**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ**

**Інститут (факультет)** Автоматизації і комп'ютерних систем
**Кафедра** Інформаційних технологій, штучного інтелекту і кібербезпеки

| **«До захисту в ЕК»** | **«До захисту допущено»** |
|---|---|
| Директор інституту(декан факультету) | Завідувач кафедри |
| _____ Андрій ФОРСЮК | _____ Сергій ГРИБКОВ |
| (підпис) (ім'я та прізвище) | (підпис) (ім'я та прізвище) |
| «05» червня 2023р. | «05» червня 2023р. |

# КВАЛІФІКАЦІЙНА РОБОТА
# НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

зі спеціальності 122 «Комп'ютерні науки»
(код та назва спеціальності)
освітньо-професійної програми Комп'ютерні науки

на тему: Розроблення Web-додатку для підтримки роботи з замовниками продукції ТОВ "Пирятинський сирзавод"

Виконав: здобувач 4 курсу, групи КН-4-3

Загарюй Ростислав Валерійович _____
(прізвище, ім'я, по батькові повністю) (підпис)

Керівник Харкянен Олена Валеріївна _____
(прізвище , ім'я та по батькові повністю) (підпис)

Консультанти _____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

Рецензент _____ _____
(ім'я та прізвище) (підпис)

Я як здобувач(ка) Національного університету харчових технологій розумію і підтримую політику університету з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволеної допомоги під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач _____
(підпис)

Київ - 2023р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) Автоматизації і комп'ютерних систем
Кафедра Інформаційних технологій, штучного інтелекту і кібербезпеки
Освітній ступінь бакалавр
**Спеціальність** 122 «Комп'ютерні науки»
**Освітньо-професійна програма** Комп'ютерні науки

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри**
Інформаційних технологій, штучного інтелекту і кібербезпеки
Сергій ГРИБКОВ

**" 04 " квітня 2023 року**

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
**Загарюя Ростислава Валерійовича**
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення Web-додатку для підтримки роботи з замовниками продукції ТОВ "Пирятинський сирзавод"

керівник роботи Харкянен Олена Валеріївна, к.т.н., доцент
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від 03 квітня 2023 року № 204-кс
2. Строк подання здобувачем роботи 01.06.2023 р.
3. Вихідні дані до роботи
Інформація про ТОВ «Пирятинський сирзавод», дані про відділ збуту ТОВ «Пирятинський сирзавод», дані про поточний стан ТОВ «Пирятинський сирзавод»
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) системний аналіз підприємства, виявлення недоліків та визначення задач автоматизації, розробка технічного завдання, алгоритмізація та реаліація комплексу задач автоматизації, розробка інструкції користувача, висновки

5. Перелік графічного матеріалу
Організаційна структура підприємства, функціональні моделі, скріншоти програм-аналогів, скріншоти фрагментів коду, скріншоти розробленого веб-додатку

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1 | доцент к.т.н. Харкянен О.В. | 04.04.2023 | 22.04.2023 |
| 2 | доцент к.т.н. Харкянен О.В. | 04.04.2023 | 31.04.2023 |
| 3 | доцент к.т.н. Харкянен О.В. | 04.04.2023 | 24.05.2023 |
| 4 | доцент к.т.н. Харкянен О.В. | 04.04.2023 | 27.05.2023 |

7. Дата видачі завдання_____04 квітня 2023 року_____

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів виконання кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Системний аналіз діяльності підприємства | 04.04.2023-22.04.2023 | Виконано |
| 2 | Розробка технічного завдання | 23.04.2023-31.04.2023 | Виконано |
| 3 | Розробка веб-додатку | 01.05.2023-22.05.2023 | Виконано |
| 4 | Створення інструкції для користувачів | 23.05.2023-24.05.2023 | Виконано |
| 5 | Оформлення пояснювальної записки | 25.05.2023-27.05.2023 | Виконано |
| 6 | Створення презентації | 28.05.2023-29.05.2023 | Виконано |

**Здобувач** _____ _____Загарюй Р.В._____
(підпис) (прізвище та ініціали)

**Керівник роботи** _____ _____Харкянен О.В._____
( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

Ця виконана робота - втілення кваліфікаційного завдання, метою якого було вивчення і практична реалізація проекту «Створення веб-додатка для ефективного взаємодії з клієнтами ТОВ «Пирятинський сирзавод».

Вибір об'єкта для дослідження обумовлений інтересом до діяльності компанії «Пирятинський сирзавод», і особливо до її відділу продажів, від роботи якого залежить успішність всього підприємства.

Основна мета цього проекту полягала не лише у застосуванні теоретичних знань та практичних навичок, отриманих в процесі навчання, але й у демонстрації їх ефективності на реальному прикладі - ведучому підприємстві в своїй галузі - Пирятинському сирзаводу.

Виконуючи цю кваліфікаційну роботу, було зроблено глибокий системний аналіз обраного об'єкта, зокрема, проведено детальне дослідження внутрішніх процесів у відділі продажів, виявлено поточні проблеми і визначено конкретні завдання для автоматизації.

В результаті цього аналізу було висунуто інноваційну пропозицію - створити веб-додаток, який міг би стати незамінним помічником для працівників відділу продажів та споживачів продукції сирзаводу. Веб-додаток планується як інструмент, що максимально полегшує взаємодію з покупцями продукції.

Весь процес створення, включаючи вимоги до веб-додатку, були висвітлені в технічному завданні. Робота містить не лише детальний опис етапів розробки веб-додатка, але й інструкцію для користувачів, яка допоможе використовувати його максимально ефективно.

Кваліфікаційна робота складається з 81 сторінок, 5 таблиць, 69 рисунка, 5 додатків та 26 літературних джерел.

**КЛЮЧОВІ СЛОВА:** ОНЛАЙН ЗАМОВЛЕННЯ ПРОДУКЦІЇ, ПРОДАЖ, ПРОПОЗИЦІЯ, ВЕБ-ДОДАТОК, СИРЗАВОД, САТИСТИКА ПРОДАЖІВ

## ABSTRACT

This completed work represents the realization of a qualification task, the aim of which was to study and practically implement the project "Creating a web application for effective interaction with customers of LLC "Pyriatyn Cheese Factory".

The choice of object for study was determined by an interest in the activities of the "Pyriatyn Cheese Factory" company, especially its sales department, the work of which determines the success of the entire enterprise.

The main goal of this project was not only to apply theoretical knowledge and practical skills obtained in the process of learning, but also to demonstrate their effectiveness on a real example - a leading enterprise in its industry - Pyriatyn Cheese Factory.

In the process of performing this qualification work, a deep systemic analysis of the selected object was carried out, in particular, a detailed study of the internal processes in the sales department, identification of current problems and determination of specific tasks for automation.

As a result of this analysis, an innovative proposal was put forward - to create a web application that could become an indispensable assistant for employees of the sales department and consumers of the cheese factory products. The web application is planned as a tool that maximally facilitates interaction with product buyers.

The entire creation process, including the requirements for the web application, were highlighted in the technical task. The work contains not only a detailed description of the stages of web application development but also a user manual that will help use it as efficiently as possible.

The qualification work consists of 81 pages, 5 tables, 69 figures, 5 appendices, and 26 literary sources.


**KEYWORDS: ONLINE ORDERING OF PRODUCTS, SALE, PROPOSAL, WEB APPLICATION, CHEESE FACTORY, SALES STATISTICS**

# ЗМІСТ

# ВСТУП

Зараз інтернет став вельми необхідним та невід'ємним елементом розвитку різноманітних сфер людської діяльності. Він відіграє надзвичайно важливу роль у процесі надання інформації користувачу та її обробці. Однією з галузей, що активно потребує впровадження web-технологій, є сфера виробництва продукції.

У рамках дипломної роботи розглядається актуальна тема "Розробка Web-додатку для підтримки роботи з замовниками продукції Пирятинського сирзаводу". З урахуванням швидкого темпу технологічного прогресу та змін у споживацьких пристрастях споживачів, необхідно впроваджувати нові підходи до управління виробництвом та взаємодії з клієнтами. Розробка веб-додатку є одним із шляхів до покращення процесів управління та спілкування замовників з Пирятинським сирзаводом.

Веб-додаток, що розробляється, надасть користувачам можливість зручно та ефективно взаємодіяти з Пирятинським сирзаводом. Реалізація додатку передбачає наявність трьох основних ролей: Адміністратора, Користувача та Менеджера. Кожна з цих ролей виконує важливі функції, спрямовані на оптимізацію процесів управління, контролю та забезпечення якості продукції.

Розробка Web-додатку для підтримки роботи з замовниками продукції Пирятинського сирзаводу є актуальною та перспективною темою. Вона дозволить покращити ефективність роботи, спростити процеси взаємодії замовників зі сирзаводом, а також забезпечить збільшення задоволення клієнтів якістю обслуговування. Дана розробка веб-додатку має потенціал стати кроком у напрямку вдосконалення виробничих процесів та підвищення конкурентоспроможності Пирятинського сирзаводу на ринку продуктів харчування.

# РОЗДІЛ 1. АНАЛІЗ СИСТЕМИ ФУНКЦІОНУВАННЯ ТОВ «ПИРЯТИНСЬКИЙ СИРЗАВОД»

## 1.1. Загальна характеристика ТОВ «Пирятинський сирзавод»

ТОВ "Пирятинський сирзавод" є відомим виробником сирів та молочних продуктів з багатолітньою історією і стабільною репутацією. Заснований у далекому 1920 р. сирзавод пройшов шлях від скромних початків до визнаного лідера в галузі.

Однією з основних метою ТОВ "Пирятинський сирзавод" є задоволення потреб клієнтів у високоякісних та смачних молочних продуктах. Компанія прагне забезпечити своїх споживачів продукцією найвищої якості, виготовленою з використанням натуральних інгредієнтів та за допомогою передових технологій.

ТОВ "Пирятинський сирзавод" спеціалізується на виробництві різноманітних сирів різних сортів. Компанія має розгалужену сферу діяльності, постачаючи свою продукцію на ринки як внутрішні, так і зовнішні. Їх продукція користується популярністю не лише серед місцевих споживачів, але й за межами країни.

Однією з особливостей ТОВ "Пирятинський сирзавод" є постійне прагнення до вдосконалення та інноваційного розвитку. Компанія вкладає значні зусилля в дослідження і розробку нових продуктів, удосконалення рецептур та технологій виробництва. Впровадження сучасних практик і автоматизація допомагають забезпечувати ефективність та якість продукції.

"Пирятинський сирзавод" прагне до збереження традиційного смаку та якості, одночасно відповідаючи сучасним вимогам споживачів. Компанія зосереджується на створенні продуктів, які відповідають вишуканим смаковим преференціям своїх клієнтів [1].

Загальна характеристика ТОВ "Пирятинський сирзавод" відображає його успішний шлях, високу якість продукції, постійне прагнення до розвитку та задоволення потреб споживачів. Компанія займає впевнені позиції на ринку та зберігає свою репутацію як провідного виробника молочної продукції.

## 1.2. Організаційна структура ТОВ «Пирятинський сирзавод», роль і взаємодія підрозділів

Схематичний вигляд організаційної структури має наступний вигляд на зображеній на рисунку 1.1.

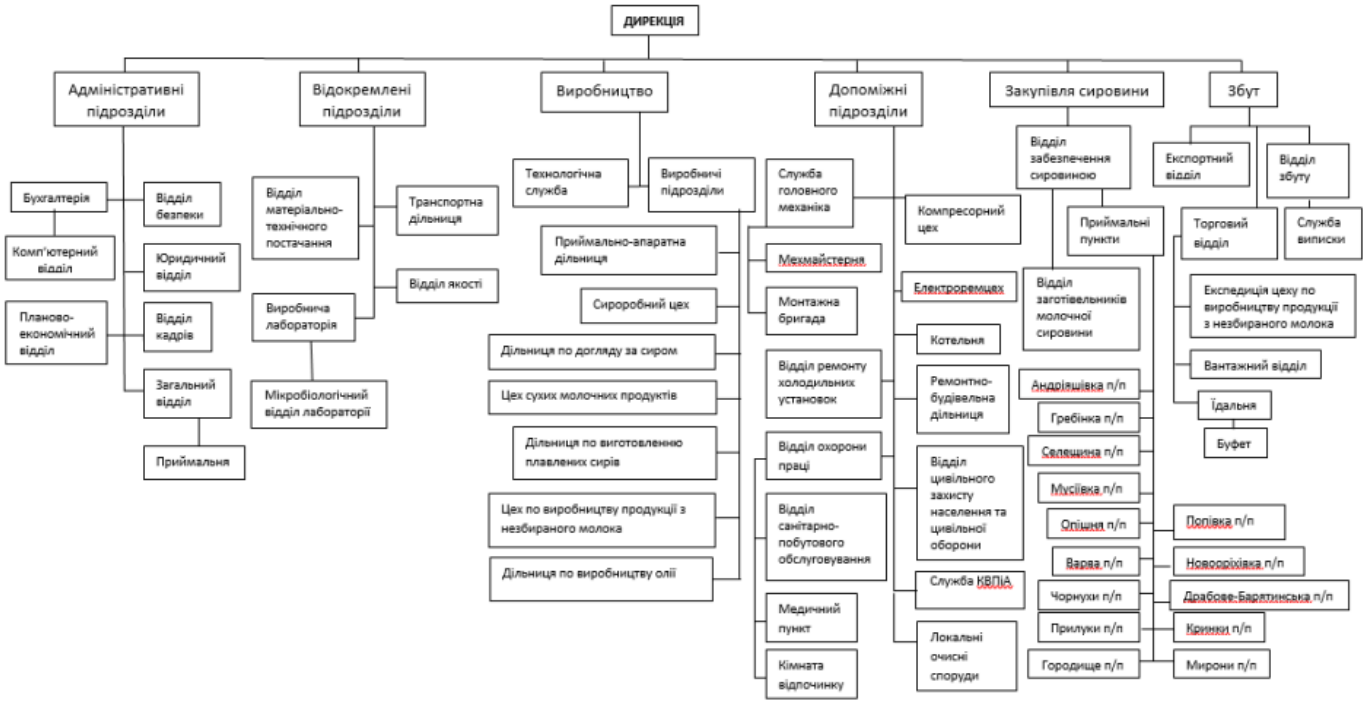



Рис. 1.1. Організаційна структура ТОВ «Пирятинський сирзавод»

Організаційна структура містить різні підрозділи, які взаємодіють один з одним для забезпечення ефективної роботи організації. Основні підрозділи можна класифікувати на наступні групи:

1. Адміністративні підрозділи:

- Бухгалтерія: відповідає за облік фінансових операцій, платежів, звітності та податкових питань.
- Відділ безпеки: забезпечує безпеку працівників, майна та інформації в організації.
- Відділ Транспортна матеріально-технічного дільниця постачання: відповідає за забезпечення потреб організації в матеріалах, обладнанні та транспортних послугах.

- Комп'ютерний відділ: займається обслуговуванням та підтримкою комп'ютерної інфраструктури, програмного забезпечення та мережі організації.
- Юридичний відділ: надає правову підтримку, займається укладанням та аналізом контрактів, вирішенням юридичних питань.

2. Виробничий відділ:

- Виробництво: здійснює безпосередню виробничу діяльність організації.
- Технологічна Виробничі служба підрозділи: відповідають за розробку та впровадження технологій виробництва.
- Приймально-апаратна Дільниця: здійснює прийом сировини, контроль якості та відправку готової продукції.
- Сироробний цех: виготовляє молочні продукти з сировини.
- Дільниця по догляду за сиром: забезпечує правильне зберігання і обробку сиру.
- Цех сухих молочних продуктів: виробляє сухі молочні продукти.

3. Допоміжні підрозділи:

- Служба головного Механіка: відповідає за технічне обслуговування та ремонт основного устаткування.
- Компресорний Мехмайстерня: забезпечує обслуговування та ремонт компресорного устаткування.
- Монтажна бригада Електродемнех: займається монтажем та обслуговуванням електричного обладнання.
- Котельня: забезпечує постачання тепла та гарячої води.
- Відділ ремонту холодильних: здійснює ремонт та технічне обслуговування холодильного устаткування.
- Ремонтно-установок будівельна дільниця: відповідає за ремонт та обслуговування будівель та споруд.
- Відділ охорони праці: забезпечує безпеку праці та дотримання норм з охорони праці.

- Відділ цивільної оборони: відповідає за планування та координацію дій у разі надзвичайних ситуацій.

4. Закупівля сировини:

- Відділ забезпечення сировиною: займається закупівлею та постачанням сировини для виробництва.
- Приймальні пункти: виконують функції прийому та контролю якості сировини.

5. Збут:

- Експортний Відділ: організовує експорт продукції організації.
- Відділ збуту: відповідає за збут продукції на внутрішньому ринку.
- Торговий Служба: здійснює торгівельну діяльність та укладання контрактів з клієнтами.
- Відділ виписки: займається випискою документів, обліком продажів та відвантажень.
- Експедиція цеху по виробництву: забезпечує організацію внутрішнього переміщення готової продукції.

Ці підрозділи взаємодіють між собою, передаючи інформацію, матеріали та послуги для забезпечення ефективності виробництва та операцій організації в цілому. Наприклад, відділ постачання забезпечує бухгалтерію матеріалами для ведення обліку, а відділ виробництва співпрацює з відділом безпеки для забезпечення безпеки на робочому місці [2].

## 1.3. Аналіз нинішнього стану комп’ютеризації підприємства

ТОВ «Пирятинський сирзавод» використовує найсучасніше технологічне обладнання, що є одним з трьох компонентів успіху, наряду з багатими традиціями та десятиліттями досвіду, а також професійними навичками, поєднаними з щирою турботою про улюблений бізнес.

Особливістю та технологічною перевагою «Пирятинського сирзаводу» є виробництво сиру виключно з натурального молока без будь-яких рослинних добавок та без використання молочного порошку. Основний акцент діяльності заводу - виробництво твердого та м'якого сиру. Крім того, завод виробляє цілісні

молочні продукти, суху демінералізовану сироватку та молочне масло. Нещодавно була здійснена реконструкція компресорного відділення заводу для безпеки та стабільного постачання підприємства холодом. Тепер він оснащений сучасним японським обладнанням, що дозволяє більш ефективно використовувати енергетичні ресурси.

У 2007 році на заводі була встановлена повністю автоматизована лінія формування та пресування сиру, вироблена іспанською компанією Fibosa. Після цього потужності підприємства з переробки молока в сир зросли до 500 тонн молока на день, тобто 50 тонн кінцевого продукту на день. Зараз сир виробляється на автоматизованій лінії, де людський контакт з продуктом зведено до мінімуму. Високий рівень механізації, автоматизації, санітарії та гігієни виробництва, а також висока кваліфікація експертів гарантують стабільну якість продукції.

Згідно з наведеною інформацією, можна зробити висновок, що ТОВ «Пирятинський сирзавод» використовує високий рівень комп'ютеризації в своїх виробничих процесах, включаючи автоматизовані лінії формування та пресування сиру, а також сучасне японське обладнання в компресорному відділенні. Однак, без додаткових деталей або більш свіжої інформації важко дати більш детальний аналіз щодо стану комп'ютеризації на підприємстві [1].

Збут Пирятинського сирзаводу відбувається за допомогою систем Microsoft Excel та 1С. Однак, варто зазначити, що ці системи можуть мати обмежені можливості зручного заповнення для звичайних користувачів.

Microsoft Excel, як електронна таблиця, може бути використана для організації та аналізу даних збуту. Однак, для користувачів з обмеженим досвідом роботи з Excel, заповнення даних може бути складним завданням. Вимагається знання структури таблиці, форматування даних та правильного введення інформації [4].

1С, як інтегрована система управління, надає більш широкі можливості для збуту та обробки даних. Однак, налаштування та використання 1С зазвичай вимагає фахівця з конфігурування та налагодження системи. Для звичайних користувачів можуть виникати труднощі при роботі з цією системою [3].

Враховуючи складнощі заповнення для звичайних користувачів, можливо доцільно розглянути альтернативні рішення, такі як розробка спеціалізованого інтерфейсу користувача або використання спеціалізованого програмного забезпечення для збуту, яке дозволяє зручне заповнення даних та автоматизовану обробку замовлень та продажів.

В кінцевому рахунку, важливо врахувати потреби та можливості організації, і, при необхідності, розглянути можливість впровадження більш зручних інструментів для заповнення даних та управління процесом збуту.

## 1.4. Розроблення функціональної моделі та визначення задач автоматизації

### 1.4.1. Функціональна модель ТОВ «Пирятинський сирзавод»

Функціональна модель "Робота з замовниками продукції Пирятинського сирзаводу" була створена у AllFusion ERWin Process Modeler з метою аналізу та виявлення переваг та недоліків у процесі взаємодії з замовниками пробукції. AllFusion ERWin Process Modeler є потужним інструментом для моделювання процесів, який дозволяє візуалізувати послідовність дій та взаємозв'язки між різними елементами процесу [4].

Створення функціональної моделі дозволило детально розглянути кожен етап процесу роботи з замовниками. Ця модель надає уявлення про послідовність дій, ролі різних стейкхолдерів та взаємодію між ними. Вона також допомагає ідентифікувати ключові вхідні та вихідні дані, контролерів та процеси, що відбуваються в рамках системи.

Аналіз функціональної моделі дозволяє виявити та описати переваги та недоліки у процесі роботи з замовниками. Це сприяє кращому розумінню поточних бізнес-процесів та ідентифікує можливі області для поліпшення. Наприклад, модель може показати, які етапи процесу потребують оптимізації, де виникають затримки або непослідовності, та які можливості для автоматизації можуть бути впроваджені [10]

Завдяки AllFusion ERWin Process Modeler та функціональній моделі, можна провести детальний аналіз роботи з замовниками продукції Пирятинського сирзаводу, виявити сильні та слабкі сторони, та розробити план дій для

вдосконалення процесу та покращення задоволення замовників. Контекстна діаграма функціональної моделі організації замовлення продукції ТОВ "Пирятинський сирзавод" наведена на рисунку 1.2.

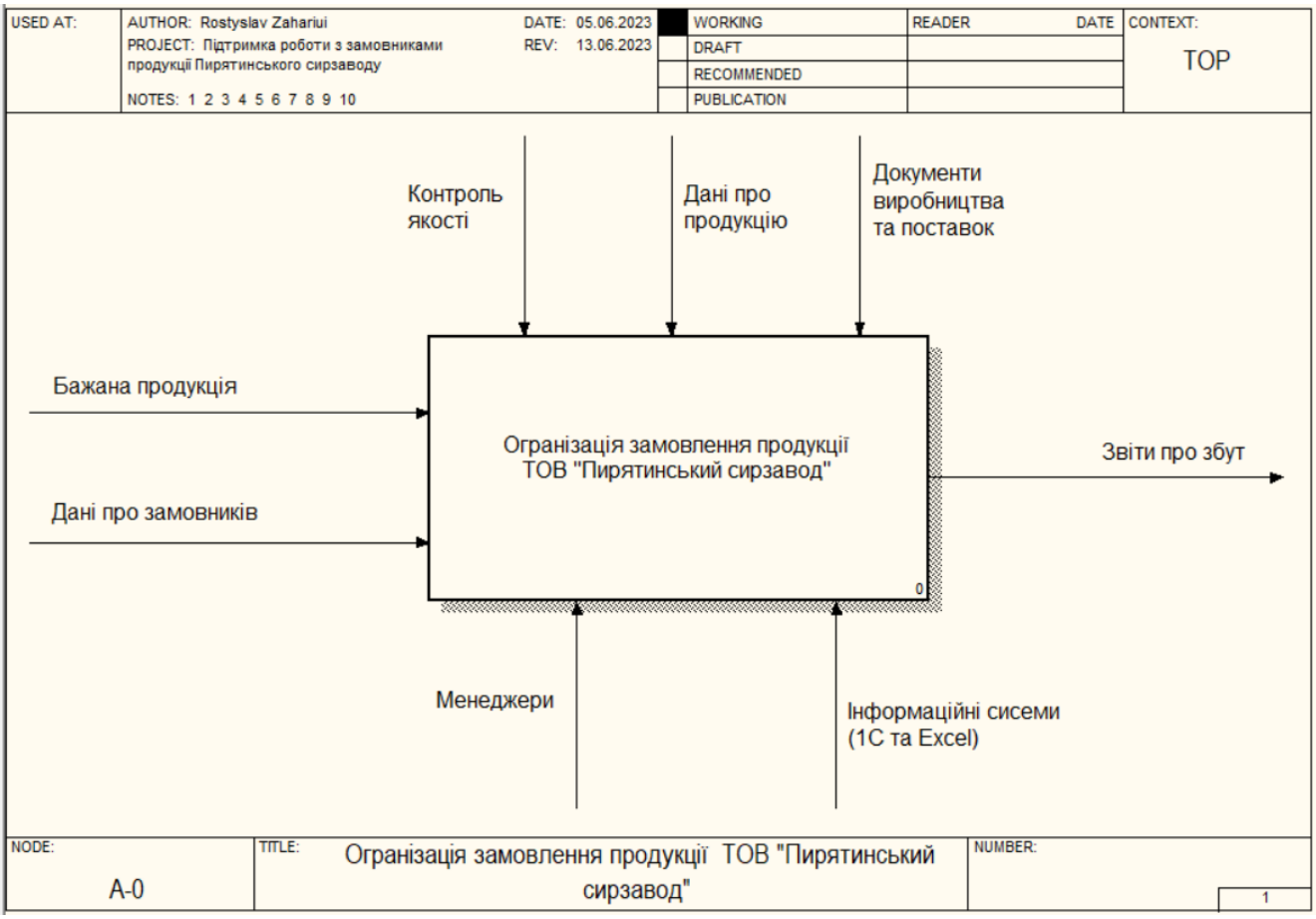



Рис.1.2. Контекстна діаграма функціональної моделі

Нижче наведено декомпозицію функціональної моделі "Робота з замовниками продукції Пирятинського сирзаводу" у AllFusion Process Manager (див. рис. 1.3). Кожен рівень декомпозиції описує більш деталізовані процеси та функції системи, що дозволяє виявити в процесі замовлення продукції.

Діяльність "Формування списку продукції для замовлення" описує процес вибору замовником продукції, яку він бажає замовити. Замовник обирає категорію продукції, переглядає доступні продукти, вибирає конкретний продукт і додає його до свого списку. Цей процес повторюється для інших потріних продуктів. Після завершення замовник має список продукції для замовлення.

Діяльність "Перевірка наявності продукції, узгодження і виконання замовлення" описує перевірку наявності продукції на складі, узгодження замовлення з клієнтом та його виконання. Включає такі етапи: перевірка наявності продукції, комунікація з замовником для узгодження деталей замовлення, підготовка замовлення до виконання, упакування та відвантаження продукції замовнику.

Діяльність "Складання звітів" описує процес збору та обробки даних для підготовки різних типів звітів. Це включає визначення типу звіту, збір необхідних даних, аналіз та обробку даних, складання самого звіту, перегляд, перевірку та затвердження його. Сформований звіт може бути розповсюджений зацікавленим сторонам.

Всі рівні взаємодіють між собою та утворюють функціональну модель, описану у контекстній діаграмі "Організація замовлення продукції ТОВ "Пирятинський сирзавод"". Вони дозволяють організації керувати процесом замовлення, виконання та обслуговування, забезпечуючи задоволення замовників та оптимальне використання ресурсів.

Проаналізувавши ці рівні можна прийти до висновків, та дати пропозиції щодо покращення робота з замовниками продукції підприємства.

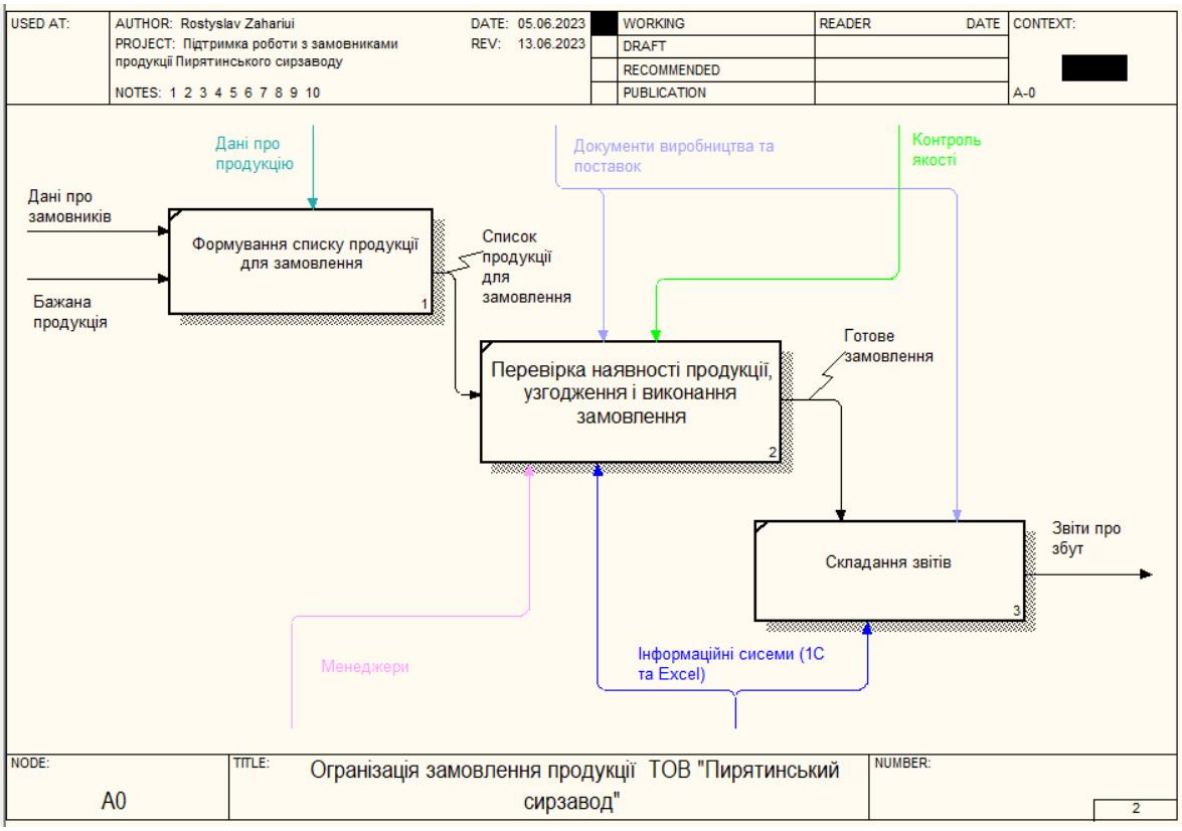



Рис.1.3. Декомпозиція функціональної моделі

### 1.4.2. Виявлені проблеми

На сьогоднішній день, на жаль, не існує офіційного веб-ресурсу, де користувач може не тільки переглянути інформацію про ціни на продукцію Пирятинського сир заводу, а й замовити її в он-лайн режимі. Це може створювати певні незручності та обмежувати доступ замовників до інформації, а також не дозволяє формувати замовлення у зручний спосіб.

Відсутність офіційного веб-ресурсу з можливістю формування он-лайн замовлень ускладнює та затримує процес купівлі продукції для потенційних замовників підприємства, що може викликати недоотримання прибутків.

### 1.4.3. Задачі автоматизації

Для Пирятинського сирзаводу, з метою автоматизації та поліпшення бізнес-процесу збуту, необхідно вирішити наступні задачі:

- створення офіційного ресурсу, де користувачі зможуть знайти повну та актуальну інформацію про продукцію сирзаводу, включаючи ціни, опис, характеристики та фотографії;

- розробка зручного та інтуїтивно зрозумілого інтерфейсу, що дозволить користувачам швидко знайти потрібну продукцію та переглянути її характеристики та ціни.
- створення інтерфейсу, який дозволить користувачам зробити замовлення он-лайн у будь який час.
- відображення статистики по он-лайн замовленням, популярності на продукції за певний період, визначення оцінки на продукцію для виявлення недоліків та проведення аналізу.

## 1.5. Огляд існуючих рішень

Проведений детальний аналіз різноманітних інформаційних систем, доступних на українському ІТ-ринку. Будуть досліджені окремі підрозділи кожної системи та надані характеристики їх функціональних можливостей, сумісності з іншими системами, мови інтерфейсу та інші фактори. Крім того, буде проведене порівняння між системами-аналогами залежно від обраних критеріїв, таких як вартість, швидкодія та наявність специфічних функцій. Цей огляд допоможе виявити найбільш підходящі рішення для вирішення проблем, з якими стикається Пирятинський сирзавод у роботі зі своїми замовниками.

Можна розглянути декілька популярних платформ та рішень, які можуть бути використані для розробки системи для підтримки роботи з замовниками продукції Пирятинського сирзаводу.

Розглянемо та порівняємо функціональні можливості систем-аналогів:

- WordPress - це одна з найпопулярніших платформ для створення веб-сайтів та блогів. Він має широкий вибір тем та плагінів, що дозволяє налаштувати функціонал під конкретні потреби сирзаводу.
- Shopify - це платформа електронної комерції, яка спеціалізується на створенні веб-магазинів. Вона надає зручний інтерфейс для додавання продуктів, обробки замовлень та керування запасами.
- Magento - це потужна платформа електронної комерції, яка підходить для великих та складних проектів. Вона має розширені можливості для каталогування продуктів, управління замовленнями та маркетингу.

- OpenCart - це легка у використанні система електронної комерції, яка дозволяє швидко створити веб-магазин. Вона має простий інтерфейс та базовий набір функцій для продажу продуктів онлайн.

Далі біль детально розглянемо ці системи щоб визначитися яка буде найбільш підходити під наші потреби.

WordPress - це веб-платформа з відкритим вихідним кодом, яка початково розроблялася як система для створення блогів, але з часом перетворилася на потужний інструмент для будь-якого типу веб-сайту [6], рисунок 1.4.

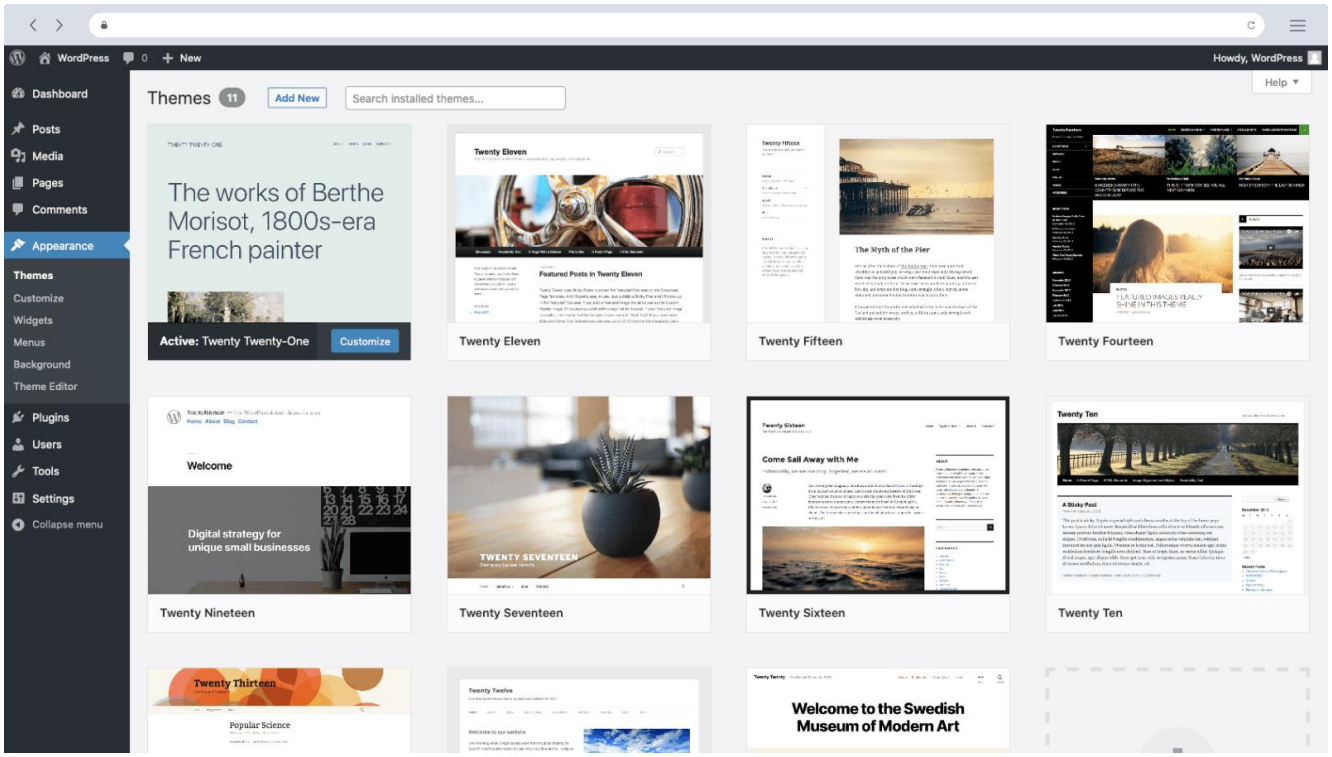



Рис 1.4 Інтерфейс WordPress

Його успіх та популярність серед користувачів полягає в кількох ключових перевагах:

- Легкість використання: він пропонує дружній та інтуїтивно зрозумілий інтерфейс, що дозволяє навіть не-технічним користувачам легко створювати, редагувати та керувати вмістом своїх веб-сайті.
- Гнучкість та розширюваність: завдяки широкому вибору безкоштовних і преміальних тем і плагінів, WordPress дозволяє налаштувати зовнішній вигляд та функціонал вашого веб-сайту за вашими унікальними потребами.

Розглянувши недоліки WordPress, слід відзначити такі аспекти:

- Швидкодія: велика кількість плагінів та тем, які доступні для WordPress, може вплинути на швидкодію веб-сайту
- Потенційні проблеми зі сумісністю: при використанні багатьох плагінів і тем, можуть виникати проблеми зі сумісністю між ними.

Shopify - це потужна платформа електронної комерції, яка дозволяє створити та управляти професійним онлайн-магазином без необхідності в глибоких технічних знаннях або програмуванні [7], з інтерфейсом можна ознайомитись на рисунку 1.5.

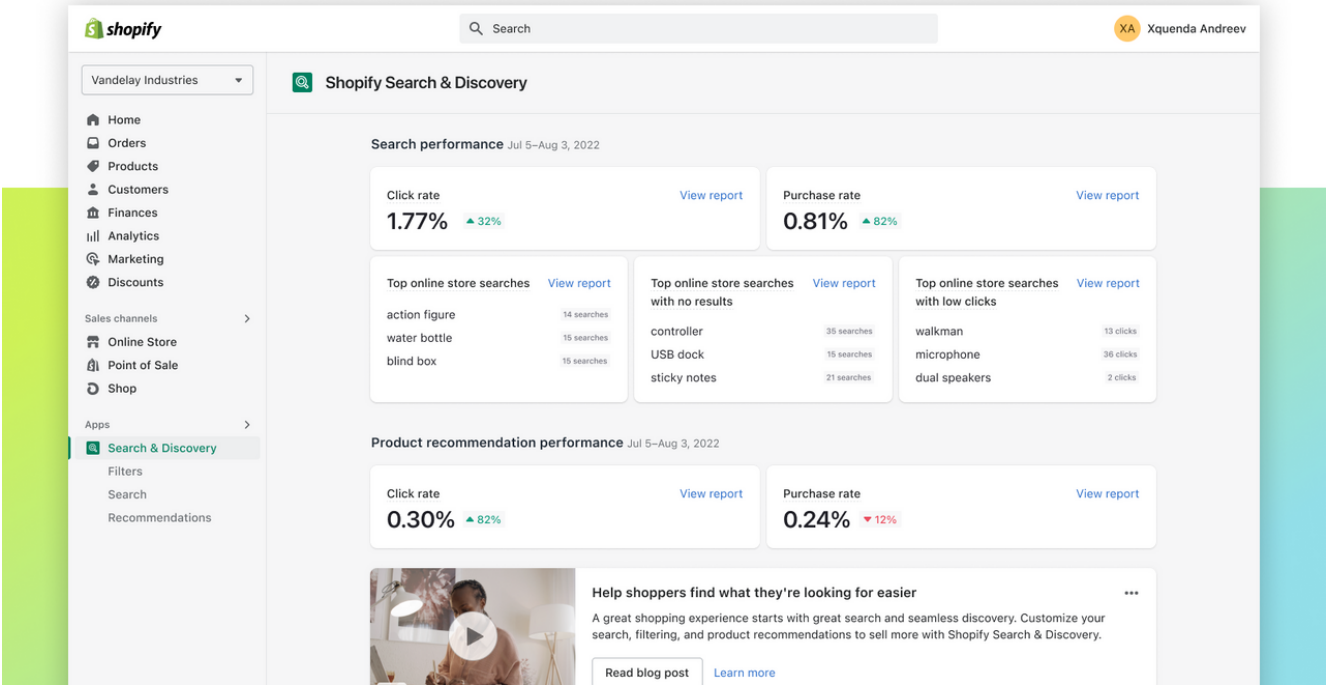



Рис. 1.5. Інтерфейс Shopify

Перейдемо до переваг Shopify, серед них можна виділити такі як:

- Простота використання: однією з головних переваг Shopify є його легкість використання. Ви можете швидко створити магазин, додати свої товари та налаштувати оплату та доставку.
- Готовий до використання функціонал: Shopify має широкий спектр вбудованих функцій, які дозволяють вам ефективно управляти своїм магазином.

- Розширення та теми: Shopify пропонує велику кількість безкоштовних та платних тем, що дозволяють вам налаштувати вигляд вашого магазину відповідно до вашого бренду.

Незважаючи на багато переваг, слід відмітити, що Shopify є платною платформою, і вартість може залежати від плану та додаткових послуг, які ви виберете. Крім того, деякі функціональні можливості можуть вимагати використання платних розширень.

Magento - це потужна відкрита платформа електронної комерції, яка надає гнучкість та розширюваність для створення розширених та високопродуктивних онлайн-магазинів, рисунок 1.6.

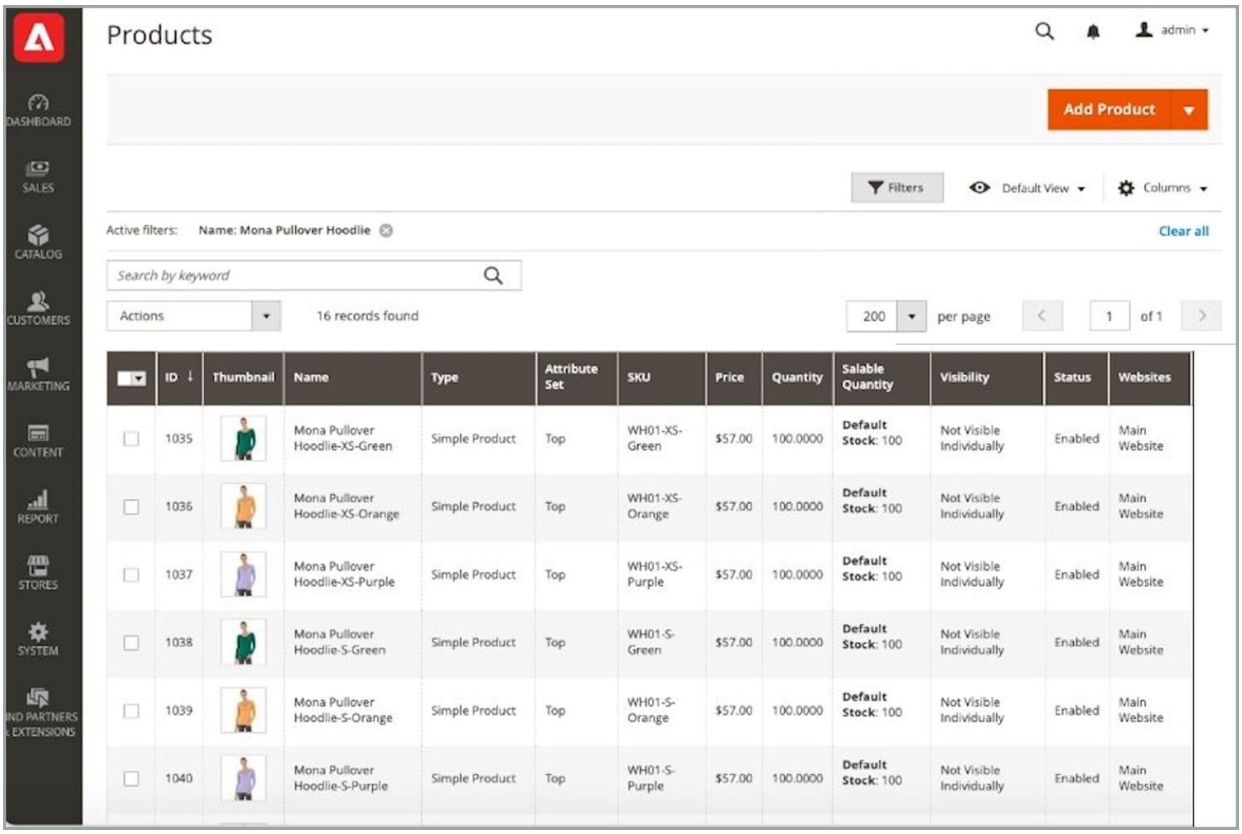



Рис. 1.6 Інтерфейс Magento

Magento пропонує широкий набір функціональності, що дозволяє вам налаштувати і керувати всіма аспектами вашого електронного магазину. Він має потужну систему управління контентом, що дозволяє вам легко додавати, редагувати та керувати вмістом вашого магазину. Крім того, Magento надає широкі можливості управління товарами, замовленнями, інвентарем, оплатою та доставкою [8]. До переваги входять:

- Гнучкість та розширюваність: ви можете використовувати тисячі розширень та плагінів, що дозволяють додавати нові функції та розширювати можливості вашого магазину. Це означає, що ви можете налаштувати магазин під свої унікальні потреби та вимоги, забезпечивши найкращий досвід для своїх клієнтів.
- Потужні інструменти для маркетингу та аналітики, що дозволяють вам відстежувати продажі, конверсії та поведінку клієнтів. Ви можете налаштувати промоакції, розсилки, сегментування клієнтів та багато іншого, щоб підвищити свою ефективність та залучити більше покупців.

На жаль, однією з недоліків Magento є складність його налаштування та використання, особливо для не-технічних користувачів. Він вимагає наявності досвіду веб-розробки та технічних знань для повного розуміння та використання його можливостей. Крім того, Magento вимагає потужного хостингу та належної налаштування сервера для забезпечення оптимальної продуктивності.

OpenCart - це безкоштовна платформа електронної комерції з відкритим вихідним кодом, призначена для створення та управління онлайн-магазинами. Вона має широкі можливості та простоту використання, що робить її популярним вибором для невеликих і середніх підприємств [9].

OpenCart надає зручний інтерфейс рисунок 1.8, що дозволяє легко створювати та налаштовувати магазини. Ви можете швидко додавати товари, категорії, змінювати дизайн та налаштовувати способи оплати та доставки. Його простий інтерфейс робить його доступним навіть для користувачів без глибоких технічних знань, перейдемо до переваг:

- Одна з переваг OpenCart - це його розширюваність та додаткові модулі. Ви можете використовувати широкий спектр додатків та розширень, щоб додати нові функції та можливості до вашого магазину
- широкий вибір тем та шаблонів для створення привабливого веб-дизайну вашого магазину. Ви можете вибрати тему, яка відповідає вашому бренду та надає гарний дизайн для вашого магазину.

Незважаючи на переваги, OpenCart як і інші аналоги також має деякі недоліки:

- Масштабованість: OpenCart може виявитися менш ефективним для великих та складних проектів.
- Нижчий рівень функціональності: у порівнянні з деякими іншими економічними платформами електронної комерції, OpenCart може бути обмежений у своїх функціях та можливостях.
- Підтримка та оновлення: OpenCart має меншу спільноту користувачів та розробників порівняно з більшими платформами, такими як Magento або Shopify.

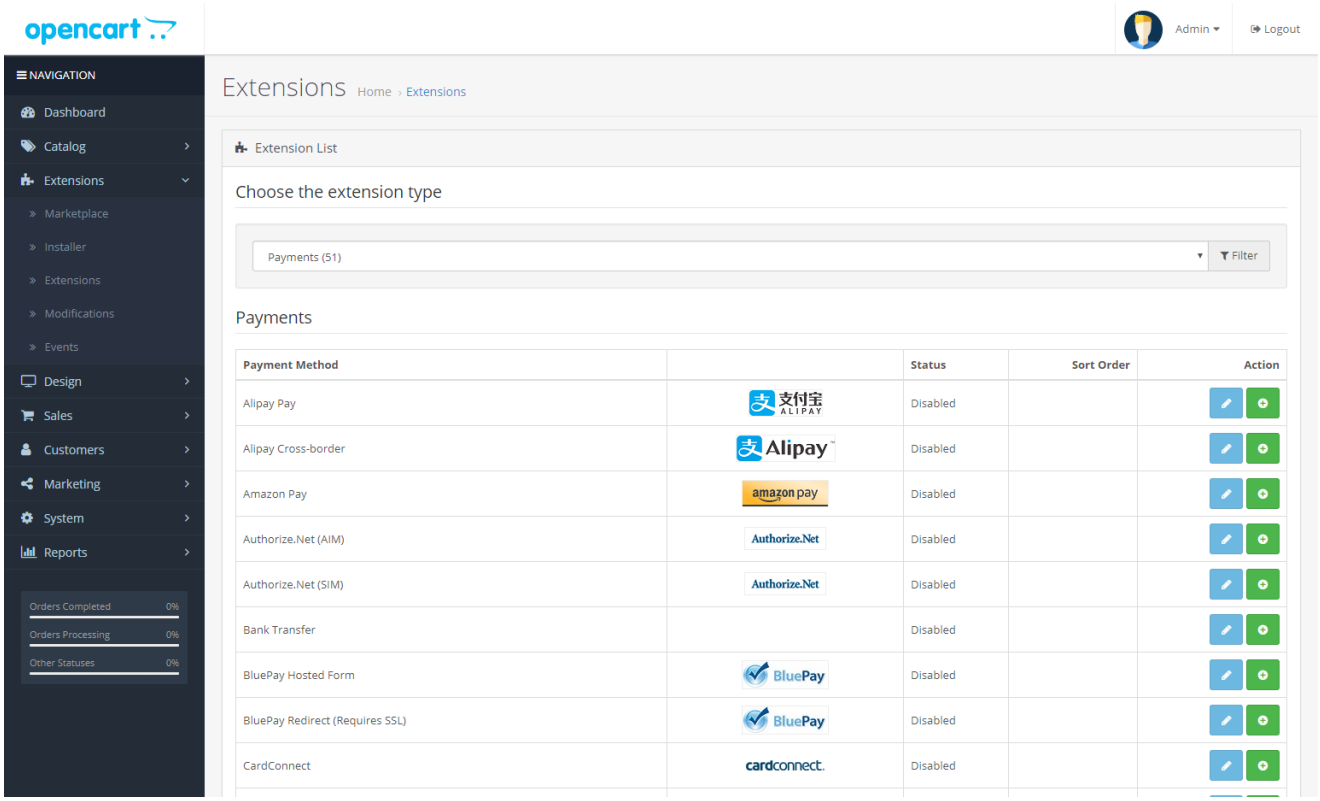



Рис. 1.7. Інтерфейс OpenCart

У таблиці 1.1 можна більш детально ознайомитися з оцінками кожної системи за певними характеристиками.

*Таблиця 1.1. Порівняльна характеристика*

| **Характеристика** | **WordPress** | **Shopify** | **Magento** | **OpenCart** |
|---|---|---|---|---|
| Вартість | 4/5 | 3/5 | 3/5 | 5/5 |
| Легкість | 4/5 | 5/5 | 4/5 | 5/5 |

| | | | | |
|---|---|---|---|---|
| використання | | | | |
| Розширюваність | 5/5 | 5/5 | 5/5 | 3/5 |
| Гнучкість | 4/5 | 3/5 | 4/5 | 3/5 |
| Вибір тем і шаблонів | 5/5 | 5/5 | 5/5 | 3/5 |
| Масштабованість | 3/5 | 4/5 | 5/5 | 4/5 |
| Функціональні можливості | 4/5 | 5/5 | 5/5 | 3/5 |
| Підтримка та оновлення | 5/5 | 5/5 | 5/5 | 3/5 |
| Технічна складність | 2/5 | 2/5 | 4/5 | 2/5 |
| Спільнота користувачів | 5/5 | 5/5 | 5/5 | 3/5 |
| Безпека | 5/5 | 5/5 | 5/5 | 3/5 |

Незважаючи на те, що фаворитом з таблиці порівняння є Shopify, кращим варіантом може бути розробка власного web-додатку для підтримки роботи з замовниками продукції Пирятинського сирзаводу. Ось кілька прикладів, чому самостійна розробка може бути кращим рішенням.

Унікальні потреби: розробка власного додатку дозволить вам точно врахувати унікальні потреби сирзаводу та його замовників. Ви зможете впровадити спеціалізовані функції, які найкраще відповідатимуть вашим вимогам і допоможуть вам надати високоякісні послуги.

Брендування та ідентичність: власний web-додаток дасть вам повний контроль над дизайном і брендуванням. Ви зможете створити унікальний вигляд, що буде відображати ідентичність сирзаводу і створить незабутнє враження на замовників.

Гнучкість та масштабованість: власний додаток дозволить вам гнучко реагувати на змінюючіться потреби та розширювати функціонал залежно від

росту вашого бізнесу. Ви зможете внести зміни, оновити функціональність та адаптувати додаток до ваших потреб у будь-який момент.

Контроль над даними та безпека: розробка власного додатку дає вам повний контроль над даними замовників. Ви зможете забезпечити високий рівень безпеки, дотримуючись всіх необхідних стандартів та вимог щодо захисту даних.

## 1.6. Обґрунтування доцільності впровадження Web-додатку для підтримки роботи з замовниками продукції Пирятинського сирзаводу

Впровадження Web-додатку для підтримки роботи з замовниками продукції Пирятинського сирзаводу має ряд переконливих аргументів і обґрунтовану доцільність. Ось кілька причин, чому варто розглянути таке впровадження:

Зручне замовлення: Web-додаток наддасть можливість звичайним користувачам замовити продукцію онлайн у зручний для них час. Вони зможуть швидко вибрати потрібні продукти зі списку, ознайомитися з цінами та характеристиками, а також вибрати необхідну кількість. Це спростить процес замовлення та забезпечить зручність для користувачів.

Розширення асортименту: Web-додаток дозволить користувачам пропонувати свою сировину, таку як молоко або сироватка, для використання на сирзаводі. Це відкриє нові можливості для співпраці та розширить асортимент продукції сирзаводу, задовольняючи потреби замовників.

Оцінка продукції: Web-додаток дасть можливість користувачам виставляти оцінки продуктам на основі 5-бальної системи. Це забезпечить зворотний зв'язок та дозволить сирзаводу виявити популярні продукти, а також виявити потенційні недоліки та зробити вдосконалення у виробництві.

Діаграма використання системи (Use Case Diagram) є візуальним засобом моделювання функціональних вимог до системи, який показує, які дії можуть виконувати актори (користувачі) і які функції системи вони можуть викликати. Більш детальне використання системи можна подиватиня у Додатоку А.

## 1.7. Концептуальна модель системи

Для більш детальної користі від провадження нашої системи, а саме web-додатку для підтримки роботи з замовниками продукції Пирятинського сирзаводу розробимо модель TO-BE з використанням нашої системи. Рис.1.8.

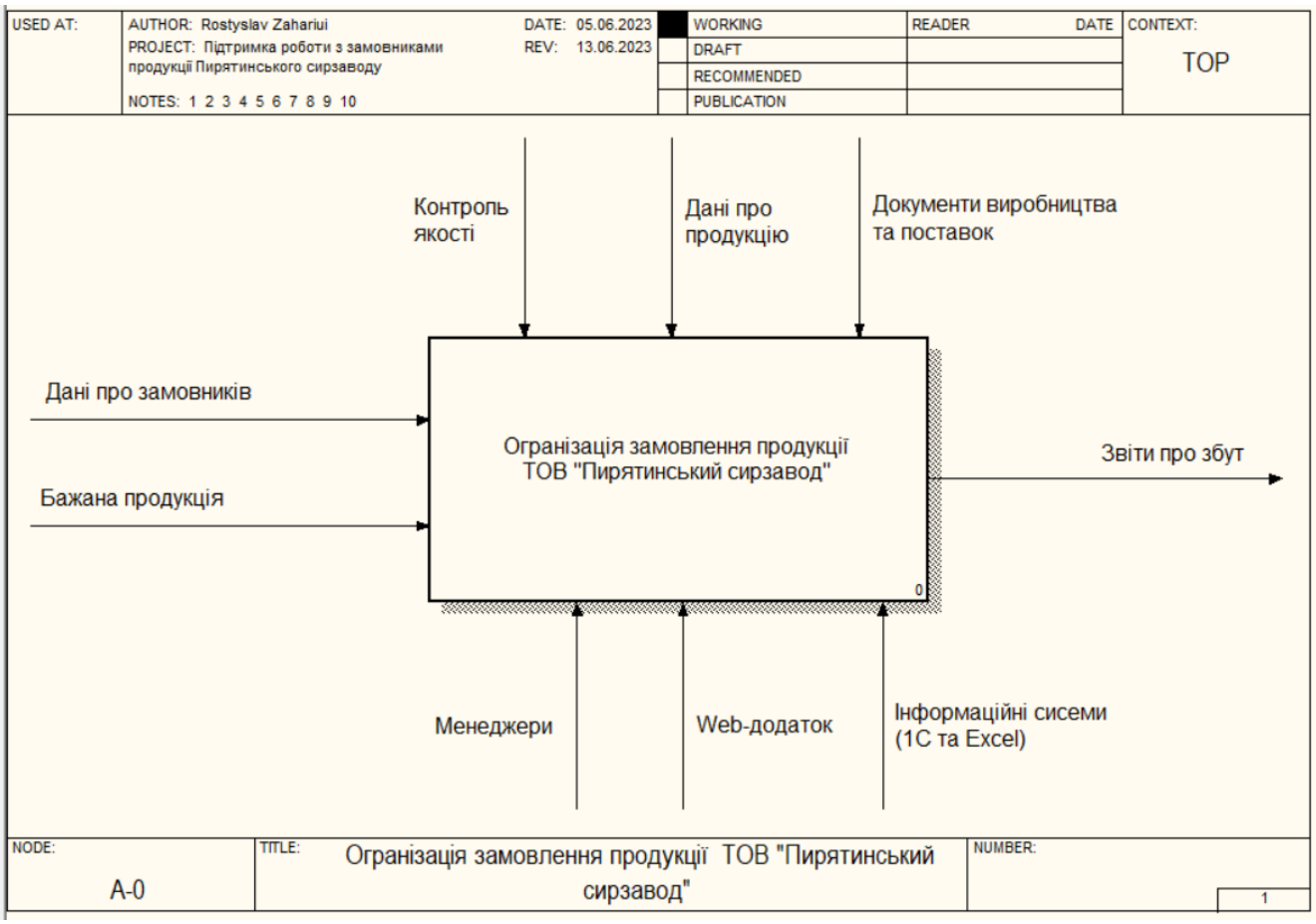



Рис.1.8. Контекстна діаграма TO-BE

Можливості моделі після впровадження системи дещо збільшились, рисунок 1.9. А саме запис замовлення здійснюється автоматично на стороні користувача, тепер менеджери повинні тільки зв’язатися з ним для підтвердження, менеджерам не слід дивитися кількість продукції у складських звітах, слід лише поповнювати її кількість у редагуванні продукта. Також користувачі зможуть оцінювати продукцію для подальшого аналізу, та запропонувати свою продукцію для підприємства.

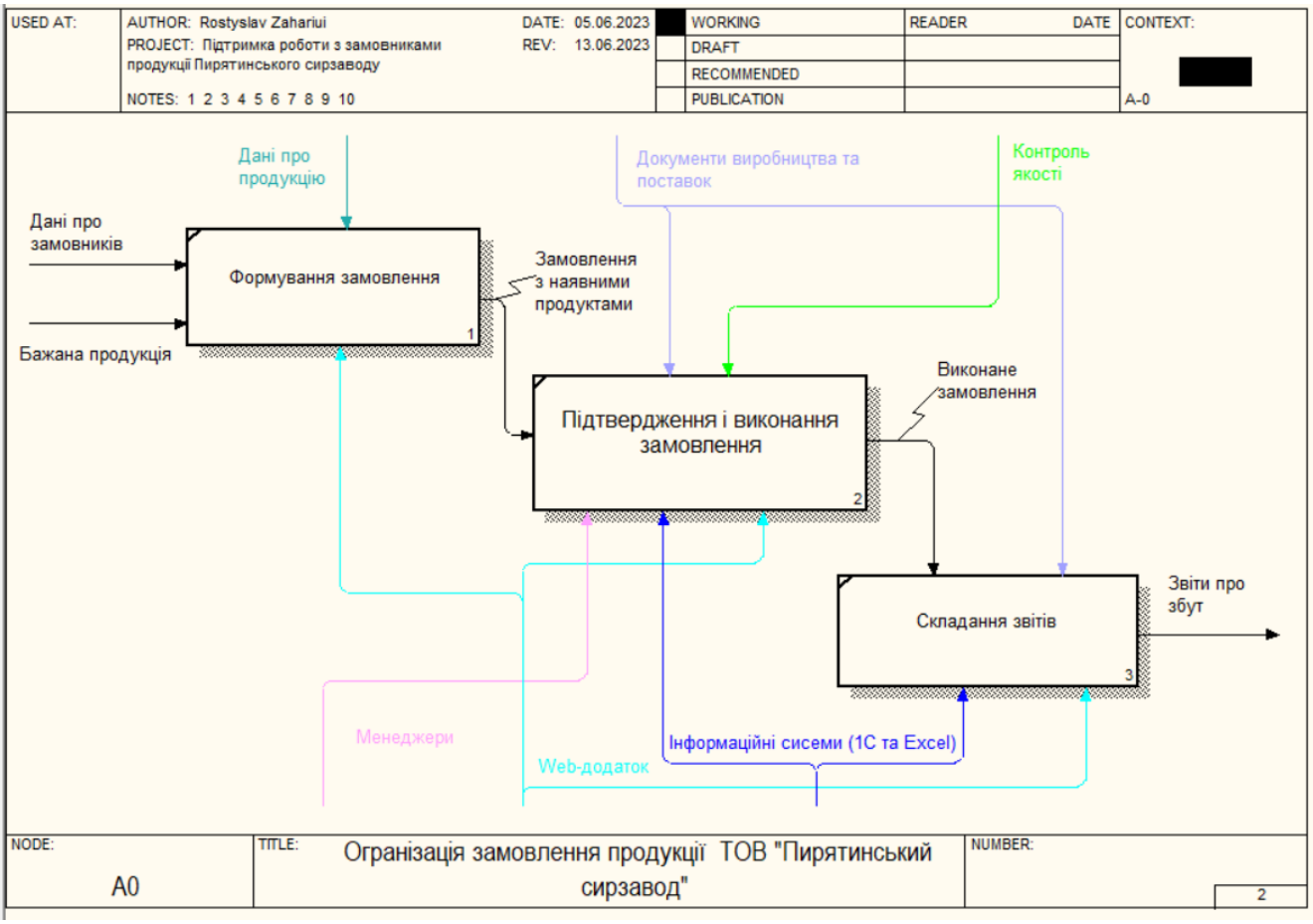



Рис.1.9. Декомпозиція моделі TO-BE

## 1.8. Розрахунок економічного ефекту від впровадження системи

Розрахунок економічного ефекту від впровадження системи є важливим кроком, оскільки дозволяє оцінити потенційну вигоду та прибутковість інвестицій, а також забезпечує обґрунтування ресурсів та розумне прийняття рішень. Точний розрахунок допомагає зрозуміти, наскільки ефективно впровадження системи вплине на економіку та фінансовий стан організації, що дозволяє забезпечити успіх проекту та оптимізувати використання ресурсів [10].

За методикою зазначеному у джерелі [11] в першу чергу розрахуємо витрати на оплату праці.

Ступінь новизни розроблюваних задач «В» – використання типових проектних рішень, що мають аналогічні рішення.

Група складності алгоритму – 2 (алгоритми обліку, звітності, статистики, пошуку).

Визначення інформації, що використовується, наведено у таблиці 1.2.

*Таблиця 1.2. Визначення виду інформації*

| Вид інформації | Позначення | К-сть наборів даних |
|---|---|---|
| Змінна | ЗІ | 4 |
| Нормативно-довідкова | НДІ | 5 |
| Банк (база) даних | БД | 1 |
| Обробка в режимі реального часу | РЧ | Так |
| Забезпечення телекомунікаційної обробки даних і управління віддаленими об’єктами | ТОУ | Ні |

У таблиці 1.3. наведені витрати на розробку проекту

*Таблиця 1.3. Витрати на розробку проекту*

| Вид системи | Стадія розробки системи | |
|---|---|---|
| | Ескізний проект | Технічне завдання |
| Управління якістю продукції, управління технологічними процесами, управління стандартизацією, управління технічною підготовкою виробництва | В | В |
| | $T_1$=67 | $T_2$=31 |

Тепер потрібно визначити витрати часу на стадіях «технічний проект», «робочий проект» і «впровадження».

Вхідними даними є

- кількість форм вхідної інформації 4;
- кількість форм вихідної інформації 5;
- базове значення витрат часу для стадії «Технічний проект» $T_{Б3}$ = 139;
- базове значення витрат часу для стадії «Робочий проект» $T_{Б4}$ = 217;
- базове значення витрат часу для стадії «Впровадження» – $T_{Б5}$ = 73;

Витрати на розробку «технічного проекту» $T_з$ визначаються за (1.1).

$$T_з = T_{Б3} * k_п * k_о \quad (1.1)$$

де, $Т_{Б3}$ – базове значення витрат часу для стадії «Технічний проект», яке визначається залежно від виду розроблюваної системи, $k_о$ – коефіцієнт новизни, $k_п$ – коефіцієнт трудомісткості робіт, що визначається за (1.2).

$$k_П = \frac{k_1 * m + k_2 * n + k_3 * p}{m + n + p} \quad (1.2)$$

де $k_1$, $k_2$, $k_3$ – поправочні коефіцієнти, m – кількість видів змінної інформації, n – кількість видів нормативно-довідкової інформації , p – кількість видів баз даних.

Для нашого проекту, ступінь новизни становить 1.26, так як у нас присутній вид обробки в реальному часі.

Поправочні коефіцієнти будуть становити для $k_1$ – 1.0, $k_2$ – 0.72 та $k_3$ – 2.08.

$$k_П = \frac{1 * 4 + 0.72 * 5 + 2.08 * 1}{4 + 5 + 1} = 0{,}968$$

$$Т_з = 139 * 0{,}968 * 1{,}26 = 169{,}53$$

Витрати часу на стадіях «робочий проект» $Т_4$ і «впровадження» ($Т_5$) визначається за формулою (1.3).

$$Т_4 = Т_{Б4} * k_п * k_о * k_с \quad (1.3)$$

де, $Т_{Б4}$ – базове значення витрат часу для стадії «Технічний проект», яке визначається залежно від виду розроблюваної системи, $k_о$ – коефіцієнт новизни, $k_п$ – коефіцієнт трудомісткості робіт, а $k_с$ – коефіцієнт складності контролю вхідної та вихідної інформації визначається з таблиці.

Тому поправочні коефіцієнти будуть становити для $k_1$ – 1.1, $k_2$ – 0.58 та $k_3$ – 0,48, коефіцієнт новизни 1.32, а коефіцієнт складності контролю вхідної та вихідної інформації для нашого проекту буде становити 1.08.

$$k_П = \frac{1.1 * 4 + 0.58 * 5 + 0{,}48 * 1}{5 + 4 + 1} = 0{,}778$$

$$Т_4 = 217 * 0{,}778 * 1{,}32 * 1{,}08 = 240{,}67$$

Для розрахунку витрат часу на стадії «впровадження» (1.4) маємо такі коефіцієнти: $k_1$ – 1.2, $k_2$ – 0.65 та $k_3$ – 0.54, коефіцієнт новизни 1.21, коефіцієнт

складності контролю вхідної та вихідної інформації для нашого проекту буде становити 1.08.

$$T_5 = T_{Б5} * k_п * k_о * k_с \quad (1.4)$$

$$T_5 = 73 * 0,778 * 1,21 * 1,08 = 74,21$$

Отже, загальні витрати складають:

$$67 + 31 + 169,52 + 240,67 + 74,67 = 582,81$$

Тепер визначимо чисельність виконавців Ч за формулою 1.5

$$Ч = \frac{T_Σ}{Ф} \quad (1.5)$$

де Ф – кількість робочих днів на виконання проекту

Тому, враховуючи, що на виконання кваліфікаційної роботи дається 360 годин із 7-годинним робочим днем, тому на розробку виділено:

$$Ф = 360 / 7 = 51.42 \text{ дні}$$

Якщо брати час в місяцях, то визначаємо кількість місяців із розрахунку 22 робочих дні.

$$М = Ф / 22 = 51,42 / 7 = 2,34 \text{ місяці}$$

Отже на виконання такого проекту потрібна чисельність виконавців:

$$Ч = \frac{582,81}{51,42} = 11,33$$

Оплата праці виконавців підраховується за формулою 1.6.

$$V_1' = Ч * М * ЗП_{ПР} \quad (1.6)$$

де $ЗП_{ПР}$ – місячна оплата праці програміста, Ч – чисельність виконавців, М – кількість місяців роботи.

Приймемо розмір заробітної плати програміста 30000 грн, тоді загальна сума заробітних плат програмістів складає:

$$V_1' = 11 * 2.34 * 30000 = 772200 = грн$$

Наступним пунктом витрат є витрати пов’язані з розробкою програми.

Дійсний річний фонд часу ПК у годинах дорівнює числу робочих годин у році для оператора, за винятком часу на технічне обслуговування і ремонт ПК (в середньому 5 год/міс + 6 робочих днів/рік)

$$T_{ПК} = 2000 - (6 * 8 + 5 * 12) = 1892 \text{ год}$$

Оскільки під час виконання кваліфікаційної роботи здобувач в середньому витрачає 300 годин машинного часу, то величина фонду часу ПК:

$$Т_{ПК} = 1892 * (300 / 2000) = 283,8 \text{ год}$$

За формулою 1.7 розрахуємо поточні витрати на експлуатацію

$$V_1^{"} = З_{ОП} + З_{АМ} + З_{ел} + З_Р \quad (1.7)$$

де $З_{оп}$ – заробітна плата обслуговуючого персоналу, $З_р$ – витрати на поточний ремонт і технічне обслуговування комп'ютера, $З_{мат}$ – непрямі витрати пов'язані з експлуатацією комп'ютера, $З_{ам}$ – амортизаційні відрахування, що обраховується за формулою 1.8.

$$З_{АМ} = \frac{Ц_{пк}}{Н_а} \quad (1.8)$$

де $Ц_{пк}$ – балансова вартість комп'ютера, $Н_а$ – норма амортизаційних відрахувань, яка для комп'ютера 5.

Балансова вартість ПК обраховується за формулою 1.9.

$$Ц_{пк} = Ц_р * (1 + k_{ун}) \quad (1.9)$$

де $Ц_р$ – ринкова вартість ПК, орієнтовно складає 40000 грн, $k_{ун}$ – коефіцієнт, що враховує витрати на установку комп'ютера, складає 0,12. Тому балансова вартість комп'ютера складає:

$$Ц_{пк} = 40000 * (1 + 0{,}12) = 44800$$

Амортизаційні відрахування будуть становити

$$З_{АМ} = \frac{44800}{5} = 8960 \text{ грн}$$

Витрати на електроенергію споживану комп'ютером, обчислюються за формулою 1.10.

$$З_{ел} = Р_{пк} * Т_{пк} * Ц_{ел} * А \quad (1.10)$$

де $Р_{пк}$ – потужність ПК, $Т_{пк}$ – фонд корисного робочого часу, $Ц_{ел}$ – вартість 1 кВт електроенергії для підприємств, А – коефіцієнт інтенсивного використання ПК, складає 0,9.

Враховуючи потужність всіх комп'ютерів 0,4 кВт, ціна на електроенергію 1,86 грн/кВт, витрати на електроенергію складають:

$$З_{ел} = 0{,}4 * 283{,}8 * 1{,}86 * 0{,}9 = 190\ грн$$

Витрати на поточний ремонт складають і технічне обслуговування комп'ютера визначаються як 6% від балансової вартості комп'ютера.

$$З_р = 44800 * 0{,}06 = 2688 \text{ грн}$$

Непрямі витрати, пов'язані з експлуатацією комп'ютера визначаються як 5% від балансової вартості комп'ютера.

$$З_{мат} = 44800 * 0{,}05 = 2240 \text{ грн}$$

Заробітна плата обслуговуючого персоналу в середньому становить 10000 грн, то поточні витрати на експлуатацію будуть становити

$$V_1^{"} = 10000 + 8960 + 190 + 2688 + 2240 = 24078\ грн$$

А загальні витрати

$$V_1 = V_1^{`} + V_1^{"} = 772200 + 24078 = 796278\ грн$$

Наступним кроком є розрахунок витрат на підготовку приміщення та навчання персоналу.

Витрати на підготовку приміщення відсутні – $V_3 = 0$, так як приміщення вже присутнє.

Витрати на навчання персоналу – можна припустити, що навчання буде тривати один місяць, тому можна сказати $V_4 = 4500$ грн.

Витрати на придбання та встановлення комп'ютера дорівнюють балансовій вартості комп'ютера, тому $V_2 = Ц_{пк} = 44800$ грн.

Загальна вартість розробки і впровадження системи вираховуються за формулою 1.11

$$V_\Sigma = V_1 + V_2 + V_3 + V_4 \qquad (1.11)$$

$$V_\Sigma = 796278 + 44800 + 0 + 4500 = 845578\ грн$$

Оскільки норма амортизаційних втрат для комп'ютерних систем $Н_А = 5$, то для обрахування річного економічного ефекту слід брати до розгляду величину:

$$V_p = \frac{V_\Sigma}{H_A} = \frac{845578}{5} = 169115{,}6\ грн$$

Термін окупності визначається за формулою 1.12

$$T_{ок} = \frac{1}{K_{ЕФ}} \quad (1.12)$$

де $K_{еф}$ економічної ефективності визначається за формулою 1.13

$$K_{ЕФ} = \frac{П_р}{V_р} \quad (1.13)$$

де $П_р$ – річний прибуток від впровадження системи, який можна орієнтовно оцінити в 90000 грн, тому економічний ефект від впровадження складатиме

$$K_{ЕФ} = \frac{90000}{169115{,}6} = 0{,}53$$

Відповідно термін окупності

$$T_{ок} = \frac{1}{0{,}53} = 1{,}88\ роки$$

# РОЗДІЛ 2. ТЕХНІЧНЕ ЗАВДАННЯ

На основі дослідженого підприємства складаємо технічне завдання [12].

## 2.1. Загальні положення

2.1.1. Найменування системи: «Web-додаток для підтримки роботи з замовниками продукції Пирятинського сирзаводу».

2.1.2. Відповідно до ДСТУ [13], результати розробки системи мають бути представлені відповідно до вимог, що ставляться на кожному етапі розробки. Процес подання і передачі цих результатів має бути визначений відповідно до сутності робіт і графіка їх виконання.

2.1.3 В процесі подальших стадій розробки системи може відбутися уточнення та доробка окремих аспектів, якщо це вимагатиметься.

## 2.2. Призначення та цілі створення системи

2.2.1. Призначення створення системи: поліпшення і автоматизації процесів, пов'язаних зі замовленням, оцінюванням та пропонуванням продукції та їх аналіз.

2.2.2. Цілі створення системи:

- Автоматизація роботи з замовниками: Система автоматизує процеси замовлення продукції, що забезпечує швидке та ефективне обслуговування клієнтів. Вона забезпечує одночасне виконання кількох замовлень, що підвищує продуктивність роботи сирзаводу.
- Ефективне управління продуктами: Менеджери можуть легко редагувати інформацію про продукти, ціни та знижки, що дозволяє швидко реагувати на зміни на ринку.
- Аналітика та прогнозування продажів: Система надає менеджерам важливі графіки продажів та інші аналітичні дані, що допомагають приймати обґрунтовані бізнес-рішення.
- Підвищення задоволеності клієнтів: Користувачі мають можливість оцінювати продукцію та пропонувати свою сировину. Це забезпечує високий рівень взаємодії з клієнтами та збільшує їхнє задоволення від співпраці з сирзаводом

## 2.3. Характеристика об’єкта автоматизації

Об’єктом автоматизації є діяльність відділу збуту ТОВ «Пирятинський сирзавод».

## 2.4. Вимоги до системи

2.4.1. Вимоги до системи в цілому.

- Автентифікація та авторизація: Система повинна мати механізм автентифікації користувачів, який забезпечує перевірку їхньої ідентичності перед наданням доступу до функціоналу. Крім того, система повинна мати механізм авторизації, який контролює права доступу користувачів до різних функцій на основі їхніх ролей (Адмін, Користувач, Менеджер)
- Управління продуктами: Менеджер повинен мати можливість редагувати ціни, знижки та іншу інформацію про продукти. Система повинна забезпечувати зручний інтерфейс для виконання цих завдань.
- Аналітика та звіти: Менеджер повинен мати доступ до різних графіків та звітів. Це включає графіки продажів за певний період, графіки популярності продуктів (який тип сиру купують найбільше), графіки продуктів з найнижчим рейтингом для приділення їм уваги та графіки, які продукти заповзяють найбільше. Система повинна обробляти дані та генерувати звіти в зручному форматі.
- Обробка заявок користувачів: Менеджер повинен мати можливість переглядати заявки користувачів та отримувати інформацію про замовника. Система повинна забезпечувати простий спосіб взаємодії з заявками, включаючи можливість перегляду деталей замовлення, його статусу, контактної інформації замовника та іншої необхідної інформації. Менеджер повинен мати можливість здійснювати дії зі заявками, наприклад, приймати, відхиляти або змінювати їх статус.
- Каталог продуктів: Система повинна мати зручний інтерфейс для перегляду доступної продукції сирзаводу. Користувачі повинні мати можливість переглядати детальну інформацію про продукти, включаючи опис, ціни, знижки, рейтинги та наявність.

- Захист даних: Система повинна забезпечувати надійний рівень захисту персональних даних користувачів, включаючи конфіденційні дані про замовлення та платежі. Вона повинна відповідати сучасним стандартам безпеки даних та включати заходи для запобігання несанкціонованому доступу до інформації.
- Мобільна сумісність: Для забезпечення більш широкого доступу користувачів, система може бути розроблена з урахуванням мобільної сумісності. Це може означати розробку мобільного додатку або адаптацію веб-додатку для зручного відображення та використання на мобільних пристроях.

2.4.2. Функціональні вимоги до веб-додатку.

Веб-додаток розроблений таким чином, щоб задовольняти потреби різних користувачів. Він має гнучку архітектуру, що дозволяє адаптуватися до ролі кожного користувача, надаючи зміст, специфічний для їхніх потреб. Залежно від того, чи є вони споживачами продукції, чи аналітиками, чи постачальником сировини, веб-додаток надає відмінний набір функціональності та вмісту. Це гарантує, що кожен користувач отримає відповідну інформацію та можливості, необхідні для їхньої ролі в процесі використання додатку.

Перелік функцій, які повинна виконувати система показаний в таблиці 2.1.

*Таблиця 2.1. Перелік функцій, вхідної та вихідної інформації*

| № п/п | Найменування функції | Вхідна інформація | Вихідна інформація |
|---|---|---|---|
| 1 | Авторизація користувача | Таблиця БД «Користувач» | Форми для авторизації |
| 2 | Перегляд продукції | Таблиця БД «Продукт» | Сторінка списку продукції |
| 3 | Оцінка продукції | Таблиця БД «Відгук» | Поле виставлення оцінки |
| 4 | Формування замовлення | Таблиця БД «Продукт» Таблиця БД «Користувач» | Список продукції у кошику та форма для створення замовлення |

| № п/п | Найменування функції | Вхідна інформація | Вихідна інформація |
|---|---|---|---|
| 5 | Формування заявки на поставку сировини | Таблиця БД «Тип продукту» | Форма для створення заявки на поставку |
| 6 | Перегляд заявок на поставку сировини | Таблиця БД «Замовлення на поставку сировини» | Список заявок на поставку сировини |
| 7 | Введення/редагування продукції | Таблиця БД «Тип продукту» Таблиця БД «Продукт» | Форма редагування/створення продукту |
| 8 | Введення типу продукту | Таблиця БД «Продукт» | Форма для створення типу продукта |
| 9 | Введення сировини для закупівлі | Таблиця БД «Тип продукту» | Форма для створення сировини |
| 10 | Введення нових менеджерів та амінів | Таблиця БД «Користувач» | Форма створення користувача |
| 11 | Аналітика продажів та відгуків | Таблиця БД «Замовлення» Таблиця БД «Відгук» | Статистика та графіки продажу, популярності, оцінок |

2.4.3. Вимоги до гарантійної підтримки

Основні вимоги до гарантійної підтримки веб-додатку можуть включати наступне:

- Гарантійний період: визначення тривалості гарантійного періоду для безкоштовної підтримки та виправлення помилок після запуску.
- Доступність підтримки: встановлення графіку та способи зв'язку з розробником для отримання підтримки.
- Відповідь на запити: визначення максимального часу реакції на запити та пріоритети відповідей.
- Виправлення помилок: встановлення строк та процедуру виправлення помилок після їх повідомлення.
- Забезпечення безпеки: забезпечити оновлення та моніторинг безпеки системи.

- Оновлення та покращення: визначення політики випуску нових версій та покращень додатку.
- Документація та навчання: надання документації та навчальних матеріалів для користувачів.
- Супровід та консультування: забезпечення консультаційних підтримок для користувачів

2.4.4. Вимоги до лінгвістичного забезпечення системи

Web-додаток має підтримку української мови як основну мову. Усі тексти в додатку будуть перекладені на українську мову для зручності користувачів. Відображення кнопок, посилань, повідомлень та інших текстових елементів буде здійснюватися на українській мові.

Валютою, яка буде використовуватися в додатку, є гривня (грн). Відповідно до цього, всі ціни, вартість продуктів, знижки та інші фінансові відомості будуть відображатися в гривнях.

Також, для зручності користувачів з українською мовою, веб-додаток буде використовувати українські формати дати та часу, які відповідають місцевим вимогам.

2.4.5. Вимоги до апаратного забезпечення

Веб-додатки, які запускаються в браузерах, будуть працювати на різних операційних системах, включаючи Windows, macOS та Linux. Нижче наведені найпоширеніші версії операційних систем, на яких можна використовувати веб-додаток:

- Windows: веб-додатки підтримуються на різних версіях Windows, починаючи з Windows XP і вище. Це включає Windows XP, Windows Vista, Windows 7, Windows 8, Windows 8.1 та Windows 10
- MacOS: веб-додатки можна використовувати на операційній системі macOS, починаючи з версії Mac OS X 10.2 і вище. Деякі з популярних версій macOS включають macOS Mojave, macOS Catalina та macOS Big Sur.
- Linux: багато веб-додатків можуть працювати на різних дистрибутивах Linux, таких як Ubuntu, Fedora, Debian, CentOS та інших. На Linux можна

використовувати веб-додатки на будь-якій актуальній версії цих дистрибутивів

Веб-додатки можна використовувати на різних пристроях, які підтримують веб-браузери. Основні категорії пристроїв, на яких можна запускати веб-додатки, включають: комп'ютери, смартфони, планшети

## 2.5. Склад і зміст робіт по створенню системи

В таблиці 2.2. наведені стадії створення системи і терміни виконання робіт

*Таблиця 2.2. Найменування робіт при створенні системи*

| № п/п | Найменування робіт | Строки виконання робіт |
|---|---|---|
| 1 | Перед проектне дослідження об’єкта автоматизації | 01.03.2023 |
| 2 | Технічне завдання | 01.04.2023 |
| 3 | Технічних проект | 08.04.2023 |
| 4 | Оформлення документації | 29.04.2023 |

## 2.6. Порядок контролю та приймання системи

Порядок контролю та приймання веб-додатку для Пирятинського сирзаводу може включати наступні етапи:

- Введення системи в дію: Веб-додаток вводиться в дію в ТОВ "Пирятинський сирзавод" згідно з вимогами та процедурами.
- Приймальні випробування: Проводяться приймальні випробування згідно встановленого стандарту або регламенту, які визначаються для перевірки функціональності та відповідності веб-додатку вимогам
- Випробування системи: Розробники спільно з замовником проводять випробування системи для оцінки його працездатності та готовності до дослідної експлуатації
- Дослідна експлуатація: Веб-додаток передається в дослідну експлуатацію на підставі технічного завдання та інструкцій користувача.
- Перелік доробок: Після дослідної експлуатації формується перелік необхідних доробок, які включають в себе виявлені недоліки або покращення, а також рекомендовані строки їх виконання.

- Акт здачі-прийому: Оформлення введення системи в дію здійснюється шляхом укладення акта здачі-прийому, який відображає прийняття веб-додатку в експлуатацію.

## 2.7. Вимоги до складу та змісту робіт із підготовки до введення системи в дію

Для введення системи в дію замовник здійснює певні підготовчі заходи, які включають:

- Підготовку технічних засобів, необхідних для роботи системи.
- Здійснення дослідної експлуатації та введення системи в дію.

## 2.8. Вимоги до документації

- Інструкції користувача: Розроблення документації, яка надає пояснення та щодо користування веб-додатком.
- Документація з безпеки: Опис заходів безпеки, включаючи контроль доступу та захист від зломів.
- Документація з розгортання: Інструкції для розгортання веб-додатку на сервері.

## 2.9. Джерела розробки

При розроблені даного технічного завдання на систему використані наступні документи:

- ДСТУ 3008-2015. Документація. Звіти у сфері науки і техніки. Структура та правила оформлювання [14];
- ДСТУ 3973–2000 Система розроблення та поставлення продукції на виробництво [15].

# РОЗДІЛ 3. ОПИС КОМПЛЕКСУ ЗАДАЧ АВТОМАТИЗАЦІЇ

## 3.1. Інформаційне забезпечення системи

Для розроблення функціоналу необхідно створити певні таблиці. Нижче наведено більш детальний опис цих таблиць, а у Додатку Б проілюстрована логічна схема бази даних.

- Продукт – включає данні про продукцію та її детальні характеристики.
- Користувач – включає персональні данні про користувача.
- Відгук – включає данні про рейтинг продукту та користувача, який його поставив.
- Замовлення – включає данні про деталі замовлення та замовника.
- Замовлення продукта - включає данні про кількість та загальну ціну.
- Роль – інформація про назву ролі.
- Ролі користувача – інформація за ролі відповідного користувача.
- Замовлення на постачання сировини – містить інформацію про сировину ціну та данні постачальника.

Для розробки серверної частини проекту буде використано комбінацію таких технологій: NestJS, TypeORM та PostgreSQL.

NestJS - це прогресивний фреймворк для розробки серверних додатків на мові JavaScript або TypeScript. Він базується на екосистемі Node.js і надає потужні інструменти для побудови масштабованих і модульних додатків. NestJS пропонує архітектурний шаблон, відомий як "Dependency Injection", який дозволяє легко управляти залежностями між компонентами додатку [16].

TypeORM - це ORM (Object-Relational Mapping) для TypeScript і JavaScript, який дозволяє зручно взаємодіяти з реляційними базами даних, зокрема з PostgreSQL. TypeORM надає розширений набір інструментів для роботи з базами даних, включаючи створення схеми, виконання запитів, міграції та кешування [17].

PostgreSQL - це потужна реляційна база даних з відкритим кодом, яка пропонує розширений набір функцій і надійність для зберігання та обробки

даних. PostgreSQL підтримує багатофункціональність, зокрема географічні об'єкти, JSON-документи і складні запити [18].

Використання комбінації NestJS, TypeORM та PostgreSQL дозволить забезпечити ефективний процес розробки серверної частини проекту. NestJS надає структуровану архітектуру і зручний спосіб організації коду, TypeORM спрощує роботу з базою даних і забезпечує зручний доступ до неї, а PostgreSQL забезпечує надійне зберігання даних. Разом вони створюють потужний інструментарій для розробки серверних додатків. Фізична модель наведена у Додатку В.

### 3.2. Алгоритмізація та реалізація комплексу задач автоматизації

Почнемо з написання серверної частини додатку, він буде написаний на NestJS для організації запитів на API та TypeORM що буде все взаємодіяти з базою даних PostgreSQL. Це новітні технології які будуть підтримувати додаток довгий час. Почнемо розробку з вхідного файла у додаток це main.ts. Це місце, де створюється та запускаєте екземпляр додатка. У наведеному коді (див. рис. 3.1), відбувається імпорт необхідних модулів з NestJS та Swagger. Функція bootstrap() визначає асинхронний процес налаштування та запуску додатка.

```
main.ts
src > main.ts > ...
      You, last week | 1 author (You)
1   import { NestFactory } from "@nestjs/core";
2   import { DocumentBuilder, SwaggerModule } from "@nestjs/swagger";
3
4   import { AppModule } from "./app.module";
5
6   async function bootstrap() {
7     const PORT = process.env.PORT || 8080;
8     const app = await NestFactory.create(AppModule, {
9       cors: {
10        credentials: true,
11        origin: true,
12      },
13    });
14
15    const config = new DocumentBuilder()
16      .setTitle("Pyryatin REST API")
17      .setDescription("Документація для REST API")
18      .setVersion("1.0.0")
19      .build();
20
21    const document = SwaggerModule.createDocument(app, config);
22
23    SwaggerModule.setup("/api/docs", app, document);
24
25    await app.listen(PORT);
26
27    app.enableCors({
28      origin: true,
29      methods: "GET,HEAD,PUT,PATCH,POST,DELETE,OPTIONS",
30      credentials: true,
31    });
32  }
33        You, last week • feat: first commit
34  bootstrap();
35
```

Рис.3.1. Базова конфігурація серверу

Встановлюється порт, на якому додаток буде прослуховувати запити. Створюється екземпляр додатка, базований на AppModule, та встановлюється конфігурація CORS для обробки запитів з інших доменів. За допомогою DocumentBuilder визначається конфігурація для документації API, а SwaggerModule.createDocument() створює документ Swagger. За допомогою SwaggerModule.setup() встановлюється маршрут для доступу до документації API. Запускається додаток за допомогою app.listen(). Включається CORS для дозволу запитів з інших доменів. Функція bootstrap() викликається для налаштування та запуску додатка. Документацію до API можна поглянути за посиланням, яке було створено у корні.

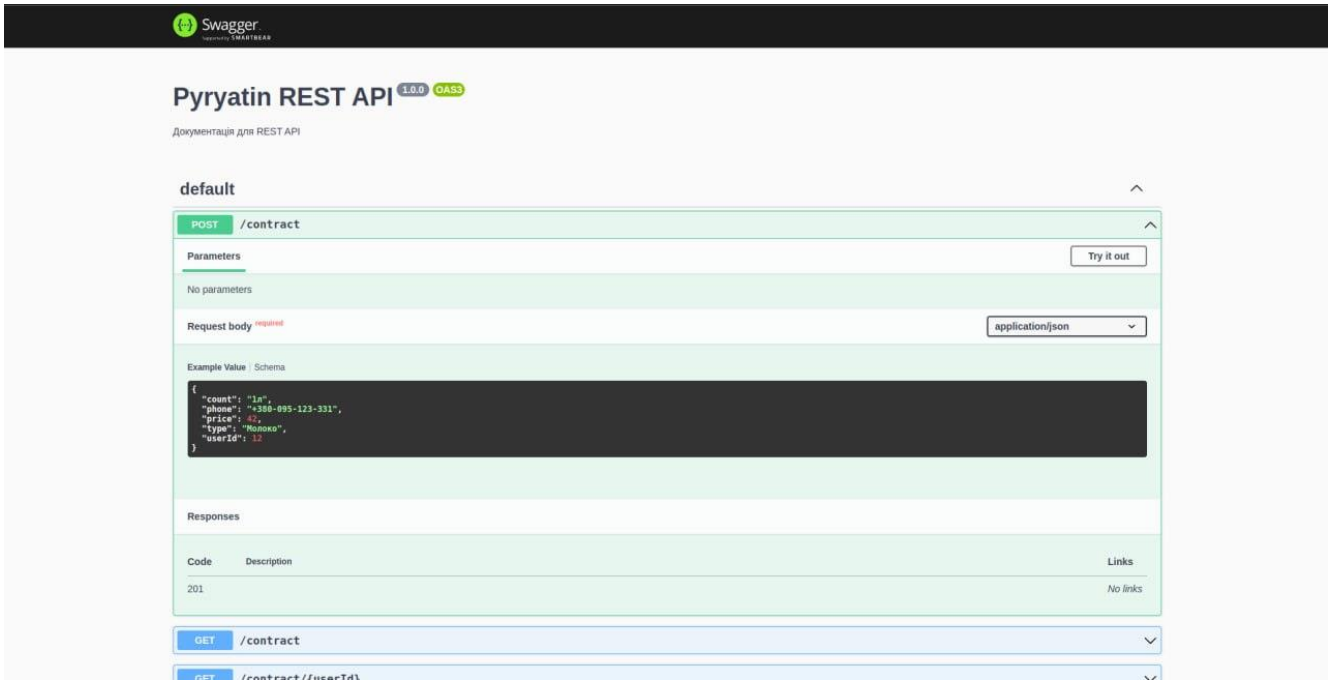



Рис.3.2 Документація API за допомогою Swagger

Використовуючи TypeOrmModule.forRoot(), встановлюється конфігурація підключення до бази даних PostgreSQL. Параметри підключення отримуються зі змінних середовища, таких як POSTGRESS_HOST, POSTGRESS_PORT тощо (див. рис. 3.3).

```
    RolesModule,
    AuthModule,
    ConfigModule.forRoot({
      envFilePath: `.${process.env.NODE_ENV}.env`,
    }),
    MulterModule.register({
      dest: "./static",
    }),
    ServeStaticModule.forRoot({
      rootPath: join(__dirname, "..", "static"),
      serveRoot: "/static",
    }),
    TypeOrmModule.forRoot({
      type: "postgres",
      host: process.env.POSTGRESS_HOST,
      port: Number(process.env.POSTGRESS_PORT),
      username: process.env.POSTGRESS_USER,
      password: process.env.POSTGRESS_PASSWORD,
      database: process.env.POSTGRESS_DB,
      synchronize: true,
      logging: true,
      ssl: {
        rejectUnauthorized: false,
      },
      entities: [
        Role,
        User,
        ProductType,
        Product,
```

Рис.3.3. Підключення до бази даних

Змінні POSTGRESS_HOST, POSTGRESS_PORT, POSTGRESS_USER та POSTGRESS_PASSWORD є змінними оточення, які використовуються для зберігання конфігурації підключення до бази даних PostgreSQL. Вони зазвичай зберігаються у файлі середовища (env файлі), який містить набір змінних оточення і значень, що використовуються в додатку.

У даному коді, використовуючи ConfigModule.forRoot(), встановлюється налаштування для конфігурації додатку, а саме для зчитування змінних оточення з файлу env. Змінна envFilePath вказує шлях до env файлу.

Таким чином, значення POSTGRESS_HOST, POSTGRESS_PORT, POSTGRESS_USER та POSTGRESS_PASSWORD повинні бути визначені у відповідному env файлі. Коли додаток запускається, він використовує ці значення для підключення до бази даних PostgreSQL. Це дає можливість змінювати ці значення в залежності від конкретного середовища, у якому працює додаток (наприклад, розробка, тестування, продакшн) без необхідності внесення змін у самому коді додатка. Приклад файлу для оточення наведено на рисунку 3.4.

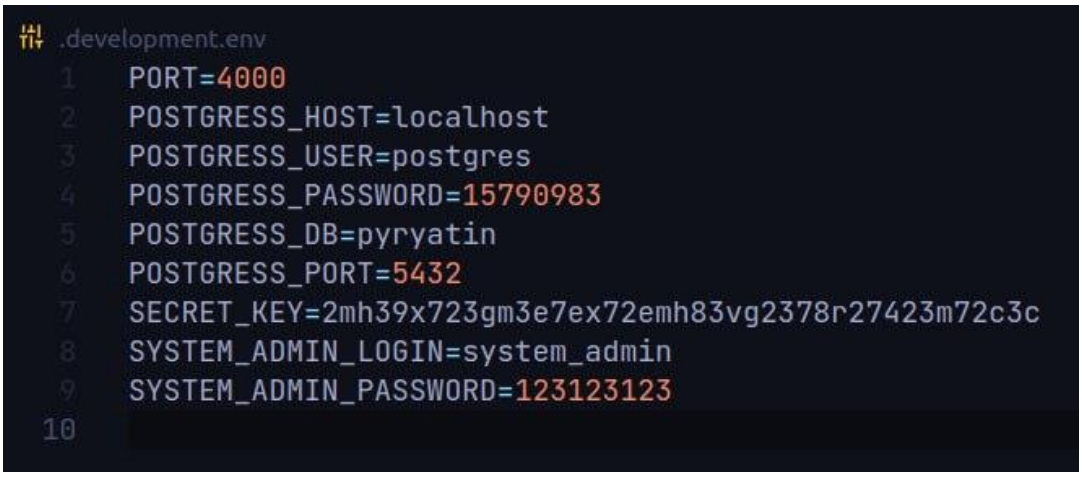



Рис.3.4. Приклад .env файлу

Під час розгортання веб-додатку на хостингу файли env використовуються для налаштування середовища. Необхідно вказувати значення параметрів у файлах env, які будуть використовуватися під час виконання додатка на сервері хостингу. Це дозволяє зберігати конфіденційну інформацію та змінювати налаштування безпосередньо на сервері, не змінюючи вихідний код додатка.

Таким чином, файли env допомагають забезпечити безпеку та гнучкість веб-додатків, а хостинг забезпечує надання ресурсів та доступу до цих файлів для виконання вашого веб-сайту.

Перед формуванням таблиць, запитів тощо слід зрозуміти як працює стек.

Комбінація NestJS, TypeORM і PostgreSQL дозволяє побудувати потужні та масштабовані серверні додатки з базою даних. Основна ідея полягає в тому, що використовується NestJS для створення модулів, контролерів та сервісів, TypeORM для взаємодії з базою даних PostgreSQL, а PostgreSQL для зберігання та доступу до даних. У NestJS використовується модульна структура, де модулі групують пов'язані компоненти разом. Модулі використовуються для організації коду вашого додатка і використовуються для імпорту необхідних залежностей, таких як сервіси, контролери та репозиторії. Контролери у NestJS відповідають за обробку HTTP-запитів. Вони містять маршрутизацію та виклики методів сервісу для виконання бізнес-логіки та обробки запитів. Сервіси у NestJS містять бізнес-логіку вашого додатка. Вони виконують операції з базою даних, обчислення, валідацію та інші операції, які потрібні для обробки запитів. TypeORM

використовується для опису моделей даних вашого додатка та взаємодії з базою даних. Визначаються сутності, які відповідають таблицям бази даних, та репозиторії, які надають методи для взаємодії з цими таблицями. Отже, наявний модуль, який імпортує необхідні модулі, включаючи модулі для кожної сутності додатка. У кожному модулі наявний контролер, який визначає маршрути та обробники HTTP-запитів, та сервіс, який містить бізнес-логіку та виконує операції з базою даних з використанням репозиторіїв TypeORM.

Далі створюємо модуль користувачів та їх ролей. Почнемо з модуля користувачів (див. рис. 3.5).

```
You, last week | 1 author (You)
@Module({
  imports: [TypeOrmModule.forFeature([User, Role])],
  controllers: [UsersController],
  providers: [UsersService, JwtService, RolesService],
  exports: [UsersService],
})
export class UsersModule {}      You, last week • feat: DB co
```

Рисунок 3.5 Модуль користувачів

Цей код створює модуль UsersModule. Модуль імпортує TypeOrmModule.forFeature([User, Role]), щоб використовувати сутності User та Role у цьому модулі. Контролер UsersController використовується для обробки HTTP-запитів. Сервіси UsersService, JwtService та RolesService надають бізнес-логіку та інші функціональності для модуля. UsersService експортується, щоб його можна було імпортувати в інші модулі. Цей модуль служить для групування компонентів, пов'язаних з користувачами, і забезпечує їх взаємодію у вашому NestJS додатку.

Наступним кроком напишемо таблицю користувачів, вори створюються за допомоги декоратора Entity, код таблиці – рисунок 3.6.

```
You, 6 days ago | 1 author (You)
@Entity("user")
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column("text")
  login: string;

  @Column("text")
  name: string;

  @Column("text")
  email: string;

  @Column({ type: "text", nullable: true })
  lastName: string;

  @Column("text")
  password: string;

  @ManyToMany(() => Role, (role) => role.users)
  @JoinTable()
  roles: Role[];

  @OneToMany(() => Review, (review) => review.user)
  reviews: Review[];

  @OneToMany(() => Contract, (contract) => contract.user)
  contracts: Contract[];
}
```

Рис.3.6 Створення таблиці користувача

Цей код визначає сутність User у додатку. Сутність представляє таблицю user в базі даних. Вона містить різні поля, такі як id, login, name, email, lastName і password. Поле id є первинним ключем, який генерується автоматично. Поля login, name, email, lastName та password є текстовими колонками у базі даних.

Також у сутності є різні відношення. Відношення @ManyToMany вказує на багато-до-багатьох зв'язок між сутністю User та Role. Це означає, що один користувач може мати декілька ролей, і кожна роль може бути пов'язана з декількома користувачами. Для цього використовується проміжна таблиця, яка створюється за допомогою @JoinTable().

Також є два відношення @OneToMany. Вони вказують на один-до-багатьох зв'язок між сутністю User та Review, а також між сутністю User та Contract. Це означає, що один користувач може мати декілька відгуків (Reviews) та декілька контрактів (Contracts) [19].

За попереднім прикладом створюємо таблицю ролей та модуль ролей.

```
You, last week | 1 author (You)
@Entity("role")
export class Role {
  @ApiProperty({ example: "42" })
  @PrimaryGeneratedColumn()
  id: number;

  @ApiProperty({
    example: "Admin",
  })
  @Column("text")
  value: string;

  @ManyToMany(() => User, (user) => user.roles)
  users: User;
}       You, last week • feat: prpz_lb_2
```

Рис.3.7 Створення таблиці ролей

Після розробки сутностей ролей та користувачів можна приступити до створення функціоналу авторизації, для цього напишемо AuthModule який буде контролювати поведінку авторизації.

```
You, last week | 1 author (You)
@Module({
  imports: [
    JwtModule.register({
      secret: process.env.SECRET_KEY || "secret",
      signOptions: {
        expiresIn: "7d",
      },
      secretOrKeyProvider: (requestType: JwtSecretRequestType) => {
        switch (requestType) {
          default:
            return process.env.SECRET_KEY || "secret";
        }
      },
    }),

    TypeOrmModule.forFeature([User, Role]),
  ],
  controllers: [AuthController],
  providers: [AuthService, UsersService, RolesService],
})
export class AuthModule {}
```

Рис.3.8 Створення модуля авторизації

Цей код визначає модуль AuthModule. У модулі виконується імпорт двох інших модулів: JwtModule та TypeOrmModule.forFeature(). JwtModule використовується для налаштування аутентифікації з використанням JWT-токенів. TypeOrmModule.forFeature() вимагає імпортувати сутності User та Role для використання в цьому модулі.

Модуль AuthModule також визначає контролер AuthController, який відповідає за обробку HTTP-запитів, пов'язаних з аутентифікацією. Він також визначає провайдери (сервіси) AuthService, UsersService та RolesService, які надають функціональність для аутентифікації, операцій з користувачами та операцій з ролями, відповідно.

Модуль AuthModule використовується для організації всіх компонентів, пов'язаних з аутентифікацією, в одному місці. Це дозволяє логічно групувати всі функції, пов'язані з аутентифікацією, та забезпечує їх доступність в інших частинах додатку, якщо AuthModule імпортується в цих частинах.

Зараз напишемо UsersService, який відповідає за операції над користувачами. Сервіс має кілька методів для взаємодії з базою даних та обробки користувачів, рисунок 3.9.

```
@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(User)
    private userRepository: Repository<User>,
    private rolesService: RolesService
  ) {}

  async getAllUsers() {
    const users = await this.userRepository.find();
    return users;
  }

  async getByLogin(login: string, withError = false) {
    const user = await this.userRepository.findOne({
      where: {
        login,
      },
      relations: {      You, 6 days ago • feat: a lot
        roles: true,
      },
    });

    if (!user && withError) {
      throw new HttpException(
        "User with this login was not found",
        HttpStatus.NOT_FOUND
      );
    }

    return user;
  }
```

```
async getById(id: number) {
  const user = await this.userRepository.findOne({
    where: {
      id,
    },

    relations: {
      roles: true,
    },
  });

  if (!user) {
    throw new HttpException(
      "User with this id was not found",
      HttpStatus.NOT_FOUND
    );
  }

  return user;
}

async create(dto: CreateUserDto) {
  const role = await this.rolesService.getByValue(dto.initialRole);
  const user = await this.userRepository.save({
    email: dto.email,
    lastName: dto.lastName,
    login: dto.login,
    name: dto.name,
    password: dto.password,
    roles: [role],
  });

  return user;      You, last week • feat: prpz_lb_2
}
```

Рис.3.9 Частина коду UsersService

У конструкторі сервісу ін'єктується репозиторій User, який надає доступ до операцій з базою даних пов'язаних з користувачами, та сервіс RolesService, який використовується для отримання інформації про ролі користувачів.

Метод getAllUsers() повертає всіх користувачів, викликаючи метод find() на репозиторії User.

Метод getByLogin(login: string, withError = false) повертає користувача за його логіном. Він викликає метод findOne() на репозиторії User з відповідним запитом із зазначеним логіном. Якщо користувача не знайдено і параметр withError встановлений на true, генерується HttpException з повідомленням про помилку.

Метод getById(id: number) повертає користувача за його ідентифікатором. Він викликає метод findOne() на репозиторії User з відповідним запитом із зазначеним ідентифікатором. Якщо користувача не знайдено, генерується HttpException з повідомленням про помилку.

Метод create(dto: CreateUserDto) створює нового користувача на основі наданого об'єкта CreateUserDto. Він отримує дані для створення нового користувача, включаючи ім'я, прізвище, логін, електронну пошту та пароль. За допомогою сервісу RolesService отримується відповідна роль для користувача. Потім створений користувач зберігається за допомогою методу save() на репозиторії User, і його дані повертаються як результат.

Для авторизації напишемо сервіс AuthService у якому будуть видаватися токени користувачам та хешируватися їх паролі при реєстрації. Розглянемо функцію реєстрації користувача покроково.

```
async register(userDto: CreateUserDto) {
  const candidate = await this.usersService.getByLogin(userDto.login);
  if (candidate) {
    throw new HttpException(
      "User with this login already exists",
      HttpStatus.BAD_REQUEST
    );
  }
  const hashedPassword = await bcrypt.hash(userDto.password, 5);
     You, last week • feat: prpz_lb_2
  const user = await this.usersService.create({
    ...userDto,
    password: hashedPassword,
  });
  const token = await this.generateToken(user);

  return {
    user,
    token,
  };
}
```

Рис.3.10. Функція реєстрації користувача

const candidate = await this.usersService.getByLogin(userDto.login);: Виконується пошук користувача за логіном, використовуючи метод getByLogin() сервісу usersService. Якщо користувач з таким логіном вже існує, значення candidate буде непустим.

if (candidate) { throw new HttpException("User with this login already exists", HttpStatus.BAD_REQUEST); }: Якщо знайдений користувач з вказаним логіном (тобто candidate не є пустим), генерується HttpException з повідомленням про помилку. Це означає, що користувач з таким логіном вже існує, і новий користувач не може бути зареєстрований з таким самим логіном. Заголовок статусу відповіді буде HttpStatus.BAD_REQUEST.

const hashedPassword = await bcrypt.hash(userDto.password, 5);: Пароль, вказаний у userDto, хешується за допомогою функції bcrypt.hash(). Хешування паролю забезпечує безпеку зберігання паролів у базі даних. Константа hashedPassword містить отриманий хеш паролю.

const user = await this.usersService.create({...userDto, password: hashedPassword});: Створюється новий користувач шляхом виклику методу create() сервісу usersService. Використовується розширення об'єкта {...userDto,

password: hashedPassword}, щоб замінити пароль у userDto на хешований пароль. Константа user містить створеного користувача.

const token = await this.generateToken(user);: Генерується токен для створеного користувача за допомогою методу generateToken(). Токен може використовуватись для аутентифікації користувача після успішної реєстрації.

return { user, token };: Повертається об'єкт, який містить створеного користувача (user) і токен (token). Це стане відповіддю на запит реєстрації і може бути використано на клієнтській стороні для подальшої роботи з автентифікацією та авторизацією.

Також були написані функції verifyUser, generateToken та login рисунок 3.11

```
private async verifyUser(userDto: LoginDto) {
  const user = await this.usersService.getByLogin(userDto.login, true);
  const passwordEqual = await bcrypt.compare(userDto.password, user.password);

  if (passwordEqual) return user;

  throw new UnauthorizedException({
    message: "Incorrect login or password",
  });
}

private async generateToken(user: User) {
  const payload = { login: user.login, id: user.id, roles: user.roles };
  return this.jwtService.sign(payload);
}

async login(userDto: LoginDto) {
  const user = await this.verifyUser(userDto);
  const token = await this.generateToken(user);

  delete user.password;

  return {
    user,
    token,
  };
}
```

Рис.3.11 Функції для авторизації

Функції verifyUser() і generateToken() виконують важні завдання в контексті аутентифікації та авторизації користувачів. Ось їх пояснення:

verifyUser(userDto: LoginDto): Ця функція перевіряє введені дані користувача для входу (логін і пароль) та перевіряє їх зі збереженими даними користувача в базі даних. Вона використовує метод getByLogin() з сервісу

usersService, щоб отримати користувача за вказаним логіном і перевірити, чи співпадає введений пароль зі збереженим паролем користувача. Якщо перевірка пройшла успішно, функція повертає об'єкт користувача.

generateToken(user: User): Ця функція генерує JWT-токен (JSON Web Token) для користувача. Вона створює токен на основі об'єкта payload, який містить інформацію про користувача, таку як логін, ідентифікатор і ролі. Згенерований токен повертається як результат функції.

login(userDto: LoginDto): Ця функція обробляє процес входу користувача. Вона викликає функцію verifyUser() для перевірки введених даних користувача і отримання об'єкта користувача. Потім вона викликає функцію generateToken() для генерації JWT-токена для цього користувача. Нарешті, вона видаляє поле пароля з об'єкта користувача та повертає об'єкт користувача та токен як результат функції.

Після написання сервісу можна написати контроллер який буде користуватися методами сервісів рисунок 3.12

```
@Controller("auth")
export class AuthController {
  constructor(private authService: AuthService) {}

  @Post("/login")
  login(@Body() dto: LoginDto) {
    return this.authService.login(dto);
  }

  @Post("/verify")
  verify(@Req() req: Request) {
    return this.authService.verify(req.headers.authorization);
  }

  @Post("/register")
  register(@Body() dto: CreateUserDto) {
    return this.authService.register(dto);
  }
}
```

Рис. 3.12 AuthController для визначення ендпоінтів

Цей код визначає контролер AuthController, який відповідає за обробку HTTP-запитів, пов'язаних з аутентифікацією та реєстрацією користувачів. Ось пояснення кожного методу контролера:

- @Post("/login"): Цей декоратор вказує, що метод login() обробляє HTTP POST-запит на шляху /login.

  login(@Body() dto: LoginDto): Цей метод отримує об'єкт dto з тіла запиту, який містить дані для входу користувача. Метод передає отриманий об'єкт до сервісу authService через метод login(), який відповідає за обробку процесу входу користувача.

- @Post("/verify"): Цей декоратор вказує, що метод verify() обробляє HTTP POST-запит на шляху /verify.

  verify(@Req() req: Request): Цей метод отримує об'єкт req з інформацією про HTTP-запит, включаючи заголовки. Метод передає заголовок авторизації з запиту до сервісу authService через метод verify(), який відповідає за перевірку валідності токену авторизації.

- @Post("/register"): Цей декоратор вказує, що метод register() обробляє HTTP POST-запит на шляху /register.

  register(@Body() dto: CreateUserDto): Цей метод отримує об'єкт dto з тіла запиту, який містить дані для реєстрації нового користувача. Метод передає отриманий об'єкт до сервісу authService через метод register(), який відповідає за обробку процесу реєстрації нового користувача.

Тепер розгляемо біль детально класи-DTO (Data Transfer Object) вони використовується для передачі даних між різними компонентами системи, зазвичай між клієнтом та сервером або між різними частинами серверної сторони. Основна мета DTO - це створення інтерфейсу для обміну даними, що спрощує комунікацію та забезпечує її стабільність [20].

```
You, last week | 1 author (You)
import { IsNotEmpty, IsString } from "class-validator";

You, last week | 1 author (You)
export class LoginDto {
  @IsString()
  @IsNotEmpty()
  login: string;

  @IsString()
  @IsNotEmpty()
  password: string;
}

You, last week | 1 author (You)
export class LoginByTokenDto {
  @IsString()
  @IsNotEmpty()
  token: string;
}
```

Рис.3.13 Data Transfer Object для входу у систему

Для того щоб переконатися що данні передані правильно використаємо бібліотеку class-validator.

У цьому випадку, class-validator використовується для валідації даних, що передаються в об'єктах LoginDto та LoginByTokenDto. Основні анотації, що використовуються, це @IsString() та @IsNotEmpty().

- @IsString(): Ця анотація перевіряє, чи є значення властивості рядком. Якщо значення не є рядком, то валідація вважається неуспішною.
- @IsNotEmpty(): Ця анотація перевіряє, чи є значення властивості не порожнім рядком або null. Якщо значення є порожнім рядком або null, то валідація вважається неуспішною.

При отриманні даних для валідації, class-validator застосовує ці анотації до властивостей об'єкту та перевіряє, чи вони відповідають вказаним умовам. Якщо яка-небудь з анотацій не пройшла валідацію, генерується помилка валідації, яку можна обробити в коді [21].

Наприклад, якщо об'єкт LoginDto має поле login, то валідація перевірить, чи це поле є рядком та чи не є порожнім рядком або null. Якщо які-небудь з цих умов не виконується, генерується помилка валідації.

Використання class-validator допомагає забезпечити, що дані, які передаються у валідованому об'єкті, відповідають вказаним критеріям валідації. Це спрощує обробку та перевірку вхідних даних у системі.

Аналогідно було створено інші таблиці та взаємодії між ними, кінцева структура проекту виглядає наступним чином:

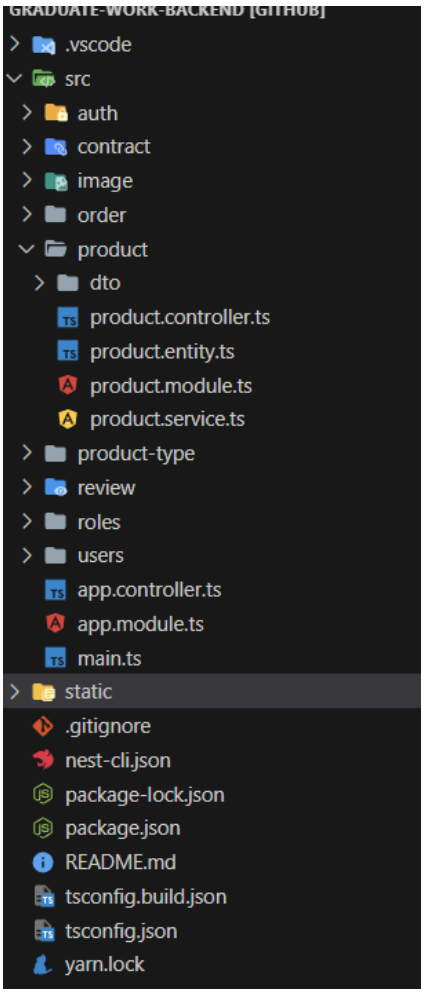



Рис.3.14 Кінцева структура проекту

Закінчивши розробку серверної частини проекту можна приступити до написання клієнтської зі стеком React/TS/MUI, дещо про ці технології:

React - це відкрита JavaScript бібліотека для створення користувацьких інтерфейсів. Вона дозволяє розробникам будувати ефективні та масштабовані веб-додатки, використовуючи компонентний підхід. React використовує віртуальний DOM (Document Object Model), що дозволяє здійснювати швидкий і ефективний оновлення інтерфейсу, враховуючи тільки необхідні зміни [22].

TypeScript (TS) - це розширення мови JavaScript, яке додає статичну типізацію до JavaScript. З TypeScript можна визначати типи для змінних, функцій, параметрів і повертаємого значення. Це допомагає під час розробки, оскільки забезпечує перевірку типів на етапі компіляції, зменшуючи кількість помилок і полегшуючи рефакторинг коду. React разом з TypeScript надають більшу надійність і чіткість при написанні компонентів і логіки додатків [23].

Material-UI (MUI) - це популярна бібліотека компонентів для React, яка забезпечує готові стилізовані елементи і компоненти інтерфейсу користувача, які можна використовувати для швидкої розробки стильних додатків. MUI базується на дизайні Material Design, розробленому Google, і пропонує широкий набір готових компонентів, які добре працюють разом і забезпечують єдинообразний вигляд додатку. З використанням MUI ви можете створювати інтерактивні, адаптивні та зручні використовувати веб-інтерфейси [24].

Почнемо з написання API для взяємодії з серверною частиною, кожна папка матиме назву яка характеризує данні з якими буде працювати. Також типи данних які повертаються с бека та index.ts файл для функцій запитів на бек, детальніше на рисунку 3.15.

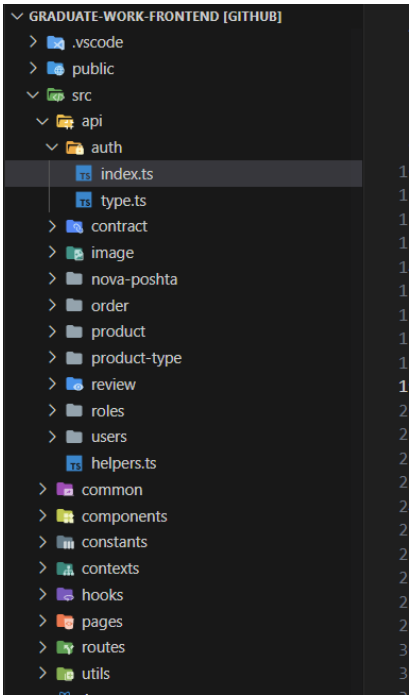



Рис. 3.15 Структура api на клієнті

Кожен вайл запитів буде мати адреси для запитів на функції для їх посилання, рисунок 3.16.


Рис. 3.16. Файл auth запитів на серверну частину

Другим файлом у кожній папці API буде файл типізації у якому будуть різні типи даних пов’язані с беком як просто тип відповіді так і тип так званого payload, який використовується для корисного навантаження. Як можна побачити на рисунку 3.17 у нас типізовані RegisterUserParams (payload - для створення користувача), як з відповідними даними так і RegisterUserResponse тобто тип даних відповіді на успішне створення того ж користувача.

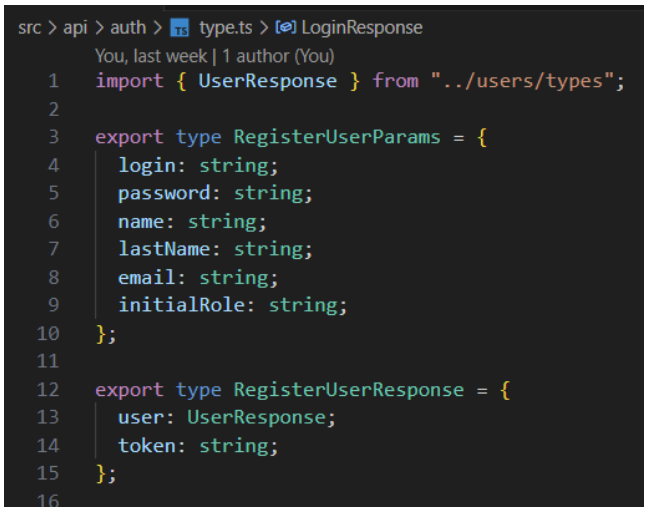



Рис.3.17 Типи даних для авторизації

Також, слід створити константи такі як BASE_URL та AUTH_TOKEN_KEY (див. рис. 3.18).

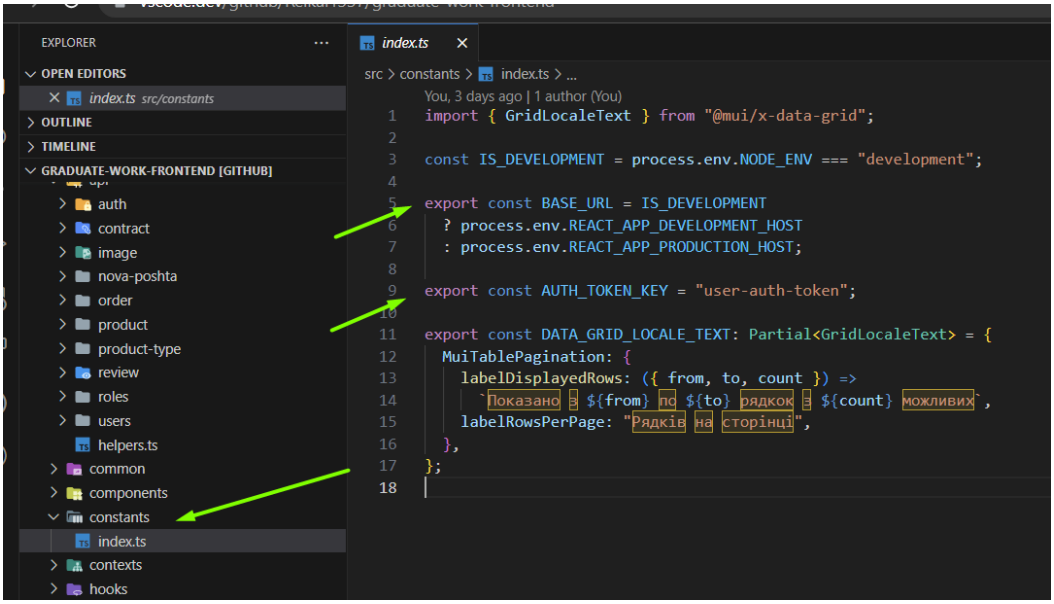



Рис. 3.18 Константи

Константа BASE_URL використовується для збереження адреси сервера API, до якого будуть здійснюватися запити з фронтенду. Вона вказується відповідно до налаштувань серверної сторони, де розгорнуто API.

Константа AUTH_TOKEN_KEY використовується для вказівки назви ключа, за допомогою якого токен автентифікації зберігається в локальному сховищі (наприклад, localStorage) на фронтенді. Це дозволяє зручно використовувати токен для автентифікації користувача під час звернення до серверних ресурсів.

Наступним кроком буде навігація.

Файл "DynamicPages.tsx" (див. рис.3.19) містить компонент "DynamicPages", який відповідає за створення динамічних сторінок. Використовується бібліотека "react-router-dom" для маршрутизації. Масив "routes" мапиться на компоненти "Route", які відповідають за шляхи та компоненти сторінок. Є також шлях "*", який перенаправляє на сторінку, визначену константою "PRODUCT_PAGE_ROUTE".

Файл "Pages.tsx" містить компонент "Pages", який створює сторінку зі затримкою завантаження. Використовується компонент "Suspense" для затримки

завантаження компонента "DynamicPages". При затримці показується компонент "CircularProgress".

Файл "dictionary.ts" містить константи з шляхами до сторінок.

Файл "routes.ts" визначає масив "routes", який містить об'єкти "Route" з властивостями для кожної сторінки.

```
export const routes: Route[] = [
  {
    name: PAGES.ProductsPage,
    path: PRODUCT_PAGE_ROUTE,
    exact: true,
    Component: ProductsPage,
    icon: "products-page",
  },      You, 6 days ago • faat: a lot
  {
    name: PAGES.ProductsPage,
    path: lOGIN_PAGE_ROUTE,
    exact: true,
    Component: LoginPage,
    icon: "login-page",
  },
  {
    name: PAGES.RegisterPage,
    path: REGISTER_PAGE_ROUTE,
    exact: true,
    Component: RegisterPage,
    icon: "register-page",
  },
  {
    name: PAGES.ManagementPage,
    path: MANAGEMENT_PAGE_ROUTE,
    exact: true,
    Component: ManagementPage,
    icon: "management-page",
  },
```

```
You, 6 days ago | 1 author (You)
import { Navigate, Route, Routes } from "react-router-dom";

import { routes } from "../routes";
import { PRODUCT_PAGE_ROUTE } from "../routes/dictionary";

const DynamicPages = () => {
  return (
    <Routes>
      {routes.map(({ Component, exact, icon, name, path, withAuth }, index) => {
        return (
          <Route
            key={index}
            {...{ exact, icon, name, path }}
            element={<Component />}
          />
        );
      })}
      <Route path="*" element={<Navigate to={PRODUCT_PAGE_ROUTE} />} />
    </Routes>
  );
};

export default DynamicPages;
```

Рис. 3.19 Маршрути та їх сторінки

Перед розробкою сторінок напишемо два корисних хука які потім допоможуть там вирішити проблеми з мобільною версією та затримкою перед запитом.

useDebounce в React є хуком, який допомагає оптимізувати запити на сервер під час введення даних у поле введення. Він затримує виклики функцій протягом заданого часу після останньої зміни, що дозволяє зменшити кількість надсиланих запитів і покращити продуктивність додатку (див. рис. рис.3.20). У нашому випадку він буде використовуватися коли користувач буде шукати продукцію та коли він буде вибирати відділення нової пошти для обрання адреси доставки.

```
src > hooks > useDebounce.ts > ...
      You, 3 days ago | 1 author (You)
1   import { useEffect, useState } from "react";
2
3   export const useDebounce = <T>(value: T, delay?: number): T => {
4     const [debouncedValue, setDebouncedValue] = useState<T>(value);
5
6     useEffect(() => {
7       const timer = setTimeout(() => setDebouncedValue(value), delay || 500);
8
9       return () => {
10        clearTimeout(timer);
11      };
12    }, [value, delay]);
13
14    return debouncedValue;
15  };
16
```

Рис. 3.20 useDebounce – хук для затримки виклику

useIsMobile в React є хуком, який допомагає виявити, чи використовується додаток на мобільному пристрої. Він повертає булеве значення, яке може бути використано для зміни дизайну та поведінки додатку для мобільних пристроїв. (див. рис. рис.3.21). У нас від буде використовуватися для ширини карточок продуктів у списку, зміни розміру шрифта та приховування деяких елементів на мобільних пристроях.

```
src > hooks > isMbile.ts > ...
1   import { useLayoutEffect, useState } from "react";
2
3   export const useIsMobile = (): boolean => {
4     const [isMobile, setIsMobile] = useState(false);
5
6     useLayoutEffect(() => {
7       const handleResize = () => {
8         setIsMobile(window.innerWidth <= 768);
9       };
10      handleResize();
11      window.addEventListener("resize", handleResize);
12      return () => {
13        window.removeEventListener("resize", handleResize);
14      };
15    }, []);
16
17    return isMobile;
18  };
```

Рис. 3.21 useIsMobile – хук для встановлення пристрою користувача

При спільному використанні yup, useFormik та MUI можна створити форму для створення продукту. Валідацією данних буде займатися yup (рис. 3.22),

контролем полів вводу – useFormik (рис. 3.22 ), а поля вводу візьмемо з MUI (рис. 3.23)

```
const validationSchema = yup.object({
  name: yup.string().required("Ім'я обовязкове"),
  description: yup.string().required("Опис обовязковий"),
  price: yup.number().required("Ціна is обовязкова"),
  sale: yup.number().max(100, "Максимальна знижка це 100%"),
  type: yup.string().required("Тип обовязковий"),
  weight: yup.number().required("Вага обовязкова"),
  weightType: yup.string().required("Тип ваги обовязковий"),
  barcode: yup.string().required("Штрих-код обовязковий"),
  boxSize: yup.string().required("Розмір ящика обовязковий"),
  package: yup.string().required("Упаковка обовязкова"),
  availableQuantity: yup.number().required("Кількість обовязкова"),
  quantityPerBox: yup.number().required("Кількість у ящику обовязково"),
  storageConditions: yup
    .string()
    .required("Строк та умови зберігання обовязково"),
});      You, 6 days ago • faat: a lot
```

Рис. 3.22 Валідація форми за допогою yup

Даний код використовує бібліотеку Yup для валідації даних в об'єкті. Він описує схему валідації для об'єкта, який має наступні поля: name, description, price, sale, type, weight, weightType, barcode, boxSize, package, quantityPerBox і storageConditions. Для кожного поля визначається тип валідації та повідомлення про помилку, якщо валідація не пройшла успішно. Наприклад, для поля "name" використовується метод yup.string(), який перевіряє, що значення поля є рядком. Якщо значення відсутнє (null або undefined) або не є рядком, то виникає помилка "Ім'я обовязкове".

Аналогічно, для полів "description", "price", "type", "weight", "weightType", "barcode", "boxSize", "package", "quantityPerBox" та "storageConditions" визначаються відповідні типи валідації та повідомлення про помилку.

Для поля "sale" використовується метод yup.number().max(100, "Максимальна знижка це 100%"), який перевіряє, що значення поля є числом і не перевищує 100. Якщо значення не є числом або перевищує 100, то виникає помилка "Максимальна знижка це 100%".

Після опису схеми валідації, об'єкт validationSchema можна використовувати для перевірки валідності даних.

```
<Stack spacing={1} direction="row">
  <TextField
    fullWidth
    label="Вага"
    name="weight"
    variant="outlined"
    size="small"
    disabled={formik.isSubmitting}
    value={formik.values.weight}
    onChange={formik.handleChange}
    error={Boolean(formik.touched.weight && formik.errors.weight)}
    helperText={formik.touched.weight && formik.errors.weight}
  />
  <TextField
```

Рис. 3.23 Контроль полів вводу – useFormik

Цей код створює компонент TextField, який представляє текстове поле для введення значення ваги. Він налаштовує вигляд та поведінку поля, включаючи його ширину, мітку, ім'я, стиль, розмір, активність, значення, обробку змін, відображення помилок валідації та допоміжні повідомлення.

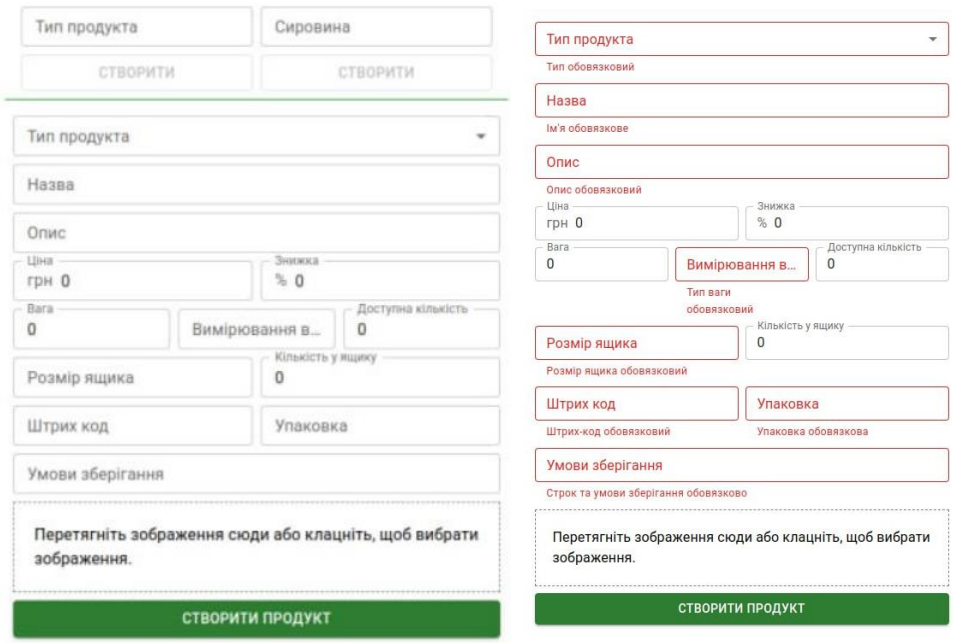



Рис. 3.24 Готова форма на MUI

Таким чином створювалися різні форми, сторінки створювалися за допомогою використання хука isMobile та компонентів MUI.

## 3.3. Інструкція користувача

Кінцевий продукт має такі сторінки:

- «Продукція» – пошук, фільтрація редагування та перегляд детальної інформації;
- «Авторизація» – вхід на реєстрація у системі;
- «Кошик» - редагування продукції для замовлення, створення замовлення;
- «Поставка» - створення заявок на поставку сировини;
- «Мої поставки» - список поставок сировини користувача та їх статусу;
- «Створення продукції» – створення продуктів, їх типів та сировини;
- «Створення персоналу» – створення нових адмінів та менеджерів;
- «Замовлення та пропозиції» - менеджмент заявок на продукцію та на поставку сировини;
- «Статистика» - перегляд діаграм сум замовлень, популярності продуктів та типів за проміжок часу;

Додаток адаптивний та зможе підлаштуватися під будь-який дивайс користувача, після завантаження користувач потрапляє на домашню сторінку з невеликим описом діяльності підприємства, навігацією та кнопкою для переправлення на сторінку продукції.

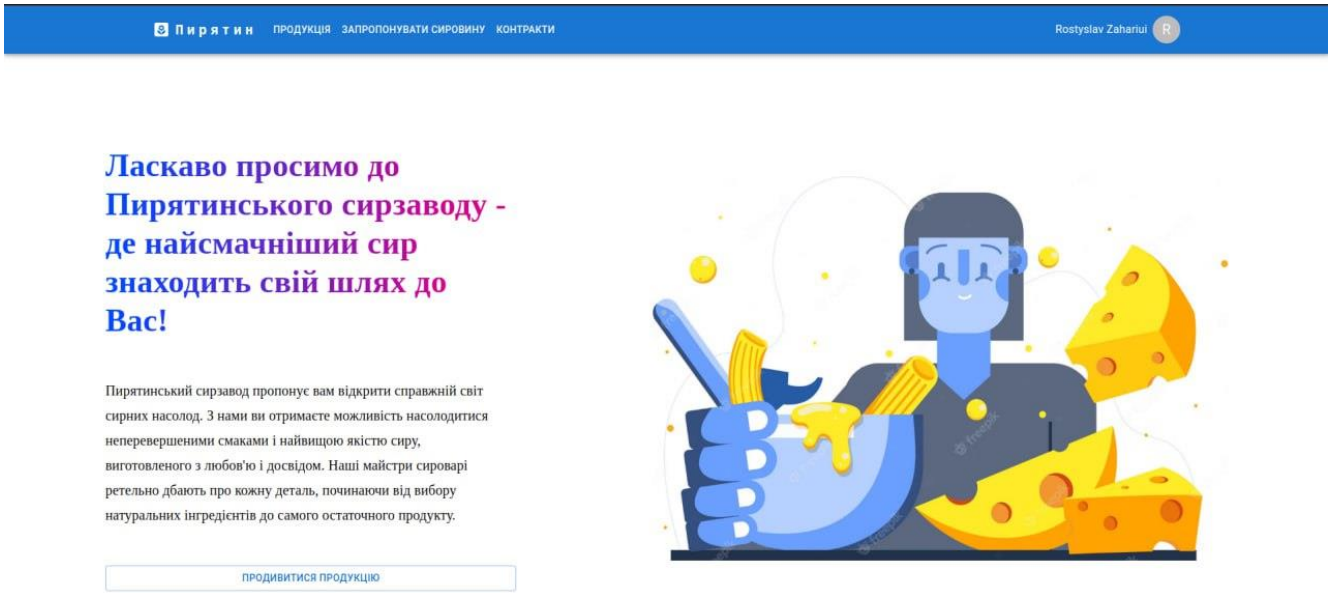



Рис. 3.25 Домашня сторінка додатку

Сторінка продукції буде дещо відрізнятися відповідно до ролі чи авторизації. Авторизовані користувачі матимуть змогу оцінити продукцію, на відміну від не авторизованих користувачів, але вони можуть зробити замовлення, продивитися деталі продукції скористуватися зручним фільтром по типу сира на пошуком по назві продукту, рисунок 3.26.

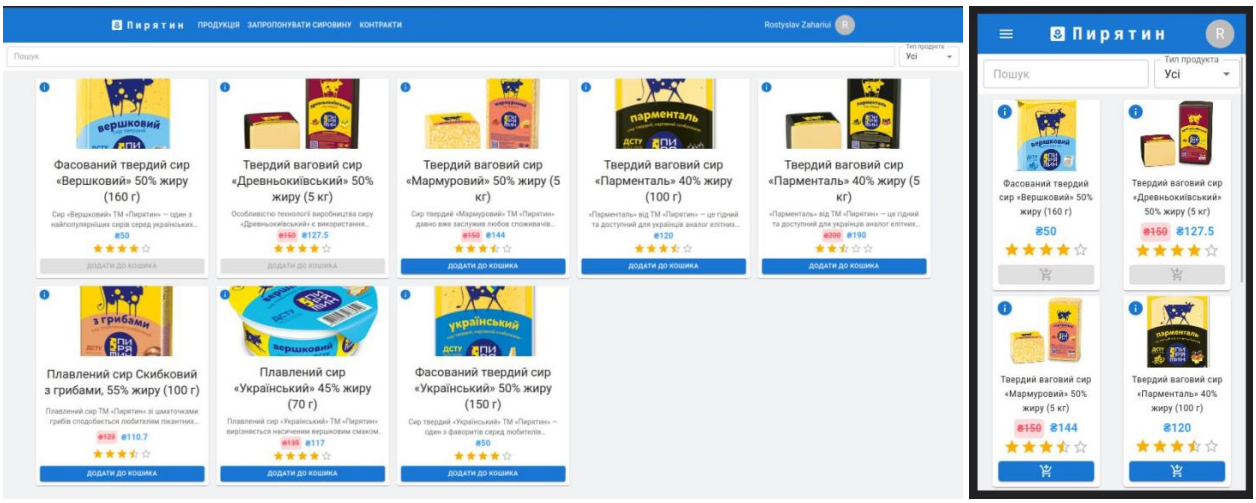



Рис. 3.26 Сторінка продукції

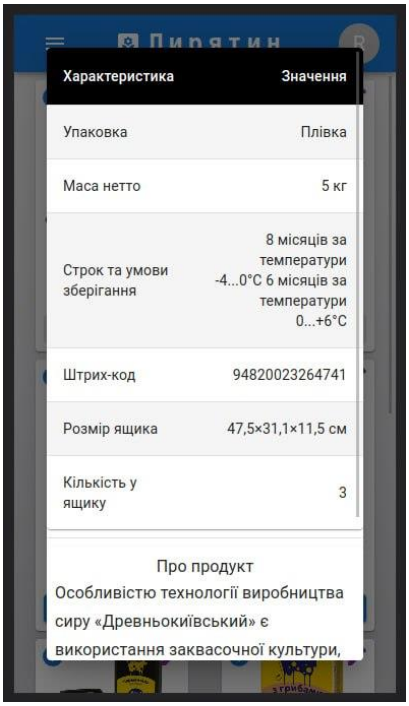



Рис.3.27. Перегляд деталей продукту

Після додавання продукції у кошик, користувач може встановити кількість товару, рисунок 3.28, переглянути загальний рахунок, вибрати адресу доставки та відділення Нової Пошти для оформлення замовлення. Зручний інтерфейс

дозволяє користувачу швидко та з легкістю здійснити ці кроки, забезпечуючи зручність та швидкість у процесі оформлення замовлення.

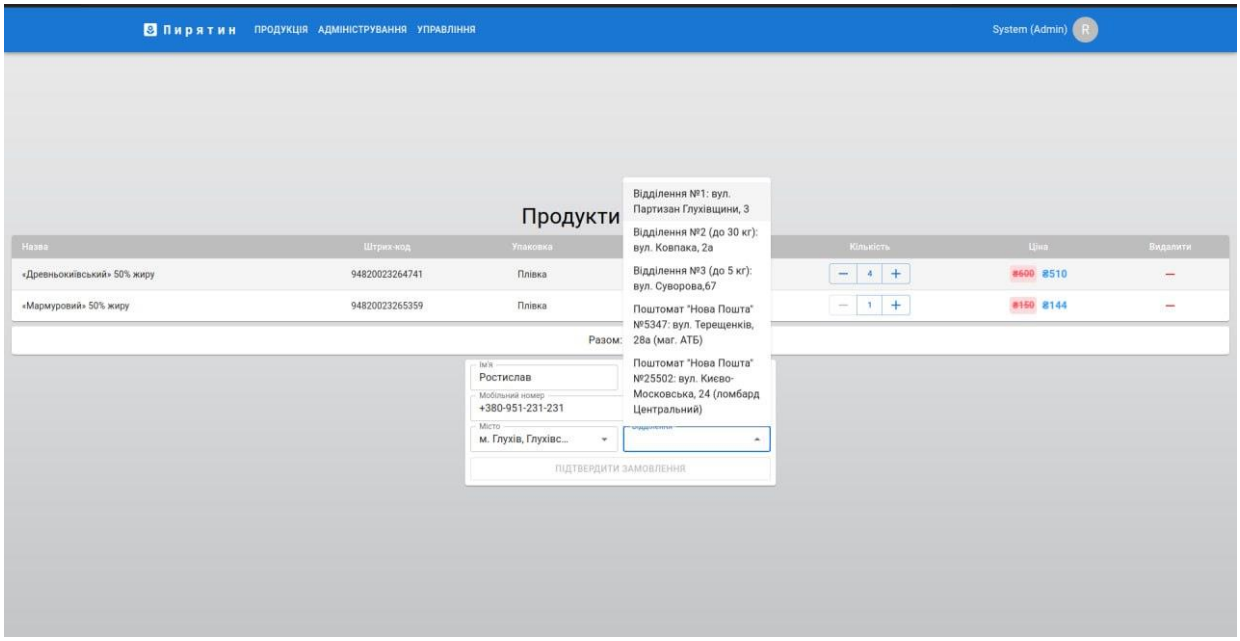



Рис.3.28 Формування замовлення

Користувач матиме змогу запропонувати свою сировину для підприємства. (Рисунок 3.29) Загалом, дозволити користувачам запропонувати свою сировину для підприємства може мати багато переваг, включаючи розширення вибору, внесення інновацій, розвиток партнерства та залучення спільноти. Це може сприяти покращенню якості продукції та забезпечити конкурентну перевагу підприємству.

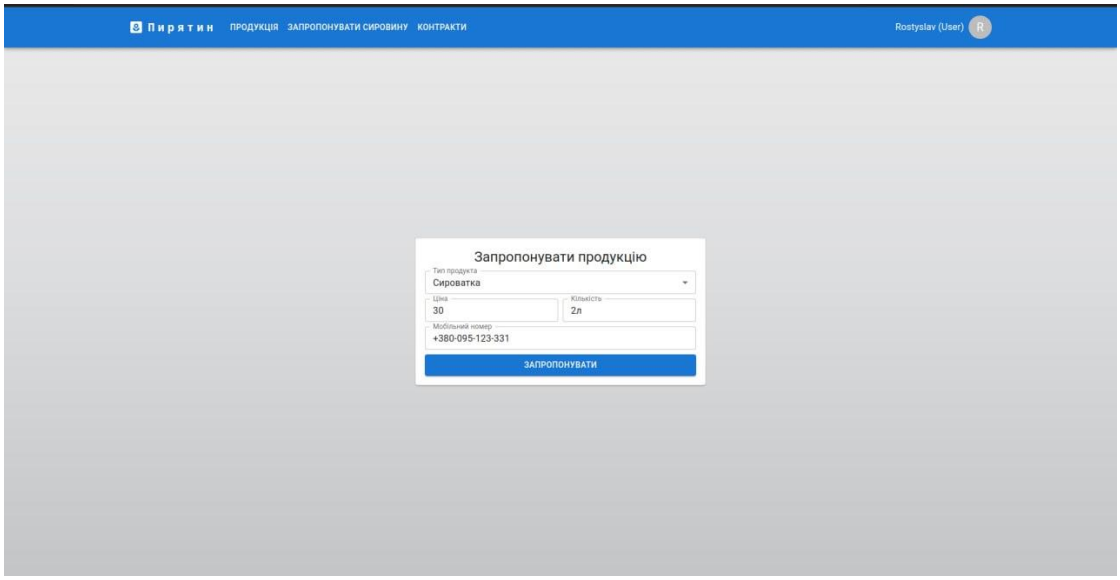



Рис. 3.29 Форма подання заявки на пропозицію сировини

Користувач матиме можливість відстежувати статус своєї пропозиції сировини, рисунок 3.30, та виконати додаткові дії щодо неї. Зручний інтерфейс дозволяє користувачу отримувати інформацію про статус своєї пропозиції, а також здійснювати дії, такі як перегляд деталей або видалення пропозиції.

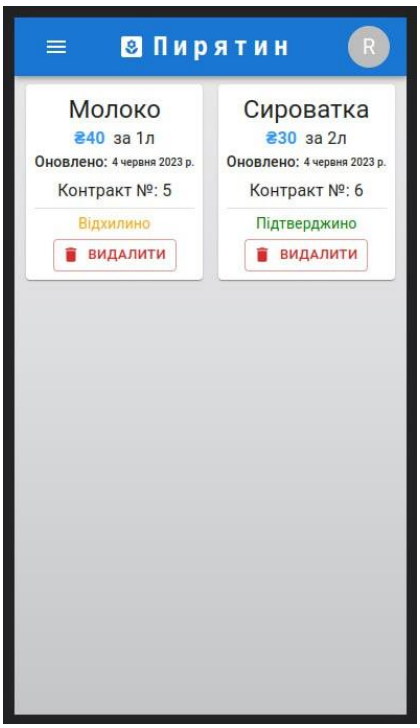



Рис. 3.30 Перегляд списку пропозицій

Тепер перейдемо до функціоналу, який доступний для менеджерів та адміністрації. Цей функціонал надає розширені можливості для управління та контролю роботи замовниками та продукцією Пирятинського сирзаводу.

На сторінці продукції доступна можливість редагувати продукт, рисунок 3.31. Це дозволяє менеджерам та адміністрації Пирятинського сирзаводу вносити зміни до інформації про продукт, а також керувати його властивостями.

На сторінці створення веб-додатку доступні різні форми для створення продукту, типу продукту та сировини, рисунок 3.32. Ці форми надають зручні та окремі можливості для введення необхідної інформації.

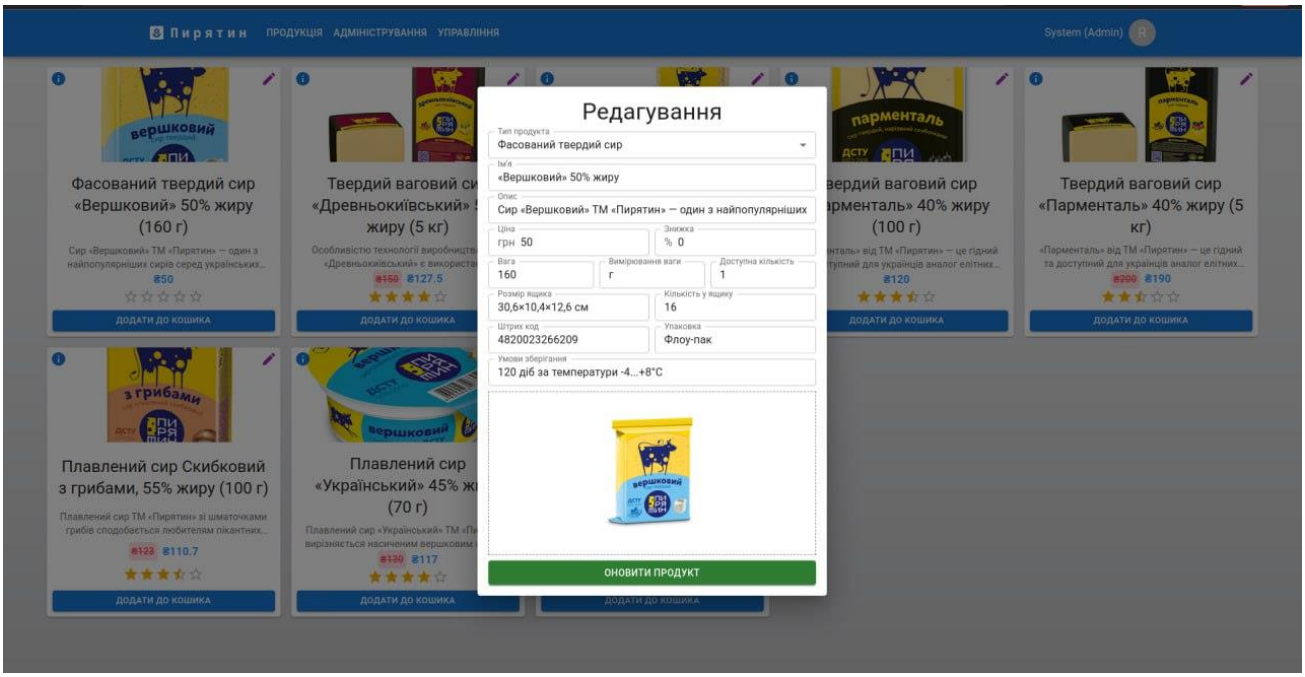



Рис. 3.31 Редагування продукту

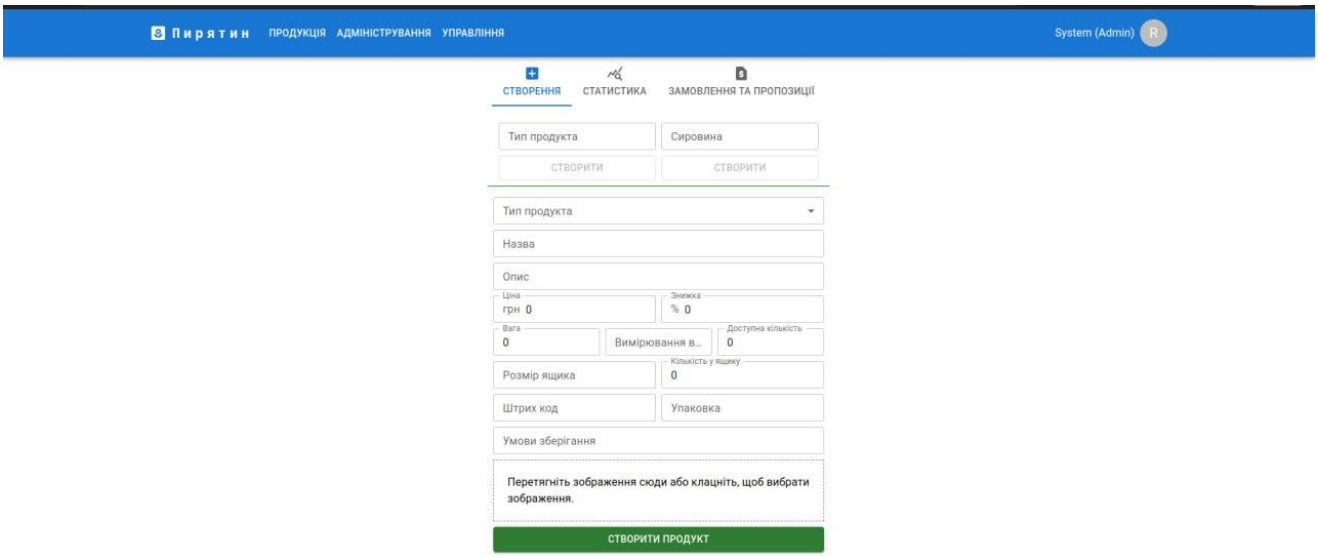



Рис. 3.32 Форми для створення

Адміністратори веб-додатку мають можливість створювати нових менеджерів та адміністраторів, рисунок 3.33. Це дозволяє розширювати команду адміністрування та делегувати відповідальність за управління системою різним співробітникам з необхідними повноваженнями.

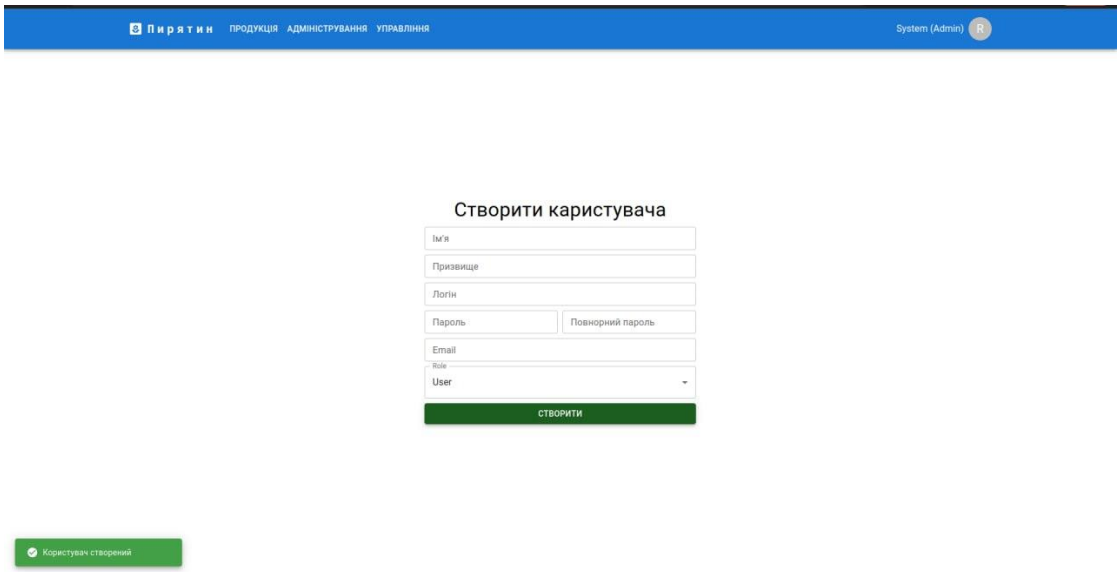



Рис. 3.33 Адмін-форма для створення персоналу

Менеджери веб-додатку мають можливість переглядати та керувати замовленнями на продукцію та заявками на поставку інгредієнтів, рисунок 3.34. Це надає їм зручний та ефективний спосіб управління процесом замовлень та поставок. Основні можливості, які доступні для менеджерів, включають:

Перегляд замовлень на продукцію: Менеджери можуть переглядати список активних та завершених замовлень на продукцію. Вони можуть отримати інформацію про замовника, деталі замовлення, кількість продукції, дату та стан замовлення, рисунок 3.35.

Керування замовленнями: Менеджери можуть змінювати статус замовлення, наприклад, позначати його як оброблене, відправлене чи доставлене. Вони також можуть оновлювати інформацію про замовлення, наприклад, змінювати кількість продукції або внести інші корективи.

Перегляд заявок на поставку інгредієнтів: Менеджери можуть переглядати список заявок на поставку інгредієнтів від постачальників. Вони можуть отримати інформацію про інгредієнти, кількість, ціну та інші деталі, необхідні для організації поставок.

Керування заявками на поставку: Менеджери можуть затверджувати або відхиляти заявки на поставку інгредієнтів. Вони також можуть змінювати інформацію про заявки, якщо необхідно.

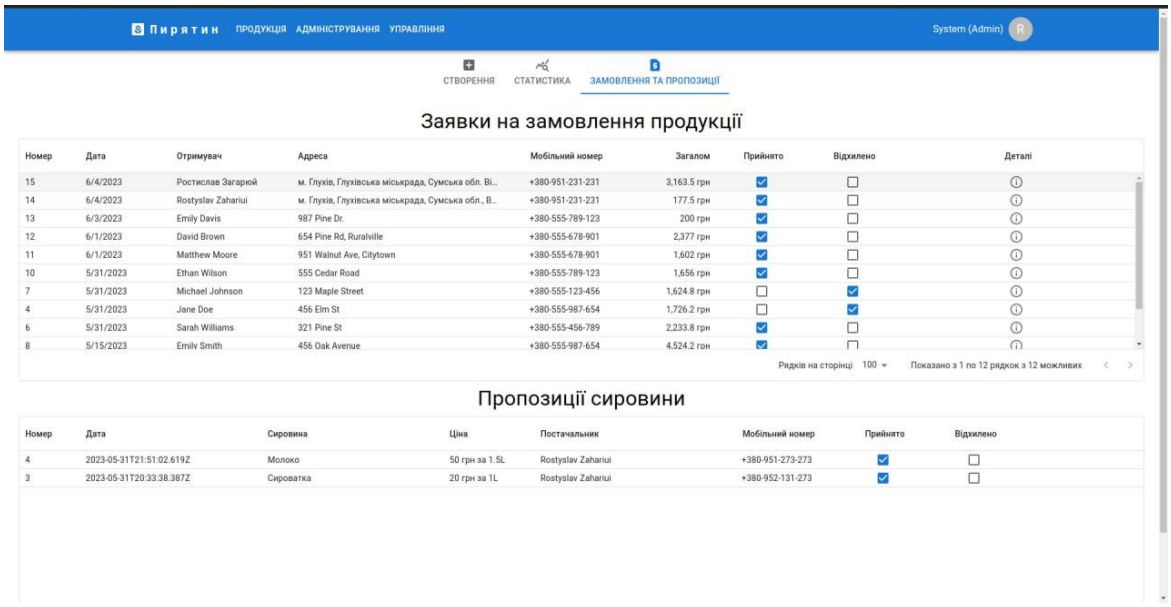



Рис. 3.34 Інтерфес для менеджменту заявок

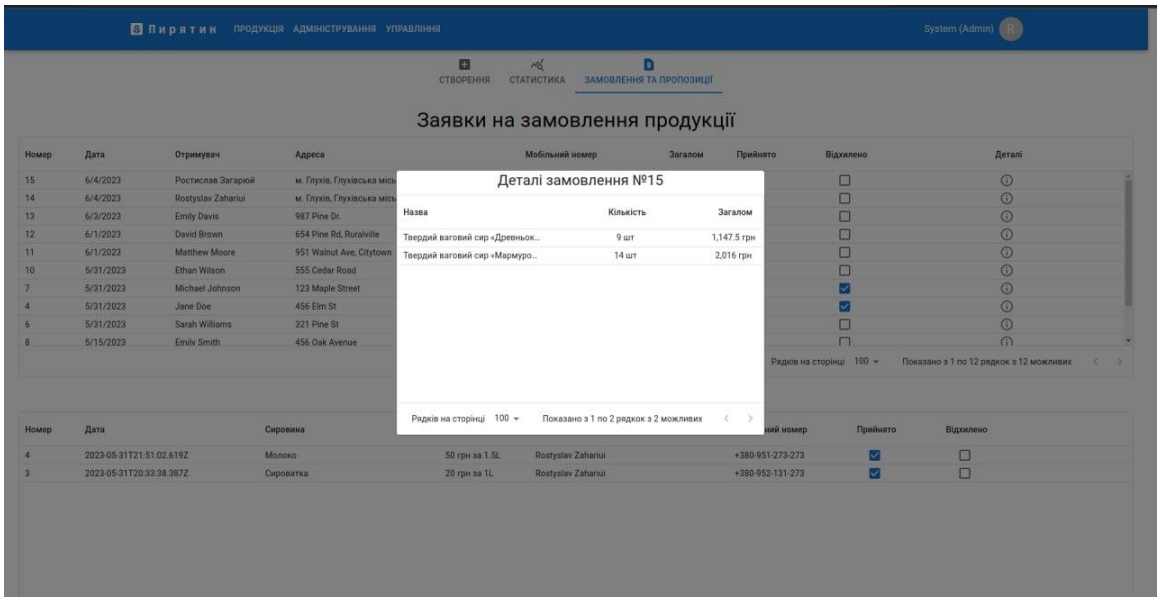



Рис. 3.35 Перегляд деталей замовлення

Менеджери мають можливість переглядати статистику замовлень за певний період, де вони можуть вказати дві дати для вибору проміжку часу, рисунок 3.36.

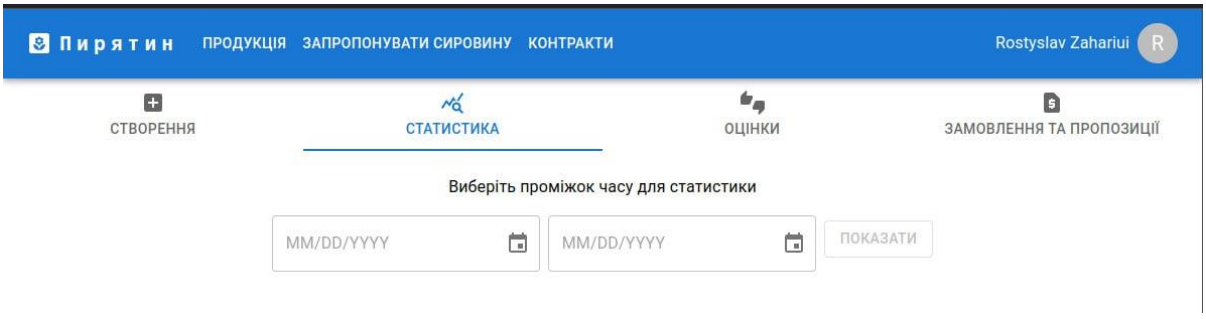



Рис. 3.36 Вибір промужку часу

Графік "Загальна сума замовлень для кожного клієнта" є інструментом для аналізу та управління замовниками. Основна мета цього графіка полягає в тому, щоб візуалізувати загальний обсяг замовлень, зроблених кожним клієнтом, та визначити, які замовники становлять значну частку в загальному обсязі продажів.

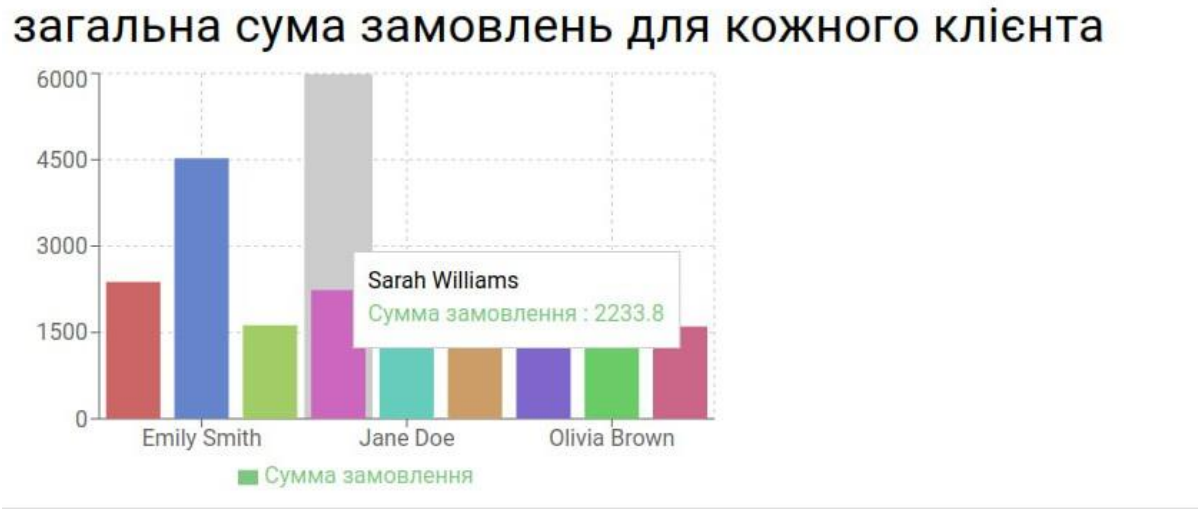



Рис. 3.37 Діаграма сум замовлень клієнтами

Кругова діаграма "Співвідношення замовлень за типом сиру" є інструментом для вивчення та аналізу популярності різних типів сирів серед замовників. Він надає візуальне представлення розподілу замовлень за типами сирів і дозволяє виявити ключові тенденції та переваги споживачів.


Рис. 3.38 Співвідношення замовлень за типом сиру

Діаграма "Найпопулярніші продукти" є інструментом для вивчення та аналізу популярності конкретних продуктів серед замовників. Він надає візуальне представлення топ-продуктів, які мають найвищий рівень популярності. та попиту серед клієнтів.

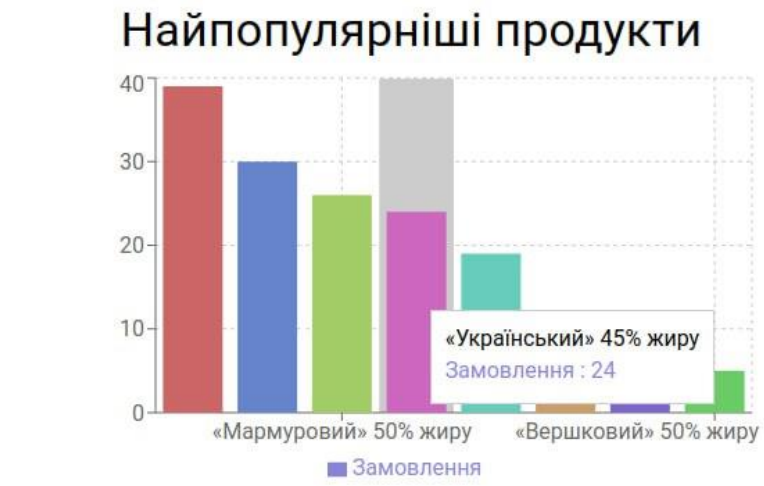



Рис. 3.39 Діаграма топ продуктів

Графік "Продукти, які потребують аналізу" є інструментом для ідентифікації продуктів з найгіршими відгуками та вимагають додаткового аналізу та уваги. Він надає візуальне представлення продуктів, які мають низький рейтинг або негативні відгуки від клієнтів.

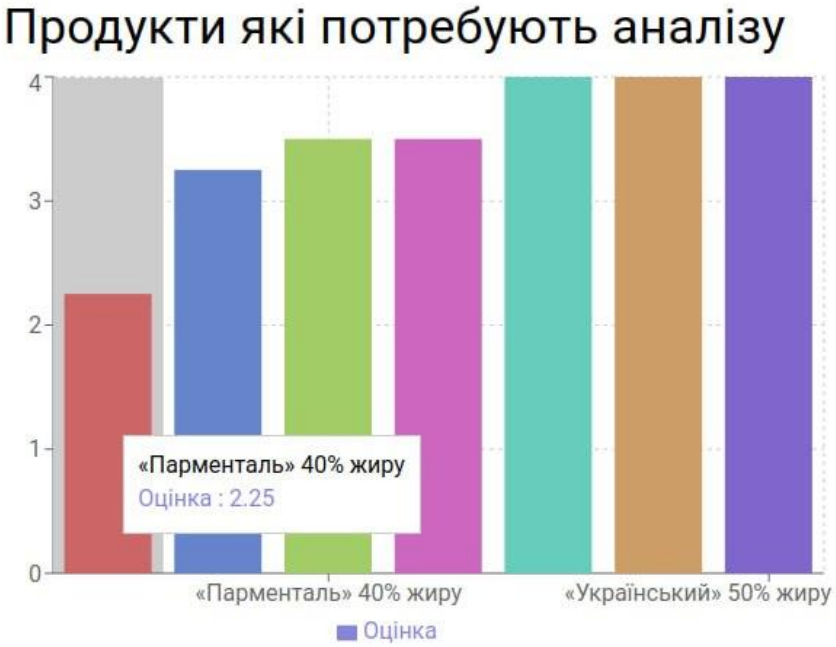



Рис. 3.40 Діаграма оцінок продуктів

Ну і звичайно, у веб-додатку присутні форми для реєстрації користувачів та авторизації. Ці форми є важливими компонентами системи, що дозволяють користувачам отримати доступ до функціоналу та персоналізованих можливостей.

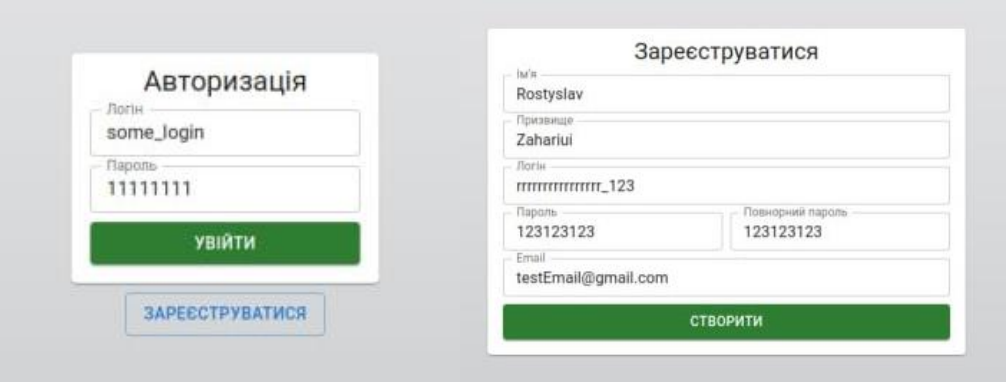



Рис. 3.41 Форми для реєстрації та авторизації

## 3.4. Технічне та системне забезпечення розробки

### 3.4.1. Обґрунтування вибору технічних засобів

- Visual Studio Code: Visual Studio Code є потужним та популярним редактором коду з великою кількістю розширень та плагінів. Він надає зручне та ефективне середовище розробки, забезпечуючи багатофункціональний редактор, підсвічування синтаксису, автодоповнення, налагодження та багато іншого. [24].
- JavaScript: JavaScript є однією з найпопулярніших мов програмування, особливо для веб-розробки. Вона має широку спільноту розробників, багатий екосистему та безліч засобів розробки. Використання JavaScript дозволяє створити динамічні та інтерактивні веб-додатки, що відповідають сучасним вимогам ринку [25].
- NestJS: NestJS є прогресивним фреймворком для створення ефективних та масштабованих серверних додатків на Node.js. Він базується на принципах Angular та використовує TypeScript як основну мову програмування. NestJS надає покращену структуру та організацію коду, механізми інжекції залежностей, пайпи, фільтри, міжпроцесову комунікацію та багато іншого.
- TypeORM: TypeORM є популярною ORM (Object-Relational Mapping) бібліотекою для TypeScript та JavaScript. Вона надає зручний спосіб взаємодії з реляційними базами даних через об'єктно-орієнтовану модель. TypeORM дозволяє зручно виконувати запити до бази даних,

використовувати міграції для керування схемою бази даних та працювати з сутностями.

- PostgreSQL: PostgreSQL є потужною та розширюваною об'єктно-реляційною системою керування базами даних (ORDBMS). Вона надає широкий набічр функцій, надійність та підтримку стандартів. PostgreSQL є популярним вибором для веб-додатків, оскільки вона підтримує географічні дані, розширення для роботи з JSON, XML та багато інших. Використання PostgreSQL дозволяє створити потужну та масштабовану базу даних.

### 3.4.2. Розробка і обґрунтування стратегії адміністрування системи

Розробка та визначення стратегії адміністрування системи є важливим етапом у процесі створення додатку для моніторингу вподобань споживачів продукції сирзаводу. Основні аспекти, що були враховані при формулюванні цієї стратегії, включають:

1. Аутентифікація та авторизація: Веб-додаток має механізми реєстрації, авторизації та аутентифікації. При вході користувача в свій обліковий запис йому надається унікальний ідентифікатор у формі cookie, який залишається активним на протязі однієї години, після чого потрібно знову ввести дані для доступу.
2. Права доступу: Додаток для моніторингу вподобань споживачів передбачає різні рівні доступу для різних користувачів, таких як адміністратори, співробітники сирзаводу та звичайні користувачі.
3. Документація: Веб-додаток має вбудований механізм самодокументування для серверної частини. Це дозволяє з легкістю тестувати запити та без проблем оновлювати серверну частину.

### 3.4.3. Заходи захисту від несанкціонованого доступу до системи

Для запобігання несанкціонованому доступу до системи, веб-додаток для моніторингу вподобань споживачів продукції сирзаводу впроваджує наступні заходи безпеки:

- Процес авторизації користувача.

- Використання системи ролей, що визначають права доступу (адміністратор, аналітик, споживач).
- Використання технології JWT (JSON Web Token) для ідентифікації користувача.
- Захист вразливих даних шляхом їх приховування.

# РОЗДІЛ 4. ОХОРОНА ПРАЦІ

Охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

Роботодавець - власник підприємства, установи, організації або уповноважений ним орган, незалежно від форм власності, виду діяльності, господарювання, і фізична особа, яка використовує найману працю.

Працівник - особа, яка працює на підприємстві, в організації, установі та виконує обов'язки або функції згідно з трудовим договором (контрактом).

При розробці веб-додатку для підтримки роботи з замовниками продукції Пирятинського сирзаводу, важливо враховувати законодавчі норми України, що стосуються охорони праці. Основний закон України, який регулює цю сферу - це Закон України "Про охорону праці" [26].

Цей Закон регламентує основні положення з питань охорони праці, зокрема відносно безпеки, гігієни праці, промислової санітарії, а також права та обов'язки працівників та роботодавців у цій сфері.

У контексті розробки веб-додатку, основними аспектами охорони праці є забезпечення безпеки та здоров'я програмістів під час процесу розробки, а також забезпечення безпеки даних користувачів додатку.

Під час розробки веб-додатку, роботодавець повинен забезпечити безпечні та здорові умови праці для програмістів. Відповідно до статті 14 Закону України "Про охорону праці" працівник зобов'язаний:

- дбати про особисту безпеку і здоров'я, а також про безпеку і здоров'я оточуючих людей в процесі виконання будь-яких робіт чи під час перебування на території підприємства;
- знати і виконувати вимоги нормативно-правових актів з охорони праці, правила поводження з машинами, механізмами, устаткуванням та іншими засобами виробництва, користуватися засобами колективного та індивідуального захисту;

- проходити у встановленому законодавством порядку попередні та періодичні медичні огляди.

Працівник несе безпосередню відповідальність за порушення зазначених вимог. При виконанні роботи за трудовим договором про дистанційну роботу, про надомну роботу працівник самостійно визначає своє робоче місце та несе відповідальність за забезпечення безпечних і нешкідливих умов праці на ньому, а роботодавець несе відповідальність за безпечність і належний технічний стан обладнання та засобів виробництва, що передаються працівнику для виконання дистанційної або надомної роботи. При виконанні роботи за трудовим договором про надомну роботу визначене працівником робоче місце має характеризуватися наявністю закріпленої зони, технічних засобів (основних виробничих і невиробничих фондів, інструменту, приладів, інвентарю) або їх сукупності, необхідних для виробництва продукції, надання послуг, виконання робіт або функцій, передбачених установчими документами.

# ВИСНОВКИ

Протягом написання кваліфікаційної роботи було проведено дослідження щодо співпраці з замовниками продукції ТОВ "Пирятинський сирзавод". Досліджено та здійснені оцінки різних аспектів бізнес-процесу, визначено ключові виклики та можливості, які він представляє.

Аналіз підприємства включав загальний огляд організації, оцінку поточної комп'ютеризації, виявлення проблем і пропозицію щодо рішення - створення веб-додатку. Було проведено аналіз подібних програмних продуктів, виділені їх переваги та недоліки, обговорена доцільність розробки власного веб-додатку, підрахований економічний ефект від його впровадження.

Особливий акцент зроблено на формуванні технічного завдання, яке включає вимоги до додатку, строки його створення, вимоги до переходу та гарантійного обслуговування, а також функціональні можливості.

Веб-додаток для підтримки роботи з замовниками продукції Пирятинського сир заводу було розроблено використовуючи передові технології, такі як React, TypeORM, PostgreSQL, NestJS та новаторські підходи до розподілу логіки між серверною та клієнтською сторонами.

Розроблена інструкція для користувачів системи, яка описує основні модулі додатку з погляду споживача та аналітика.

В цілому, у рамках кваліфікаційної роботи було проведено детальний аналіз діяльності ТОВ "Пирятинський сирзавод", виявлені існуючі проблеми при роботі з замовленнями та запропоноване впровадження веб-додатку для інформаційної підтримки замовників продукції сирзаводу. Додаток, створений за допомогою сучасних технологій і ефективно виконує визначені функції.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний веб-сайт «Молочного Альянсу». [Електронний ресурс] – URL: https://milkalliance.com.ua/company/enterprises/piryatinskij-sirzavod/ (дата звернення 24.04.2023)

2. Звіт про управління за 2020 рік ТОВ «Пирятинський сирзавод». [Електронний ресурс] – URL: https://milkalliance.com.ua/tools/cms/site/download.php?url=/uploads/site_factory_docs/file/0008/24.pdf&name=zvit-pro-upravlinnya-za-2020-rik (дата звернення 25.04.2023)

3. Офіційний веб-сайт «1С Підприємство». [Електронний ресурс] - URL: https://itez.com.ua/what-is-1c.html (дата звернення 25.04.2023)

4. Офіційний веб-сайт «Microsoft». [Електронний ресурс] - URL: https://www.microsoft.com/en-us/microsoft-365/excel (дата звернення 26.04.2023)

5. Офіційний веб-сайт «ERwin Data Modeler». [Електронний ресурс] - URL: https://www.erwin.com/ (дата звернення 26.04.2023)

6. Офіційний веб-сайт «Wordpress». [Електронний ресурс] - URL: https://wordpress.org/ (дата звернення 29.04.2023)

7. Офіційний веб-сайт «Shopify». [Електронний ресурс] - URL: https://www.shopify.com/ (дата звернення 29.04.2023)

8. Офіційний веб-сайт «Magento». [Електронний ресурс] - URL: https://business.adobe.com/products/magento/magento-commerce.html (дата звернення 29.04.2023)

9. Офіційний веб-сайт «Opencart». [Електронний ресурс] - URL: https://www.opencart.com/ (дата звернення 29.04.2023)

10. Фінансово-економічний словник. Загородній А.Г., Вознюк Г.Л. Фінансово-економічний словник. – К.: Знання , 2008. – 1072с.

11. М'якшило, О. М. CASE-технології у проектуванні інформаційних систем [Електронний ресурс] [Текст] : навч. посіб. / О. М. М'якшило, Л. Г. Загоровська. — Київ : НУХТ, 2017. — 190 с. — каф. інформаційних систем.

12. Управління IT проектами [Електронний ресурс]: методичні рекомендації до самостійної роботи для здобувачів освітнього ступеня «Бакалавр» спеціальності 122 «Комп'ютерні науки» освітньо-професійних програм «Комп'ютерні науки» та «Інформаційні системи та штучний інтелект» денної та заочної форм навч./уклад. С. В. Грибков, О. Л. Сєдих – К.: НУХТ, 2022 – 27 с.

13. ЗАКОН УКРАЇНИ Про стандартизацію. [Електронний ресурс] - URL: https://zakon.rada.gov.ua/laws/show/1315-18#Text (дата звернення 02.05.2023)

14. ДСТУ 3008-2015. Документація. Звіти у сфері науки і техніки. Структура та правила оформлювання.

15. ДСТУ 3973–2000 Система розроблення та поставлення продукції на виробництво.

16. Офіційний веб-сайт «NestJS». [Електронний ресурс] - URL: https://nestjs.com/ (дата звернення 02.05.2023)

17. Офіційний веб-сайт «TypeORM». [Електронний ресурс] - URL: https://typeorm.io/ (дата звернення 02.05.2023)

18. Офіційний веб-сайт «PostgreSQL». [Електронний ресурс] - URL: https://www.postgresql.org/ (дата звернення 02.05.2023)

19. Веб-сайт вз'язків у TypeORM. [Електронний ресурс] - URL: https://orkhan.gitbook.io/typeorm/docs/many-to-many-relations (дата звернення 02.05.2023)

20. Веб-сайт «MSDN Об'єкт передачі даних». [Електронний ресурс] - URL: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff649585(v=pandp.10)?redirectedfrom=MSDN (дата звернення 06.05.2023)

21. Документація «class-validator». [Електронний ресурс] - URL: https://github.com/typestack/class-validator (дата звернення 06.05.2023)

22. Офіційний веб-сайт «React». [Електронний ресурс] - URL: https://react.dev/learn (дата звернення 06.05.2023)

23. Офіційний веб-сайт «Typescript». [Електронний ресурс] - URL: https://www.typescriptlang.org/ (дата звернення 06.05.2023)

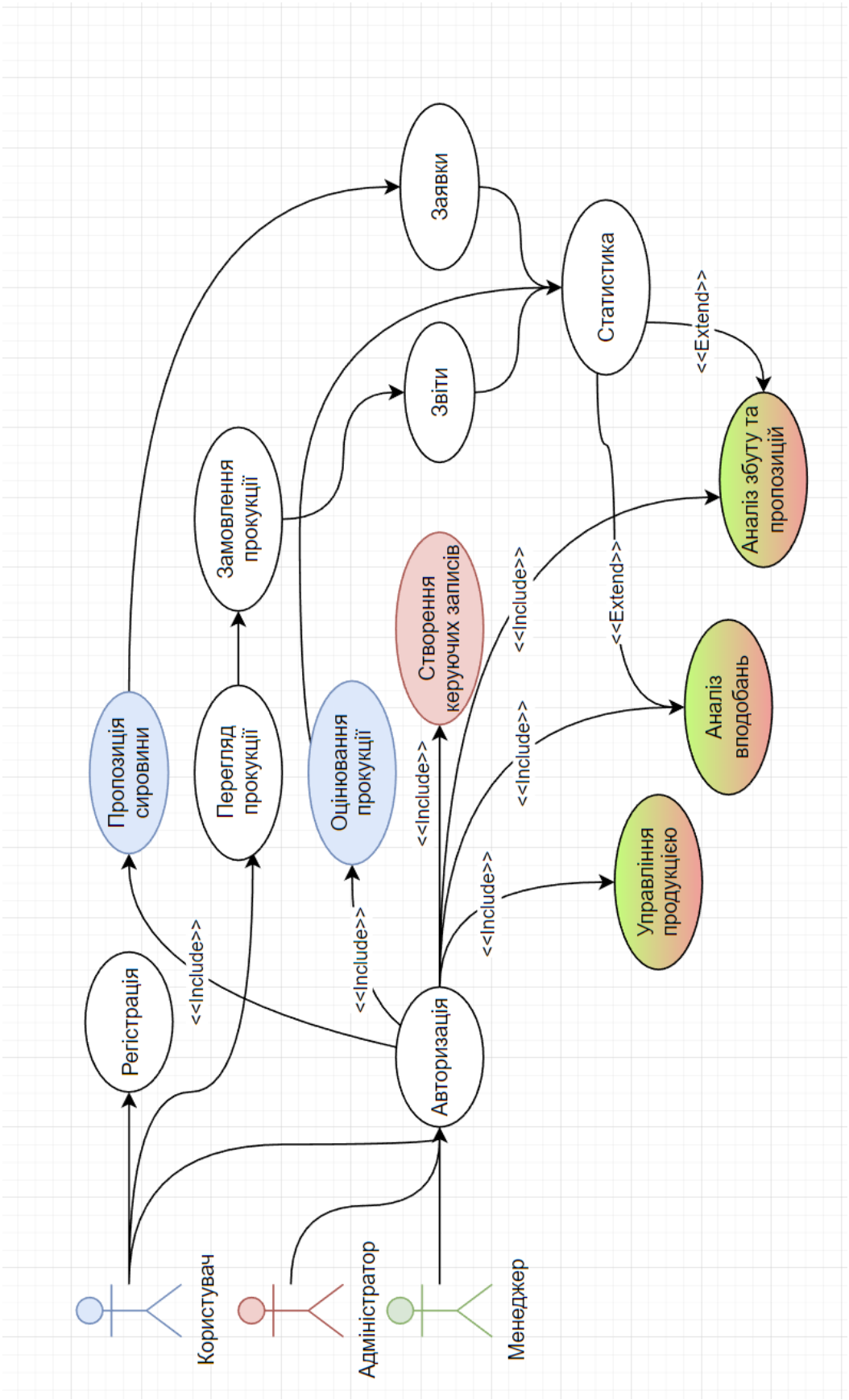
24. Офіційний веб-сайт «Visual Studio Code». [Електронний ресурс] - URL: https://code.visualstudio.com/docs (дата звернення 10.05.2023)

25. Офіційний веб-сайт «Mozilla». [Електронний ресурс] - URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення 10.05.2023)

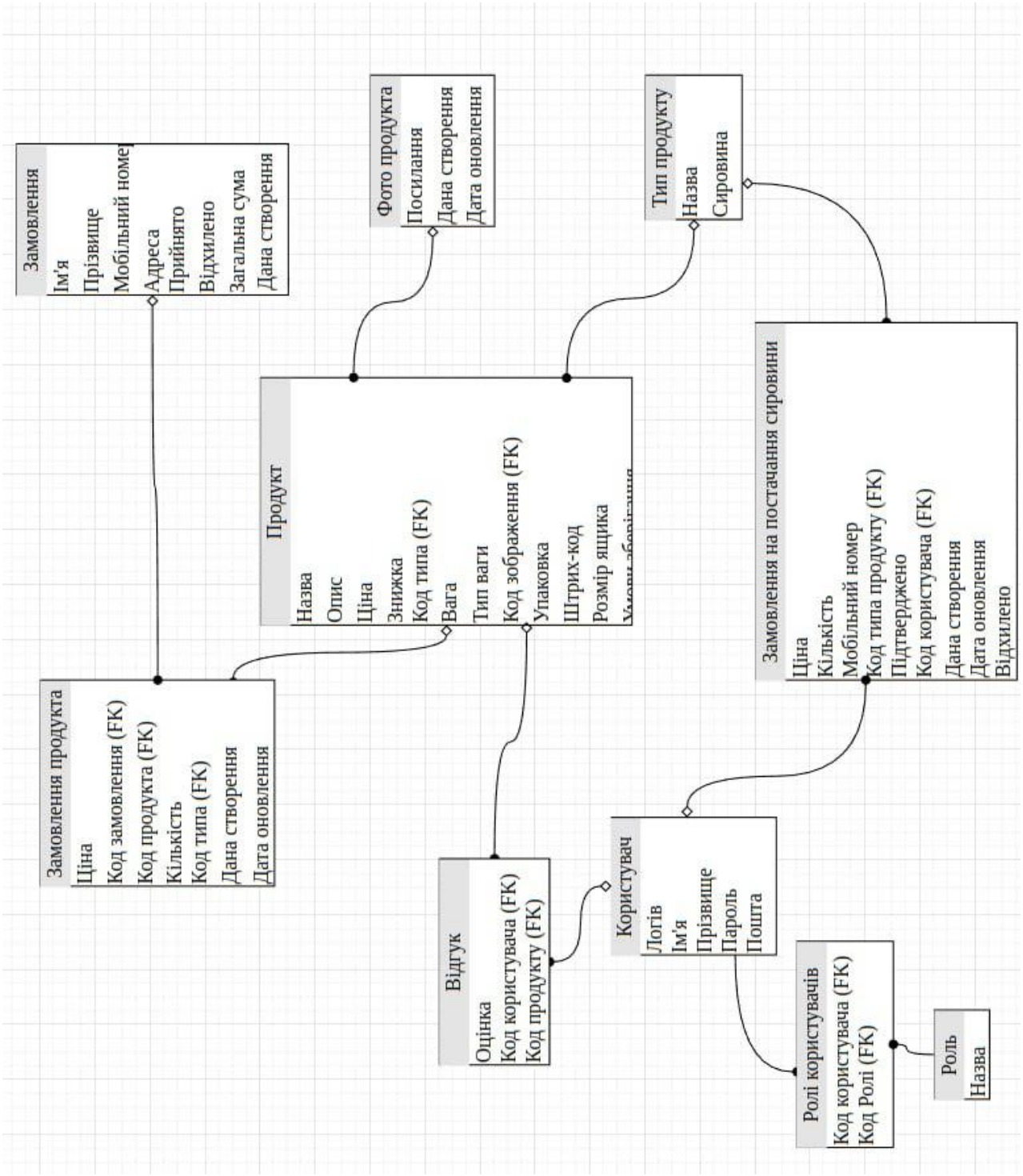
26. ЗАКОН УКРАЇНИ Про охорону праці. [Електронний ресурс] - URL: https://zakon.help/zakonodavstvo-ukraini/2694-12 (дата звернення 10.05.2023)

# ДОДАТКИ

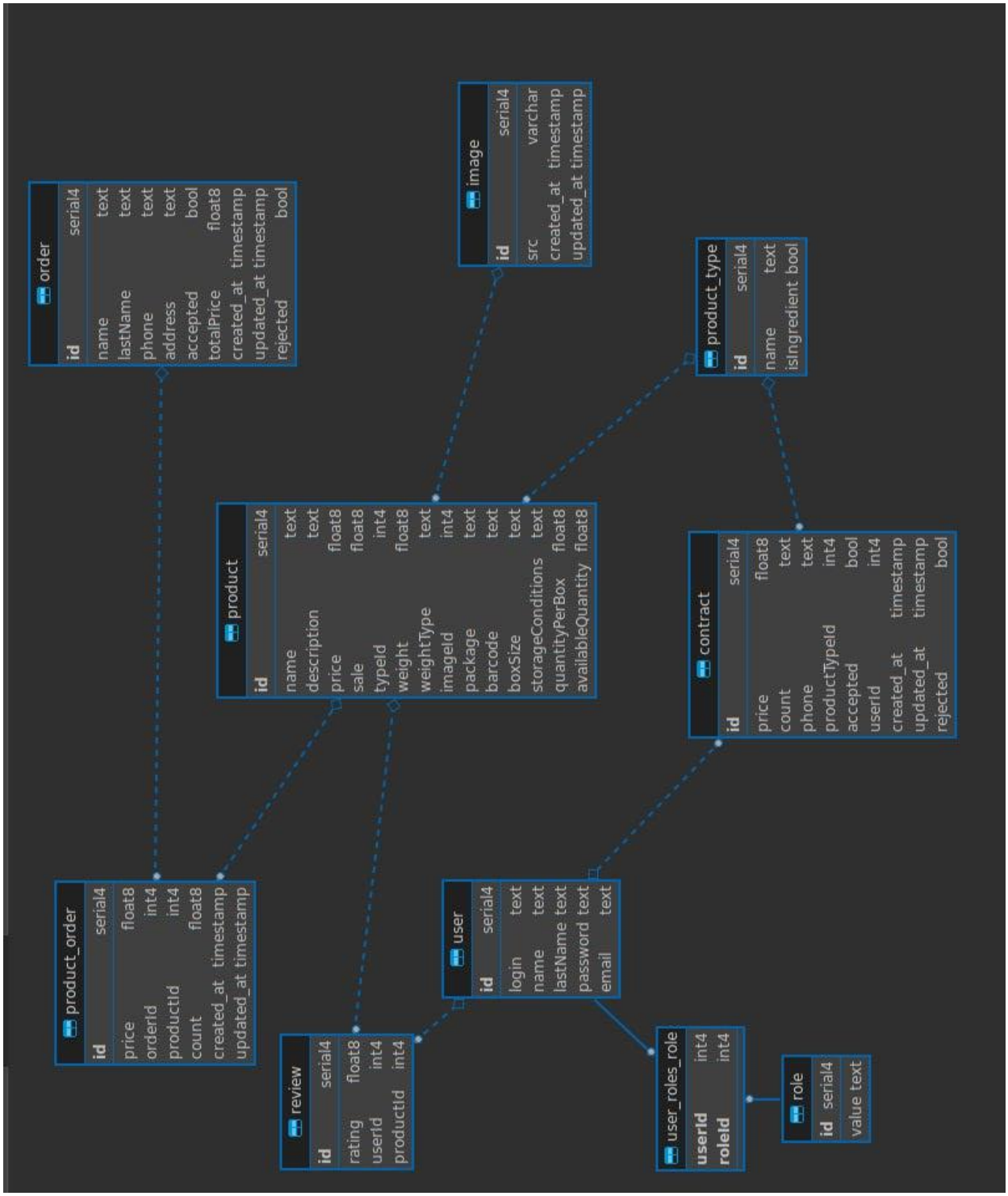
## Додаток А. Діаграма використання системи



## Додаток Б. Логічна схема бази даних



# Додаток В. Фізична схема бази даних



# Додаток Г. Інтерфейс веб-додатку

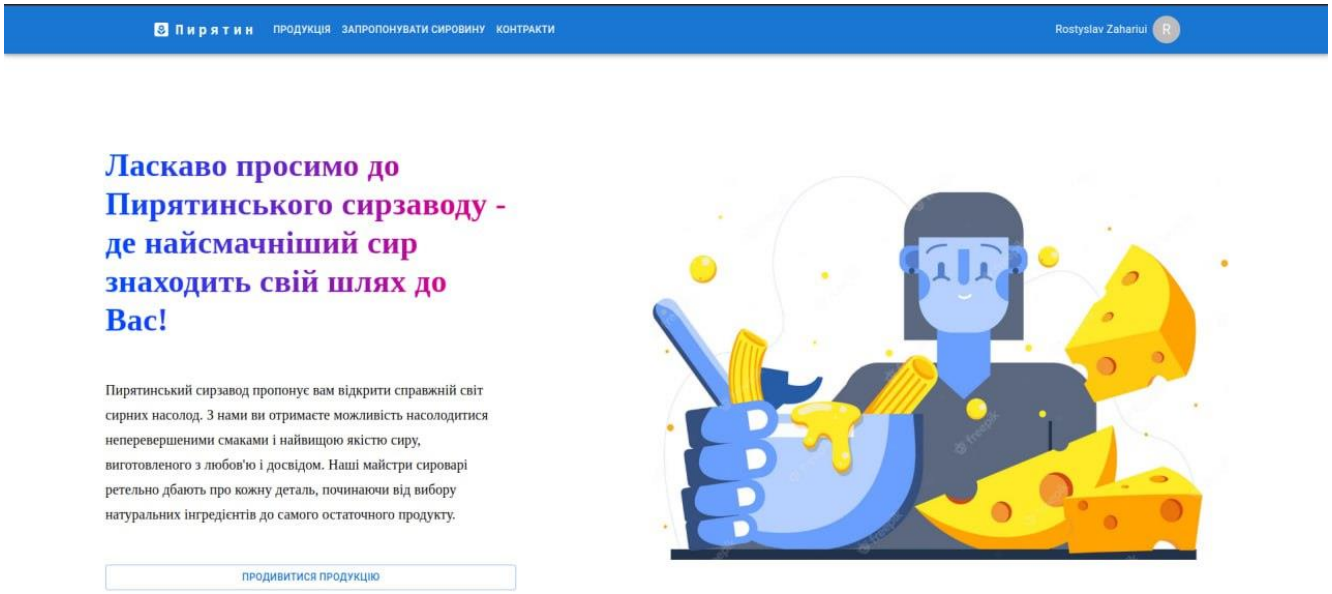



Рис.Г.1 Домашня сторінка

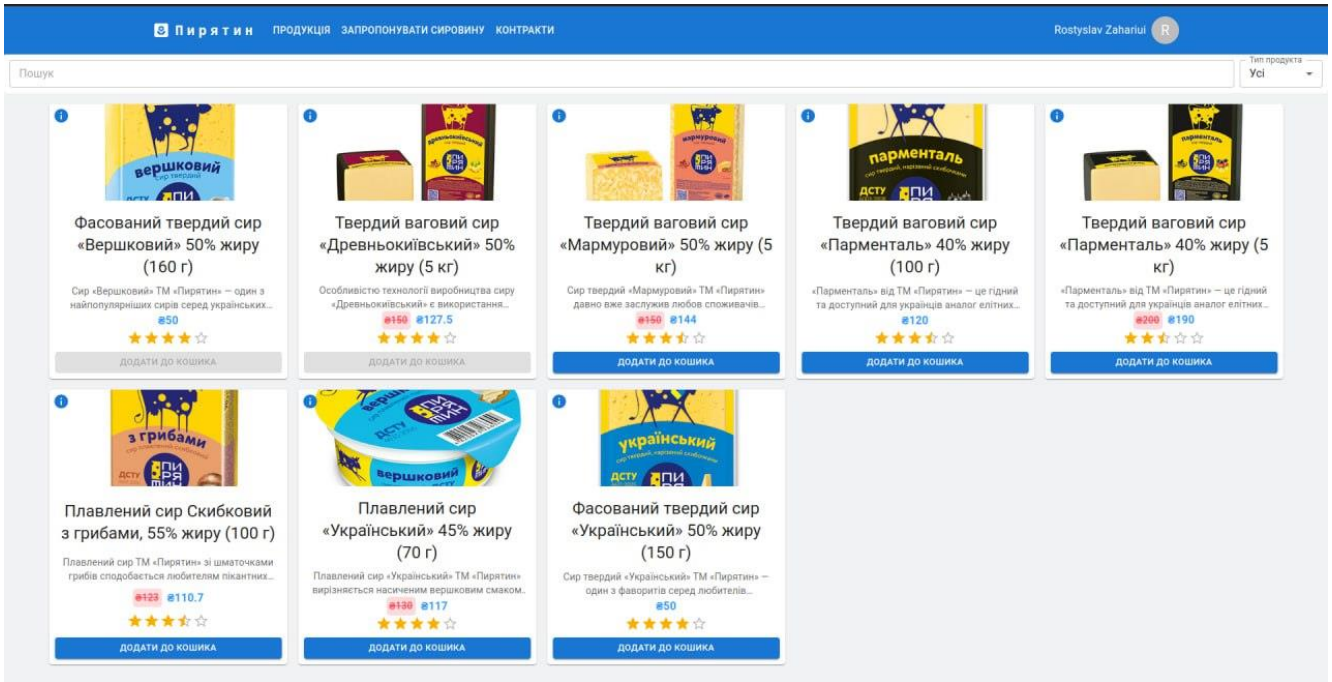



Рис.Г.2 Список продукції

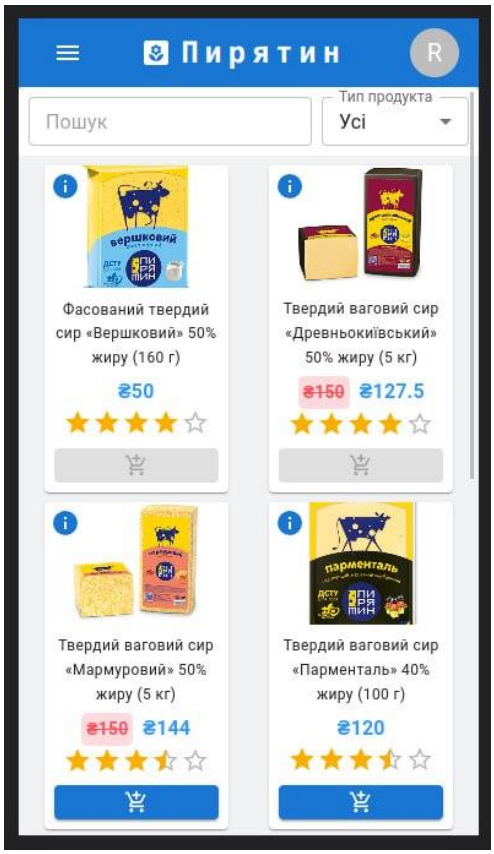



Рис.Г.3 Список продукції на мобільному пристрої

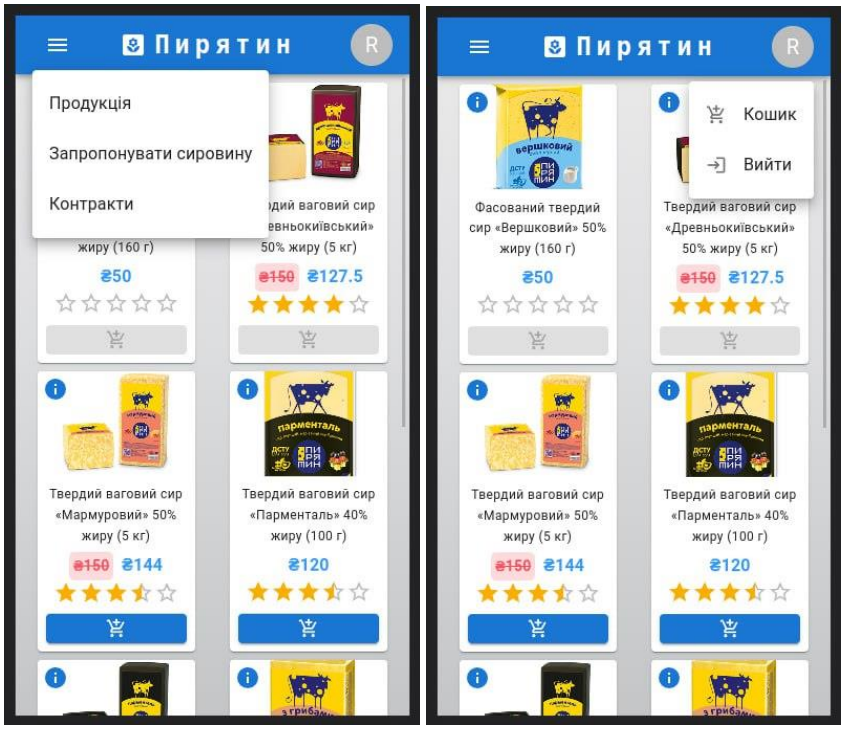



Рис.Г.4 Навігація на мобільному пристрої

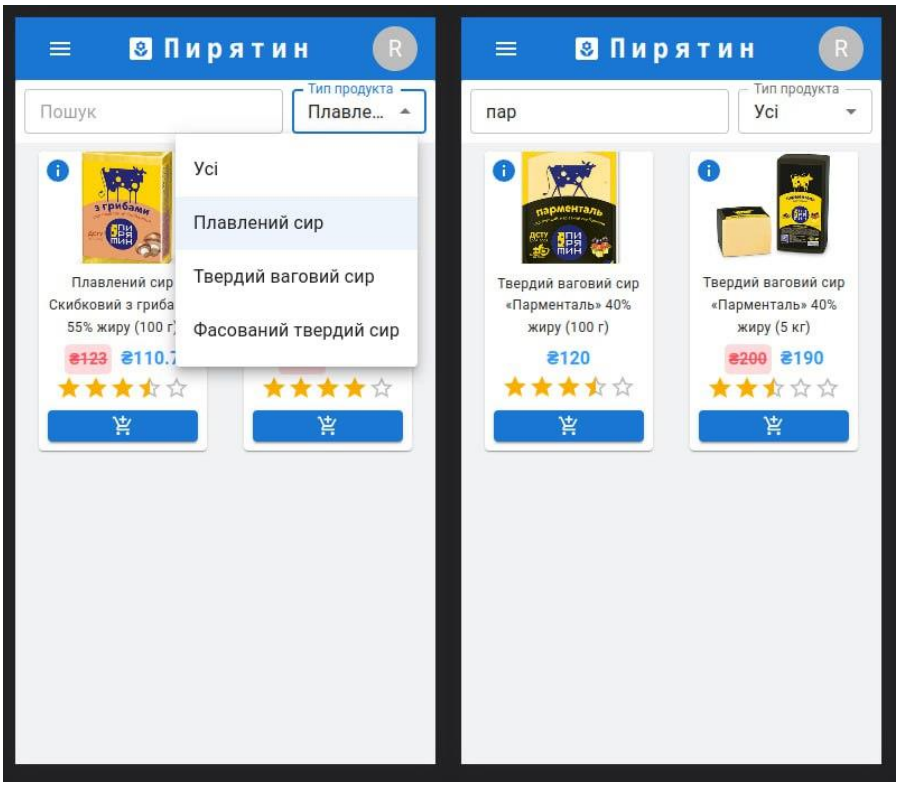



Рис.Г.4 Пошук продукції за фільтрами на мобільному пристрої

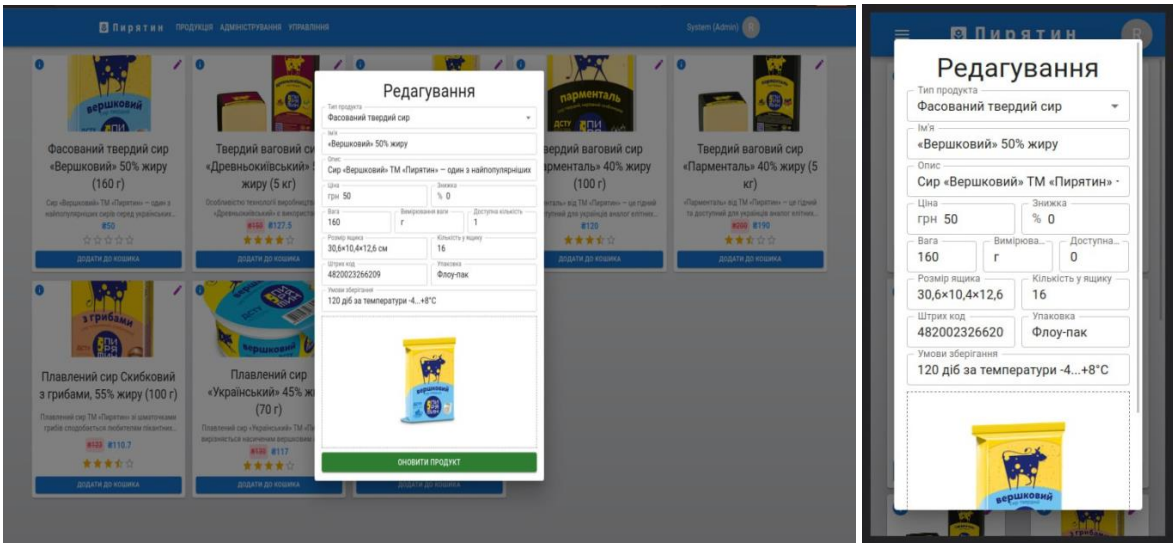



Рис.Г.5 Редагування продукції

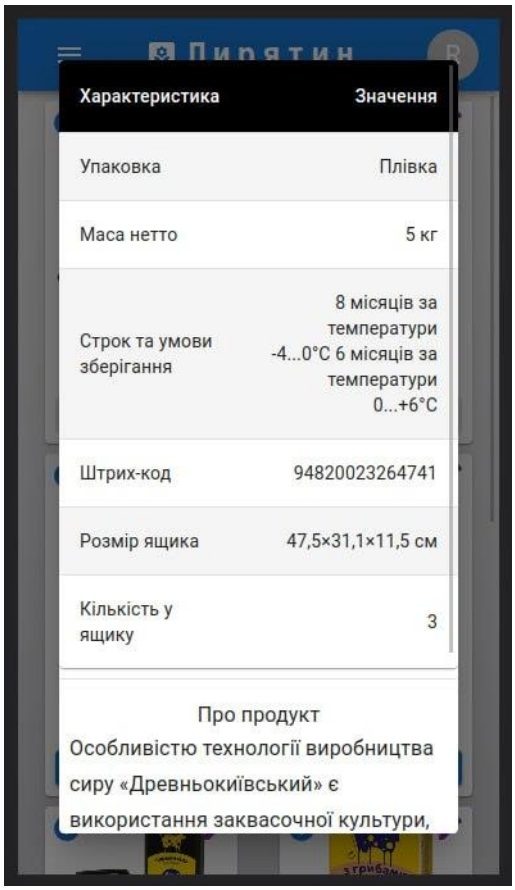



Рис.Г.6 Перегляд детальної інформації про продукцію

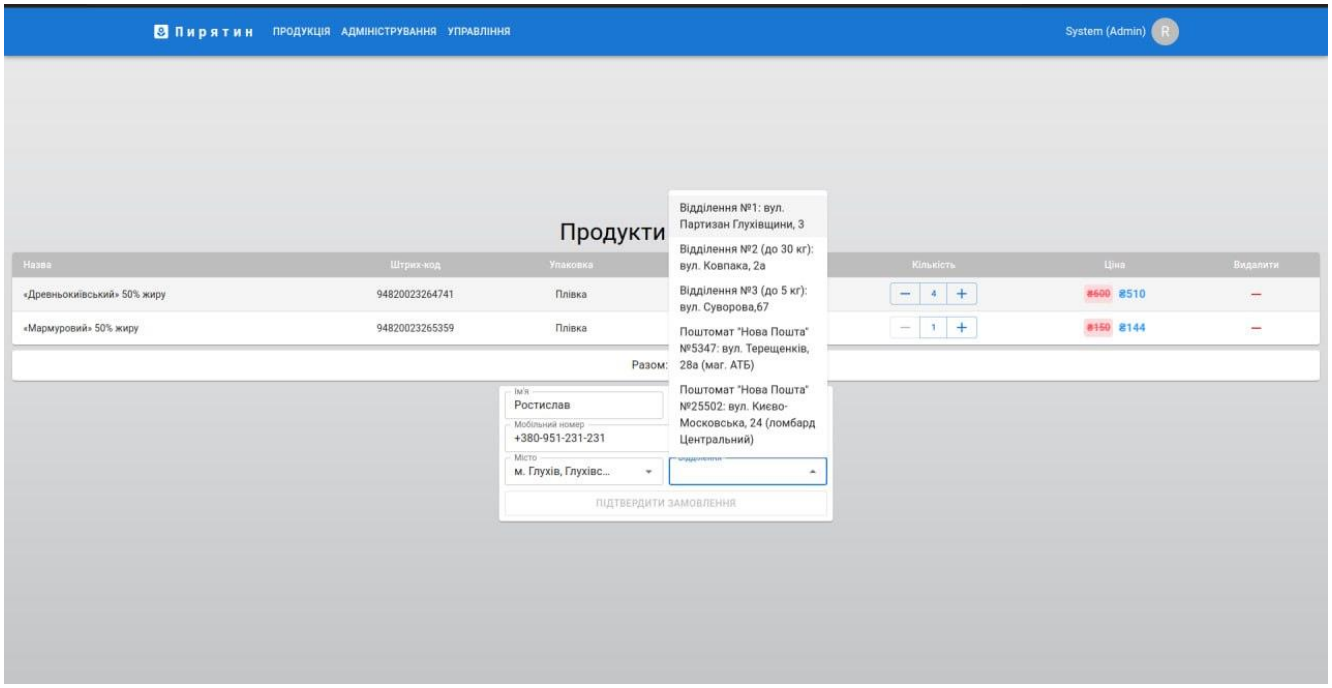



Рис.Г.7 Формування замовлення

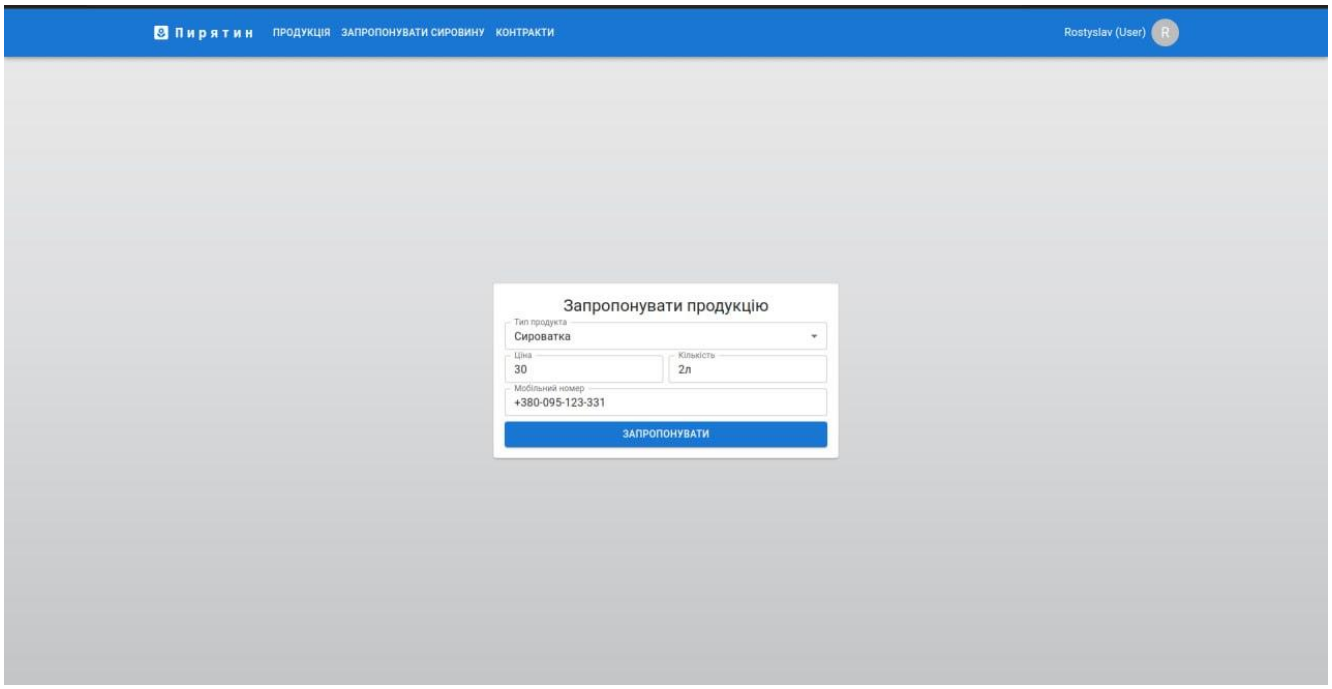



Рис.Г.8 Подання заявки на поставку сировини

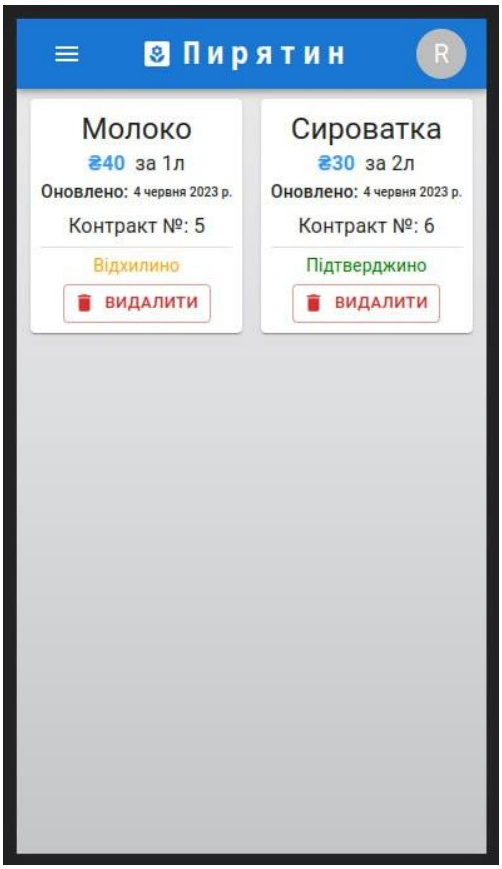



Рис.Г.9 Сторінка перегляду контрактів

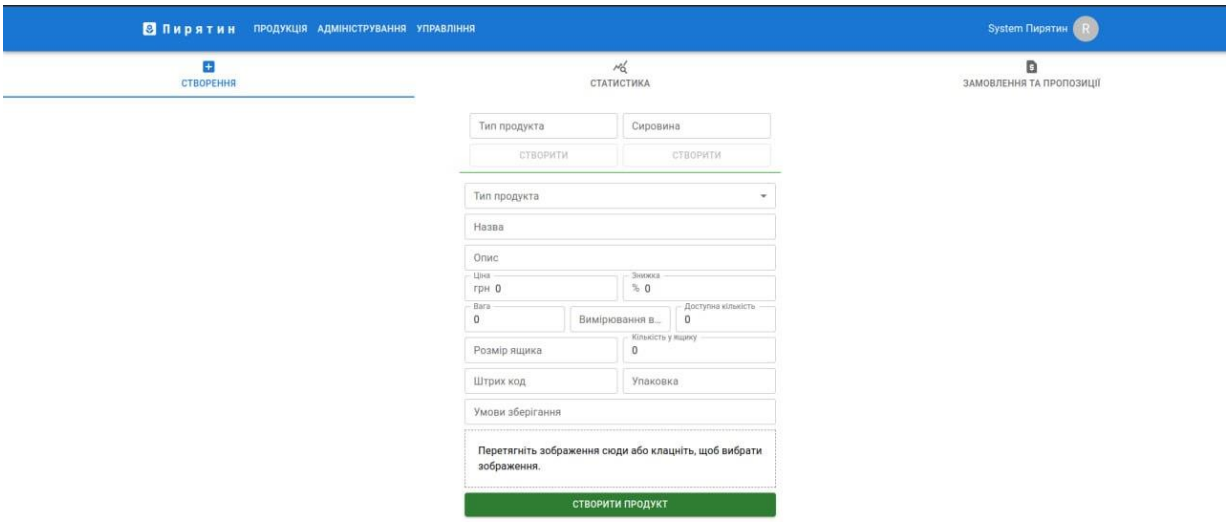



Рис.Г.10 Сторінка для створення типа продукції, продукції та сировини

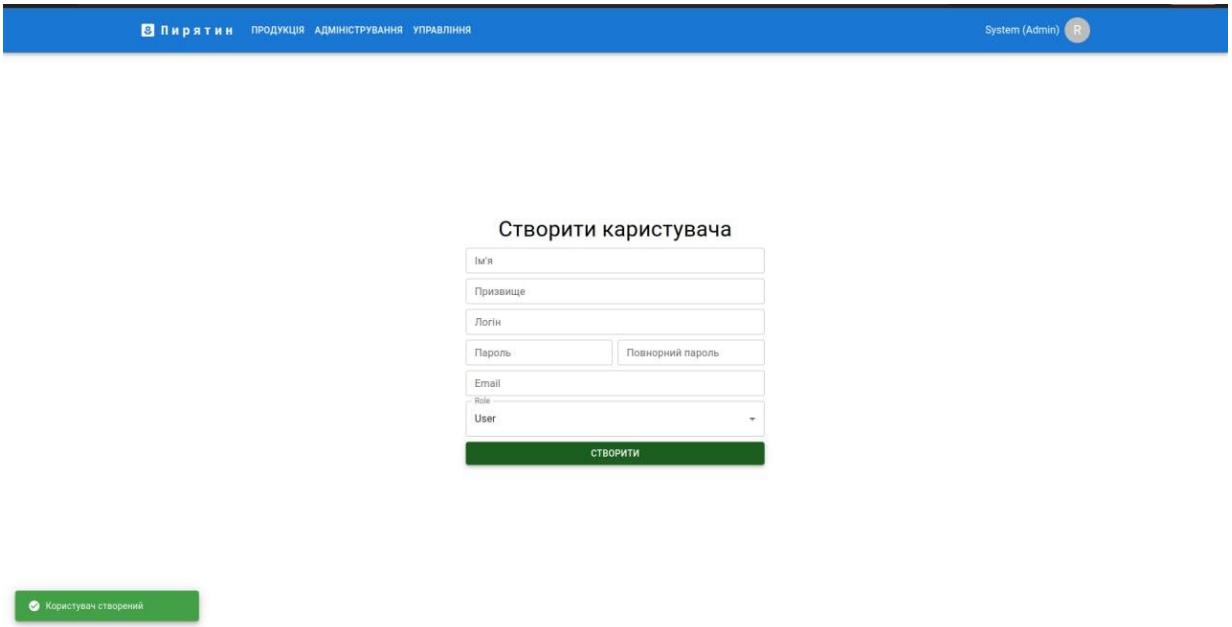



Рис.Г.11 Сторінка для створення нових менеджерів та адмінів

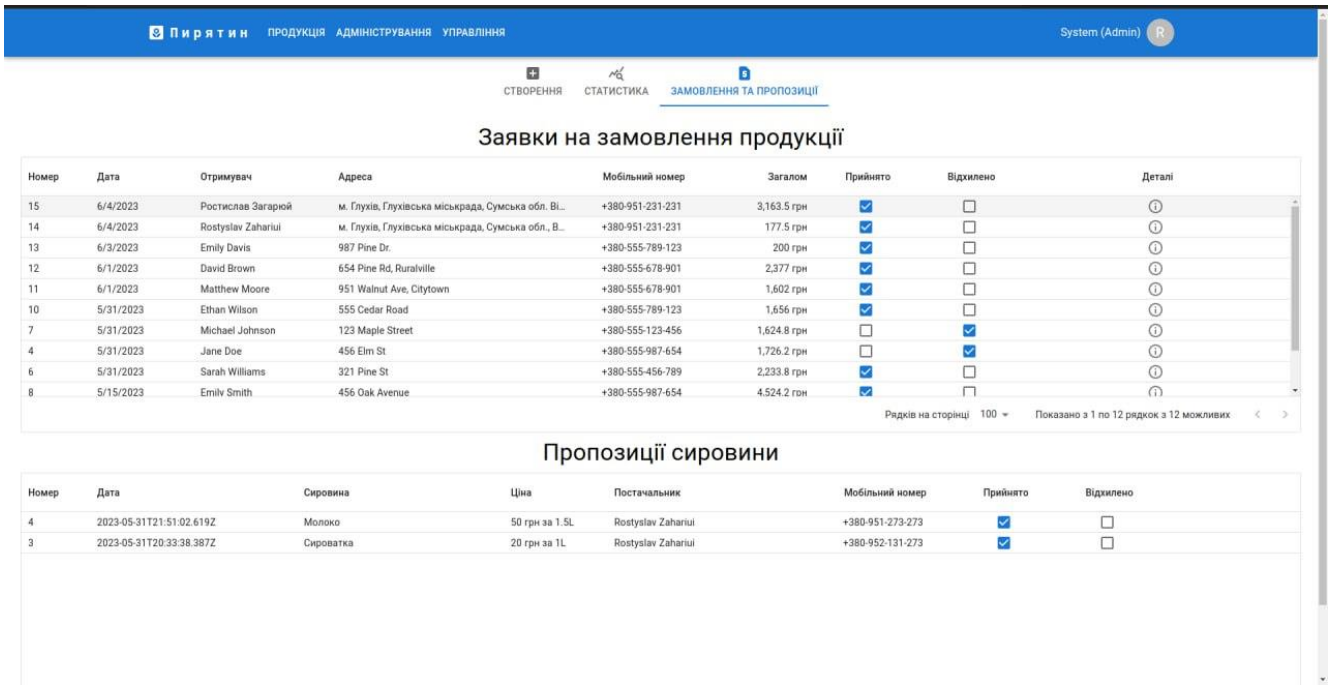



Рис.Г.12 Сторінка менеджменту заповлень та пропозицій

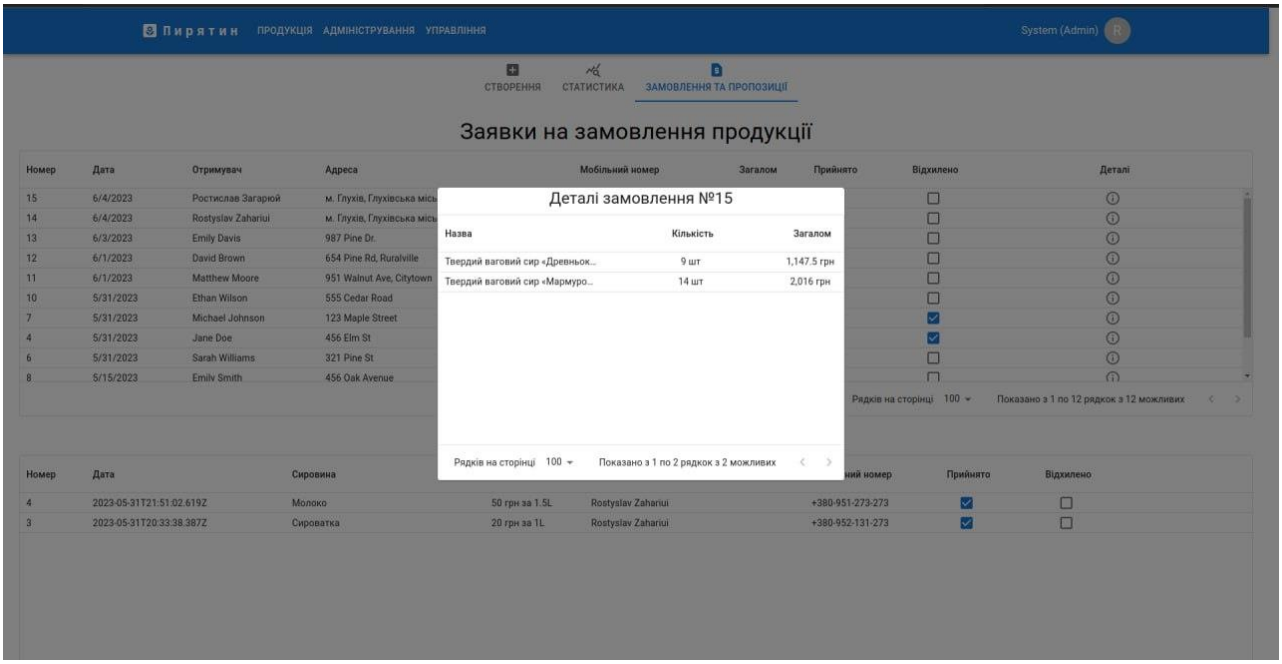



Рис.Г.13 Детальний перегляд замовлення

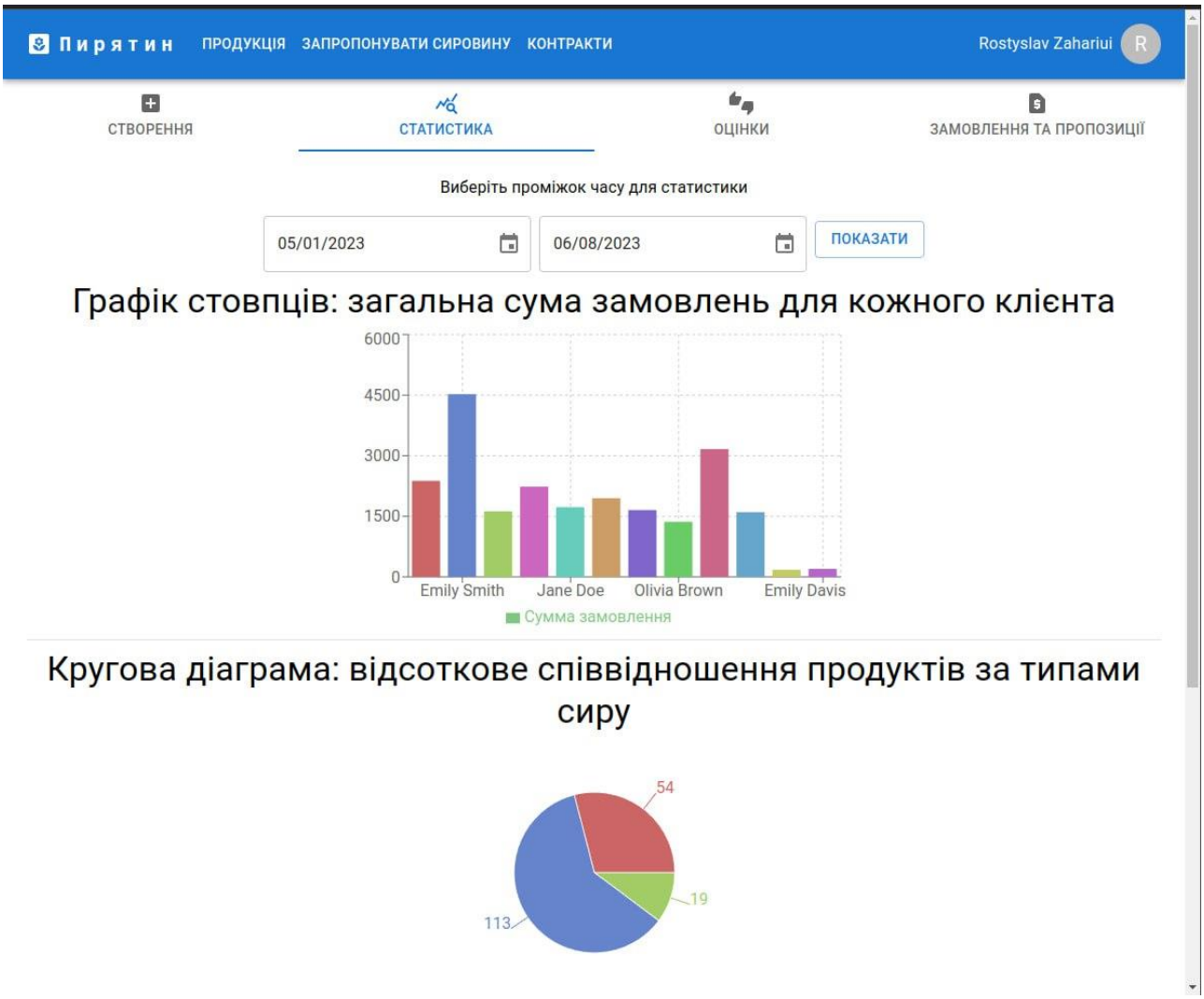



Рис.Г.14 Частина сторінки сратистики замовлень

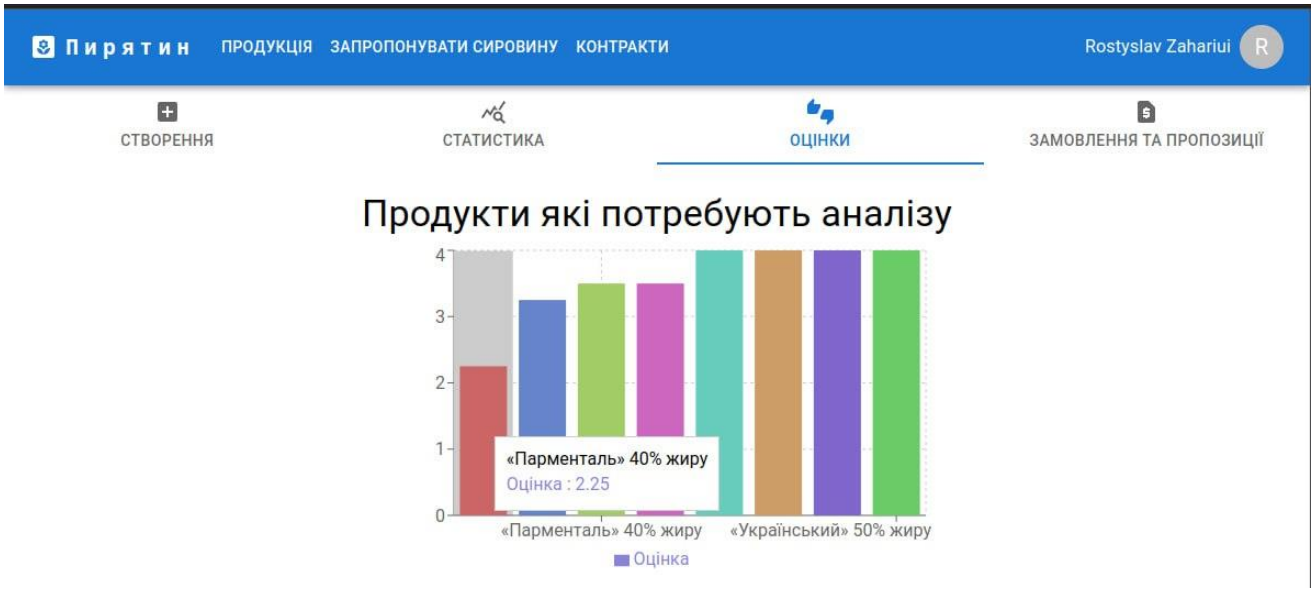



Рис.Г.15 Сторінка статистики оцінок продукції

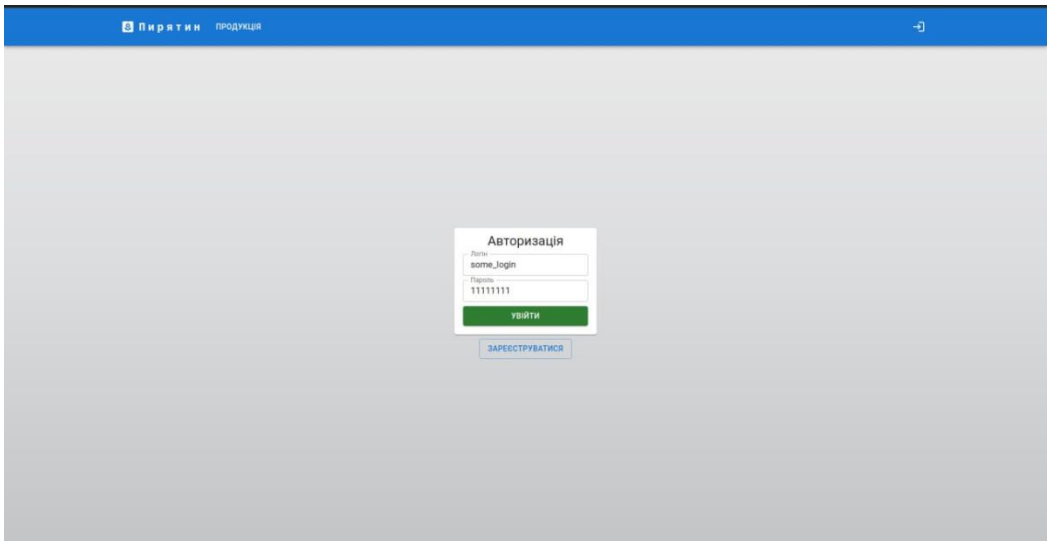



Рис.Г.15 Сторінка авторизації

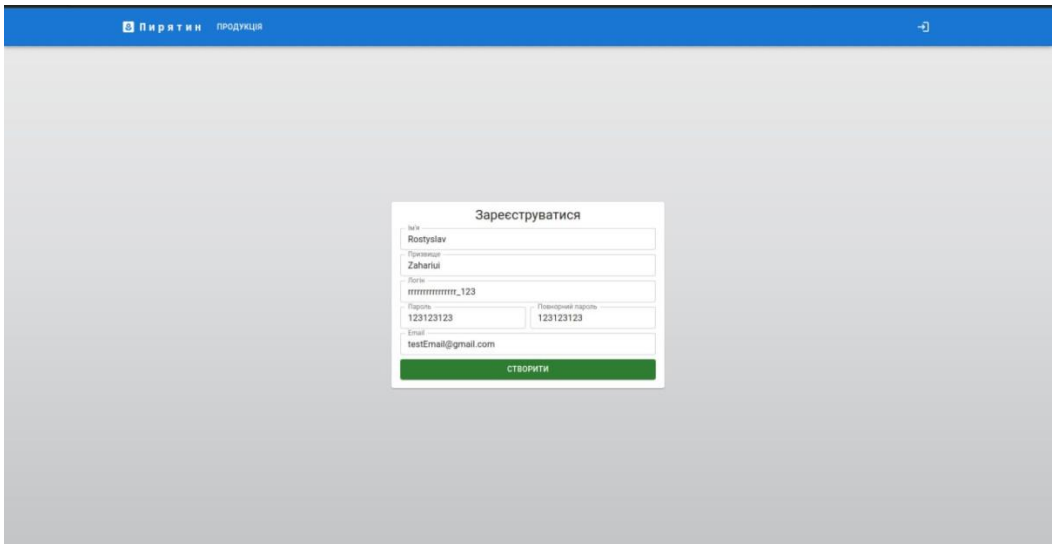



Рис.Г.16 Сторінка реєстрації

## Додаток Д. Програмний код програми

### 1. Хук для виявлення мобільного пристрою (frontend)

```
import { useLayoutEffect, useState } from "react";

export const useIsMobile = (): boolean => {
  const [isMobile, setIsMobile] = useState(false);

  useLayoutEffect(() => {
    const handleResize = () => {
      setIsMobile(window.innerWidth <= 768);
    };
    handleResize();
    window.addEventListener("resize", handleResize);
    return () => {
      window.removeEventListener("resize", handleResize);
    };
  }, []);

  return isMobile;
};
```

### 2. Хук для затримки (frontend)

```
import { useEffect, useState } from "react";

export const useDebounce = <T>(value: T, delay?: number): T => {
  const [debouncedValue, setDebouncedValue] = useState<T>(value);

  useEffect(() => {
    const timer = setTimeout(() => setDebouncedValue(value), delay ||
500);

    return () => {
      clearTimeout(timer);
    };
  }, [value, delay]);

  return debouncedValue;
};
```

### 3. Контекст користувача (frontend)

```
import {
  createContext,
  FC,
  ReactNode,
  useContext,
```

```
  useLayoutEffect,
  useState
} from "react";

import { verifyUser as verifyUserRequest } from "../../api/auth";
import { UserResponse } from "../../api/users/types";
import { AUTH_TOKEN_KEY } from "../../constants";
import { UserContextValue } from "./types";

const UserContext = createContext<UserContextValue>({} as
UserContextValue);

export const UserContextProvider: FC<{ children: ReactNode }> = ({
  children,
}) => {
  const [user, setUser] = useState<UserResponse |
undefined>(undefined);

  const setUserData = (user: UserResponse | undefined) => {
    setUser(user);
    if (!user) {
      localStorage.removeItem(AUTH_TOKEN_KEY);
    }
  };

  const verifyUser = async () => {
    try {
      const token = localStorage.getItem(AUTH_TOKEN_KEY);
      if (token) {
        const res = await verifyUserRequest();
        if (res.data) setUser(res.data);
      }
    } catch (error) {}
  };

  const value = {
    user,
    setUserData,
  };

  useLayoutEffect(() => {
    verifyUser();
  }, []);

  return <UserContext.Provider
value={value}>{children}</UserContext.Provider>;
};
```

```
  export const useUserContext = () => {
    const value = useContext(UserContext);
    return {
      ...value,
    };
  };
```

## 4. Контекст кошика продуктів (frontend)

```
import { useSnackbar } from "notistack";
import { createContext, FC, ReactNode, useContext, useState } from
"react";

import { ProductResponse } from "../../api/product/types";
import { BasketContextValue, BasketProduct } from "./types";

const BasketContext = createContext<BasketContextValue>(
  {} as BasketContextValue
);

export const BasketContextProvider: FC<{ children: ReactNode }> = ({
  children,
}) => {
  const { enqueueSnackbar } = useSnackbar();
  const [basketState, setBasketState] = useState<BasketProduct[]>([]);

  const addToBasket = (product: ProductResponse) => {
    const candidate = basketState.find((p) => p.product.id ===
product.id);
    if (candidate) {
      enqueueSnackbar("Данний продукт вже у кошику");
      return;
    }
    setBasketState((prev) => [...prev, { product, count: 1 }]);
    enqueueSnackbar("Продукт доданий до кошика", { variant: "success"
});
  };

  const countChange = (productID: number, count: number) => {
    setBasketState((prev) =>
      prev.map((bp) => (bp.product.id === productID ? { ...bp, count }
: bp))
    );
  };

  const removeFromBasket = (productId: number) => {
```

```
    setBasketState((prev) => prev.filter((bp) => bp.product.id !==
productId));
  };

  const clearBasket = () => {
    setBasketState([]);
  };

  const value = {
    addToBasket,
    countChange,
    basketState,
    removeFromBasket,
    clearBasket,
    total: Number(
      basketState
        .reduce((acc, { product: { price, sale }, count }) => {
          const priceWithSale = price - (price / 100) * sale;

          return acc + priceWithSale * count;
        }, 0)
        .toFixed(2)
    ),
  };

  return (
    <BasketContext.Provider
value={value}>{children}</BasketContext.Provider>
  );
};

export const useBasketContext = () => {
  const value = useContext(BasketContext);
  return {
    ...value,
  } as typeof value;
};
```

## 5. Компонет для вибору адреси доставки «Nova Poshta» (frontend)

```
import { ChangeEvent, FC, useEffect, useState } from "react";

import { Autocomplete, Stack, TextField } from "@mui/material";

import {
  getNovaPoshtaCities,
  getNovaPoshtaPostOffices
} from "../../api/nova-poshta";
```

```
import {
  NovaPostOfficesToInput,
  PoshtaAddressToInput
} from "../../api/nova-poshta/types";
import { useDebounce } from "../../hooks/useDebounce";

export type NovaPoshtaData = {
  city: string | null;
  address: string | null;
};

export type NovaPoshtaAddressInputProps = {
  onChange: (data: NovaPoshtaData) => void;
};

export const NovaPoshtaAddressInput: FC<NovaPoshtaAddressInputProps> =
({
  onChange,
}) => {
  const [searchCityValue, setSearchCityValue] = useState<string>("");
  const debouncedCityValue = useDebounce<string>(searchCityValue, 500);
  const [poshtaAddress, setPoshtaAddress] =
useState<PoshtaAddressToInput[]>(
    []
  );
  const [selectedCity, setSelectedCity] = useState<PoshtaAddressToInput
| null>(
    null
  );

  const [searchPostOfficesValue, setPostOfficesValue] =
useState<string>("");
  const [postOffices, setPostOffices] =
useState<NovaPostOfficesToInput[]>([]);
  const [selectedPostOffice, setSelectedPostOffice] =
    useState<NovaPostOfficesToInput | null>(null);

  const handleSearchCityValueChange = (
    event: ChangeEvent<HTMLInputElement>
  ) => {
    setSearchCityValue(event.target.value);
  };
  const handlePostOfficesValueChange = (
    event: ChangeEvent<HTMLInputElement>
  ) => {
    setPostOfficesValue(event.target.value);
  };
```

```
  useEffect(() => {
    const fetch = async () => {
      const res = await getNovaPoshtaCities(debouncedCityValue);
      setPoshtaAddress(res);
    };
    fetch();
  }, [debouncedCityValue]);

  useEffect(() => {
    if (!selectedCity) {
      setPostOffices([]);
      setSelectedPostOffice(null);
      return;
    }
    const fetch = async () => {
      const res = await
getNovaPoshtaPostOffices(selectedCity.DeliveryCity);
      setPostOffices(res);
    };
    fetch();
  }, [selectedCity]);

  return (
    <Stack spacing={1} direction="row">
      <Autocomplete
        disablePortal
        options={poshtaAddress}
        fullWidth
        size="small"
        value={selectedCity}
        onChange={(_e, v) => {
          setSelectedCity(v);
        }}
        noOptionsText="Міст не знайдено"
        renderInput={(params) => (
          <TextField
            {...params}
            onChange={handleSearchCityValueChange}
            value={searchCityValue}
            label="Місто"
          />
        )}
      />
      <Autocomplete
        disablePortal
        options={postOffices}
        fullWidth
        size="small"
```

```
        value={selectedPostOffice}
        onChange={(_e, v) => {
          setSelectedPostOffice(v);
          onChange({
            address: v?.label || null,
            city: selectedCity?.Present || null,
          });
        }}
        noOptionsText="Відділень не знайдено"
        renderInput={(params) => (
          <TextField
            {...params}
            onChange={handlePostOfficesValueChange}
            value={searchPostOfficesValue}
            label="Відділення"
          />
        )}
      />
    </Stack>
  );
};
```

## 6. Компонет для вибору проміжку дат (frontend)

```
import { Dayjs } from "dayjs";
import { FC, useState } from "react";

import { Button, Stack } from "@mui/material";
import { AdapterDayjs } from "@mui/x-date-pickers/AdapterDayjs";
import { DatePicker as BaseDatePicker } from "@mui/x-date-
pickers/DatePicker";
import { DemoContainer } from "@mui/x-date-pickers/internals/demo";
import { LocalizationProvider } from "@mui/x-date-
pickers/LocalizationProvider";

import { useIsMobile } from "../../hooks/isMbile";

type DatePickerProps = {
  onSubmit: (first: Dayjs | null, second: Dayjs | null) => void;
};

export const DatePicker: FC<DatePickerProps> = ({ onSubmit }) => {
  const isMobile = useIsMobile();
  const [firstValue, setFirstValue] = useState<Dayjs | null>(null);
  const [secondValue, setSecondValue] = useState<Dayjs | null>(null);
  const [valid, setValid] = useState(true);
```

```
const handleChange = (first: Dayjs | null, second: Dayjs | null) => {
  if (!first || !second) {
    setValid(false);
    return;
  }

  if (!first.isValid() || !second.isValid()) {
    setValid(false);
    return;
  }

  if (second.isBefore(first)) {
    setValid(false);
    return;
  }

  if (first.isSame(second)) {
    setValid(false);
    return;
  }

  setValid(true);
};

return (
  <Stack
    spacing={1}
    direction={isMobile ? "column" : "row"}
    alignItems="center"
  >
    <LocalizationProvider dateAdapter={AdapterDayjs}>
      <DemoContainer components={["DatePicker"]}>
        <BaseDatePicker
          value={firstValue}
          onChange={(newValue) => {
            handleChange(newValue, secondValue);
            setFirstValue(newValue);
          }}
        />
      </DemoContainer>
    </LocalizationProvider>
    <LocalizationProvider dateAdapter={AdapterDayjs}>
      <DemoContainer components={["DatePicker"]}>
        <BaseDatePicker
          value={secondValue}
          onChange={(newValue) => {
            handleChange(firstValue, newValue);
```

```
              setSecondValue(newValue);
            }}
          />
        </DemoContainer>
      </LocalizationProvider>
      <Button
        disabled={!valid || !firstValue || !secondValue}
        variant="outlined"
        onClick={() => {
          onSubmit(firstValue, secondValue);
        }}
      >
        Показати
      </Button>
    </Stack>
  );
};
```

## 7. Компонет діаграми оцінок продукції (frontend)

```
import React, { FC, useEffect, useMemo, useState } from "react";
import {
  Bar,
  BarChart,
  CartesianGrid,
  Legend,
  Tooltip,
  XAxis,
  YAxis
} from "recharts";

import { getProducts } from "../../../api/product";
import { ProductResponse } from "../../../api/product/types";
import { generateColors } from "../../../utils/colors";

export const TextChart: FC<{}> = ({}) => {
  const [products, setProducts] = useState<ProductResponse[]>([]);

  const { chartData } = useMemo(() => {
    const sortedProducts = [...products]
      .sort(
        (a, b) =>
          a.reviews.reduce((sum, review) => sum + review.rating, 0) /
            a.reviews.length -
          b.reviews.reduce((sum, review) => sum + review.rating, 0) /
            b.reviews.length
      )
      .filter((p) => p.reviews.length > 0);
```

```
    const worstProducts = sortedProducts.slice(0, 10);
    const colors = generateColors(worstProducts.length);

    const chartData = worstProducts.map((product, i) => ({
      name: product.name,
      Оцінка:
        product.reviews.reduce((sum, review) => sum + review.rating, 0)
/
        product.reviews.length,
      fill: colors[i],
    }));

    return { chartData };
  }, [products]);

  useEffect(() => {
    const fetch = async () => {
      const res = await getProducts();
      setProducts(res.data);
    };
    fetch();
  }, []);

  return (
    <BarChart width={600} height={400} data={chartData}>
      <CartesianGrid strokeDasharray="3 3" />
      <XAxis dataKey="name" />
      <YAxis />
      <Tooltip />
      <Legend />
      <Bar dataKey="Оцінка" fill="#8884d8" />
    </BarChart>
  );
};
```

## 8. Модуль додатка (backend)

```
import { join } from "path";

import { Module } from "@nestjs/common";
import { ConfigModule } from "@nestjs/config";
import { MulterModule } from "@nestjs/platform-express";
```

```
import { ServeStaticModule } from "@nestjs/serve-static";
import { TypeOrmModule } from "@nestjs/typeorm";

import { AuthModule } from "./auth/auth.module";
import { Contract } from "./contract/contract.entity";
import { ContractModule } from "./contract/contract.module";
import { Image } from "./image/image.entity";
import { ImageModule } from "./image/image.module";
import { Order } from "./order/order.entity";
import { OrderModule } from "./order/order.module";
import { ProductOrder } from "./order/product-order.entity";
import { ProductType } from "./product-type/product-type.entity";
import { ProductTypeModule } from "./product-type/product-
type.module";
import { Product } from "./product/product.entity";
import { ProductModule } from "./product/product.module";
import { Review } from "./review/review.entity";
import { ReviewModule } from "./review/review.module";
import { Role } from "./roles/role.entity";
import { RolesModule } from "./roles/roles.module";
import { User } from "./users/user.entity";
import { UsersModule } from "./users/users.module";

@Module({
  imports: [
    ContractModule,
    OrderModule,
    ReviewModule,
    ImageModule,
    ProductModule,
    ProductTypeModule,
    UsersModule,
    RolesModule,
    AuthModule,
    ConfigModule.forRoot({
      envFilePath: `.${process.env.NODE_ENV}.env`,
    }),
    MulterModule.register({
      dest: "./static",
    }),
    ServeStaticModule.forRoot({
      rootPath: join(__dirname, "..", "static"),
      serveRoot: "/static",
    }),
    TypeOrmModule.forRoot({
      type: "postgres",
      host: process.env.POSTGRESS_HOST,
      port: Number(process.env.POSTGRESS_PORT),
```

```
        username: process.env.POSTGRESS_USER,
        password: process.env.POSTGRESS_PASSWORD,
        database: process.env.POSTGRESS_DB,
        synchronize: true,
        logging: true,
        ssl: {
          rejectUnauthorized: false,
        },
        entities: [
          Role,
          User,
          ProductType,
          Product,
          Image,
          Review,
          Order,
          ProductOrder,
          Contract,
        ],
        subscribers: [],
        migrations: [],
      }),
    ],
})
export class AppModule {}
```

## 9. Модуль авторизації (backend)

```
import { Role } from "src/roles/role.entity";
import { RolesService } from "src/roles/roles.service";
import { User } from "src/users/user.entity";
import { UsersService } from "src/users/users.service";

import { Module } from "@nestjs/common";
import { JwtModule, JwtSecretRequestType } from "@nestjs/jwt";
import { TypeOrmModule } from "@nestjs/typeorm";

import { AuthController } from "./auth.controller";
import { AuthService } from "./auth.service";

@Module({
  imports: [
    JwtModule.register({
      secret: process.env.SECRET_KEY || "secret",
      signOptions: {
        expiresIn: "7d",
      },
      secretOrKeyProvider: (requestType: JwtSecretRequestType) => {
```

```
          switch (requestType) {
            default:
              return process.env.SECRET_KEY || "secret";
          }
        },
      }),

      TypeOrmModule.forFeature([User, Role]),
    ],
    controllers: [AuthController],
    providers: [AuthService, UsersService, RolesService],
  })
  export class AuthModule {}
```

## 10. Сервіс продуктів (backend)

```
import { ImageService } from "src/image/image.service";
import { ProductTypeService } from "src/product-type/product-
type.service";
import { Product } from "src/product/product.entity";
import { ILike, Repository } from "typeorm";

import { Injectable } from "@nestjs/common";
import { InjectRepository } from "@nestjs/typeorm";

import { CreateProductDto } from "./dto/create-product.dto";
import { UpdateProductDto } from "./dto/update-product.dto";

@Injectable()
export class ProductService {
  constructor(
    @InjectRepository(Product)
    private productRepository: Repository<Product>,
    private productTypeService: ProductTypeService,
    private imageService: ImageService
  ) {}

  async create(dto: CreateProductDto) {
    const productType = await
this.productTypeService.getByName(dto.type);

    const image = await this.imageService.getById(dto.imageId);

    const product = await this.productRepository.save({
      name: dto.name,
      description: dto.description,
```

```
      price: dto.price,
      sale: dto.sale,
      weight: dto.weight,
      weightType: dto.weightType,
      type: productType,
      barcode: dto.barcode,
      boxSize: dto.boxSize,
      package: dto.package,
      quantityPerBox: dto.quantityPerBox,
      storageConditions: dto.storageConditions,
      availableQuantity: dto.availableQuantity,
      image,
    });

    return product;
  }

  async getAll() {
    const products = await this.productRepository.find({
      order: {
        name: "ASC",
      },
      relations: {
        type: true,
        image: true,
        reviews: true,
      },
    });

    return products;
  }

  async getById(id: number) {
    const products = await this.productRepository.findOne({
      where: { id },
      relations: {
        type: true,
        image: true,
        reviews: true,
      },
    });

    return products;
  }

  async findByName(name: string, productType: string) {
    const products = await this.productRepository.find({
      where: {
```

```
        name: ILike(`%${name}%`),
        type: {
          name: ILike(`%${productType}%`),
        },
      },
      order: {
        name: "ASC",
      },
      relations: {
        type: true,
        image: true,
        reviews: true,
      },
    });

    return products;
  }

  async update(id: number, dto: UpdateProductDto) {
    const product = await this.productRepository.findOne({
      where: { id },
      relations: {
        image: true,
        type: true,
        reviews: true,
      },
    });

    const productType = await
this.productTypeService.getByName(dto.type);

    const image = await this.imageService.getById(dto.imageId);

    await this.productRepository.save({
      id,
      name: dto.name,
      description: dto.description,
      price: dto.price,
      sale: dto.sale,
      weight: dto.weight,
      weightType: dto.weightType,
      type: productType,
      barcode: dto.barcode,
      boxSize: dto.boxSize,
      package: dto.package,
      quantityPerBox: dto.quantityPerBox,
      storageConditions: dto.storageConditions,
      availableQuantity: dto.availableQuantity,
```

```
      image,
    });

    await this.imageService.remove(product.image.src);

    const updatedProduct = await this.productRepository.findOne({
      where: { id },
      relations: {
        image: true,
        type: true,
        reviews: true,
      },
    });

    return updatedProduct;
  }
}
```

## 11. Сервіс замовлень (backend)

```
import { Product } from "src/product/product.entity";
import { Between, Repository } from "typeorm";

import { Injectable } from "@nestjs/common";
import { InjectRepository } from "@nestjs/typeorm";

import { CreateOrderDto } from "./dto/create-order.dto";
import { Order } from "./order.entity";
import { ProductOrder } from "./product-order.entity";

@Injectable()
export class OrderService {
  constructor(
    @InjectRepository(Order)
    private orderRepository: Repository<Order>,
    @InjectRepository(Product)
    private productRepository: Repository<Product>,
    @InjectRepository(ProductOrder)
    private productOrderRepository: Repository<ProductOrder>
  ) {}

  async getActive() {
    const res = await this.orderRepository.find({
      order: {
        created_at: "DESC",
      },
```

```
      relations: {
        productOrders: {
          product: {
            type: true,
          },
        },
      },
    });
    return res;
  }

  async create(dto: CreateOrderDto) {
    const order = await this.orderRepository.save({
      accepted: false,
      address: dto.address,
      lastName: dto.lastName,
      name: dto.name,
      phone: dto.phone,
      totalPrice: dto.totalPrice,
    });

    const productOrdersPromises = dto.orders.map(async (productOrder)
=> {
      const product = await this.productRepository.findOne({
        where: {
          id: productOrder.productId,
        },
      });

      await this.productOrderRepository.save({
        order,
        price: productOrder.price,
        count: productOrder.count,
        product,
      });

      await this.productRepository.save({
        id: product.id,
        availableQuantity: product.availableQuantity -
productOrder.count,
      });
    });

    await Promise.all(productOrdersPromises);

    const res = await this.orderRepository.findOne({
      where: { id: order.id },
      relations: {
```

```
      productOrders: {
        product: {
          type: true,
        },
      },
    },
  });

  return res;
}

async updateAccepted(value: boolean, orderId: number) {
  await this.orderRepository.save({
    id: orderId,
    accepted: value,
    rejected: !value,
  });

  const res = await this.orderRepository.findOne({
    where: { id: orderId },
    relations: {
      productOrders: {
        product: {
          type: true,
        },
      },
    },
  });

  return res;
}

async updateRejected(value: boolean, orderId: number) {
  await this.orderRepository.save({
    id: orderId,
    rejected: value,
    accepted: !value,
  });

  const res = await this.orderRepository.findOne({
    where: { id: orderId },
    relations: {
      productOrders: {
        product: {
          type: true,
        },
      },
    },
```

```
    });

    return res;
  }

  async getOrdersBetween(f: Date, s: Date) {
    // const currentDate = new Date();
    // const startOfMonthDate = startOfMonth(currentDate);
    // const endOfMonthDate = endOfMonth(currentDate);
    const res = await this.orderRepository.find({
      where: {
        created_at: Between(f, s),
      },
      relations: {
        productOrders: {
          product: {
            type: true,
            reviews: true,
          },
        },
      },
    });

    return res;
  }
}
```