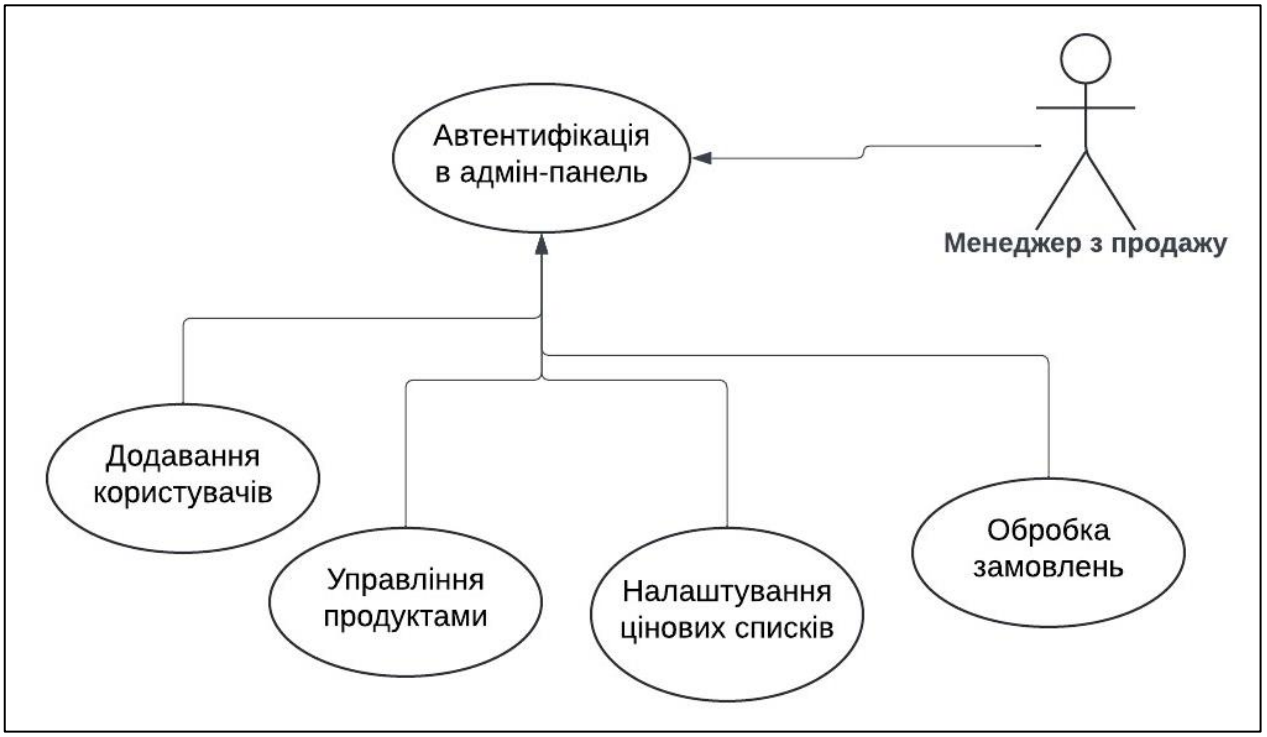


Рисунок 1.9 – Діаграма варіантів використання адміністратора

Відповідно до поданого вище контексту, було визначено перелік вимог до розроблюваної інформаційної системи інтернет-магазину:

– можливість для менеджерів з продажу створювати профілі оптових клієнтів, забезпечуючи індивідуальний підхід до кожного покупця;

– управління асортиментом продукції з можливістю групування по категоріях та налаштування індивідуальних цінових пропозицій;

– можливість повнотекстового пошуку товарів використовуючи ключові слова з опису, надаючи змогу користувачам ефективно знаходити продукцію;

– розробка системи промо-цін, яка дозволяє адміністратору налаштовувати спеціальні цінові пропозиції для обраних клієнтів або груп;

– функціонал для оформлення замовлень з детальним вибором продукції, кількості, а також варіантів оплати замовлення;

– розробка функціоналу екологічного оподаткування продуктів, виробництво яких може негативно впливати на навколишнє середовище;

– забезпечення надійного захисту конфіденційності за допомогою передових технологій шифрування та систем безпеки.

Таким чином, в даному розділі було проведено змістовний аналіз предметної області, який підтвердив актуальність впровадження інформаційних технологій та автоматизації процесів в B2B-секторі, зокрема для молочного ринку. Визначено основні проблеми та виклики, пов'язані з цими процесами, а також ключові доменні сутності, такі як продукти, категорії, списки покупок, замовлення, клієнти та користувачі клієнтів. На основі отриманих даних було сформовано перелік вимог та технічне завдання, яке можна більш детально переглянути в Додатку А «Технічне завдання», що включає функціональні і нефункціональні аспекти системи, які необхідно реалізувати для забезпечення ефективної роботи інтернет-магазину. Розроблена відповідно до даних вимог система спростить процеси управління і стане міцною основою для розвитку та масштабування бізнесу.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз та вибір архітектури системи

В сучасному світі цифрових технологій та електронної комерції, архітектура програмних рішень відіграє ключову роль у створенні ефективних та масштабованих систем. Однією з найбільш розповсюджених мережевих архітектурних моделей є клієнт-серверна архітектура, яка становить основу багатьох веб-застосунків, включаючи інтернет-магазини.

Клієнт-серверна модель включає взаємодію двох ключових елементів: клієнта, яким зазвичай є веб-браузер, що користувач використовує для доступу до системи, і сервера, як потужного обчислювального вузла, який обробляє запити від клієнта, управляє даними і бізнес-логікою, і надсилає відповіді клієнту з необхідним контентом. Ця модель дозволяє відділити інтерфейс користувача від обробки даних, що підвищує гнучкість системи і дозволяє легко масштабувати та оновлювати окремі частини без впливу на інші.

У контексті розроблюваного інтернет-магазину основними компонентами цієї архітектури є клієнт, представлений у вигляді веб-браузера та сервер, що обслуговує веб-сайт інтернет-магазину.

Клієнтська сторона надає інтерфейс для перегляду товарів, управління кошиком покупок, оформлення замовлень та виконання інших дій, пов'язаних з покупкою. Коли користувач виконує будь-яку дію на сайті інтернет-магазину, клієнтська сторона генерує запити до сервера для обробки цих дій.

Серверна сторона, зі свого боку, відповідає за обробку запитів від клієнтів, управління базою даних, включаючи виконання бізнес-логіки, такої як обробка замовлень, розрахунок вартості доставки, знижок тощо, опрацювання інформації про товари, замовлення, користувачів, і в кінцевому рахунку надсилання відповідей на запит назад до клієнтської сторони. Відповіді можуть містити оновлену інформацію для відображення на сторінці, результати пошуку товарів, підтвердження замовлень тощо.

У сфері веб застосунків, для взаємодії між клієнтами та серверами зазвичай використовується протокол HTTP [11] (Hyper Text Transfer Protocol) – стандартний протокол передачі даних в інтернеті, який спрощує процес обміну гіпертекстовою інформацією та іншими типами даних між клієнтськими додатками та серверами.

За допомогою HTTP реалізується обробка запитів від клієнтів. Кожен запит відповідно до цього протоколу складається з методу (до прикладу, GET для отримання інформації або PUT для оновлення даних), URI (Uniform Resource Identifier) для ідентифікації ресурсу, заголовків, що надають додаткову інформацію про контекст запит, а також основної частини – тіла запиту, де містяться дані для передачі серверу.

Протокол HTTP, який лежить в основі взаємодії в інтернет-магазині, оперує на верхньому рівні стеку мережевих протоколів, інкапсулюючи свої дані у TCP-сегменти для транспортування. Принцип інкапсуляції гарантує, що кожен HTTP-запит та відповідь доставляються від клієнта до сервера надійно та у встановленому порядку. Це означає, що HTTP спирається на надійність TCP для управління підтвердженням доставки пакетів, перевіркою помилок і контролем потоку даних між віддаленими хостами, забезпечуючи тим самим стабільну основу для обміну даними в інтернет-магазині.

З огляду на розвиток серверних API та потребу в більш динамічному обміні даними, інтернет-магазини часто вдаються до використання більш високорівневих архітектурних підходів, таких як REST (Representational State Transfer), який оптимізує використання HTTP методів для створення інтуїтивно зрозумілих та легко масштабованих веб-сервісів. RESTful [12] архітектура дозволяє чітко визначати, які дії (створення, оновлення, отримання, видалення) можна виконувати з ресурсами веб-сайту, встановлюючи при цьому чітко визначений набір можливих статус-кодів відповідей, що сприяє зрозумілості в розробці комунікації між клієнтом і сервером.

Так само, як і HTTP інкапсулює свої дані в TCP для транспортування, архітектурний стиль REST розгортається поверх HTTP, використовуючи його

стандартизовані методи для взаємодії з ресурсами веб-сервера. REST описує набір обмежень, дотримання яких веде до створення чітких контрактів взаємодії між клієнтами та сервером. Така інкапсуляція дозволяє використовувати можливості HTTP для реалізації операцій над ресурсами, забезпечуючи легкість в розробці, гнучкість в управлінні станом і спрощення інтеграції між різними системами, які можуть розробляються різними командами розробників.

Концепція REST-архітектури зображена на рисунку 2.1.

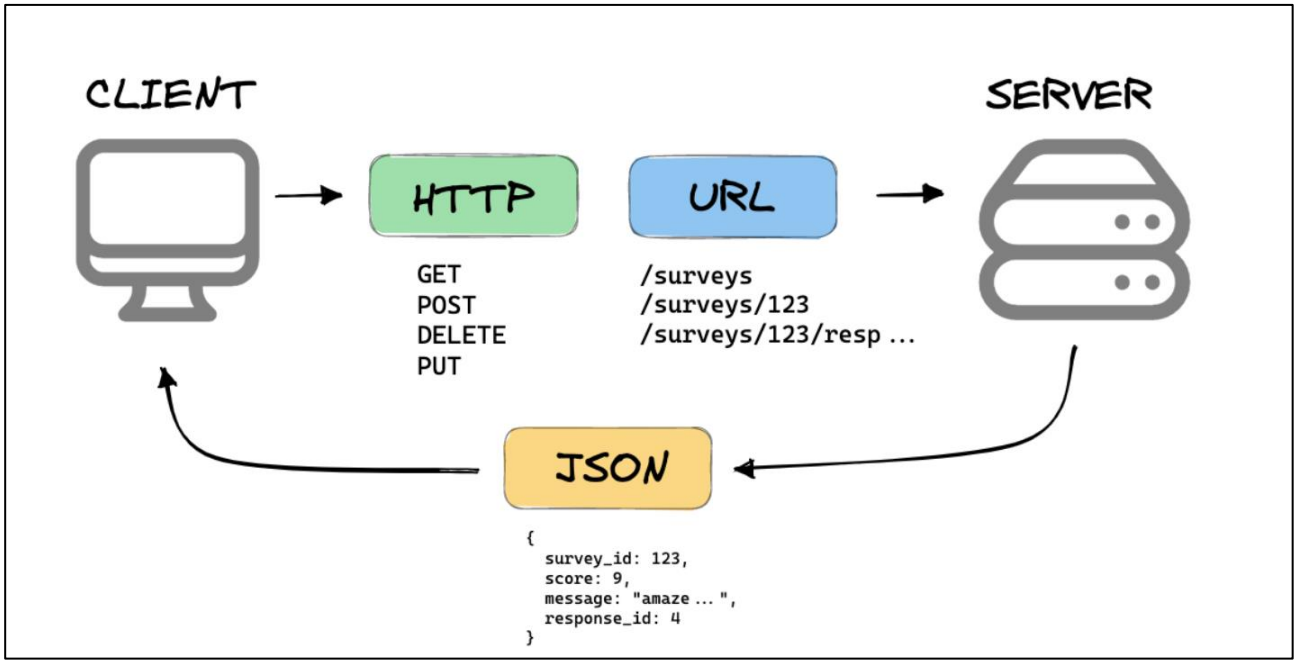


Рисунок 2.1 – Концепція клієнт-серверної REST-архітектури

Зазвичай REST-взаємодія з сервером відбувається через методи GET, POST, PUT, DELETE без збереження стану між запитами, а відповідь сервера може бути у будь-якому форматі, який підтримується клієнтом, включаючи HTML, XML, JSON, бінарні формати тощо. З технічної точки зору, якщо бекенд повертає HTML у відповідь на HTTP запити, дотримуючись принципів REST, тобто використовуючи HTTP методи згідно їх призначення і не зберігаючи стан між запитами, то даний підхід також можна вважати RESTful. Тим не менш, зазвичай коли мова йде про RESTful API, зазвичай мають на увазі використання форматів як JSON або XML, адже їх легше обробляти в клієнтських застосунках.

Слідування принципу безстановості (statelessness) у REST архітектурі сприяє значному спрощенню системи в цілому. Відсутність необхідності зберігати стан сесії користувача на сервері дозволяє кожному запиту бути самодостатнім, інкапсулюючи в собі всю необхідну інформацію для обробки. Це означає, що серверу не потрібно зберігати попередній контекст користувача або стан сесії, що забезпечує високий рівень масштабованості.

Кожен запит клієнта всередині себе містить повний набір даних, що необхідні для його виконання на сервері, включаючи ідентифікатор користувача, зазвичай у вигляді JWT (Json Web Token [13]), що дозволяє серверу обробляти запити в ізольованому середовищі, без необхідності збереження стану між транзакціями. Такий підхід покращує надійність системи та її готовність до обслуговування великої кількості одночасних запитів, що важливо для веб-сервісів, орієнтованих на велику аудиторію користувачів.

Завдяки принципу безстановості в архітектурі REST, система інтернет-магазину може ефективно горизонтально масштабуватися за допомогою розгортання додаткових інстансів бекенду, як наприклад, в середовищі AWS ECS (Amazon Web Services Elastic Container Service [14]), який дозволяє автоматично створювати нові контейнери та розподіляти навантаження між ними по мірі росту навантаження на сервіс, забезпечуючи тим самим неперервну доступність системи. Очевидно, що принцип безстановості дуже сильно спрощує процес управління навантаженням, бо запит може бути оброблений будь-яким доступним розгорнутим сервером.

Таким чином, вибір клієнт-серверної архітектури, побудованої на основі протоколів HTTP та принципів REST, допоможе створити інтернет-магазин, який відповідає високим вимогам до надійності та масштабованості, дозволяючи ефективно керувати взаємодією між сторонами клієнта та сервера, оптимізуючи процес розробки та забезпечуючи легкість у впровадженні змін та оновлень. Обрана архітектура створює міцну основу для гнучкого розширення функціоналу та адаптації до мінливих вимог, що є особливо важливим для динамічного B2B ринку.

2.2 Проектування архітектури та структури системи

При розробці інтернет-магазину для оптового продажу молочної продукції дуже важливо обрати правильний архітектурний підхід, який оптимально відповідатиме потребам проекту. Що стосується організації серверної частини, існує дві ключові моделі: монолітна та мікросервісна архітектура.

Монолітна архітектура передбачає розробку системи як єдиного цілісного застосунку, де всі компоненти інтегровані та взаємодіють в рамках єдиної платформи. Перевагою такого підходу є простота початкової розробки та розгортання, а також легкість у тестуванні, так як всі компоненти системи тісно пов'язані та знаходяться в одному місці. Однак, із зростанням масштабу проекту, монолітна архітектура може стати складною у підтримці та масштабуванні, оскільки зміни в одній частині системи можуть впливати на інші, що збільшує ризик виникнення помилок. Модель даної архітектури зображена на рисунку 2.2.

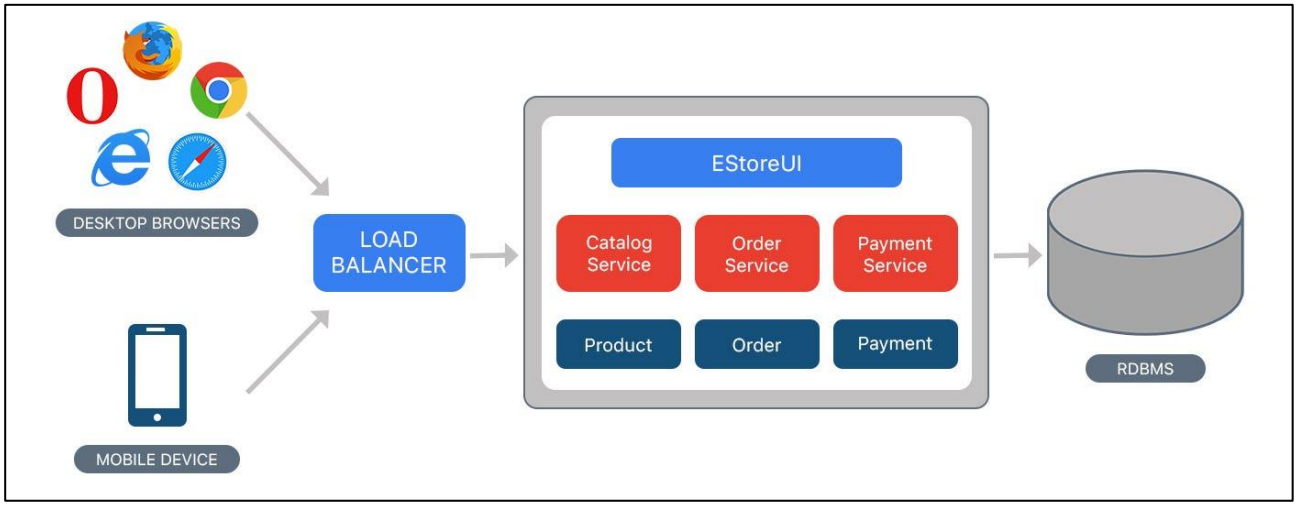


Рисунок 2.2 – Модель монолітної архітектури

Мікросервісна архітектура, з іншого боку, пропонує альтернативний підхід, розділяючи систему на незалежні сервіси, кожен з яких відповідає за певну функціональну область. Це забезпечує високий рівень масштабованості та гнучкості, дозволяючи розробляти, тестувати та розгортати кожен сервіс незалежно. Мікросервісна архітектура також сприяє можливості паралельної

роботи різних команд над окремими частинами проекту. Недоліками ж цього підходу є підвищена складність управління зв'язками між сервісами та потреба в більш складній інфраструктурі для оркестрації сервісів. Модель даної архітектури зображена на рисунку 2.3.

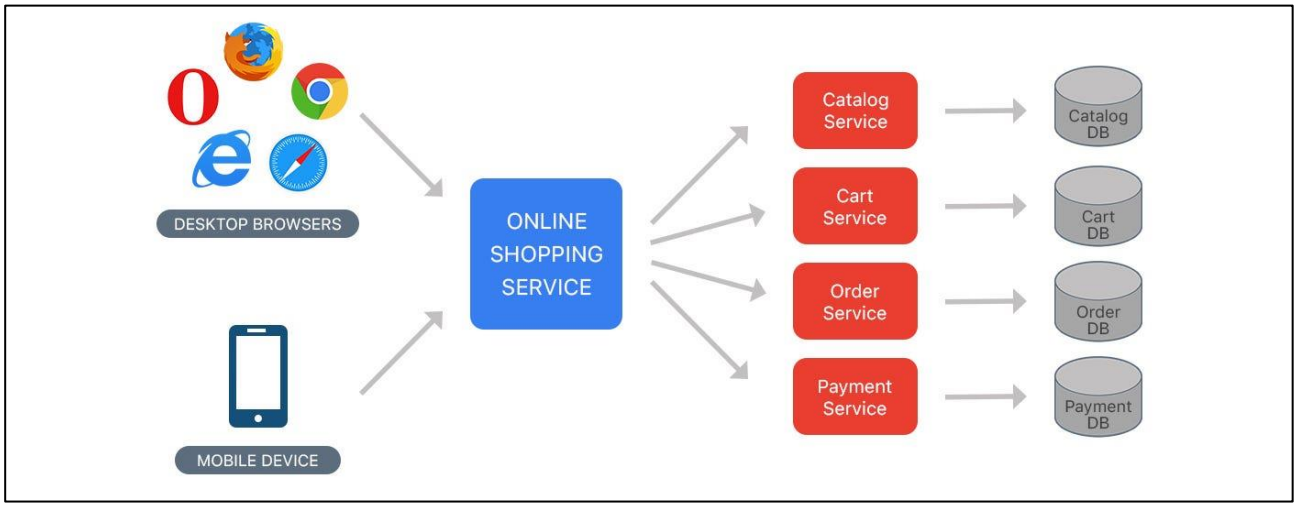


Рисунок 2.3 – Модель мікросервісної архітектури

Для розроблюваного проекту B2B інтернет-магазину було вирішено обрати монолітну архітектуру з поділенням на функціональні модулі. Вибір мотивований тим, що фреймворк ORO, який лежить в основі системи, надає готову до використання структуру з великою кількістю підключуваних модулів, що спрощує розробку та інтеграцію необхідного функціоналу. Монолітна архітектура забезпечить легкість розробки та розгортання, що є важливим аспектом для дотримання встановлених термінів проекту. Обраний підхід також сприятиме зменшенню складності інтеграції з існуючими рішеннями та забезпечить більш просте управління системою в цілому.

Тим не менш, в процесі розвитку інтернет-магазину все ж допускається можливість переходу від модульного моноліту до мікросервісної архітектури. Експерти індустрії, такі як Роберт Мартін [15] та Мартін Фаулер [16], підтримують ідею початкової реалізації проекту з використанням модульного підходу до монолітної структури. Реалізувавши таким чином мінімально життєздатний продукт, можна краще зрозуміти предметну область та

особливості вимог системи без зайвих зусиль щодо підримки мікросервісів. Втім, для збереження незалежності модулів в монолітній архітектурі, необхідна висока технічна дисципліна. Важливо чітко визначити межі кожного модуля та уникати появи тісних зв'язків між ними, щоб забезпечити їх незалежність і спростити потенційний перехід на мікросервіси.

У контексті цієї стратегії розвитку, система інтернет-магазину складатиметься з кількох ключових модулів, кожен з яких обслуговуватиме специфічну область функціоналу. Ці модулі включатимуть автентифікацію користувачів, керування каталогом продуктів, систему пошуку, управління кошиком покупців, оформлення замовлень, компонент фінансових операцій, а також адміністративні функції для управління системою. Кожен модуль розробляється таким чином, щоб забезпечити легкість інтеграції з іншими частинами системи та зберегти можливість майбутнього відділення у вигляді незалежних мікросервісів.

Модуль автентифікації забезпечує безпечний вхід користувачів до інтернет-магазину, використовуючи логін та пароль. Він відповідає за верифікацію ідентифікаційних даних, надаючи доступ до функціоналу лише автентифікованим користувачам. Цей модуль також надає можливість відновлення паролю та зміну користувацьких даних, забезпечуючи високий рівень безпеки даних користувачів.

Модуль каталогу продуктів є основоною частиною інтернет-магазину, де представляється асортимент молочних продуктів. Він дозволяє систематизувати товари за категоріями та характеристиками, надаючи детальні описи та зображення для кожного продукту. Цей модуль спрощує навігацію користувачів по сайту та допомагає їм легко підібрати потрібний товар.

Модуль пошуку в інтернет-магазині дозволяє користувачам швидко знаходити продукти за ключовими словами, описом або категоріями. Він використовує алгоритми повнотекстового пошуку для ефективного відображення відповідних результатів, сприяючи поліпшенню користувацького досвіду та зручності покупок.

Модуль кошика покупця дозволяє користувачам групувати обрані товари перед оформленням замовлення. Він забезпечує функціонал для додавання, видалення або зміни кількості продуктів у кошику, а також взаємодію з модулем фінасових операцій для загального підрахунку вартості товарів. Даний модуль є важливим інструментом для забезпечення зручності покупок, адже зазвичай після наповнення кошика, дані продукти мігрують в модуль оформлення замовлень, де їх можна буде оплатити.

Фінансовий модуль відповідає за управління ціноутворенням, включаючи екологічне оподаткування, функціонал промо-цін, а також логіка формування індивідуальних цінових пропозицій для різних категорій користувачів. Цей модуль забезпечує гнучке налаштування цін та знижок, а також автоматизує процеси розрахунку вартості замовлень.

Модуль оформлення замовлення керує підтвердженням покупки та включає інтеграцію з платіжною системою для зручності оплати товарів. Користувачі можуть вибирати способи доставки та платежу, переглядати загальний підсумок замовлення та, погодившись з умовами користування, завершити процес покупки через платіжну платформу.

Адміністративний модуль дозволяє адміністраторам реєструвати нових користувачів, заповнювати їхні платіжні адреси та налаштовувати індивідуальні цінові пропозиції, екологічні податки та промо-акції. Модуль надає інструменти для всебічного управління магазином з акцентом на опрацювання замовлень, сприяючи ефективній взаємодії з користувачами та оптимізації процесів продажу.

Таким чином, в рамках проектування B2B інтернет-магазину, було зроблено вибір на користь монолітної архітектури, виходячи із зручностей, які пропонує платформа ORO, що забезпечує міцну основу для майбутнього росту та розвитку системи, спрощуючи процес розробки та інтеграції. Функціональність магазину була декомпозована на ключові модулі, які відповідають за різні аспекти роботи системи, забезпечуючи легкість внесення змін та додавання нових функцій відповідно до принципу SRP [17].

2.3 Визначення архітектурних границь системи

У розробці системи інтернет-магазину, важливу роль відіграє принцип ізоляції модулів, який забезпечує, що кожен з ключових компонентів системи виконує свою специфічну функцію, маючи мінімальну залежність від інших компонентів. Для цього потрібно, щоб взаємодія між модулями відбувалася через чітко визначені інтерфейси, що дозволить уникнути непередбачуваного впливу змін в одному модулі на роботу інших частин системи.

Таке відокремелення компонентів сприяє полегшенню тестування, підтримки та розвитку кожного модуля окремо, плавне розширення системи, а також забезпечує можливість майбутнього переходу на мікросервісну архітектуру без значних перебудов. Крім того, даний підхід є важливим складником у забезпеченні безпеки системи, оскільки обмежує потенційний вплив вразливостей одного модуля на інші частини проекту.

Забезпечення ізоляції модулів у системі інтернет-магазину має першочергове значення для стабільності та надійності платформи. Для реалізації даної мети, можна застосувати спеціалізовані інструменти, такі як Deptrac [18] та PHPAT [19] (PHP Architecture Tester). Ці інструменти дозволяють аналізувати код на предмет дотримання визначених архітектурних правил та забезпечують можливість виявлення й усунення небажаних залежностей, які можуть порушувати принципи модульності та гнучкості системи.

Deptrac, до прикладу, дозволяє розробникам визначати архітектурні рівні в системі, а також правила, які вказують, які рівні можуть залежати один від одного. Це допомагає запобігти небажаних залежностей одного модуля від функціоналу іншого, що сприяє збереженню чистоти коду та архітектури.

Аналогічно, PHPAT працює як інструмент тестування, що дозволяє визначати архітектурні правила та перевіряти їх дотримання в коді. Написані правила можуть контролювати які простори імен можуть взаємодіяти між собою, а які не можуть. Це також сприяє підтримці чітко визначеної структури за рахунок автоматизованої перевірки архітектурної цілісності системи.

Використання даних інструментів для контролю залежностей між модулями в інтернет-магазині не тільки підвищить якість коду, але й спростить підтримку та розвиток проекту, дозволяючи легко адаптувати систему до нових вимог без ризику порушення архітектурних принципів та супутніх проблем.

У розроблюваній системі, яка має монолітну архітектуру з попередньо визначеними модулями, варто також визначити правила залежностей між цими шарами системи, щоб забезпечити ефективне функціонування та легку підтримку. У контексті налаштування конфігурації інструмента Deptrac, кожен визначений модуль виступає як окремий шар. Це означає, що автентифікація, каталог продуктів, пошук, кошик покупця, оформлення замовлення, фінансовий модуль та адміністративний модуль — кожен з них конфігурується як ізольований шар у файлі налаштувань Deptrac. Так можна вказати залежностями між модулями, визначаючи, які з них можуть залежати один від одного та які зв'язки заборонені.

Модуль автентифікації повинен бути незалежним, але інші модулі, такі як модуль оформлення замовлень, модуль кошик покупця та інші модулі, які потребують функціонал автентифікації, можуть залежати від нього для верифікації користувачів. Хоча, з іншого боку, також є можливий варіант досягнення повної незалежності від даного модуля, якщо використовувати JWT-автентифікацію для всіх сервісів системи.

Модуль каталогу продуктів має служити основою для пошукового модуля та модуля кошика покупця, надаючи їм необхідні дані про продукти, але не повинен мати прямих залежностей від інший компонентів.

Модуль пошуку може взаємодіяти з модулем каталогу продуктів для отримання інформації, але не має взаємодіяти безпосередньо з модулем оформлення замовлень, та іншими пакетами.

Модуль кошика покупця може взаємодіяти з модулем каталогу продуктів для додавання товарів до кошика і з фінансовим модулем для розрахунку вартості. Втім цей модуль не повинен мати прямих залежностей від модуля оформлення замовлення.

Модуль оформлення замовлення має інтегруватися з модулем кошика покупця для фіналізації покупки, та з фінансовим модулем для здійснення оплати замовлення.

Фінансовий модуль повинен служити центральним компонентом для обробки ціноутворення, податків та промо-цін. Із ним має інтегруватися модуль оформлення замовлення, але сам він не повинен мати прямих залежностей від інших компонентів системи.

Адміністративна частина системи може мати зв’язки з усіма модулями для управління системою, але ці зв'язки мають бути односторонніми, щоб уникнути небажаного впливу на основну бізнес-логіку.

Таким чином, в рамках проектування застосунку було ретельно проаналізовано та визначено можливі допустимі залежності між компонентами системи, що сприяє дотриманню чітких правил взаємодії між модулями, за рахунок чого вони залишається ізольованими, а система легко розширюваною.

2.4 Аналіз ти вибір технологій для реалізації програмної системи

Детальний аналіз та вибір технологій, які будуть використані для реалізації розроблюваного інтернет-магазину є важливим кроком в проектуванні програмної системи, оскільки правильний вибір технологій впливає на швидкість розробки, легкість підтримки та подальшу можливість адаптації до змінних вимог ринку. В рамках даного аналізу буде розглянуто ряд технологій відповідно до таких критеріїв вибору як зрілість, сумісність із вимогами проекту, простота використання, а також наявність та якість підтримки зі сторони спільноти розробників. Ретельне опрацювання даних аспектів допоможе вибудувати надійну реалізацію програмної системи відповідно до спроектованої архітектури.

Для реалізації інтернет-магазину можуть вкористовуватись різні мови програмування, однак найчастіше в якості основної мови вибір падає на PHP [20], через її спеціалізацію на веб-розробці та широкому розповсюдженню.

PHP є мовою загального призначення, оптимізованою для створення веб-додатків і може бути легко вбудована у HTML код, що робить її прекрасним варіантом для швидкої розробки проектів, де взаємодія з користувачами відбувається через веб-інтерфейс.

Завдяки великій спільноті розробників, PHP забезпечує багатий набір фреймворків та готових рішень, які можуть значно прискорити розробку та знизити витрати на підтримку. Фреймворки Laravel та Symfony пропонують готові компоненти для реалізації стандартних функцій веб-додатків, включаючи автентифікацію, роботу з базами даних, кешування, а також ORM.

Мова також характеризується високою гнучкістю і простотою роботи, що дозволяє дость просто поєднувати її з різноманітними базами даних і технологіями. Крім того, проект на PHP можна легко розгорнути на більшості веб-серверів та платформ через її широку підтримку зі сторони хостинг-провайдерів. Важливо також зазначити, що для мови існує має багато готових інтеграцій з платіжними системами та іншими зовнішніми сервісами, що робить її зручною для створення комерційних платформ.

В ході аналізу існуючих рішень для створення інтернет-магазину було розглянуто такі платформи як Sylius, Magento, OpenCart та OroCommerce, кожна з яких пропонує свій набір функцій електронної комерції відповідно до спеціалізації. В якості фундаменту для розробки, було обрано платформу ORO, оскільки вона найкраще відповідає вимогам B2B-ринку.

Будучи побудованою на базі потужного PHP фреймворку Symfony, ORO надає широкі можливості для створення комплексних B2B рішень, відповідаючи специфічним потребам цього сектору. Тому, зважаючи на багатий функціонал платформи ORO, було прийнято рішення щодо використання мови PHP та фреймворку Symfony як найбільш оптимальних технологій для розробки.

Після вибору мови програмування PHP та платформи ORO в якості основи інтернет-магазину, особлива увага приділяється фреймворку Symfony [21], на якому базується OROCommerce. Широкі можливості фреймворку надають всі інструменти для реалізації кастомного функціоналу в магазині на основі ORO.

Symfony пропонує розробникам компонентний підхід, який дозволяє вибирати та інтегрувати лише ті функції, які потрібні для конкретного проекту. Це сприяє простому процесу розробки та оновлення системи. Крім того, Symfony має потужні інструменти для роботи з базами даних, формами, автентифікацією та авторизацією, кешуванням та іншими стандартними задачами веб-розробки.

Фреймворк відомий також своєю високою продуктивністю, що є важливим аспектом для будь-якого інтернет-магазину з високими вимогами до швидкодії. Архітектура Symfony підтримує розробку з використанням найкращих практик та патернів проектування, таких як ADR та Command Bus, що дозволяє створювати проекти великого рівня. На рисунку 2.4 зображено цикл обробки запиту всередині фреймворку.

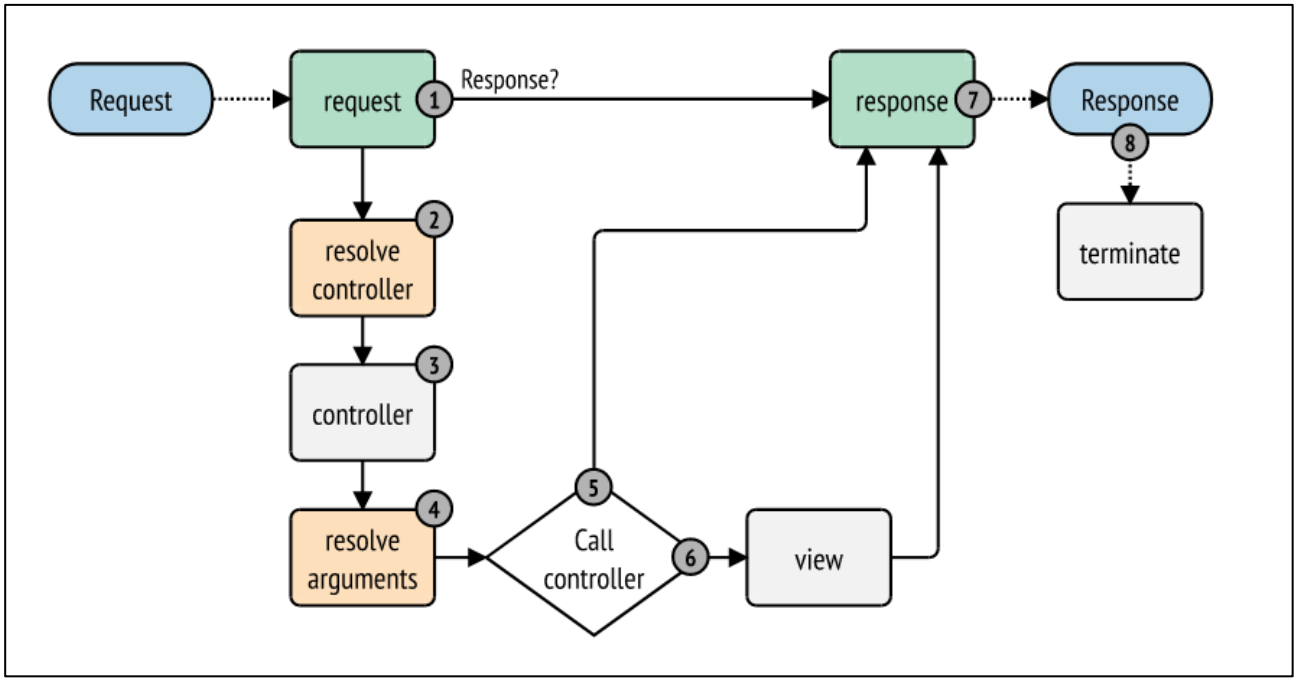


Рисунок 2.4 – Цикл обробки запиту в Symfony

Кожний HTTP запит обробляється через серію визначених кроків свого життєвого циклу. Запит, який надходить до системи, перетворюється в об'єкт Request, який Symfony використовує для подальшої обробки. Фреймворк спочатку визначає контролер для обробки запиту, використовуючи компонент маршрутизації, що мапить URL на відповідний контролер, після чого створює потрібні аргументи для його виклику, і фінально викликає метод контролера.

Контролер, отримавши всі необхідні аргументи, обробляє запит і генерує відповідь. Ця відповідь може бути у вигляді JSON, HTML або іншого формату, зазвичай створюючи її з шаблонів, наприклад, за допомогою системи шаблонів Twig. Після створення відповіді, надається змога ще будь-яким чином кастомізувати відповідь, після чого вона буде надіслана клієнту. Відправивши відповідь, система може виконати будь-які заключні дії, такі як закриття реісурсів, журналювання або відправлення електронних повідомлень.

Наступним технічним рішенням є питання пошукової системи, адже, одним із набільш часто використовуваним функціоналом інтернет-магазину, є саме пошук продуктів, так він забезпечує користувачам можливість швидко та легко знаходити потрібні товари серед великого асортименту. Ефективна пошукова система значно покращує користувацький досвід, підвищуючи шанси на успішну транзакцію.

Для реалізації повнотекстового пошуку в інтернет-магазині було обрано ElasticSearch [22], оскільки дана пошукова система є найбільш перевіреним рішенням, і платформа ORO має вже готову інгеграцію з нею. Даний пошуковий рушій відомий своєю швидкістю, масштабованістю та гнучкістю, що забезпечує широкий набір можливостей для організації та оптимізації пошуку.

Можливості ElasticSearch включають, власне, текстовий пошук, який дозволяє знаходити товари, використовуючи складні запити з врахуванням синонімів та морфології слів. Крім того, підтримується фасетний пошук, що дає можливість користувачам фільтрувати результати по різним характеристикам товарів, як-от категорія, бренд, ціна та інші параметри. Рангування результатів пошуку дозволяє піднімати найбільш релевантні продукти на верхні позиції у відповідях на запити, що полегшує користувачам вибір товарів.

На рисунку 2.5 відображено деталізовану діаграму послідовності для варіанту використання «Пошук товару». Клієнт розпочинає з введення пошукового запиту, текст якого передається на бекенд, де на сервері ініціюється endpoint пошуку. Далі, ElasticSearch виконує повнотекстовий пошук, і повернуті результати перевіряються на доступність в базі даних. Ті результати, які

відповідають критеріям доступності, форматуються та передаються користувачеві назад через захищене https з'єднання. В результаті клієнт отримує перелік знайдених товарів для перегляду.

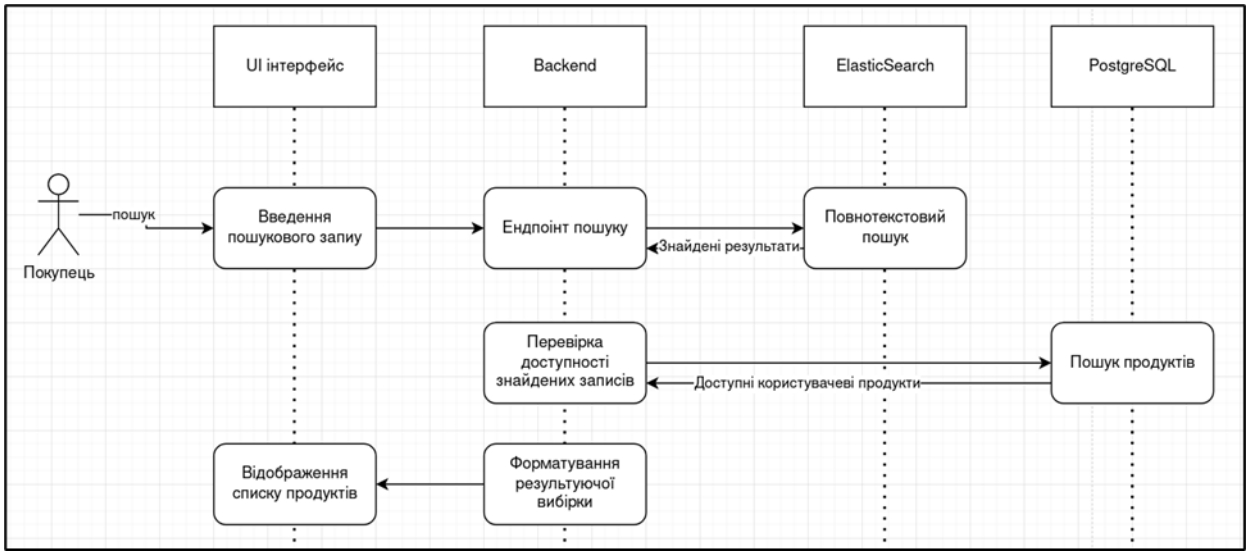


Рисунок 2.5 – Діаграма послідовності пошуку товарів

Наступним важливим аспектом в архітектурі веб-додатків є кешування, оскільки воно надає можливості щодо оптимізації продуктивності системи, дозволяючи зберігати часто запитувані дані в тимчасовій пам'яті, що зменшує навантаження на сервери. Для зменшення кількості запитів до бекенд-сервісів, кешування може застосовуватись на кількох рівнях.

Зі сторони клієнта зазвичай використовується кеш браузера, дозволяючи швидко завантажити статичний контент, якщо користувач вже раніше відвідував сторінки магазину. На стороні сервера може застосовуватись ESI-кеш (Edge Side Includes), що дозволяє кешувати окремі частини веб-сторінок, такі як заголовки, навігаційні меню та футери, динамічно вставляючи їх на сторінку за допомогою конфігурації зворотнього проксі-сервера, тим самим підвищуючи ефективність доставки контенту. Втім, ESI не є може використовуватись для SPA-додатків.

Найбільш гнучким в налаштуваннях, а також найбільш часто використовуваним рівнем кешування зі сторони сервера є кеш прикладного рівня. Application cache організовує збереження комплексних даних програми,

таких як результати складних запитів чи конфігурації. Такі дані можна кешувати за допомогою таких серісів як Redis та MemCached, які відзначаються високою швидкістю та можливістю горизонтального масштабування. В рамках розробки B2B інтернет-магазину планується використання Redis, як найбільш популярного та перевіреного часом рішення.

Redis [23], що за скороченням означає Remote Dictionary Server, є високопродуктивною базою даних для зберігання структур даних, яка може також використовуватись як кеш та брокер повідомлень. Це сховище ключ-значення підтримує різні типи даних, такі як рядки, списки, карти, множини, відсортовані множини з можливістю пошуку за діапазоном, бітмапи та ін.

Redis славиться своєю великою швидкістю, оскільки всі дані зберігаються в оперативній пам'яті, в порівнянні з традиційними базами даних, що зберігають дані на диску або SSD. Це дозволяє виконувати читання та запис майже миттєво, що ідеально підходить для кешування операцій, чутливих до затримок, таких як деякі запити до бази даних, проміжні результати складних обчислень тощо.

Однією з ключових особливостей Redis є підтримка різних моделей видалення та закінчення терміну дії для кешованих даних, що дозволяє автоматично очищати старі або невикористані дані, а також швидко оновлювати кеш при зміні даних в базовій базі даних.

Використання наведених вище моделей кешування забезпечить високу швидкодію інтернет-магазину, ефективне використання ресурсів та загалом поліпшить користувацький досвід, зменшуючи час очікування завантаження сторінок та забезпечуючи оперативну обробку запитів.

Наступним, найбільш критично важливим рішенням у вибрі стеку технологій для реалізації бізнес-логіки інтернет-магазину є вибір бази даних, так як це є центральним компонентом, з яким буде взаємодіяти додаток.

Незважаючи на те, що сучасні бібліотеки об’єктно-реляційного відображення (ORM) пропонують певний рівень абстракції, що може спрощувати взаємодію з базою даних, все ж питання вибору конкретної бази даних залишається дуже важливим. Реалізація певних запитів та функцій може

бути тісно пов'язана зі специфікою бази, що робить її подальшу зміну на іншу надзвичайно складною або навіть неможливою за виключенням суттєвого переписування коду. Крім того, процес міграції даних між різними базами даних може бути пов'язаний з труднощами та потребує додаткових ресурсів.

MySQL [24] є однією з найпопулярніших реляційних баз даних, яка відома своєю швидкодією і легкістю управління. Це відмінний вибір для веб-додатків з невеликою або середньою навантаженістю. MySQL підтримує різноманітність платформ та часто використовується у поєднанні з PHP, що робить її популярним вибором для розробки веб-сайтів. Проте, вона може виявитися менш гнучкою у випадку складних транзакційних систем із багатим функціоналом, який може потребувати підтримки геопросторових даних або комплексних запитів.

PostgreSQL [25], з іншого боку є вибором для проектів, де потрібна висока надійність, масштабованысть та складний функціонал, адже підтримує більш розширені функції, які не пропонуються MySQL, такі як вищий рівень ізоляції транзакцій, підтримка складних запитів, а також геопросторових даних.

Однією з переваг PostgreSQL перед MySQL є вдосконалена підтримка для роботи з JSON-даними. БД пропонує вбудовані функції для зберігання об'єктів та ефективного виконання запитів, які дозволяють не лише зберігати JSON-об’єкт, але й безпосередньо взаємодіяти з його вмістом на рівні бази, надаючи швидкий доступ до вкладених структур даних, що можуть зустрічатись у веб-сервісах. Хоча MySQL також підтримує JSON-формат, втім функції для роботи з такими даними є більш обмеженими і менш продуктивними у порівнянні з PostgreSQL.

Крім того, база PostgreSQL відома своєю розширеною підтримкою геоданих через модуль PostGIS, що є одним з найпотужніших інструментів у галузі просторових баз даних. Розширюючи основні можливості БД, PostGIS додає підтримку геопросторових об'єктів, дозволяючи здійснювати складні запити, що включають географічні розміщення та відстані.

Даний модуль робить PostgreSQL особливо привабливим вибором для B2B магазинів, які можуть мати потребу в розширенні функціоналу, пов'язаного з

геопросторовим аналізом. Зокрема, в майбутньому може виникнути потреба в реалізації функціоналу трекінгу посилок, особливо для міжнародних замовлень. В такому випадку можливості PostGIS безсумнівно стануть в нагоді для точного визначення маршрутів, оптимізації логістики та надання клієнтам детальної інформації про розташування їх вантажів та відстані до пункту призначення.

Роблячи вибір системи керування базою даних під потреби розроблюваного інтернет-магазину в контексті B2B ринку з високими вимогами до надійності, продуктивності та масштабованості, було прийнято рішення на користь PostgreSQL. Ця база забезпечить міцну основу для керування великими обсягами даних, побудови складних запитів та можливість гнучкої інтеграції розширеного функціоналу, що необхідно для вимогливого B2B середовища.

Наступним важливим рішенням у проектуванні архітектури програмної системи є вибір брокера повідомлень, адже фонова обробка задач є обов'язковою умовою, коли йдеться про масштабованість додатків. Брокер повідомлень забезпечує асинхронну комунікацію між компонентами системи, що дозволяє розподіляти навантаження та забезпечити високу доступність сервісів. Найпопулярніші рішення в цій області включають Apache Kafka, RabbitMQ та Amazon SQS.

Apache Kafka [26] розроблений для обробки потокових даних великого обсягу і часто використовується для побудови надійних розподілених систем. Kafka забезпечує високу пропускну здатність, надійне зберігання повідомлень та здатність до горизонтального масштабування. Це робить Kafka ідеальним рішенням для сценаріїв, де потрібно ефективно обробляти великі обсяги даних в реальному часі. Основним недоліком Kafka може бути складність в налаштуванні та обслуговуванні.

RabbitMQ [27] є одним з найпопулярніших відкритих брокерів повідомлень, який підтримує протокол комунікації AMQP 0-9-1. Брокер славиться своєю легкістю у використанні, гнучкістю у налаштуванні маршрутизації повідомлень та високою надійністю. Він ідеально підходить для задач, де потрібні гарантії доставки повідомлень та складна логіка

маршрутизації. Втім, у випадках з дуже великою кількістю повідомлень, RabbitMQ може потребувати більше ресурсів для управління чергами.

Amazon SQS [28] — це повністю керований сервіс черг повідомлень від компанії AWS, який дозволяє легко відправляти, зберігати та отримувати повідомлення між компонентами системи без їхньої втрати. SQS забезпечує просте масштабування без необхідності управління інфраструктурою, але може виявитися менш гнучким, а також трохи дорожчим у порівнянні з відкритими рішеннями.

Для розроблюваного B2B інтернет-магазину вибір впав на RabbitMQ, враховуючи його гнучкість у налаштуванні та маршрутизації повідомлень, що є дуже важливим для B2B застосунків, де можуть бути складні вимоги до обробки повідомлень і інтеграції з сторонніми ERP системами. Даний брокер надає широкий спектр можливостей для взаємодії компонентів системи, зокрема за допомогою direct, topic та fanout exchanges. Direct exchanges дозволяють здійснювати точну адресацію повідомлень за ключами маршрутизації, topic exchanges підтримують більш гнучкі шаблони маршрутизації з використанням шаблонів тем, а fanout exchanges розсилають повідомлення всім воркерам черги без урахування ключа маршрутизації, що ідеально підходить для задач розповсюдження подій.

Для забезпечення найкращої продуктивності системи, планується делегувати ряд важливих операцій на фонову обробку за допомогою RabbitMQ. Основними завданнями фонової обробки стане оновлення пошукового індексу ElasticSearch для ефективного пошуку продуктів, а також обробка імпорту даних продуктів, спрощуючи оновлення асортименту та інтеграцію з даними постачальників. Крім того, відправка електронних листів клієнтам, як от лист для відновлення паролю, чи інформаційні розсилки, також планується здійснювати в фоновому режимі.

Таким чином, використання брокера повідомлень дозволить суттєво оптимізувати процес обробки запитів з клієнтської частини, виділивши певні частини процесу в фонові задачі без впливу на основну роботу системи.

Таким чином, в даному розділі було здійснено аналіз та вибір ключових технологій для реалізації інтернет-магазину, заснованого на платформі OROCommerce. На рисунку 2.6 представлена діаграма розгортання, яка високорівнево зображує взаємодію між архітектурними компонентами магазину. Веб-сервер Nginx обробляє користувацькі запити, направляючи їх до PHP-бекенду. Для кешування дани використовується Redis. База даних PostgreSQL забезпечує зберігання даних, ElasticSearch обробляє пошукові запити, а RabbitMQ, в якості системи повідомлень забезпечує фонову обробку завдань.

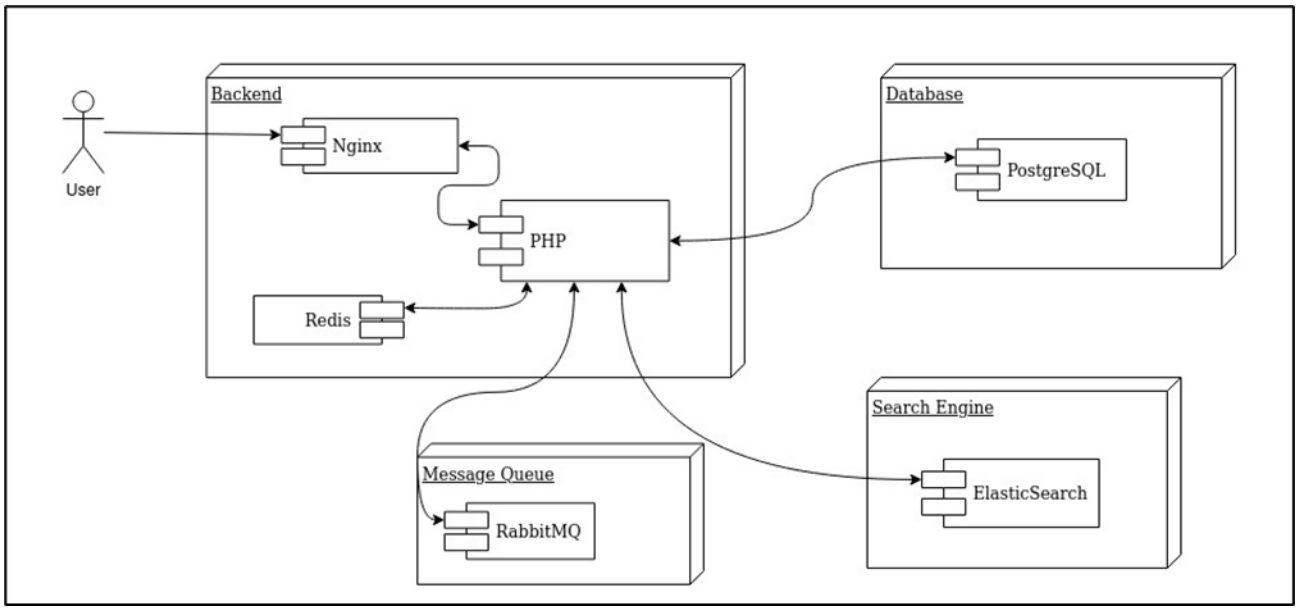


Рисунок 2.6 – Діаграма розгортання магазину

В якості основної мови програмування було обрано PHP з огляду на широке поширення у веб-розробці та сумісність з фреймворком Symfony, який є основою для пакетів OROCommerce. Symfony було обрано за його гнучкість, безпеку та потужні інструменти для розробки, які забезпечують ефективну реалізацію кастомного функціоналу магазину.

Для повнотекстового пошуку було вибрано ElasticSearch через його високу продуктивність та готову інтеграцію з ORO. Кешування даних в системі буде організовано з використанням кешу браузера та кешу на application-рівні, для якого обрано Redis через його швидкість та гнучкість.

Вибір PostgreSQL як основної бази даних обумовлений її масштабованістю, надійністю та розширеними можливостями, такими як робота з JSON та геоданими через PostGIS.

В якості брокера повідомлень для фонової обробки задач в системі було обрано RabbitMQ, що відповідає високим вимогам B2B сектору завдяки своїй гнучкості та надійності. Фонова обробка включатиме задачі такі, як оновлення пошукового індексу, імпорт даних продуктів та відправка електронних листів.

Обраний стек технологій надає міцну основу для побудови масштабованого, високопродуктивного та гнучкого інтернет-магазину, який зможе адаптуватися до майбутніх вимог ринку та розширення бізнесу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Втілення архітектури в програмних модулях

У процесі розробки проекту одним з ключових архітектурних рішень було структурування коду у вигляді модульного моноліту. Такий підхід дозволяє кожному модулю системи бути схожим на окремий мікросервіс, але водночас залишатися частиною єдиної платформи. Це значно спрощує процеси інтеграції та розгортання.

Розробка модулів системи полягатиме в реалізації функціоналу інтернет-магазину у вигляді окремих компонентів, де кожен модуль буде відкривати певні точки комунікації, специфічні до його бізнес-вимог. Така модель є класичною для багатьох великих веб-застосунків. В попередніх розділах було розглянуто ключові моменти даного архітектурного рішення.

Монолітний підхід з модульною структурою дозволить забезпечити кожному модулю високу ступінь автономії. До прикладу, модуль AppPricingBundle, що відповідає за ціноутворення і промо-пропозиції, повністю незалежний від модуля AppWebsiteSearchBundle, який реалізує функціонал пошуку по каталогу продуктів.

Клієнтська сторона інтернет-магазину, представлена у вигляді клієнту веб-браузера, що надає користувачам інтерфейс для взаємодії з системою, відповідатиме за формування HTTP запитів до ендпоінту відповідного модуля на сервері і опрацювання відповідей сервера, що включають оновлену інформацію для відображення на сторінці, результати пошуку товарів, оформлення замовлень, тощо.

Серверна сторона ж оброблятиме всі запити від клієнтської частини, керуючи даними в БД, що включає виконання основної бізнес-логіки застосунку, такої як розрахунок цін і промо-пропозицій, обробка замовлень, і передаватиме відповіді назад до клієнтської сторони. Використання серверної логіки на базі HTTP та RESTful архітектури дозволить ефективно використовувати мережеві протоколи для динамічного обміну даними, забезпечуючи простоту взаємодії.

Проекти на основі фреймворку Symfony можуть реалізовувати модулі у вигляді окремих бандлів, що дозволяє повністю ізолювати функціональність, включаючи рівень налаштування фреймворку. Приклад такого оголошення окремого бандла в Symfony наведено нижче:

```
use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\HttpKernel\Bundle\Bundle;

final class AppWebsiteSearchBundle extends Bundle
{
    public function build(ContainerBuilder $container)
    {
        $container->addCompilerPass(new ModifyElasticSearchIndexAgentPass());
    }
}
```

Всередині кожного модуля системи налаштовується сервіс-контейнер. Це включає налаштування конфігурації для завантаження сервісів з yaml-файлу. Нижче наведено приклад коду, який демонструє, яким чином відбувається така реєстрація сервісів через Dependency Injection Extension модуля.

```
use Symfony\Component\DependencyInjection\Extension\Extension;
use Symfony\Component\DependencyInjection\Loader\YamlFileLoader;

final class AppWebsiteSearchExtension extends Extension
{
  public const ALIAS = 'App_website_search';

  public function load(array $configs, ContainerBuilder $container)
  {
    $configuration = new Configuration();
    $config = $this->processConfiguration($configuration, $configs);
    $loader = new YamlFileLoader($container, new FileLocator(__DIR__ . '/../Resources/config'));

    $loader->load('services.yml');
    $container->prependExtensionConfig($this->getAlias(), array_intersect_key($config,
array_flip(['settings'])));
  }

  public function getAlias()
  {
    return self::ALIAS;
  }
}
```

Нижче наведено приклад оголошення сервісів в файлі services.yaml, який використовується для визначення та налаштування оголошень Symfony-сервісів для DI-контейнера [29] в рамках одного модуля:

```
services:
  _defaults:
    autowire: true
    autoconfigure: true

  App\Bundle\WebsiteSearchBundle\Engine\ElasticSearchSynonymsRequestData:
    arguments:
      - '@oro_config.website'

  app_website_elastic_search.request_builder.where.contains:
    class:
App\Bundle\WebsiteSearchBundle\RequestBuilder\FuzzyContainsWherePartBuilderDecorator
    decorates: oro_website_elastic_search.request_builder.where.contains
    arguments:
      - '@app_website_elastic_search.request_builder.where.contains.inner'
      - '@property_accessor'
```

Зрештою, кожен окремий модуль містить свої специфічні класи, об'явлені сервісами, які включають код, необхідний для реалізації функціональності, дозволяючи бізнес-сценаріям бути розділеними по аспектах і незалежно реалізовувати свої задачі в межах загальної архітектури системи.

Компоненти системи, які відкривають точки комунікації для клієнтів, використовують архітектурний підхід REST, реалізуючи взаємодію за допомогою HTTP-запитів. В рамках Symfony розробка HTTP-ендпоінтів полягає у використанні API-контролерів для управління ресурсами (до прикладу як слоти доставки) за допомогою анотацій, що дозволяють деталізувати поведінку маршрутів та контроль доступу.

До прикладу, одним із важливих аспектів інтернет-магазину є правильна реалізація бізнес-логіки для управління слотами доставки. Нижче наведено код реалізації контролера DeliverySlotController, що надає API-ендпоінт для взаємодії з сутностями слотів доставки, включаючи створення, перегляд, редагування та видалення даних сутностей з адмін панелі.

```
final class DeliverySlotController extends AbstractController
{
/**
 * @Route("/", name="app_delivery_slot_index")
 * @Template()
 * @AclAncestor("app_delivery_slot_view")
 */
public function indexAction(): array
{
    return ['entity_class' => DeliverySlot::class];
}
```

```
    /**
     * @Route("/view/{id}", name="app_delivery_slot_view")
     * @Template()
     * @Acl(id="app_delivery_slot_view", type="entity", class="AppDeliverySlotsBundle:DeliverySlot",
permission="VIEW")
     */
    public function viewAction(DeliverySlot $deliverySlot): array
    {
        return ['entity' => $deliverySlot];
    }

    /**
     * @Route("/create", name="app_delivery_slot_create", options={"expose"=true})
     * @Template("AppDeliverySlotsBundle:DeliverySlot:update.html.twig")
     * @Acl(id="app_delivery_slot_create", type="entity", class="AppDeliverySlotsBundle:DeliverySlot",
permission="CREATE")
     */
    public function createAction()
    {
        return $this->update(new DeliverySlot());
    }

    /**
     * @Route("/update/{id}", name="app_delivery_slot_update")
     * @Template()
     * @Acl(id="app_delivery_slot_update", type="entity", class="AppDeliverySlotsBundle:DeliverySlot",
permission="EDIT")
     */
    public function updateAction(DeliverySlot $deliverySlot)
    {
        return $this->update($deliverySlot);
    }

    protected function update(DeliverySlot $deliverySlot)
    {
        $form = $this->createForm(DeliverySlotType::class, $deliverySlot);
        $savedMessage = $this->get('translator')-
>trans('app.deliveryslot.controller.deliveryslot.saved.message');

        return $this->get('oro_form.update_handler')->update($deliverySlot, $form, $savedMessage);
    }

    /**
     * @Route("/widget/info/{id}", name="app_delivery_slot_widget_info")
     * @Template("AppDeliverySlotsBundle:DeliverySlot/widget.html.twig")
     * @AclAncestor("app_delivery_slot_view")
     */
    public function infoAction(DeliverySlot $entity): array
    {
        return ['entity' => $entity];
    }
```

Контролер реалізує ендпоінт серверної частини для управління слотами доставки, і включає логіку відповідей на запити, забезпечення контролю доступу, а також форматування даних для користувачів, використовуючи утиліти для авторизації та шаблонізації, що надаються фреймворком.

Сутність DeliverySlot, з якою працює контролер, представляє слот доставки, який користувач може обрати для планування своїх замовлень на конкретні часові періоди в певні дні, використовує анотації Doctrine ORM для взаємодії з базою даних, що дозволяє ефективно зберігати та читати дані.

```
/**
* @ORM\Entity(repositoryClass="App\Bundle\DeliverySlotsBundle
\Repository\DeliverySlotRepository")
 * @ORM\Table(
 *  name="app_delivery_slot",
 *  indexes={
 * @ORM\Index(name="app_delivery_slot_zip_unique_idx",columns={"organization_id", "zip", "day",
"time_begin", "time_end"}),
 * @ORM\Index( name="app_delivery_slot_zone_unique_idx", columns={"organization_id", "zone",
"day", "time_begin", "time_end"}),
 * @ORM\Index(name="idx_delivery_slots_import_session", columns={"import_session"})
 * },
 *  uniqueConstraints={
 * @ORM\UniqueConstraint(name="app_delivery_slot_zip_unique_idx", columns={"organization_id",
"zip", "day", "time_begin", "time_end"}),
 * @ORM\UniqueConstraint(name="app_delivery_slot_zone_unique_idx",
columns={"organization_id", "zone", "day", "time_begin", "time_end"})
 * }
 * )
 */
class DeliverySlot
{
use DatesAwareTrait;

/**
 * @ORM\Id
 * @ORM\Column(type="integer", name="id")
 * @ORM\GeneratedValue(strategy="AUTO")
 */
private ?int $id = null;

/** @ORM\Column(type="string", name="zip") */
private ?string $zip = null;

/**@ORM\Column(type="string", name="zone", options={"check":"CONSTRAINT
zip_or_zone_not_null CHECK (zip is not null or zone is not null)"}) */
private ?string $zone = null;

/** @ORM\Column(type="integer") */
private ?int $day = null;

/** @ORM\Embedded(class="App\Bundle\DeliverySlotsBundle\Embedded\DeliveryTimeRange",
columnPrefix="time_") */
private DeliveryTimeRange $deliveryTimeRange;

/** @ORM\Column(type="integer", name="cutoff_day", nullable=true) */
private ?int $cutoffDay = null;

/** @ORM\Column(type="time_immutable", name="cutoff_time", nullable=true) */
private ?DateTimeImmutable $cutoffTime = null;
//..
}
```

В даній сутності кожен слот має унікальний ідентифікатор, що автоматично генерується базою даних, поштовий індекс, а також зону доставки, для якої він діє. Крім того, вказується день тижня і часовий діапазон, на коли можна оформити доставку, а також крайній строк слота (день тижня і час), до коли можна запланувати доставку, після чого слот не можна буде використати.

Система використовує унікальні індекси для забезпечення консистентності, таким чином що для кожної комбінації поштового індексу, дня тижня та часу доставки може існувати лише один слот. Це допомагає запобігти конфліктам даних і гарантує їх унікальність.

Таким чином, кожен модуль інтернет-магазину розроблений так, що має власний набір сервісів контейнера залежностей, власні класи сервісів, контролерів і сутностей, які реалізують кожний конкретний сценарій використання даного модуля. Така архітектура мінімізує залежності між компонентами, забезпечуючи водночас їх інтегрованість із загальною системою через під’єднання до спільного ядра фреймворку.

Всі компоненти розміщуються в єдиному git-репозиторії [30], що дозволяє централізовано управляти кодом, тим самим спрощуючи процес розробки магазину, оскільки всі модулі доступні в одному місці. Такий підхід сприяє можливості майбутнього переходу на більш децентралізовану архітектуру, таку як мікросервіси, забезпечуючи при цьому стабільність вже написаної системи.

3.2 Розробка функціоналу промо-цін

Для будь-якого продукту для будь-якого користувача або групи користувачів передбачається можливість встановлення персональної промо-ціни. Такі ціни мають вищий пріоритет над стандартними цінами у системі, і відображаються з спеціальним маркуванням “Промо”. До прикладу, якщо для певного продукту зазначена ціна і в звичайному списку цін, і в промо-списку, то при продажу буде застосована ціна з промо-списку.

Візуальний вигляд розроблюваних промо-цін зображено на рисунку 3.1, де активною є ціна, вказана для користувача, а стара ціна відображається в перекресленому вигляді. Це відображення допомагає користувачеві ідентифікувати вигоду від запропонованої промо-ціни, що підвищує привабливість такої пропозиції та спонукає до покупки.

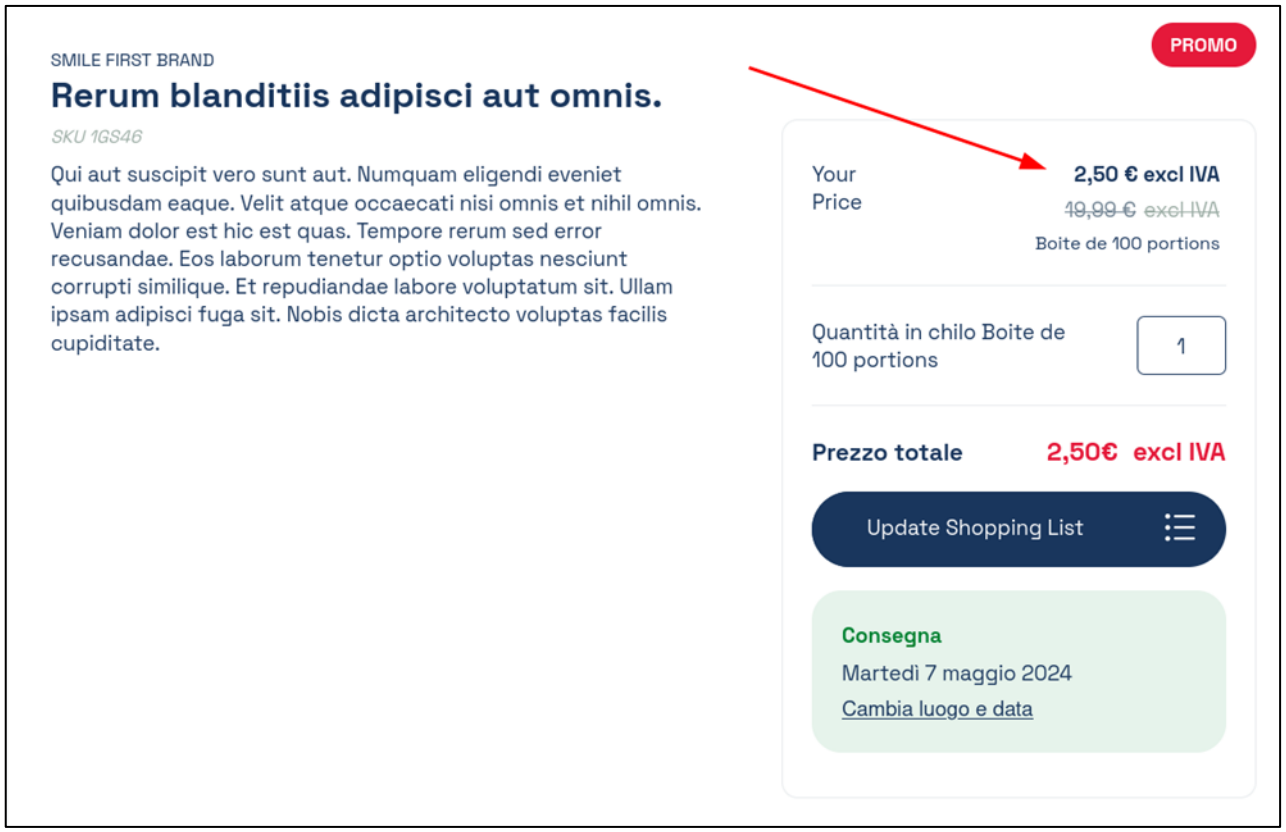


Рисунок 3.1 – Відображення промо-ціни

В якості основи для технічної реалізації функціоналу комбінованих промо-цін використовується система управління цінами, яка надається платформою ORO. Основна концепція цінової стратегії полягає у використанні ієрархічної системи визначення цін, дозволяючи адміністраторам керувати цінами на різних рівнях, забезпечуючи гнучкість в налаштуванні залежно від потреб бізнесу та індивідуальних умов.

На найвищому (глобальному) рівні встановлюється список цін, який використовується за замовчуванням, тобто коли більш специфічні рівні ціноутворення не задають власних списків.

Наступним рівнем є група користувачів, яка може включати певний набір користувачів, і для неї можна задати спеціальний список цін. Ціни на рівні групи мають пріоритет над глобальними цінами для всіх продуктів, що входять до цього списку. Якщо ж певний продукт відсутній у списку цін групи, то застосовується ціна з глобального списку.

На найнижчому рівні системи цін знаходиться індивідуальний список цін для кожного користувача, що дає можливість встановити персональні ціни на певні продукти. Якщо продукт не включений до індивідуального списку, то система отримує ціну з списку рівня групи, до якої належить користувач, і лише після цього — у глобальному списку.

Ієрархічна структура взаємозв'язків між різними рівнями цін у системі інтернет-магазину зображена на рисунку 3.2. На основі наданої діаграми, можна більш детально зрозуміти як ціни на продукти встановлюються на різних рівнях, а також як списки цін пов’язані між собою залежно від приналежності користувача до певної групи чи встановлених індивідуальних умов.

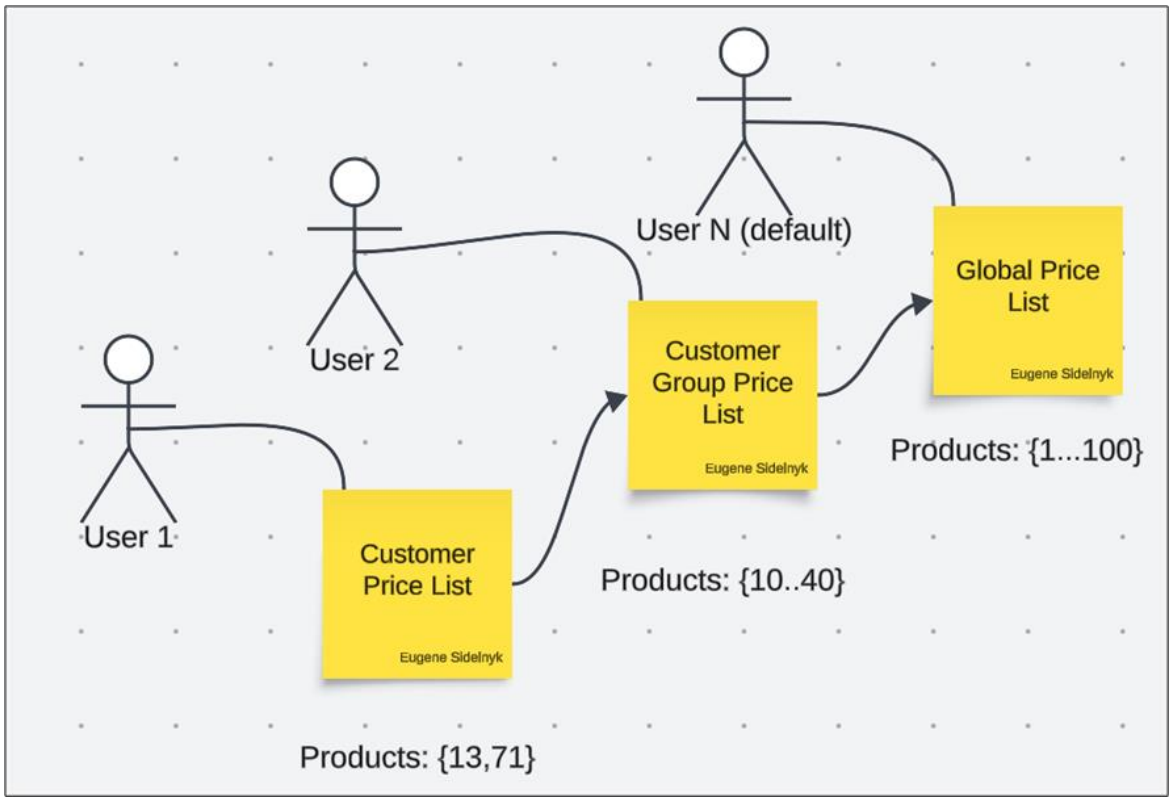


Рисунок 3.2 – Ієрархія визначення цін

Як видно з схеми вище, списки ціни в системі можуть визначатись на основі кількох рівнів: глобального, групи користувачів та індивідуального користувача, залежно від наявності цін продуктів у відповідних списках цін.

Глобальний список служить як базовий рівень, що містить стандартні ціни для широкого асортименту продуктів. В даному прикладі сюди входять продукти з номерами від 1 до 100. Це основний довідник цін, що використовується, коли на інших рівнях специфічніші списки цін не задані.

На другому рівні, ціни можуть бути уточнені під конкретну групи, до якої належить користувач. До прикладу, якщо користувач 2 належить до групи з списком цін, який охоплює продукти з 10 по 40, то дані продукти для цього користувача будуть мати задану ціну відповідно до значення саме з цього списку, в той час як продукти з номерами 1-9 та 41-100 матимуть ціну з глобального рівня.

Зрештою, індивідуальний рівень цін дає можливість налаштування цін для окремих користувачів на конкретні продукти. На прикладі вище, користувач 1 має персональні ціни для продуктів №13 та №71, які відрізняються від цін, доступних іншим користувачам або групам. Якщо для певного продукту не існує індивідуального визначення, то ціна визначається зі списків вищого рівня.

Таким чином, кожень рівень ієрархії ціноутворення може містити набір від нуля до декількох цінових списків. З технічної точки зору, внутрішньо механізм ціноутворення працює таким чином, що для кожного можливого рівня формується один або декілька комбінованих списків цін. Комбінований список – це такий список цін, що містить об’єднаний набір цін з даного рівня, а також зі списків по ієрархії вище. Простіше кажучи, це зібраний список цін для конкретного користувача. Якщо для конкретного користувача задано список цін (можливо лише для декількох продуктів), то спеціально для нього буде створено комбінований список, що включатиме ціни для всіх доступних продуктів в системі. На рисунку 3.3 представлено діаграму activity, що відображає логіку вибору комбінованого списку цін для користувача інтернет-магазину. Діаграма розгалужується на кілька шляхів залежно гранулярності наявних списків цін.

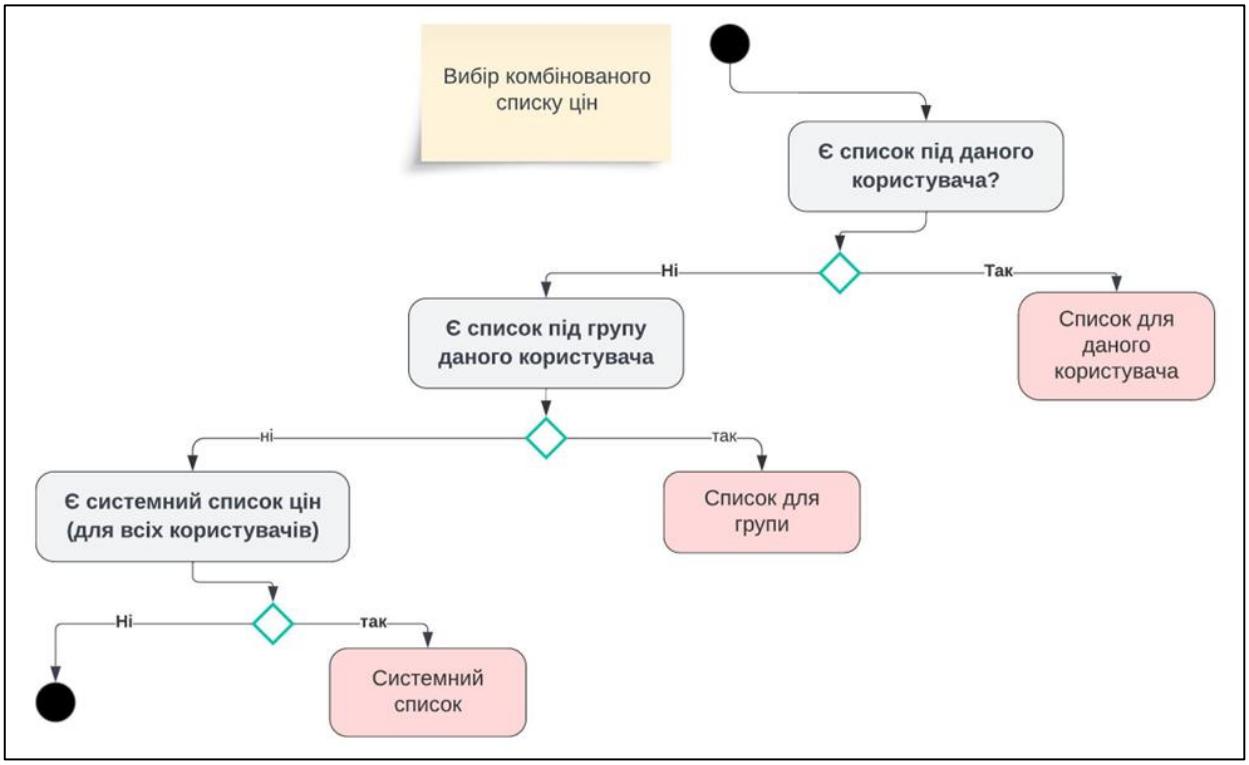


Рисунок 3.3 – Діаграма activity вибору цінового списку

Як можна зрозуміти з блок-схеми вище, концепція роботи з комбінованими списками стає дуже простою, адже маючи зібраний для даного користувача список цін, можна дуже просто отримати ціну на будь-який продукт.

Базуючись на описаній вище моделі ціноутворення, схожим реалізовується й функціонал промо-цін. В загальному, концепція промо-цін полягає в тому, що надається можливість задавати спеціальні цінові пропозиції, які мають вищий пріоритет над звичайними цінами. До прикладу, якщо продукт №71 з прикладу нижче має зазначену ціну одночасно і в звичайному списку цін, і в промо-списку, то ціною оплати буде промо-ціна.

Крім того, промо-списки цін мають здатність перекривати звичайні списки цін на нижчих рівнях ієрархії. До прикладу, як зображено на рисунку 3.4, якщо глобальний рівень задає промо-список для продуктів з номерами від 30 до 40, то ця промоційна ціна переважатиме над звичайними цінами навіть на ті продукти, ціна яких задана на рівні групи, тобто список цін для продкутів з номерами від 10 до 40 в даному прикладі. В результаті продукти з номерами від 30 до 40 будуть продаватися за промо-цінами з глобального рівня.

Ієрархічна структура ціноутворення, з врахуванням наявності промо-списків цін, зображена на рисунку 3.4. Тут чітко видно співвідношення між звичайними ціновими пропозиціями та промо-пропозиціями.

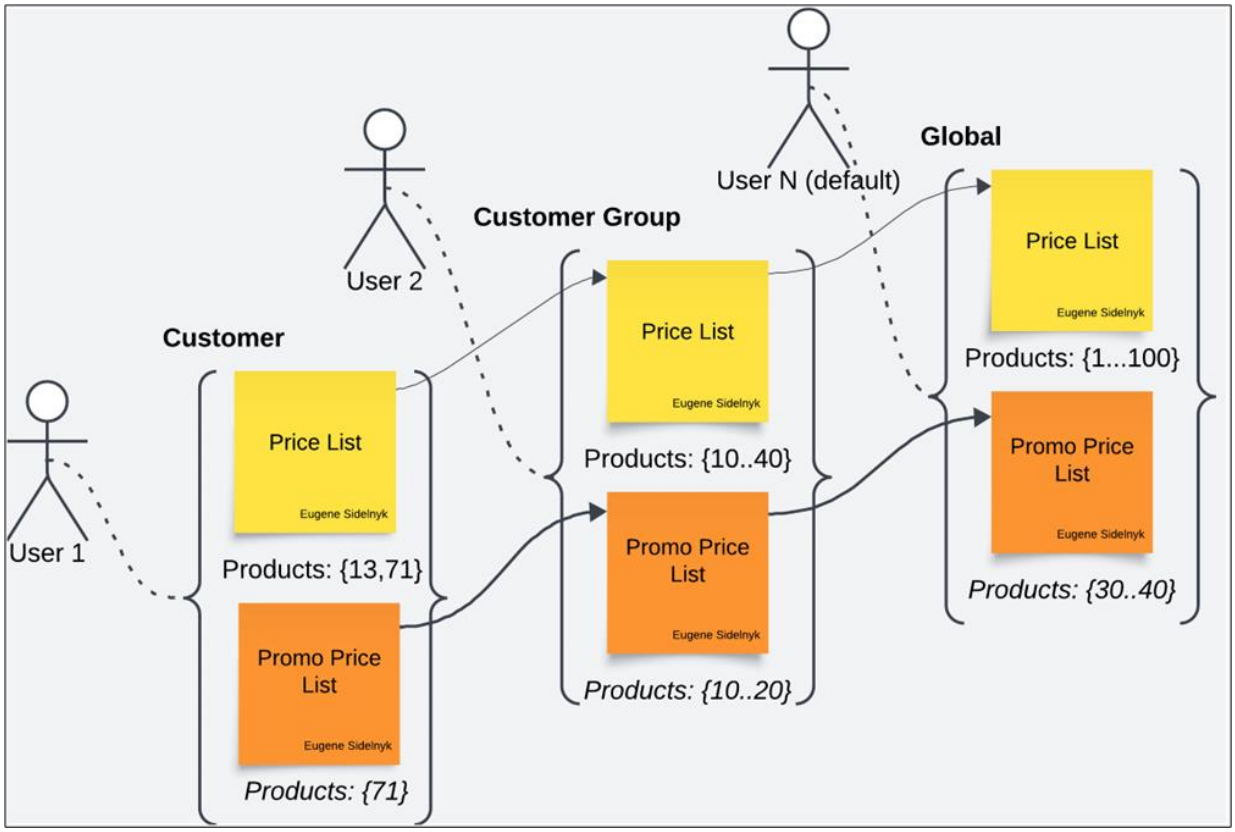


Рисунок 3.4 – Ієрархія промо-цін

Розглянемо випадок, коли системою користується звичайний середньостатистичний користувач, який не має власного індивідуального списку цін і його група також не асоційована з окремим ціновим списком. В такому випадку промо-ціни буде застосовано лише для продуктів з номерами від 30 до 40, оскільки дані продукти входять до промо-списку на глобальному рівні. Для решти ж продуктів, з номерами від 1 до 29 та від 41 до 100, будуть застосовані стандартні ціни, визначені у звичайному списку цін.

Наступний можливий випадок – коли користувач належить до групи, що має спеціальні цінові пропозиції. Якщо для даної групи задано промо-пропозицію на продукти з номерами від 10 до 20, то ці продукти будуть продаватись за промо-цінами, адже такі мають перевагу над будь-якими іншими

цінами. Продукти з номерами від 1 до 9, які не покриваються спеціальними пропозиціями на рівні групи, будуть мати ціни відповідно до глобального цінового списку. Це стандартні ціни, які використовуються за замовчуванням у відсутність інших спеціалізованих цінових пропозицій. Продукти з номерами від 21 до 29 відносяться до цінових пропозицій, заданих для групи, але без промо-маркування, тому для них застосовується звичайна ціна зі списку цін групи. Продукти, позначені номерами від 30 до 40, для яких встановлені промо-пропозиції на глобальному рівні, продаватимуться за промо-ціною, не зважаючи на те, що на рівні групи є вказано їх звичайні ціни. Наостанок, продукти з номерами від 41 до 100 використовуватимуть звичайні ціни з глобального цінового списку, оскільки для них не визначено жодних спеціальних промо цінових пропозицій на рівні групи.

Описаний вище механізм ієрархічного визначення цін дозволить системі дуже гнучко керувати ціноутворенням, забезпечуючи з одного боку єдність та послідовність у вартості товарів для широкого набору нових користувачів, а з іншого – можливість надавати індивідуальні умови та домовленості з клієнтами, що є специфікою B2B сегменту інтернет-магазинів.

Технічна реалізація промо-цін у системі базуватиметься на вже існуючих сервісах та подіях платформи Oro, що дозволить швидко інтегрувати новий функціонал без необхідності його розробки з нуля. Фундамент реалізації полягає у використанні нового спеціального налаштування в адмін-панелі, яке позначатиме списки цін (а також і ціни) маркуванням промо.

Коли адміністратор через адмін-панель створює новий список цін, він має змогу визначити його як промо-список. Всередині такого списку можуть міститись лише промо-ціни. Це дозволяє чітко сегментувати промоційні пропозиції від стандартних, забезпечуючи змогу на цій основі будувати логіку в коді.

Після створення та призначення промо-списків цін окремим користувачам, групам або на глобальному рівні, система автоматично використовує вбудований механізм для комбінування цін, що вже надається платформою ORO. Відповідно

розробка функціоналу промо-цін передбачає розширення функціоналу даного вбудованого механізму, щоб гарантувати, що промо-ціни мають пріоритет над звичайними.

Реалізація промо-ціноутворення передбачає декорування сервісів та використання класів pricing-подій для додавання нової логіки обробки. Декорування класів та підписка на події дозволяють розширити функціональність без порушення існуючої архітектури системи. Це дає змогу гнучко та ефективно інтегрувати новий функціонал, використовуючи вже відтестовану систему цін, яка надається платформою.

Для інтеграції промо-цін у систему із забезпеченням їх пріоритету над звичайними цінами під час злиття цінових списків розроблено декоратор PriceListCollectionProviderDecorator, який розширює функціонал існуючого сервісу PriceListCollectionProvider, що використовується платформою ORO для управління колекціями цінових списків.

```
final class PriceListCollectionProviderDecorator extends PriceListCollectionProvider
{
    /** @required */
    public PriceListCollectionProvider $priceListCollectionProvider;

    public function getPriceListsByConfig(): array
    {
        return $this->sortPriceListsByPromoFirst(
$this->priceListCollectionProvider->getPriceListsByConfig());
    }

    public function getPriceListsByWebsite(Website $website): array
    {
        return $this->sortPriceListsByPromoFirst(
$this->priceListCollectionProvider->getPriceListsByWebsite($website));
    }

    public function getPriceListsByCustomerGroup(CustomerGroup $customerGroup, Website
$website): array
    {
        return $this->sortPriceListsByPromoFirst(
$this->priceListCollectionProvider->getPriceListsByCustomerGroup($customerGroup, $website));
    }

    public function getPriceListsByCustomer(Customer $customer, Website $website): array
    {
        return $this->sortPriceListsByPromoFirst(
$this->priceListCollectionProvider->getPriceListsByCustomer($customer, $website));
    }

    private function sortPriceListsByPromoFirst(array $priceListMembers): array
    {
```

```
        $promoMembers = array_filter(
            $priceListMembers,
            static fn(PriceListSequenceMember $member) => $member->getPriceList()
->priceListType()->isPromo()
        );
        $notPromoMembers = array_diff_key($priceListMembers, $promoMembers);

        return array_merge($promoMembers, $notPromoMembers);
    }

    public function containMergeDisallowed(array $collection): bool
    {
        return $this->priceListCollectionProvider->containMergeDisallowed($collection);
    }

    public function containScheduled(array $collection): bool
    {
        return $this->priceListCollectionProvider->containScheduled($collection);
    }
}
```

Основне завдання даного декоратора — забезпечити, що промо-ціни завжди обробляються першими під час злиття цінових списків. Для цього реалізовано метод sortPriceListsByPromoFirst, що змінює порядок цінових списків, розташовуючи промо-списки на початку. Метод викликається в рамках кожного запиту до списків цін, включно з конфігурацією за замовчуванням, за вебсайтом, за групою користувачів, та за окремим користувачем.

Використання такого підходу з декораторами дозволяє гнучко і ефективно взаємодіяти з різними аспектами цінової стратегії в системі, забезпечуючи при цьому, що промоційні пропозиції завжди враховуються насамперед. Тому схожим чином зроблено і декілька інших декораторів, додаючи спеціальну логіку обробки для промо-списків.

Наступний важливий момент в реалізації функціоналу промо-цін полягає у відображенні оригінальної ціни, яка не використовується для покупки, а лише показується користувачам в перекресленому вигляді. Оскільки платформа ORO обробляє всі наявні ціни, враховуючи кастомізований пріоритет промо-списків, то наразі користувачі бачать лише фінальну ціну, яка може бути промо-ціною.

Для того щоб користувачі могли бачити також оригінальну ціну продуктів, потрібно реалізовувати додаткову логіку створення списків оригінальних цін. Це може бути зроблено використовуючи такий самий підхід з використанням

декораторів та слухачів подій, аналогічно до того, як це вже зроблено для обробки промо-цін. Основна ідея полягає в підписці на події створення комбінованих списків основних цін для запуску додаткової логіки, що побудує ще один так званий "no-promo" список. В результаті будуть створені два списки: один, який включає промо-ціни об’єднані з звичайними цінам, і використовується для покупок, та інший, який включає лише звичайні ціни за винятком промо-цін, і який використовується виключно для візуального відображення оригінальних цін користувачам.

Нижче приведено клас, що реалізовує логіку автоматичного запуску побудови no-promo списку цін, призначеного виключно для візуальних цілей.

```
final class MessageProducerDecorator implements MessageProducerInterface
{
    private MessageProducerInterface $wrapped;

    public function __construct(MessageProducerInterface $wrapped)
    {
        $this->wrapped = $wrapped;
    }

    public function send($topic, $message): void
    {
        if ($topic === RebuildCombinedPriceListsTopic::getName()) {
            // additionally we need to send this topic in order to rebuild no-promo price lists
            // every time the ordinary price lists are rebuilt
            $this->wrapped
                ->send(RebuildCombinedPriceListsWithoutPromoTopic::getName(), $message);
        }

        $this->wrapped->send($topic, $message);
    }
}
```

Коли до даного класу приходить фонова команда на відправку повідомлення про пребудову цінових списків RebuildCombinedPriceListsTopic, даний декоратор автоматично ініціює відправку в чергу додаткового повідомлення RebuildCombinedPriceListsWithoutPromoTopic. Це забезпечує, що будь-яке оновлення цінових списків одночасно веде до оновлення списків цін без промо-цін, забезпечуючи актуальність даних для користувачів.

Схожим чином, використовуючи шаблони проектування decorator та event subscriber, було розроблено й решту функціоналу промо-цін у системі. Цих два

шаблони дають максимальну гнучкість в коді, за рахунок чого і вийшло доволі легко інтегрувати новий функціонал ціноутворення без зміни основної архітектури платформи. Завдяки застосуванню цих шаблонів, система здатна ефективно адаптуватися та розширюватися, забезпечуючи розробникам можливості для інновацій та швидкої реакції на мінливі умови ринку.

3.3 Розробка інтеграції з платіжною системою

Наступним важливим аспектом для забезпечення зручності фінансових транзакцій в інтернет-магазині є інтеграція платіжної системи. Для даного проекту передбачається використання Unicredit, представлена банківською групою, що надає широкий спектр фінансових послуг, включаючи сучасні рішення для обробки електронних платежів, зокрема в Італії. Система оплати широко використовується в бізнес-секторі, забезпечуючи компаніям можливість здійснювати безпечні та ефективні транзакції з бізнес-рахунків. Вона займає провідні позиції на ринку завдяки своїй надійності, безпеці та низьким комісіям.

Використання платіжного шлюзу в B2B секторі дозволяє компаніям оптимізувати фінансові операції та знизити витрати на обробку платежів. Завдяки інтеграції з бізнес-рахунками, клієнти можуть легко здійснювати платежі, що спрощує управління фінансами і підвищує ефективність операцій.

Що стосується аспекту тестування, то платіжна система надає можливість використання тестових аккаунтів та обширний набір тестових карток. Ці картки дозволяють моделювати різні сценарії платежів, включаючи як успішні, так і неуспішні транзакції, завдяки чому можна ретельно перевірити логіку програми і забезпечити правильну обробку замовлень в кожному окремому випадку.

Процес інтеграції платіжної системи Unicredit на платформі ORO включає кілька основних етапів. Спочатку необхідно створити новий платіжний метод, який відповідатиме за обробку транзакцій. Це можна зробити за допомогою реалізації відповідних інтерфейсів, таких як PaymentMethodInterface, що і показано кодом нижче.

```
final class CreditCardPaymentMethod implements PaymentMethodInterface
{
    public function __construct(private UniCreditGateway $gateway, private RouterInterface $router,
private CreditCardPaymentConfigInterface $config, private LocaleSettings $localeSettings, private
TokenAccessorInterface $tokenAccessor, private Session $session) {}

    public function purchase(PaymentTransaction $paymentTransaction): array
    {
        $paymentTransaction->setAction($this->config->getPaymentAction());
        $this->execute($paymentTransaction->getAction(), $paymentTransaction);

        if (!$paymentTransaction->isActive()) {
            return [];
        }
        return ['purchaseRedirectUrl' => $paymentTransaction->getResponse()['redirectURL']];
    }

    public function complete(PaymentTransaction $paymentTransaction): void
    {
        $request = (new VerifyRequest())->setShopId($this->getShopId($paymentTransaction))
            ->setPaymentId($paymentTransaction->getReference());
        try {
            $response = $this->gateway->paymentVerify($request);
        } catch (\Exception $exception) {
            return $paymentTransaction->setActive(true)->setSuccessful(false); // Set pending
        }
        $paymentTransaction->setSuccessful(!$response->getError())->setActive(false)
            ->setRequest($request->toArray())->setResponse((array)$response->getResponseData());

        if ($paymentTransaction->isSuccessful()
            && $response->getResponseCode() ===
ResponseCodes::TRANSACTION_IN_PROGRESS
        ) {
            $paymentTransaction->setActive(true)->setSuccessful(false);
        }
        if ($paymentTransaction->getAction() === self::AUTHORIZE && $paymentTransaction-
>isSuccessful()) {
            $paymentTransaction->setActive(true);
        }
    }

    public function charge(PaymentTransaction $paymentTransaction): void
    {
        $this->doInitRequest($paymentTransaction, TransactionType::PURCHASE());
    }

    public function capture(PaymentTransaction $paymentTransaction): array
    {
        $sourcePaymentTransaction = $paymentTransaction->getSourcePaymentTransaction();
        if (!$sourcePaymentTransaction) {
            $paymentTransaction->setSuccessful(false)->setActive(false);
            return ['successful' => false];
        }
        $request = (new ConfirmRequest())->setShopId($this->getShopId($paymentTransaction))
            ->setAmount(Amount::fromString($paymentTransaction->getAmount()))
            ->setTranId((int)$sourcePaymentTransaction->getResponse()['tranID']);

        try {
            $response = $this->gateway->paymentConfirm($request);
        } catch (\Exception $exception) {
            $paymentTransaction->setSuccessful(false)->setActive(false);
            return ['successful' => false, 'message' => $exception->getMessage()];
```

```
            }
            $paymentTransaction->setSuccessful(!$response->hasError())->setActive(false)
                ->setReference($response->getTranId())->setRequest($request->toArray())
                ->setResponse((array)$response->getResponseData());

            $sourcePaymentTransaction->setActive(!$paymentTransaction->isSuccessful());

            if ($paymentTransaction->isSuccessful()
                && $response->getResponseCode() ===
ResponseCodes::TRANSACTION_IN_PROGRESS
            ) {
                $paymentTransaction->setActive(true)->setSuccessful(false);
            }
            return [
                'message' => $response->getResponseCode().' '.$response->getErrorDesc(),
                'successful' => !$response->hasError(),
            ];
        }

        protected function doInitRequest(PaymentTransaction $paymentTransaction, TransactionType
$transactionType): void
        {
            $request = (new InitRequest())->setShopId($this->getShopId($paymentTransaction))
                ->setTrType($transactionType)
                ->setAmount(Amount::fromString($paymentTransaction->getAmount()))
                ->setCurrencyCode(new Currency($paymentTransaction->getCurrency()))
                ->setLangId(Language::createFromLocaleOrDefault($this->localeSettings->getLocale()));

            $this->setReturnUrls($request, $paymentTransaction)->setCustomerData($request);

            $response = $this->gateway->paymentInit($request);
            $paymentTransaction->setSuccessful(false)->setActive(!$response->hasError())
                ->setRequest($request->toArray())->setResponse((array)$response->getResponseData());

            if ($response->hasError()) {
                $paymentTransaction->setActive(false);
                return $this->session->getFlashBag()->add('error', $response->getErrorDesc());
            }
            $paymentTransaction->setReference($response->getPaymentId());
        }

        protected function setReturnUrls(InitRequest $request, PaymentTransaction
$paymentTransaction): void
        {
            $request->setNotifyUrl($this->router->generate(
                'oro_payment_callback_return',
                ['accessIdentifier' => $paymentTransaction->getAccessIdentifier()],
                UrlGeneratorInterface::ABSOLUTE_URL,
            ))->setErrorUrl($this->router->generate(
                'oro_payment_callback_error',
                ['accessIdentifier' => $paymentTransaction->getAccessIdentifier()],
                UrlGeneratorInterface::ABSOLUTE_URL,
            ));
        }

        protected function setCustomerData(InitRequest $request): void
        {
            $customerUser = $this->tokenAccessor->getUser();
            $request->setShopUserRef($customerUser->getEmail())->setShopUserName($customerUser-
>getFullName())->setShopUserAccount($customerUser->getCustomer()->getName());
        }
    }
```

Даний клас CreditCardPaymentMethod реалізує інтерфейс PaymentMethodInterface платформи ORO і відповідає за інтеграцію з платіжною системою. Основна його функція полягає у виконанні платіжних транзакцій через шлюз, включаючи такі дії, як купівля, авторизація, зняття коштів і завершення транзакції. Клас використовує UniCreditGateway для взаємодії з API платіжної системи та містить методи для обробки транзакцій.

Наступним кроком потрібно додати конфігураційні параметри для нового платіжного методу, включаючи ключі API, секрети та інші параметри, необхідні для інтеграції. В коді всі налаштування можна отримати через фасад CreditCardPaymentConfig, що бере їх з env-файлу. Після цього реалізуються фактичні сервіси для обробки платіжних запитів, створення запитів до API платіжної системи, обробку відповідей та керування статусами транзакцій.

Важливим моментом в реалізації платіжної інтеграції є створення механізмів обробки помилок та повідомлень для користувачів, включаючи зміну статусів відповідних замовлень, що не були оплочені, відображення контекстних повідомлень у випадку неуспішних транзакцій та логування помилок для подальшого аналізу. Інтерфейс даної логіки вже надається платформою, відповідно для реалізації обробки помилок було достатньо реалізувати декілька класів, і зареєструвати їх для використання платформою.

Платформа представляє декілька класів подій, підписавшись на які можна реалізувати логіку обробки як успішного, так і неуспішного сценаріїв оплати. Нижче наведено клас UniCreditCallbackListener, який має 2 методи для обробки результату оплати: onError – у випадку помилки та onReturn – у випадку успішної оплати.

```
final class UniCreditCallbackListener
{
    use LoggerAwareTrait;

    protected PaymentMethodProviderInterface $paymentMethodProvider;

    public function __construct(PaymentMethodProviderInterface $paymentMethodProvider)
    {
        $this->paymentMethodProvider = $paymentMethodProvider;
    }
```

```
        public function onError(AbstractCallbackEvent $event): void
        {
            $paymentTransaction = $event->getPaymentTransaction();

            if (!$paymentTransaction) {
                return;
            }

            if (false === $this->paymentMethodProvider->hasPaymentMethod($paymentTransaction-
>getPaymentMethod())) {
                return;
            }

            $paymentTransaction
                ->setSuccessful(false)
                ->setActive(false);
        }

        public function onReturn(AbstractCallbackEvent $event): void
        {
            $paymentTransaction = $event->getPaymentTransaction();

            if (!$paymentTransaction) {
                return;
            }

            $paymentMethodId = $paymentTransaction->getPaymentMethod();

            if (false === $this->paymentMethodProvider->hasPaymentMethod($paymentMethodId)) {
                return;
            }

            try {
                $paymentMethod = $this->paymentMethodProvider-
>getPaymentMethod($paymentMethodId);
                $paymentMethod->execute(CreitCardPaymentMethod::COMPLETE, $paymentTransaction);

                $event->markSuccessful();
            } catch (\InvalidArgumentException $e) {
                if ($this->logger) {
                    // do not expose sensitive data in context
                    $this->logger->error($e->getMessage(), []);
                }
            }
        }
    }
```

Для тестування успішних та неуспішних транзакцій можна використовувати тестові картки, які надаються платіжною системою, що довзолить переконатися, що інтеграція коректно обробляє всі можливі сценарії. На рисунку 3.5 представлена сторінка тестової оплати замовлення через платіжну систему. Користувачеві необхідно ввести ім’я та прізвище, зазначені на карті, а також номер карти, CVV-код, після чого можна буде здійснити оплату замовлення.

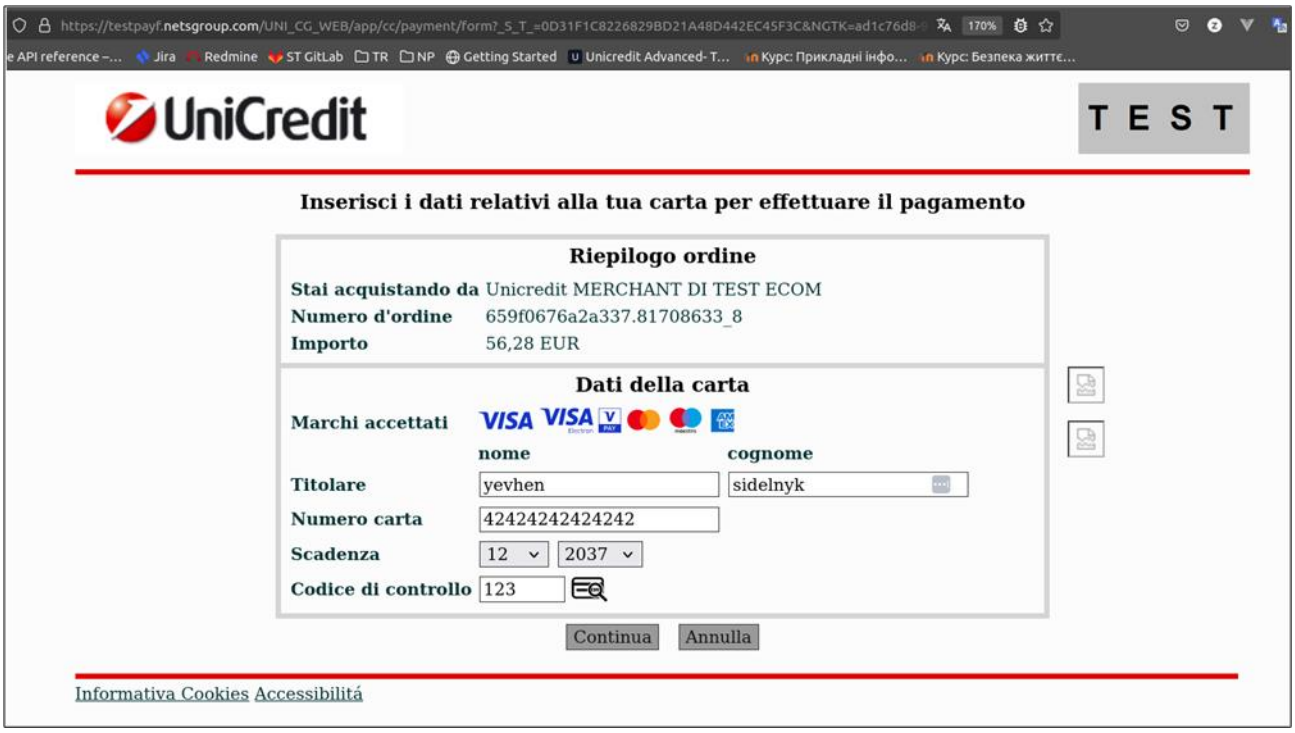


Рисунок 3.5 – Тестова сторінка оплати замовлення

Таким чином, інтеграція платіжної системи Unicredit на платформі ORO дозволила реалізувати швидкі та безпечні транзакції для користувачів інтернет-магазину, знизити витрати на обробку платежів та підвищити задоволеність клієнтів завдяки зручності та надійності платіжної системи.

3.4 Технічні характеристики застосунку

Розроблений інтернет-магазин для дистрибуції молочної продукції вимагає надійної та високопродуктивної серверної інфраструктури, а також забезпечення легкості доступу для користувачів через клієнтську частину. Нижче наведено технічні характеристики, які забезпечують оптимальну роботу серверної та клієнтської частин системи.

Серверні характеристики:

- процесор: мінімум 2-ядерний, 64-розрядний, з частотою 2 GHz;
- оперативна пам'ять: від 2 GB RAM;

- дисковий простір: мінімум 64 GB SSD, з можливістю збільшення для адаптації до зростаючого об'єму БД;

- операційна система: будь-який дистрибутив Debian-based Linux;

- інтернет-підключення: швидкість не менше 100 Мбіт/с.

Клієнтська частина системи може працювати на будь-якій платформі, що підтримує встановлення сучасних веб-браузерів. Єдиними умовами для користування системою є стабільне інтернет-підключення та браузер. Для оптимальної роботи рекомендується використовувати такий браузер, як Firefox, оскільки він є одним з найлегших та найбільш адаптивних для різних операційних систем. Вимоги до системи для запуску браузера Firefox:

- операційна система: Windows 7 або новіша, macOS 10.12 або новіша;

- процесор: Pentium 4 або новіший, який підтримує SSE2;

- оперативна пам'ять: 512 MB RAM для 32-бітної версії, 2 GB RAM для 64-бітної версії;

- дисковий простір: мінімум 200 МБ вільного місця на диску.

Дані характеристики гарантують оптимальне функціонування та високу продуктивність роботи системи. Надійна серверна інфраструктура та зручний доступ з клієнтської сторони забезпечать високий рівень задоволення користувачів і ефективність комерційної діяльності.

3.5 Тестування програмного продукту

Тестування розробленого програмного забезпечення є важливим етапом у процесі розробки будь-якого програмного продукту, що забезпечує перевірку його якості та відповідність вимогам з метою виявлення можливих помилок перед тим, як продукт може бути передано для користування клієнтам. Основна мета тестування полягає у ідентифікації помилок, дефектів або відхилень від заданих вимог, які могли бути допущені під час розробки продукту задля забезпечення того, щоб програмний продукт функціонував належним чином, був безпечним і зручним у використанні для кінцевих користувачів. В процесі

тестування, окрім технічної складової системи, аналізується також і поведінка при різних сценаріях її використання, взаємодія з іншими системами тощо.

Існує ряд підходів та методик тестування. До прикладу, функціональне тестування перевіряє відповідність бізнес вимогам, в той час як нефункціональне тестування оцінює аспекти продуктивності, безпеки та інші моменти, які безпосередньо не пов'язані з його функціональними характеристиками. Інтеграційне тестування допомагає визначити коректність взаємодії окремих частин програми, а системне тестування оцінює продукт у комплексі.

Важливо мати оптимальне співвідношення різних типів тестів для оптимального співвідношення покриття коду програмного продукту до швидкості виконання тестів. Однією з методик визначення даного співвідношення є концепція пірмади тестування, яка складається з трьох шарів: модульні тести, інтеграційні тести та end-to-end (E2E) тести. Візуальний вигляд співвідношення тестів у вигляді піраміди зображено на рисунку 3.6. В основі піраміди знаходяться модульні тести, які є найчисельнішими, середній шар складають інтеграційні, а на вершині розміщуються E2E та ручні тести, яких повинно бути найменше.

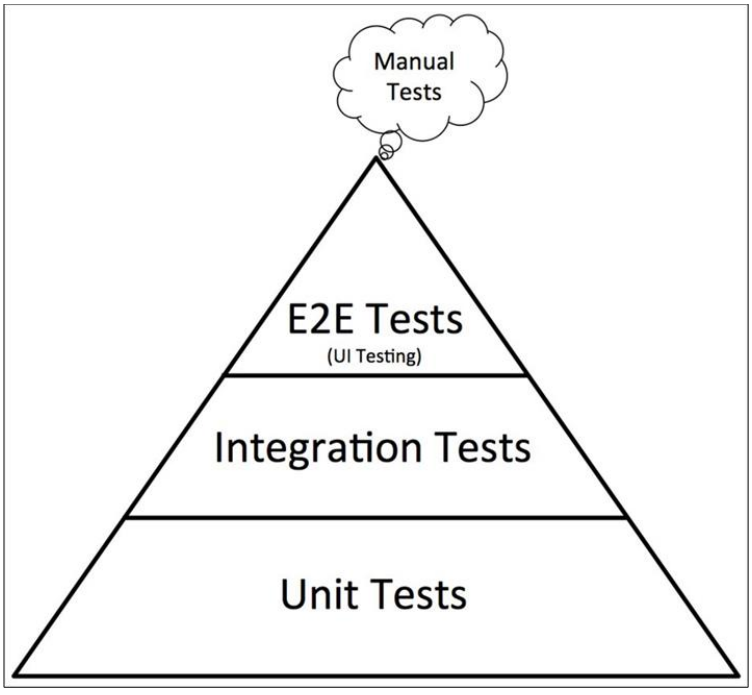


Рисунок 3.6 – Піраміда тестування

Підхід піраміди тестування допомагає забезпечити ефективне тестування, оскільки значна кількість модульних тестів дозволяє швидко і дешево виявляти помилки на ранніх етапах розробки, тоді як менша кількість інтеграційних і E2E тестів забезпечують верифікацію правильності роботи інтегрованих компонентів та кінцевої системи відповідно. Це пов'язано з тим, що модульні тести є найбільш швидкими у виконанні, адже перевіряють компоненти системи на рівні вихідного коду, в той час як інтеграційні тести є більш повільними, адже перевіряють взаємодію компонентів, а E2E тести є найдорожчими щодо часу виконання проводяться, втім найпростішими в реалізації, адже перевіряють роботу системи в цілому з точки зору кінцевого користувача і охоплюють широкий спектр використання програми від початку до кінця. Таким чином, використання піраміди тестування дозволяє досягнути високої якості програмного продукту при оптимальних витратах ресурсів.

Для написання тестових сценаріїв у вигляді коду часто зустрічаються два протилежнх підходи: тестування методом білого ящика та тестування методом чорного ящика. Обидва методи мають свою специфіку, переваги та недоліки.

Тестування методом білого ящика зосереджене на внутрішній структурі програмного продукту. Використовуючи даний метод, розробники мають доступ до всіх внутрішніх деталей модуля що підлягає тестуванню, що дозволяє перевірити алгоритми, логіку виконання програми, внутрішні дані, шляхи проходження даних та інші аспекти, які не видимі ззовні. Перевагами методу білого ящика є можливість виявлення складних помилок та детального аналізу логіки програми. Зазвичай представляється модульними тестами, і використовується для перевірки коректності логіки в компонентах системи, які є критичними для правильного функціонування продукту. До прикладу, у випадку інтернет-магазину це може бути логіка обрахунку вартості продукту, де потрібно покрити тестами всі можливі шляхи та випадки розрахунків. Одним з основних недоліків даного методу є те, що він вимагає глибоких знань у програмуванні та розумінні програмного продукту, що може бути часовитратним, адже дані тести зазвичай не самодокументовані.

Тестування методом чорного ящика, з іншого боку, зміщує акцент на функціональність системи без знання про її внутрішню реалізацію. Так перевіряється чи відповідає програма наданим вимогам, виконуючи тестові сценарії та перевіряючи вхідні й вихідні дані без знання про деталі того, як саме система внутрішньо обробляє ці дані. Цей метод ефективний для перевірки загальної функціональності програмного продукту. Втім, тестування методом чорного ящика не дає можливості виявити проблеми, що пов'язані з внутрішньою структурою коду, і може не виявити деякі види помилок, які можливі при нестандартному використанні програми.

Для переважної більшості програмних продуктів, тестування методом чорного ящика вважається більш рекомендованим підходом, особливо під час розробки та підтримки великих систем. Один з ключових аспектів даного методу полягає в тому, що він дозволяє вносити зміни в внутрішню реалізацію програми, не змінюючи тестові сценарії, що дозволяє підтримувати гнучкість програмної архітектури, можливість її розширення та радикального рефакторингу. Завдяки тому, що тести фокусуються на спостережуваному поводженні компонетів системи, а не на конкретній реалізації, вони стають менш залежними від конкретних архітектурних рішень. Це робить саме тестування більш стабільним і менш витратним у часі, а програму – більш адаптованою до змін, що сприяє легкому розширення існуючого функціоналу новим, можливість оновлення без ризику порушення існуючого функціоналу.

Також є так заване “димове тестування” (Smoke testing), що використовується для базової перевірки стабільності і функціональності системи. Цей вид тестування зазвичай проводиться для розгортання нової версії програмного забезпечення або окремої функціональної частини щоб переконатися, що основні функції системи працюють коректно, а критичні помилки відсутні. Smoke tests є поверхневими, швидкими і охоплюють лише основні аспекти програми, тому вони не замінюють глибоке тестування, але дозволяють швидко ідентифікувати великі проблеми, які можуть критично вплинути на доступність системи.

Прикладом smoke testing може бути перевірка, що основні веб сторінки сайту завантажуються успішно без помилок. Наприклад, тест може полягати в тому, щоб виконати HTTP запити до критично важливих роутів системи, як-от головна сторінка, сторінка логіну, основні розділи або панель управління, і перевірити, що ці запити повертають у відповідь 200-й статус-код, що означає, що сторінка була успішно завантажена без помилок зі сторони серверу. Відповідно, даний тип тестування дає швидке уявлення про те, чи система загалом функціонує і чи доступні основні функції в новій версії коду.

Для тестування системи, розробленої на основі фреймворку Symfony, можна використовувати ряд технічних інструментів, які допомагають забезпечити якість і стабільність програмного коду. Symfony надає потужний набір базових класів для написання різного роду тестів, що дозволяє перевіряти як окремі компоненти системи, так і їхню інтеграцію та взаємодію.

Одним з основних інструментів для тестування в Symfony є базовий клас KernelTestCase, що використовується для інтеграційних тестів і забезпечує можливість завантаження ядра Symfony для виконання тестів у повному контексті. З даним класом, можна перевірити взаємодію сервісів, компонентів і конфігурацій системи, що допомагає виявляти проблеми на рівні інтеграції.

Для тестування веб-застосунків Symfony надає клас WebTestCase, який розширює можливості KernelTestCase, дозволяючи створювати функціональні тести для перевірки поведінки додатку з точки зору клієнта. За допомогою WebTestCase можна емулювати HTTP-запити, проводити роботу з формами, і перевіряти відповіді сервера на дані запити. Це дозволяє перевірити роботу маршрутизації, контролерів, валідаторів та інших компонентів веб-застосунку.

Крім того, спільнота Symfony підтримує інтеграції з різними зовнішніми бібліотеками для тестування, такими як Behat для поведінкового тестування (end-to-end), який нативно інтегрований в платформу ORO, що дозволяє описувати тестові сценарії мовою, близькою до природної. Такі інструменти як DoctrineFixturesBundle можуть використовуватись для завантаження тестових даних у базу даних, що є корисним для підготовки середовища для тестування.

В рамках розробленого інтернет-магазину, більшість тестових сценаріїв була проведена в ручному режимі через нестачу часу для написання автоматизованих тестів. Втім, для оптимізації деяких аспектів тестування, частину підготовки даних було автоматизовано за допомогою скрипта для завантаження фікстурних демо-даних.

Фікстури дозволили значно скоротити час на підготовку тестових даних та забезпечити легкість відновлення системи до початкового стану. Однією з таких фікстур є LoadCustomerUsersAppDemoData, код якої наведено нижче, яка завантажує дані про користувачів із CSV-файлу в систему. Використання цієї фікстури дозволило автоматично створювати тестові записи користувачів, що полегшило процес тестування за рахунок наявності необхідних даних.

```
final class LoadCustomerUsersAppDemoData extends AbstractLoadCustomerUserAppDemoData
implements VersionedFixtureInterface, LoadedFixtureVersionAwareInterface
{
    use LoadedFixtureVersionAwareTrait;

    public const USERS_PATH = '@AppCustomerBundle/Migrations/Data/App/ORM/data/customer-
users.csv';
    public const VERSION = '1.1';

    public function getVersion(): string
    {
        return self::VERSION;
    }

    public function getDependencies(): array
    {
        return [LoadCustomerAppDemoData::class];
    }

    protected function getCustomerUsersCSV(): string
    {
        return self::USERS_PATH;
    }

    protected function getCustomerUserRole($roleLabel, ObjectManager $manager, array $row)
    {
        return $this->container->get('doctrine')
            ->getManagerForClass(CustomerUserRole::class)
            ->getRepository(CustomerUserRole::class)->findOneBy(['label' => $roleLabel, 'organization'
=> $this->getOrganization($manager, $row)]);
    }

    protected function getCustomer(ObjectManager $manager, array $row)
    {
        $ref = LoadCustomerAppDemoData::customerReference(
    $row['organization'], $row['customer']);
        return $this->hasReference($ref) ? $this->getReference($ref) :
```

```
                $manager->getRepository(Customer::class)
                ->findOneBy(['organization' => $row['organization'], 'name' => $row['customer']]);
        }

        protected function getOrganization(ObjectManager $manager, array $row)
        {
            $findOrganization = new FindOrganizationReferenceByInternalName($this-
>referenceRepository);

            return $findOrganization($row['organization']);
        }

        protected function getWebsite(ObjectManager $manager, array $row)
        {
            return $manager->getRepository(Website::class)->findOneBy([
                'organization' => $this->getOrganization($manager, $row),
                'name' => LoadWebsiteData::DEFAULT_WEBSITE_NAME,
            ]);
        }

        protected function shouldProcessRow(array $row): bool
        {
            $version = $row['version'] ?? '0.0';

            return !$this->versionIsLoaded($version);
        }
    }
```

Окрім користувачів, було створено ряд інших фікстур, для підготовки тестових даних магазину. До прикладу, фікстура LoadDeliverySlotsAppDemoData використовується для завантаження даних про слоти доставки, які необхідні для функціональності вибору дати доставки замовлень. Фікстури LoadProductAppDemoData та LoadProductPriceAppDemoData відповідальні за завантаження інформації про товари та їх ціни відповідно, без яких неможливе тестування каталогу продуктів та розрахунку вартості замовлень. Також фікстура LoadPriceListToCustomerAppDemoData завантажує тестові списки цін для користувачів, маючи які можна робити покупки та оформляти замовлення.

Окрім ручного тестування, деяка складна бізнес-логіка інтернет-магазину була перевірена за допомогою модульних тестів. До прикладу, тест NearestDeliverySlotProviderTest перевіряє функціональність визначення найближчого доступного слота доставки для користувача, що реалізована класом NearestDeliverySlotProvider. Для перевірки правильності роботи сценаріїв пошуку найближчого слота доставки, тест використовує утиліти бібліотеки PHPUnit. Вихідний код даного тесту наведено нижче.

```
/** @covers NearestDeliverySlotProvider */
final class NearestDeliverySlotProviderTest extends TestCase
{
    private NearestDeliverySlotProvider $provider;

    private WorkingDaysProviderInterface&MockObject $workingDaysProvider;

    private LocalizedCarbonFactory&MockObject $localizedCarbonFactory;

    protected function setUp(): void
    {
        parent::setUp();

        $this->workingDaysProvider = $this->createMock(WorkingDaysProviderInterface::class);
        $localeSettings = $this->createMock(LocaleSettings::class);
        $localeSettings->method('getTimeZone')->willReturn(date_default_timezone_get());
        $this->localizedCarbonFactory = new LocalizedCarbonFactory($localeSettings);
        $configProvider = $this->createMock(ConfigProvider::class);
        $configProvider->method('getDisplayedWeeks')->willReturn(3);
        $deliverySlotWrapperFactory = new DeliverySlotsWrapperFactory($this->localizedCarbonFactory);
        $this->provider = new NearestDeliverySlotProvider($this->workingDaysProvider, $this->localizedCarbonFactory, $deliverySlotWrapperFactory, $configProvider);
    }

    public function testThrowsExceptionIfNoSlotsGiven(): void
    {
        $this->expectException(NoResultException::class);

        $this->provider->getNearestDeliverySlot([], 'FR');
    }

    public function testSelectsFirstAvailableSlotForUser(): void
    {
        // dayOfWeek 4 (Thursday)
        CarbonImmutable::setTestNow('2021-12-09 09:00:00');

        $slotExpired = $this->slot(
            4, // Thursday, today
            '17:00',
            '18:00',
            4,  // Thursday, today
            '08:59',
        );

        $slotToBeUsed = $this->slot(
            6, // Saturday
            '09:00',
            '10:00',
            4, // Today, Thursday
            '18:00',
        );

        $nextAfterUsed = $this->slot(
            5, // Friday
            '08:00',
            '08:30',
            4, // Today, Thursday
            '18:01' // one minute after slot to be used
        );
        $this->workingDaysProvider->method('isWorkingDay')
            ->willReturnCallback(
```

```
                static function (DateTimeInterface $date, string $code) {
                    $format = $date->format('Y-m-d');
                    return $code === 'FR' && ('2021-12-09' === $format || '2021-12-10' === $format ||
                        '2021-12-11' === $format || '2021-12-16' === $format); // for $slotExpired next week
                }
            );

        $userSlot = $this->provider->getNearestDeliverySlot([$slotExpired, $slotToBeUsed,
$nextAfterUsed], 'FR');

        self::assertFalse($userSlot->isExpired($this->localizedCarbonFactory->now()));
        self::assertSame('2021-12-11', $userSlot->getDeliveryDate()->format('Y-m-d'));
        self::assertSame('2021-12-09 18:00:00', $userSlot->getCutoffDateTime()->format('Y-m-d
H:i:s'));
        self::assertSame('2021-12-11 09:00:00', $userSlot->getStartDateTime()->format('Y-m-d H:i:s'));
        self::assertSame('2021-12-11 10:00:00', $userSlot->getEndDateTime()->format('Y-m-d H:i:s'));
    }

    public function testTriesNextWeekDayIfHoliday(): void
    {
        // dayOfWeek 4 (Thursday)
        CarbonImmutable::setTestNow('2021-12-09 09:00:00');

        $slot = $this->slot(
            5, // Friday
            '09:30',
            '09:45',
            4, // Today, Thursday
            '19:00',
        );

        $this->workingDaysProvider->method('isWorkingDay')
            ->willReturnCallback(
                static function (DateTimeInterface $date, string $code) {
                    $format = $date->format('Y-m-d');

                    return $code === 'FR' && '2021-12-10' !== $format
                        && ('2021-12-09' === $format || '2021-12-17' === $format);
                }
            );

        $userSlot = $this->provider->getNearestDeliverySlot([$slot], 'FR');

        self::assertFalse($userSlot->isExpired($this->localizedCarbonFactory->now()));
        self::assertSame('2021-12-17', $userSlot->getDeliveryDate()->format('Y-m-d'));
        self::assertSame('2021-12-16 19:00:00', $userSlot->getCutoffDateTime()->format('Y-m-d
H:i:s'));
        self::assertSame('2021-12-17 09:30:00', $userSlot->getStartDateTime()->format('Y-m-d H:i:s'));
        self::assertSame('2021-12-17 09:45:00', $userSlot->getEndDateTime()->format('Y-m-d H:i:s'));
    }

    private function slot($day, $begin, $end, $cutoffDay, $cutoffTime): DeliverySlot
    {
        $slot = new DeliverySlot();
        $slot->setDay($day);
        $slot->getDeliveryTimeRange()->setBegin(CarbonImmutable::createFromTimeString($begin));
        $slot->getDeliveryTimeRange()->setEnd(CarbonImmutable::createFromTimeString($end));
        $slot->setCutoffDay($cutoffDay);
        $slot->setCutoffTime(CarbonImmutable::createFromTimeString($cutoffTime));
        return $slot;
    }
}
```

Тестовий клас перевіряє кілька сценаріїв, щоб упевнитися, що логіка вибору найближчого слота доставки працює належним чином, покриваючи випадки з відсутністю слотів, вибором першого доступного слота і врахуванням робочих днів та свят.

Модульні тести, як показано в прикладі вище, дозволяють детально перевірити складну бізнес-логіку і забезпечити, щоб кожен компонент системи працював відповідно до очікувань у всіх тестових сценаріях. Це особливо важливо для критичних частин програмного забезпечення, де коректність обчислень та правильність алгоритмів є надзвичайно важливими.

Таким чином, тестування відіграло значну роль у забезпеченні якості та надійності інтернет-магазину. Складна бізнес-логіка була перевірена за допомогою модульних тестів, що гарантувало коректність ключових компонентів системи. Також, зважаючи на обмежений час, значну частину тестових сценаріїв основної функціональності системи було перевірено вручну. Автоматизація підготовки даних за допомогою фікстур спростила процес тестування і забезпечила стабільність тестового середовища. Використання різних підходів та інструментів для тестування забезпечило всебічну перевірку продукту, підвищуючи його загальну якість і готовність до використання.

3.6 Інструкція користувача

Робота користувача з системою інтернет-магазину починається з входу в обліковий запис. Щоб мати доступ до повного функціоналу системи, необхідно, щоб менеджер з продажу спочатку створив аккаунт для користувача через адміністративну панель. Після цього користувач може перейти на сайт і автентифікуватись, використовуючи свої логін та пароль. Неавтентифікований користувач все ж має змогу переглядати деякі сторінки сайту, однак йому не доступна інформація про фінальні ціни продуктів, які він міг би придбати будучи зареєстрованим. Відтак, першим і основним кроком у використанні системи є вхід в систему через форму логіну на головній сторінці веб-сайту.

Вигляд сайту для неавтентифікованого користувача представлено на рисунку 3.7. Анонімний користувач має змогу переглядати список продуктів, але не бачить їхні ціни. Це зроблено через те, що ціна на один і той самий продукт може відрізнятися від користувача до користувача. Така практика є типовою для B2B систем, де ціни можуть змінюватися в залежності від умов співпраці, обсягів закупівель та індивідуальних домовленостей. Тому для того, щоб побачити свою персональну ціну на продукт, користувач обов'язково повинен автентифікуватись в системі.

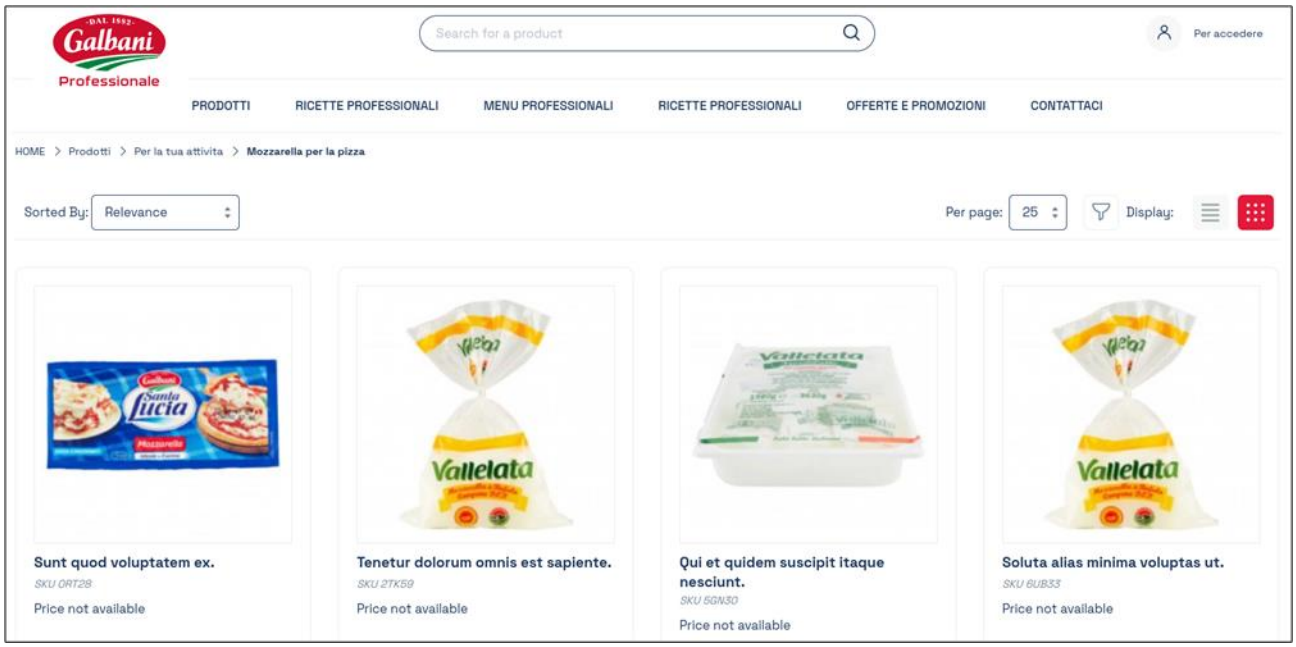


Рисунок 3.7 – Вигляд сайту для неавтентифікованого користувача

Автентифікація в системі здійснюється через форму логіну, яка вимагає введення адреси електронної пошти та паролю. Дані користувача мали бути заздалегідь внесені в систему менеджером з продажу в рамках попередньої домовленості з клієнтом. Відповідно, без цих даних доступ користувача до функціоналу сайту буде неможливий. Форма логіну представлена на рисунку 3.8, де користувачі можуть ввести дані автентифікації, щоб ввійти в систему і отримати повноцінний доступ до всіх функцій магазину.

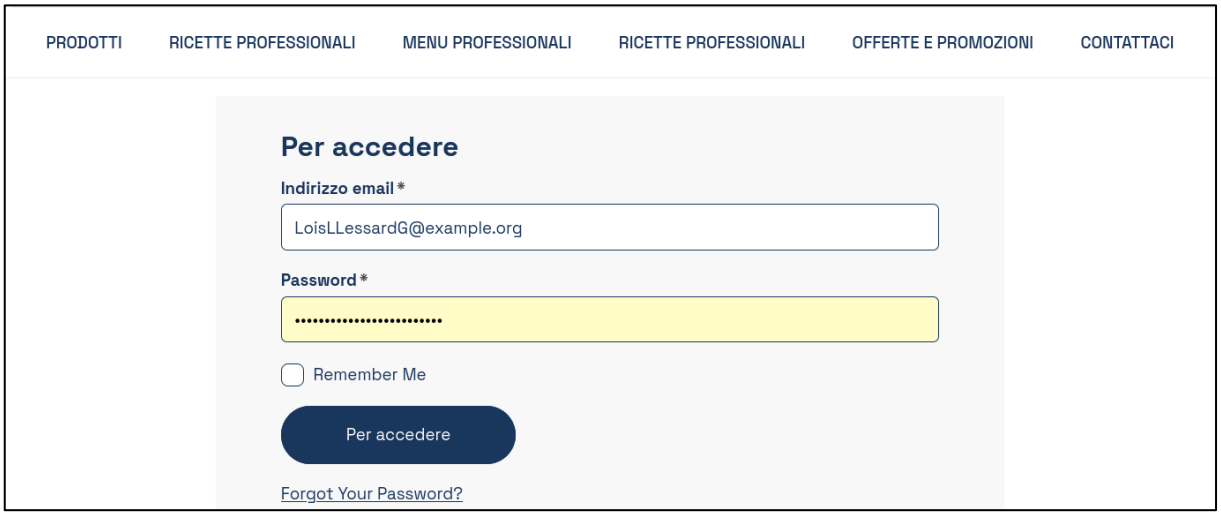


Рисунок 3.8 – Форма автентифікації

Наступним кроком після логіну користувача, найбільш імовірно, є пошук продуктів. Для цього в системі є веб-каталог, представлений у верхньому меню, де користувач може обрати категорію продуктів для перегляду. Крім того, передбачено спеціальне вікно повнотекстового пошуку, куди можна ввести ключові слова для знаходження бажаних продуктів. Ввівши пошуковий запит, користувачеві відображається список знайдених продуктів, включаючи персоналізовану ціну, як зображено на рисунку 3.9.

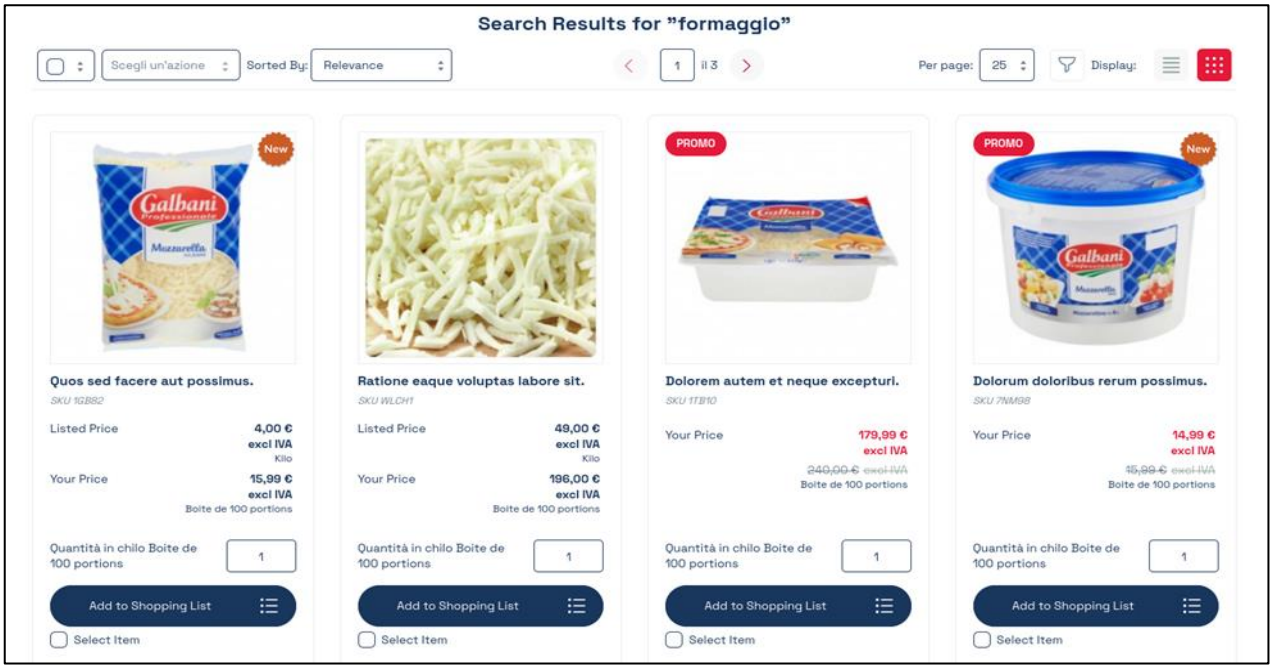


Рисунок 3.9 – Пошук продуктів

Далі, знайшовши потрібний продукт, користувач має можливість додати його до списку покупок. Це можна зробити за допомогою відповідної кнопки, розміщеної поряд з кожним продуктом в списку результатів пошуку. Після цього користувач може або продовжити пошук інших продуктів, додаючи їх до кошика, або перейти до перегляду свого списку покупок.

Коли користувач обрав усі продукти, які він бажає придбати, він може перейти до перегляду вмісту свого кошика на сторінку списку покупок. Тут відображається перелік обраних продуктів, зазначена кількість по кожному з них, вказана ціна за одиницю, наявність та сума екологічних податків для відповідних продуктів, а також попередньо підбита сума за кожен елемент переліку. В правій частині сторінки відображається підсумкова ціна, яка представляє загальну вартість всіх продуктів даного списку покупок.

На рисунку 3.10 можна наочно побачити вигляд цієї сторінки, де показано як користувач може переглядати та управляти своїм списком покупок.

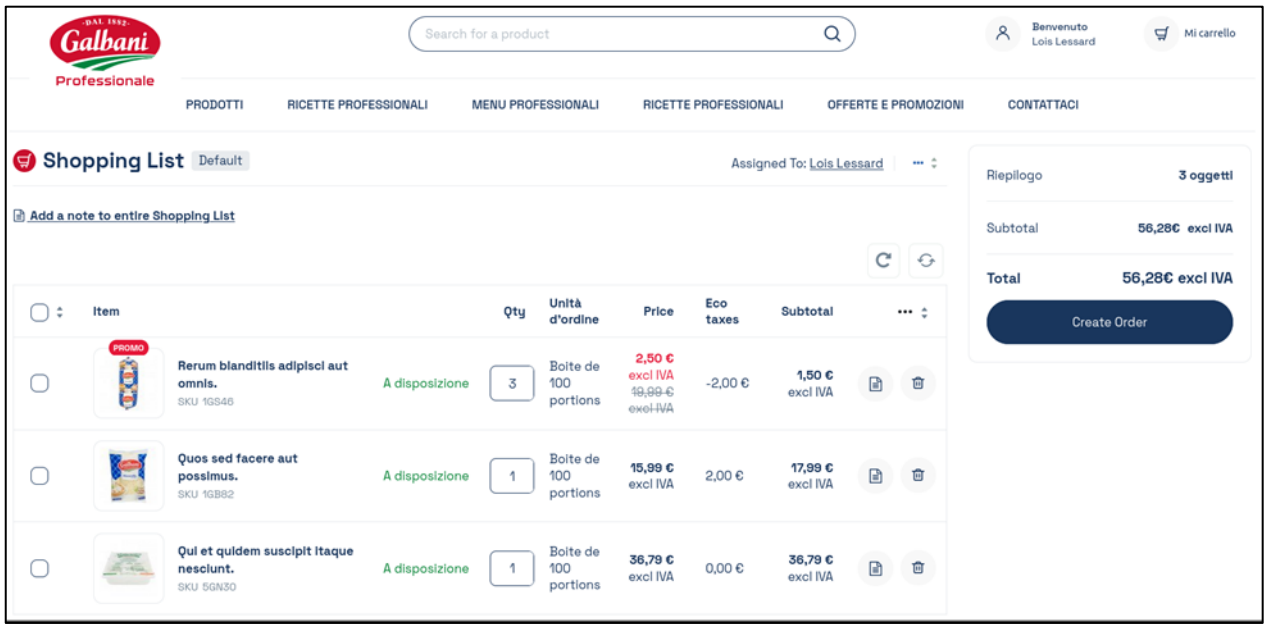


Рисунок 3.10 – Сторінка списку покупок

На прикладі вище, в списку покупок є три продукта. Перший з них має спеціальне маркування "промо", розміщене над зображенням продукту. Ціна цього продукту виділена червоним кольором і становить 2,5 євро, під якою

розміщена перекреслена ціна 19,99 євро. Таке оформлення вказує на те, що на продукт зараз діє спеціальна промоційна пропозиція, що дозволяє користувачеві придбати його за значно зниженою ціною.

Наступним логічним кроком у процесі покупки є оформлення замовлення. Користувач, натиснувши кнопку «створити замовлення», потрапляє на сторінку оформлення замовлення. Тут він має можливість обрати спосіб оплати, вказати інформацію про платника, а також вказати спосіб та адресу доставки продуктів. Крім того, дана сторінка передбачає фінальний перегляд всіх обраних елементів замолення разом з їхніми цінами, що дає змогу впевнитись в точності замовлення перед його фіналізацією. Вигляд сторінки зображено на рисунку 3.11.

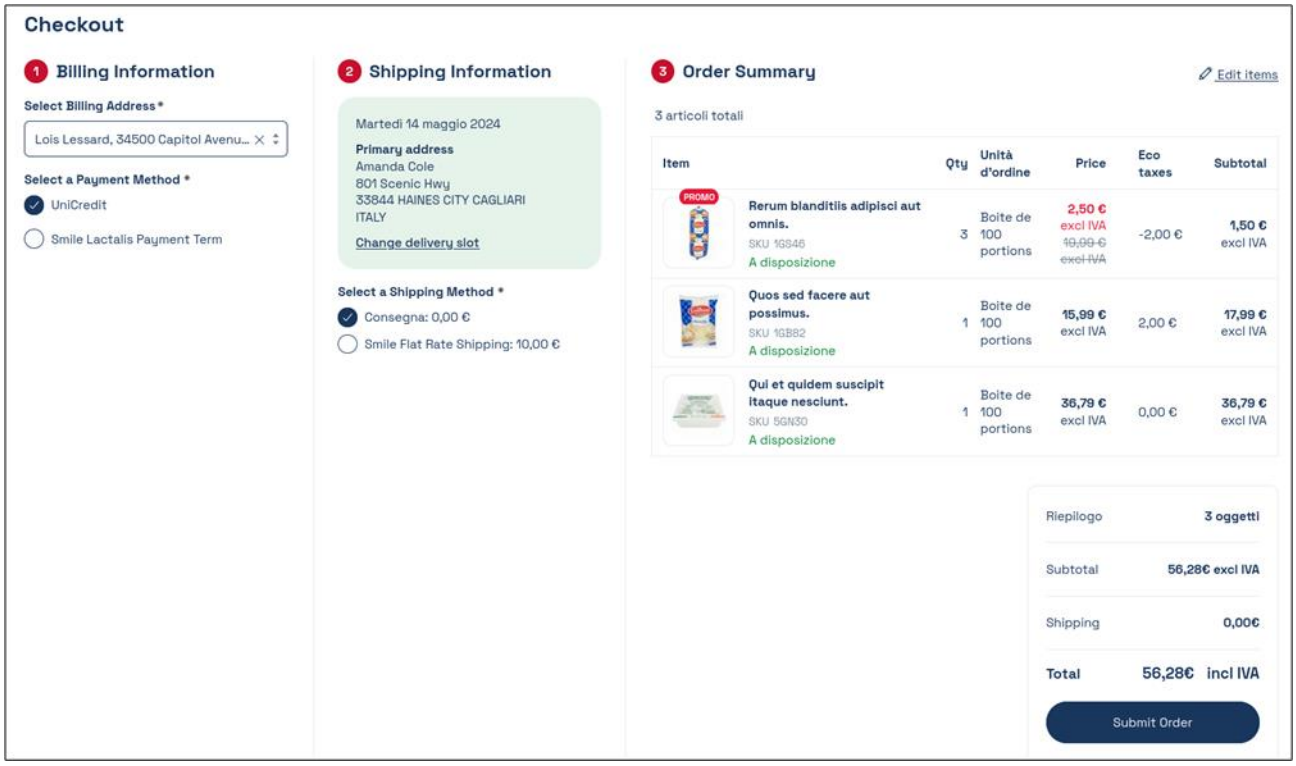


Рисунок 3.11 – Сторінка оформлення замовлення

Переглянувши всі деталі замовлення та переконавшись у їх коректності, користувач готовий до оплати. При натисненні на кнопку «Підтвердити замовлення», система переходить до фіналізації покупки, логіка якої залежить від обраного методу оплати. Зокрема, як видно з рисунку вище, користувачеві доступно два основних способи оплати: через платіжну систему Unicredit та

домовлений платіж у визначений строк. Обравши домовлений платіж, розрахунок буде здійснюватися під час особистої зустрічі готівкою або будь-яким іншим погодженим способом у зазначений строк.

Якщо користувач обирає оплату через систему Unicredit, він потрапляє на сторінку оплати цієї платіжної системи. Сторінка оплати містить поля для введення даних про платіжну карту, включаючи ім'я та прізвище платника, номер карти, термін дії та CVV-код, які необхідні для проведення транзакції. Сторінка оплати замовлення ілюструється на рисунку 3.12, де покупець заповнює всі необхідні для оплати дані.

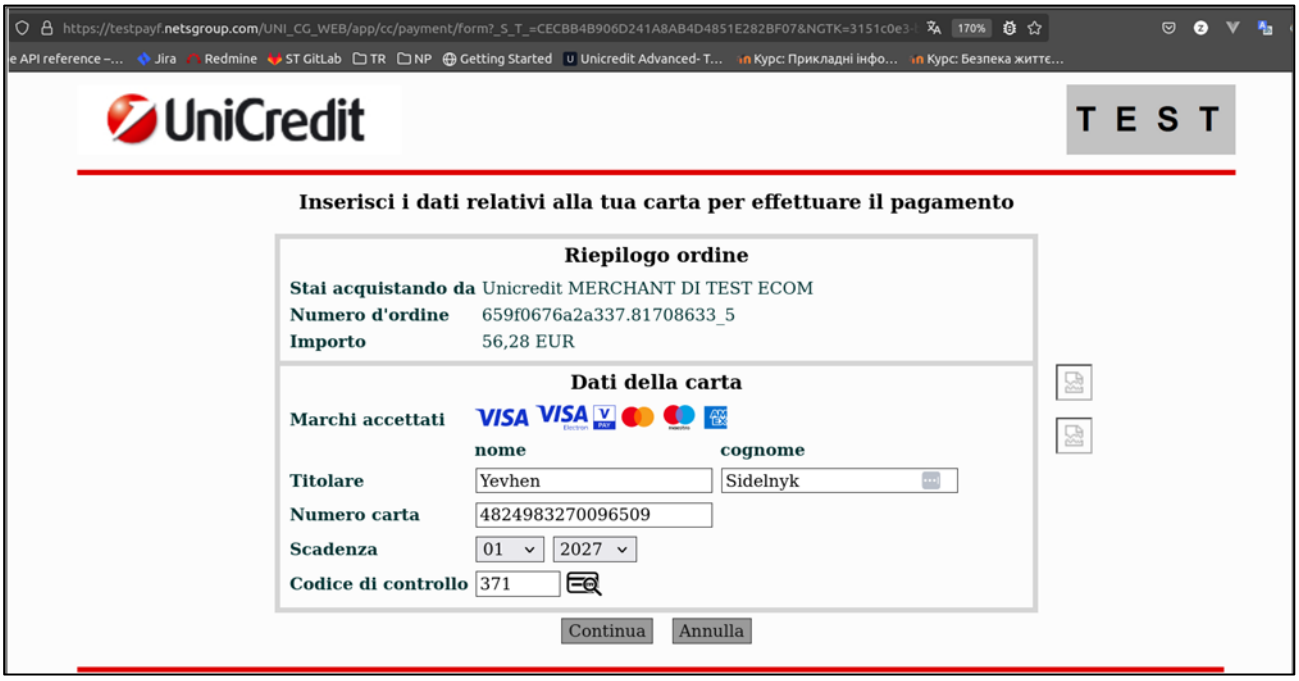


Рисунок 3.12 – Сторінка оплати замовлення

Після натискання на кнопку «продовжити», користувачеві пропонується ще одне вікно для підтвердження платежу, де він може останній раз перевірити дані своєї транзакції. Цей крок процесу оплати зображено на рисунку 3.13. Він є важливим для попередження помилок, спричинених людським фактором, які могли б виникнути через некоректне введення даних карти. Після підтвердження платежу з облікового запису користувача автоматично знімаються кошти відповідно до суми замовлення.

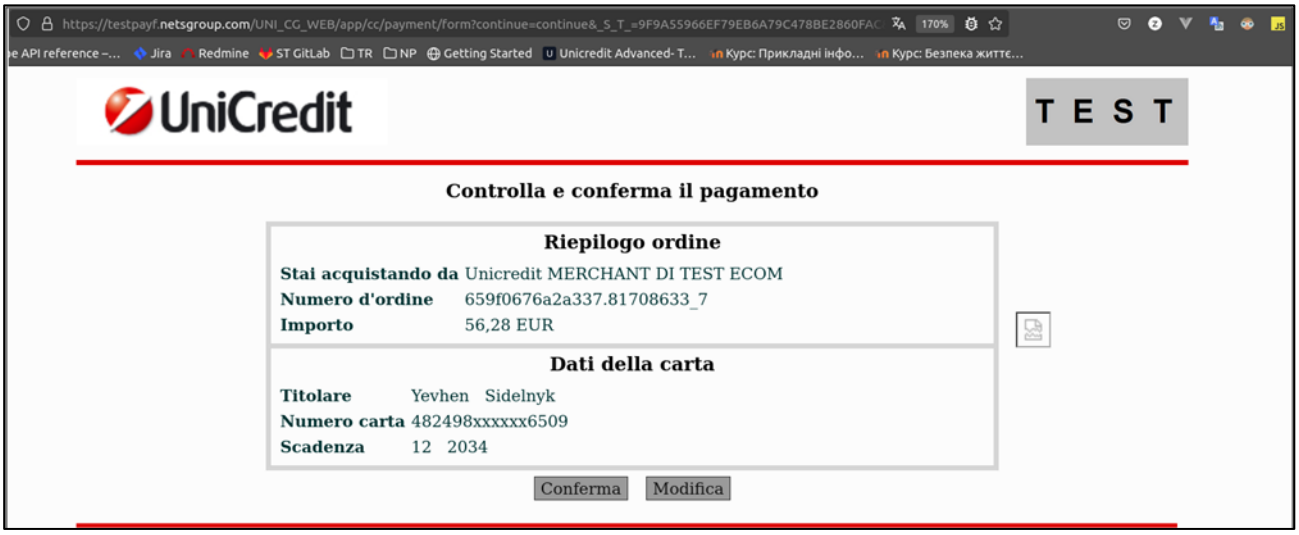


Рисунок 3.13 – Останній крок підтвердження замовлення

Після завершення оплати, користувач перенаправляється на вікно з результатами проведеної транзакції. Тут він може побачити статус та суму транзакції, короткі деталі замовлення, такі як його номер, дату й час, а також відомості про карту, з якої були зняті кошти. Крім того, дана сторінка містить кнопку для повернення на сайт магазину, так що користувач може продовжити покупки. Вигляд успішно проведеної оплати зображено на рисунку 3.14.

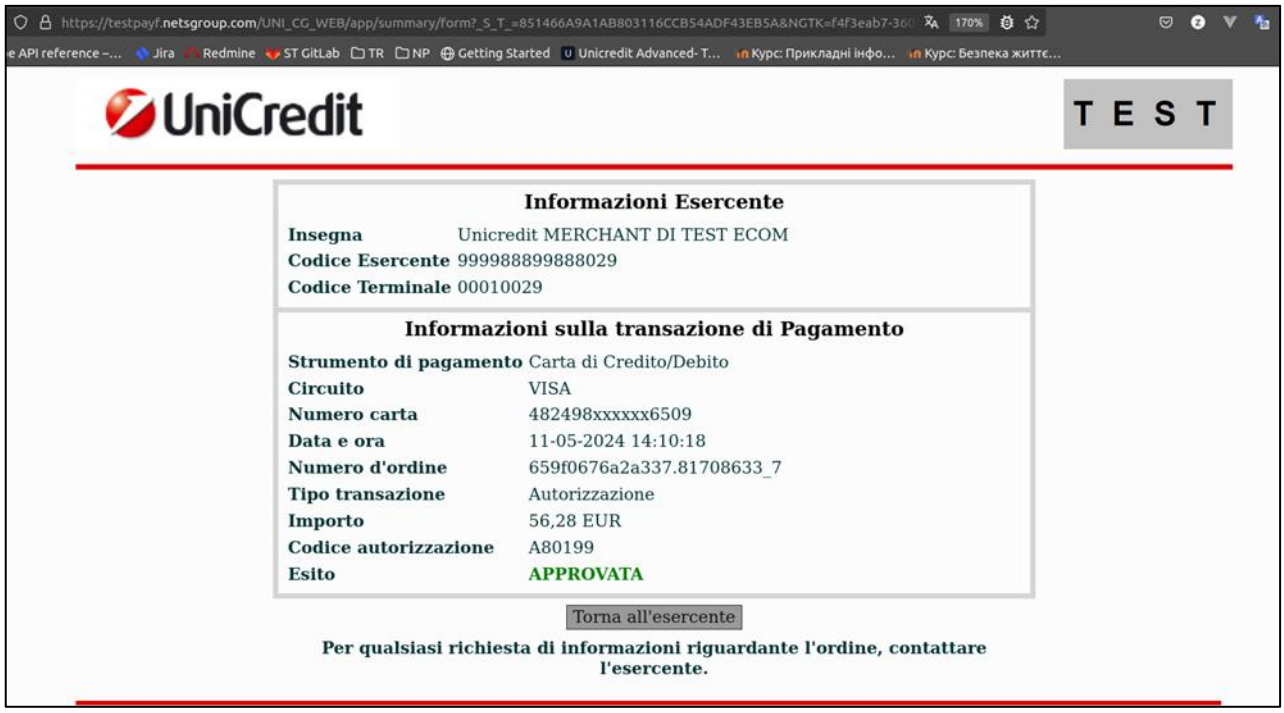


Рисунок 3.14 – Вікно успішно проведеної оплати

Після завершення оплати, покупцеві надається можливість повернутися до сайту магазину, натиснувши на відповідну кнопку. Там його зустрічає інформаційна сторінка, схожа на ту, що зображена на рисунку 3.15, де користувач повідомляється про те, що замовлення було успішно створено і взято в обробку. У цьому вікні також міститься посилання для перегляду деталей замовлення, і повідомлення про те, що на електронну адресу користувача буде надіслано лист з деталями замовлення, надаючи зручний доступ до всієї необхідної інформації.

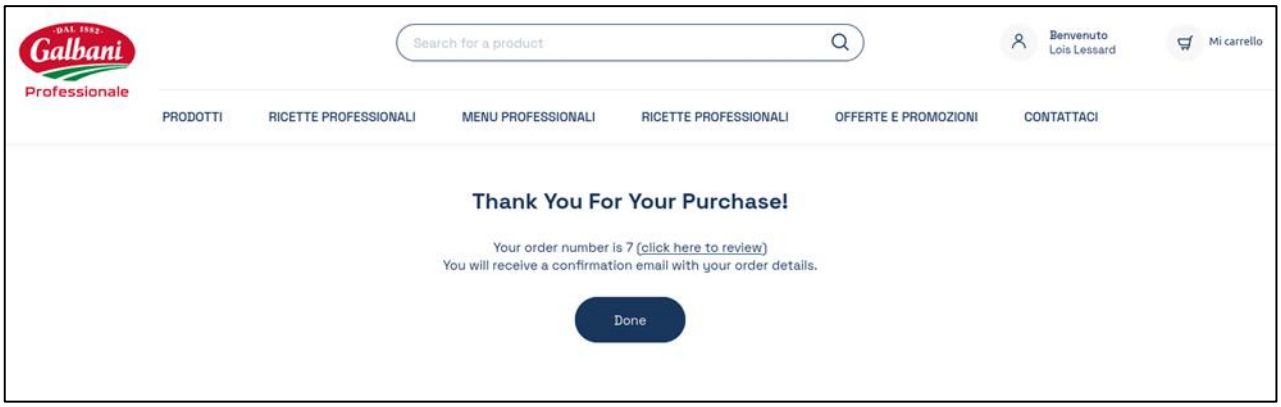


Рисунок 3.15 – Інформаційна сторінка оплоченого замовлення

Після оформлення замовлення користувач має можливість продовжити свої покупки. Він може шукати інші потрібні продукти, додавати їх до кошика та оформлювати нові замовлення, оплачуючи ці продукти таким самим чином, як було показано вище. Крім того, покупець має змогу в будь-який момент часу переглянути інформацію про статус існуючих замовлень у списку своїх замовлень в особистому кабінеті.

Таким чином, розроблена система інтернет-магазину оптового продажу молочної продукції надає зручний інтерфейс для дистриб'юторів задля ефективного ведення комерційної діяльності в інтернеті. Завдяки інтуїтивно зрозумілій навігації, можливості авторизації та персоналізованого підходу до кожного клієнта, користувачі мають змогу легко шукати та вибирати продукти, оформляти замовлення та здійснювати платежі.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи спершу було проведено дослідження предметної області, що дозволило глибше зрозуміти особливості B2B-сектора італійського ринку та специфіки роботи з оптовими клієнтами у сфері продажу молочної продукції. Після цього було проаналізовано існуючі програмні рішення, їх переваги та недоліки, що в подальшому допомогло обрати найбільш підходящий для реалізації проекту стек технологій. Згодом після аналізу існуючих рішень було сформульовано вимоги до розробки інтернет-магазину, відповідно до яких і відбувалась реалізація проекту. Розроблені вимоги та концепція системи були детально описані за допомогою діаграм варіантів використання, діаграм активностей та інших інструментів візуалізації бізнес-процесів, що забезпечило наочне та структуроване розуміння задач, які стоять перед проектом.

Наступним кроком стало проектування програмного забезпечення, яке почалося з аналізу та вибору архітектури системи. Було обрано клієнт-серверну архітектуру з використанням протоколу REST для комунікації між клієнтом та сервером, що є найпоширенішим варіантом для веб-додатків. Далі було проведено роботу з проектування архітектури та структури системи, де розглядалися два можливі підходи до реалізації: модульний моноліт та мікросервісна архітектура. Для реалізації було надано перевагу модульному моноліту, оскільки це простіший в реалізації варіант. Завершальним етапом у проектуванні став аналіз та вибір технологій для реалізації програмної системи, в рамках якого було визначено стек технологій, на якому і була в кінцевому рахунку побудована система.

Основним етапом став етап програмної реалізації, в рамках якого були виконані технічні вимоги до проекту. Було проведено детальне проектування модулів, де описані основні технічні аспекти практичної реалізації функціоналу згідно з обраним стеком технологій. Також, особлива увага була приділена розробці функціоналу комбінованих промо-цін та супутніх критичних аспектів

бізнес-вимог, пов'язаних з оплатою. По завершенню реалізації веб-додатку в коді, було сформовано перелік технічних характеристик застосунку, які повною мірою відповідають можливостям середньостатистичних користувачів.

Фінальним етапом стало тестування програмного застосунку та написання керівництва користувача. За рахунок тестування було належним чином перевірено функціональність системи, виявлено та виправлено знайдені помилки. Було також сформовано документ, де описано керівництво користувача, в якому описано докладні інструкції щодо використання системи, охоплюючи всі основні функції інтернет-магазину, надаючи користувачам розуміння як ефективно взаємодіяти з системою, включаючи процеси авторизації, пошуку та замовлення продуктів, а також оплати та управління замовленнями.

Таким чином, результатом роботи над кваліфікаційною роботою стало створення інтернет-магазину для оптового продажу молочної продукції дистриб’юторам, який включає повний цикл обробки замовлень для оптових покупців. Система задовольняє поставлені вимоги та працює стабільно, забезпечуючи оптимізацію внутрішніх процесів бізнесу для власників продукту, а також зручний інтерфейс і функціональність для користувачів магазину.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Електронна комерція бізнесу до бізнесу [Електронний ресурс] // Стаття на Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/B2B_e-commerce

2. Аналіз зростаючої кількості B2B компаній та щільності мережі [Електронний ресурс] // Блог компанії Crossbeam. – Режим доступу: https://www.crossbeam.com/blog/b2b-network-crossbeams-data-and-learnings/

3. Прогнози на значний ріст ринку молочної продукції до 2029 року [Електронний ресурс] // Публікація компанії Mordor Intelligence. – Режим доступу: https://www.mordorintelligence.com/industry-reports/europe-dairy-market

4. Статистика розподілення ринку збуту молочної продукції в Італії [Електронний ресурс] // Веб-сайт компанії Clal.it. – Режим доступу: https://www.clal.it/en/?section=quadro_italia

5. Опис ключових сутностей інтернет-магазину [Електронний ресурс] // Документація ORO Commerce. – Режим доступу: https://doc.oroinc.com/user/glossary/#term-Shopping-List

6. Платформа для електронної комерції Sylius [Електронний ресурс] // Веб-сайт Sylius. – Режим доступу: https://sylius.com/

7. Платформа для розробки інтернет-магазинів Magento [Електронний ресурс] // Веб-сайт Adobe Commerce. – Режим доступу: https://business.adobe.com/products/magento/magento-commerce.html

8. Платформа для електронної комерції OpenCart [Електронний ресурс] // Веб-сайт OpenCart. – Режим доступу: https://www.opencart.com/

9. Спеціалізована платформа для B2B комерції OROCommerce [Електронний ресурс] // Веб-сайт компанії ORO inc. – Режим доступу: https://oroinc.com/b2b-ecommerce/

10. Персоналізоване налаштування цін під користувача за допомогою можливостей ORO [Електронний ресурс] // Документація ORO Commerce. – Режим доступу: https://doc.oroinc.com/user/concept-guides/catalog-promotions/pricing/#merge-by-priority

11. Огляд протоколу HTTP [Електронний ресурс] // Документація MDN Web Docs. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

12. Опис архітектури RESTful API [Електронний ресурс] // Стаття від Amazon AWS. – Режим доступу: https://aws.amazon.com/what-is/restful-api/

13. Офіційний сайт Json Web Token [Електронний ресурс] // Веб сайт JWT. – Режим доступу: https://jwt.io/

14. Amazon Elastic Container Service [Електронний ресурс] // Лендінг сторінка Amazon AWS ECS. – Режим доступу: https://aws.amazon.com/ecs/

15. Переваги мікроядерної архітектури над мікросервісами [Електронний ресурс] // Блог Роберта Мартіна. – Режим доступу: http://blog.cleancoder.com/uncle-bob/2014/09/19/MicroServicesAndJars.html

16. Стратегія початкової розробки системи у вигляді моноліту з подальшим переходом на мікросервіси [Електронний ресурс] // Блог Мартіна Фовлера. – Режим доступу: https://martinfowler.com/bliki/MonolithFirst.html

17. Принципи SOLID [Електронний ресурс] // Стаття від Sam Millington. – Режим доступу: https://www.baeldung.com/solid-principles

18. Статичний аналізатор архітектури Deptrac [Електронний ресурс] // Репозиторій на Github. – Режим доступу: https://github.com/qossmic/deptrac

19. Статичний аналізатор архітектури PHPAT [Електронний ресурс] // Репозиторій на Github. – Режим доступу: https://github.com/carlosas/phpat

20. Мова програмування PHP [Електронний ресурс] // Веб-сайт PHP. – Режим доступу: https://www.php.net/

21. Фреймворк Symfony [Електронний ресурс] // Веб-сайт Symfony. – Режим доступу: https://symfony.com/

22. Система повнотекстового пошуку ElasticSearch [Електронний ресурс] // Веб-сайт ElasticSearch. – Режим доступу: https://www.elastic.co/

23. Високопродуктивна база даних ключ-значення Redis [Електронний ресурс] // Веб-сайт Redis. – Режим доступу: https://redis.io/

24. База даних MySQL [Електронний ресурс] // Веб-сайт MySQL. – Режим доступу: https://www.mysql.com/

25. База даних PostgreSQL [Електронний ресурс] // Веб-сайт PostgreSQL. – Режим доступу: https://www.postgresql.org/

26. Розподілена платформа стірімінгу повідомлень Apache Kafka [Електронний ресурс] // Веб-сайт Kafka. – Режим доступу: https://kafka.apache.org/

27. Брокер повідомлень RabbitMQ [Електронний ресурс] // Веб-сайт RabbitMQ. – Режим доступу: https://www.rabbitmq.com/

28. Сервіс черги повідомлень Amazon Simple Queue Service [Електронний ресурс] // Лендінг сторінка Amazon AWS SQS. – Режим доступу: https://aws.amazon.com/sqs/

29. Dependency Injection Container фреймворку Symfony [Електронний ресурс] // Документація Symfony. – Режим доступу: https://symfony.com/doc/current/service_container.html

30. Розподілена система контролю версій GIT [Електронний ресурс] // Веб-сайт git. – Режим доступу: https://www.git-scm.com/

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проєкту розробки інтернет-магазину для продажу молочної продукції дистриб'юторам з функціоналом повного циклу оформлення оптових замовлень. Технічне завдання розроблено у відповідності до стандарту ГОСТ 19.201–78.

1 Підстава для розробки

Підставою для розробки є «Завдання на кваліфікаційну роботу», затверджене завідувачем кафедри інженерії програмного забезпечення.

Найменування розробки: «Інтернет-магазин з продажу молочної продукції дистриб’юторам».

2 Призначення розробки

2.1 Функціональне призначення

Додаток призначений для використання в якості інтернет-магазину для оптового продажу молочної продукції дистриб’юторам.

Передбачається повний цикл створення замовлень, включаючи авторизацію, пошук товарів, управління списком покупок, оформлення замовлення та оплата платіжною системою.

Користувачами системи є підприємці та компанії B2B сектору, що здійснюють оптові закупівлі молочної продукції.

2.2 Експлуатаційне призначення

Інтернет-магазин для дистрибуції молочної продукції призначений для здійснення оптових закупівель товарів, організації та обробки замовлень.

Користувачам надається змога здійснювати пошук товарів, управління станом списку покупок, оформлення та оплата замовлень.

Адміністратори можуть реєструвати нових користувачів, управляти каталогом продуктів та ціновими списками з можливістю встановлення промо-пропозицій.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Програмна система має забезпечувати наступні функції:

Можливість створювати профілі оптових клієнтів. Реєструвати користувачів в систему можуть лише адміністратори, тобто покупці не можуть власноруч створити свій профіль. Менеджери з продажу повинні мати можливість створювати та редагувати профілі оптових покупців та інформацію про них.

Управління каталогом продукції. Платформа повинна дозволяти групування товарів по категоріях для зручного управління асортиментом.

Повнотекстовий пошук товарів. Система повинна надавати можливість повнотекстового пошуку товарів за ключовими словами з опису, що дозволить користувачам швидко знаходити потрібні продукти, покращуючи рівень їх задоволеності та підвищуючи ефективність роботи.

Механізм промо-цін. Адміністратори повинні мати змогу налаштовувати спеціальні цінові пропозиції для обраних клієнтів або груп клієнтів, гарантуючи можливість персонального підходу до кожного покупця. Функціонал промо-цін дозволить проводити акції та пропонувати знижки, що стимулюватиме ріст продаж.

Екологічне оподаткування. Система повинна надавати можливість задати суму екологічного податку для продуктів, виробництво яких може негативно впливати на навколишнє середовище. Дана сума повинна враховуватись в фінальній ціні оплати при оформленні замовлення користувачем.

Функціонал оформлення замовлень. Користувачі повинні мати можливість наповнити кошик продкутами, вказавши кількість відповідних товарів, і перейти до оформлення замовлення продуктів даного кошка. Процес замовлення включає вибір способу оплати, вказання адреси доставки та адреси платника.

3.2 Вимоги до надійності

Інтернет-магазин повинен відповідати наступним вимогам надійності:

– забезпечення захисту від несанкціонованого доступу;
– розмежування прав адміністраторів від звичайних користувачів;
– використання алгоритмів хешування паролів;
– валідація введених даних;
– безперебійна робота системи з мінімальним часом простоювання.

3.3 Умови експлуатації та вимоги до технічних засобів

Система може використовуватися на будь-якому пристрої з доступом до інтернету та веб-браузером. Додаткових налаштувань для початку роботи застосунок не потребує. Для повного доступу до фукнцій системи достатньо зайти на сайт та авторизуватись.

3.4 Вимоги до інформаційної та програмної сумісності

Розроблена система може бути розгорнути на сервері з операційною системою Linux Ubuntu 22.04 або новішою, Debian 12, або новішими версіями. Залежності пакетного менеджера composer потребують PHP версії 7.4. Версія PostgreSQL – 13, або вище.

4 Вимоги до програмної документації

У момент здачі проекту замовнику надається наступний набір документів:

– технічне завдання;
– опис програми;
– текст програми;
– діаграма розгортання;
– керівництво користувача.

5 Стадії розробки

Стадія та період	Етап	Зміст
1	2	3
Технічне завдання, січень	Обґрунтування необхідності розробки програми	Вибір теми дипломного проєкту, короткий опис ПЗ, вимоги до ПЗ, етапи розробки ПЗ
Ескізний проєкт, січень-лютий	Розробка ескізного проекту	Створення базової структури системи, вибір платформи та технологій
Технічний проєкт, лютий-березень	Розробка технічного проекту	Уточнення структури, вибір кінцевих технологій для розробки та проектування дизайну ПЗ
Робочий проєкт, квітень	Розробка програмного забезпечення	Програмна розробка ПЗ, створення додатку, що відповідає всім стандартам
Розробка документації, травень	Розробка документації для програмного забезпечення	Підготовка супровідної документації для проекту, створення ілюстративних матеріалів

1	2	3
Тестування, травень	Проведення тестування програмного забезпечення	Тестування окремих модулів і системи в цілому, усунення виявлених неузгоджень у програмному забезпеченні
Впровадження, червень	Підготовка і передача програми	Підготовка необхідної документації для затвердження роботи, включаючи відгуки та рецензії.
Захист, червень	Розробка документації	Захист КвР

6 Порядок контролю та приймання

Контроль і приймання програми здійснюються на основі розробленої методики випробувань. При цьому перевіряється коректність всіх функцій програми QA-спеціалістами та групою користувачів. Прийом ПЗ замовником здійснюється після успішного тестування та виправлення можливих дефектів.

ДОДАТОК Б

(обов'язковий)

ФРАГМЕНТ КОДУ ПРОГРАМНОЇ СИСТЕМИ

```
final class AppPricingBundle extends Bundle
{
}

trait BasePriceListTrait
{
    /** @ORM\Column(name="price_list_type", type="integer") */
    protected int $priceListType = PriceListType::DEFAULT;

    public function priceListType(): PriceListType
    {
        return PriceListType::fromInt($this->getPriceListType());
    }

    public function setPriceListTypeVO(PriceListType $listType): void
    {
        $this->setPriceListType($listType->intVal());
    }

    public function getPriceListType(): int
    {
        return $this->priceListType;
    }

    public function setPriceListType(int $priceListType): void
    {
        $this->priceListType = $priceListType;
    }
}


trait BaseProductPriceTrait
{
    /** @ORM\Column(name="price_type", type="integer") */
    protected int $priceType = PriceType::DEFAULT;

    public function getPriceType(): int
    {
        return $this->priceType;
    }

    public function setPriceType(int $priceType): void
    {
        $this->priceType = $priceType;
    }
}

final class CombinedPriceListWithoutPromoTreeHandler extends
AbstractPriceListTreeHandler
{
    private CombinedPriceListTreeHandler
$combinedPriceListTreeHandler;

    private CombinedPriceListRepositoryInterface
$combinedPricePromoRepository;

    public function
setCombinedPriceListTreeHandler(CombinedPriceListTreeHandler
$combinedPriceListTreeHandler): void
    {
        $this->combinedPriceListTreeHandler =
$combinedPriceListTreeHandler;
    }

    public function setCombinedPricePromoRepository(
        CombinedPriceListRepositoryInterface
$combinedPricePromoRepository
    ): void {
        $this->combinedPricePromoRepository =
$combinedPricePromoRepository;
    }

    protected function loadPriceListByCustomer(Customer $customer,
Website $website): ?BasePriceList
    {
        return $this->combinedPricePromoRepository
            ->getPriceListWithoutPromoPricesByCustomer($customer,
$website);
    }

    protected function loadPriceListByCustomerGroup(CustomerGroup
$customerGroup, Website $website): ?BasePriceList
    {
        return $this->combinedPricePromoRepository
            -
>getPriceListWithoutPromoPricesByCustomerGroup($customerGroup,
$website);
    }

    protected function getPriceListByWebsite(Website $website):
?BasePriceList
    {
        return $this->combinedPricePromoRepository
            ->getPriceListWithoutPromoPricesByWebsite($website);
    }

    /** @warning promo price lists are not supported on config level */
    protected function getPriceListFromConfig()
    {
        // fallback to original tree handler
        return $this->combinedPriceListTreeHandler-
>getPriceListFromConfig();
    }
}


final class CombinedProductPriceRepository extends
OroCombinedProductPriceRepository
{
    protected OroCombinedProductPriceRepository $priceRepository;

    public function
setDecoratedRepository(OroCombinedProductPriceRepository
$priceRepository): void
    {
        $this->priceRepository = $priceRepository;
    }

    protected function getPricesBatchQueryFromBuilder(QueryBuilder
$qb): Query
    {
        return parent::getPricesBatchQueryFromBuilder(
            $qb->leftJoin(ProductPrice::class, 'originPrice', Join::WITH,
'price.originPriceId = originPrice.id')
                ->addSelect('originPrice.refValue as ref_value')
                ->addSelect('price.priceType as price_type')
        );
    }

    protected function getPricesBatchQueryResult(Query $query)
```

```
    {
        $this->_em->getConfiguration()
            ->addCustomHydrationMode('ProductPriceDTOHydrator',
ProductPriceDTOHydrator::class);

        return parent::getPricesBatchQueryResult($query);
    }
 }



 final class PriceListRepositoryDecorator extends
OroPriceListRepository implements PriceListRepository,
PriceListToCustomerRelationQueryVisitor
 {
    public function findPendingResolvePriceLists(): iterable
    {
        return PriceListWithRelationQuery
            ::createFromEm($this->_em)
            ->addSelect('oro_price_list.*')
            ->addSelect(
                <<<'SQL'
                    (SELECT ARRAY_AGG(DISTINCT
oro_price_product.product_id)
                        FROM oro_price_product
                        WHERE oro_price_product.price_list_id =
oro_price_list.id
                          AND oro_price_product.import_session =
oro_price_list.importsession)
                SQL,
                'product_ids',
            )
            ->wherePendingResolvePrices()
            ->orderByPriceListType()
            ->buildQuery()
            -
>setHydrationMode(PriceListToResolvePricesHydrator::class)
            ->toIterable();
    }

    public function findPostImportEmptyPriceLists(): array
    {
        // please, note that it doesn't deal with sharding
        return $this
            ->createQueryBuilder('price_list')
            ->select('price_list')
            ->where('price_list.importSession IS NOT NULL')
            ->leftJoin('price_list.prices', 'prices')
            ->andWhere('prices.priceList IS NULL')
            ->getQuery()
            ->execute();
    }

    public function findPriceListRelation(
        Organization $organization,
        PriceListRelationToCustomerQuery $relationData,
        ?PriceListToCustomerRelationQueryVisitor $self = null
    ): ?PriceListToCustomerRelation {
        return $relationData->acceptFindRelationVisitor($organization,
$self ?? $this);
    }
 }

 final class ProductPriceRepositoryDecorator extends
OroProductPriceRepository implements
    OutdatedPricesVisitor
 {
    private OroProductPriceRepository $productPriceRepository;

    public function
setProductPriceRepository(OroProductPriceRepository
$productPriceRepository): void
    {
        $this->productPriceRepository = $productPriceRepository;
    }

    protected function extraSaveColumns(BaseProductPrice $price):
array
    {
        $extraSaveColumns = $this->productPriceRepository-
>extraSaveColumns($price);

        if ($price instanceof ProductPrice) {
            $extraSaveColumns += ['ref_value' => ':ref_value'];
        }

        return $extraSaveColumns;
    }

    /** @param QueryBuilder $qb */
    protected function setExtraSaveParameters($qb,
BaseProductPrice $price): void
    {
        if (!$price instanceof ProductPrice) {
            return;
        }

        $this->productPriceRepository->setExtraSaveParameters($qb,
$price);
        $qb->setParameter('ref_value', $price->getRawRefValue());
    }

    public function findOutdatedPrices(
        string $outdatedPricesQueryClass,
        Organization $organization
    ): iterable {
        if (!is_a($outdatedPricesQueryClass,
OutdatedPricesQuery::class, true)) {
            throw new \Exception("Bad class
$outdatedPricesQueryClass");
        }
        return
$outdatedPricesQueryClass::acceptOutdatedPricesVisitor(
            $this,
            $organization,
        );
    }

    private function getOutdatedPricesQueryBuilder(
        Organization $organization,
        PriceListType $listType,
        string $joinClass
    ): Query {
        return $this->createQueryBuilder('p')
            ->leftJoin('p.priceList', 'priceList')
            ->andWhere('priceList.organization = :organization')
            ->setParameter('organization', $organization)
            ->andWhere('priceList.priceListType = :priceListType')
            ->setParameter('priceListType', $listType->intVal())
            ->innerJoin($joinClass, 'priceListToCustomerRelation',
Join::WITH, 'priceListToCustomerRelation.priceList = priceList')
            ->getQuery();
    }
 }

 final class PriceListSystemConfigSubscriber implements
EventSubscriberInterface
 {
    private bool $isFormRequest = false;

    public function __construct(
        private ConfigManager $globalConfig,
        private PriceListConfigConverter $converter,
    ) {
    }

    public static function getSubscribedEvents(): array
    {
        return [
            ConfigSettingsUpdateEvent::FORM_PRESET =>
[['formPreSet', -100]],
            self::beforeConfigSave() => [['beforeSave', 100]],
        ];
    }

    private static function beforeConfigSave(): string
```

```
    {
        return sprintf(
            '%s.%s',
            ConfigSettingsUpdateEvent::BEFORE_SAVE,

OroConfiguration::getConfigKeyByName(OroConfiguration::DEFAULT
_PRICE_LISTS)
        );
    }

    /** Filter default price lists to show only by organization */
    public function formPreSet(ConfigSettingsUpdateEvent $event):
void
    {
        $settingKey = $this->getSettingsKey();
        $settings = $event->getSettings();
        $organizationId = $this->organizationAccessor-
>getOrganizationId();

        if (!is_array($settings) || !array_key_exists($settingKey,
$settings) || null === $organizationId) {
            return;
        }

        $settings[$settingKey]['value'] = array_filter(
            $settings[$settingKey]['value'],
            static fn (PriceListConfig $config) =>
self::configIsForOrganization($config, $organizationId)
        );

        $event->setSettings($settings);
        $this->isFormRequest = true;
    }

    public function beforeSave(ConfigSettingsUpdateEvent $event):
void
    {
        $settings = $event->getSettings();
        $organizationId = $this->organizationAccessor-
>getOrganizationId();

        if (!array_key_exists('value', $settings) || null === $organizationId
|| !$this->isFormRequest) {
            return;
        }

        $existingConfig = $this->retrieveExistingConfig();

        $configsOfNotCurrentOrganization = array_filter(
            $existingConfig,
            static fn (PriceListConfig $config) =>
!self::configIsForOrganization($config, $organizationId),
        );

        $settings['value'] =
array_merge($configsOfNotCurrentOrganization, $settings['value']);

        $event->setSettings($settings);
    }

    private function retrieveExistingConfig(): array
    {
        $configsFromStore = $this->globalConfig
            -
>get(OroConfiguration::getConfigKeyByName(OroConfiguration::DEFA
ULT_PRICE_LISTS));

        if (
            count($configsFromStore) > 0
            && $configsFromStore[array_key_first($configsFromStore)]
instanceof PriceListConfig
        ) {
            return $configsFromStore;
        }

        return $this->converter
            ->convertFromSaved($configsFromStore);
    }

    private static function configIsForOrganization(PriceListConfig
$config, int $organizationId): bool
    {
        return $config->getPriceList()->getOrganization() !== null
            && $config->getPriceList()->getOrganization()->getId() ===
$organizationId;
    }

    private function getSettingsKey(): string
    {
        return implode(
            ConfigManager::SECTION_VIEW_SEPARATOR,
            [
                OroPricingExtension::ALIAS,
                OroConfiguration::DEFAULT_PRICE_LISTS,
            ]
        );
    }
}


final class AssignPriceListsToCustomersPostImportProcessor
implements
    MessageProcessorInterface,
    TopicSubscriberInterface
{
    public function __construct(
        private PriceListForAllAssignmentService $assignmentService,
        private MessageProducerInterface $messageProducer,
    ) {
    }

    public static function getSubscribedTopics(): array
    {
        return [Topics::PRICES_EMPTY_LISTS_CLEARED];
    }

    public function process(MessageInterface $message): string
    {
        $this->assignmentService->reassignPriceLists();

        $this->messageProducer-
>send(Topics::PRICES_POST_ASSIGN_LISTS, []);

        foreach ($this->assignmentService->pullProcessed() as
$websiteId => $processedByWebsite) {
            foreach ($processedByWebsite as $customerId) {
                $this->messageProducer->send(
                    RebuildCombinedPriceListsTopic::getName(),
                    [
                        'website' => $websiteId,
                        'customer' => $customerId,
                    ],
                );
            }
        }

        return self::ACK;
    }
}


final class ResolveImportedPriceListsProcessor implements
MessageProcessorInterface, TopicSubscriberInterface
{
    /** @required */
    public PriceListBatchRecalculationService $service;


    public static function getSubscribedTopics(): array
    {
        return [Topics::PRICES_POST_ASSIGN_LISTS];
    }

    public function process(MessageInterface $message): string
    {
        $this->service->resolvePendingPriceListPrices();
```

```
        return self::ACK;
    }
}

class CreditCardPaymentMethodProvider extends
AbstractPaymentMethodProvider
{
    public function __construct(
        protected CreditCardPaymentConfigProviderInterface
$configProvider,
        protected UniCreditPaymentMethodFactoryInterface $factory,
    ) {
        parent::__construct();
    }

    protected function collectMethods()
    {
        $configs = $this->configProvider->getPaymentConfigs();
        foreach ($configs as $config) {
            $this->addPaymentMethod($config);
        }
    }

    protected function
addPaymentMethod(CreditCardPaymentConfigInterface $config): void
    {
        $this->addMethod(
            $config->getPaymentMethodIdentifier(),
            $this->factory->create($config)
        );
    }
}

class CreditCardPaymentMethodFactory implements
UniCreditPaymentMethodFactoryInterface
{
    public function __construct(
        protected GatewayFactory $gateway,
        protected RouterInterface $router,
        protected LocaleSettings $localeSettings,
    ) {
    }

    public function create(CreditCardPaymentConfigInterface $config):
UniCreditPaymentMethod
    {
        $client = $this->gateway->create($config->getSignatureKey(),
$config->getMerchantId(), $config->isTestMode(), $config-
>isDebugMode());

        return new UniCreditPaymentMethod($client, $config, $this-
>router, $this->localeSettings);
    }
}


class UniCreditCallbackListener
{
    use LoggerAwareTrait;

    protected PaymentMethodProviderInterface
$paymentMethodProvider;

    public function __construct(PaymentMethodProviderInterface
$paymentMethodProvider)
    {
        $this->paymentMethodProvider = $paymentMethodProvider;
    }

    public function onError(AbstractCallbackEvent $event): void
    {
        $paymentTransaction = $event->getPaymentTransaction();

        if (!$paymentTransaction) {
            return;
        }

        if (false === $this->paymentMethodProvider-
>hasPaymentMethod($paymentTransaction->getPaymentMethod())) {
            return;
        }

        $paymentTransaction
            ->setSuccessful(false)
            ->setActive(false);
    }

    public function onReturn(AbstractCallbackEvent $event): void
    {
        $paymentTransaction = $event->getPaymentTransaction();

        if (!$paymentTransaction) {
            return;
        }

        $paymentMethodId = $paymentTransaction-
>getPaymentMethod();

        if (false === $this->paymentMethodProvider-
>hasPaymentMethod($paymentMethodId)) {
            return;
        }

        try {
            $paymentMethod = $this->paymentMethodProvider-
>getPaymentMethod($paymentMethodId);
            $paymentMethod-
>execute(CreitCardPaymentMethod::COMPLETE,
$paymentTransaction);

            $event->markSuccessful();
        } catch (\InvalidArgumentException $e) {
            if ($this->logger) {
                // do not expose sensitive data in context
                $this->logger->error($e->getMessage(), []);
            }
        }
    }
}

class CreditCardPaymentConfigProvider implements
CreditCardPaymentConfigProviderInterface
{
    /**
     * @var CreditCardPaymentConfigInterface[]
     */
    protected array $configs = [];

    public function __construct(
        protected string $type,
        protected ManagerRegistry $doctrine,
        protected CreditCardPaymentConfigFactoryInterface $factory,
        protected LoggerInterface $logger,
    ) {
    }

    public function getPaymentConfigs(): array
    {
        if (0 === count($this->configs)) {
            return $this->configs = $this->collectConfigs();
        }

        return $this->configs;
    }

    public function getPaymentConfig(string $identifier):
?CreditCardPaymentConfigInterface
    {
        if ($this->hasPaymentConfig($identifier)) {
            $configs = $this->getPaymentConfigs();

            return $configs[$identifier];
        }

        return null;
```

```
    }

    public function hasPaymentConfig(string $identifier): bool
    {
        $configs = $this->getPaymentConfigs();

        return array_key_exists($identifier, $configs);
    }

    protected function getType(): string
    {
        return $this->type;
    }

    protected function getEnabledIntegrationSettings(): array
    {
        try {
            return $this->doctrine-
>getManagerForClass(UniCreditSettings::class)
                ->getRepository(UniCreditSettings::class)
                ->getEnabledSettingsByType($this->getType());
        } catch (\UnexpectedValueException $e) {
            $this->logger->critical($e->getMessage());

            return [];
        }
    }

    protected function collectConfigs(): array
    {
        $configs = [];
        $settings = $this->getEnabledIntegrationSettings();

        foreach ($settings as $setting) {
            $config = $this->factory->createConfig($setting);
            $configs[$config->getPaymentMethodIdentifier()] = $config;
        }

        return $configs;
    }
 }


 final class LineItemPriceProvider
 {
    private ProductTaxesProvider $productTaxesProvider;

    public function __construct(ProductTaxesProvider
$productTaxesProvider)
    {
        $this->productTaxesProvider = $productTaxesProvider;
    }

    public function getLineItemPrice(PriceAwareInterface $lineItem,
Product $product): BigDecimal
    {
        return $this->getPriceWithTaxes(
            $lineItem->getPrice(),
            $product
        );
    }

    public function getPriceWithTaxes(Price $price, Product $product):
BigDecimal
    {
        return BigDecimal::of((float)$price->getValue())
            ->plus($this->productTaxesProvider->ecoTaxes($product));
    }

    public function addEcotaxToSubtotal(float $subtotal, Product
$product, float $qty): float
    {
        return $subtotal + $this->productTaxesProvider-
>ecoTaxes($product)->toFloat() * $qty;
    }
 }
```

```
 class EcoTaxFrontendDataGridsListener
 {
    public function __construct(private ConfigManager
$configManager,)
    {
    }

    public function checkoutLineItemsOnBeforeBuild(BeforeBuild
$event): void
    {
        if (!$this->configManager-
>get(Configuration::ENABLE_ECO_TAXES())) {
            return;
        }

        $config = $event->getConfig();
        $config->offsetAddToArrayByPath('[source][query][select]',
['product.ecoTaxAmount as ecoTax']);
        $config->offsetAddToArrayByPath(
            '[columns][subtotal]',
            ['order' => 100],
        );
        $config->offsetSetByPath(
            '[columns][ecoTax]',
            [
                'label' =>
'app.frontend.eco_tax.checkout.line_items.grid.column.label',
                'type' => 'twig',
                'template' =>
'AppTaxBundle:Taxes:Datagrid/format_eco_tax_grid_column.html.twig'
,
                'order' => 90,
            ]
        );
    }

    public function userShoppingListOnBeforeBuild(BeforeBuild
$event): void
    {
        if (!$this->configManager-
>get(Configuration::ENABLE_ECO_TAXES())) {
            return;
        }

        $config = $event->getConfig();
        $config->offsetAddToArrayByPath('[source][query][select]',
['product.ecoTaxAmount as ecoTax']);
        $config->offsetAddToArrayByPath('[columns][subtotal]', ['order'
=> 100]);
        $config->offsetSetByPath(
            '[columns][ecoTax]',
            [
                'label' =>
'app.frontend.eco_tax.shopping_list.grid.column.label',
                'type' => 'twig',
                'template' =>
'AppTaxBundle:Taxes:Datagrid/format_eco_tax_grid_column.html.twig'
,
                'order' => 90,
            ]
        );
    }
 }

 class LineItemSubtotalProvider extends
BaseLineItemSubtotalProvider
 {
    private LineItemPriceProvider $lineItemPriceProvider;

    public function setLineItemPriceProvider(LineItemPriceProvider
$lineItemPriceProvider): void
    {
        $this->lineItemPriceProvider = $lineItemPriceProvider;
    }

    /**
```

```
     * @param
PriceAwareInterface&PriceTypeAwareInterface&QuantityAwareInterfac
e $lineItem
     * @param $baseCurrency
     *
     * @return float|int
     */
    public function getRowTotal(PriceAwareInterface $lineItem,
$baseCurrency)
    {
        if (!method_exists($lineItem, 'getProduct')) {
            return parent::getRowTotal($lineItem, $baseCurrency);
        }

        if (null === $lineItem->getPrice() || null === $lineItem-
>getProduct()) {
            return parent::getRowTotal($lineItem, $baseCurrency);
        }

        $valueIncludingTaxes = $this->lineItemPriceProvider
            ->getLineItemPrice($lineItem, $lineItem->getProduct());

        return parent::getRowTotal(
            LineItemDTO::with(
                (clone $lineItem->getPrice())
                    ->setValue($valueIncludingTaxes->toFloat()),
                $lineItem->getQuantity(),
                $lineItem->getPriceType(),
            ),
            $baseCurrency
        );
    }
}


class ProductSearchController extends AbstractController
{
    private const GRID_NAME = 'frontend-product-search-grid';

    public function __construct(
        private DataGridThemeHelper $gridThemeHelper,
        private DataGridExtension $dataGridTwigExtension,
    ) {
    }

    /**
     * Search products
     *
     * @Route("/search", name="product_frontend_product_search")
     * @Layout(vars={"entity_class", "theme_name", "grid_config",
"no_result"})
     *
     * @return array
     */
    public function searchAction()
    {
        // Directly use twig extension to call same method that is called
in twig templates via 'oro_datagrid_build'
        $grid = $this->dataGridTwigExtension-
>getGrid(self::GRID_NAME);
        return [
            'entity_class' => Product::class,
            'theme_name' => $this->gridThemeHelper
                ->getTheme(self::GRID_NAME),
            'grid_config' => [
                self::GRID_NAME
            ],
            'no_result' => !$grid->getData()->getTotalRecords(),
            'data' => [
                'datagrid' => $grid
            ]
        ];
    }
}

final class DeliverySlotController extends AbstractController
{
/**
 * @Route("/", name="app_delivery_slot_index")
 * @Template()
 * @AclAncestor("app_delivery_slot_view")
 */
public function indexAction(): array
{
    return ['entity_class' => DeliverySlot::class];
}

/**
 * @Route("/view/{id}", name="app_delivery_slot_view")
 * @Template()
 * @Acl(id="app_delivery_slot_view", type="entity",
class="AppDeliverySlotsBundle:DeliverySlot", permission="VIEW")
 */
public function viewAction(DeliverySlot $deliverySlot): array
{
    return ['entity' => $deliverySlot];
}

/**
 * @Route("/create", name="app_delivery_slot_create",
options={"expose"=true})
 *
@Template("AppDeliverySlotsBundle:DeliverySlot:update.html.twig")
 * @Acl(id="app_delivery_slot_create", type="entity",
class="AppDeliverySlotsBundle:DeliverySlot", permission="CREATE")
 */
public function createAction()
{
    return $this->update(new DeliverySlot());
}

/**
 * @Route("/update/{id}", name="app_delivery_slot_update")
 * @Template()
 * @Acl(id="app_delivery_slot_update", type="entity",
class="AppDeliverySlotsBundle:DeliverySlot", permission="EDIT")
 */
public function updateAction(DeliverySlot $deliverySlot)
{
    return $this->update($deliverySlot);
}

protected function update(DeliverySlot $deliverySlot)
{
    $form = $this->createForm(DeliverySlotType::class, $deliverySlot);
    $savedMessage = $this->get('translator')-
>trans('app.deliveryslot.controller.deliveryslot.saved.message');

    return $this->get('oro_form.update_handler')-
>update($deliverySlot, $form, $savedMessage);
}

/**
 * @Route("/widget/info/{id}", name="app_delivery_slot_widget_info")
 *
@Template("AppDeliverySlotsBundle:DeliverySlot/widget.html.twig")
 * @AclAncestor("app_delivery_slot_view")
 */
public function infoAction(DeliverySlot $entity): array
{
    return ['entity' => $entity];
}

/** @covers NearestDeliverySlotProvider */
final class NearestDeliverySlotProviderTest extends TestCase
{
    private NearestDeliverySlotProvider $provider;

    private WorkingDaysProviderInterface&MockObject
$workingDaysProvider;

    private LocalizedCarbonFactory&MockObject
$localizedCarbonFactory;

    protected function setUp(): void
    {
```

```
        parent::setUp();

        $this->workingDaysProvider = $this->createMock(WorkingDaysProviderInterface::class);
        $localeSettings = $this->createMock(LocaleSettings::class);
        $localeSettings->method('getTimeZone')->willReturn(date_default_timezone_get());
        $this->localizedCarbonFactory = new LocalizedCarbonFactory($localeSettings);
        $configProvider = $this->createMock(ConfigProvider::class);
        $configProvider->method('getDisplayedWeeks')->willReturn(3);
        $deliverySlotWrapperFactory = new DeliverySlotsWrapperFactory($this->localizedCarbonFactory);
        $this->provider = new NearestDeliverySlotProvider($this->workingDaysProvider, $this->localizedCarbonFactory, $deliverySlotWrapperFactory, $configProvider);
    }

    public function testThrowsExceptionIfNoSlotsGiven(): void
    {
        $this->expectException(NoResultException::class);

        $this->provider->getNearestDeliverySlot([], 'FR');
    }

    public function testSelectsFirstAvailableSlotForUser(): void
    {
        // dayOfWeek 4 (Thursday)
        CarbonImmutable::setTestNow('2021-12-09 09:00:00');

        $slotExpired = $this->slot(
            4, // Thursday, today
            '17:00',
            '18:00',
            4,  // Thursday, today
            '08:59',
        );

        $slotToBeUsed = $this->slot(
            6, // Saturday
            '09:00',
            '10:00',
            4, // Today, Thursday
            '18:00',
        );

        $nextAfterUsed = $this->slot(
            5, // Friday
            '08:00',
            '08:30',
            4, // Today, Thursday
            '18:01' // one minute after slot to be used
        );
        $this->workingDaysProvider->method('isWorkingDay')
            ->willReturnCallback(
                static function (DateTimeInterface $date, string $code) {
                    $format = $date->format('Y-m-d');
                    return $code === 'FR' && ('2021-12-09' === $format || '2021-12-10' === $format ||
                        '2021-12-11' === $format || '2021-12-16' === $format); // for $slotExpired next week
                }
            );

        $userSlot = $this->provider->getNearestDeliverySlot([$slotExpired, $slotToBeUsed, $nextAfterUsed], 'FR');

        self::assertFalse($userSlot->isExpired($this->localizedCarbonFactory->now()));
        self::assertSame('2021-12-11', $userSlot->getDeliveryDate()->format('Y-m-d'));
        self::assertSame('2021-12-09 18:00:00', $userSlot->getCutoffDateTime()->format('Y-m-d H:i:s'));
        self::assertSame('2021-12-11 09:00:00', $userSlot->getStartDateTime()->format('Y-m-d H:i:s'));
        self::assertSame('2021-12-11 10:00:00', $userSlot->getEndDateTime()->format('Y-m-d H:i:s'));
    }

    public function testTriesNextWeekDayIfHoliday(): void
    {
        // dayOfWeek 4 (Thursday)
        CarbonImmutable::setTestNow('2021-12-09 09:00:00');

        $slot = $this->slot(
            5, // Friday
            '09:30',
            '09:45',
            4, // Today, Thursday
            '19:00',
        );

        $this->workingDaysProvider->method('isWorkingDay')
            ->willReturnCallback(
                static function (DateTimeInterface $date, string $code) {
                    $format = $date->format('Y-m-d');

                    return $code === 'FR' && '2021-12-10' !== $format
                        && ('2021-12-09' === $format || '2021-12-17' === $format);
                }
            );

        $userSlot = $this->provider->getNearestDeliverySlot([$slot], 'FR');

        self::assertFalse($userSlot->isExpired($this->localizedCarbonFactory->now()));
        self::assertSame('2021-12-17', $userSlot->getDeliveryDate()->format('Y-m-d'));
        self::assertSame('2021-12-16 19:00:00', $userSlot->getCutoffDateTime()->format('Y-m-d H:i:s'));
        self::assertSame('2021-12-17 09:30:00', $userSlot->getStartDateTime()->format('Y-m-d H:i:s'));
        self::assertSame('2021-12-17 09:45:00', $userSlot->getEndDateTime()->format('Y-m-d H:i:s'));
    }

    private function slot($day, $begin, $end, $cutoffDay, $cutoffTime): DeliverySlot
    {
        $slot = new DeliverySlot();
        $slot->setDay($day);
        $slot->getDeliveryTimeRange()->setBegin(CarbonImmutable::createFromTimeString($begin));
        $slot->getDeliveryTimeRange()->setEnd(CarbonImmutable::createFromTimeString($end));
        $slot->setCutoffDay($cutoffDay);
        $slot->setCutoffTime(CarbonImmutable::createFromTimeString($cutoffTime));
        return $slot;
    }
}

final class PriceListCollectionProviderDecorator extends PriceListCollectionProvider
{
    /** @required */
    public PriceListCollectionProvider $priceListCollectionProvider;

    public function getPriceListsByConfig(): array
    {
        return $this->sortPriceListsByPromoFirst(
            $this->priceListCollectionProvider->getPriceListsByConfig());
    }

    public function getPriceListsByWebsite(Website $website): array
    {
        return $this->sortPriceListsByPromoFirst(
            $this->priceListCollectionProvider->getPriceListsByWebsite($website));
    }
```

```
    public function getPriceListsByCustomerGroup(CustomerGroup
$customerGroup, Website $website): array
    {
        return $this->sortPriceListsByPromoFirst(
 $this->priceListCollectionProvider-
>getPriceListsByCustomerGroup($customerGroup, $website));
    }

    public function getPriceListsByCustomer(Customer $customer,
Website $website): array
    {
        return $this->sortPriceListsByPromoFirst(
 $this->priceListCollectionProvider-
>getPriceListsByCustomer($customer, $website));
    }

    private function sortPriceListsByPromoFirst(array
$priceListMembers): array
    {
        $promoMembers = array_filter(
            $priceListMembers,
            static fn(PriceListSequenceMember $member) => $member-
>getPriceList()
 ->priceListType()->isPromo()
        );
        $notPromoMembers = array_diff_key($priceListMembers,
$promoMembers);

        return array_merge($promoMembers, $notPromoMembers);
    }

    public function containMergeDisallowed(array $collection): bool
    {
        return $this->priceListCollectionProvider-
>containMergeDisallowed($collection);
    }

    public function containScheduled(array $collection): bool
    {
        return $this->priceListCollectionProvider-
>containScheduled($collection);
    }
 }

 final class AppWebsiteSearchExtension extends Extension
 {
  public const ALIAS = 'App_website_search';

  public function load(array $configs, ContainerBuilder $container)
  {
   $configuration = new Configuration();
   $config = $this->processConfiguration($configuration, $configs);
   $loader = new YamlFileLoader($container, new
FileLocator(__DIR__ . '/../Resources/config'));

   $loader->load('services.yml');
   $container->prependExtensionConfig($this->getAlias(),
array_intersect_key($config, array_flip(['settings'])));
  }

  public function getAlias()
  {
   return self::ALIAS;
  }
 }


 class WebsiteSearchProductIndexerListener
 {
    public function __construct(
        private WebsiteContextManager $websiteContextManager,
        private ManagerRegistry $registry,
    ) {
    }

    public function onWebsiteSearchIndex(IndexEntityEvent $event)
    {
        $website = $this->getWebsite($event);
        if (!$website) {
            $event->stopPropagation();

            return;
        }

        /** @var Product[] $products */
        $products = $event->getEntities();

        foreach ($products as $product) {
            $event->addField($product->getId(),
PriceUnitLabel::ELASTIC_FIELD_NAME, $product-
>getPriceUnitLabel());
            $event->addField($product->getId(),
InformationUnitOfOrder::ELASTIC_FIELD_NAME, $product-
>getInformationUnitOfOrder());
            $event->addField($product->getId(),
MainPreOrderProduct::ELASTIC_FIELD_NAME, (int)$product-
>getMainArticleEnPrecommande());
        }
    }

    private function getWebsite(IndexEntityEvent $event): ?Website
    {
        $websiteId = $this->websiteContextManager-
>getWebsiteId($event->getContext());
        if (!$websiteId) {
            return null;
        }

        return $this->registry->getManagerForClass(Website::class)-
>find(Website::class, $websiteId);

    }
 }


 final class WebsiteSearchProductPriceTypeIndexerListener
implements FeatureToggleableInterface
 {
    use FeatureCheckerHolderTrait;

    public function __construct(
        private WebsiteContextManager $websiteContextManger,
        private ManagerRegistry $doctrine,
        private ConfigManager $configManager,
    ) {
    }

    public function onWebsiteSearchIndex(IndexEntityEvent $event):
void
    {
        if (!$this->isFeaturesEnabled()) {
            return;
        }

        $websiteId = $this->websiteContextManger-
>getWebsiteId($event->getContext());

        if (!$websiteId) {
            $event->stopPropagation();

            return;
        }

        /** @var CombinedProductPriceRepository $repository */
        $repository = $this->doctrine-
>getManagerForClass(CombinedProductPrice::class)-
>getRepository(CombinedProductPrice::class);

        $configCpl = $this->configManager-
>get(Configuration::getConfigKeyToPriceList());

        $prices = $repository->findMinByWebsiteForSort($websiteId,
$event->getEntities(), $configCpl);

        foreach ($prices as $price) {
            Assert::keyExists($price, 'type');
```

```
            $event->addPlaceholderField(
                $price['product'],
                PromoFilter::SEARCH_FIELD,
                (int)$price['type'],
                [
                    CplIdPlaceholder::NAME => $price['cpl'],
                    CurrencyPlaceholder::NAME => $price['currency'],
                ]
            );
        }
    }
}
```

```
services:
    App\Bundle\PricingBundle\Provider\:
        resource: '../../Provider/*'
        autowire: true

    App\Bundle\PricingBundle\Service\PriceListRelationService:
        alias: app_pricing.price_list.relation_service

    app_pricing.price_list.relation_service:
        class:
App\Bundle\PricingBundle\Service\PriceListRelationService
        arguments:
            $tokenAccessor: '@oro_security.token_accessor'
            $currencyProvider: '@oro_currency.config.currency'

    app_pricing.price_list_repository.cache_listener:
        class:
App\Bundle\PricingBundle\EventListener\PriceListRepositoryCacheCle
arListener
        arguments: [ '@app_pricing.repository.price_list.memory_cache'
]
        tags:
            - { name: app_import.doctrine_subscriber }


App\Bundle\PricingBundle\Model\ProductDBVisbilityByHavingPriceMo
difier:
        arguments:
            - '@oro_pricing.cpl.placeholder'


App\Bundle\PricingBundle\DataGrid\Extension\PromoFilterCountsExte
nsion:
        autowire: true
        arguments:
            $searchQueryProvider:
'@oro_product_pro.datagrid.provider.search_query'
            $filtersStateProvider: '@oro_filter.provider.state.filters'
            $searchEngine: '%oro_website_search.engine%'
        calls:
            - [ addApplicableGrid, [ 'frontend-product-search-grid' ] ]
            - [ addApplicableGrid, [ 'frontend-catalog-allproducts-grid' ] ]
            - [ addApplicableGrid, [ 'order-products-previously-purchased-
grid' ] ]
        tags:
            - { name: oro_datagrid.extension, priority: 250 }


App\Bundle\PricingBundle\Search\PromoFilterCountsProviderInterface
:
        class:
App\Bundle\PricingBundle\Search\PromoCalculationRepository
        parent: oro_website_search.repository.abstract


App\Bundle\PricingBundle\Search\PromoFilterCountsProviderCaching
Decorator:
        decorates:
App\Bundle\PricingBundle\Search\PromoFilterCountsProviderInterface
        arguments:
            -
'@App\Bundle\PricingBundle\Search\PromoFilterCountsProviderCachi
ngDecorator.inner'
            - '@oro_product_pro.search.raw_query_string_provider'
            - '@app_pricing.cache.promo_counts'
    app_pricing.cache.promo_counts:
        parent: oro.cache.abstract
        public: true
        calls:
            - [ setNamespace, [ 'app_pricing_promo_counts' ] ]


App\Bundle\PricingBundle\EventSubscriber\PriceListSystemConfigSub
scriber:
        autoconfigure: true
        autowire: true
        arguments:
            $globalConfig: '@oro_config.global'
            $priceListConfigConverter:
'@oro_pricing.system_config_converter'

    app_pricing.formatter.product_price_formatter:
        class:
App\Bundle\PricingBundle\Formatter\ProductPriceFormatter
        parent: oro_pricing.formatter.product_price_formatter
        decorates: oro_pricing.formatter.product_price_formatter
        calls:
            - [ setPriceConfigsProvider, [
'@App\Bundle\PricingBundle\Provider\PriceConfigsProvider' ] ]


App\Bundle\PricingBundle\Service\Assignment\PriceListForAllAssignm
entService:
        autowire: true
        arguments:
            $customerRepositoryService:
'@app_pricing.repository.service.customer'


App\Bundle\PricingBundle\Service\Assignment\PriceListBatchAssignm
entService:
        autowire: true
        arguments: [ '@doctrine' ]


App\Bundle\PricingBundle\Service\Assignment\PriceListAssignmentSe
rvice:
        arguments:
            - '@oro_pricing.price_list_with_priority_collection.handler'

    App\Bundle\PricingBundle\Provider\ProductOldPriceProvider:
        arguments:
            - '@oro_pricing.storage.prices'
            - '@oro_pricing.user_currency_manager'
        calls:
            - [ setMemoryCacheProvider, [
'@oro_pricing.provider.product_price.cache' ] ]

    app_pricing.provider.frontend_product_old_prices:
        class:
'Oro\Bundle\PricingBundle\Provider\FrontendProductPricesDataProvid
er'
        arguments:
            -
'@App\Bundle\PricingBundle\Provider\ProductOldPriceProvider'
            - '@oro_pricing.user_currency_manager'
            -
'@oro_pricing.model.product_price_scope_criteria_request_handler'
```

```
 final class LoadCustomerUsersAppDemoData extends
AbstractLoadCustomerUserAppDemoData implements
VersionedFixtureInterface, LoadedFixtureVersionAwareInterface
 {
    use LoadedFixtureVersionAwareTrait;

    public const USERS_PATH =
'@AppCustomerBundle/Migrations/Data/App/ORM/data/customer-
users.csv';
    public const VERSION = '1.1';

    public function getVersion(): string
```

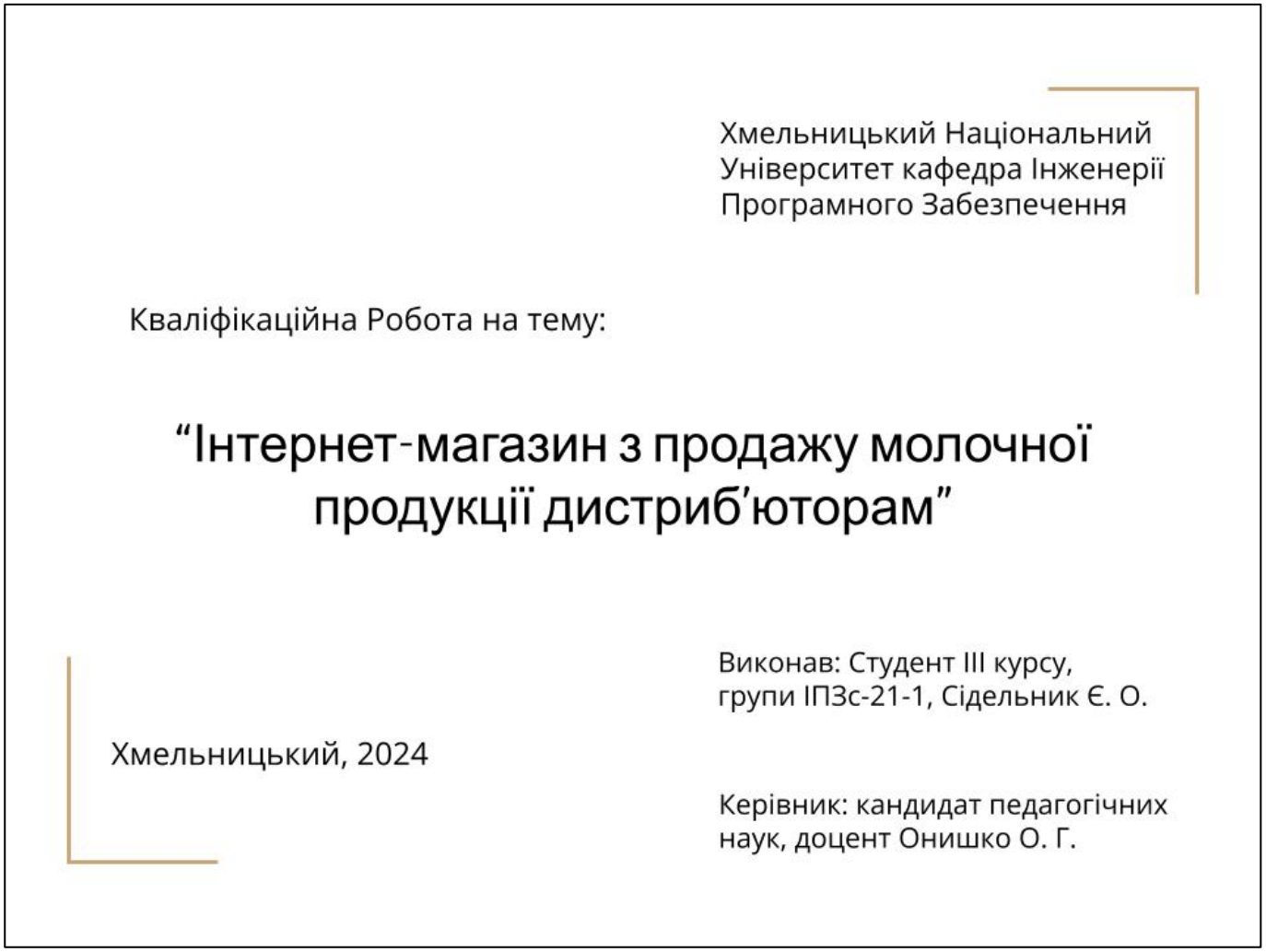
```
    {
        return self::VERSION;
    }

    public function getDependencies(): array
    {
        return [LoadCustomerAppDemoData::class];
    }

    protected function getCustomerUsersCSV(): string
    {
        return self::USERS_PATH;
    }

    protected function getCustomerUserRole($roleLabel,
ObjectManager $manager, array $row)
    {
        return $this->container->get('doctrine')
                        ->getManagerForClass(CustomerUserRole::class)
            ->getRepository(CustomerUserRole::class)-
>findOneBy(['label' => $roleLabel, 'organization' => $this-
>getOrganization($manager, $row)]);
    }

    protected function getCustomer(ObjectManager $manager, array
$row)
    {
        $ref = LoadCustomerAppDemoData::customerReference(
  $row['organization'], $row['customer']);
        return $this->hasReference($ref) ? $this->getReference($ref) :
                        $manager->getRepository(Customer::class)
                        ->findOneBy(['organization' => $row['organization'],
'name' => $row['customer']]);
    }

    protected function getOrganization(ObjectManager $manager,
array $row)
    {
        $findOrganization = new
FindOrganizationReferenceByInternalName($this-
>referenceRepository);

        return $findOrganization($row['organization']);
    }

    protected function getWebsite(ObjectManager $manager, array
$row)
    {
        return $manager->getRepository(Website::class)->findOneBy([
              'organization' => $this->getOrganization($manager, $row),
              'name' => LoadWebsiteData::DEFAULT_WEBSITE_NAME,
            ]);
    }

    protected function shouldProcessRow(array $row): bool
    {
        $version = $row['version'] ?? '0.0';

        return !$this->versionIsLoaded($version);
    }
 }


  final class LoadProductDemoData extends AbstractFixture
implements
    ContainerAwareInterface,
    DependentFixtureInterface
 {
    use UserUtilityTrait;

    private const ENUM_CODE_INVENTORY_STATUS =
'prod_inventory_status';

    public const FILE_PATH =
'@AppProductBundle/Migrations/Data/App/ORM/data/products.csv';
    private const IMAGES_PATH =
'@AppProductBundle/Migrations/Data/App/ORM/data/images';
    private const RESIZED_PATH =
'@OroProductBundle/Migrations/Data/Demo/ORM/images/resized';

    private const FAMILY_CODE =
LoadProductDefaultAttributeFamilyData::DEFAULT_FAMILY_CODE;

    private array $productUnits = [];

    / @var array|Filesystem[] */
    private $filesystems = [];

    private ContainerInterface $container;
    private FileLocator $fileLocator;

    public function setContainer(?ContainerInterface $container = null):
void
    {
        $this->container = $container;
        $this->fileLocator = $container->get('file_locator');
    }

    public function getDependencies(): array
    {
        return [
            LoadBrandAppDemoData::class,
        ];
    }

    public function load(ObjectManager $manager): void
    {
        $allImageTypes = $this->getImageTypes();

        $this->container->get('oro_layout.loader.image_filter')->load();

        $slugGenerator = $this->container-
>get('oro_entity_config.slug.generator');

        $filePath = $this->fileLocator->locate(self::FILE_PATH);
        $rawProducts = new CsvIterator($filePath);

        $loadedProducts = [];
        foreach ($rawProducts as $rawProduct) {
            $product = new Product();

            $this->loadData($product, $rawProduct, $manager);

            $slugPrototype = new LocalizedFallbackValue();
            $slugPrototype->setString($slugGenerator-
>slugify($rawProduct['name']));
            $product->addSlugPrototype($slugPrototype);

            $productUnit = $this->getProductUnit($manager,
ProductDefaultValue::UNIT_CODE);
            $productUnitPrecision = new ProductUnitPrecision();
            $productUnitPrecision
                ->setProduct($product)
                ->setUnit($productUnit)
                ->setPrecision(ProductDefaultValue::UNIT_PRECISION)
                ->setConversionRate(1)
                ->setSell(true);
            $product->setPrimaryUnitPrecision($productUnitPrecision);

            $this->addImageToProduct($product, $manager,
$rawProduct['sku'], $allImageTypes);

            $manager->persist($product);
            $loadedProducts[] = $product;
        }

        $manager->flush();

        $this->createSlugs($loadedProducts, $manager);
    }

    private function loadData(Product $product, array $row,
ObjectManager $manager): void
    {
```

```
        $findOrganization = new
FindOrganizationReferenceByInternalName($this-
>referenceRepository);
        $findAttributeFamily = new FindAttributeFamilyReference($this-
>referenceRepository);
        $groupedCategories = $this-
>categoriesGroupedByOrganizations($manager);

        $inStockStatus = $this->getInventoryStatus($manager,
Product::INVENTORY_STATUS_IN_STOCK);
        $organization = $findOrganization($row['organization']);
        $attributeFamily = $findAttributeFamily($organization,
self::FAMILY_CODE);
        $businessUnit = $organization->getBusinessUnits()->first();

        $categories = $groupedCategories[$organization->getId()] ?? [];

        $name = new ProductName();
        $name->setString($row['name']);
        $isolateScopeId = \uniqid();
        $text = nl2br('<div id="isolation-scope-' . $isolateScopeId . '">' .
$row['description'] . '</div>');

        / @noinspection PhpUnhandledExceptionInspection */
        $shouldSetCategory = 85 >= random_int(1, 100);
        $lctlsPriceUnitClass =
ExtendHelper::buildEnumValueClassName('prod_lctls_price_unit');
        $product->addManualDescription((new
LocalizedFallbackValue())
            ->setWysiwyg($text)->setWysiwygProperties(['codeMode' =>
false, "isolateScopeId" => $isolateScopeId]));
        $product->setOwner($businessUnit)


  ->setOrganization($organization)
            ->setAttributeFamily($attributeFamily)
            ->setSku($row['sku'])
            ->setInventoryStatus($inStockStatus)
            ->setStatus(Product::STATUS_ENABLED)
            ->addName($name)
            ->addDescription(new ProductDescription())
            ->addShortDescription(new ProductShortDescription())
            ->setType('simple')
            ->setFeatured($row['featured'])
            ->setLctlsPriceUnitLabel($row['price_unit_label'])
            -
>setLctlsInformationUnitOfOrder($row['information_unit_of_order'])
            ->setNewArrival($row['new_arrival'])
            ->setLctlsPriceUnit($manager->find($lctlsPriceUnitClass,
strtolower($row['price_unit'])))
            ->setCategory($shouldSetCategory ?
$categories[array_rand($categories)] : null)
            ->setTaxEcoparticipation($row['tax_ecoparticipation'] ?: 0);
        $taxCodes = $this->getTaxCodes($manager, $organization);
        $product->setTaxCode($taxCodes[array_rand($taxCodes)]);
        $this->setOrganizationRelatedData($product, $row, $manager,
$organization);
        if (
            $row['brand']
            && $this->hasReference(
                $brandRef =
LoadBrandAppDemoData::brandReference($row['organization'],
$row['brand'])
            )
        ) {
            / @noinspection PhpParamsInspection */
            $product->setBrand($this->getReference($brandRef));
        }
    }

    private function setOrganizationRelatedData(
        Product $product,
        array $row,
        ObjectManager $manager,
        Organization $organization
    ): void {
        if (OrganizationInternalName::MAIN()-
>correspondsToOrganization($organization)) {
            $product-
>setMainArticleEnPrecommande($row['is_preorder']);
            $product-
>setMainDelaiPrecommandeNbreDeJours($row['preorder_days']);
            $product->setMainTypeDeProduit((int)$row['type']);
            $product->setMainProduitAop(65 >= random_int(1, 100));
            $product->setMainProduitBio(65 >= random_int(1, 100));
            $product->setMainProduitEnBretagne(65 >= random_int(1,
100));
            $orderUnits = $this->getOrderUnits($manager);
            $product-
>setMainUniteDeCommande($orderUnits[array_rand($orderUnits)]);
        }
        if (OrganizationInternalName::EXTRA()-
>correspondsToOrganization($organization)) {
            $product->setExtraSinLatte(70 >= random_int(1, 100));
            $product->setExtraSinGlutine(70 >= random_int(1, 100));
            $product->setExtraSinLattosio(70 >= random_int(1, 100));
            $product->setExtraSinSoia(70 >= random_int(1, 100));
            $product->setExtraSinUovo(70 >= random_int(1, 100));
        }
    }

    / @param Product[] $products */
    private function createSlugs(array $products, ObjectManager
$manager): void
    {
        / @var SlugEntityGenerator $slugRedirectGenerator */
        $slugRedirectGenerator = $this->container-
>get('oro_redirect.generator.slug_entity');

        foreach ($products as $product) {
            $slugRedirectGenerator->generate($product, true);
        }

        $cache = $this->container->get('oro_redirect.url_cache');
        if ($cache instanceof FlushableCache) {
            $cache->flushAll();
        }

        $manager->flush();
    }

    protected function getInventoryStatus(ObjectManager $manager,
$status): AbstractEnumValue
    {
        $inventoryStatusClassName =
ExtendHelper::buildEnumValueClassName(self::ENUM_CODE_INVE
NTORY_STATUS);

        return $manager
            ->getRepository($inventoryStatusClassName)
            ->findOneBy(['id' => $status]);
    }

    /** @return array<AbstractEnumValue> */
    protected function getOrderUnits(ObjectManager $manager): array
    {
        $inventoryStatusClassName =
ExtendHelper::buildEnumValueClassName('prod_main_unite_de_com
mande');

        return $manager
            ->getRepository($inventoryStatusClassName)
            ->findAll();
    }


. . .
```

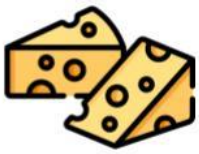
ДОДАТОК В

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ



Мета проєкту



Мета проекту полягає в розробці *інтернет-магазину для оптового продажу молочної продукції* дистриб'юторам з функціоналом повного циклу створення замовлень.

Цей магазин дозволятиме компаніям B2B ринку (бізнес до бізнесу) здійснювати *закупівлю продуктів* за *персоналізованими цінами,* проводячи *оплату платіжною системою*.

Завдання роботи

Для досягнення поставленої мети було сформовано наступний перелік завдань роботи:

-провести *аналіз B2B ринку* для визначення його особливостей;

-проаналізувати *наявне програмне забезпечення*, порівняти його переваги та недоліки;

-сформувати *перелік вимог* до розроблюваної системи;

-прийняти рішення щодо *використовуваної архітектури*;

-*спроектувати архітектуру* інтернет-магазину;

-обрати *оптимальний стек технологій* для реалізації застосунку;

-*розробити проєкт* відповідно до технічних вимог;

-провести *тестування системи,* виявлення і виправлення помилок;

-підготувати *керівництво користувача* для забезпечення легкості використання системи.

Актуальність теми

Сучасний бізнес вже активно використовує цифрові технології для оптимізації роботи. Електронна *комерція у сфері B2B* є однією з *найбільших за об'ємом продажів*.

B2B ринок є надзвичайно вибагливим щодо бізнес-процесів. Компанії, які займаються оптовим продажем товарів потребують таке програмне забезпечення, яке дасть змогу налаштовувати *умови співпраці з кожним клієнтом індивідуально*.

Крім того, прогнози аналітиків свідчать про *зростання ринку молочної продукції* в Європі на 4,15% на період з 2024 до 2029 року, досягаючи вартості в 253,08 мільярда доларів.

Таким чином створення Інтернет-магазину для оптового збуту продукції є надзвичайно актуальним.

Перелік вимог

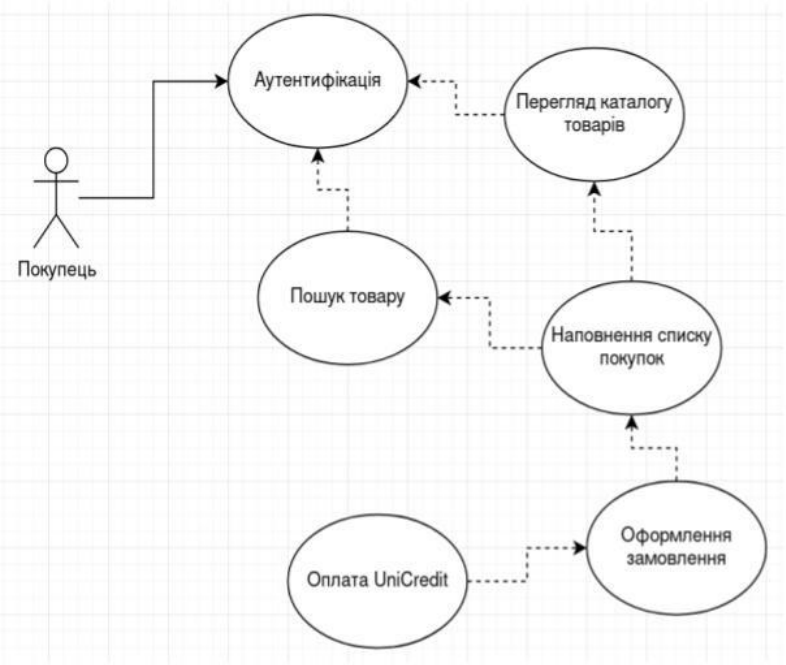
- управління асортиментом продуктів з адмін-панелі;
- налаштування *спеціальних цінових пропозицій* для обраних клієнтів;
- *пошук товарів* за ключовими словами;
- *екологічне оподаткування* на товари, що впливають на навколишнє середовище;
- *функціонал оформлення* замовлень;
- інтеграція оплати *платіжною системою* UniCredit.

Аналіз існуючого ПЗ

Для бізнесу, якому потрібен інтернет-магазин, оптимальним рішенням є *не написання його з нуля*, а використання певної платформи, яка задовольняє більшість потреб. Особливої уваги заслуговують наступні B2B та B2C платформи:

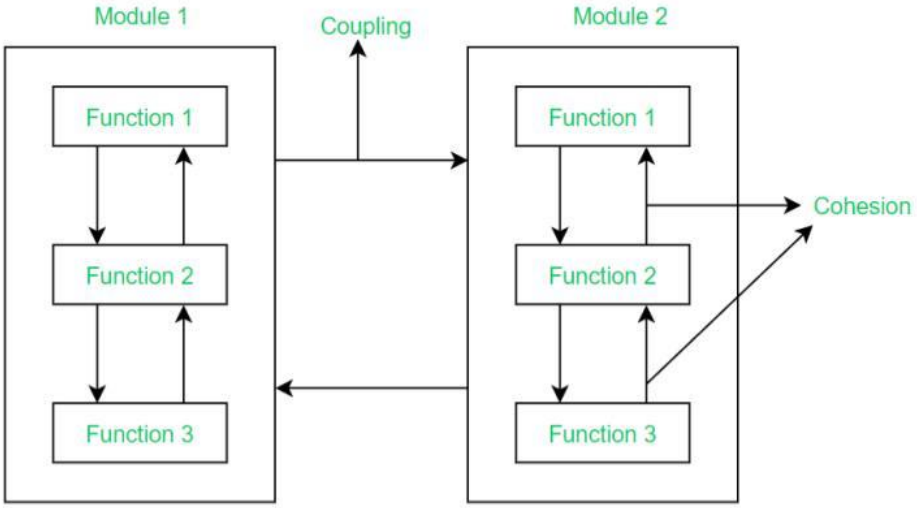
Назва платформи	***Переваги***	***Недоліки***
Sylius	Безкоштовна Написана на Symfony Легко розширюється (модульна)	Лише базовий функціонал, Спеціалізація на B2C
Magento	Гнучке ціноутворення Інтеграції з ERP та CRM	Написана без фреймворків Спеціалізація переважно на B2C
OpenCart	Безкоштовна Має звіти і метрики Широкий community	Лише базовий функціонал Написана без фреймворків Важко розширюється Спеціалізується виключно на B2C
ORO Commerce	Спеціалізується виключно на B2B Широкий функціонал Написана на Symfony Легко розширюється	Вартість (enterprise-edition) Потребує спеціальних знань

Варіанти використання системи



Архітектура додатку

Застосунок реалізовано з використанням архітектури *модульного моноліту*, яка поєднує переваги монолітної архітектури (*простота розгортання)*, та *ізольованістю*, яка притаманна мікросервісам.



Інструменти та технології

php

Symfony

ORO

PostgreSQL

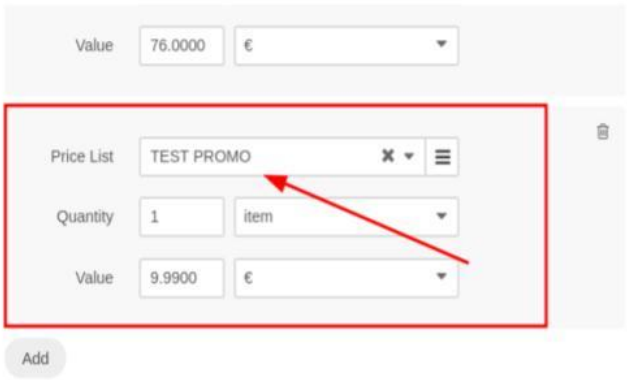
Rabbit MQ

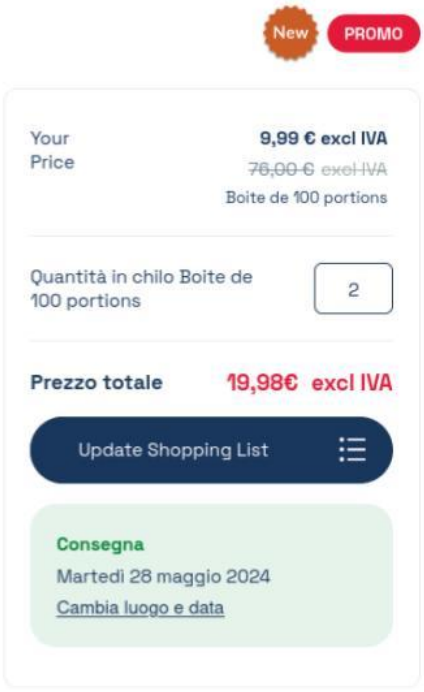
elasticsearch

Функціонал промо-цін

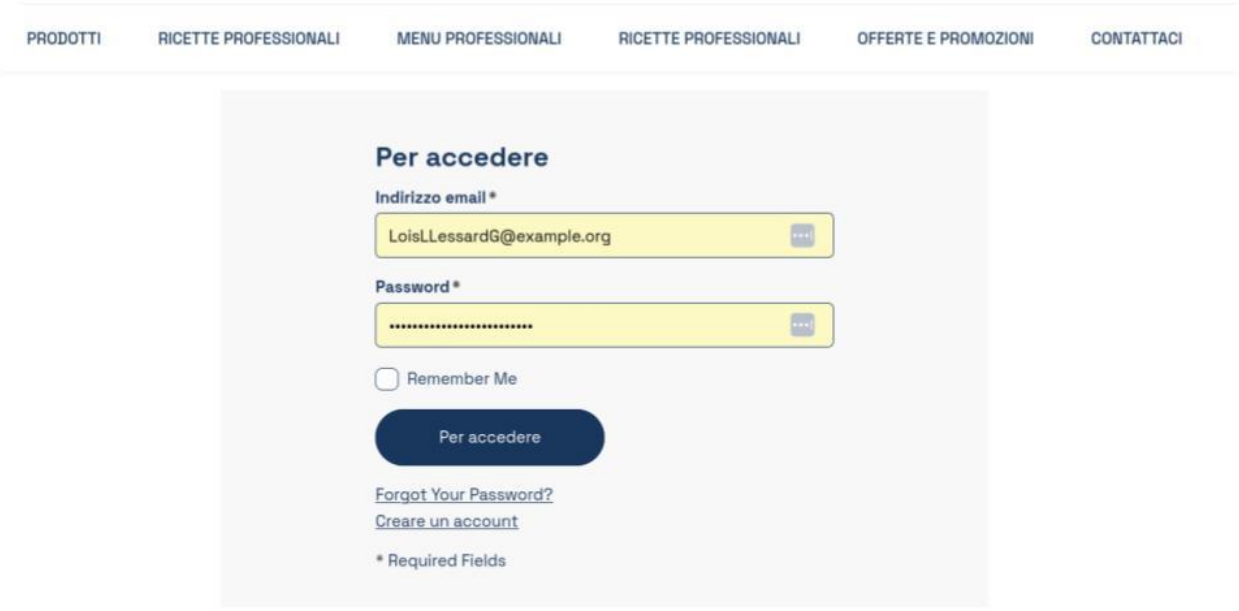
Для будь-якого продукту під будь-якого користувача або групи можна встановити персоналізовану промо-ціну.

Така ціна має пріоритет над звичайною ціною продукта, і відображається з спеціальним маркуванням “Промо”.

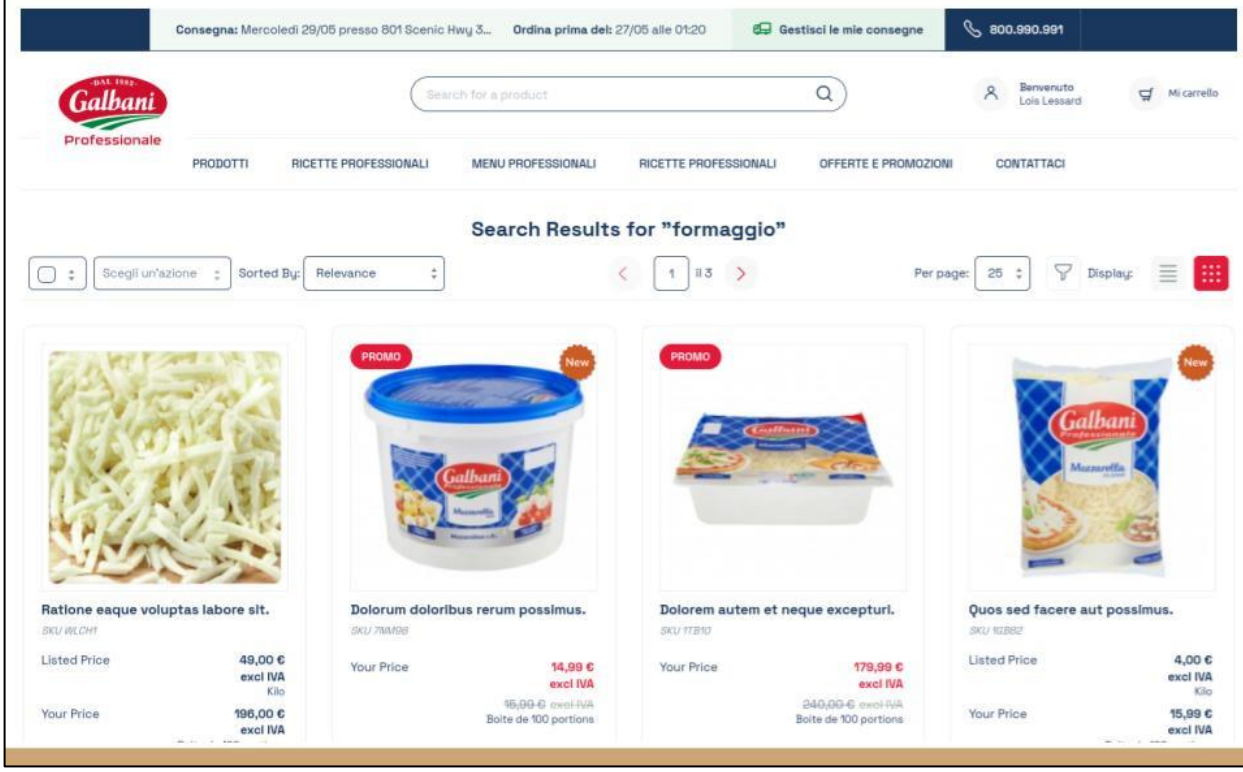




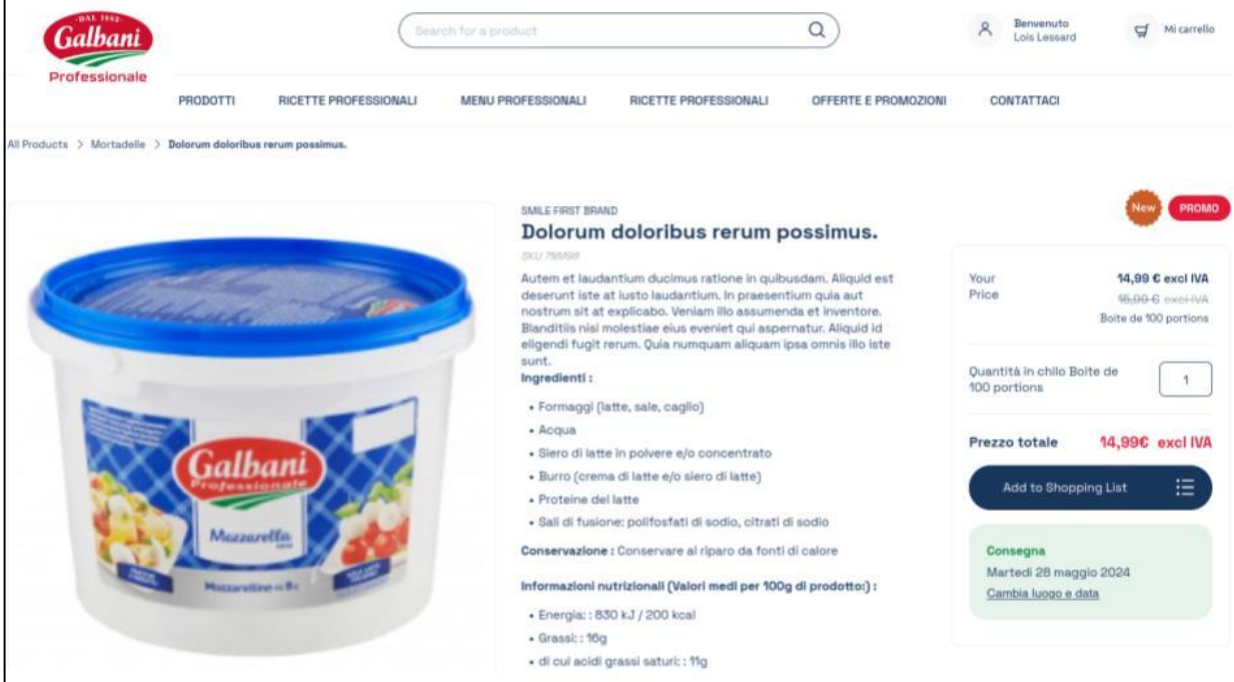
Автентифікація



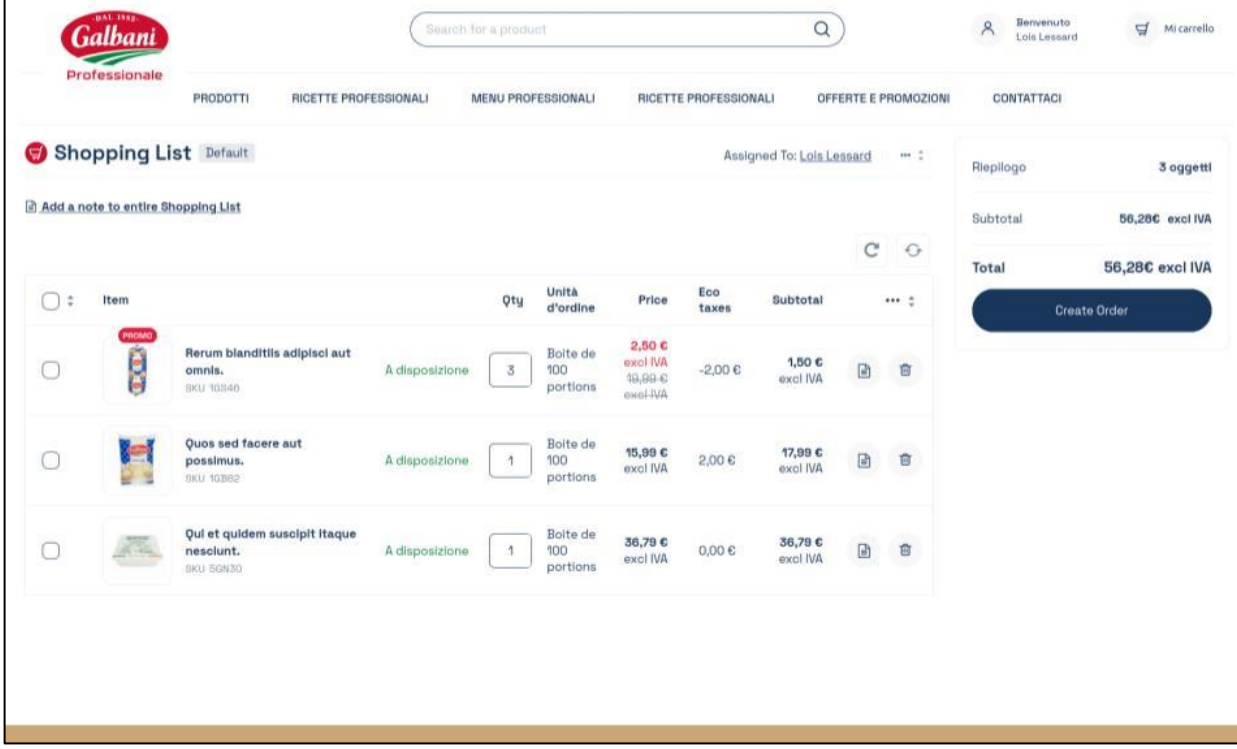
Пошук продуктів



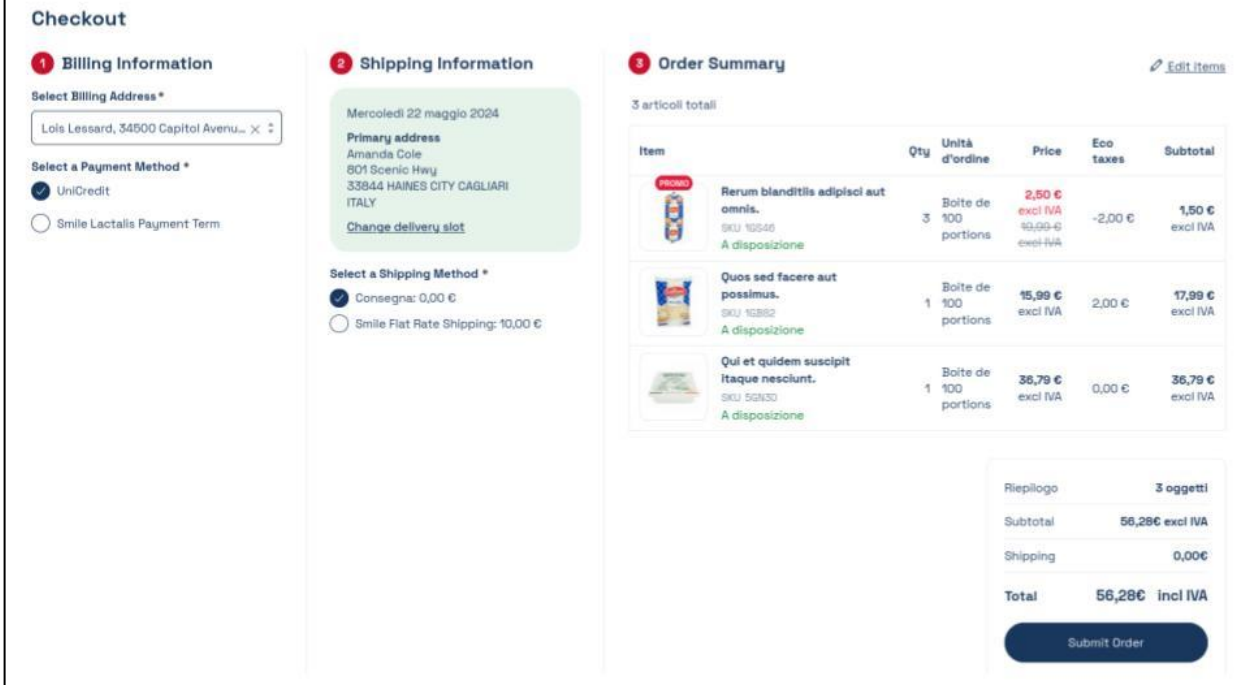
Сторінка продукту



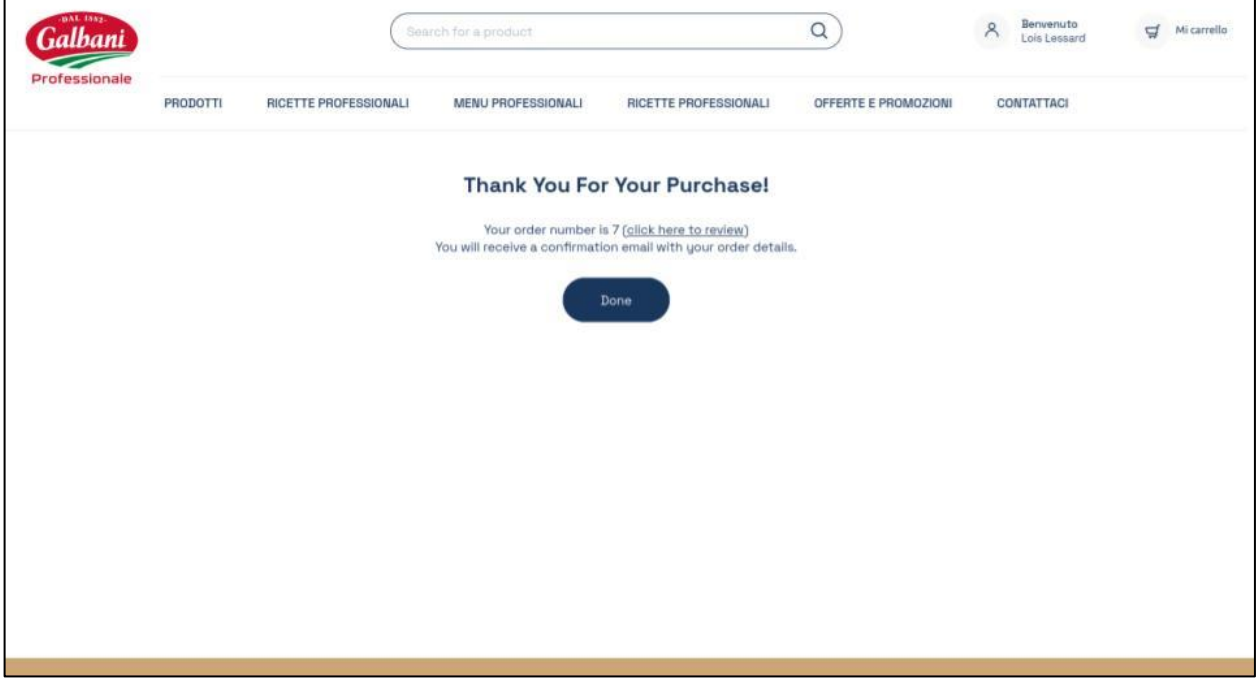
Список покупок



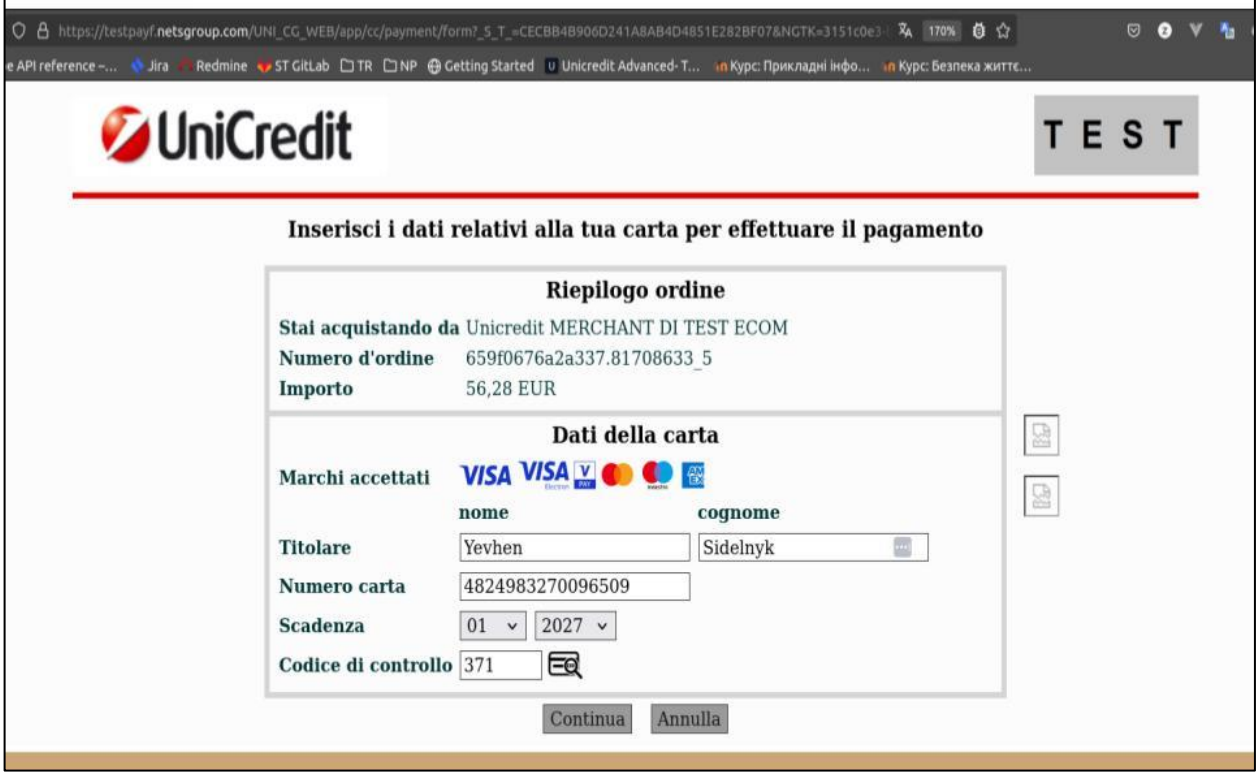
Оформлення замовлення



Замовлення успішно прийнято



Оплата замовлення



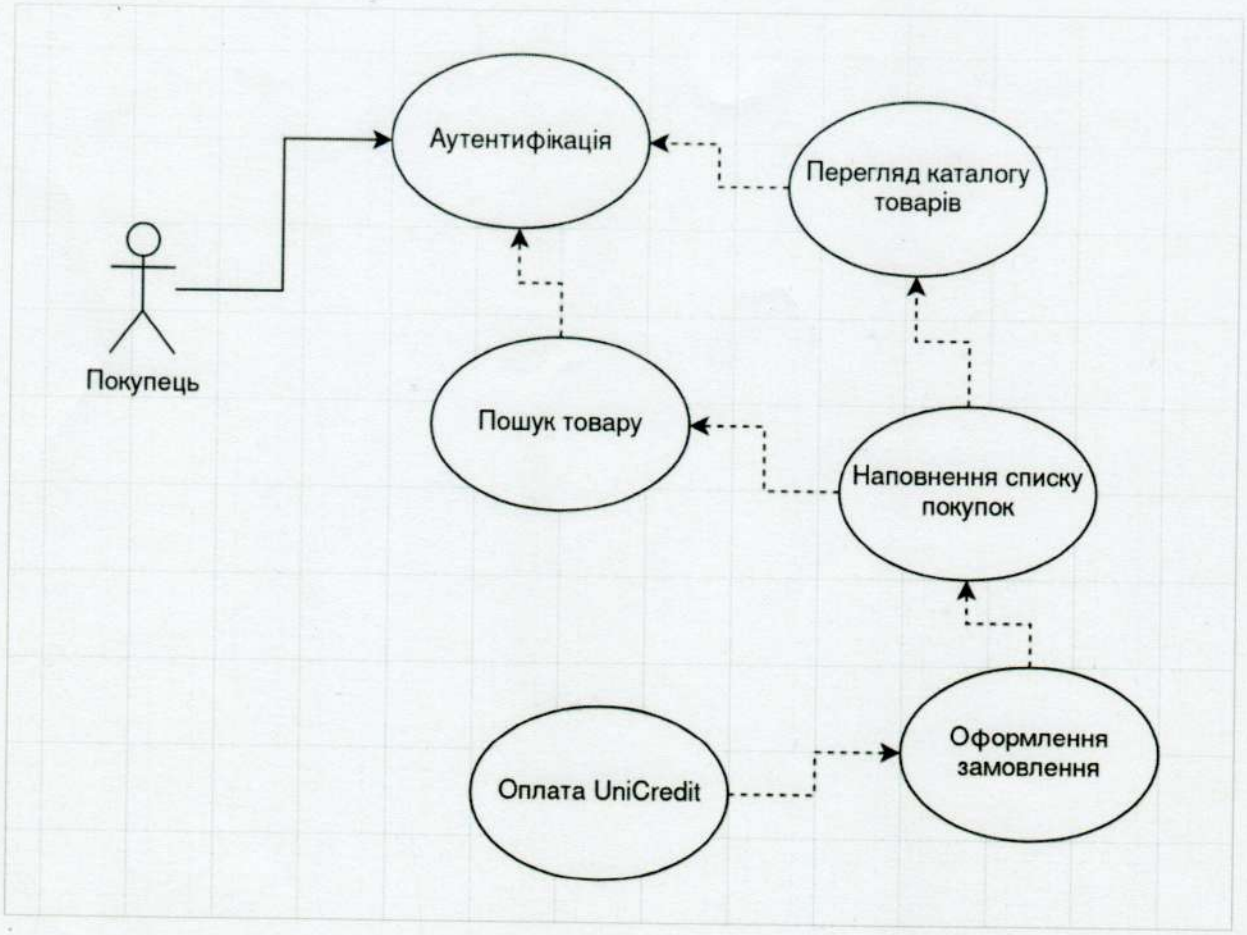
Висновки

Таким чином, в рамках даної кваліфікаційної роботи було створено інтернет-магазин для оптового продажу молочних продуктів:

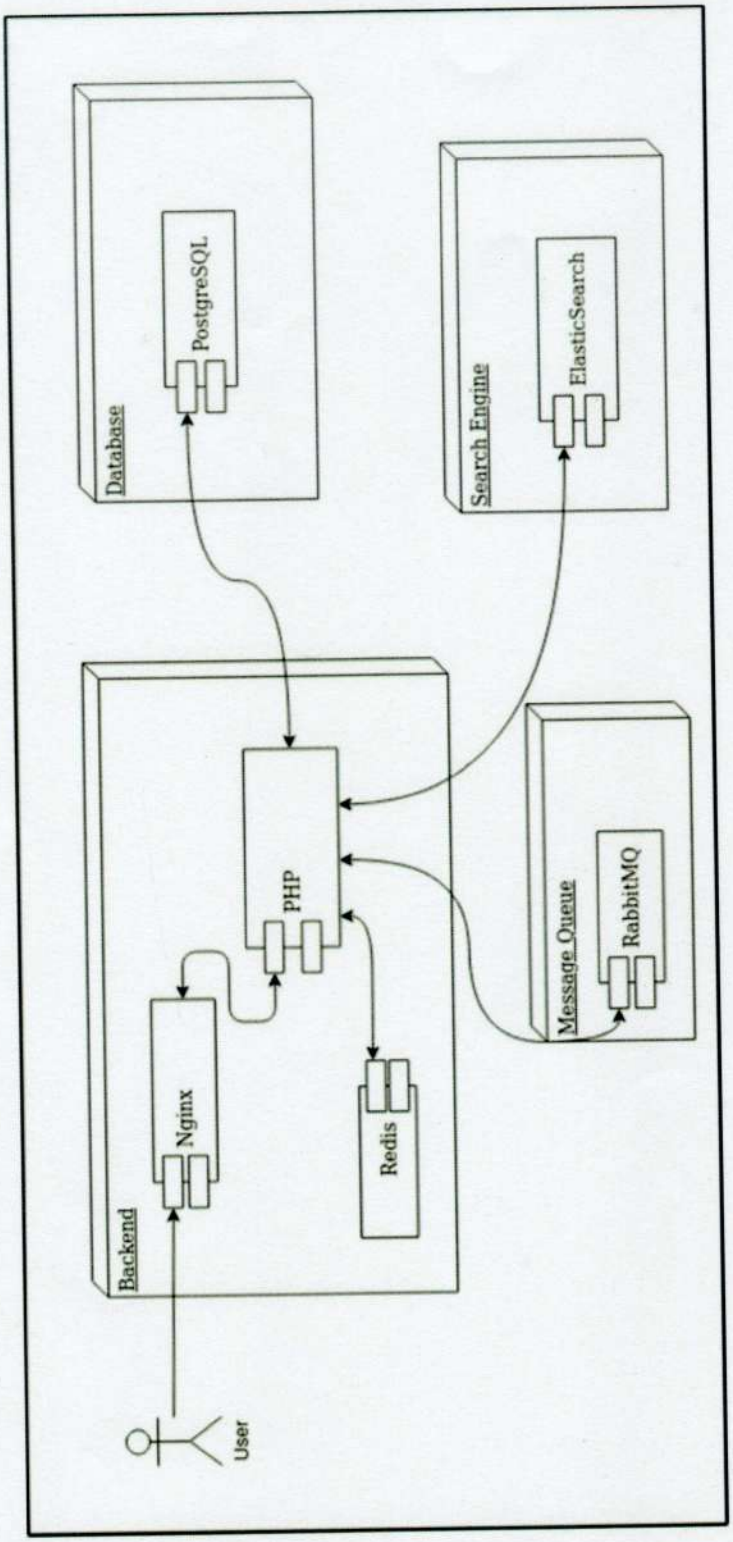
- проведено дослідження B2B-сектору ринку та специфіки роботи з оптовими клієнтами;
- проаналізовано існуючі програмні рішення предметної області;
- сформульовано перелік вимог до інтернет-магазину й зроблено візуалізацію бізнес-процесів за допомогою діаграм;
- спроектовано архітектуру у вигляді модульного моноліту;
- обрано стек технологій на основі Symfony;
- виконано технічну реалізацію проекту, зокрема функціонал промо-цін та оплат;
- проведено тестування системи;
- написано керівництво користувача.

Дякую за увагу!

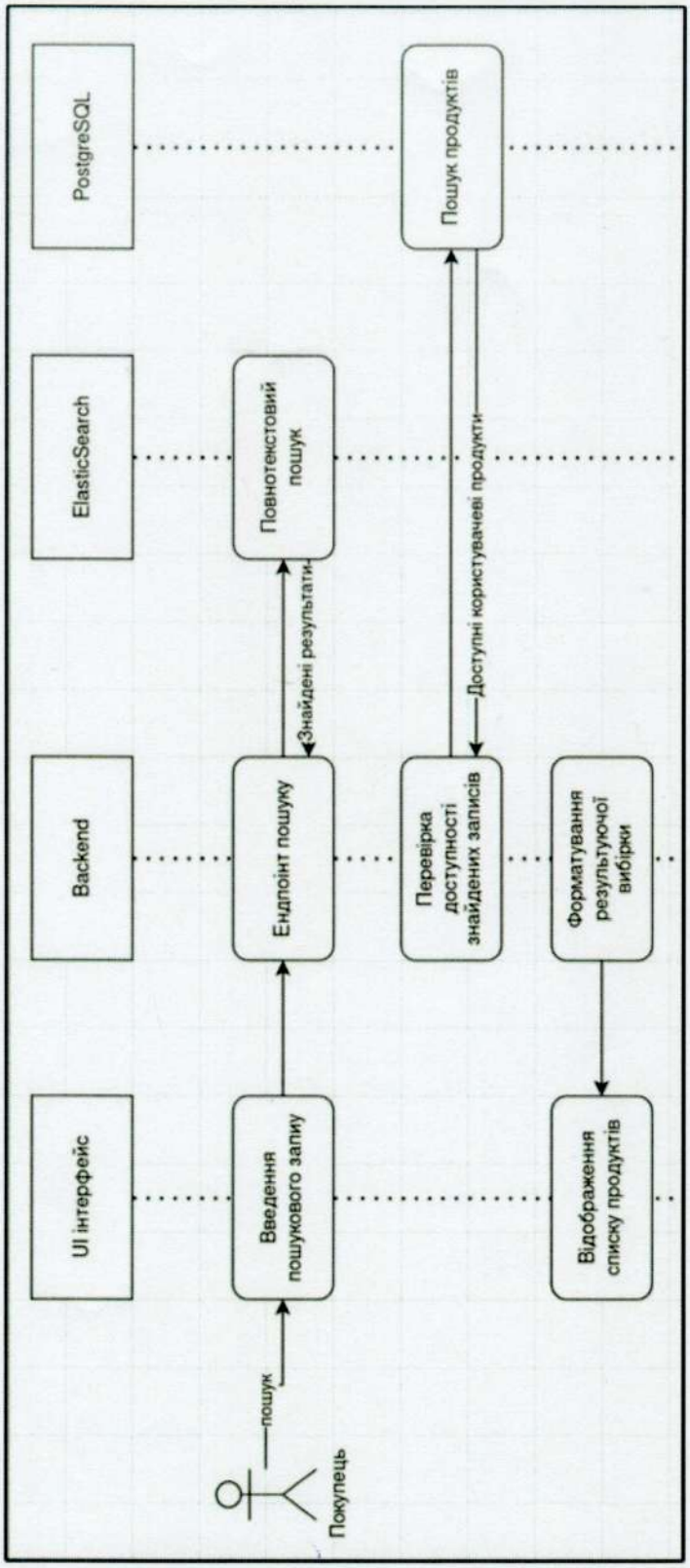
ГРАФІЧНА ЧАСТИНА



Змін.	Аркуш	№ документа	Підпис	Дата	КвРІПЗ. 2101103.01.04.Е8			
Виконав		Сідельник Є. О.		05.06	Діаграма варіантів використання	Літ	Маса	Масштаб
Керівник		Онишко О. Г.		05.06		Н		
						Аркуш 1	Аркушів 3	
Рецензент		Говорущенко Т.О.		05.06				
Н. контр.		Праворська Н. І.		05.06		ХНУ, ІПЗс-21-1		
Зав. каф.		Бедратюк Л.П.		05.06				



					КвРІПЗ. 2101103.01.04.Е8	
Змін.	Аркуш	№ документа	Підпис	Дата		
Виконав		Сідельник Є. О.		05.06	Діаграма розгортання	Літ: Н; Маса; Масштаб
Керівник		Онишко О. Г.		05.06		Аркуш 2 Аркушів 3
Рецензент		Говорущенко Т.О		05.06		ХНУ, ІПЗс-21-1
Н. контр.		Праворська Н. І.				
Зав. каф.		Бедратюк Л.П.		05.06		



КвРІПЗ. 2101103.01.04.Е8

Змін.	Аркуш	№ документа	Підпис	Дата
Виконав		Сідельник Є. О.		05.06
Керівник		Онишко О. Г.		05.06
Рецензент		Говорущенко Т.О.		05.06
Н. контр.		Праворська Н. І.		05.06
Зав. каф.		Бедратюк Л.П.		05.06

Діаграма послідовності пошуку товарів

Літ	Маса	Масштаб
Н		

Аркуш 3 Аркушів 3

ХНУ, ІПЗс-21-1

СУПРОВІДНІ ДОКУМЕНТИ

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Сідельник Євген Олександрович

студент III курсу спеціальності 121 – Інженерія програмного забезпечення, група ІПЗс-21-1

здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомився з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

02 січня 2024 р.

Підпис

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗс-21-1
Сідельник Є. О.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»:
"Інтернет-магазин з продажу молочної продукції дистриб'юторам"

(керівник роботи – канд. пед. наук, доцент Онишко Оксана Григорівна)
Прізвище, ім'я, по батькові

02 січня 2024р.
Дата

Підпис студента

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Сідельника Є. О.

Прізвище, ініціали

факультет ІТ, 3 курс, група ІПЗс-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті», згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

02 Січня 2024р.
дата

підпис

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 13%**

ID: 128459 Назва: БКР Інтернет-магазин з продажу молочної продукції дистриб'юторам Додано в БД: 2024-06-05 Автора: СІДЕЛЬНИК Євген Керівники: ОНИШКО О.Г., канд. пед. наук, доцент Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	124309	920	3599 (3%)	45 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

ID перевірки:
1016322650

Дата перевірки:
05.06.2024 11:20:24 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
08.06.2024 08:15:06 EEST

ID користувача:
100012953

Назва документа: БКР_Інтернет_магазин_з_продажу_молочної_продукції_дистриб'юторам_СІДЕЛЬНИК_Ониш...

Кількість сторінок: 86 **Кількість слів:** 16396 **Кількість символів:** 135667 **Розмір файлу:** 3.55 MB **ID файлу:** 1016121109

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

4.38%
Схожість

Найбільша схожість: 2.08% з джерелом з Бібліотеки (ID файлу: 1016114512)

2.76% Джерела з Інтернету	721	Сторінка 88
3.17% Джерела з Бібліотеки	267	Сторінка 92

0.04% Цитат

Цитати	1	Сторінка 93

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	2
Підозріле форматування	24 сторінки

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів:

Назва кваліфікаційної роботи: «Інтернет-магазин з продажу молочної продукції дистриб'юторам»
Автор: Сідельник Євген Олександрович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Онишко Оксана Григорівна кандидат педагогічних наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системою перевірки на плагіат Unicheck виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів, у рамках основних написів, у назвах публікацій переліку джерел посилання;

2) в якості запозичень системою Unicheck було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) запозичення, виявлені в тексті роботи, є фрагментарними.

Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 1.0%. Обсяг запозичень, визначений системою Unicheck виявлення збігів ідентичності/схожості, складає 4.38% і адресується до 721 джерел з Інтернету і 267 джерела з бібліотеки, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 10.06.24

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ОНИШКО

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «Бакалавр»

Дипломник Сідельник Євген Олександрович

Тема: "Інтернет-магазин з продажу молочної продукції дистриб'юторам"

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень: 3 ; кількість сторінок записки: 81

1. Короткий зміст пояснювальної записки та прийнятих рішень: У кваліфікаційній роботі Сідельника Євгена Олександровича розглянуто створення інтернет-магазину для продажу молочної продукції дистриб'юторам. Робота містить в собі аналіз предметної області, порівняння існуючих рішень, проектування архітектури та структури системи, програмну реалізацію й тестування, а також керівництво користувача. Використано мову програмування PHP, фреймворк Symfony й пакети платформи ORO, PostgreSQL в якості бази даних.

2. Висновок про відповідність роботи поставленому завданню: Робота повністю відповідає поставленому завданню. Виконані всі етапи, включаючи дослідження предметної області, проектування, реалізацію та тестування інтернет-магазину.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи:

Дослідження предметної області: виконано детальний аналіз B2B ринку молочної продукції, проаналізовано існуючі платформи для створення інтернет-магазинів.

Проектування програмного забезпечення: спроектовано архітектуру у вигляді модульного моноліту; обрано оптимальний стек технологій; розроблено діаграми.

Програмна реалізація: розроблено функціонал інтернет магазину з особливим акцентом на персоналізацію цінових каталогів та оплату платіжною системою.

Тестування системи: реалізовано загрузку фікстурних даних для ручного тестування; частково використано модульне та інтеграційне тестування функціоналу.

Технічні досягнення: Використано сучасний технологічний стек на основі фреймворку Symfony, об'єктно-орієнтоване програмування; популярні бібліотеки для тестування; загальноприйняті архітектурні підходи тощо.

4. Позитивні сторони роботи: Реалізовано досить складний і гнучкий функціонал промо-цін; влучно використано шаблони проектування; високий рівень володіння архітектурими принципами, включаючи модульне розділення функціоналу; використання передових технологій для розробки; розуміння фундаментальних принципів тестування; докладне керівництво користувача.

5. Негативні сторони роботи: Певні аспекти функціоналу могли б бути реалізовані більш оптимальним чином; деякі сценарії використання системи не були в достатній мірі висвітлені в описі.

6. Оцінка графічного оформлення та пояснювальної записки: Графічне оформлення та пояснювальна записка виконані на належному, включаючи всі необхідні схеми, діаграми та ілюстрації, додаючи контекст розуміння до загального тексту записки.

7. Відгук про кваліфікаційну роботу в цілому: кваліфікаційна робота Сідельника Євгена Олександровича є завершеним проектом, який демонструє високий рівень знань та навичок у галузі інженерії програмного забезпечення. Робота виконана відповідно до поставлених завдань та вимог, містить теоретичні та практичні результати, які можуть бути використані в подальшій розробці веб-додатків.

8. Інші зауваження: Варто було б додати більше тестових сценаріїв під різні варіанти використання для більш широкої оцінки стабільності системи.

9. Оцінка кваліфікаційної роботи: Кваліфікаційна робота заслуговує на оцінку "добре" за змістом, якістю виконання та відповідністю поставленим завданням.

РЕЦЕНЗЕНТ: Говорущенко Тетяна Олександрівна, доктор технічних наук, зав кафедри комп'ютерної інженерії та інформаційних систем (КІІС) ХНУ

"07" 06 2024 р. _____

(підпис)