Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та
комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

другий (магістерський)
(рівень вищої освіти)

на тему: "Розробка інформаційної систем обліку споживчих товарів у магазинах"

Виконав: студент VI курсу групи КН-61М
Спеціальності 122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Піманчиков В.В.
(прізвище та ініціали)

Керівник Шабатура Ю.В.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Львів – 2021

РЕФЕРАТ

Дипломна робота містить 65 сторінок пояснювальної записки, 51 рисунків, 1 додаток, 7 джерел.

В даній роботі розроблено програмне забезпечення по типу Warehouse Management System (система управління складами) та інші супутні рішення зв’язані із специфікою бізнес-процесів підприємств по торгівлі у малому об’ємі.

Ключові слова:

SOA, **ReactJS**, **NodeJS**, **REST**, **Java**, **Spring Framework**, **ORM**, **JPA**, **HTTP**, **WMS**, **CRM**, **SPA**, системи складського обліку та аудиту

ABSTRACT

The Master thesis contains 65 pages of explanatory note, 51 pictures, 1 application, 7 sources.

In this paper, we developed software such as Warehouse Management System and other related solutions related to the specifics of business processes of small businesses.

Keywords:

SOA, ReactJS, NodeJS, REST, Java, Spring Framework, ORM, JPA, HTTP, WMS, CRM, SPA, warehousing and auditing systems

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити та спроектувати програмне забезпечення для системи складського обліку, а саме:

1. Складення вимог до майбутньої системи на основі аналізу існуючої ситуації;
2. Проектування серверного **ПЗ** для реалізації бізнес-процесів згідно із вимогами;
3. Реалізація серверного **ПЗ** та бази даних;
4. Реалізація користувацького інтерфейсу для можливості взаємодії із системою;
5. Розробка необхідних конфігурацій для автоматизованого встановлення та запуску систем.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ 7

ВСТУП 8

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ 10

1.1 Внутрішній устрій роздрібної торгівлі 10

1.1.1 Визначення та основні функції

1.1.2 Особливості бізнес-процесів

1.2 Складання вимог згідно інтерпретованих процесів 1

Висновки до розділу 1

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ 15

2.1 Перелік технічних вимог до систем 15

2.2 Проектування сервісної системи 15

2.2.1 Проектування відношень в реляційній базі даних

2.2.2 Проектування об’єктів доменної області

2.2.3 Діаграми сценаріїв використання

2.2.4 Діаграми інтерфейсів

2.3 Проектування клієнтської системи 15

Висновки до розділу 2

РОЗДІЛ 3. ОПРИДІЛЕННЯ РЕАЛІЗУЮЧИХ ТЕХНОЛОГІЙ 25

3.1 Сервісні технології 15

3.2 Клієнтські технології 15

Висновки до розділу 22

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ 25

4.1 Архітектура систем 25

4.2 Реалізація сервісів 27

4.3 Реалізація клієнтської частини 28

Висновки до розділу 37

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ 50

5.1 Опис ідеї проекту 5

5.2 Розроблення ринкової стратегії 5

5.3 Розроблення маркетингової програми 5

5.4 Вимоги до технічного та програмного забезпечення 5

Висновки до розділу 5

ВИСНОВКИ 54

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 55

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ПЗ — програмне забезпечення;

WMS — система управління складами;

CRM — система управління взаємовідносинами із клієнтами;

ORM — об’єктно-реляційне відображення;

SOA — сервіс-орієнтована архітектура;

Доменна область — це об'єкти в об'єктно-орієнтованих комп'ютерних програмах, що виражають сутності з моделі предметної області, що відноситься до програми, та реалізують бізнес-логіку програми.

ВСТУП

Автоматизація складу (впровадження **WMS**) забезпечує ефективне та раціональне управління складськими процесами, що, в свою чергу, є необхідною умовою стабільного контролю та розвитку бізнесу. Завдяки автоматизації роботи складу зростає швидкість обробки замовлень, у кілька разів прискорюється проведення складських операцій, збільшується пропускна спроможність підприємства, значно знижуються втрати товару. За рахунок скорочення обсягу рутинної роботи підвищується ефективність роботи складського персоналу, а час навчання після впровадження системи автоматизації складу скорочується до мінімуму.

Даний проект є втіленням однієї із таких систем та забезпечує автоматизацію управління бізнес-процесами складської роботи, складення необхідних договорів, ведення обліку постачальників на профільних підприємствах.

У даній дипломній роботі взято за мету розробка комплексного рішення по керуванню товарообігом в підприємствах невеликого обсягу. На ринку, звісно, присутня достатня кількість рішень для проблем сучасного бізнесу. Цей проект не переслідує ціль скласти їм стійку конкуренцію або ж освітлити інноваційні методики, але дозволяє краще зрозуміти фундаментальні потреби бізнесу та реалізує їх у комплексному рішенні.

Розділ 1

1.1 Огляд та аналіз проблемної області

1.1.1 Визначення та основні функції

Роздрібна торгівля включає підприємницьку діяльність, пов'язану з продажом товарів і послуг кінцевим споживачам для особистого, сімейного і домашнього користування.

Роздрібна торгівля є кінцевою стадією каналів товаро-просування. Вона постійно намагається розширити товарний асортимент, збільшувати обсяги продажу. Цей вид торгівлі є надзвичайно важливою ланкою маркетингової діяльності на ринку, тому що вона допомагає підприємству зрозуміти значимість товарів, їх споживчі властивості, рівень задоволення потреби та власний престиж товарної марки і фірми.

Роздрібна торгівля виконує такі основні функції:

- Здійснює набір товарів і послуг та розміщення у торговельних приміщеннях;
- Здійснює сортування товару за призначенням та способом застосування;
- Здійснює інформування споживачів через рекламу, вітрини та особисте спілкування з покупцями;
- Задовольняє інтереси каналів збуту товарів;
- Зберігає товари;
- Встановлює ціни;
- Здійснює розрахунок з постачальниками;

1.1.2 Особливості бізнес-процесів

Після огляду базових функцій можна скласти наступні узагальнені бізнес-процеси, які відбуваються на підприємстві:

a) Знаходження по певним критеріям вигідного постачальника товарів;

b) Узгодження та складання із постачальником контракту в якому описано перелік товарів, які підприємство бажає отримувати;

c) Отримання замовлення згідно складеного із постачальником контракту;

d) Зберігання цих товарів до моменту реалізації відповідно до затверджених стандартів;

e) Аналіз продажів та збережених статистичних даних щодо збутих товарів.

Кожен із процесів, в свою чергу, має специфіку притаманну саме йому. Так, наприклад, контракт із пункту **b** має мати терміни укладення, список товарів які пропонує постачальник згідно цього контракту та дані засновника контракту.

Також варто пам’ятати про те, що працівники підприємства володіють своєю зоною відповідальності. Працівник, що відповідає за властиві йому обов’язки не повинен мати змогу вчинити те, що за межами цієї зони. Таким чином кожен працівник буде пов’язаний із своїми бізнес-процесами, що, в свою чергу, дасть додаткову ясність та чіткість.

1.2 Складання вимог згідно інтерпретованих процесів

Почнемо із первинних доменних сутностей — користувачів. Користувач повинен мати наступні дані:

- Ім’я та прізвище;
- Електронну скриньку;
- Пароль;

- Номер телефону;
- Посаду;

По типу посади користувачу буде надаватись або приховуватись доступ до тих, чи інших функціональних можливостей.

Список посад та їх особливості:

- **Менеджер**. Це головна посада, якій надаються можливості по залученню постачальників, утворенню контрактів, створення замовлень та аналізу статистики.
- **Робітник**. Цій посаді надаються данні про товари, їх дані та очікуване поповнення товарів згідно замовлення;
- **Отримувач**. Цій посаді надаються повноваження по прийому або відхиленню придбаного замовлення від постачальника. Також надається інформація щодо замовлень, які повинні прибути найближчим часом.

На черзі вимоги щодо постачальника товарів. Він повинен мати наступні дані в системі :

- Країну, місто, вулицю;
- Контактний телефон, ім’я, електронну скриньку;
- Ім’я компанії.

Також повинна бути можливість укладення контракту із постачальником. Видалення постачальника із системи можливе тільки у випадку відсутності діючих контрактів, які були укладені із ним або відсутності замовлень, які були підтверджені постачальником та знаходяться в дорозі.

Контракти повинні володіти наступними даними:

- Ідентифікатор постачальника, з яким вкладений контракт;
- Ідентифікатор користувача, який створив даний контракт;
- Дата укладення та кінця контракту.

Працівники повинні мати можливість створити замовлення із переліком товарів по конкретному контракту. Контракт не може бути видалений, якщо він діючий або до підприємства прямує підтверджене замовлення.

Сутність продукту в системі повинна володіти наступними даними:

- Кодом;
- Назвою;
- Кількістю одиниць в наявності;
- Ціною;

Ідентифікатором контракту по якому можна додати продукт до замовлення постачальнику;
Критичним залишком.

Продукт повинен бути пов’язаний із конкретним контрактом, щоб по ньому, в свою чергу, можна було створити замовлення до постачальника. Продукт не можна видалити із системи, якщо до підприємства прямує підтверджене замовлення, яке містить цей продукт. Критичний залишок потрібен для швидкого визначення товарів, які потрібно якнайшвидше поповнити, та встановлюється користувачем системи.

Замовлення до постачальника повинно володіти такими даними:

- Ідентифікацією користувача, який створив замовлення;
- Ідентифікацією контракту, по якому створено замовлення;

- Дата орієнтовного прибуття та дата створення замовлення;
- Статусом замовлення;
- Списком товарів та їх кількістю, що були замовлені;
- PDF документом замовлення.

Замовлення повинно надсилатись постачальнику на розгляд. Також у сучасному світі технологій передбачається автоматичне підтвердження замовлень, якщо у постачальника є реалізовані такі системи. Така система може проаналізувати наявність товарів постачальника і надати відповідь негайно. У випадку відсутності такої системи, працівник розгляне PDF документ та прийме або відхилить надіслане замовлення на підставі прийнятих ним рішень.

Також в системі присутні дані про покупки. Вони повинні володіти наступною інформацією:

- Датою здійснення покупки;
- Загальна сума покупки;
- Списком товарі, які були придбані.
- Ці дані потрібні для статистики та відслідковування рівнів прибутку.

Висновки розділу

Було розглянуто основні життєві цикли підприємств, які зв’язані із сферою роздрібної торгівлі. Це базові, концептуальні процеси, які відбуваються у кожному підприємстві, але із своїми внутрішніми особливостями та правилами. Такі індивідуальні особливості створюють ряд проблем при розробці автоматизованих систем керування, особливо універсальних. Технології розвиваються досить швидко і тому розробка таких систем вимагає чіткого бачення і формулювання бізнес-аналітики.

Також було визначено ряд правил, які дозволяють скласти чітку картину фундаментальних потреб любого підприємства із сфери торгівлі і припущені сценарії їх реалізації. За призначенням майбутня система не схожа на **WMS**, або **CRM**. Натомість вона реалізує кожну із цих систем частково.

РОЗДІЛ 2

2.1 Перелік технічних вимог до систем

Важливим показником в корпоративних системах є гнучкість і масштабування, отож класична архітектура моноліту в даному випадку не підходить по ряду причин:

1. Тяжко підтримувати у випадку розростання функціональних можливостей. Всі модулі існують в межах одного програмного процесу і не можуть існувати за його межами;
2. Складності в тестуванні;
3. Відсутність ситуативного розширення якоїсь певної функціональної одиниці моноліту. Додатковий сервер буде містити всі функціональні одиниці, навіть якщо в розширенні потребують тільки деякі.

Вирішити ці і додаткові ряди проблем призвана мікро-сервісна архітектура. Ця архітектура полягає в розбиття моноліту системи на окремі компоненти, які є незалежними сервісами. Сервіси мають чіткі фізичні межі, що робить їх масштабованими, дає можливість використовувати для написання різні мови програмування. Необхідні зміни вносяться, не чіпаючи весь додаток повністю.

В даному проекті реалізується компромісне рішення. Використовується архітектурний підхід **SOA (Service Oriented Architecture)**, який реалізується як моноліт, але всі програмні сутності поділяються на сервіси. Таким чином при потребі перейти на мікро-сервісну архітектуру, достатньо лише розділити всі сервіси по особистим процесам та надати доступ до власних джерел даних замість одного спільного.

Вимоги до клієнтської сторони мінімальні по причині особливості переліку зобов’язань. Клієнтська сторона повинна реалізувати зручний та простий, але водночас, інформативний інтерфейс. Вона виконує роль мосту між даними сервісів, їх відображенням та переліком функціональних можливостей, а значить не володіє ніякою додатковою бізнес-логікою. Вона також повинна виступати окремим сервісом використання даних.

2.2 Проектування сервісної системи

Система реалізує кожен бізнес-процес як окремий сервіс із набором інструкцій та правил, описаних у **Розділі 1.1**. Таким чином досягається ізоляція та слабка зв’язність між компонентами системи, що дозволяє розширити спектр їхнього застосування, адаптивність, а також їх повторне використання.

2.2.1 Проектування відношень в реляційній базі даних

На основі вимог із **розділу 1.2** було складено наступну схему відношень доменних сутностей:

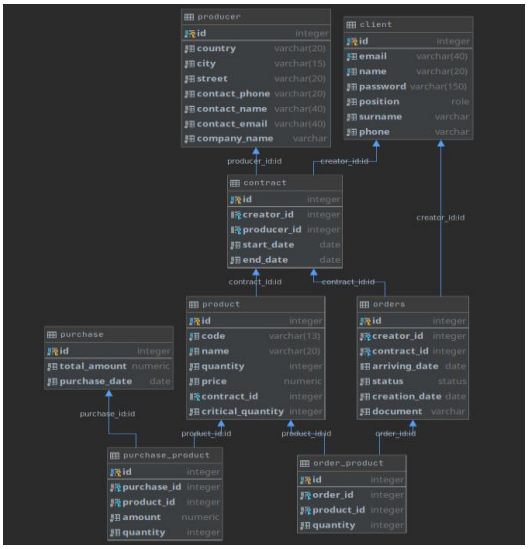


Рис. 2.1. SQL relationships.

2.2.2 Проектування об’єктів доменної області

На основі вимог із **розділу 1.2** було складено наступну UML діаграму доменних об’єктів:

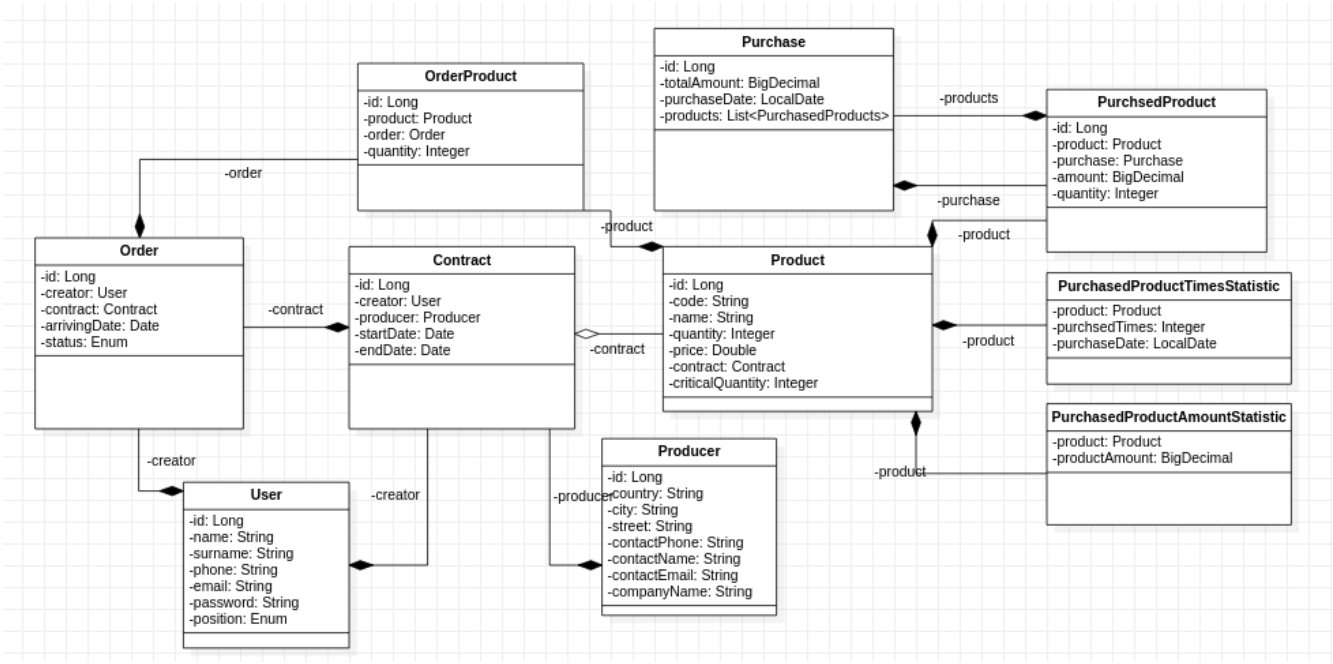


Рис.2. 2 Objects UML

2.2.3 Діаграми сценаріїв використання

На даному зображенні показується сценарій створення замовлення з відмітками на ключових етапах процесу.

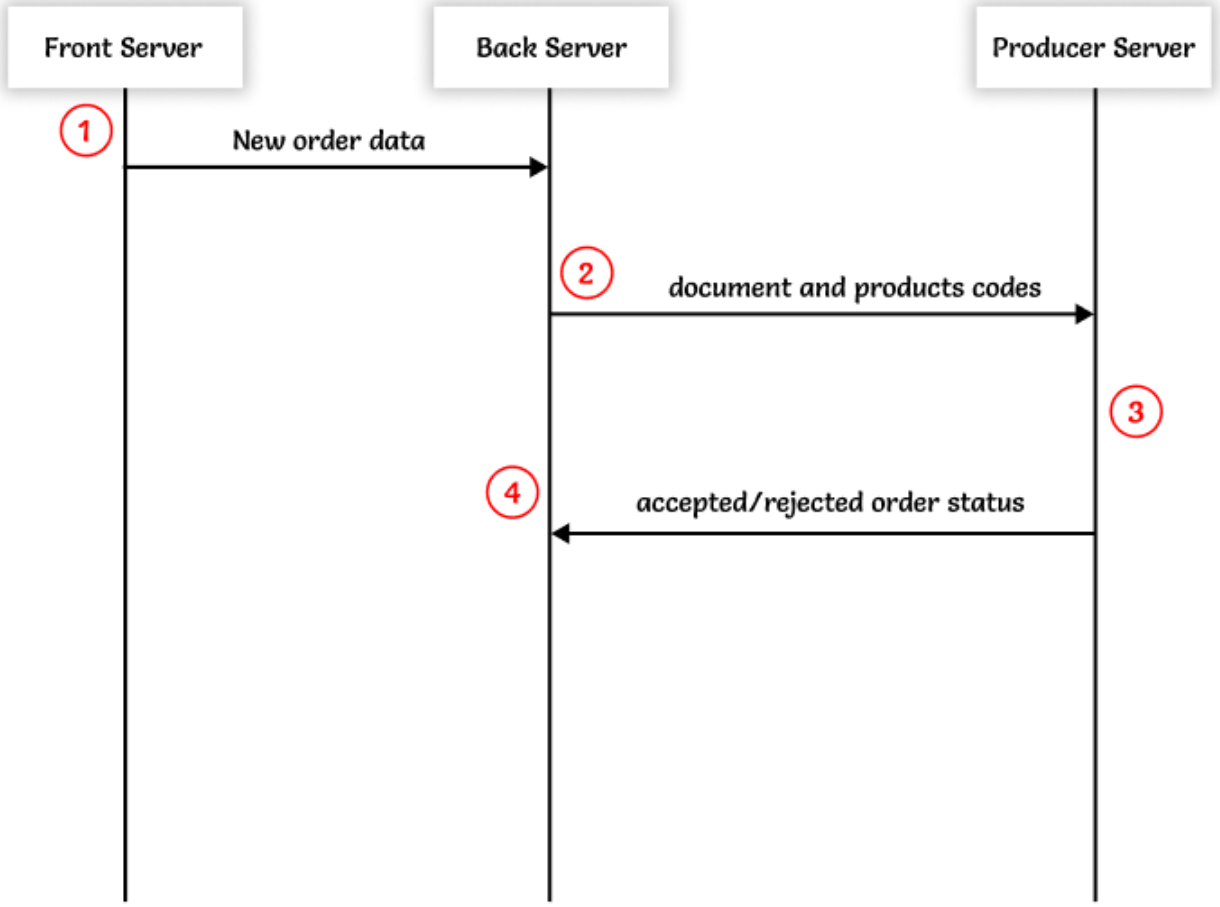


Рис. 2.3 Order UML use-case

1. Користувач створив замовлення по контракту із постачальником;
2. Сервер обробив замовлення додавши його в базу даних із статусом “очікує підтвердження”. На цьому етапі генерується документ із списком товарів та даними засновника замовлення. Цей документ із списком потрібних товарів надсилається на сервер постачальника з метою підтвердження;
3. Сервер постачальника отримав дані і робить спробу автоматичного підтвердження. На цьому етапі сервер може автоматично надіслати у відповідь позитивний або негативний статус. Якщо цей сервер із якихось причин не зміг однозначно визначити статус, він залишить це замовлення на розгляд працівників. В такому випадку відповідь прийди пізніше, а не миттєво.
4. Сервер отримує статус і модифікує існуюче замовлення цим статусом.

2.2.4 Діаграми інтерфейсів

Діаграма інтерфейсів по роботі із постачальників:

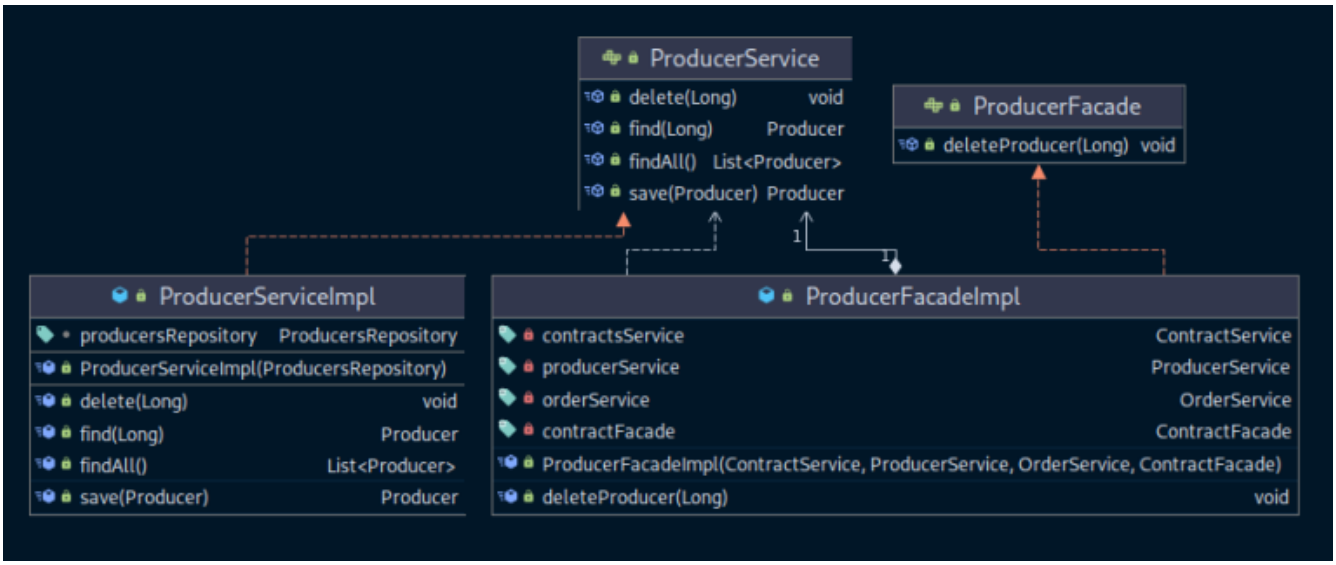


Рис. 2.4. (Producers services)

Діаграма інтерфейсів по роботі із контрактами:

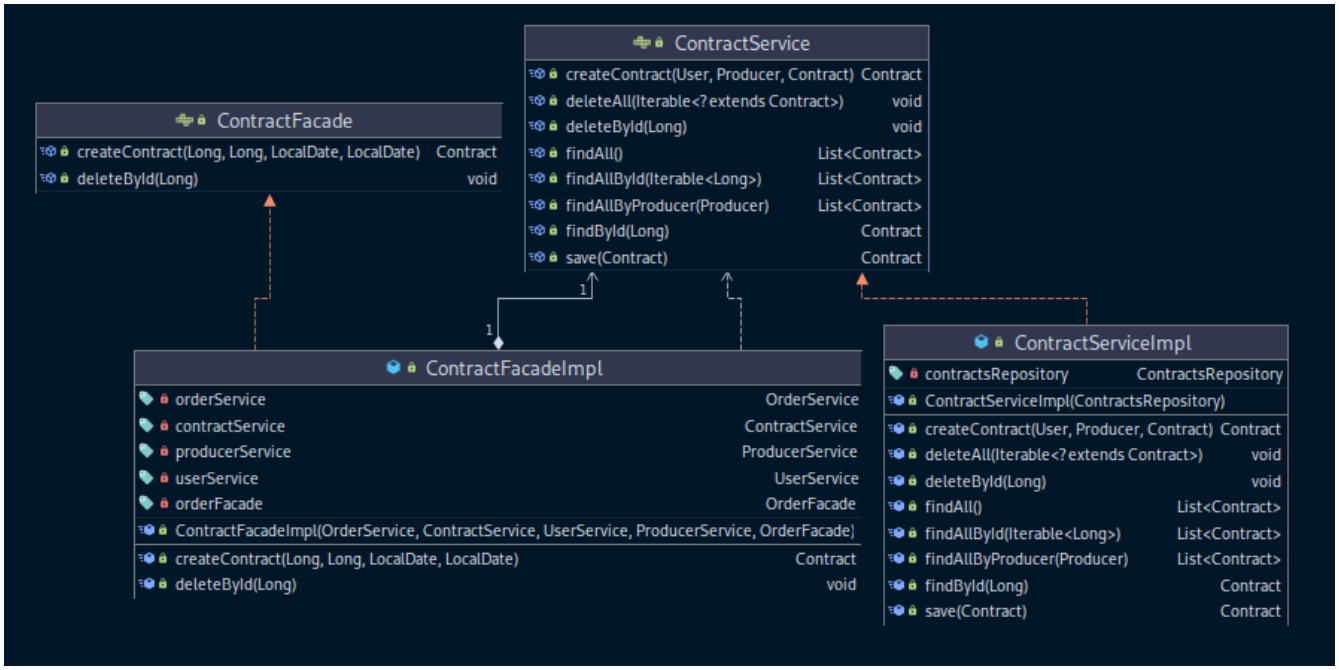


Рис. 2. 5. (Contracts services)

Діаграма інтерфейсів по роботі із продуктами:

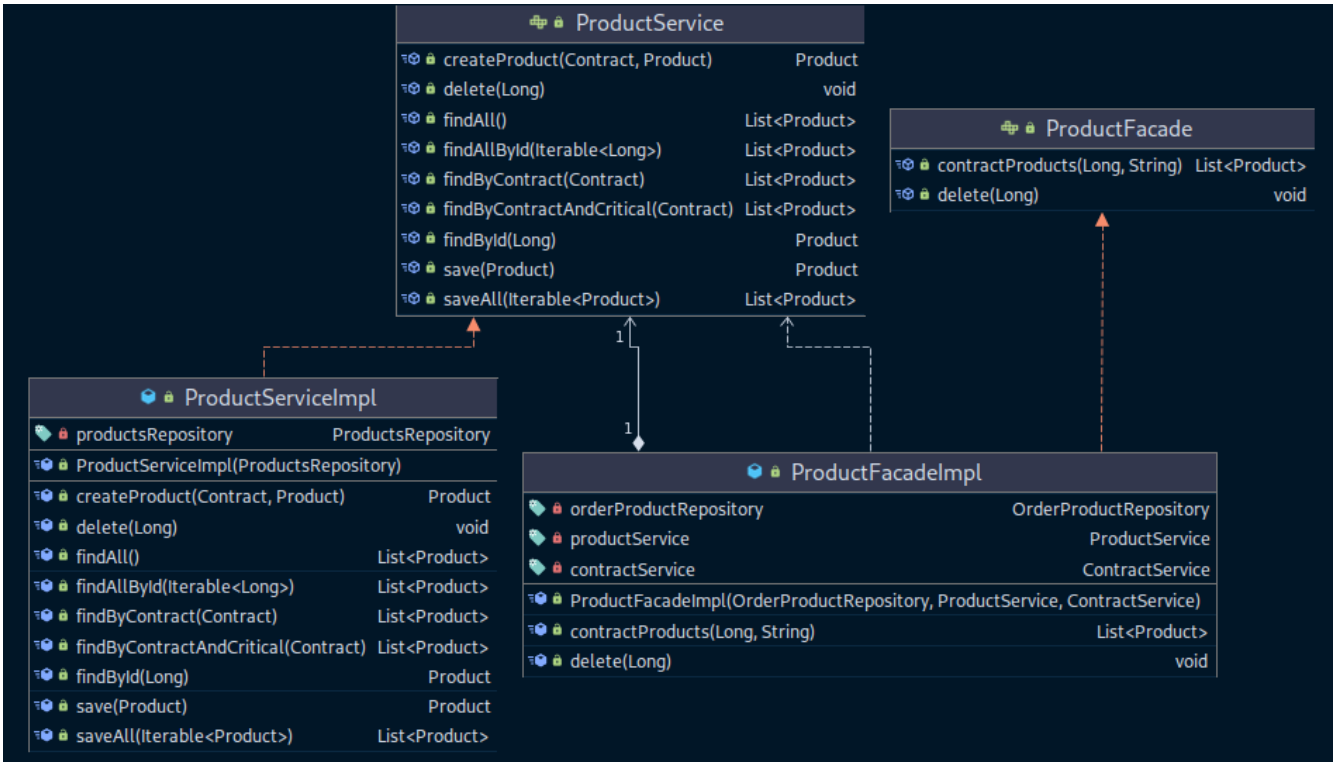
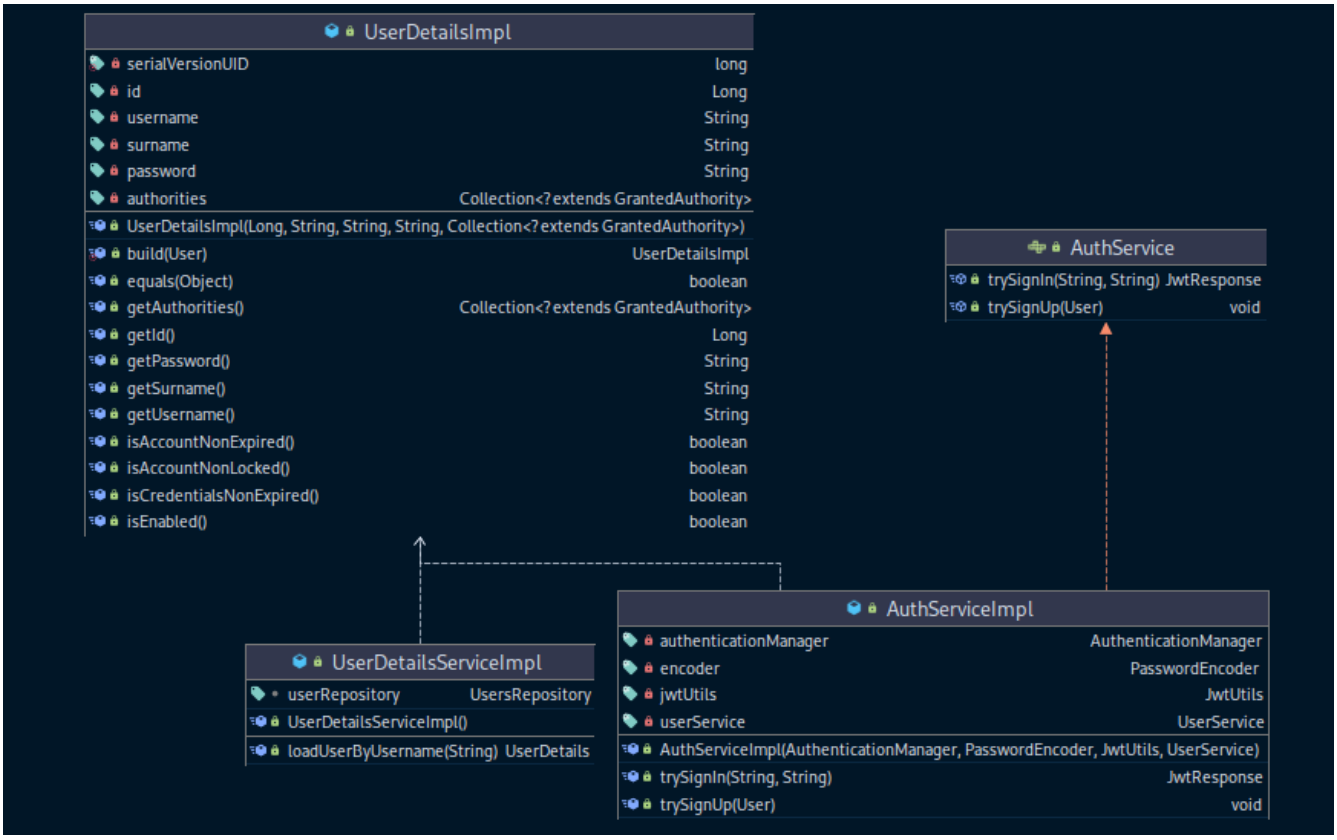
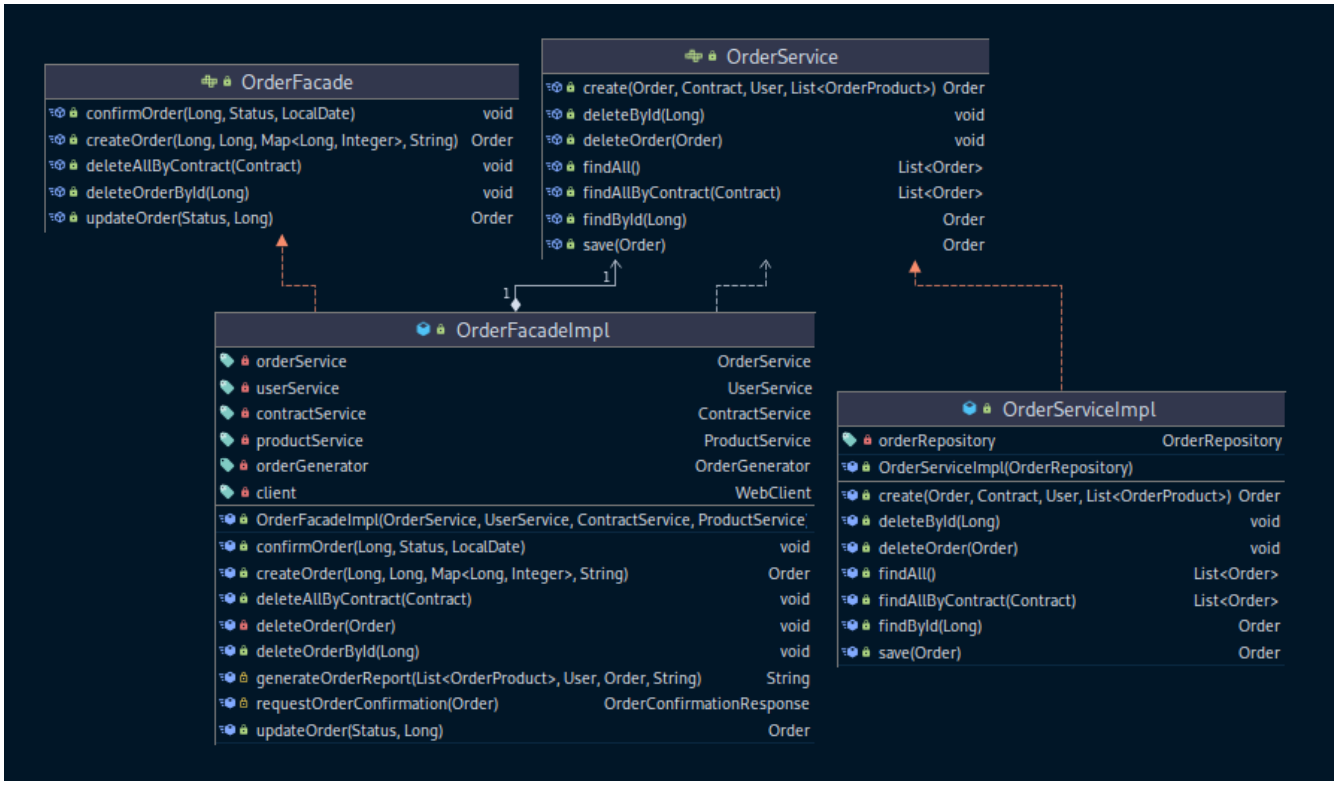


Рис. 2.6 (Products services)



Діаграма інтерфейсів по роботі із автентифікацією:

Рис. 2.7 (Authentification services)



Діаграма інтерфейсів по роботі із замовленнями:

Рис. 2.8 (Orders services)

Діаграма інтерфейсів по роботі із покупками:

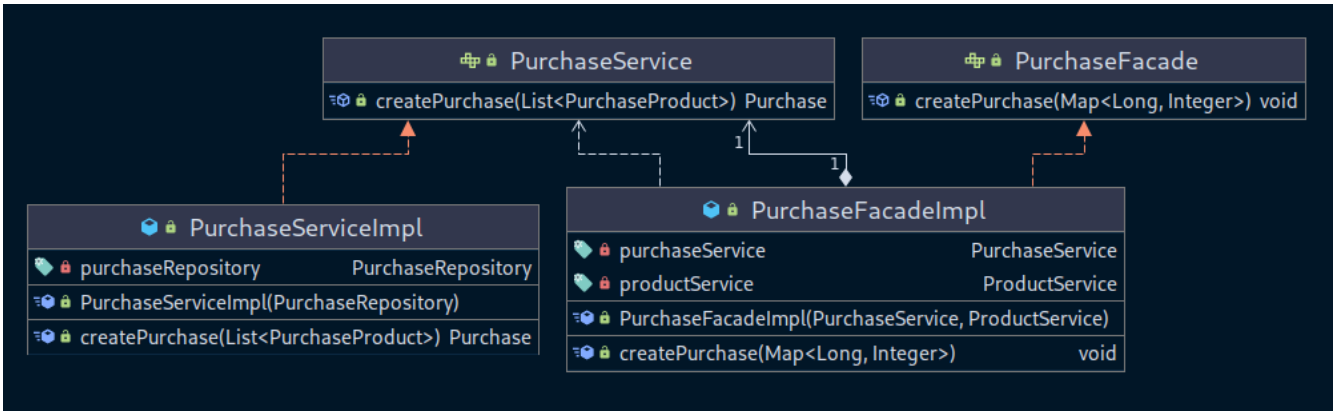


Рис. 2.9 (Purchase services)

Діаграма інтерфейсів по роботі із статистикою:

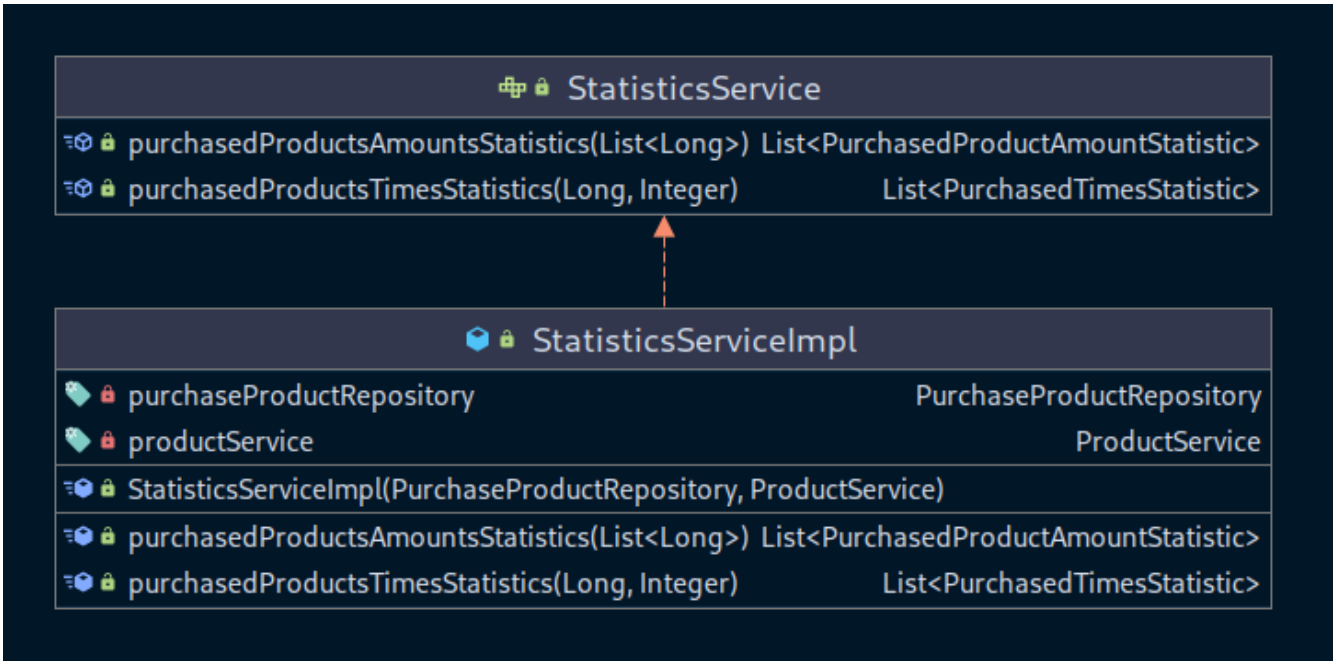


Рис. 2.10 (Statistics services)

Діаграма інтерфейсів по роботі із користувачами системи:

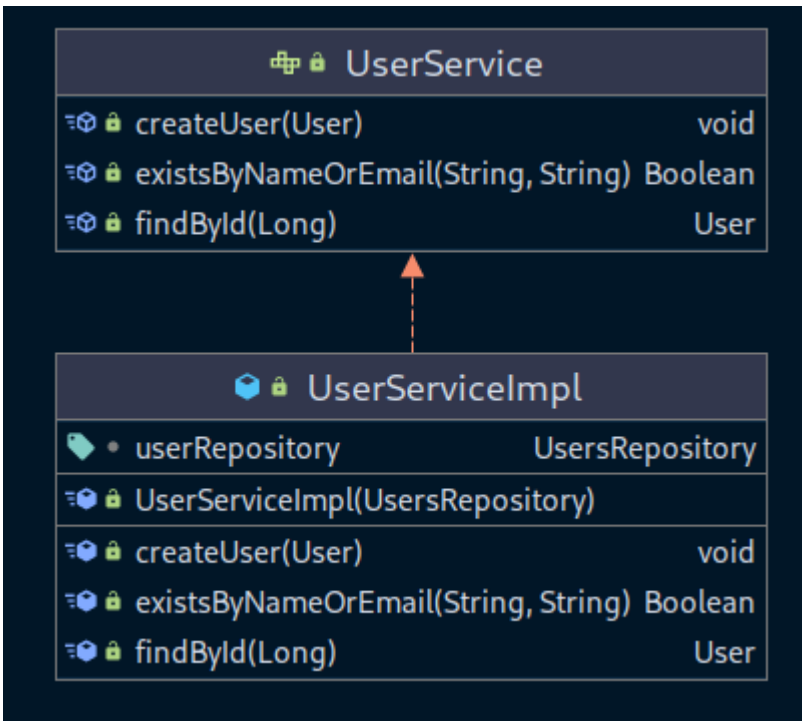


Рис. 2.11 (Users services)

2.4 Проектування клієнтської системи

Клієнтська частина використовує дані як результат роботи сервісів і реалізує лише звертання до API. Відповідно головною відповідальністю є вид та відображення цих даних. Передбачено наступні функціональні можливості:

1. Реєстрація нових працівників підприємства (користувачів);
2. Автентифікація користувачів;
3. Надання інформативного відображення даних;
4. Реалізація можливостей взаємодії з системою відповідно до потреби та посади користувача.

Висновки до розділу

Завдяки бізнес-правилам та їх вимогам, ми змогли вивести та сформувати базову структуру усіх потрібних доменних сутностей, що стосуються бізнес процесів. Також було встановлено зв’язки між цими об’єктними структурами (композиції та агрегації) та їх життєвий цикл в межах процесів системи.

Також було розглянуто найоб’ємніші сценарії використання системи та проаналізовано їх потік виконання для подальшого врахування їх вагомості у проектування.

Було визначено чіткі обов’язки та межі відповідальності сервісів системи. По значенню це основні програмні одиниці, які йдуть після об’єктних сутностей. Вони виконують роль оркестраторів, втілюючи основні напрямки взаємодії об’єктів та інших сервісів, та поряд виконання всіх бізнес-процесів.

РОЗДІЛ 3

3.1 Сервісні технології

Кодова база сервісів реалізована на мові Java, як одною із передових у сфері web-програмування. Однією з основних переваг мови Java є можливість перенесення програм із однієї системи до іншої . Оскільки програми Java не залежать від платформи як на рівні вихідного коду (завдяки **JVM**), так і на двійковому рівні, їх можна запускати в різних системах, що особливо важливо для програм, призначених для World Wide Web. Також існує безліч бібліотек для вирішення загальних проблем в цьому напрямку розробки.

Для швидкої і ефективної реалізації всіх функціональних можливостей та задач, поставлених бізнес-правилами, були використані додаткові бібліотеки та фреймворки, які надають готову програмну реалізацію.

Джерелом даних для зберігання було обрано базу даних **PostgreSQL**. Вона відзначається високою ефективністю, підтримкою БД необмеженого розміру та потужні і надійні механізми транзакцій та реплікації.

Для відображення та маніпуляції записами із бази даних як об’єктними сутностями використовується бібліотека **Hibernate.** Вона спрощує обробку та маніпуляції існуючих даних у вигляді об’єктів і усуває проблематику переходу ООП парадигми у SQL.

Із фреймворку **Spring** були взяті наступні модулі:

1. **Spring MVC;**

   Фреймворк Spring MVC забезпечує архітектуру патерна Model - View - Controller (Модель - Відображення - Контролер) за допомогою слабко

пов'язаних готових компонентів. Паттерн MVC поділяє аспекти програми (логіку введення, бізнес-логіку та логіку UI), забезпечуючи вільний зв'язок між ними. Він дозволяє з допомогою готових програмних конструкцій використовувати та обробляти вхідні та вихідні HTTP запити.

2. **Spring Data JPA;**

   Це технологія, що забезпечує об'єктно-реляційне відображення простих Java об'єктів та надає API для збереження, отримання та керування такими об'єктами. Мета **Spring Data** - значно скоротити обсяг шаблонного коду, необхідного для реалізації рівнів доступу до даних різних сховищ.

3. **Spring Security;**

   Java-фреймворк, що надає механізми побудови систем автентифікації та авторизації, а також інші можливості безпеки для промислових додатків, створених за допомогою **Spring Framework**.

4. **Spring Boot.**

   Spring Boot дозволяє вам легко створювати повноцінні, виробничого класу Spring-додатки. Він автоматично завантажує всі програмні бібліотеки і залежності, які потрібні для успішного функціонування Spring додатків.

3.2 Клієнтські технології

ReactJS це бібліотека для створення користувацьких інтерфейсів. Однією з її відмінних особливостей є можливість використовувати JSX, мову програмування з близьким до синтаксису HTML, який компілюється в JavaScript. Розробники можуть отримати високу продуктивність програми за допомогою Virtual DOM. Також в цій бібліотеці використовується **SPA(Single Page Application)** підхід. Він особливий тим, що документ завантажується лише одноразово, а отримання нових даних відбувається з допомого асинхронних HTTP запитів.

NodeJS це програмна платформа, яка робить JavaScript мовою загального призначення. Її також називають середовищем виконання JS. Вона вміє зв'язуватися із зовнішніми бібліотеками, викликати команди з коду та виконувати роль веб-сервера.

Висновок до розділу

Ринок програмних продуктів, для ведення розробки, на сьогоднішній день переповнений великою кількість пропозицій. Для любого типу задач знайдеться потрібний інструмент, а нетривіальний фреймворк спроможний виконати любі однотипні задачі за лічений час і з мінімальною витратою ресурсів. Вони спрощують життя величезній кількості розробників.

В даному розділі був здійснений короткий огляд одних із самих передових бібліотек та фреймворків як для фронтенду, так і бекенду. Вони надають зручний програмний інтерфейс для швидкої та ефективної реалізації потреб системи, та навіть реалізують готові комплексні рішення для ряду задач.

РОЗДІЛ 4

4.1 Архітектура систем

Як вже було зазначено, головною архітектурою для системи сервісів було обрано **SOA**. Ця архітектура досить легко інтегрується із серверним середовищем та не суперечить архітектурним стилям мережевого обміну інформацією (таким як REST).

Для запуску сервісів використовується Embedded Tomcat сервер. Tomcat це сервер застосунків, що дозволяє запускати веб-програми та містить ряд програм для авто-конфігурування. Tomcat використовується як самостійний веб-сервер і сервер контенту в поєднанні з веб-сервером Apache HTTP Server. Він дозволяє обробляти HTTP запити та повертати їх результат.

Платформою для запуску клієнтського додатку служить NodeJS із широким спектром функціональних можливостей. Він дозволяє компіляцію та виконання JS коду та реалізує HTTP сервер для звертання до веб-серверу.

Сервіси обмінюються даними за допомого RESTful API: стандартизованих правил та набором обмежень, що враховуються при проектуванні розподіленої гіпермедіа-системи. Цей стиль має ряд властивих йому характеристик:

1. Кожна сутність має мати унікальний ідентифікатор – URI;
2. Сутності мають бути пов'язані між собою;
3. Для читання та зміни даних слід використовувати стандартні методи;
4. Має бути підтримка кількох типів ресурсів;
5. Взаємодія має здійснюватися без стану.

Ось стандартний набір HTTP методі, який веб-сервіси використовують для виклику віддалених процедур:

GET – отримання даних без зміни;

POST – метод, який передбачає вставку нових записів;

PUT – метод, який передбачає зміну наявних записів;

PATCH – метод, який передбачає часткову зміну існуючих записів;

DELETE – метод, який передбачає видалення записів.

Також використовується шаблон проектування **MVC (Model-View-Controller)**. Цей шаблон, який реалізує схему поділу даних програми, і керуючої логіки на три окремі компоненти: модель, уявлення та контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно. Основна мета застосування цієї концепції полягає у відділенні бізнес-логіки (моделі) від її візуалізації (уявлення, виду). За рахунок такого розподілу підвищується можливість повторного використання коду.

4.2.1 Реалізація сервісів

Для отримання даних реалізовано **Controller**, який отримує HTTP запити в JSON форматі та делегує їх об’єктні представлення відповідним сервісам. Сервіси ж, в свою чергу, аналізують та проводять маніпуляції над сутностями предметної області з допомогою класів Spring Repository, які надають доступ до бази даних та об’єктів. URL сервісів мають вигляд по шаблону **METHOD /api/[name]**, де **name** — ім’я предметної області, а **METHOD —** HTTP метод. Сервіси мають схожу структуру і поведінку. Кожен із них реалізує основні методи: **GET, POST, PATCH, DELETE.** Також у логіці кожного сервісу зберігається бізнес-логіка та обробка помилок або виняткових ситуацій, описані у **Розділі 1.2**.

Сервіс по автентифікації та реєстрації користувачів надає згенерований ключ який свідчить про автентифікацію працівника. API сервісу має наступну структуру:

```
@PostMapping("/signin")
@CrossOrigin
public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {...}

@PostMapping("/signup")
@CrossOrigin
public ResponseEntity<?> registerUser(@Valid @NotNull @RequestBody SignupRequest signUpRequest) throws Exception {...}
```

Рис. 12 (Auth API)

URL /signin оброблює автентифікацію користувачів:

```
POST http://localhost:8080/api/auth/signin

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Transfer-Encoding: chunked
Date: Fri, 26 Nov 2021 20:15:37 GMT
Keep-Alive: timeout=60
Connection: keep-alive

{
  "id": 15,
  "username": "TestName",
  "roles": [
    "ROLE_MANAGER"
  ],
  "accessToken": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJUZXN0TmFtZSIs
  "tokenType": "Bearer"
}
```

Рис. 13 (Auth API)

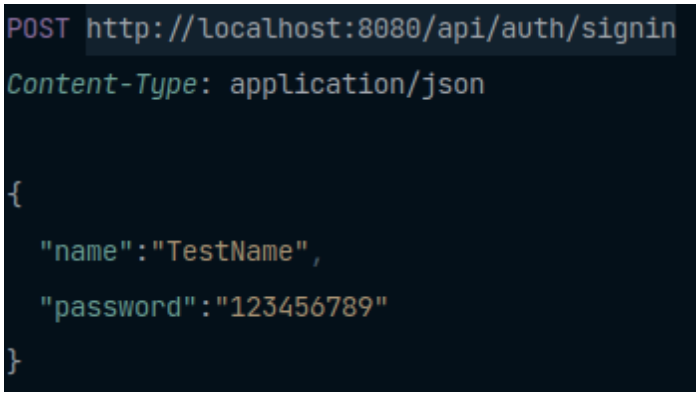


Рис. 14 (Auth API)

URL /signup оброблює реєстрацію нових користувачів:

Рис. 15 (Auth API)

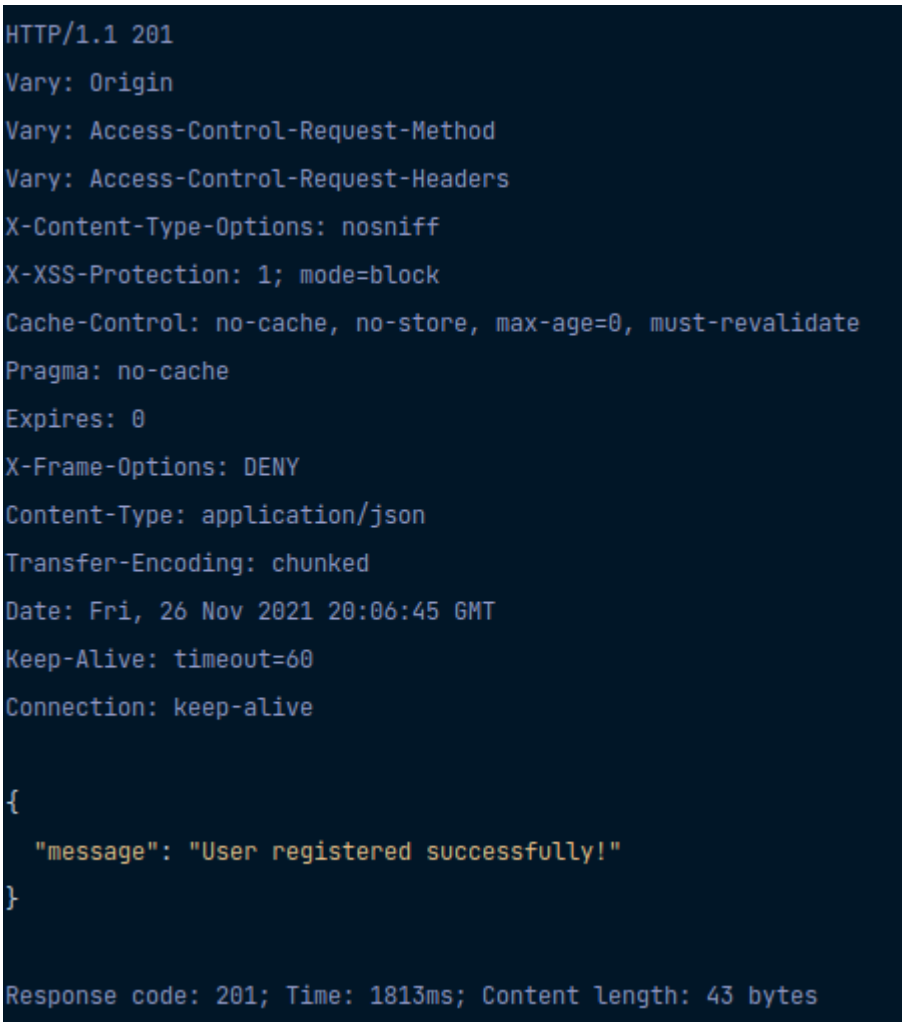


Рис. 16 (Auth API)

```
@PreAuthorize("hasRole('ROLE_MANAGER')")
@GetMapping(path = {"/producer"}, produces = {"application/json"})
ResponseEntity<String> producersAll() {...}

@PreAuthorize("hasRole('ROLE_MANAGER')")
@GetMapping(path = {"/producer/{id}"}, produces = {"application/json"})
ResponseEntity<String> producerById(@NotNull @PathVariable String id) {...}

@PreAuthorize("hasRole('ROLE_MANAGER')")
@GetMapping(path = {"/producer/{id}/contract"}, produces = {"application/json"})
ResponseEntity<String> producerContracts(@NotNull @PathVariable String id) {...}

@CrossOrigin
@PreAuthorize("hasRole('ROLE_MANAGER')")
@DeleteMapping(path = {"/producer/{id}"})
ResponseEntity<String> deleteProducer(@NotNull @PathVariable String id) {...}

@PreAuthorize("hasRole('ROLE_MANAGER')")
@PatchMapping(path = {"/producer/{id}"}, consumes = "application/json", produces = "application/json")
ResponseEntity<String> updateProducer(@NotNull @RequestBody ProducerRequest producerRequest, @PathVariable String id) {...}
```

Сервіс постачальників:

Рис. 17 (Producer API)

Прикла

```
GET http://localhost:8080/api/producer
Authorization: "Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJUZXN0TmFtZSIsImlu
```

ди HTTTP запитів та їх відповіді:

Рис. 18 (Producer API)

```
[
  {
    "id": 9,
    "country": "Updated Country",
    "city": "Updated City",
    "street": "testStreet",
    "contactPhone": "+390 88 88 88 888",
    "contactName": "Updated Contact Name",
    "contactEmail": "test@mail.com",
    "companyName": "Updated Company Name"
  },
  {
    "id": 10,
    "country": "testCountry2",
    "city": "testCity2",
    "street": "testStreet2",
    "contactPhone": "+390 88 88 88 888",
    "contactName": "testContactName2",
    "contactEmail": "test2@mail.com",
    "companyName": "Test Company 2"
  }
]

Response code: 200; Time: 202ms; Content length: 430 bytes
```


Рис. 19 (Producer API)

Ри с. 20 (Producer API)

```
POST http://localhost:8080/api/producer
Authorization: "Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJUZXN0TmFtZSIsImlhdCI6M
Content-Type: application/json

{
  "country": "testCountry",
  "city": "testCity",
  "street": "testStreet",
  "contactPhone": "+390 88 88 88 888",
  "contactName": "testContactName",
  "contactEmail": "test@mail.com",
```

```
GET http://localhost:8080/api/producer/9/contract
Authorization: "Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJUZXN0TmFtZSIsImlhdCI6MTYzNz
```

Рис. 21 (Producer API)

```
PATCH http://localhost:8080/api/producer/9
Authorization: "Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJUZXN0TmFtZSIsImlhdCI6MT
Content-Type: application/json

{
  "country":"Updated Country",
  "city":"Updated City",
  "street":"testStreet",
  "contactPhone":"+390 88 88 88 888",
  "contactName":"Updated Contact Name",
  "contactEmail":"test@mail.com",
  "companyName": "Updated Company Name"
}
```

Рис. 22 (Producer API)

```
DELETE http://localhost:8080/api/producer/8
Authorization: "Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJUZXN0TmFtZSIsImlhdCI6MTYzNzk1NzczNiwiZXhwI
```

Рис. 23 (Producer API)

```java
@PreAuthorize("hasRole('ROLE_MANAGER')")
@GetMapping(path = {"/contract/{id}"}, produces = {"application/json"})
ResponseEntity<String> contractById(@NotNull @PathVariable String id) {...}

@PreAuthorize("hasRole('ROLE_MANAGER')")
@GetMapping(path = {"/contract/{id}/product"}, produces = {"application/json"})
ResponseEntity<String> contractProduct(@NotNull @PathVariable final String id, @RequestParam final String type) {...}

@PreAuthorize("hasRole('ROLE_MANAGER')")
@GetMapping(path = {"/contract/{id}/order"}, produces = {"application/json"})
ResponseEntity<String> contractOrders(@NotNull @PathVariable final String id) {...}

@PreAuthorize("hasRole('ROLE_MANAGER')")
@DeleteMapping(path = {"/contract/{id}"})
ResponseEntity<String> deleteContract(@NotNull @PathVariable String id) {...}

@PreAuthorize("hasRole('ROLE_MANAGER')")
@PostMapping(path = {"/contract"}, consumes = "application/json", produces = "application/json")
ResponseEntity<String> createContract(@NotNull @Valid @RequestBody NewContractRequest contractData) {...}
```

Сервіс контрактів:

Рис. 24 (Contract API)

Приклад HTTP запитів та їх відповіді:

```
GET http://localhost:8080/api/contract
Authorization: "Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJUZXN0TmFtZSIsImlhdCI6MTY
```

Рис. 25 (Contract API)

Рис. 26 (Contract API)

```
DELETE http://localhost:8080/api/contract/7
Authorization: "Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJUZXN0TmFtZSIsImlhdC
```

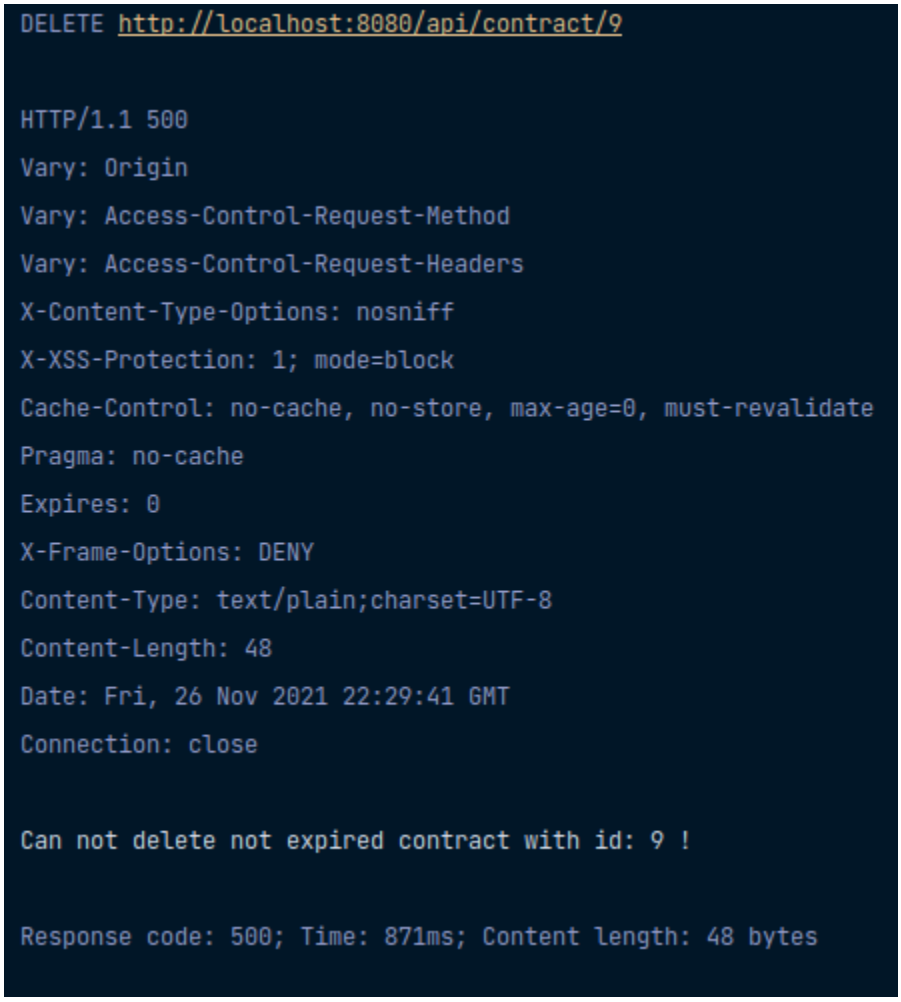


Рис. 27 (Contract API)

Рис. 28 (Contract API)

```
DELETE http://localhost:8080/api/product/4

HTTP/1.1 500
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: text/plain;charset=UTF-8
Content-Length: 62
Date: Sat, 27 Nov 2021 02:03:47 GMT
Connection: close

Can not delete product! Order with id 13 is still on the road!

Response code: 500; Time: 200ms; Content length: 62 bytes
```

Рис(Contract API)

```
GET http://loca

HTTP/1.1 200
Vary: Origin
Vary: Access-C
Vary: Access-C
X-Content-Type
X-XSS-Protecti
Cache-Control:
Pragma: no-cac
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Content-Length: 120
Date: Fri, 26 Nov 2021 23:48:31 GMT
Keep-Alive: timeout=60
Connection: keep-alive

[
  {
    "id": 3,
    "code": "44433235090",
    "name": "Coca-Cola 1.5L",
    "quantity": 50,
    "price": 20.00,
    "criticalQuantity": 10,
    "contractId": 9
  }
]
```

(Contract API)

Рис. 30

Сервіс продукції:

```java
@PreAuthorize("hasAnyRole('ROLE_MANAGER', 'ROLE_USER')")
@GetMapping(path = {"/product"}, produces = {"application/json"})
ResponseEntity<String> productsAll() {...}

@PreAuthorize("hasAnyRole('ROLE_MANAGER', 'ROLE_USER')")
@GetMapping(path = {"/product/{id}"}, produces = {"application/json"})
ResponseEntity<String> product(@PathVariable String id) {...}

@PreAuthorize("hasRole('ROLE_MANAGER')")
@DeleteMapping(path = {"/product/{id}"})
ResponseEntity<String> deleteProduct(@NotNull @PathVariable String id) {...}

@PreAuthorize("hasRole('ROLE_MANAGER')")
@PatchMapping(path = {"/product/{id}"}, consumes = {"application/json"}, produces = "application/json")
ResponseEntity<String> modifyProduct(@NotNull @RequestBody ProductData newProduct, @PathVariable String id) {...}

@PreAuthorize("hasRole('ROLE_MANAGER')")
@PostMapping(path = {"/product"}, consumes = {"application/json"}, produces = {"application/json"})
ResponseEntity<String> createProduct(@NotNull @RequestBody ProductData newProduct) {...}
```

Рис. 31 (Product API)

Приклад HTTP запитів та їх відповіді:

```
GET http://localhost:8080/api/product/3
Authorization: "Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJUZXN0TmFtZSIsImlhc

###
```

Рис. 32 (Product API)

```
GET http://localhost:8080/api/product/3

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Content-Length: 118
Date: Sat, 27 Nov 2021 00:03:04 GMT
Keep-Alive: timeout=60
Connection: keep-alive

{
  "id": 3,
  "code": "44433235090",
  "name": "Coca-Cola 1.5L",
  "quantity": 50,
  "price": 20.00,
  "criticalQuantity": 10,
  "contractId": 9
}

Response code: 200; Time: 210ms; Content length: 118 bytes
```

Рис. 33 (Product API)

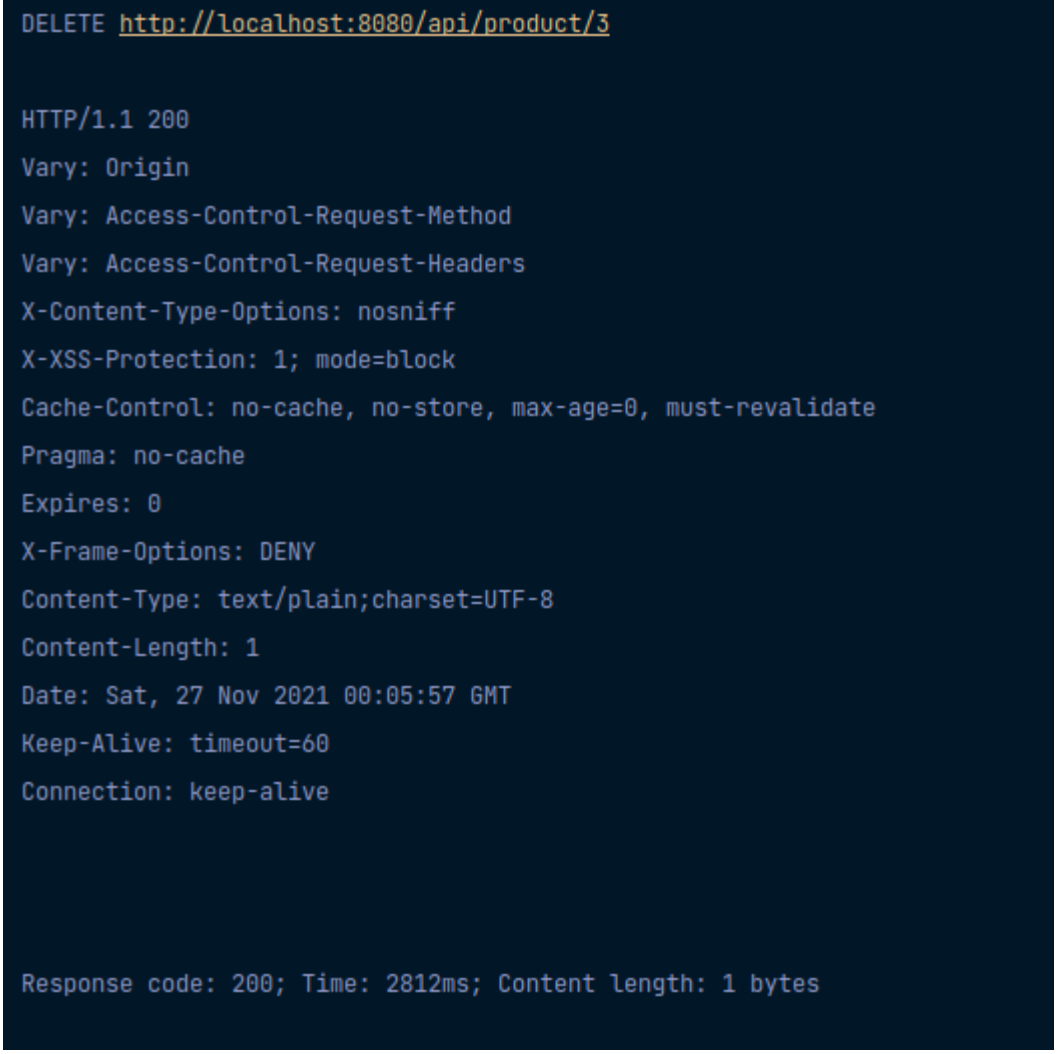


Рис. 34 (Product API)

API всіх сервісів схожі за структурою, але реалізуються згідно індивідуальних бізнес-правил.

API всіх сервісів, які залишились:

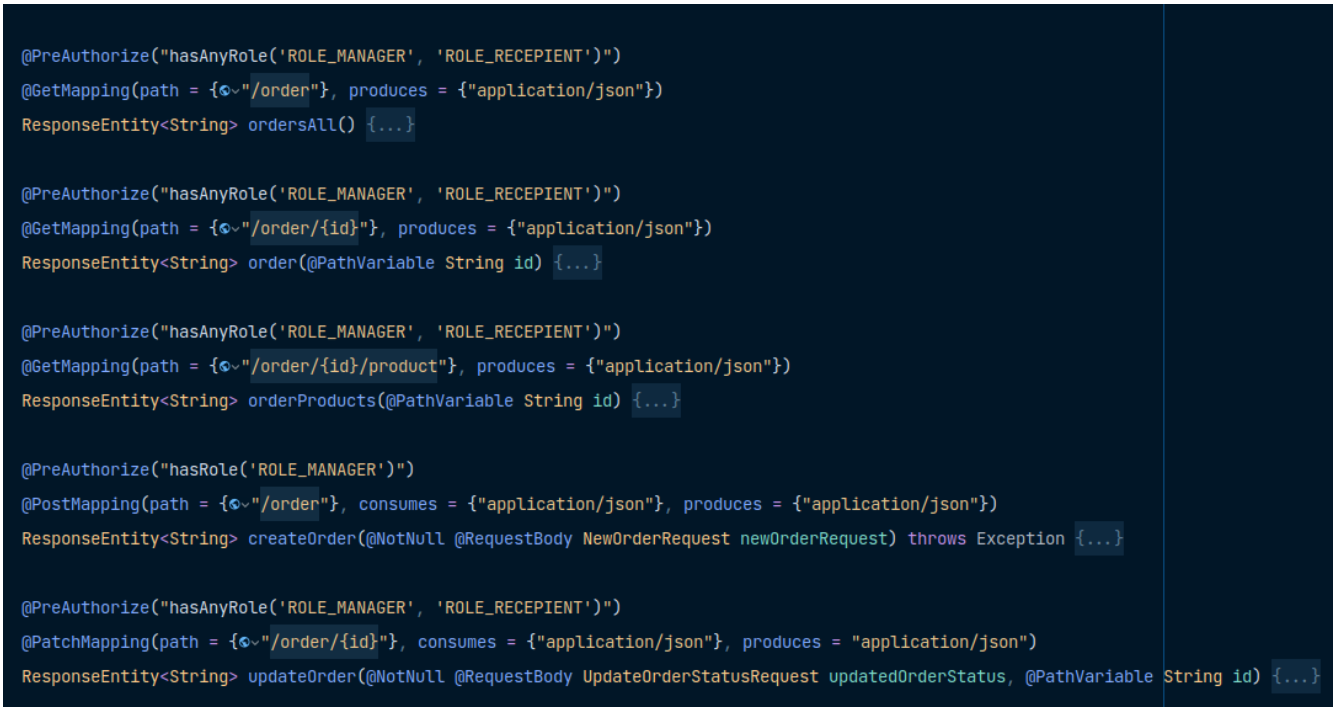


Рис. 35 (Order API)

Рис. 36 (Statistics and Purchases API)

Сервіс підтвердження замовлень постачальником був реалізований в якості імітації реального серверу:

```java
@RestController
@RequestMapping("/api/confirmation")
public class ConfirmationController {

    private final OrderFacade orderFacade;


    public ConfirmationController(final OrderFacade orderFacade) { this.orderFacade = orderFacade; }


    @PatchMapping(path = {"/order/{id}"}, consumes = "application/json")
    ResponseEntity<String> confirmOrder(@NotNull @Valid @RequestBody OrderConfirmationResponse orderConfirmationRequest,
                                        @PathVariable @NotNull String id) {...}
}
```

Рис. 37 (Producer Confirmation API)

4.3 Реалізація клієнтської частини

Клієнтська частина реалізована як компонент View із паттерну **MVC.** Вона відповідає за валідацію введених користувачем даних і надання впорядкованого представлення інформації, отриманих в результаті взаємодій із сервісами. Для звернення використовується звичайний HTTP клієнт, який здатний надсилати та отримувати запити.

Для початку роботи із сервісом користувачу потрібно зареєструватись. Так виглядає компонент реєстрації нових користувачів з введеними даними:

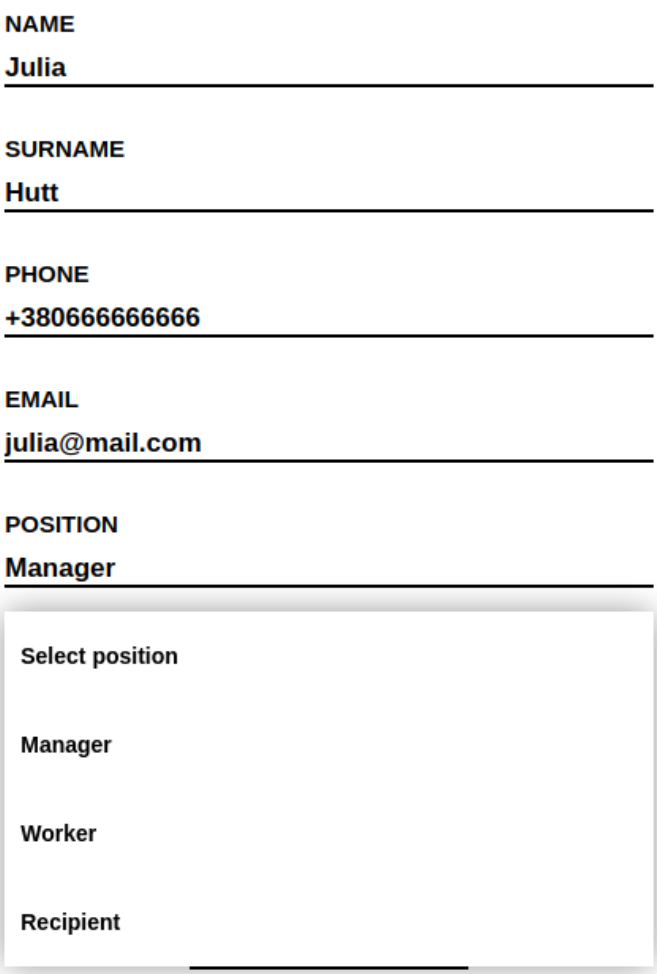


Рис. 38 (Regestartion Page)

Також реалізована валідація введених даних:

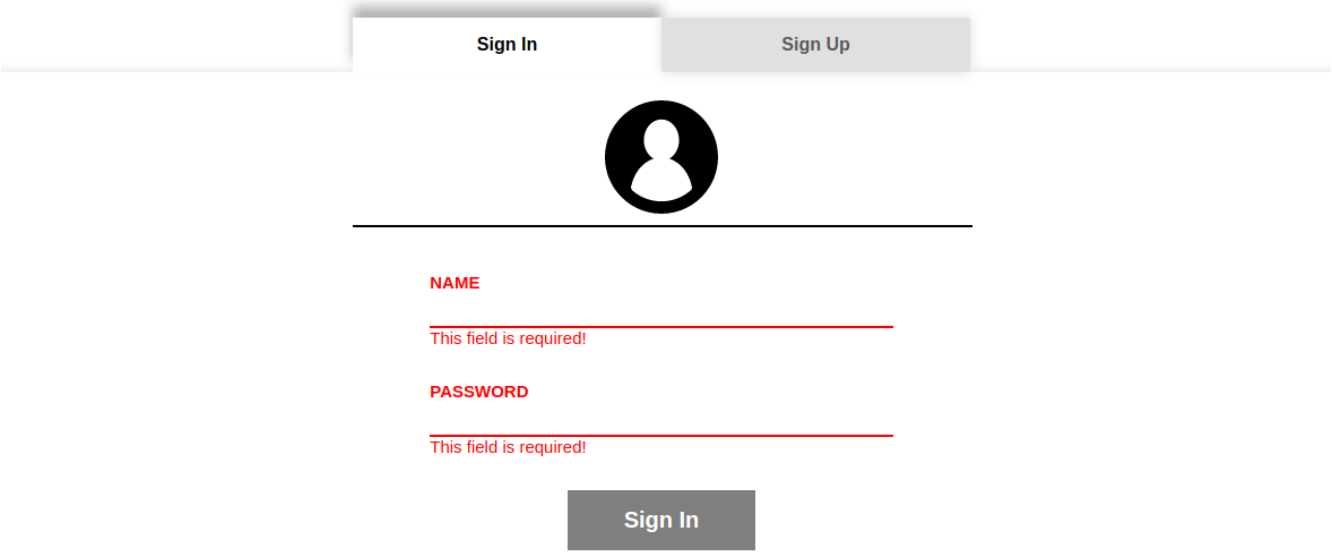


Рис. 39 (Login Page)

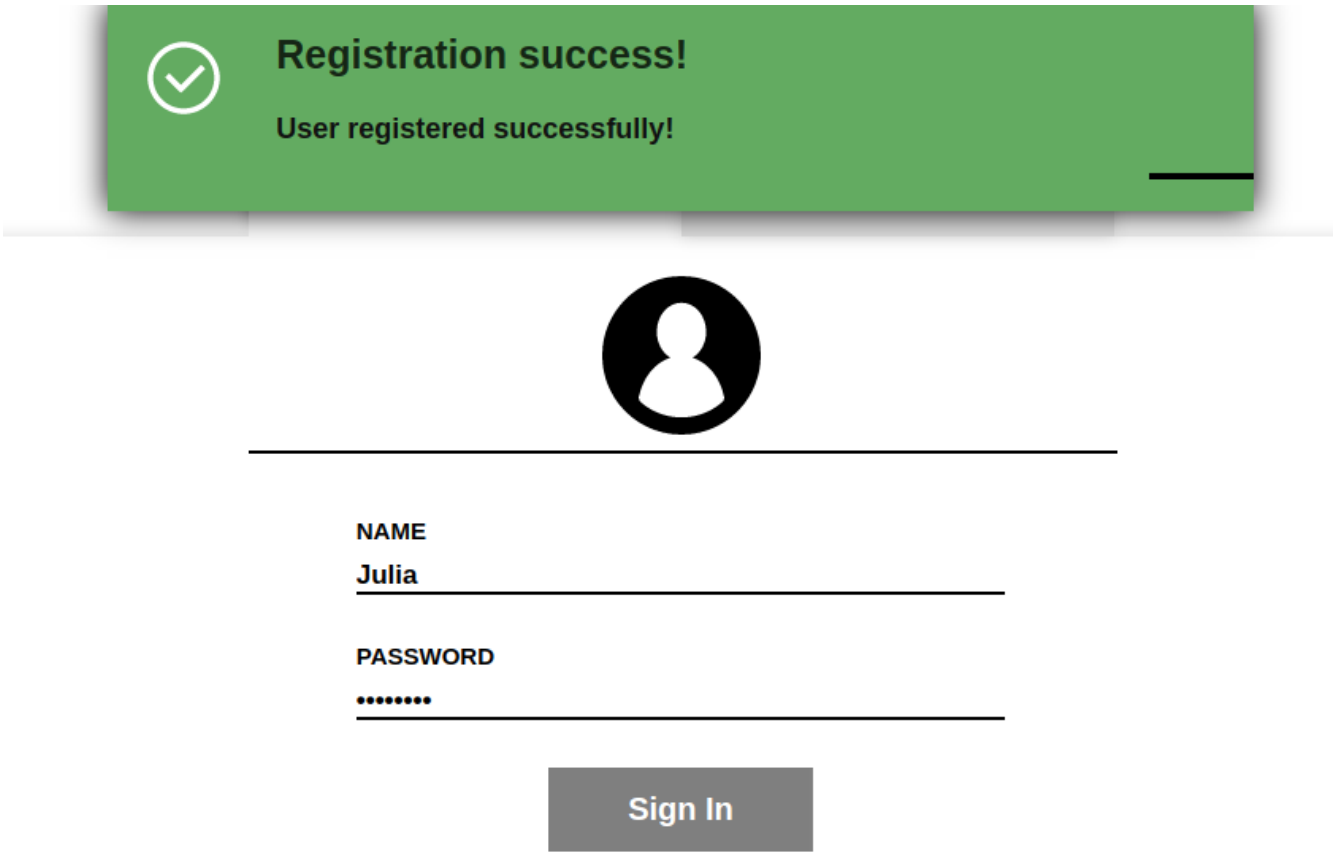


Рис. 40 (Login Page)

Так виглядає панель менеджерів. На ній знаходяться графіки продажів обраних товарів за певний інтервал часу та процентні відношення сум з кожного товару:

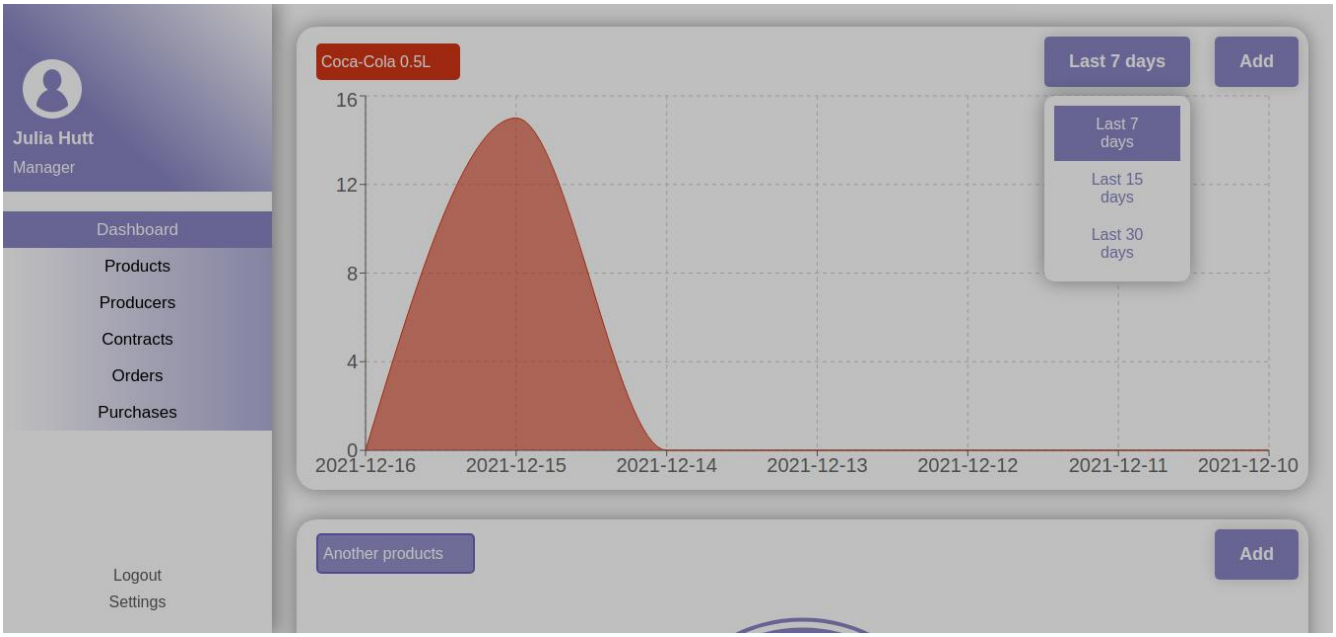


Рис. 41 (Manager Dashboard)

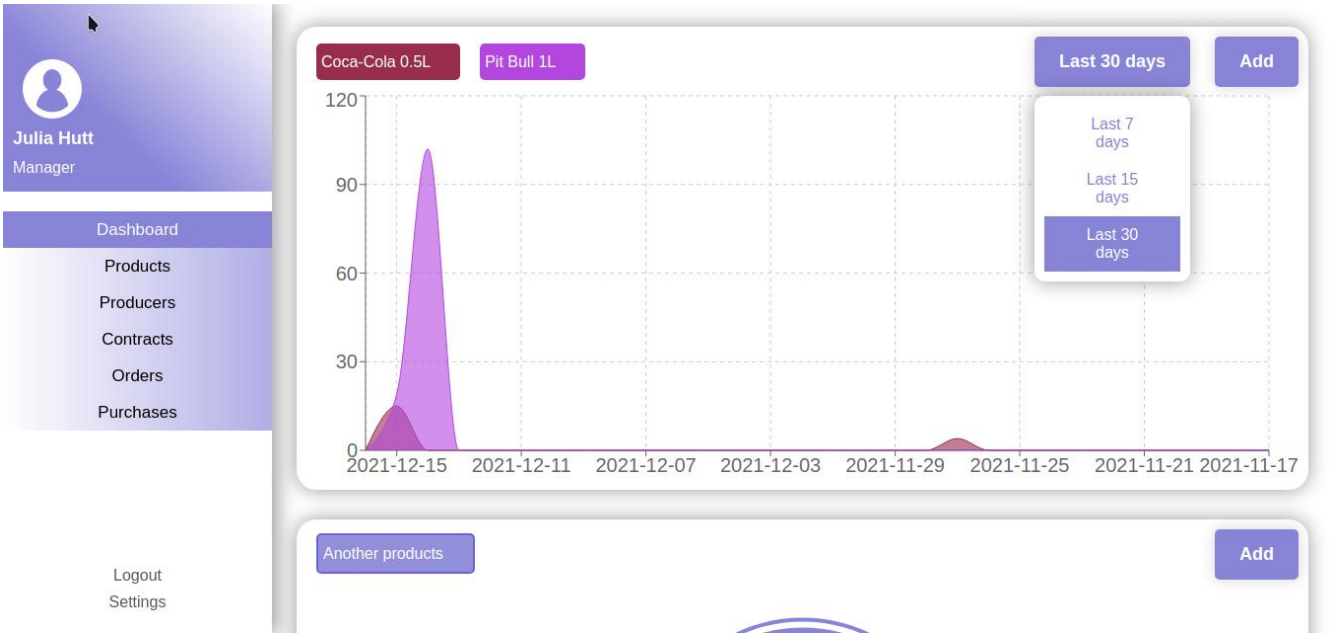


Рис. 42 (Manager Dashboard)

Рис. 43 (Manager Dashboard)

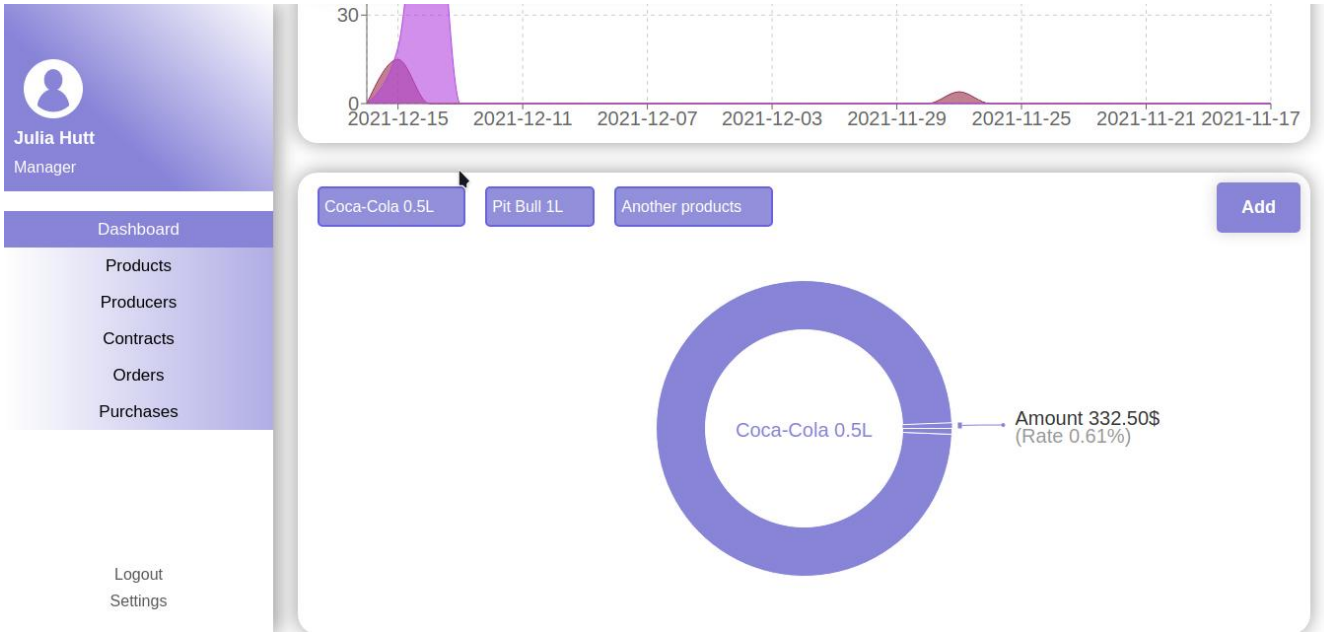


Рис. 44 (Manager Dashboard)

Також реалізовано відображення предметних сутностей у табличному вигляді та індикація згідно їх особливостей. Наприклад контракти, в яких вийшов термін дії мають червону індикацію, а замовлення, які в очікуванні — жовтим:

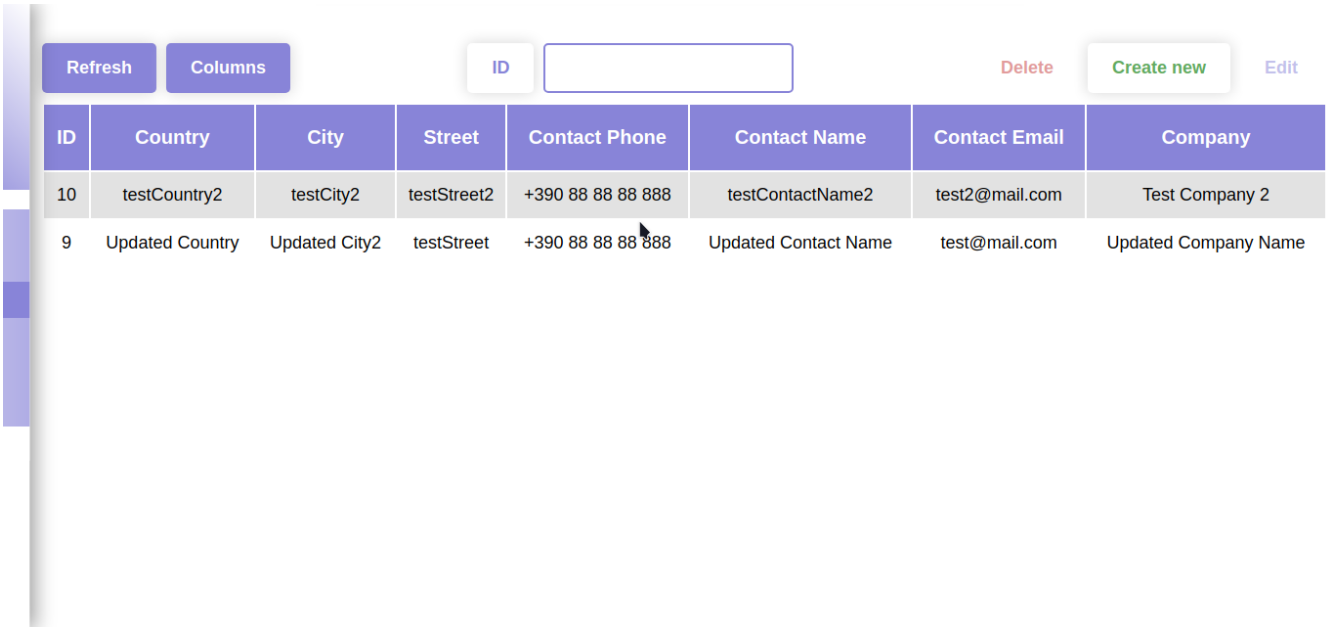


Рис. 45 (Producr Table)

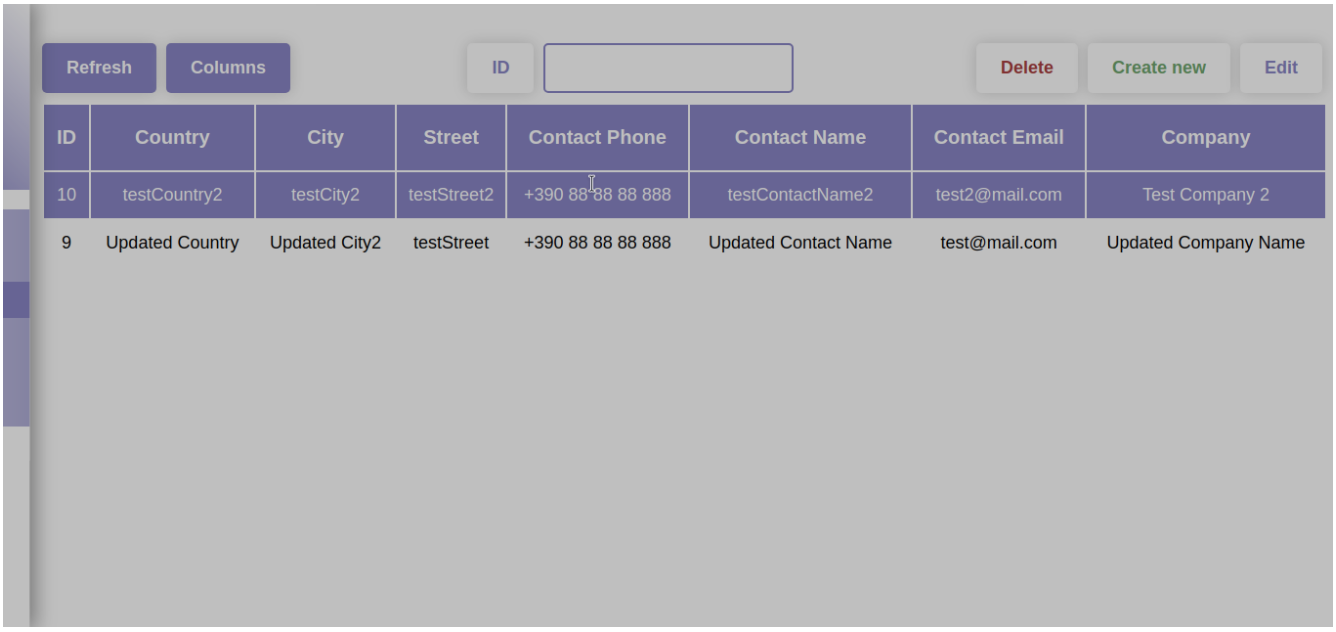
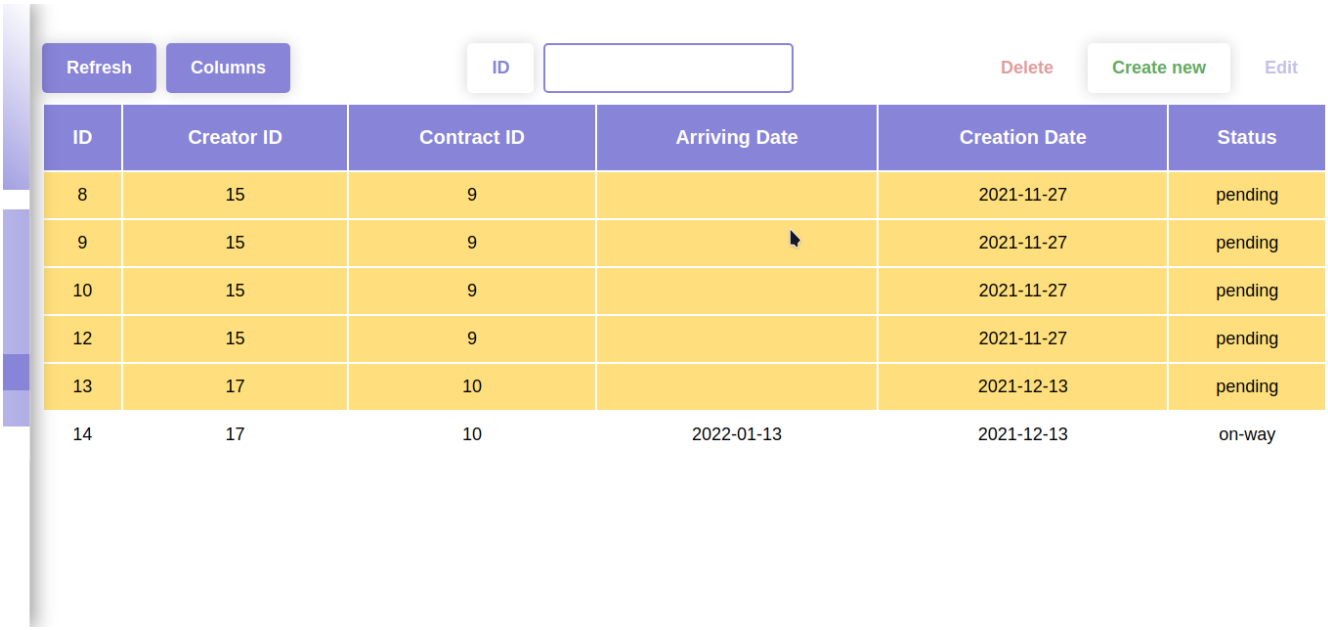


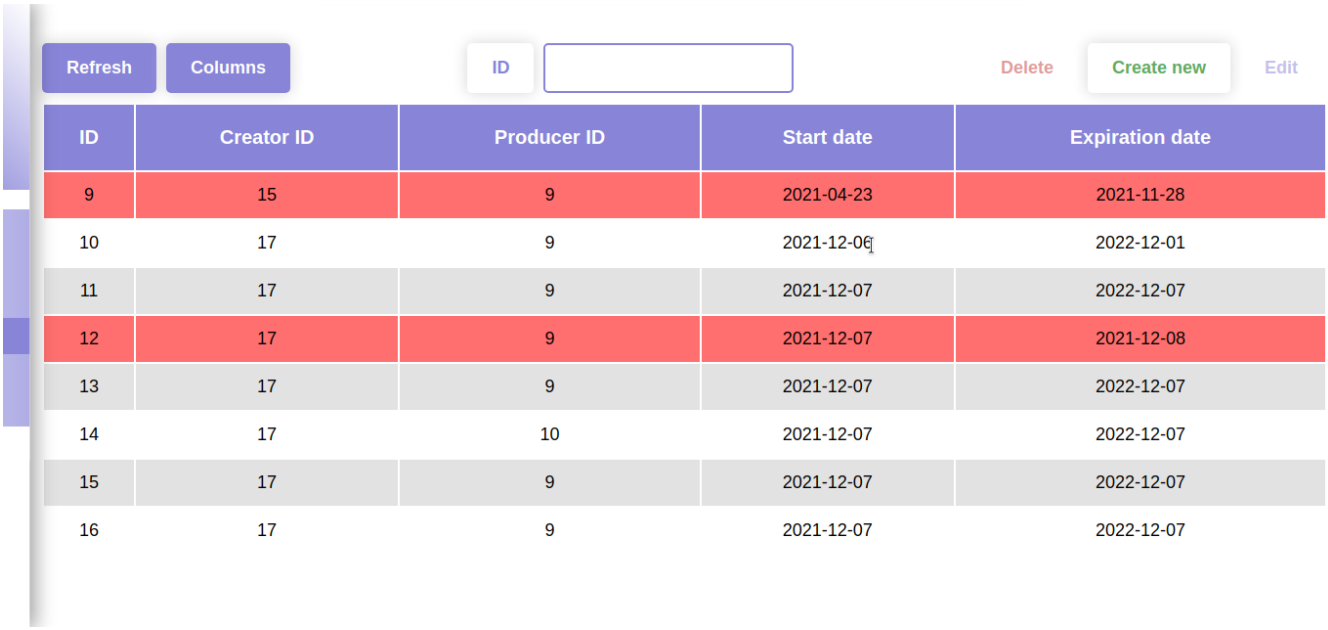
Рис. 46 (Manager Dashboard)



Refresh Columns ID Delete Create new Edit

ID	Creator ID	Contract ID	Arriving Date	Creation Date	Status
8	15	9		2021-11-27	pending
9	15	9		2021-11-27	pending
10	15	9		2021-11-27	pending
12	15	9		2021-11-27	pending
13	17	10		2021-12-13	pending
14	17	10	2022-01-13	2021-12-13	on-way

Рис. 47 (Order Table)



Refresh Columns ID Delete Create new Edit

ID	Creator ID	Producer ID	Start date	Expiration date
9	15	9	2021-04-23	2021-11-28
10	17	9	2021-12-06	2022-12-01
11	17	9	2021-12-07	2022-12-07
12	17	9	2021-12-07	2021-12-08
13	17	9	2021-12-07	2022-12-07
14	17	10	2021-12-07	2022-12-07
15	17	9	2021-12-07	2022-12-07
16	17	9	2021-12-07	2022-12-07

Рис. 48 (Contracts Table)

Також кожну передбачені компоненти для редагування кожної сутності згідно правил:

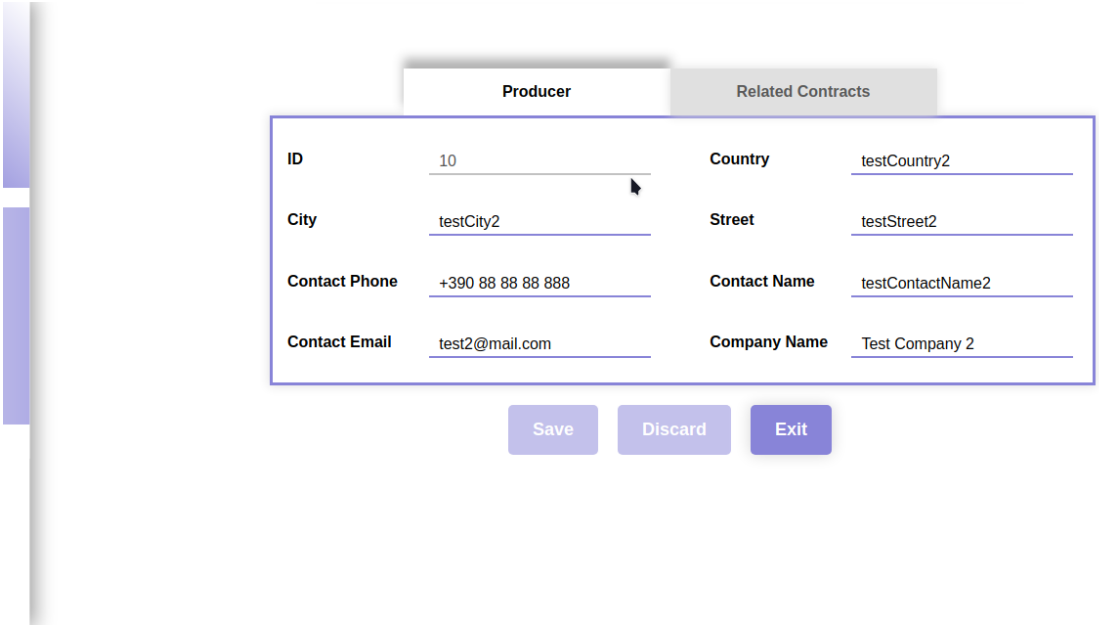


Рис. 49 (Producer Edit View)

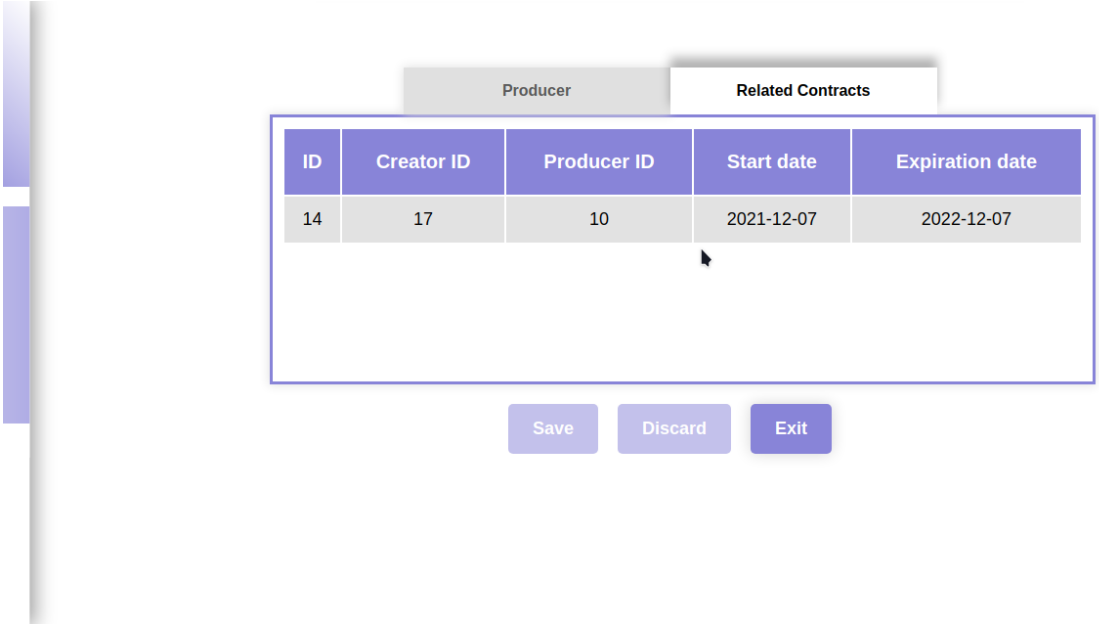


Рис. 49 (Producer Edit View)

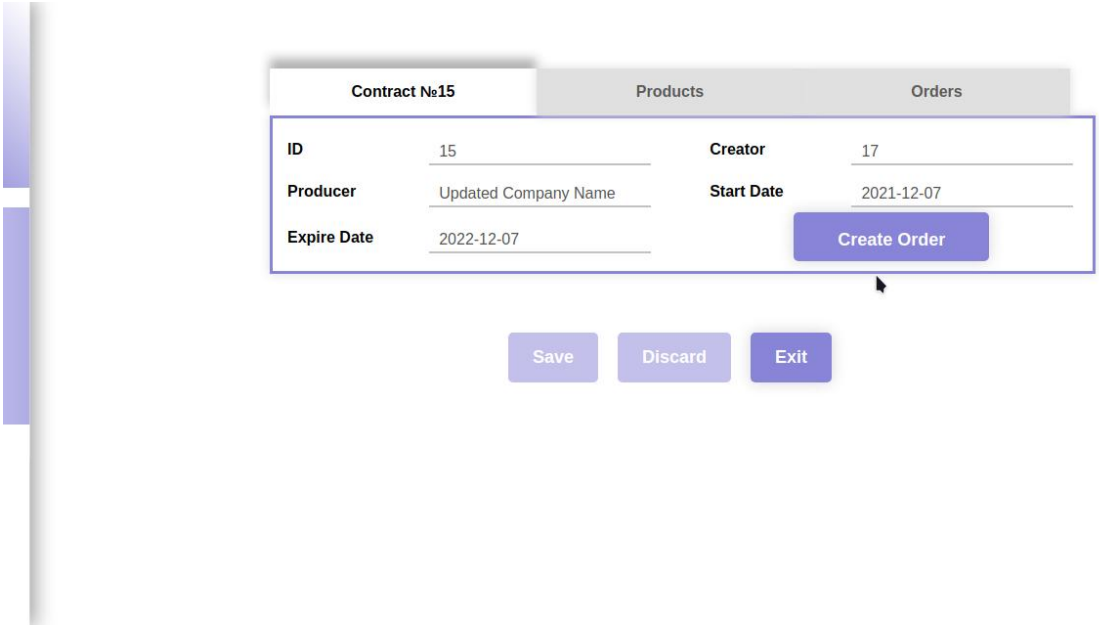


Рис. 50 (Contract Edit View)

При невірному введенні даних активується індикація, всі способи збереження і підтвердження змін блокуються:

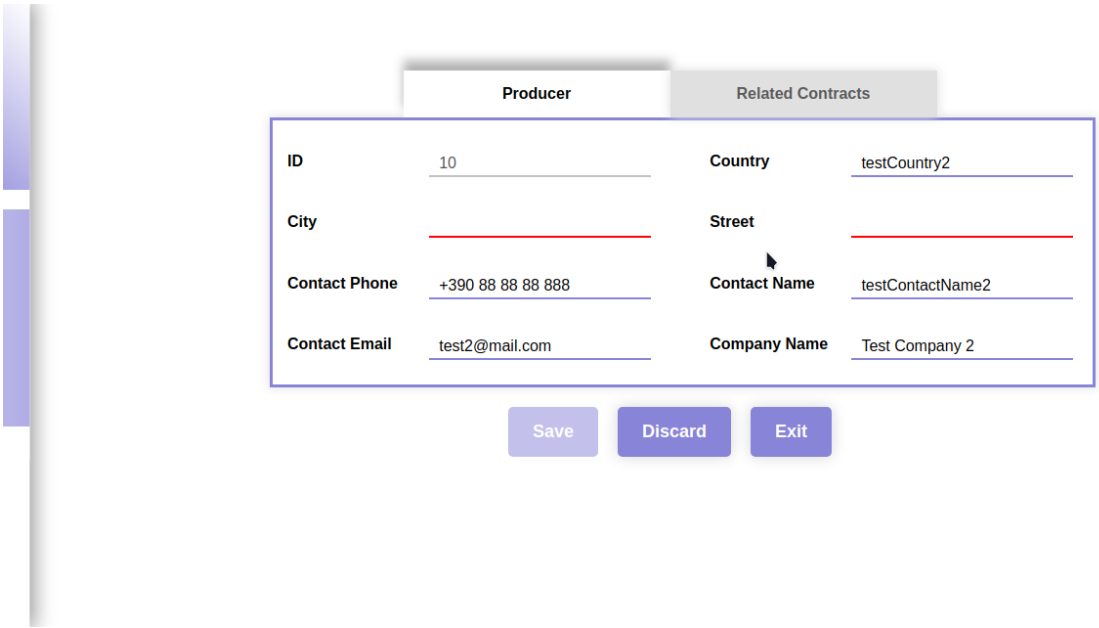


Рис. 51 (Invalid Contract Edit View)

Висновки до розділу

В даному розділі виконано кінцеву фазу реалізації системи та всіх її запланованих можливостей. Було виконано етапи аналітики бізнес-області, проектування систем, аналіз та передбачення майбутнього розширення функціональних можливостей та тестування обох систем з реальними даними.

ПЗ сервісів було реалізовано без жорсткої прив’язки до конкретних інструментів (фреймворків) та із слабкою залежністю один від одного, а клієнтська частина не зберігає ніяких даних щодо бізнес-логіки та реалізує приємний, простий та інтуїтивно зрозумілий інтерфейс. Це і було технічною метою даного розділу - максимально можлива відповідність до загальноприйнятих концепцій програмування.

РОЗДІЛ 5

5.1 Опис ідеї проекту

Почнемо із визначення того чим ж є даний проект. Ця система втілює в себе не одну, а одразу декілька систем, тим самим виконуючи роль комплексного рішення для проблем та потреб потенційних користувачів і підприємств у сфері роздрібної торгівлі. Система реалізує можливість:

1. Відстеження та аудиту товару в наявності;
2. Аналіз продажів;
3. Візуалізація статистики збутих товарів та прибутку;
4. Створення та взаємодія з актуальними постачальниками ресурсу;
5. Вкладення контрактів з постачальниками;
6. Ведення процесів поповнення цінних ресурсів;
7. Збереження користувачів системи та їх повноважень і надання їм обов’язків по чіткій схемі;
8. Зрозуміла та проста візуалізація всіх процесів.

Таким чином зміст проекту полягає в автоматизованому способі ведення процесів підприємства та зведення втрат до мінімуму за рахунок усунення людського фактору.

Також не варто забувати, що сервіси(сервер по обробці даних) і клієнтська сторона, що реалізує інтерфейс користувача та інтерактивність, не зв’язані між собою програмно, отож можуть бути взаємозамінні, модифіковані та розширені без внесення змін в іншу. Це дозволяє вести розробку над кожним компонентом

системи абсолютно роздільно, що, в свою чергу, зменшує фінансове навантаження на розробку продукту та пришвидшує її.

Користувачі такої системи отримують більший контроль за тим, як відбуваються процеси і, як наслідок, приймають більш зважені та доцільніші рішення на основі збільшення корисного інформаційного поля.

Одним із аналогів у даному виді **ПЗ** є CHAMELEON від компанії CHM Software. Це впливовий, на дану сферу, український розробник IT-рішень для автоматизації торгівлі. Основна і, мабуть, головна відмінність цих систем — покриття CHAMELEON'ом набагато більшої сфери процесів із детально продуманою і глибокою аналітикою. Це **ПЗ** займає стійку позиції на ринку.

5.2 Аналіз технологічних можливостей реалізації ідей проекту

Детальний опис технологій, які використовувались у реалізації кожної із систем, описані у **Розділі 3.** Кожна із наступних технологій доступна широкій аудиторії розробникам різного рівня та є самодостатньою.

Дані технології використовуються для реалізації користувацького сервер:

1. JavaScript;
2. NodeJS;
3. ReactJS(JSX);
4. HTML/CSS.

Дані технології використовуються для реалізації сервісної системи:

1. Java;
2. Maven;

3. PostgreSQL;
4. Spring Framework;
5. Hibernate;
6. JWT.

5.3 Аналіз ринкових можливостей стартап продукту

Сфера споживчого ринку, зокрема роздрібної торгівлі, є найбільш динамічно прогресуючою і оперативно реагуючою на зміну економічних чинників і коливання споживчого попиту. Так як, по статистиці, економічна платоспроможність середнього покупця зростає також, відповідно зацікавленість підприємств в системах такого типу тільки зростатиме. Виходячи із цього можна припустити, що для збільшення фінансового обороту підприємства роздрібної торгівлі потрібно збільшити його об’єм товарів, а значить і створити додаткове навантаження на всі відділи та групи по контролюванню внутрішніх процесів підприємства(менеджмент, аудит і т.д). Це можна компенсувати за рахунок збільшення персоналу та активних територій, які належать підприємству, але цей варіант призведе до більших фінансових затрат на оплату праці.

Основною групою підприємств, що зацікавлені в **ПЗ**, яке розглядається в даній дипломній роботі є підприємства з невеликим фінансовим та товарним оборотом. Цьому є ряд раціональних причин:

1. Вимоги, що були сформовані під час аналізу проблематики із **Розділу 1** охоплюють не всі можливі процеси, які існують у даній сфері. Відповідно система також не має відповідної функціональності;
2. Деталізація кожного процесу оброблена не максимально, із за відсутності повної інформаційної забезпеченості;

3. Відсутність реальної області застосування та тестування в конкретному підприємстві із реальними ситуаціями.

Всі ці пункти наводять та висновки, що застосування даної системи до підприємств із масштабними бізнес-процесами може спричинити реальні ризики як для самого підприємства, так і для репутації самого продукту, і компанії-засновника.

Вище було зазначено, що дана система є легко адаптивною та розширювана, за рахунок своєї невеликої складності та архітектурним рішенням. Також із цих характеристик виходить ще одна — дешевша вартість обслуговування, на відміну від конкурентів.

5.4 Розроблення ринкової стратегії

На підставі висновків із **Розділів 5.1 — 5.3** найоптимальнішою ринковою стратегією є концентрований маркетинг. В такій стратегії передбачається орієнтація діяльності на одному невеликому ринковому сегменті – ринковій ніші.

Переваги для даного продукту(системи) наступні:

1. Ніша має порівняно менший розмір, аніж сегмент;
2. Ніша є результатом розподілення споживачів за сукупністю ознак;
3. Ніша має менш агресивну конкуренцію, ніж ніша.

Опис цільової групи потенційного споживача також був виявлений у цих розділах.

Висновки до розділу

В даному розділі було визначено, що для реалізації такого роду систем не обов’язкова наявністю складних інструментів та технологій. Натомість варто загострити увагу на реалізації широкого діапазону внутрішніх процесі та потенціальних можливостей в програмному просторі системи. Основні ідеї проекту розглядаються у **Розділі 1**, а в поточному розділі лише доповнюються ключовими деталями згідно економічних та ринкових аспектів.

ВИСНОВКИ

У цій дипломній роботі було реалізовано програмне забезпечення для системи по автоматизованому контролю, моніторингу та аудиту бізнес-процесів для підприємств малого бізнесу. Було проведено первинний аналіз характерної предметної області, її етапів та особливостей. На базі аналізу було складено узагальнені функції підприємства, на основі яких було сформовано бізнес-вимоги до майбутнього програмного забезпечення.

Були пройдені всі етапи життєвого циклу по розробці системи, а саме:

1. Проектування сховища бази даних, для зберігання програмних сутностей з метою подальшої обробки, відображення та моніторингу;
2. Проектування об’єктних представлень, що відображають базові структури бізнесу на основі їх теоретичного та інформаційного представлення;
3. Завчасна оцінка внутрішніх процесів та декларування їх в UML-діаграмах з метою врахування їх при проектуванні сервісів;
4. Первинне проектування сервісів з допомогою UML-діаграм. Передбачення їх взаємодії із об’єктами доменної області та іншими сервісами, які відображають певні бізнес-процеси.
5. Вибір архітектури систем(сервісної та клієнтської) згідно із завчасного аналізу технологій;
6. Реалізація обраної архітектури згідно спроектованих UML-діаграм сервісів та об’єктів предметної області. Також інтеграція обраних технологій (фреймворків) на етапі розробки.
7. Реалізація клієнтського серверу, інтерфейсу та доступу до функціоналу сервісів згідно вимог;

8. Тестування реалізованих систем тестовими даними та перевірка результату на відповідність до вимог.

Кінцевим етапом було аналізування продукту у розрізі економічного та ринкового аспектів. Було здійснено визначення теоретичного рівня зацікавленості клієнтів продуктом, як рішенням із певним потенціалом до вирішення їх підприємницьких проблем. На базі цього заключення виявилось в якому саме секторі та які унікальні проблеми вирішує дана система.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Джошуа Блох, «Java, Эффективное программирование», г. 2001 – 871 ст;
2. Сем Ньюмен, «Создание микросервисов», г. 2016 – 1235 ст;
3. Юлиана Козмина, Крис Шефер, Роб Харроп, Кларенс Хо, «Spring для профессионалов», г. 2019;
4. Кристиан Бауэр, «Java Persistence Api и Hibernate», г. 2017 – 439ст;
5. Тиленс Томас Марк, «React в действии», г. 2018 – 341ст;
6. Кантелон Майк, Хартер Марк, Райлих Натан, Головайчук TJ, «Node.js в действии», г. 2014 – 658 ст;
7. Арно Лоре, «Проектирование Web-API », г. 2020 – 431 ст;