БАКАЛАВРСЬКА РОБОТА

БР. ІП - 24.00.00.000 ПЗ

Група ІП-21-2

Ганчук Олександр

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ганчук Олександр Дмитрович

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Створення прототипу екосистеми для рішення суміжних задач клієнтів

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач освітнього рівня Ганчук О.Д.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Процюк Галина Ярославівна, асистент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Ганчуку Олександру Дмитровичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) “ Створення прототипу екосистеми для рішення суміжних задач клієнтів”

керівник проекту (роботи) Процюк Г.Я., асистент

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області суміжності задач, комп’ютерних продуктів і компонентів
2. Проектування архітектури прототипу екосистеми для рішення конфігурацій суміжних задач
3. Опис категорій суб'єктів взаємодії (акторів) в системі
4. Розробка та опис моделі бази даних
5. Програмна реалізація прототипу екосистеми для рішення конфігурацій суміжних задач

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Функціональні можливості пропонованої системи з точки зору користувача (рис. 1.1)
2. Функціональні можливості пропонованої системи з точки зору адміністратора (рис. 1.2)
3. Архітектура Spring MVC (рис. 1.3)
4. Конфігурація Spring Security (рис. 1.4)
5. Контейнери Bootstrap (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз предметної області суміжності задач, комп’ютерних продуктів і компонентів	06.05.2025	виконано
2	Проектування архітектури прототипу екосистеми для рішення конфігурацій суміжних задач	16.05.2025	виконано
3	Опис категорій суб'єктів взаємодії (акторів) в системі	26.05.2025	виконано
4	Розробка та опис моделі бази даних	01.06.2025	виконано
5	Програмна реалізація прототипу екосистеми для рішення конфігурацій суміжних задач	06.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	10.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 77 сторінок, 57 рисунків, список використаних джерел із 33 найменуваннями, 1 додаток.

Метою даної роботи є дослідження, проєктування та реалізація прототипу програмної екосистеми, орієнтованої на вирішення задач конфігураційної сумісності продукції в електронній комерції.

Об'єкт дослідження - цифрові інформаційні системи для управління конфігурацією та сумісністю компонентів продуктів.

Предмет дослідження - методи, моделі та програмні інструменти реалізації прототипу екосистеми для вирішення задач суміжності та конфігурації.

В першому розділі розглянуто проблематику сумісності компонентів у цифрових продуктах, обґрунтовано мету системи, описано функціональність і технології, використані для її розробки.

В другому розділі розроблено архітектуру прототипу, описано категорії користувачів, застосовано шаблони проєктування та розроблено модель бази даних для забезпечення цілісності системи

В третьому розділі Реалізовано програмні модулі адміністратора і користувача, впроваджено механізми безпеки, здійснено інсталяцію та розгортання прототипу в середовищі Java EE.

Висновок: реалізовано прототип системи із розмежуванням ролей користувачів, підтримкою адміністративного управління, базою даних і механізмами безпеки.

КЛЮЧОВІ СЛОВА: КОНФІГУРАЦІЯ ПРОДУКТІВ, СУМІСНІСТЬ, ЕЛЕКТРОННА КОМЕРЦІЯ, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ПРОГРАМНА ЕКОСИСТЕМА, SPRING, HIBERNATE

ANNOTATION

The Bachelor's thesis comprises 77 pages, 57 figures, a list of 33 references, and 1 appendix.

The objective of this work is to research, design, and implement a prototype of a software ecosystem aimed at solving product configuration compatibility issues in e-commerce.

The object of study is digital information systems for managing product component configuration and compatibility.

The subject of study is methods, models, and software tools for implementing an ecosystem prototype to solve compatibility and configuration problems.

In the first chapter, the issues of component compatibility in digital products are examined, the system's objective is justified, and the functionality and technologies used for its development are described.

In the second chapter, the prototype's architecture is developed, user categories are described, design patterns are applied, and a database model is developed to ensure system integrity.

In the third chapter, administrator and user software modules are implemented, security mechanisms are introduced, and the prototype is installed and deployed in a Java EE environment.

Conclusion: A prototype system with user role differentiation, administrative management support, a database, and security mechanisms has been implemented.

KEYWORDS: PRODUCT CONFIGURATION, COMPATIBILITY, E-COMMERCE, SOFTWARE ARCHITECTURE, SOFTWARE ECOSYSTEM, SPRING, HIBERNATE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CSS - Cascading Style Sheets

CSRF - Cross Site Request Forgery

DAO - Data Access Object

DI - Dependency Injection

GUI - Graphical User Interface

IDE - Integrated Development Environment

JRE - Java Runtime Environment

JDBC - Java Database Connectivity

JVM Java Virtual Machine.

JPA - Java Persistence API

POJO - Plain Old Java Object

ВСТУП

У сучасних умовах цифровізації та динамічного розвитку електронної комерції проблема ефективного управління конфігурацією продукції стає особливо актуальною. Швидка адаптація продуктів до вимог ринку, забезпечення їх сумісності та уніфікації потребує інтелектуальної підтримки з боку програмних засобів. На практиці часто виникають труднощі при інтеграції різнорідних компонентів, обміні метаданими, управлінні варіантністю та супроводі продукції. Розробка гнучких архітектурних рішень і програмних екосистем для вирішення суміжних задач конфігурації стає необхідною умовою для забезпечення конкурентоспроможності підприємств та оптимізації бізнес-процесів.

Дана робота присвячена аналізу, проєктуванню та реалізації програмного прототипу екосистеми, здатної автоматизовано вирішувати задачі конфігурації сумісних продуктів. У дослідженні розглянуто проєктну архітектуру, функціональні модулі, ролі користувачів, шаблони проєктування та реалізацію всіх необхідних компонентів з використанням сучасних технологій на базі Java EE, Hibernate, Spring MVC, Spring Security та Apache Solr.

Актуальність роботи

Актуальність зумовлюється відсутністю універсальних рішень, здатних охопити повний цикл керування конфігурацією продуктів з урахуванням вимог різних категорій користувачів.

Зростання обсягів даних, ускладнення асортиментної структури продукції та розвиток індивідуалізованих підходів у бізнесі формують запит на нові програмні рішення, здатні забезпечити ефективне управління конфігурацією продуктів і їх сумісністю. Традиційні підходи до конфігураційного управління виявляються недостатніми в умовах постійної

зміни вимог, багатоваріантності продукції та інтеграції з численними IT-сервісами.

Метою даної роботи є дослідження, проєктування та реалізація прототипу програмної екосистеми, орієнтованої на вирішення задач конфігураційної сумісності продукції в електронній комерції.

У рамках дослідження було проаналізовано предметну область, виявлено основні проблеми та виклики, а також визначено функціональні та технічні вимоги до майбутньої системи.

Завдання дослідження:

1. Проаналізувати предметну область проблеми суміжності задач і компонентів.

2. Розробити концепцію архітектури інформаційної системи.

3. Визначити ролі користувачів і функціональні вимоги.

4. Впровадити шаблони проєктування та ORM-підходи.

5. Розробити базу даних та відповідні модулі взаємодії.

6. Реалізувати програмний прототип із функціоналом керування користувачами та конфігураціями.

Об’єкт дослідження - цифрові інформаційні системи для управління конфігурацією та сумісністю компонентів продуктів.

Предмет дослідження - методи, моделі та програмні інструменти реалізації прототипу екосистеми для вирішення задач суміжності та конфігурації.

Методи дослідження:

- системний аналіз предметної області;
- моделювання взаємодій та ролей (акторів);
- об'єктно-орієнтоване проєктування (шаблони MVC, DAO, Factory);
- проєктування бази даних;
- програмна реалізація засобами Java EE, Spring, Hibernate;
- тестування функціональності та розгортання на Tomcat.

Наукова новизна

Запропоновано архітектурну модель цифрової екосистеми, що враховує множинні категорії користувачів та використовує інтегровані шаблони проектування для забезпечення ефективної обробки сумісності компонентів у межах складних конфігурацій.

Практичне застосування

Результати роботи можуть бути використані як основа для розгортання повноцінних корпоративних систем управління конфігурацією продукції, а також для подальших наукових досліджень у галузі інформаційних технологій, програмної інженерії та цифрової трансформації бізнесу.

На етапі проєктування розроблено архітектуру, яка базується на сучасних шаблонах проєктування та підтримує гнучкість, масштабованість і безпеку. Реалізація системи здійснювалася із використанням актуальних технологій (J2EE, Spring, Hibernate, Bootstrap, Apache Solr), що забезпечує високу продуктивність та розширюваність розробленого рішення. Особливу увагу приділено моделюванню бази даних, об’єктно-реляційному відображенню та структурі інтерфейсів користувачів.

Бакалаврська робота містить 77 сторінок, 57 рисунків, 3 розділи список використаних джерел із 33 найменуванням, 1 додаток.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПРОБЛЕМИ СУМІЖНОСТІ ЗАДАЧ, КОМП'ЮТЕРНИХ ПРОДУКТІВ І КОМПОНЕНТІВ

1.1. Оптимізація процесів сумісності конфігурації та управління продукцією в електронній комерції

В умовах сучасного ринку електронної комерції, особливо в сегменті інформаційних технологій, споживачі часто стикаються зі складністю вибору та конфігурації складних продуктів. Процес вибору IT-обладнання в онлайн-магазинах може ускладнюватися через потенційну несумісність компонентів або необхідність придбання додаткових елементів для забезпечення повноцінного функціонування обраної конфігурації. Паралельно, власники електронних магазинів стикаються з викликами ефективного управління асортиментом, ціноутворенням, встановленням бізнес-правил для формування замовлень та інтеграцією даних про клієнтів з різнорідних джерел, що часто вимагає ручної обробки або використання множинних неінтегрованих систем.

1.1.1. Концепція та призначення пропонованої системи

Для вирішення вищезазначених проблем запропоновано впровадження системи Configurator. Configurator позиціонується як спеціалізований програмний інструмент, призначений для оптимізації та автоматизації процесів конфігурації продуктів для кінцевих користувачів, а також для централізованого управління даними та бізнес-правилами для адміністраторів системи. Для користувачів-споживачів Configurator забезпечує інтуїтивно зрозумілий та структурований інтерфейс для конфігурації складних комп'ютерних продуктів та систем. Ключові функціональні можливості включають (рис. 1.1):

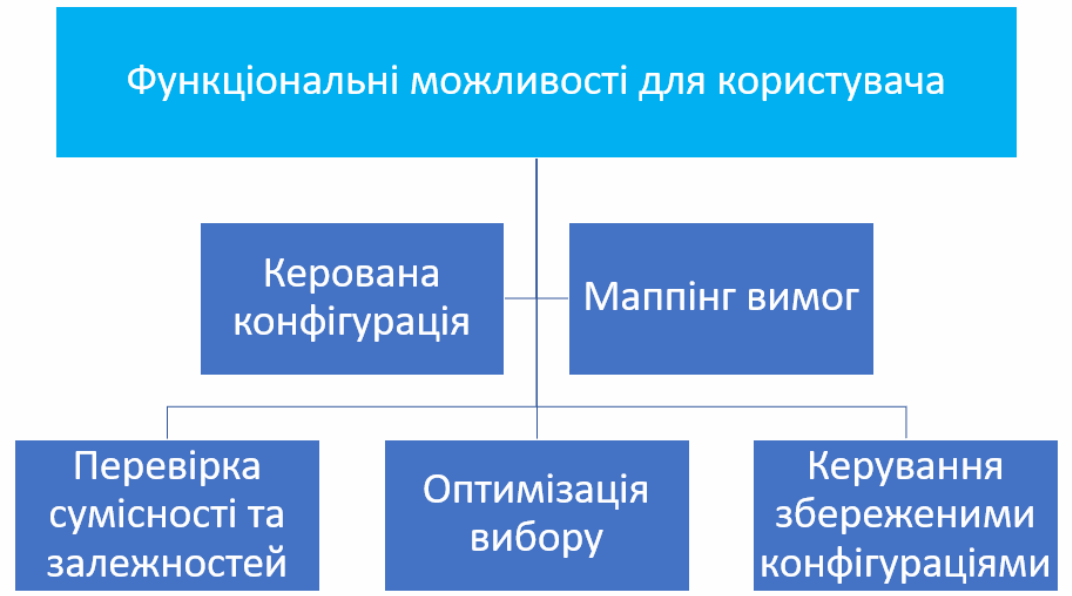

Рисунок 1.1 – Функціональні можливості пропонованої системи з точки зору користувача

Керована конфігурація - система активно збирає вимоги користувача шляхом інтерактивного діалогу або структурованих запитань, що дозволяє точно ідентифікувати потреби.

Маппінг вимог - на основі отриманих вимог Configurator здійснює відображення на релевантний набір продуктів, компонентів та супутніх сервісів, доступних для конфігурації.

Перевірка сумісності та залежностей - вбудовані механізми перевірки автоматично ідентифікують та попереджають користувача про можливі несумісності між обраними компонентами або про необхідність додавання обов'язкових залежних елементів, мінімізуючи ризик помилок при формуванні конфігурації.

Оптимізація вибору - система надає рекомендації та направляє користувача до формування оптимального рішення, яке найкращим чином відповідає його специфічним потребам та бюджетним обмеженням.

Керування збереженими конфігураціями - користувачі мають можливість зберігати незавершені або фінальні конфігурації для подальшого перегляду, редагування або використання.

1.1.2. Функціональні можливості для адміністраторів

З перспективи адміністратора, Configurator виступає як централізована платформа для управління продуктовим каталогом та бізнес-правилами. Функції адміністратора включають (рис. 1.2):

Рисунок 1.1 – Функціональні можливості пропонованої системи з точки зору адміністратора

- Управління продуктовими даними - можливість додавання, редагування та завантаження нових продуктів, їх атрибутів та характеристик.

- Встановлення правил конфігурації - гнучкий інструментарій для визначення правил сумісності, залежностей, обмежень та умов, які регулюють процес конфігурації продуктів для кінцевих користувачів. Це дозволяє формалізувати експертні знання щодо сумісності обладнання.

- Керування користувачами - функціонал для адміністрування облікових записів користувачів системи.

- Інтеграційні можливості - потенціал для інтеграції з іншими корпоративними системами (наприклад, ERP, CRM) для синхронізації даних про продукти, ціни та клієнтів, хоча глибина інтеграції може варіюватися.

1.1.3. Архітектурні переваги

Використання технологій з відкритим вихідним кодом в основі Configurator забезпечує гнучкість, масштабованість та полегшує подальший розвиток системи. Така архітектура сприяє простоті покращення функціоналу, розширення можливостей та інтеграції з новими технологіями та сторонніми сервісами, що робить платформу адаптивною до мінливих вимог ринку.

Важливо зазначити, що Configurator функціонує виключно як інструмент для конфігурації продуктів і не включає функціонал обробки платежів або оформлення замовлень. Результати конфігурації, отримані за допомогою системи, призначені для надання локальним дистриб'юторам або торговим представникам з метою отримання офіційної комерційної пропозиції та подальшого процесу закупівлі. Таким чином, система Configurator оптимізує початкові етапи передпродажної підготовки, фокусуючись на точному визначенні потреб клієнта та формуванні валідної продуктової конфігурації.

1.2. Проєктний огляд системи конфігурації сумісності продуктів

1.2.1. Мотивація розробки

Представлений матеріал описує розробку програмного інструменту, іменованого Configurator, яка була виконана в рамках дипломної роботи. Ініціація проєкту зумовлена досвідом, набутим під час професійної діяльності на посаді інженера-програміста у компанії-виробнику комп'ютерного обладнання. Було виявлено, що значна частина внутрішніх систем таких підприємств інтегрує функціонал конфігураційного двигуна, критично важливого для валідації специфікацій продуктів, сформованих клієнтами для потреб генерації комерційних пропозицій. Поточний проєкт спрямований на моделювання та реалізацію цього аспекту процесу валідації

конфігурацій, а також розширення функціоналу за рахунок включення модуля управління, що надає адміністратору системи засоби для керування продуктовим каталогом, визначення правил конфігурації та адміністрування користувацької інформації.

1.2.2. Цілі проєкту

Основна мета цього проєкту полягає у створенні комплексної програмної платформи. Ця платформа має потенціал до функціонального розвитку та може бути адаптована для використання іншими комп'ютерними реселерами як уніфікований інструмент для управління взаємовідносинами з клієнтами, каталогом продукції та процесами конфігурації комп'ютерного обладнання.

Особисті цілі автора проєкту включали поглиблення практичного досвіду у застосуванні зрілих технологій з відкритим вихідним кодом, вивчення принципів архітектурування програмних додатків, засвоєння методологій відображення функціональних вимог у технічні рішення, а також створення надійної основи для подальшого імплементування додаткових функціональних можливостей шляхом інтеграції новітніх технологічних рішень.

Детальний опис основних функціональних можливостей системи, алгоритмів взаємодії для різних категорій користувачів, обґрунтування вибору застосованих технологій та аспекти реалізації будуть викладені у даній роботі.

1.2.3. Структура рівнів користувачів та функціональні можливості

Розроблена система передбачає наявність двох основних типів користувачів з диференційованими рівнями доступу та функціональними можливостями:

- Користувач-споживач (Consumer User). Має доступ до функціоналу, орієнтованого на процес конфігурації продукту:

- Реєстрація та авторизація в системі.

- Пошук доступних продуктів та компонентів.

- Створення нових конфігурацій продукту.

- Збереження поточних конфігурацій.

- Редагування збережених конфігурацій.

- Видалення конфігурацій.

- Генерація звітів (друк) по окремій конфігурації або списку конфігурацій.

- Користувач-адміністратор (Administrator User). Наділений повноваженнями для управління системою та даними:

- Адміністрування облікових записів користувачів.

- Керування продуктовим каталогом (додавання, редагування, видалення продуктів).

- Завантаження та встановлення правил валідації та сумісності конфігурацій.

1.3. Інструменти розробки

Цей проект був розроблений за допомогою мови програмування Java, фреймворку Spring (включаючи його модулі Spring Core, Spring MVC та Spring Security) та Apache Solr для індексації та пошуку різних продуктів. Для фронтенд-частини додатку я використовував такі технології, як JavaScript, JQuery (для обробки викликів AJAX та анімації HTML-елементів) та Bootstrap для адаптивного дизайну HTML та CSS. У бекенд-частині додатку я використовував Java Server Pages (JSP), контроллери Spring MVC та Hibernate ORM для відображення між об'єктами Java та таблицями бази даних.

1.3.1. Мова J2EE

Java Platform, Enterprise Edition (Java EE) - це мова програмування, яка використовується для розробки корпоративного програмного забезпечення. Java EE дотримується процесу Java Community, який є відкритою спільнотою, що отримує внески від експертів галузі, комерційних та відкритих організацій, груп користувачів Java та багатьох глобальних користувачів. Ця мова програмування постійно додає нові функції, які задовольняють потреби галузі та переносимість додатків у кожному новому випуску.

1.3.2. Hibernate

Hibernate ORM - це фреймворк, який допомагає розробникам створювати додатки, що використовують постійний шар, це означає додатки, дані яких створюються, зберігаються навіть після виконання програми. Фреймворк Hibernate є чудовим інструментом для відображення об'єктів/відносин (ORM), він також надає готові до використання функції, які спрощують реалізацію поширених вимог при взаємодії з базою даних, замість доступу до неї лише через JDBC. Hibernate використовує JDBC внутрішньо та також дозволяє розробнику виконувати нативні SQL-запити, якщо це необхідно.

1.3.3. Spring MVC

Фреймворк Spring Web MVC визначає шаблон Model-View-Controller (MVC), який надає готовий до використання фреймворк, що спрощує реалізацію цього шаблону розробником. Компоненти фреймворку Spring MVC допомагають розробляти дуже гнучкі та слабко зв'язані веб-додатки. Шаблон дизайну MVC розділяє різні аспекти додатку, які включатимуть прийом та обробку введення, виконаного користувачем, бізнес-логіку в бекенді додатку, яка оброблятиме це введення, та повернення результуючого

інтерфейсу користувача (UI). Опис компонентів Spring MVC наведений нижче:

- Модель інкапсулює дані додатку, які складаються з простих старих об'єктів Java (POJO).

- Вид відображає дані моделі та генерує вихід HTML, який може бути інтерпретований браузером клієнта.

- Контролер обробляє запити користувача, створює відповідну модель та передає її Виду для відображення.

Основним елементом фреймворку Spring Web MVC є сервлет DispatcherServlet. Він обробляє всі HTTP-запити та відповіді, які взаємодіють з додатком. Взаємодія між елементами DispatcherServlet фреймворку Spring Web MVC показана на рисунку 1.3.

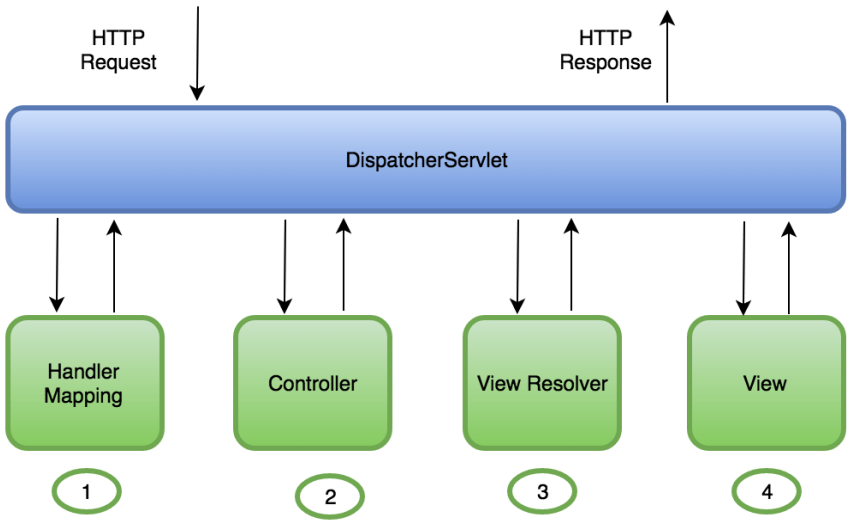

Рисунок 1.3 – Архітектура Spring MVC

Послідовність подій, що відповідають взаємодії вхідного HTTP-запиту з DispatcherServlet, наступна.

Після отримання HTTP-запиту, DispatcherServlet консультується з HandlerMapping, щоб викликати відповідний контролер. Залежно від методу запиту (GET або POST), контролер викликає відповідні методи служби.

Ці методи служби генеруватимуть або отримуватимуть дані моделі та повертатимуть конкретну назву виду DispatcherServlet.

DispatcherServlet покладається на інтерфейс ViewResolver, який відображає назви видів на фактичні види та вибирає визначений вид для запиту.

Після визначення виду ViewResolver, DispatcherServlet передає дані моделі Виду, який нарешті відображається в HTML.

Всі згадані компоненти, такі як HandlerMapping, Controller та ViewResolver, є частинами WebApplicationContext. Це розширення звичайного інтерфейсу ApplicationContext з деякими додатковими функціями, необхідними для веб-додатків.

1.3.4. Spring Security

Spring Security надає дуже широке рішення для безпеки веб-додатків на основі Java. Spring Security надає підтримку шару безпеки додатку, а також різні реалізації, які задовольняють різноманітні вимоги для численних проблемних доменів бізнесу.

Spring Security орієнтований на дві основні області безпеки додатку: "аутентифікація" та "авторизація" (або "контроль доступу"). "Аутентифікація" - це процес перевірки, чи є актор тим, ким він або вона себе називає, а "авторизація" - це процес перевірки, чи може актор виконувати дію в межах додатку. Spring Security підтримує інтеграцію з такими технологіями:

- Аутентифікація HTTP BASIC
- Обмін клієнтськими сертифікатами HTTP X.509
- Аутентифікація LDAP
- Аутентифікація на основі форм (для простих потреб користувацького інтерфейсу)
- Служба аутентифікації та авторизації Java (JAAS)

Готова до використання аутентифікація "запам'ятати мене" (залежно від конфігурації, це уникає повторної аутентифікації користувача протягом визначеного періоду часу). Spring Security дуже налаштовується та може надавати безперебійну інтеграцію з іншими системами, якщо це необхідно.

Елементи, пов'язані з аутентифікацією та авторизацією в Spring Security, описані в просторі імен Spring Security. Основними будівельними блоками Spring Security є такі компоненти:

- UserDetails. Цей об'єкт є реалізацією інтерфейсу UserDetailsService та надає необхідну інформацію постачальнику аутентифікації фреймворку безпеки. Реалізація UserDetails зберігатиме рядок для імені користувача, а також рядок для пароля, і матиме колекцію об'єктів, які називаються GrantedAuthority.

- GrantedAuthority. Цей об'єкт відстежує різні дозволи, надані принципалу.

- AuthenticationManager. Цей інтерфейс отримує авторизації, що містяться в об'єкті UserDetails, і делегуватиме це завдання екземпляру AuthenticationProvider.

- AuthenticationProvider. Реалізація цього інтерфейсу викличе об'єкт, який реалізує інтерфейс UserDetailsService, і поверне повністю заповнений об'єкт UserDetails. Після того, як AuthenticationProvider повернув цей об'єкт, він перевірить, чи є введені облікові дані дійсними, і в веб-орієнтованому додатку, такому як представлений у цьому звіті, будь-яка взаємодія з додатком буде фільтруватися сервлетним фільтром, відомим як SpringSecurityFilterChain, який відповідає за всю безпеку (захист URL-адрес додатку, перевірка введених імен користувачів та паролів, перенаправлення на форму входу тощо).

Налаштування, використане для Spring Security, є конфігурацією простору імен; це XML-базоване та має елемент http, який містить URL-

адресу для перехоплення, форму входу та сторінку успіху або помилки, це можна побачити на рисунку 1.4.

```
<http use-expressions="true" pattern="/jsp/admin**">
    <intercept-url pattern="/jsp/admin*** access="hasRole('ROLE_ADMIN')" />
    <access-denied-handler error-page="/403" />
    <form-login login-processing-url="/j_spring_security_check"
    login-page='/jsp/loginadmin.html'
        authentication-failure-url="/jsp/loginadmin.html?error"
        authentication-success-handler-ref="myAuthenticationSuccessHandler"/>
    <logout/>
    <csrf/>
</http>
```

Рисунок 1.4 - Конфігурація Spring Security

Реалізація Spring Security також запобігає атакам підробки міжсайтових запитів (CSRF), шляхом додавання випадково згенерованого токена, переданого до запиту користувача як параметра HTTP.

1.3.5. Bootstrap

Bootstrap - один з найпоширеніших фреймворків інтерфейсу користувача для CSS, HTML та JavaScript; він безкоштовний та з відкритим вихідним кодом. Bootstrap відомий як один з найкращих фреймворків CSS для створення адаптивних веб-додатків. Bootstrap - це фреймворк, орієнтований на мобільні пристрої; це означає, що все, що ви розробляєте або створюєте, буде сумісним з мобільними пристроями або адаптивним.

Bootstrap вимагає doctype HTML5 для обробки елементів HTML та властивостей CSS. Це має бути включено на початку всіх проектів Bootstrap.

Bootstrap працює з контейнерами, які включають елементи, що використовуються фреймворком. Він використовує лише два типи контейнерів, як показано на рисунку 1.5.

Використовуйте .container для адаптивного контейнера з фіксованою шириною.

```
<div class="container">
</div>
```

Використовуйте .container-fluid для контейнера повної ширини, який охоплює всю ширину вашого вікна перегляду.

```
<div class="container-fluid">
</div>
```

Рисунок 1.5 - Контейнери Bootstrap

Bootstrap також надає велику кількість повторно використовуваних компонентів, включаючи іконографію, випадаючі списки, групи введення, навігацію, сповіщення та багато іншого.

Bootstrap постачається з кількома компонентами JavaScript у вигляді плагінів jQuery. Вони надають додаткові елементи інтерфейсу користувача, такі як діалогові вікна, підказки та каруселі. Вони також розширюють функціональність деяких існуючих елементів інтерфейсу, включаючи функцію автозаповнення для полів введення.

1.3.6. Apache Solr

Solr - це автономний пошуковий движок підприємства з API, подібним до REST, який дозволяє запитувати індексовані дані Solr. Запит цих індексованих даних через HTTP GET створить відповідь у форматі JSON, XML, CSV або бінарному.

Apache Solr розглядає дані як документи та належить до категорії баз даних NoSQL. Apache Solr надає такі функції:

- Розширені можливості повнотекстового пошуку.

- Висока продуктивність для великого обсягу трафіку.

- Він заснований на відкритих інтерфейсах - XML, JSON та HTTP.

- Комплексні інтерфейси адміністрування з веб-консоллю адміністрування.

- Високо масштабований та відмовостійкий.

- Індексація майже в реальному часі за допомогою високопродуктивного текстового пошуковий движок Apache Lucene.

Виконуючи різні запити до індексованих даних, Solr може використовувати такі функції:

- Пагінація та сортування

- Фасетування

- Автозаповнення

- Перевірка орфографії

- Геопросторовий пошук

Приклад синтаксису запиту Solr показано на рисунку 1.6.

```
http://localhost:8983/solr/query?debug=query&q=hello
{
    "responseHeader":{
        "status":0,
        "QTime":0,
        "params":{
            "q": "hello",
            "debug":"query"},
    "response":{"numFound":0,"start":0,"docs":[]
    },
    "debug":{
        "rawquerystring": "hello",
        "querystring": "hello",
        "parsedquery": "text:hello",
        "parsedquery_toString": "text:hello",
        "QParser":"LuceneQParser"}
}
```

Рисунок 1.6 - Синтаксис запиту Solr

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ПРОТОТИПУ ЕКОСИСТЕМИ ДЛЯ РІШЕННЯ КОНФІГУРАЦІЙ СУМІЖНИХ ЗАДАЧ

2.1. Проєктна архітектура системи

Система Configurator реалізована як трирівневий (або багатошаровий) веб-додаток. Архітектура включає наступні ключові компоненти та технології:

- Шар представлення (Frontend) - відповідає за взаємодію з користувачем. Розроблений з використанням технологій JSP (JavaServer Pages) для генерації динамічного вмісту, CSS (Cascading Style Sheets) для стилізації інтерфейсу та бібліотек JavaScript для забезпечення інтерактивності та динамічного відображення даних на стороні клієнта.

- Шар бізнес-логіки (Backend) - реалізує основні функції системи та бізнес-правила. Розроблений з використанням мови програмування Java. Цей шар обробляє запити від клієнтів, виконує операції з даними та застосовує логіку конфігурації.

- Шар доступу до даних (Persistence Layer) - відповідає за взаємодію з базою даних. Реалізований за допомогою фреймворку Hibernate, який надає об'єктно-реляційне відображення (ORM), спрощуючи роботу з даними в об'єктно-орієнтованому стилі.

Середовище виконання - додаток розгорнуто на сервері застосунків Apache Tomcat.

Система управління базами даних (СУБД) - як сховище даних використовується реляційна база даних MySQL. MySQL є одним із найбільш поширених компонентів у веб-розробці, зокрема, як частина програмних стеків LAMP (Linux, Apache, MySQL, PHP/Perl/Python) та LEMP (Linux, Nginx, MySQL, PHP/Perl/Python).

2.2 Опис категорій суб'єктів взаємодії (акторів) в системі

У системі Configurator визначено три основні категорії суб'єктів взаємодії (акторів), кожен з яких має специфічний набір дозволів та функціональних можливостей.

2.2.1. Користувач-гість (Guest User)

Користувач-гість – це неаутентифікований суб'єкт, який вперше взаємодіє з системою. Набір доступних функціональних можливостей для цієї категорії користувачів обмежений операціями читання: перегляд доступних категорій продуктів та пошук конкретних товарних позицій у каталозі. Можливість створення або збереження конфігурацій для користувачів-гостей відсутня. Проте, передбачена функціональність для реєстрації нового облікового запису. Діаграма варіантів використання для користувача-гостя представлена на рисунку 2.1.

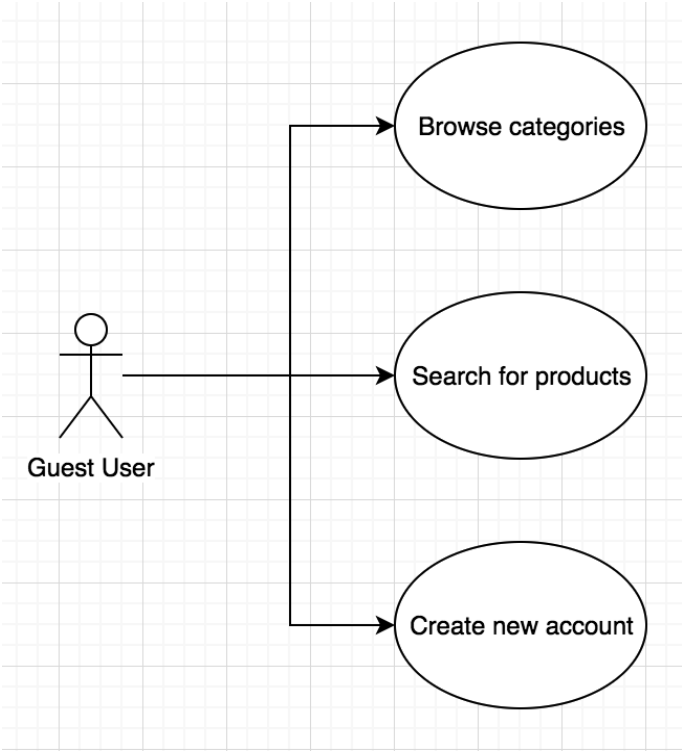

Рисунок 2.1 - Варіанти використання для користувача-гостя

2.2.2. Авторизований користувач (Logged-in User)

Авторизований користувач – це суб'єкт, який пройшов процедуру аутентифікації в системі. Ця категорія користувачів успадковує права Користувача-гостя щодо перегляду та пошуку продуктів. Додатково, Авторизований користувач має розширені можливості для управління конфігураціями продуктів, включаючи їх створення, редагування та видалення. Також доступний функціонал для генерації звітів: звіту, що містить список усіх збережених конфігурацій, та деталізованого звіту по конкретній конфігурації. Діаграма варіантів використання для авторизованого користувача представлена на рисунку 2.2.

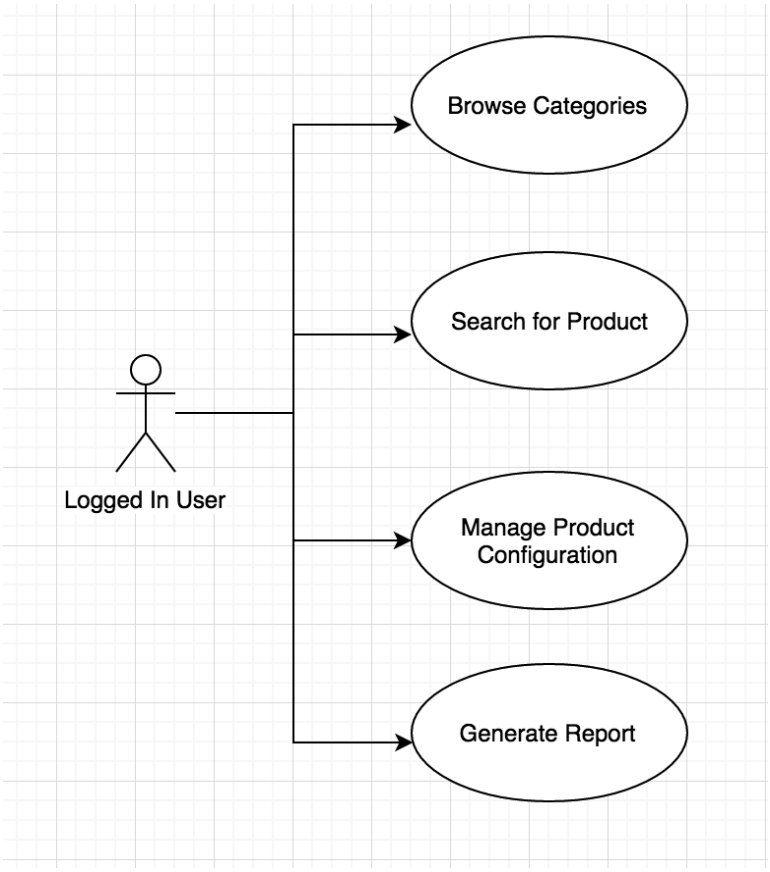

Рисунок 2.2 - Варіанти використання для авторизованого користувача

2.2.3. Користувач-адміністратор (Administrator User)

Користувач-адміністратор – це привілейований суб'єкт, який має повний доступ до функцій управління системою. До його можливостей

належить адміністрування облікових записів користувачів та повне керування продуктовим каталогом. Адміністратор може додавати нові продукти до системи як шляхом індивідуального введення даних, так і пакетним завантаженням з файлу формату .csv. Ключовою функцією є можливість встановлення правил конфігурації продуктів, що реалізується шляхом завантаження відповідного файлу .csv. Цей файл містить визначені обмеження та правила, що регулюють сумісність та допустимість комбінування окремих компонентів у рамках певної продуктової конфігурації. Крім того, Користувач-адміністратор може здійснювати операції редагування та видалення користувачів і продуктів, а також виконувати різноманітні запити пошуку для отримання системної інформації. Діаграма варіантів використання для користувача-адміністратора представлена на рисунку 2.3.

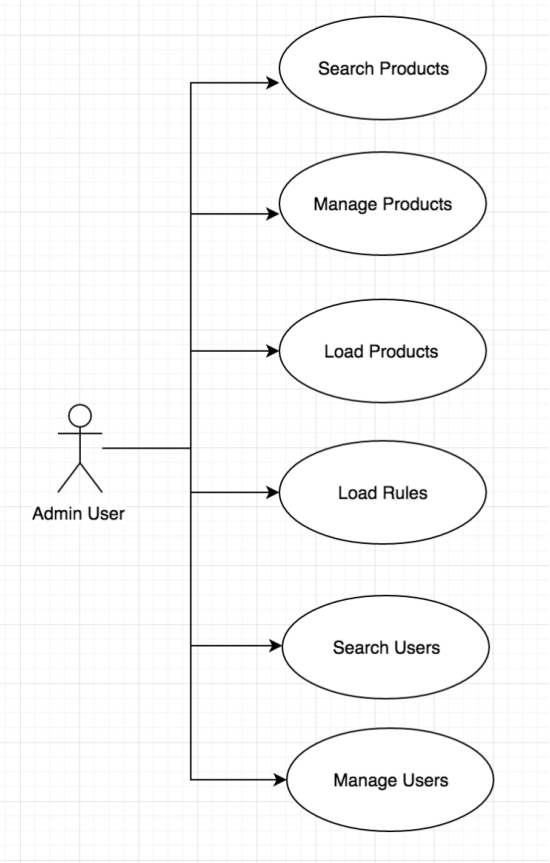

Рисунок 2.3 - Варіанти використання для користувача-адміністратора

2.3. Шаблони проектування

При розробці архітектури даного програмного застосунку було застосовано низку стандартних шаблонів проєктування, які сприяють структуризації коду, розділенню відповідальностей та полегшують подальшу підтримку та розвиток системи. Зокрема, використані такі шаблони:

- Шаблон MVC (Model-View-Controller)
- Шаблон DAO (Data Access Object)
- Шаблон Factory (Фабрика)

2.3.1 Шаблон MVC (Model-View-Controller)

Шаблон MVC є архітектурним шаблоном, який ефективно реалізує принцип розділення відповідальностей у програмних застосунках, особливо у тих, що мають користувацький інтерфейс. Він поділяє додаток на три взаємопов'язані компоненти:

Модель (Model) - цей компонент представляє дані та бізнес-логіку застосунку. Він містить об'єкти даних (часто у вигляді POJO - Plain Old Java Objects) та відповідає за керування станом даних. За необхідності, Модель може ініціювати сповіщення Контролера про зміни у своєму стані.

Вид (View) - відповідає за візуальне представлення даних, що надаються Моделлю. У даному застосунку роль Виду виконують JSP (JavaServer Pages), які генерують HTML-сторінку для відображення користувачеві. Вид отримує дані від Моделі (зазвичай через Контролер) і відображає їх, не містячи бізнес-логіки.

Контролер (Controller) - діє як посередник між Моделлю та Видом. Він приймає запити від користувача (через Вид), обробляє їх, взаємодіючи з Моделлю для отримання або модифікації даних, і обирає відповідний Вид для відображення результату користувачеві. Контролер "контролює" потік виконання та логіку взаємодії компонентів.

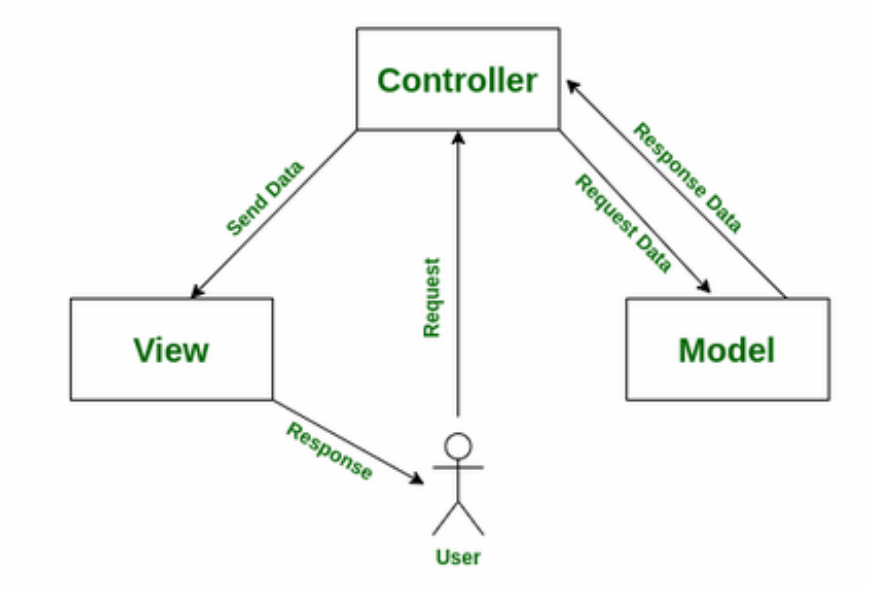

Рисунок 2.4 – Шаблон MVC

2.3.2 Шаблон DAO (Data Access Object)

Шаблон проєктування DAO (Об'єкт доступу до даних) реалізований у цьому проєкті з метою створення абстрактного рівня для операцій доступу до сховища даних (бази даних). Основне призначення цього шаблону – інкапсулювати специфічну логіку взаємодії з постійним шаром та приховати деталі реалізації конкретної СУБД або механізму доступу від бізнес-логіки застосунку.

Застосування DAO базується на принципах абстракції (визначення інтерфейсу для операцій доступу) та інкапсуляції (приховування деталей реалізації цього доступу). Це дозволяє змінювати технологію доступу до даних або СУБД без суттєвих змін у бізнес-логіці.

Для подальшого розділення відповідальностей та керування транзакціями, між шаром бізнес-логіки та шаром DAO запроваджено додатковий рівень Сервісів (Service Layer). Цей рівень використовує об'єкти DAO для виконання складніших операцій з постійними об'єктами, агрегуючи виклики до одного або кількох DAO. Структура реалізації шаблону DAO у цьому проєкті може бути проілюстрована на рисунку 2.5.

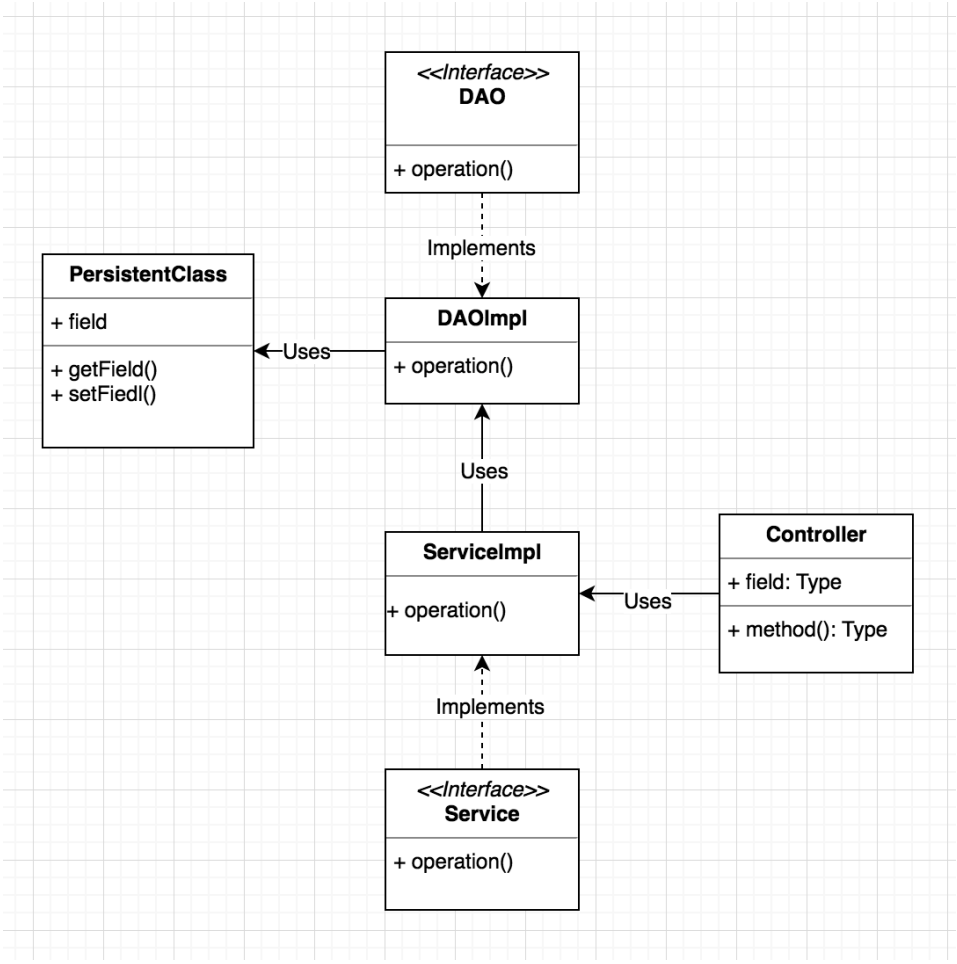

Рисунок 2.5 – Шаблон проектування DAO

2.3.3. Шаблон Factory (Фабрика)

Шаблон проєктування Factory, який належить до категорії породжувальних шаблонів, використовується для абстрагування процесу створення об'єктів. У даному застосунку цей шаблон реалізований опосередковано через функціонал контейнера інверсії управління (IoC) фреймворку Spring (зокрема, його основного модуля).

Контейнер Spring, діючи як фабрика об'єктів (так званих "beans"), керує життєвим циклом компонентів застосунку. Створення екземплярів класів та управління їхніми залежностями повністю делегується контейнеру і не розкривається клієнтському коду. Код, що потребує екземпляр об'єкта, отримує його через спільний інтерфейс, такий як ApplicationContext або BeanFactory у Spring. Ці інтерфейси, хоч і мають деякі відмінності, обидва

надають доступ до визначених "beans" та реалізують механізм впровадження залежностей (Dependency Injection - DI), автоматично надаючи об'єктам необхідні їм залежності. Такий підхід зменшує зв'язаність між компонентами та спрощує конфігурацію застосунку.

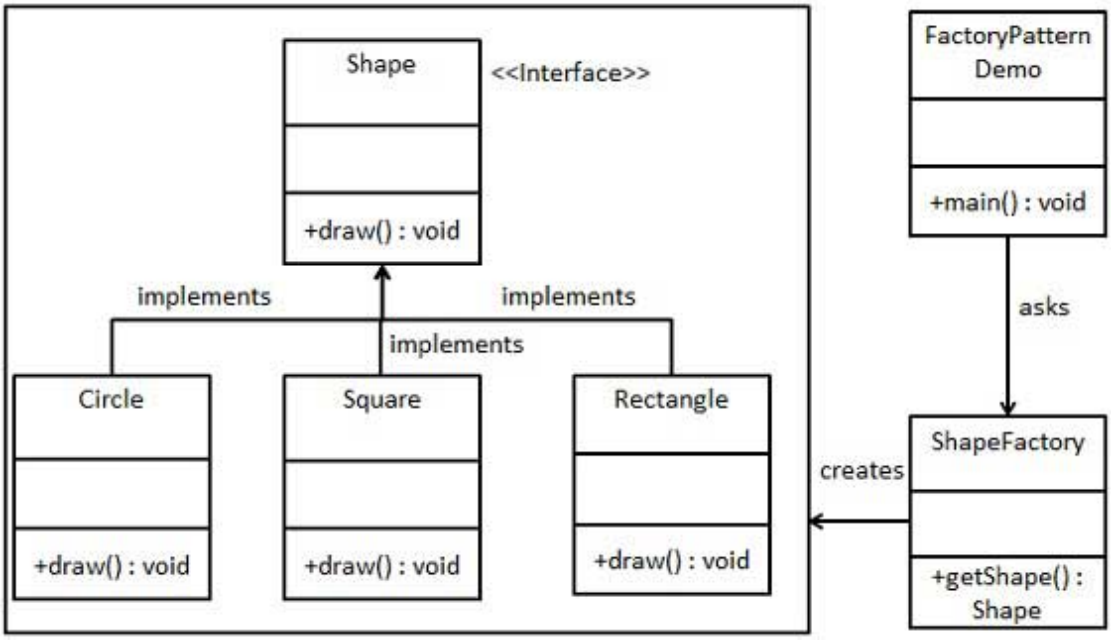

Рисунок 2.6 - Структура шаблону проєктування Фабрика (Factory Pattern)

На рисунку 2.6 зображено діаграму класів UML, яка ілюструє структуру шаблону проєктування Фабрика (Factory Pattern).

Ця діаграма демонструє, як можна абстрагувати процес створення об'єктів, делегуючи його спеціалізованому "фабричному" класу. Розглянемо ключові елементи діаграми:

- Shape (Інтерфейс). Це базовий інтерфейс (позначений <<Interface>>), який визначає загальний контракт для всіх об'єктів форм. У цьому випадку, це метод +draw() : void, який мають реалізувати всі конкретні форми.

- Circle, Square, Rectangle (Конкретні Класи). Це конкретні реалізації інтерфейсу Shape. Кожен з цих класів надає власну реалізацію методу

+draw(). Стрілки з пунктирною лінією та порожнім трикутником вказують на те, що ці класи імплементують (implement) інтерфейс Shape.

- ShapeFactory (Фабрика). Це клас, який виступає в ролі Фабрики. Його основна функція – створювати (creates) об'єкти типів, що імплементують інтерфейс Shape. Клас має публічний метод +getShape() : Shape, який, ймовірно, приймає якийсь параметр (на діаграмі не показано, але зазвичай це тип форми, яку потрібно створити) і повертає об'єкт типу Shape (тобто екземпляр Circle, Square або Rectangle). Важливо, що цей метод повертає саме інтерфейс Shape, приховуючи конкретний клас реалізації від клієнта.

- FactoryPatternDemo (Клієнт). Це клас, який використовує (asks) Фабрику для отримання об'єктів. У його методі +main() : void (точка входу програми) відбувається звернення до ShapeFactory для створення необхідних об'єктів форм. Клієнтський код взаємодіє лише з Фабрикою та отриманими об'єктами через їхній інтерфейс Shape, не знаючи і не створюючи безпосередньо екземпляри конкретних класів Circle, Square або Rectangle.

Таким чином, діаграма ілюструє:

- Абстракцію спільної функціональності через інтерфейс (Shape).

- Реалізацію цієї функціональності у конкретних класах (Circle, Square, Rectangle).

- Виділення логіки створення об'єктів у окремий клас-фабрику (ShapeFactory).

- Взаємодію клієнтського коду (FactoryPatternDemo) з фабрикою та інтерфейсом, що забезпечує гнучкість та незалежність клієнта від конкретних класів продуктів.

Це дозволяє легко додавати нові типи форм (наприклад, Triangle), створивши новий клас, що імплементує Shape, та додавши логіку його створення у ShapeFactory, без необхідності змінювати код клієнта (FactoryPatternDemo).

2.4. Об'єктно-реляційне відображення та метадані

Як було зазначено у розділі 2.1, для реалізації рівня постійності даних (persistence layer) у даному проєкті застосовується фреймворк Hibernate ORM (Object-Relational Mapping). Основна функція ORM полягає у встановленні зв'язку (відображення) між об'єктно-орієнтованою моделлю даних, представленою класами Java (так званими "сутностями" або "entities"), та реляційною моделлю даних, представленою таблицями в базі даних SQL.

Hibernate надає потужний механізм відображення метаданих SQL-таблиць на Java-об'єкти переважно за допомогою анотацій, розміщених безпосередньо у вихідному коді Java-класів сутностей. Такий підхід дозволяє декларативно визначати структуру таблиць, типи даних стовпців, зв'язки між таблицями (один до одного, один до багатьох, багато до багатьох) та інші властивості відображення. Використання анотацій значно спрощує процес розробки та полегшує розуміння відповідності між об'єктною моделлю та схемою бази даних.

```
<bean id="hibernate4AnnotatedSessionFactory" class="org.springframework.orm.hibernate4.L
    <property name="dataSource" ref="dataSource" />
    <property name="annotatedClasses">
        <list>
            <value>com.epic.ecommerce.core.model.Category</value>
            <value>com.epic.ecommerce.core.model.Country</value>
            <value>com.epic.ecommerce.core.model.Customer</value>
            <value>com.epic.ecommerce.core.model.Product</value>
            <value>com.epic.ecommerce.core.model.PartValidation</value>
            <value>com.epic.ecommerce.core.model.Menu</value>
            <value>com.epic.ecommerce.core.model.User</value>
            <value>com.epic.ecommerce.core.model.UserRole</value>
            <value>com.epic.ecommerce.core.model.ConfigurationProduct</value>
            <value>com.epic.ecommerce.core.model.QuoteConfiguration</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
</bean>
```

Рисунок 2.7 - Конфігурація моделі та відображення об'єктних відносин

Набір сутностей (класів Java), які відображаються на відповідні таблиці бази даних у цьому проєкті, представлений у конфігурації, що ілюструється на рисунку 2.7. До цих сутностей належать: Category, Country, Customer, Product, PartValidation, Menu, User, UserRole, ConfigurationProduct, QuoteConfiguration.

Приклад конфігурації фабрики сесій Hibernate (SessionFactory), інтегрованої з фреймворком Spring, що демонструє реєстрацію анотованих класів сутностей та налаштування властивостей Hibernate, наведено вище.

2.5. Розробка та опис моделі бази даних

Модель даних для програмного застосунку Configurator була розроблена та реалізована з використанням системи управління реляційними базами даних (СУБД) MySQL. Схема бази даних структурно організована для підтримки функціональності системи конфігурації та управління продукцією.

База даних системи Configurator складається з наступних взаємопов'язаних таблиць, кожна з яких представляє певну сутність у предметній області:

- CATEGORY
- CONFIGURATION_PRODUCT
- COUNTRY
- CUSTOMER
- MENU
- PART_PRODUCT
- PART_VALIDATION
- PRODUCT
- QUOTE_CONFIGURATION
- SUBCAT_CAT

- USER_ROLES

- USERS

Комплексна суб'єктно-зв'язкова діаграма (ER-діаграма) для схеми бази даних Configurator, що візуалізує сутності (таблиці) та зв'язки між ними, представлена на рисунку 2.8. Нижче наведено детальний опис структури та призначення деяких ключових таблиць.

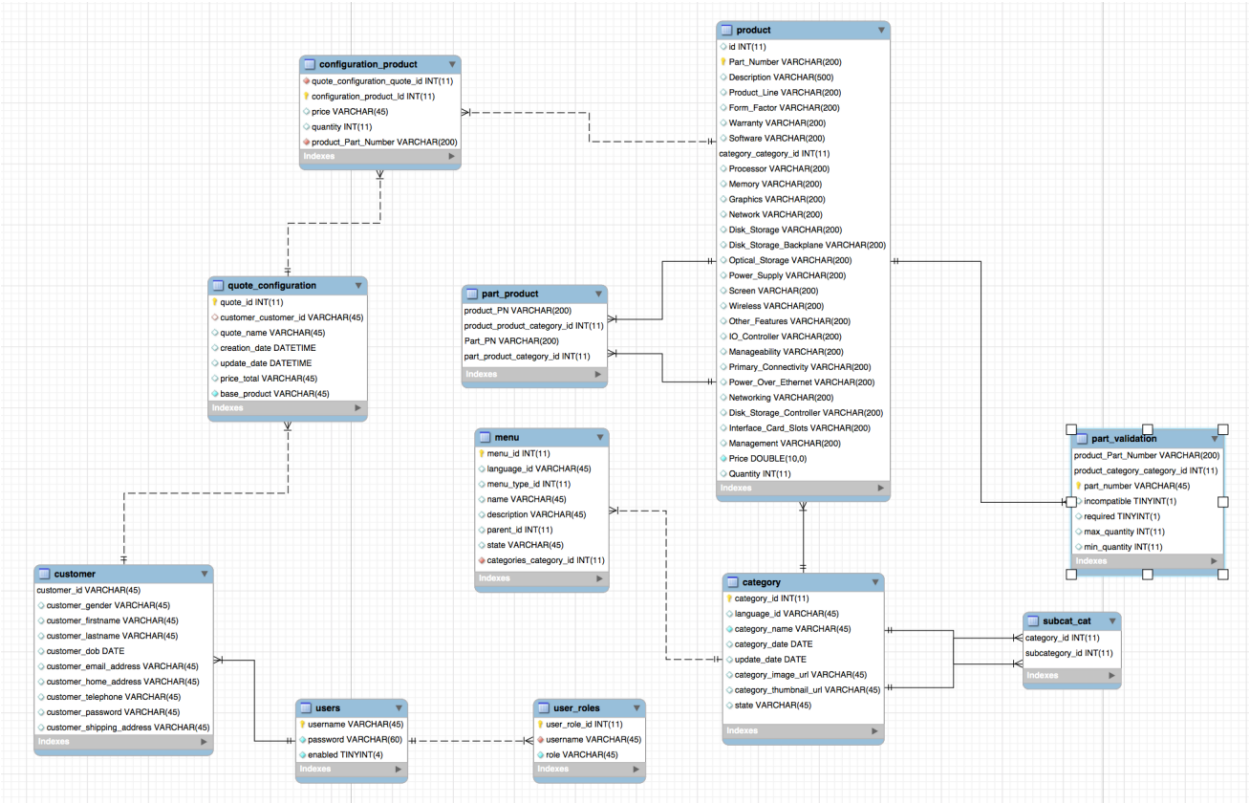

Рисунок 2.8 – ER діаграма

Таблиця CATEGORY призначена для зберігання інформації про категорії продуктів, які використовуються у процесі конфігурації. Кожен продукт пов'язаний з певною категорією (зв'язок "багато до одного" з таблицею PRODUCT). Таблиця CATEGORY також моделює ієрархічні зв'язки між категоріями (підкатегорії), які визначені у допоміжній таблиці SUBCAT_CAT, реалізуючи, по суті, самопосилальний зв'язок типу "багато до багатьох". Структура таблиці CATEGORY представлена у таблиці 2.1.

Таблиця 2.1 - Структура таблиці CATEGORY

CATEGORY			
CATEGORY_ID	INT(11)	NOT NULL AUTO_INCREMENT	PK
LANGUAGE_ID	VARCHAR(45)	NULL DEFAULT NULL	
CATEGORY_NAME	VARCHAR(45)	NOT NULL	
CATEGORY_DATE	DATE	DATE NULL DEFAULT NULL,	
UPDATE_DATE	DATE	DATE NULL DEFAULT NULL,	
CATEGORY_IMAGE_URL	VARCHAR(45)	NULL DEFAULT NULL	
CATEGORY_THUMBNAIL_URL	VARCHAR(45)	NULL DEFAULT NULL	
STATE	VARCHAR(45)	NULL DEFAULT NULL	

Таблиця SUBCAT_CAT функціонує як зв'язуюча таблиця для реалізації зв'язку типу "багато до багатьох" між записами в таблиці CATEGORY самою із собою. Вона визначає, які категорії є підкатегоріями інших категорій. Цей зв'язок забезпечує ієрархічну структуру каталогу категорій. Структура таблиці SUBCAT_CAT представлена у таблиці 2.2. Візуалізація цього зв'язку наведена на рисунку 2.9.

Таблиця 3.2 - Структура таблиці SUBCAT_CAT

SUBCAT_CAT			
CATEGORY_ID	INT(11)	NOT NULL	FK
SUBCATEGORY_ID	INT(11)	NOT NULL	FK

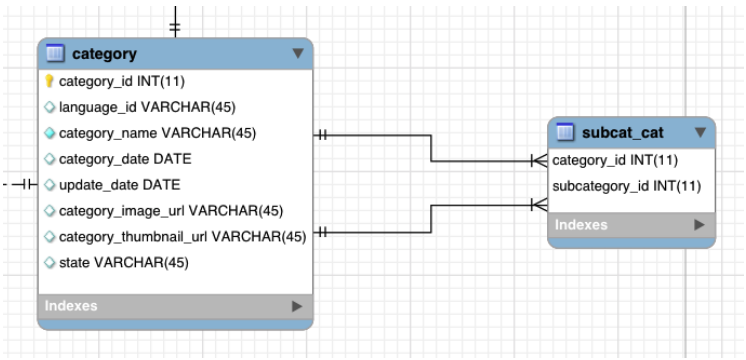

Рисунок 2.9 - Зв'язок «багато до багатьох» таблиці CATEGORY із самою собою

Таблиця MENU зберігає дані про різні меню, які можуть бути пов'язані з категоріями продуктів. Поле MENU_TYPE_ID може використовуватися для класифікації типів меню. Вона має зв'язок "один до багатьох" з таблицею CATEGORY через поле CATEGORIES_CATEGORY_ID, що дозволяє одній категорії мати кілька пов'язаних меню. Структура таблиці MENU представлена у таблиці 2.3. Візуалізація зв'язку таблиці MENU з таблицею CATEGORY показана на рисунку 2.10.

Таблиця 2.3 - Структура таблиці MENU

MENU			
MENU_ID	INT(11)	NOT NULL AUTO_INCREMENT	PK
LANGUAGE_ID	VARCHAR(45)	NULL DEFAULT NULL	
MENU_TYPE_ID	INT(11)	NULL DEFAULT NULL	
NAME	VARCHAR(45)	NULL DEFAULT NULL	
DESCRIPTION	VARCHAR(45)	NULL DEFAULT NULL	
PARENT_ID	INT(11)	NULL DEFAULT NULL	
STATE	VARCHAR(45)	NULL DEFAULT NULL	
CATEGORIES_CATEGORY_ID	INT(11)	NULL DEFAULT NULL	FK

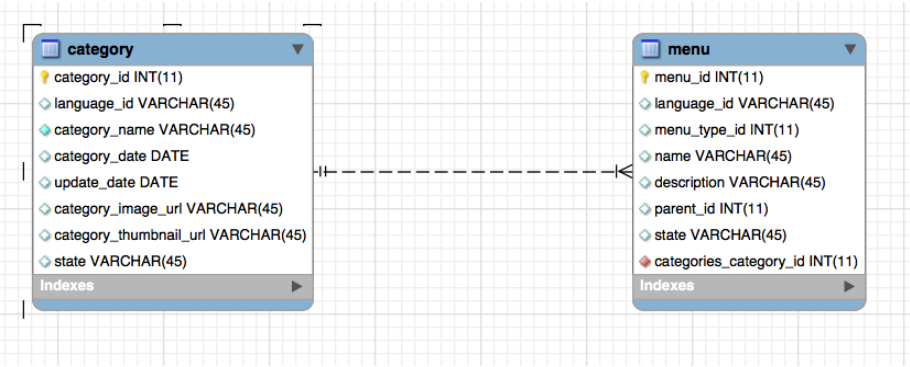

Рисунок 2.10 - Відношення таблиць CATEGORY та MENU

Таблиця PRODUCT є центральною сутністю у моделі даних, що містить вичерпну інформацію про окремі продукти (компоненти, пристрої тощо), доступні для конфігурації. Вона має наступні ключові зв'язки:

- Зв'язок "багато до одного" з таблицею CATEGORY (поле CATEGORY_CATEGORY_ID).

- Зв'язок "багато до багатьох" із самою собою, реалізований через зв'язуючу таблицю PART_PRODUCT.

- Зв'язок "один до багатьох" з таблицею PART_VALIDATION, що визначає правила сумісності та обмежень для частин.

- Зв'язок "один до багатьох" з таблицею CONFIGURATION_PRODUCT, який вказує, які продукти включені до конкретної конфігурації.

Структура таблиці PRODUCT, що містить численні атрибути, які описують технічні характеристики та властивості продукту, представлена у таблиці 2.4. Візуалізація зв'язків таблиці PRODUCT показана на рисунку 2.11.

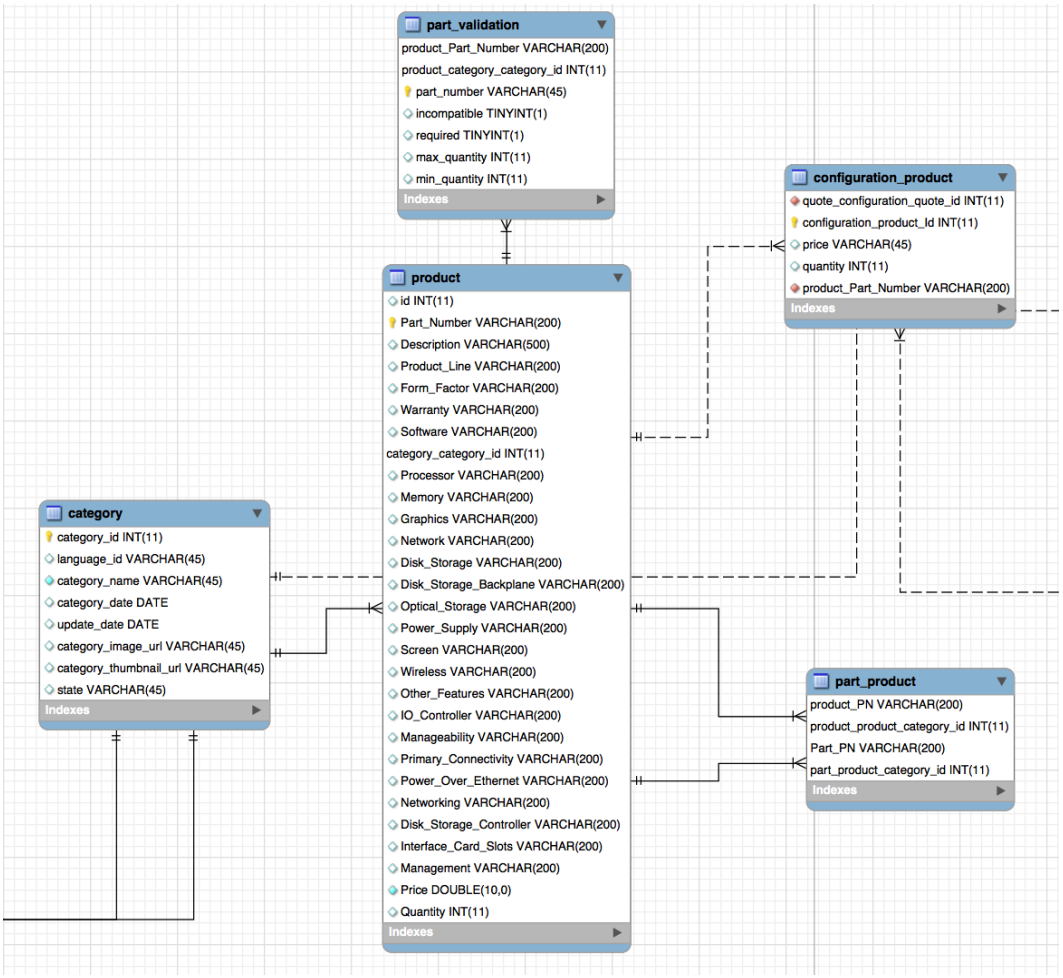

Рисунок 2.11 - Відношення таблиці PRODUCT

Таблиця 2.4 - Структура таблиці PRODUCT

PRODUCT			
ID	INT(11)	NOT NULL AUTO_INCREMENT	PK
PART_NUMBER	VARCHAR(45)	NULL DEFAULT NULL	
DESCRIPTION	VARCHAR(45)	NULL DEFAULT NULL	
PRODUCT_LINE	VARCHAR(45)	NULL DEFAULT NULL	
FORM_FACTOR	VARCHAR(45)	NULL DEFAULT NULL	
WARRANTY	DATE	NULL DEFAULT NULL	
SOFTWARE	VARCHAR(45)	NULL DEFAULT NULL	
CATEGORY_CATEGORY_ID	INT(11)	NOT NULL	FK
PROCESSOR	VARCHAR(45)	NULL DEFAULT NULL	
MEMORY	VARCHAR(200)	NULL DEFAULT NULL	
GRAPHICS	VARCHAR(200)	NULL DEFAULT NULL	
NETWORK	VARCHAR(200)	NULL DEFAULT NULL	
DISK_STORAGE	VARCHAR(200)	NULL DEFAULT NULL	
DISK_STORAGE_BACKPLANE	VARCHAR(200)	NULL DEFAULT NULL	
OPTICAL_STORAGE	VARCHAR(200)	NULL DEFAULT NULL	
POWER_SUPPLY	VARCHAR(200)	NULL DEFAULT NULL	
SCREEN	VARCHAR(200)	NULL DEFAULT NULL	
WIRELESS	VARCHAR(200)	NULL DEFAULT NULL	
OTHER_FEATURES	VARCHAR(200)	NULL DEFAULT NULL	
IO_CONTROLLER	VARCHAR(200)	NULL DEFAULT NULL	
MANAGEABILITY	VARCHAR(200)	NULL DEFAULT NULL	
PRIMARY_CONNECTIVITY	VARCHAR(200)	NULL DEFAULT NULL	
POWER_OVER_ETHERNET	VARCHAR(200)	NULL DEFAULT NULL	
NETWORKING	VARCHAR(200)	NULL DEFAULT NULL	
DISK_STORAGE_CONTROLLER	VARCHAR(200)	NULL DEFAULT NULL	
INTERFACE_CARD_SLOTS	VARCHAR(200)	NULL DEFAULT NULL	
MANAGEMENT	VARCHAR(200)	NULL DEFAULT NULL	
PRICE	DOUBLE(10,0)	NOT NULL DEFAULT '0'	
QUANTITY	INT(11)	NULL DEFAULT NULL	

Таблиця PART_PRODUCT є допоміжною зв'язуючою таблицею, що моделює самопосилальний зв'язок "багато до багатьох" в межах сутності PRODUCT. Вона фіксує відношення між головним продуктом (PRODUCT

PN, PRODUCT_PRODUCT_CATEGORY_ID) та його складовими частинами (PART PN, PART_PRODUCT_CATEGORY_ID), де кожна частина також є продуктом з таблиці PRODUCT. Ця таблиця дозволяє визначати ієрархію або комплектацію продуктів. Структура таблиці PART_PRODUCT представлена у таблиці 2.5. Візуалізація цього зв'язку типу "багато до багатьох" показана на рисунку 2.12.

Таблиця 2.5 - Структура таблиці PART_PRODUCT

PART_PRODUCT			
PRODUCT_PN	VARCHAR(200)	NOT NULL	FK
PRODUCT_PRODUCT_CATEGORY_ID	INT(11)	NOT NULL	FK
PART_PN	VARCHAR(200)	NOT NULL	FK
PART_PRODUCT_CATEGORY_ID	INT(11)	NOT NULL	FK

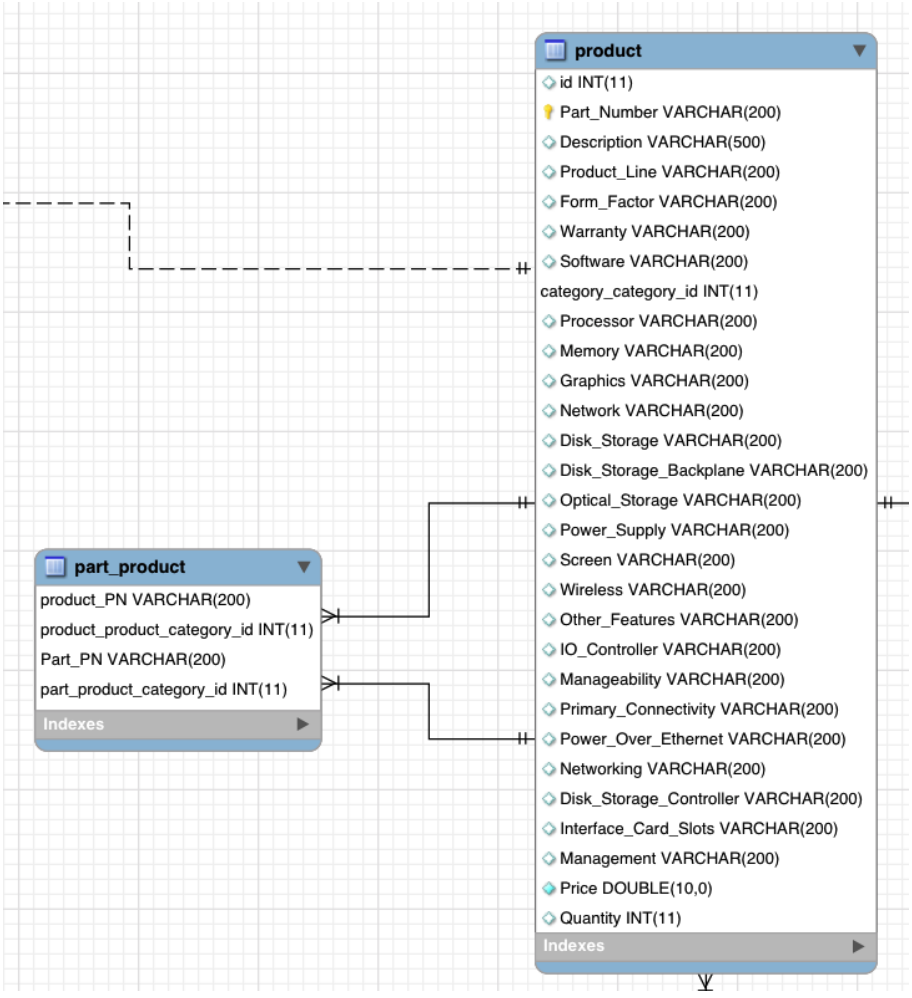

Рисунок 2.12 - Зв'язок «багато до багатьох» таблиці PRODUCT

Таблиця PART_VALIDATION містить набір правил валідації та обмежень для конфігурації продуктів. Вона визначає сумісність та допустиму кількість певних частин (PART_NUMBER) у контексті конфігурації основного продукту (PRODUCT_PART_NUMBER). Атрибути INCOMPATIBLE та REQUIRED вказують на несумісність або обов'язковість частини, а MAX_QUANTITY та MIN_QUANTITY задають діапазон допустимої кількості даної частини у конфігурації. Таблиця має зв'язок "багато до одного" з таблицею PRODUCT (за полями, що ідентифікують продукт та частину). Структура таблиці PART_VALIDATION представлена у таблиці 2.6.

Таблиця 2.6 - Структура таблиці PART_VALIDATION

PART_VALIDATION			
PRODUCT_PART_NUMBER	VARCHAR(200)	NOT NULL	PK
PRODUCT_CATEGORY_CATEGORY_ID	INT(11)	NOT NULL	FK
PART_NUMBER	VARCHAR(45)	NOT NULL	FK
INCOMPATIBLE	TINYINT(1)	NULL DEFAULT NULL	
REQUIRED	TINYINT(1)	NULL DEFAULT NULL	
MAX_QUANTITY	INT(11)	NULL DEFAULT NULL	
MIN _QUANTITY	INT(11)	NULL DEFAULT NULL	

Таблиця QUOTE_CONFIGURATION призначена для зберігання даних про сформовані комерційні пропозиції (котирування) на основі створених конфігурацій продуктів. Вона має зв'язок "один до одного" (або "один до багатьох" в залежності від інтерпретації структури) з таблицею CONFIGURATION_PRODUCT та зв'язок "багато до одного" з таблицею

CUSTOMER, вказуючи, для якого клієнта була створена дана пропозиція. Структура таблиці QUOTE_CONFIGURATION представлена у таблиці 2.7.

Таблиця 2.7 - Структура таблиці QUOTE_CONFIGURATION

CONFIGURATION_PRODUCT			
QUOTE_CONFIGURATION_QUOTE_ID	INT(11)	NOT NULL AUTO_INCREMENT	FK
CONFIGURATION_PRODUCT_ID	INT(11)	NULL DEFAULT NULL	PK
PRICE	VARCHAR(45)	NULL DEFAULT NULL	
QUANTITY	INT(11)	NULL DEFAULT NULL	
PRODUCT_PART_NUMBER	VARCHAR(200)	NOT NULL	FK

Таблиця CONFIGURATION_PRODUCT зберігає деталізацію кожної конкретної конфігурації продукту, перелічуючи складові частини (продукти), що входять до її складу. Вона має зв'язок "багато до одного" з таблицею PRODUCT (для кожної частини) та зв'язок "багато до одного" з таблицею QUOTE_CONFIGURATION, вказуючи, до якої пропозиції належить дана конфігурація. Структура таблиці CONFIGURATION_PRODUCT представлена у таблиці 2.8.

Таблиця 2.8 - Структура таблиці CONFIGURATION_PRODUCT

QUOTE_CONFIGURATION			
QUOTE_ID	INT(11)	NOT NULL AUTO_INCREMENT	PK
CUSTOMER_CUSTOMER_ID	VARCHAR(45)	NULL DEFAULT NULL	FK
QUOTE_NAME	VARCHAR(45)	NULL DEFAULT NULL	
CREATION_DATE	DATETIME	NULL DEFAULT NULL	
UPDATE_DATE	DATETIME	NULL DEFAULT NULL	
PRICE_TOTAL	VARCHAR(45)	NULL DEFAULT NULL	
BASE_PRODUCT	VARCHAR(45)	NOT NULL	

Таблиця CUSTOMER містить атрибути, що описують кінцевих користувачів (клієнтів) системи, які можуть створювати та зберігати конфігурації. Вона має зв'язок "один до одного" з таблицею USERS, оскільки кожен клієнт є аутентифікованим користувачем системи. Структура таблиці CUSTOMER представлена у таблиці 2.9.

Таблиця 2.9 - Структура таблиці CUSTOMER

CUSTOMER			
CUSTOMER_ID	VARCHAR(45)	NOT NULL	FK
CUSTOMER_GENDER	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_FIRSTNAME	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_LASTNAME	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_DOB	DATE	NULL DEFAULT NULL	
CUSTOMER_EMAIL_ADDRESS	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_HOME_ADDRESS	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_TELEPHONE	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_PASSWORD	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_SHIPPING_ADDRESS	VARCHAR(45)	NULL DEFAULT NULL	

Таблиця USERS призначена для аутентифікації користувачів системи. Вона зберігає облікові дані, такі як ім'я користувача та хеш пароля (хоча в наданій схемі вказано просто VARCHAR, що потребує уточнення щодо безпеки), а також статус активності облікового запису (ENABLED). Таблиця

має зв'язок "один до одного" з таблицею CUSTOMER (для клієнтів) та зв'язок "один до багатьох" з таблицею USER_ROLES, що визначає ролі користувача. Структура таблиці USERS представлена у таблиці 2.10.

Таблиця 2.10 - Структура таблиці USERS

USERS			
USERNAME	VARCHAR(45)	NOT NULL AUTO_INCREMENT	PK
PASSWORD	VARCHAR(45)	NOT NULL	
ENABLED	TINYINT(4)	NOT NULL DEFAULT '1'	

Таблиця USER_ROLES реалізує зв'язок "один до багатьох" між користувачами та їхніми ролями в системі, дозволяючи одному користувачеві мати кілька ролей. Вона зберігає ідентифікатор ролі, ім'я користувача, до якого належить роль, та назву ролі (наприклад, 'USER', 'ADMIN'). Структура таблиці USER_ROLES представлена у таблиці 2.11.

Таблиця 2.11 - Структура таблиці USER_ROLES

USER_ROLES			
USER_ROLE_ID	INT(11)	NOT NULL	PK
USERNAME	VARCHAR(45)	NOT NULL	FK
ROLE	VARCHAR(45)	NOT NULL	

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОТОТИПУ ЕКОСИСТЕМИ ДЛЯ РІШЕННЯ КОНФІГУРАЦІЙ СУМІЖНИХ ЗАДАЧ

Система Configurator розроблена як веб-додаток, доступ до якого здійснюється через стандартні веб-браузери. Сумісність додатку забезпечується з основними сучасними браузерами, включаючи Google Chrome, Mozilla Firefox, що гарантує доступність системи на різних операційних платформах.

Архітектура застосунку базується на розділенні на клієнтську та серверну частини.

Клієнтська частина (Frontend). Інтерфейс користувача реалізовано з використанням технології JSP (JavaServer Pages) для генерації динамічного HTML-вмісту. Для забезпечення інтерактивності та асинхронної взаємодії із сервером (AJAX-виклики) застосовується бібліотека JavaScript JQuery. Адаптивний дизайн інтерфейсу, що забезпечує коректне відображення на пристроях з різними розмірами екранів, реалізовано за допомогою CSS-фреймворку Bootstrap.

Серверна частина (Backend). Бізнес-логіка та обробка даних реалізовані мовою програмування Java.

3.1. Реалізація профілю адміністратора

Функціональність управління системою з боку адміністратора згрупована у спеціалізованому профілі. Доступ до модулів адміністрування надається виключно користувачам, які пройшли процедуру аутентифікації та мають присвоєну роль ADMIN.

Доступ до адміністративних модулів починається зі сторінки аутентифікації. Ця сторінка призначена для перевірки облікових даних користувача та визначення його прав доступу до функцій управління. Доступ

до сторінки аутентифікації адміністратора здійснюється за визначеною URL-адресою, наприклад: http://localhost:8080/jsp/admin.html. Процес аутентифікації показано на рисунку 3.1.

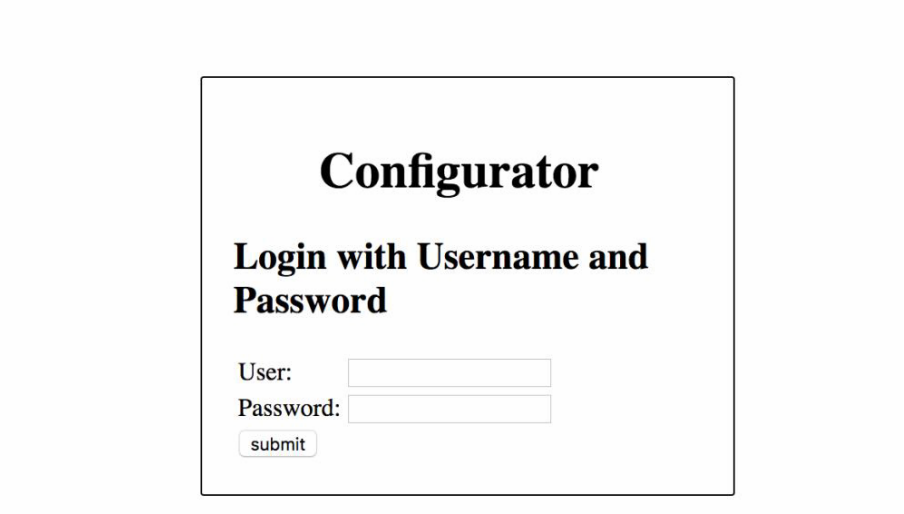

Рисунок 3.1 – Сторінка аутентифікації адміністратора

У випадку введення користувачем невірних облікових даних (логіна або пароля) система відображає повідомлення про помилку, інформуючи користувача про необхідність введення коректної комбінації імені користувача та пароля для отримання доступу. Вигляд сторінки помилки аутентифікації представлено на рисунку 3.2.

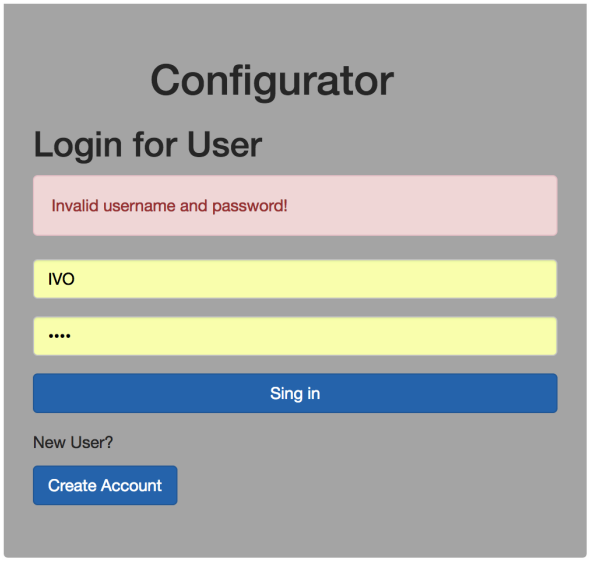

Рисунок 3.2 - Сторінка помилки входу адміністратора

Після успішної аутентифікації користувач з правами ADMIN отримує доступ до головної сторінки адміністратора. Ця сторінка слугує єдиною точкою доступу до різних адміністративних модулів: управління продуктами, управління користувачами, управління ціноутворенням та управління правилами конфігурації. Особливістю реалізації є асинхронне завантаження вмісту кожного вибраного розділу за допомогою технології AJAX, що дозволяє оновлювати лише частину сторінки без повного її перезавантаження. Структура головної сторінки адміністратора проілюстрована на рисунку 3.3.

Рисунок 3.3 – Головна сторінка адміністратора

Модуль управління продуктами надає адміністратору вичерпний набір інструментів для виконання операцій над сутностями продуктового каталогу. Функціональність включає: пошук, перегляд, редагування, видалення та створення нових продуктів. За замовчуванням, при переході до цього модуля відображається повний перелік усіх продуктів, наявних у системі. Інтерфейс модуля управління продуктами показано на рисунку 3.4.

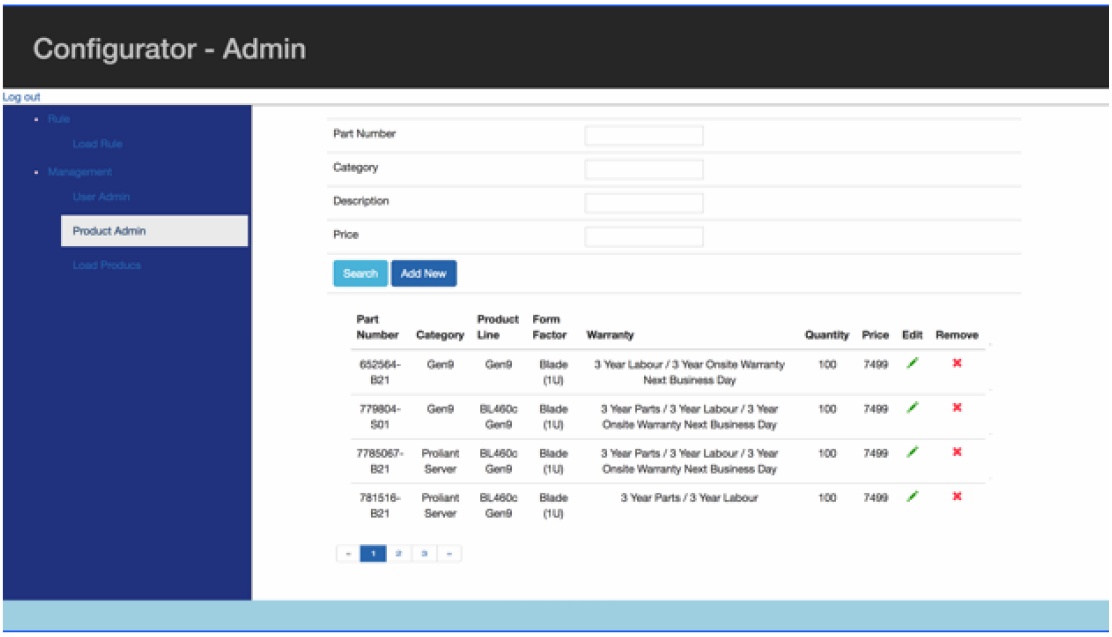

Рисунок 3.4 - Сторінка управління продуктами

Адміністратор має можливість здійснювати пошук конкретних продуктів, використовуючи різноманітні критерії фільтрації даних. Пошук може виконуватись за номером частини продукту, належністю до певної категорії, фрагментом опису або ціною продукту. Приклади інтерфейсу пошуку за різними критеріями представлені на рисунках 3.5, 3.6, 3.7 та 3.8 відповідно.

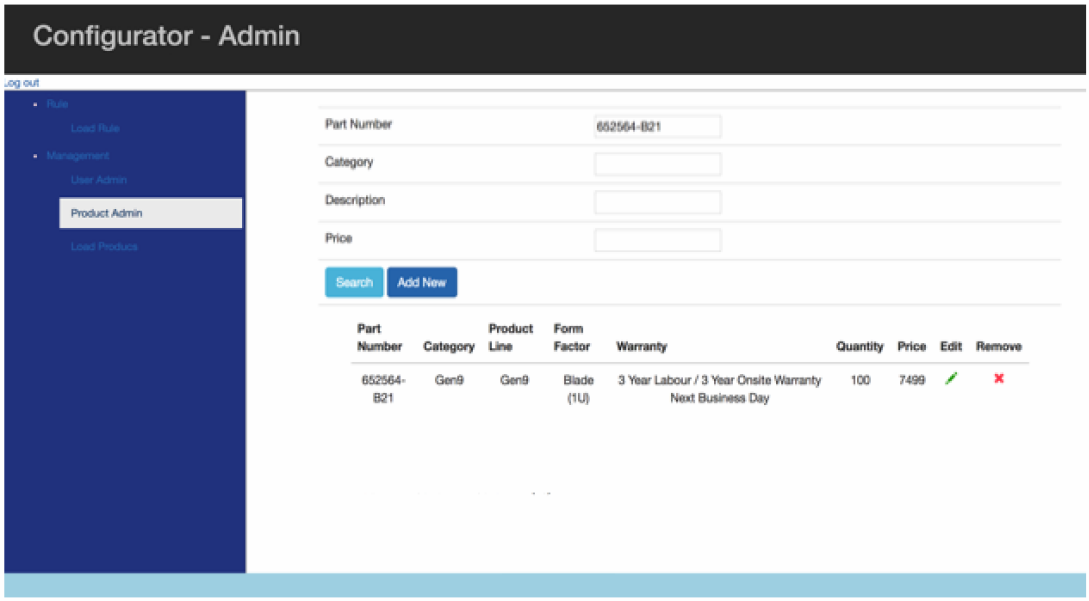

Рисунок 3.5 - Пошук продуктів за номером частини

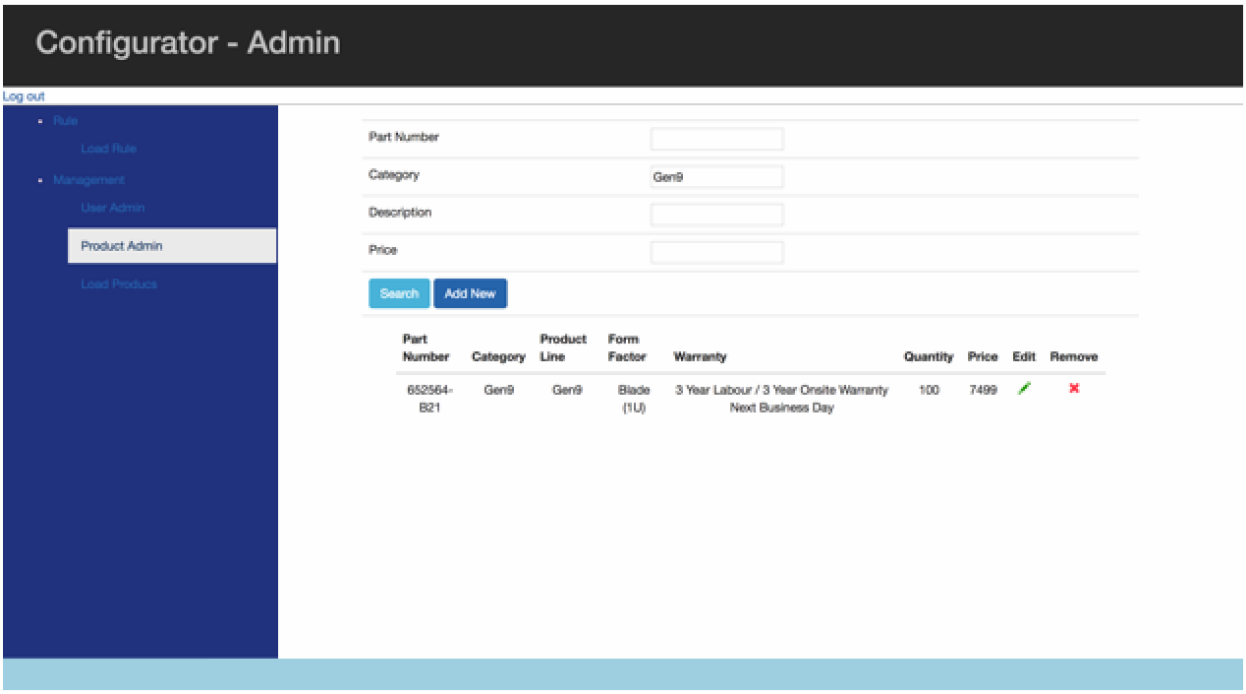

Рисунок 3.6 - Пошук продуктів за категорією

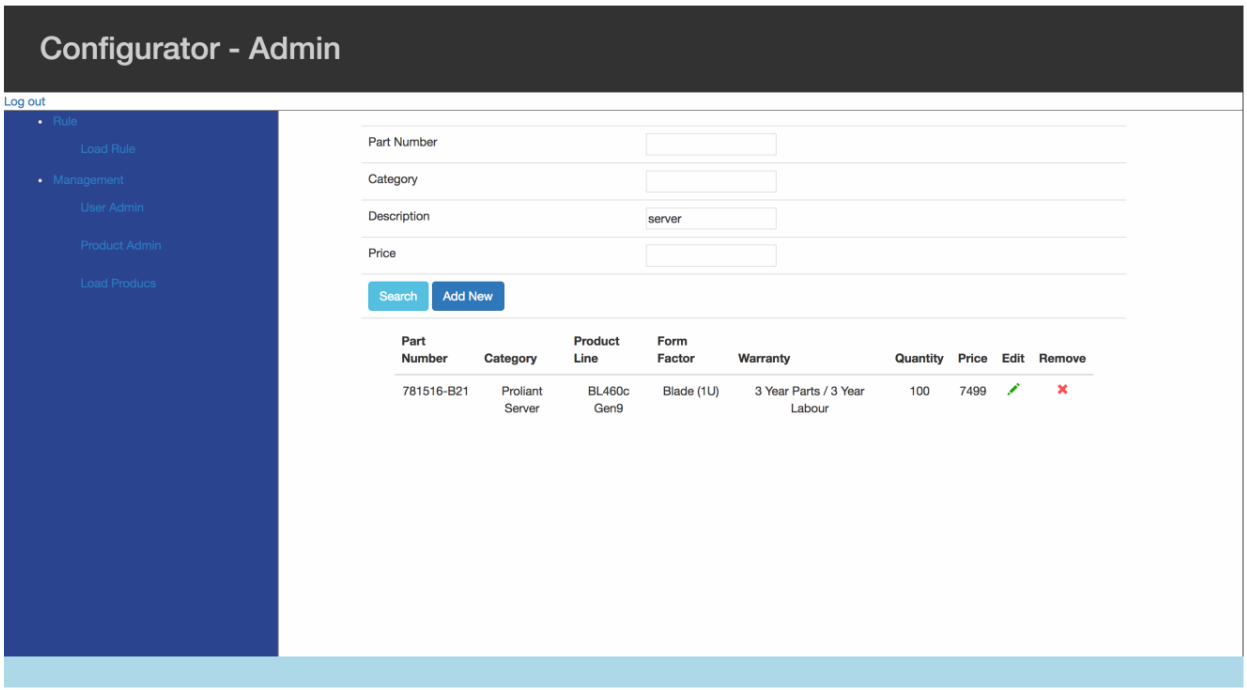

Рисунок 3.7 - Пошук продуктів за описом

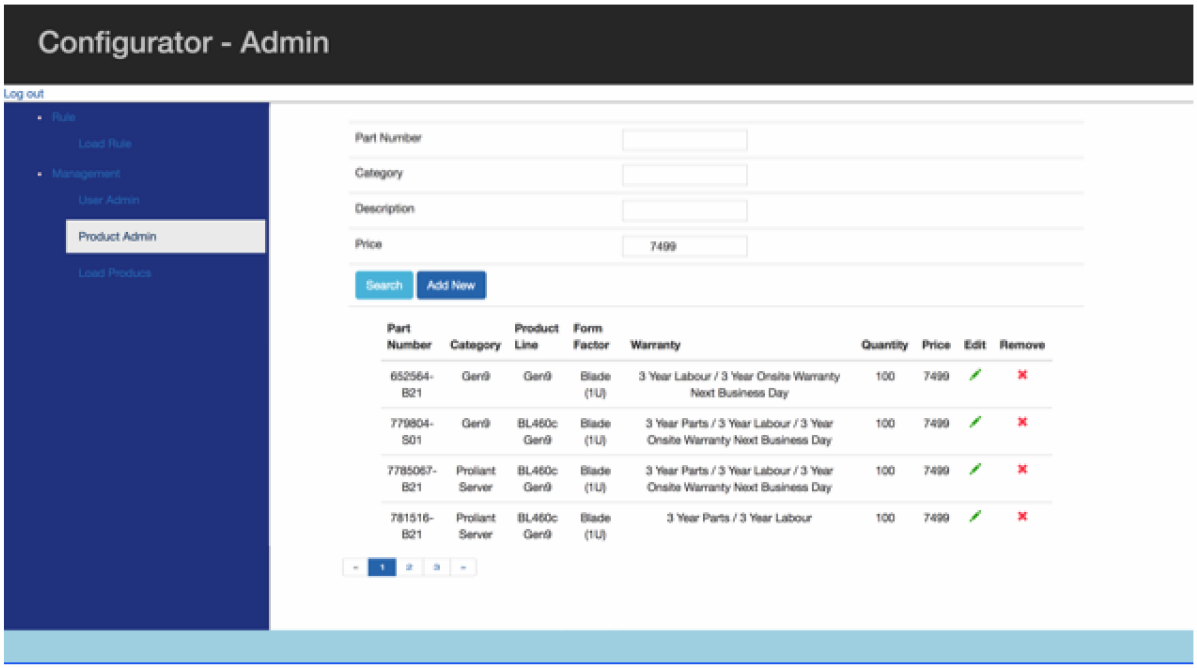

Рисунок 3.8 - Пошук продуктів за ціною

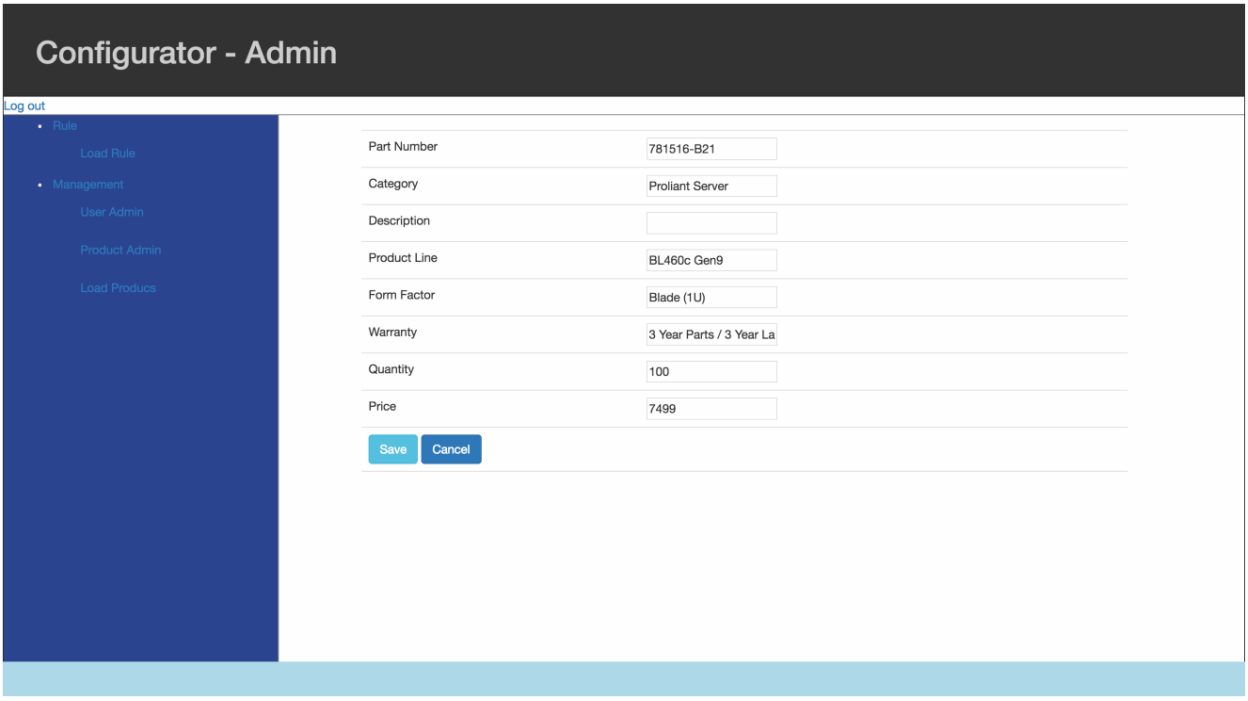

Рисунок 3.9 - Сторінка редагування продукту

Для модифікації існуючих записів продуктів адміністратор може ініціювати процедуру редагування з таблиці результатів пошуку або загального списку продуктів. Система перенаправляє користувача на спеціалізовану сторінку редагування продукту, де доступні поля для

оновлення відповідних атрибутів обраної сутності. Після внесення змін адміністратор може зберегти модифікації або скасувати операцію, що супроводжується поверненням до модуля управління продуктами. Інтерфейс сторінки редагування продукту показано на рисунку 3.9.

Адміністратор має можливість додавати нові товарні позиції до каталогу системи. У модулі управління продуктами передбачено опцію "Додати новий продукт". Активація цієї опції відкриває відповідну сторінку, де адміністратор заповнює атрибути нового продукту. Після введення даних можливе збереження нового запису або скасування операції. Інтерфейс сторінки додавання продукту представлено на рисунку 3.10.

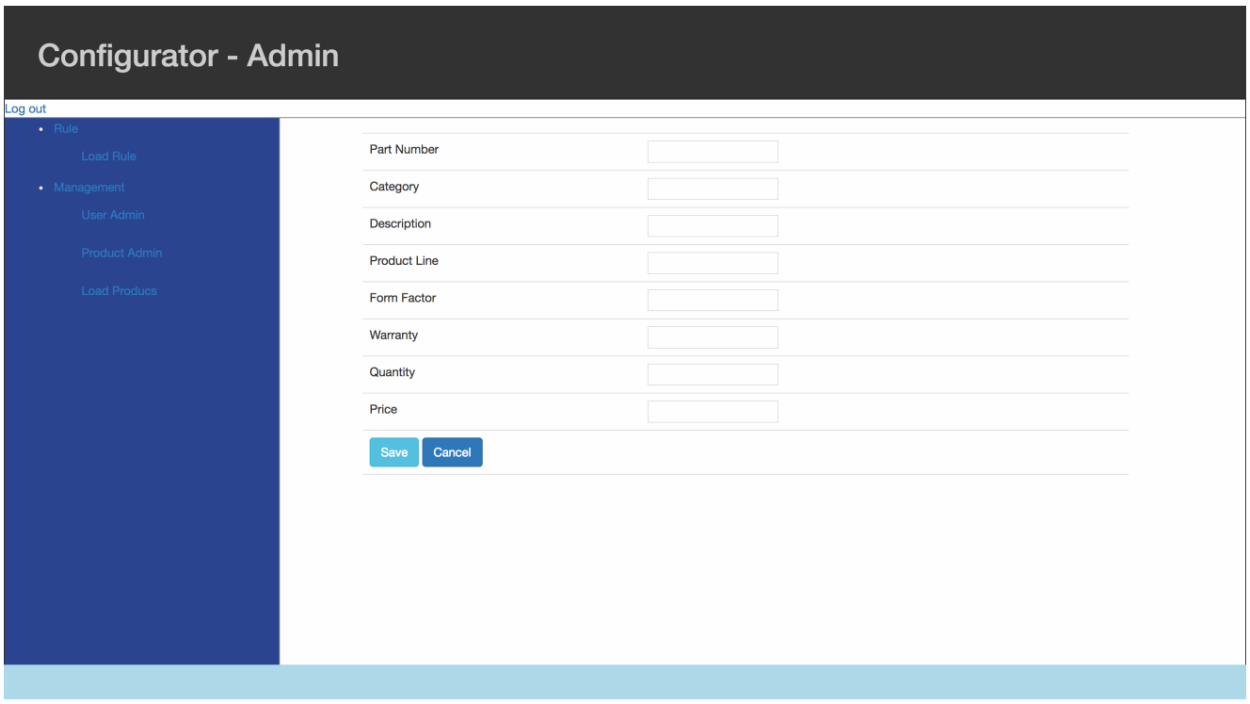

Рисунок 3.10 – Сторінка додавання нового продукту

Система надає адміністратору можливість видаляти продукти з каталогу. У модулі управління продуктами поруч з кожним записом продукту доступна опція ініціації процедури видалення (зазвичай у вигляді іконки). При виборі цієї опції система запитує підтвердження дії для запобігання випадковому видаленню даних. У разі підтвердження запис про

продукт видаляється з бази даних та зникає зі списку відображених продуктів. Приклад інтерфейсу видалення продукту показано на рисунку 3.11.

Рисунок 3.11 – Опція видалення продукту

3.2. Реалізація модуля управління користувачами в системі

Модуль управління користувачами надає адміністратору системи необхідні інструменти для повного життєвого циклу керування обліковими записами користувачів, зареєстрованих у системі. До основних функцій цього модуля належать: пошук користувачів, перегляд їхніх атрибутів, модифікація даних облікових записів, видалення користувачів та створення нових. За замовчуванням, при переході до цього модуля відображається вичерпний список усіх користувачів системи. Інтерфейс модуля управління користувачами представлено на рисунку 3.12.

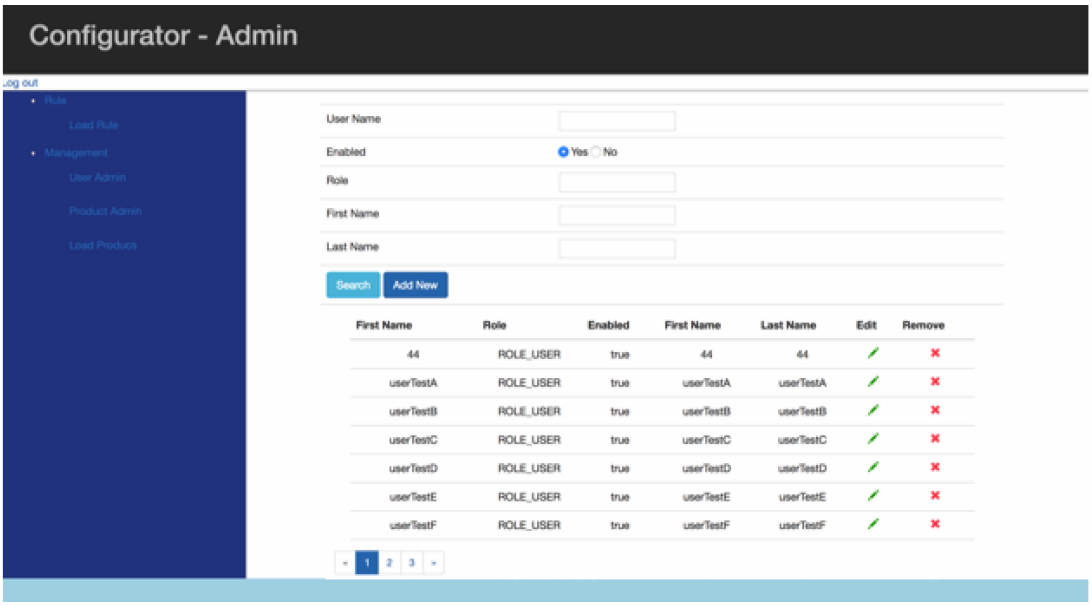

Рисунок 3.12 – Сторінка управління користувачами

Адміністратор має можливість здійснювати цілеспрямований пошук конкретних користувачів, використовуючи різноманітні критерії. Пошук може бути виконаний за одним або комбінацією таких полів: ім'я користувача (Username), статус облікового запису (активовано/деактивовано - Enabled/Disabled), присвоєна роль (Role), ім'я (First Name) або прізвище (Last Name) користувача. Приклади інтерфейсів пошуку за різними критеріями показані на рисунках 3.13 - 3.17.

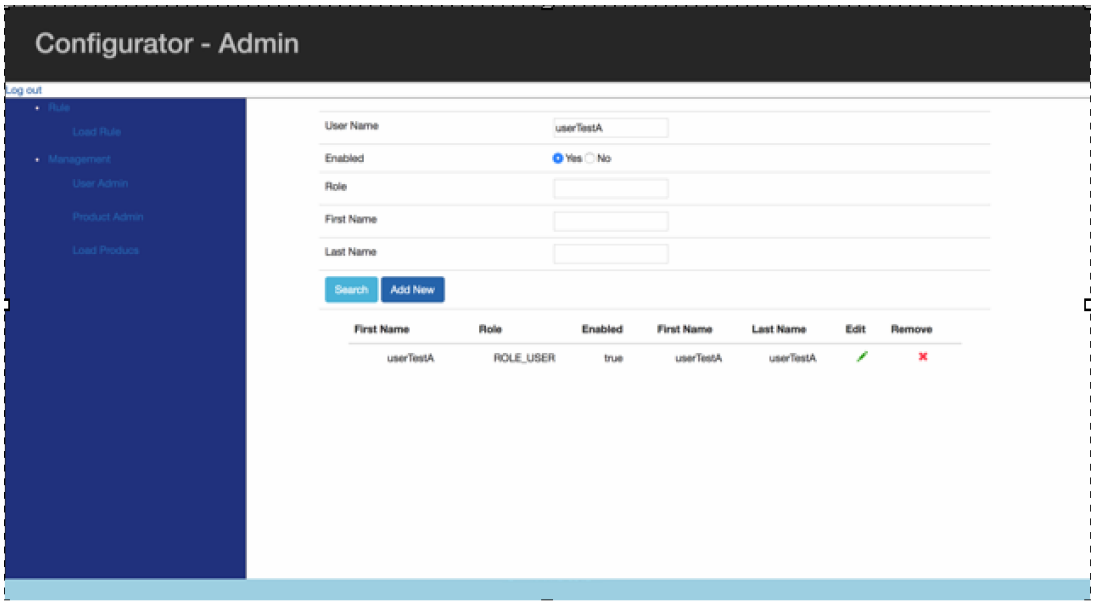

Рисунок 3.13 - Пошук користувачів за іменем користувача

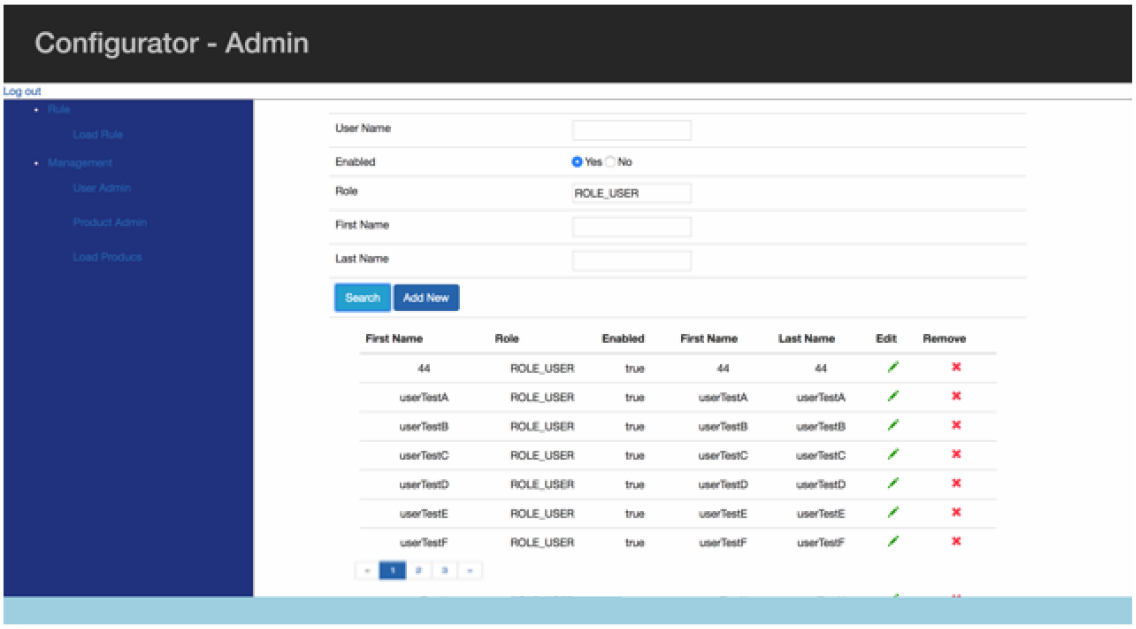

Рисунок 3.14 - Пошук користувачів за роллю

Приклад пошуку користувачів за статусом та іменем показано на рисунках 3.15 і 3.16 відповідно.

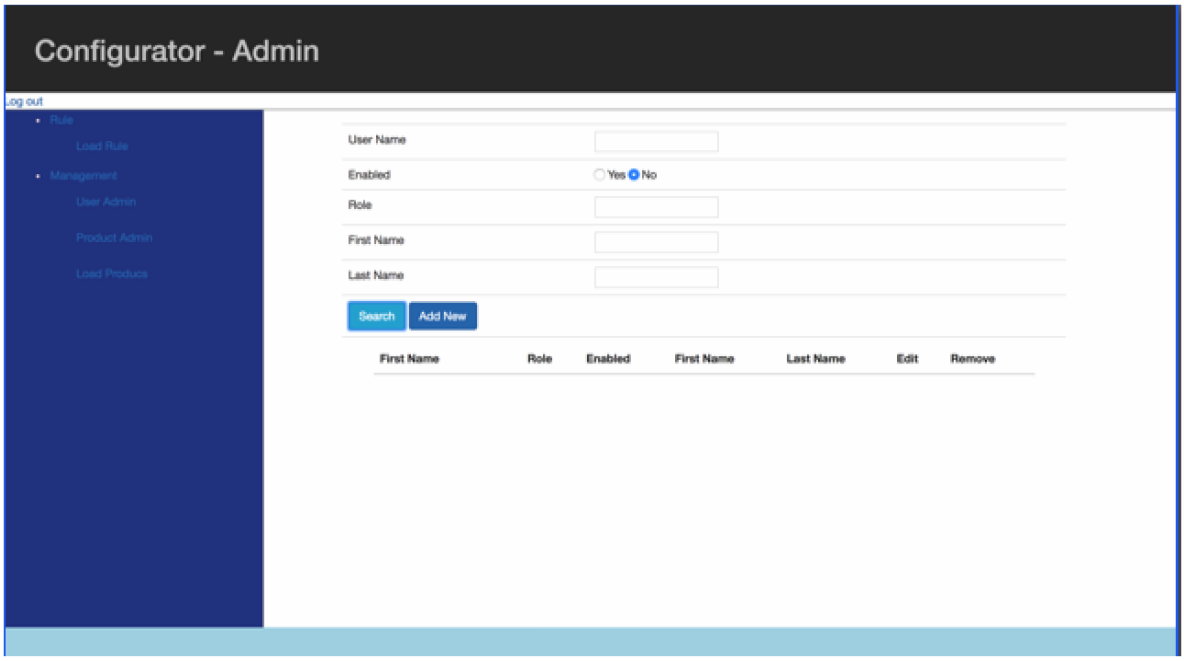

Рисунок 3.15 - Пошук користувачів за статусом (активовано/деактивовано)

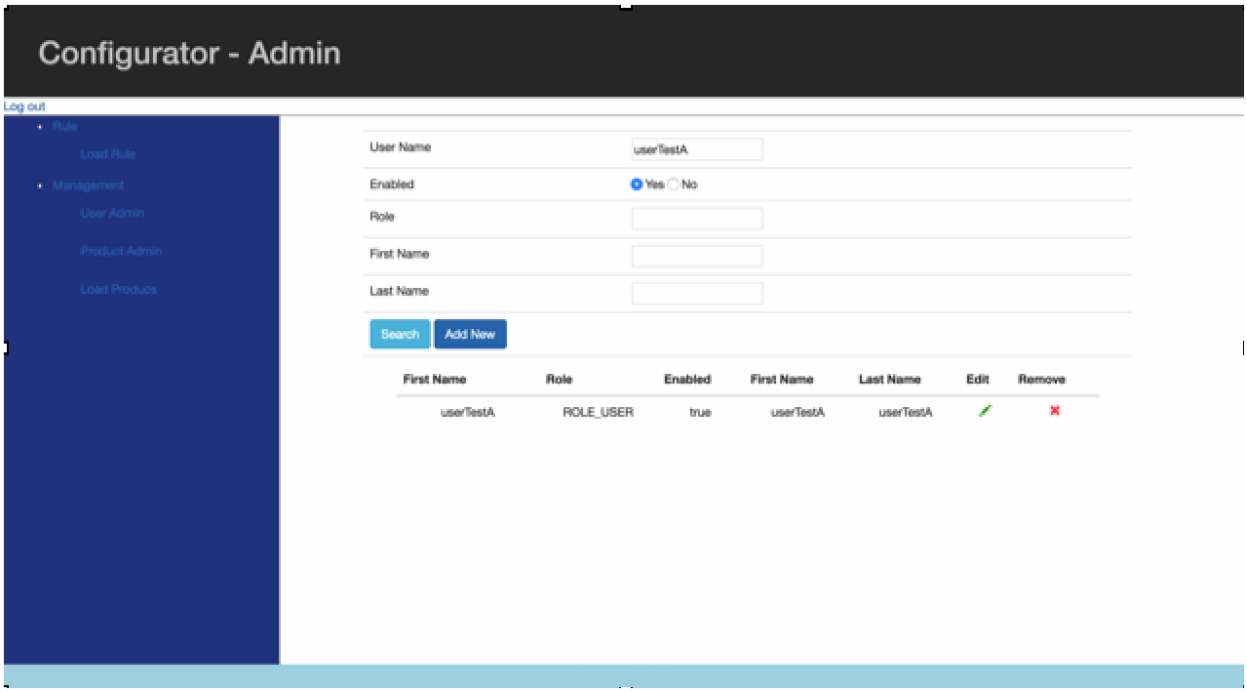

Рисунок 3.16 - Пошук користувачів за ім'ям

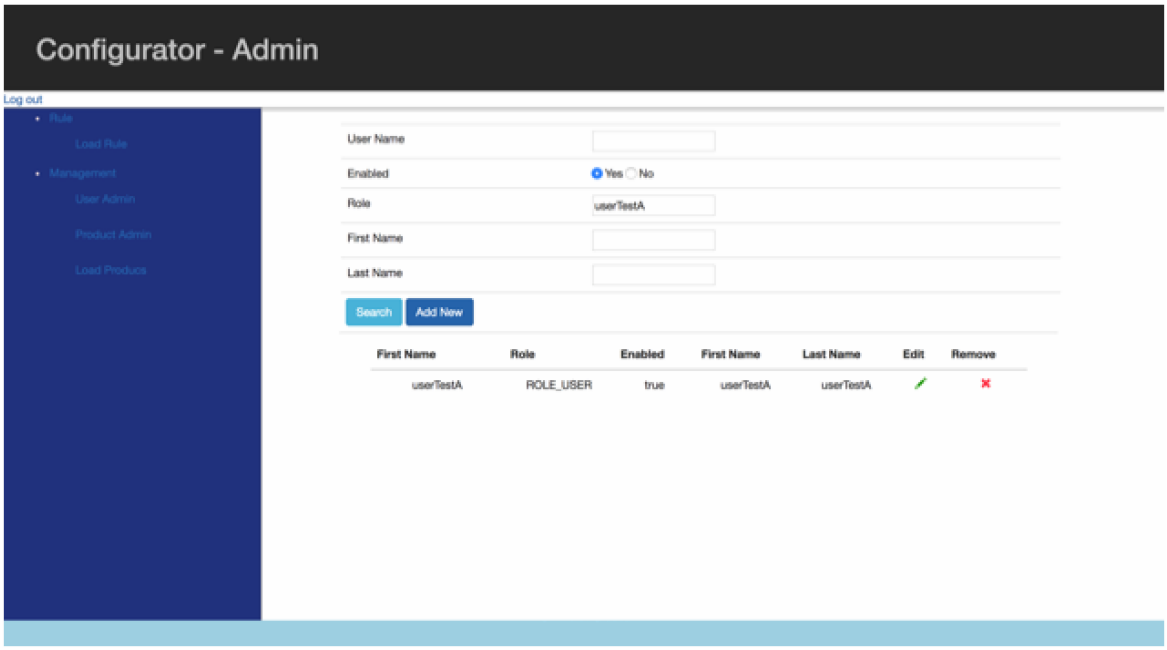

Рисунок 3.17 - Пошук користувачів за прізвищем

Для внесення змін до даних існуючих облікових записів користувачів адміністратор може ініціювати процедуру редагування безпосередньо з таблиці результатів пошуку або загального списку користувачів. Після вибору відповідного користувача система відображає спеціалізовану сторінку редагування, де адміністратор може оновити значення різних атрибутів облікового запису.

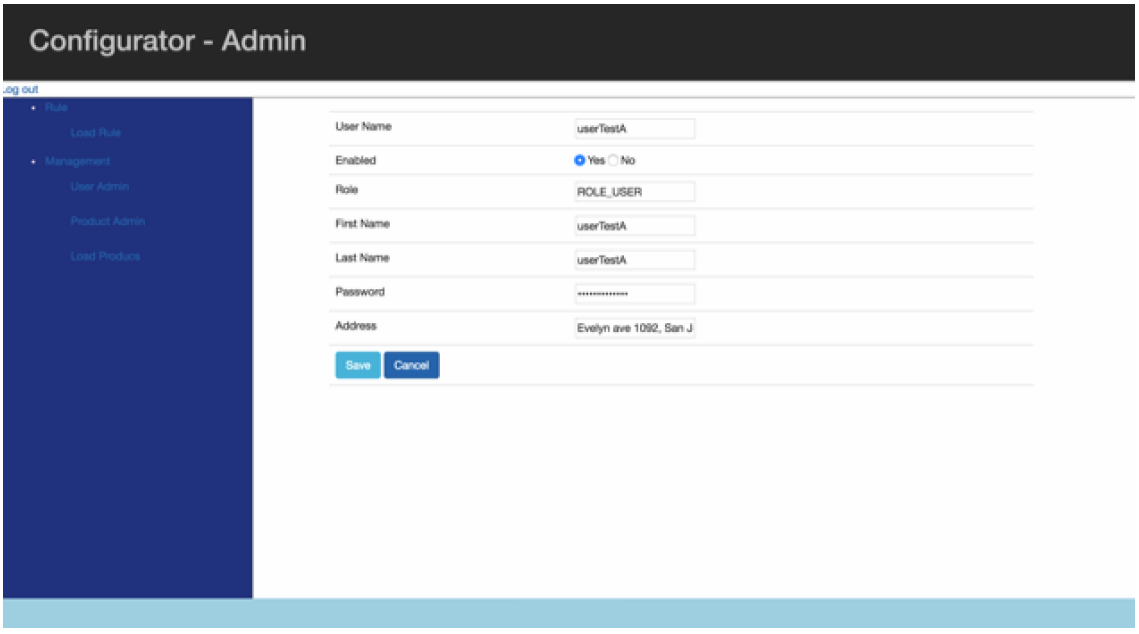

Рисунок 3.18 – Сторінка редагування користувача

Передбачено опції для збереження внесених змін або їх скасування, після чого здійснюється повернення до головної сторінки модуля управління користувачами. Інтерфейс сторінки редагування користувача проілюстровано на рисунку 3.18.

Адміністратор має можливість додавати нові облікові записи користувачів до системи. У модулі управління користувачами доступна відповідна опція "Додати нового користувача". При виборі цієї опції система відображає сторінку для введення даних нового користувача. Після заповнення необхідних полів адміністратор може або зберегти новий обліковий запис у базі даних, або скасувати операцію створення. Інтерфейс сторінки додавання користувача представлено на рисунку 3.19.

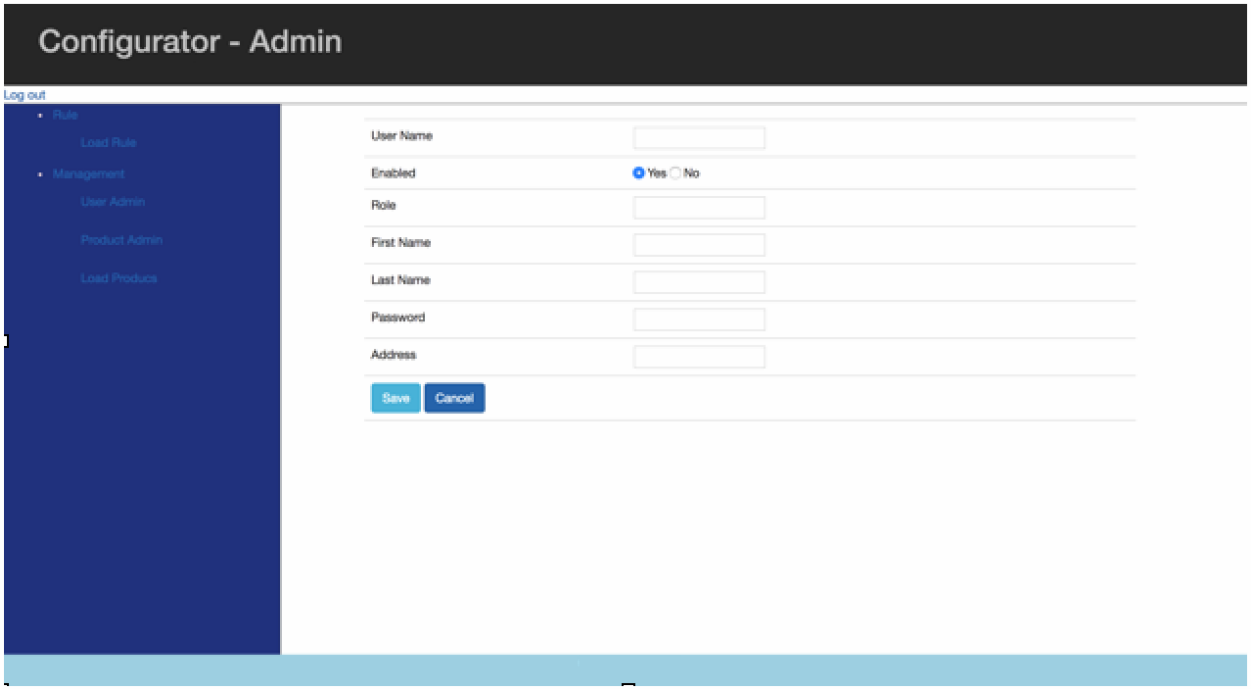

Рисунок 3.19 - Сторінка додавання користувача

Система надає адміністратору можливість видалення існуючих облікових записів користувачів. На сторінці управління користувачами поруч із записом кожного користувача передбачено опцію для ініціації процедури видалення (зазвичай у вигляді відповідної піктограми). При виборі цієї опції

система вимагає явного підтвердження від адміністратора для уникнення випадкового видалення даних. У разі підтвердження, запис користувача буде остаточно видалено з системи, і він перестане відображатися у списках користувачів. Приклад реалізації опції видалення користувача показано на рисунку 3.20.

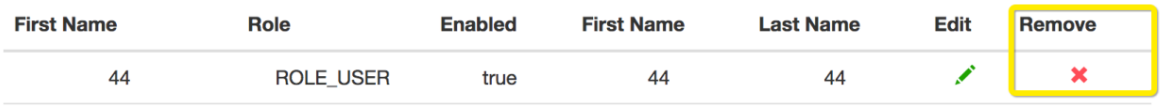

Рисунок 3.20 – Опція видалення користувача

3.3. Представлення функціональних модулів профілю адміністратора

Система надає адміністратору можливість здійснювати пакетне завантаження нових товарних позицій до продуктового каталогу. Для цього реалізовано спеціалізований інтерфейс або сторінку, яка дозволяє імпортувати дані про продукти, зазвичай у стандартизованому форматі файлів (наприклад, CSV, як згадувалося раніше). Ця функціональність оптимізує процес первинного наповнення каталогу або його масового оновлення. Інтерфейс завантаження продуктів показано на рисунку 3.21.

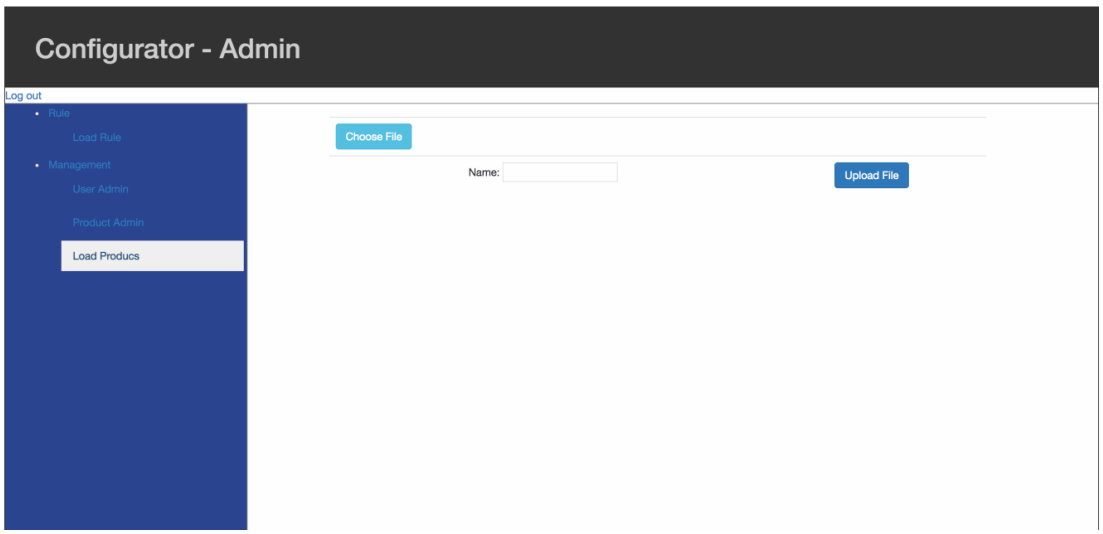

Рисунок 3.21 – Сторінка завантаження продуктів

Для забезпечення коректності та валідності конфігурацій продуктів система оперує набором специфічних правил. Модуль управління правилами конфігурації надає адміністратору інтерфейс для завантаження та оновлення цих правил. Правила, що зазвичай визначають сумісність компонентів, обов'язкові залежності та обмеження за кількістю, можуть бути імпортовані із зовнішнього файлу. Цей модуль є критично важливим для підтримки актуальності логіки конфігурації відповідно до асортименту продукції та технічних вимог. Інтерфейс управління правилами конфігурації представлено на рисунку 3.22.

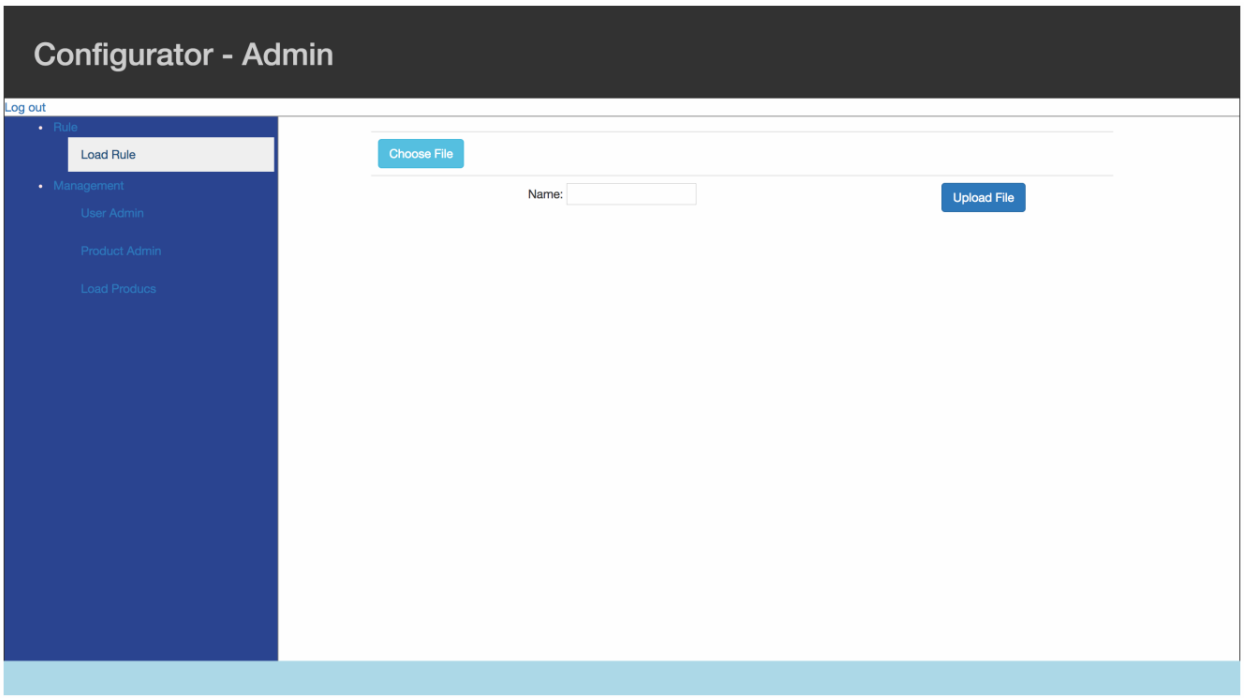

Рисунок 3.22 – Сторінка завантаження правил

Для полегшення роботи адміністратора із системою та надання допомоги у використанні різних функціональних можливостей передбачено доступ до довідкової інформації або документації. Спеціальна сторінка або розділ системи слугує точкою доступу до цих ресурсів, надаючи адміністратору необхідні відомості щодо управління продуктами, користувачами, правилами конфігурації та іншими аспектами роботи

системи. Інтерфейс доступу до довідкової інформації показано на рисунку 3.23.

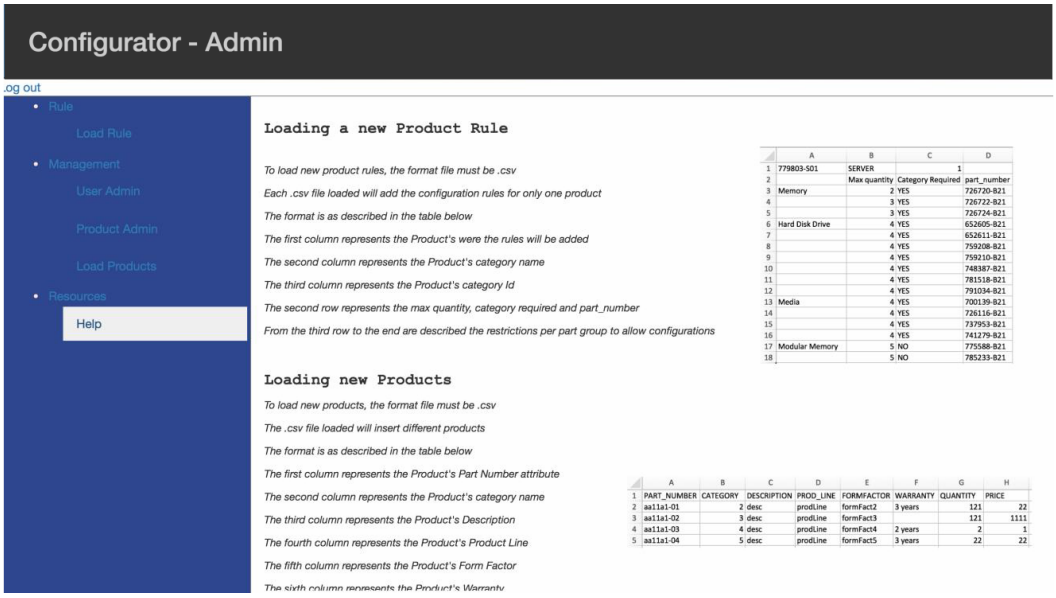

Рисунок 3.23 – Сторінка із довідковою інформацією

3.4. Розробка модуля користувача

Користувацький профіль структурований як набір функціональних модулів, призначених для забезпечення виконання різноманітних користувацьких завдань. Доступ до особистої сторінки користувача надається виключно після успішної автентифікації в системі.

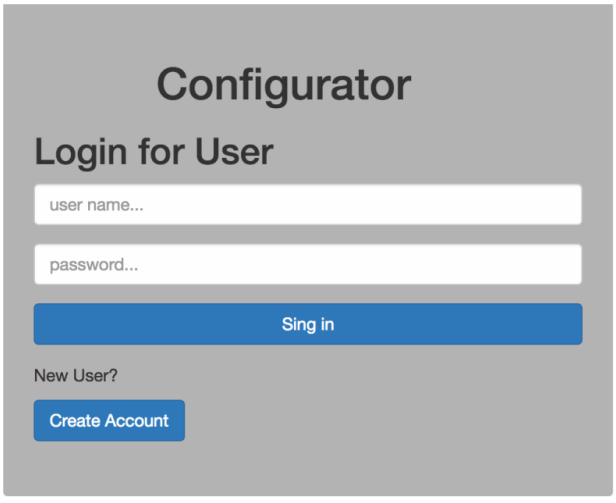

Рисунок 3.24 – Сторінка автентифікації для користувача

Сторінка автентифікації користувача призначена для здійснення процедури входу в систему і показана на рис. 3.24.

Сторінка вітання забезпечує первинне інформування користувача про вхід у систему. Деталі представлені на рис. 3.25.

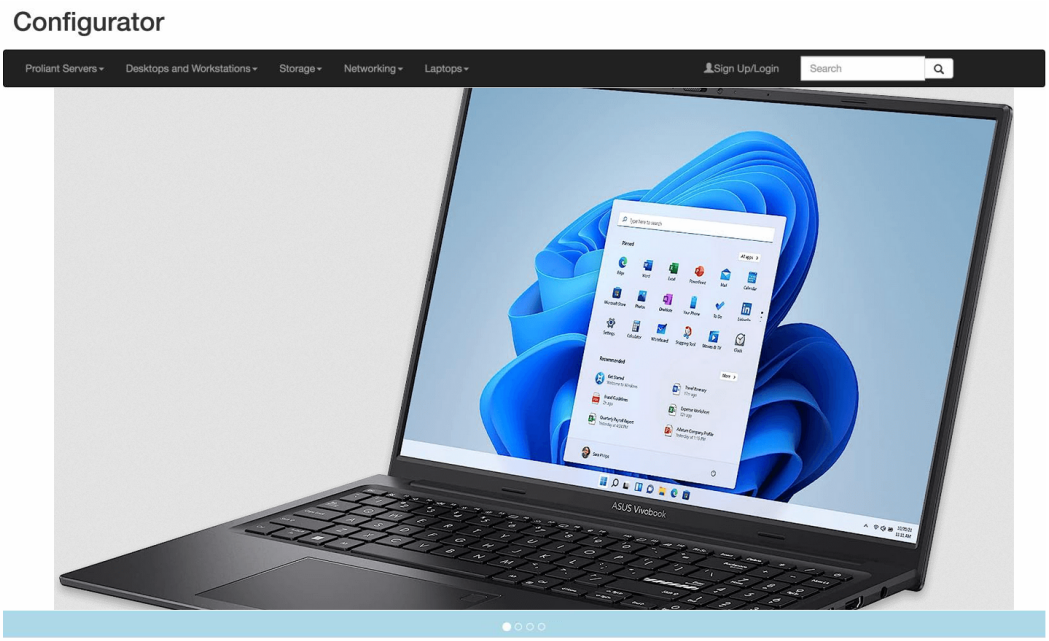

Рисунок 3.25 – Головна сторінка для користувача

Сторінка реєстрації облікового запису користувача забезпечує функціонал створення нового облікового запису (рис. 3.26).

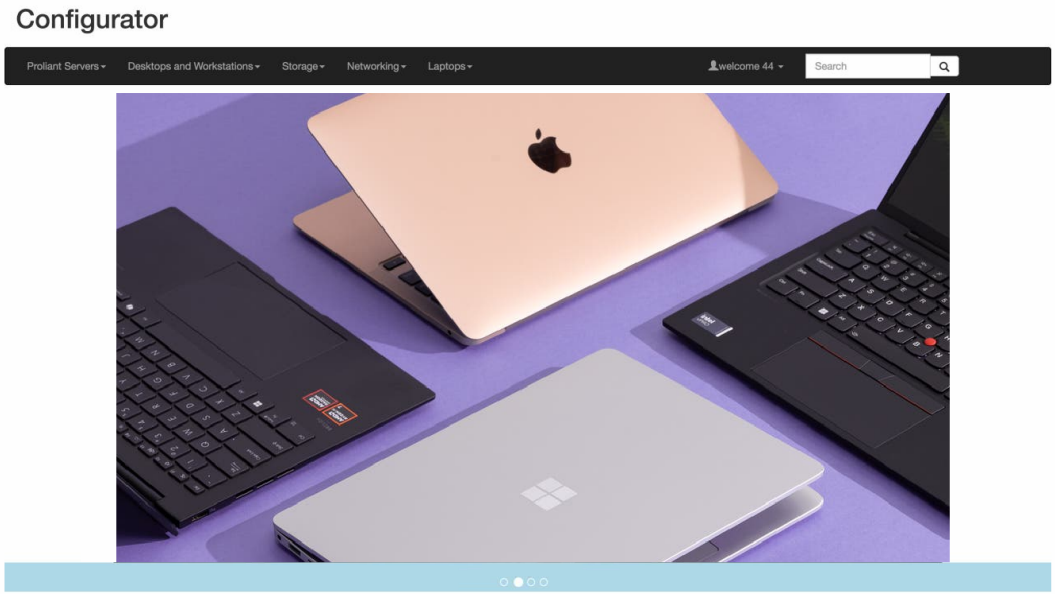

Рисунок 3.26 - Сторінка вітання автентифікованого користувача

У разі введення користувачем невірних облікових даних, буде відображено повідомлення про помилку (рис. 3.27).

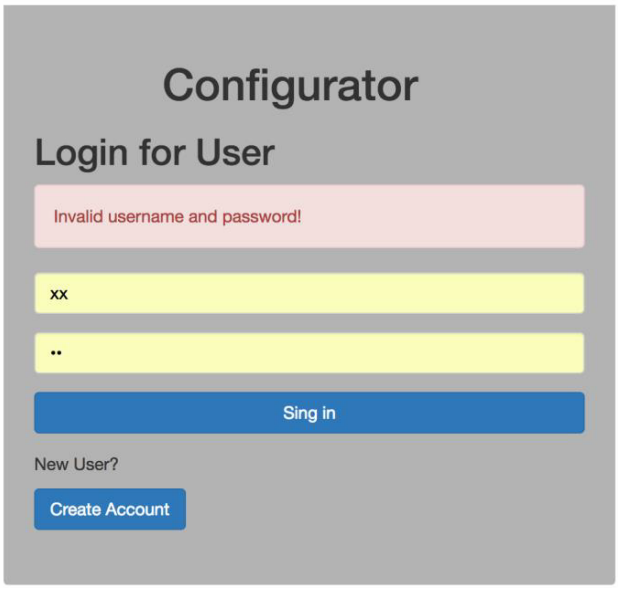

Рисунок 3.27 - Сторінка інформування про невірні облікові дані користувача

Система Configurator забезпечує можливість створення облікового запису користувача за допомогою кнопки "Sign Up/Login" та наступної опції "Create Account". Це перенаправляє користувача на сторінку реєстрації облікового запису (рис. 3.28 та 3.29).

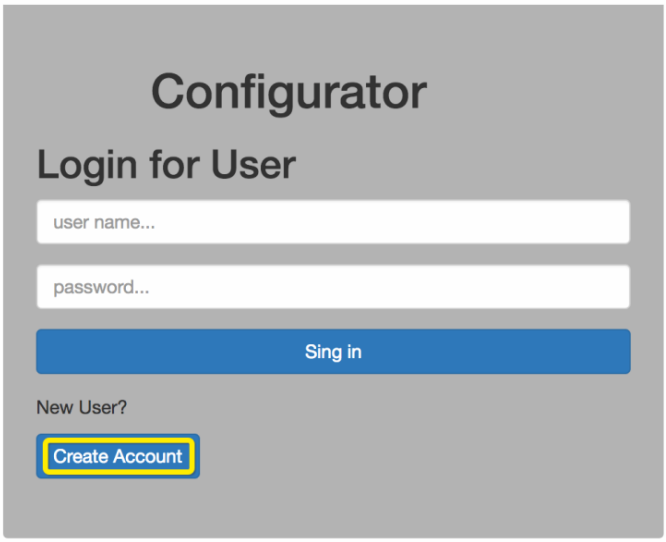

Рисунок 3.28 – Вікно створення нового акаунту

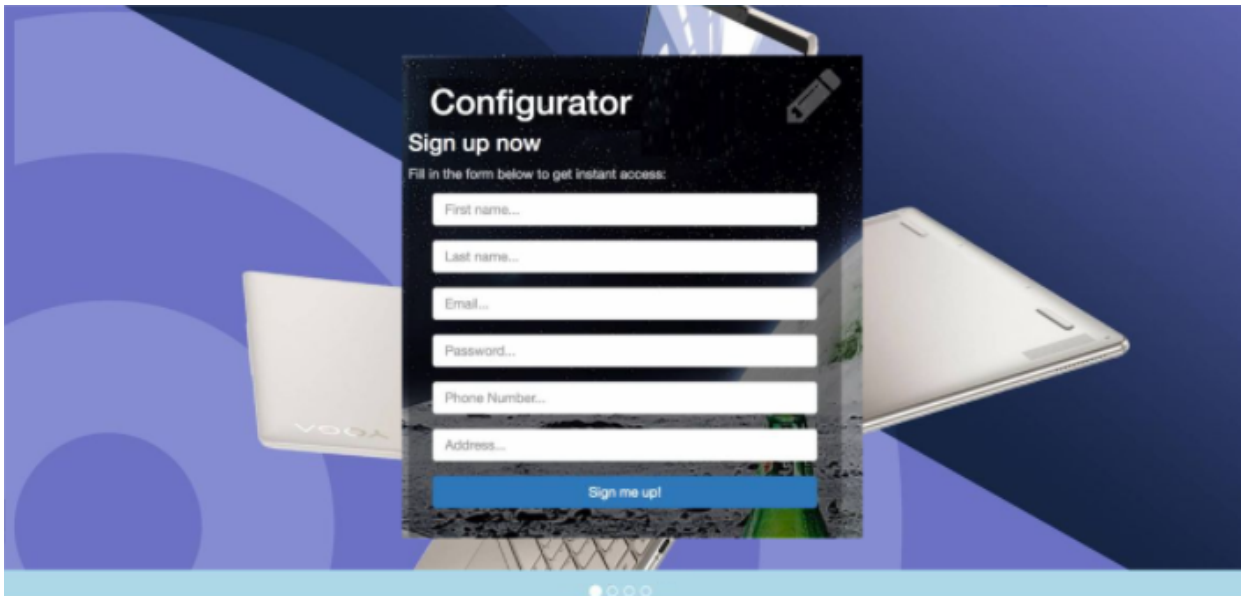

Рисунок 3.29 - Сторінка реєстрації

Після успішної автентифікації користувач отримує доступ до сторінки збережених конфігурацій, що містить перелік усіх конфігурацій, збережених користувачем. Кожна конфігурація може бути видалена або відредагована. Цей перелік оснащений механізмом пагінації, що забезпечує відображення різного набору конфігурацій залежно від обраної сторінки (рис. 3.30 та 3.31).

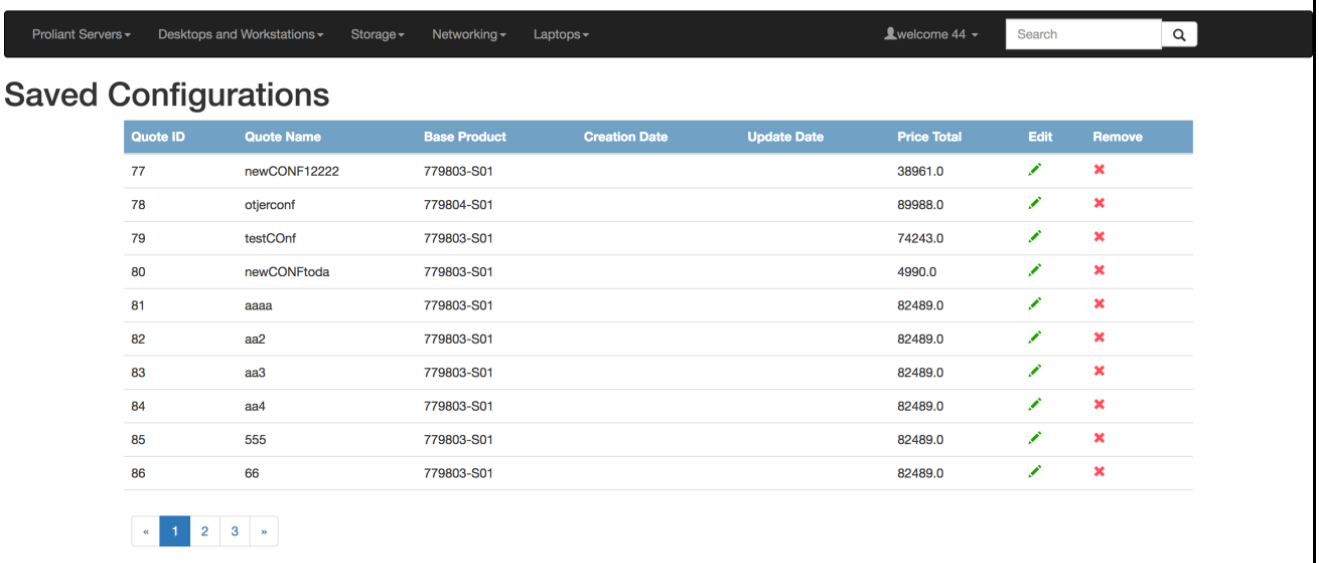

Рисунок 3.30 – Сторінка збережених конфігурацій

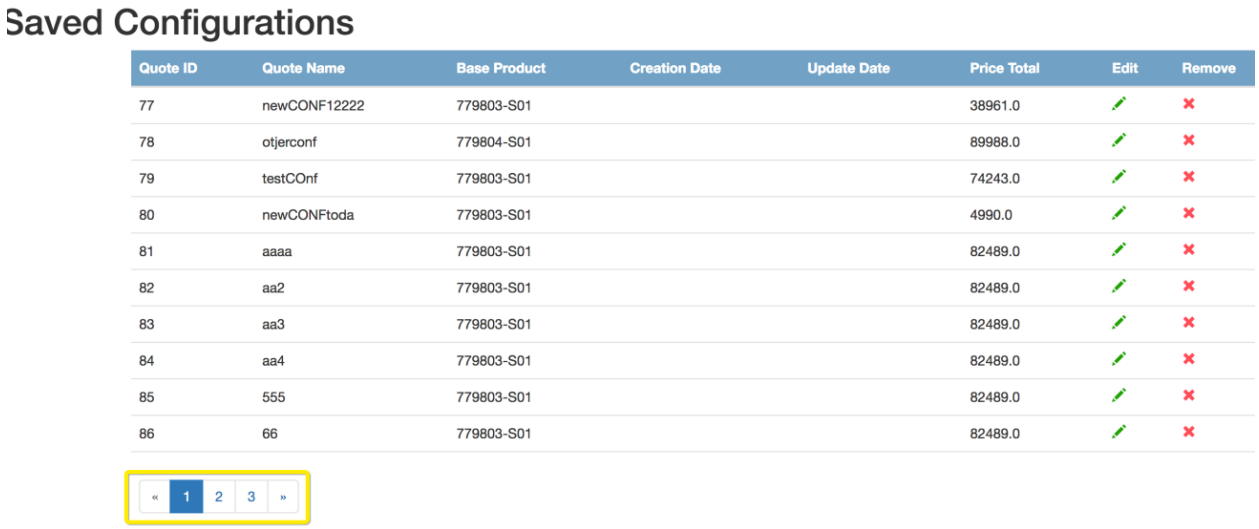

Рисунок 3.31 - Опція пагінації збережених конфігурацій

Автентифікований користувач має можливість редагування конкретної конфігурації шляхом натискання на іконку "Edit". Після вибору іконки "Edit" користувач буде перенаправлений на сторінку конфігурації, де відображається продукт, що конфігурується, його категорія, кількість доданих компонентів та ціна кожного компонента, а також результати валідації включених компонентів (рис. 3.32).

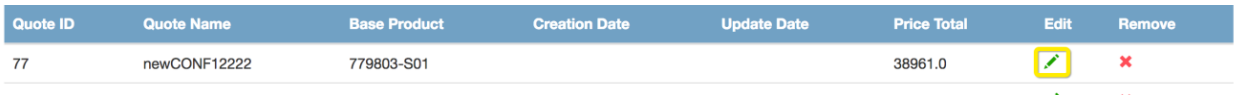

Рисунок 3.32 - Опція редагування збереженої конфігурації

Автентифікований користувач має можливість видалення конфігурації шляхом вибору опції видалення. Перед видаленням конфігурації зі збережених конфігурацій користувача, система відображає спливаюче вікно підтвердження (рис. 3.33).

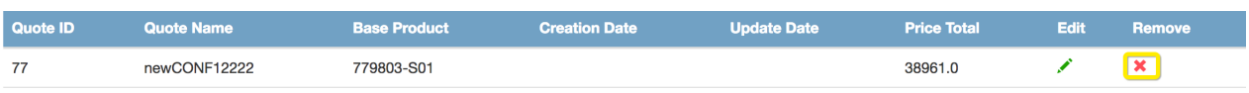

Рисунок 3.33 - Опція видалення збереженої конфігурації

Після введення нових значень до збереженої конфігурації та вибору опції "Update Configuration", користувач може здійснити оновлення збереженої конфігурації. Після цього користувачеві буде відображено повідомлення про успішне оновлення (рис. 3.34 та 3.35).

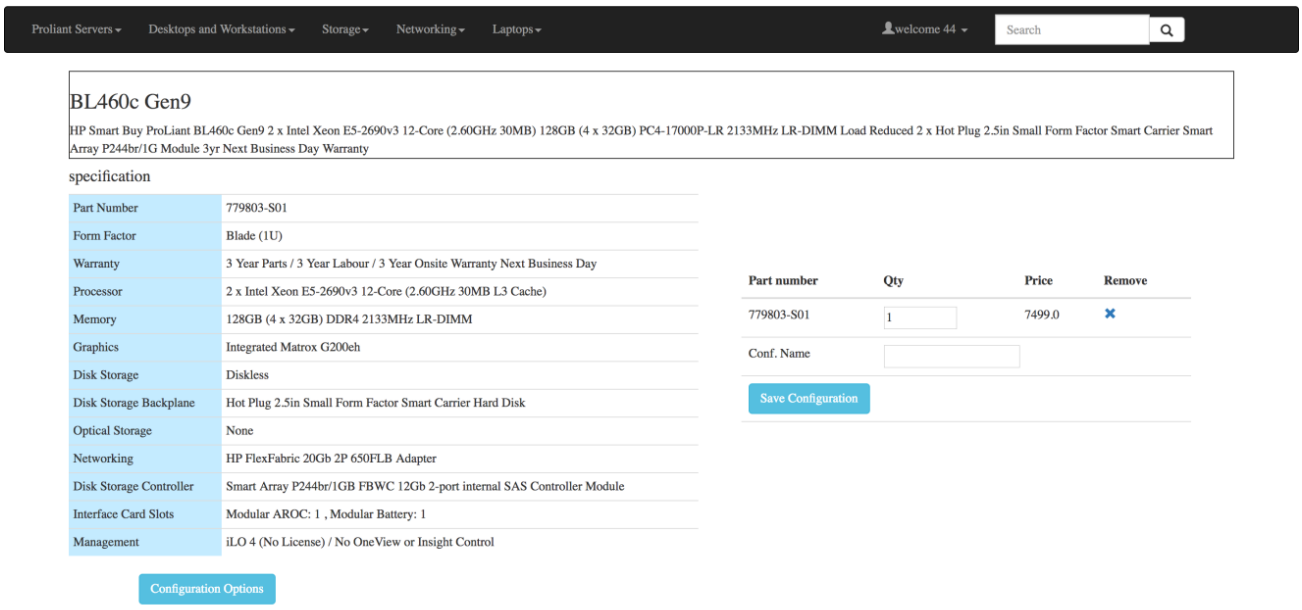

Рисунок 3.34 - Сторінка редагування конфігурації

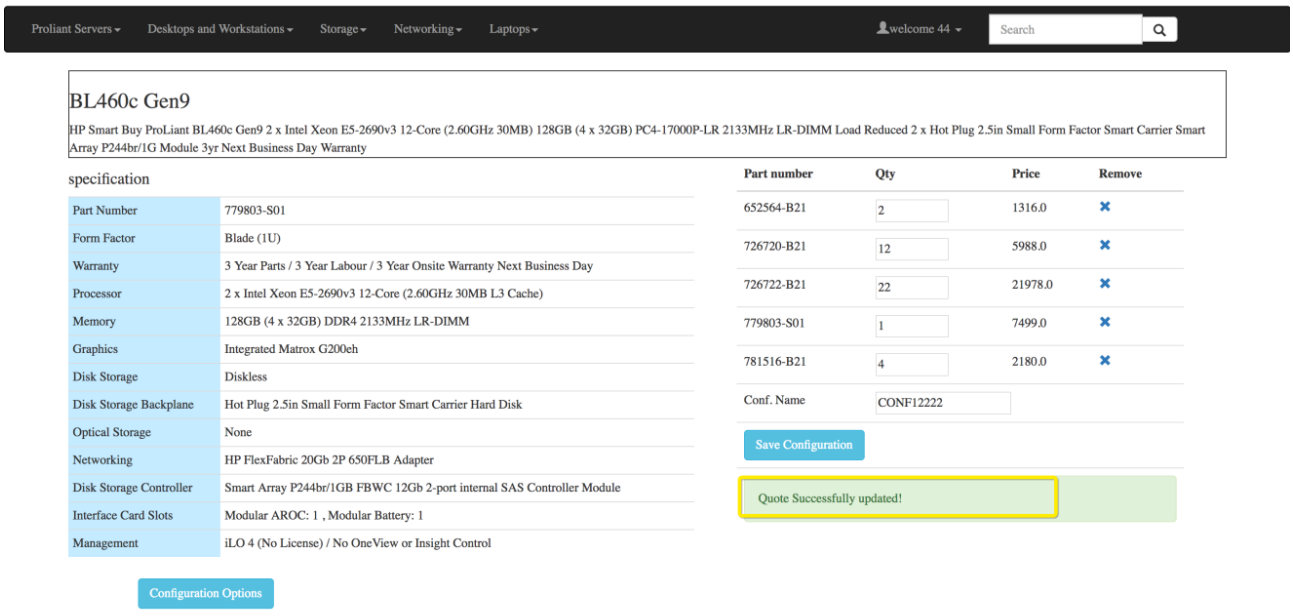

Рисунок 3.35 - Повідомлення про успішне оновлення конфігурації

Користувач може здійснити створення нової конфігурації після вибору продукту з доступних категорій у заголовку меню. Буде відображена Сторінка створення нової конфігурації (рис. 3.36).

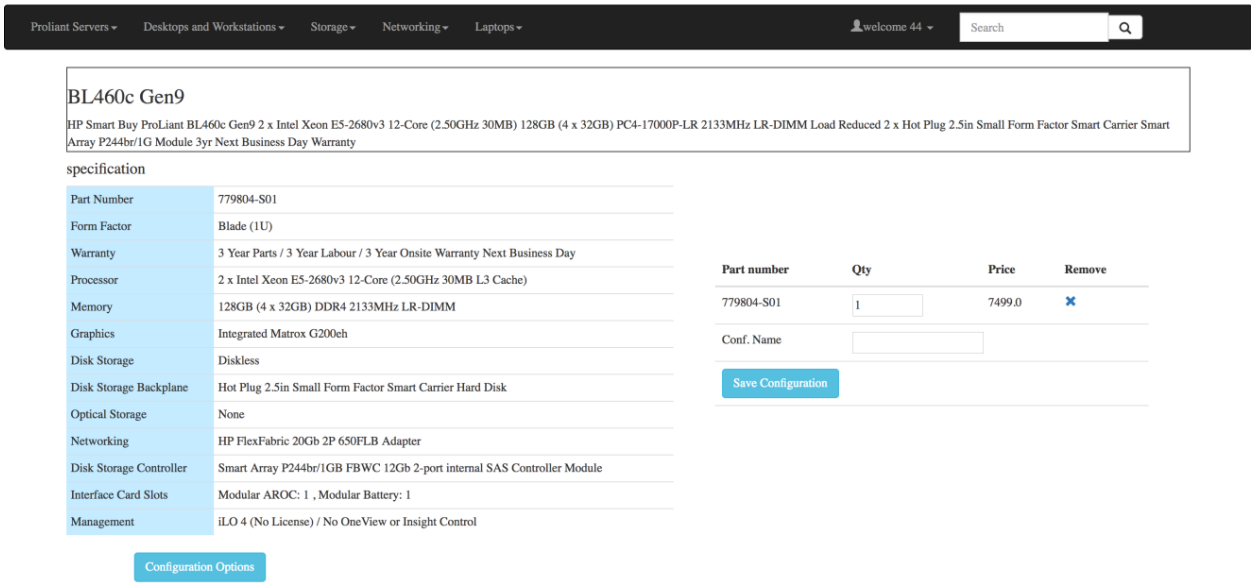

Рисунок 3.36 - Сторінка створення нової конфігурації

Опції конфігурації для продукту відображаються при виборі користувачем кнопки "Configuration Option" (рис. 3.37).

Рисунок 3.37 - Секція опцій конфігурації продукту

Для відображення доступних компонентів для кожної опції конфігурації користувач може натиснути на посилання на варіанти конфігурації для кожної опції, після чого відображається таблиця з компонентами, що належать до цієї опції. Користувач може ввести кількість

кожного необхідного компонента. Після цього нові додані компоненти будуть відображені у секції підсумків конфігурації продукту, разом із будь-якими правилами валідації, що можуть бути активовані цією конфігурацією (рис. 3.38 та 3.39).

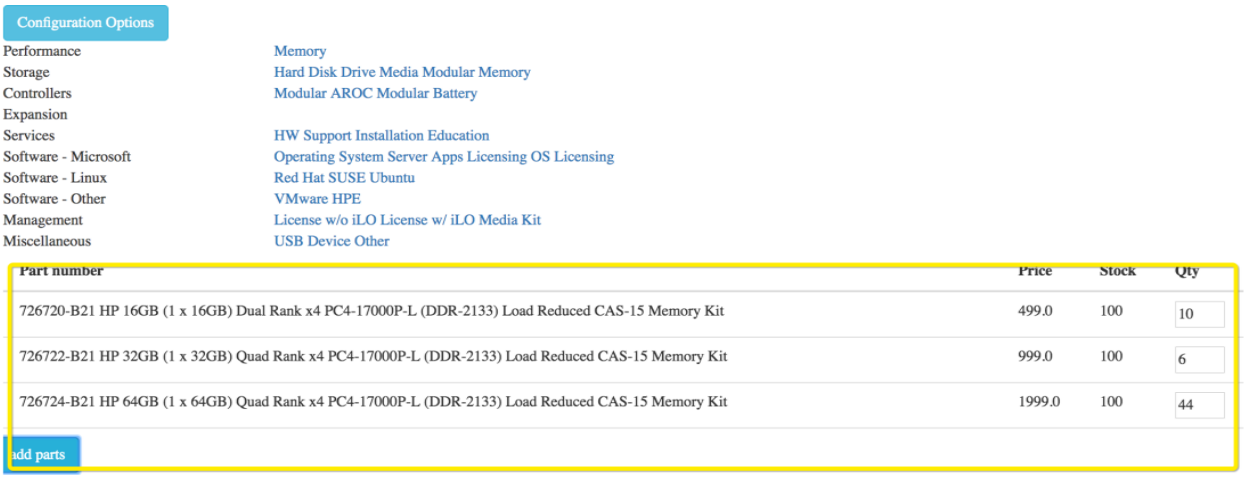

Рисунок 3.38 - Таблиця додавання компонентів для опцій конфігурації

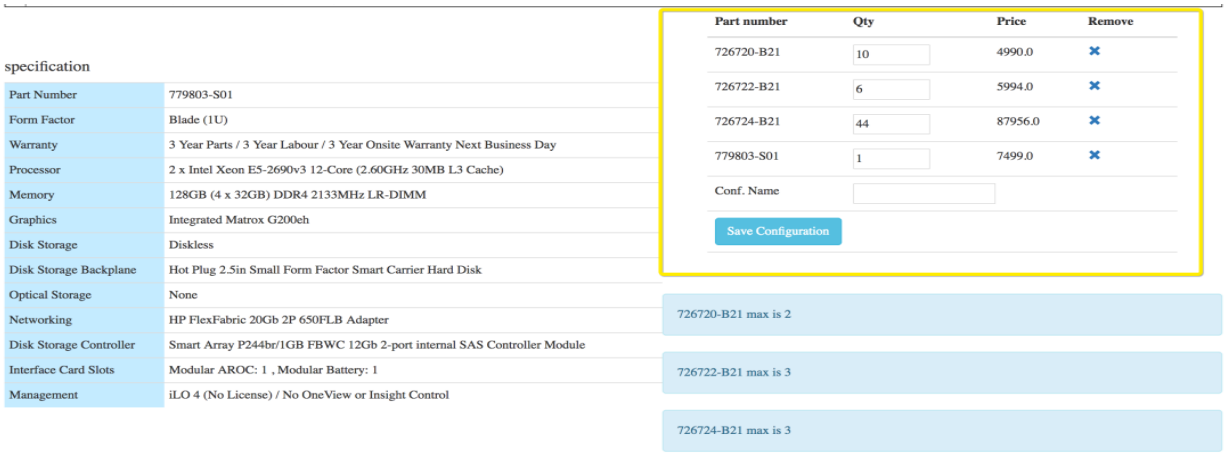

Рисунок 3.39 - Секція підсумків конфігурації

Кожен продукт належить до певної категорії, і ці категорії відображаються у заголовку меню програми (рис. 3.40).

Рисунок 3.40 - Сторінка категорії

На Сторінці списку продуктів відображаються всі продукти, що належать до обраної категорії (рис. 3.41).

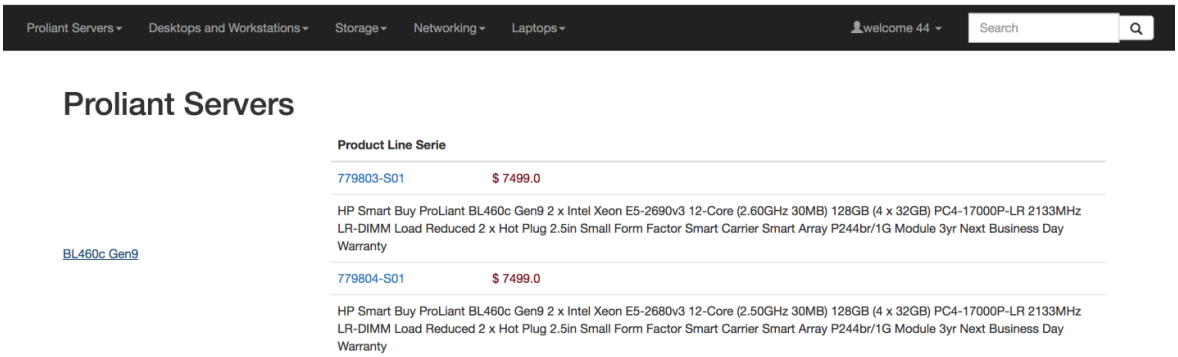

Рисунок 3.41 - Сторінка списку продуктів

Доступ до сторінки конфігурації продукту надається шляхом натискання на продукт, відображений на сторінці списку продуктів, або шляхом пошуку в полі пошуку (рис. 3.42).

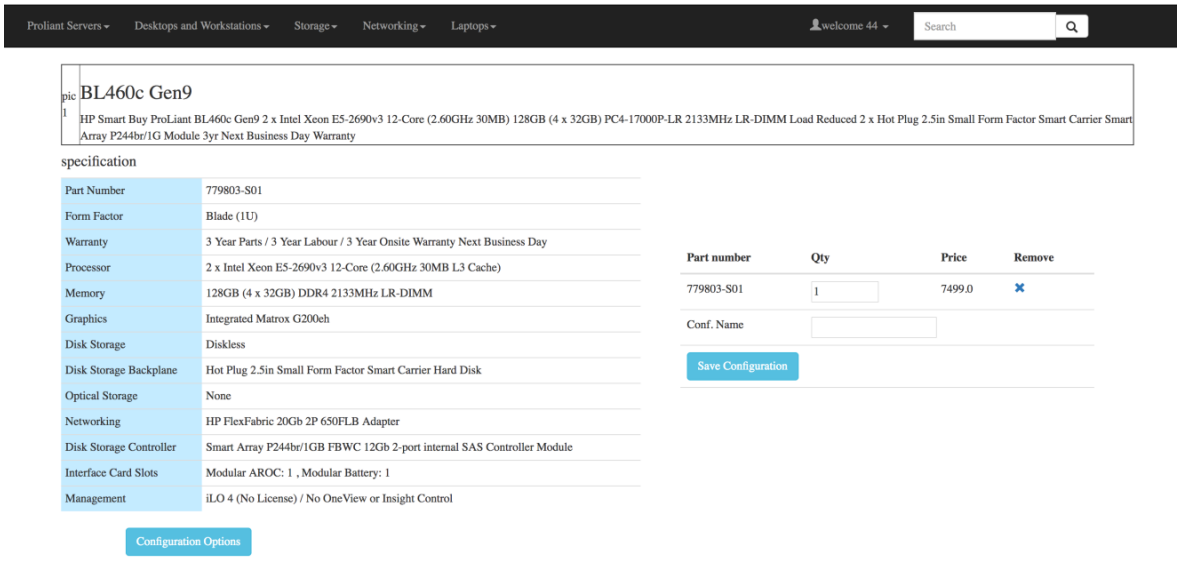

Рисунок 3.42 - Сторінка конфігурації продукту

Пошук продукту може здійснюватися безпосередньо за допомогою поля пошуку у заголовку меню. Функціонал підказки пошуку в програмі реалізовано за допомогою використання Apache Solr (рис. 3.43).

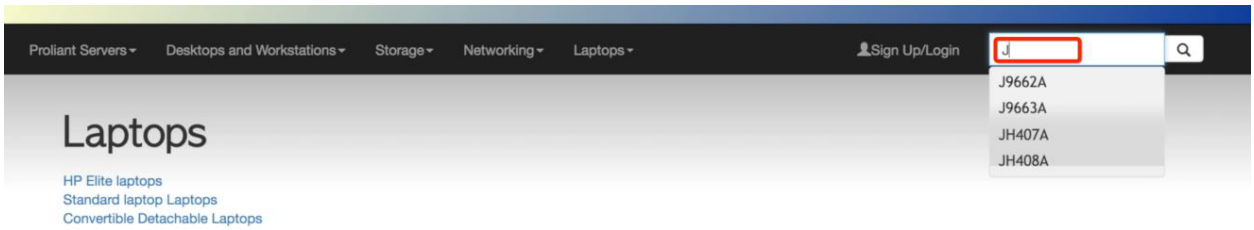

Рисунок 3.43 - Підказка пошуку

3.5. Реалізація та розгортання системи

У цьому підрозділі буде пояснено мінімальні вимоги до розгортання системи Configurator. Процес розгортання залежить від типу операційної системи (ОС), який обирає системний адміністратор. У даному випадку буде розглянуто реалізацію цього процесу в ОС Linux, хоча розгортання можливе і в інших ОС.

3.5.1. Інсталяція Java

Для даної програми використовується JDK, який доступний у розділі завантажень офіційного вебсайту Oracle. Оберіть дистрибутив RPM та завантажте його. Після завантаження цього дистрибутиву виконайте наступну команду з директорії, куди буде інстальовано файл:

```
$ chmod +x jdk-8u131-linux-x64.rpm.bin
$ ./ jdk-8u131-linux-x64.rpm
```

Прийміть умови та завершіть інсталяцію. Наступним кроком є налаштування змінних середовища та створення файлу /etc/profile.d/javaInstall.sh з наступним вмістом:

```
export JAVA_HOME=/user/java/latest
export PATH=$PATH:$JAVA_HOME/bin
```

Потім виконайте наступну команду для застосування змінних середовища:

```
$ chmod +x /etc/profile.d/javaInstall.sh
```

3.5.2. Налаштування Tomcat

Configurator є програмою, розробленою з використанням Spring MVC Framework, що потребує сервлет-контейнера. Для розгортання програми використовується вебсервер Apache Tomcat. Tomcat вимагає, щоб JRE був інстальований у гостьовій ОС.

Бінарні файли Tomcat можна завантажити з репозиторію шляхом виконання наступних команд:

```
# cd /opt/
# wget http://ftp.itu.edu/Mirror/Apache/tomcat/tomcat-8/v8.0.9/bin/apache-tomca
# tar -xvf apache-tomcat-8.0.9.tar.gz
```

Тепер можна запустити сервер Tomcat за допомогою наступної команди:

```
# cd /opt/apache-tomcat-8.0.9/bin
# ./startup.sh
```

3.5.3. Інсталяція MySQL

MySQL надає репозиторій для кількох дистрибутивів Linux, який містить останню стабільну версію MySQL. Для продовження необхідно додати новий репозиторій MySQL до системи.

Додайте новий репозиторій MySQL до сервера CentOS, виконавши наступну команду yum:

```
# yum localinstall https://dev.mysql.com/get/mysql57-community-release-el7-9.noarc
```

Потім буде запропоновано додати новий репозиторій; введіть 'y' та натисніть 'Enter' для підтвердження. До системи було додано новий репозиторій.

Новий репозиторій MySQL тепер доступний у системі, і можна розпочати інсталяцію останньої стабільної версії MySQL з репозиторію. Назва пакета — 'mysql-community-server'.

В ОС CentOS інсталюємо 'mysql-community-server' за допомогою yum.

```
# yum -y install mysql-community-server
```

Після інсталяції MySQL запустіть його та додайте до автозавантаження під час старту системи за допомогою команди systemctl. Для сервера CentOS використовуйте службу 'mysqld'.

```
# systemctl start mysqld
# systemctl enable mysqld
```

MySQL запущено, для з'єднання використовується порт 3306.

3.5.4. Міграція бази даних

Після інсталяції MySQL необхідно виконати скрипт схеми ConfiguratorDB, визначений у таблиці 2.12. Цей скрипт містить лише таблиці, що використовуються програмою.

Для таблиць Menu та Category потрібні деякі дані, а також для створення користувача з правами адміністратора, виконайте наступні оператори INSERT:

```
INSERT INTO `epicConfiguratorDb`.`users` (username,password,enabled) VALUES ('ADMI
INSERT INTO `epicConfiguratorDb`.`user_roles` (username, role) VALUES ('ADMIN', 'R
INSERT INTO `epicConfiguratorDb`.`category` (`language_id`, `category_name`, `stat
INSERT INTO `epicConfiguratorDb`.`subcat_cat` (category_id,category_name,subcatego
INSERT INTO `epicConfiguratorDb`.`menu` (language_id,menu_type_id,name,description
```

Дані продуктів будуть внесені користувачем-адміністратором та автентифікованим користувачем.

3.5.5. Розгортання програми

Для збирання програми та генерації WAR-файлу, необхідного для її розгортання, було використано утиліту експорту, що надається Eclipse IDE. Після отримання цього WAR-файлу сервер Tomcat необхідно зупинити, перемістити цей WAR до директорії “webapps” у Tomcat, після чого перезапустити сервер. Це здійснить розгортання програми та зробить її готовою до використання та доступною у веб-браузері.

Отже, в рамках даного проєкту було здійснено розробку програмного інструменту Configurator для конфігурації комп'ютерів, що забезпечує користувачам функціонал конфігурування сумісності комп'ютерних продуктів та управління ними. Реалізація програми базується на застосуванні сучасних технологій та фреймворків, зокрема, Java, Spring, Hibernate та Apache Solr, що зумовлює його гнучкість та можливості подальшого розширення.

В якості перспектив подальшого розвитку можуть бути розглянуті реалізація функціоналу придбання, інтеграція із зовнішніми системами та вдосконалення інтерфейсу користувача. Також можливе розширення функціоналу конфігурування та додавання підтримки для збільшення номенклатури продуктів та категорій.

ВИСНОВКИ

В дипломній роботі було проведено всебічний аналіз предметної області, що стосується конфігураційної сумісності продуктів в електронній комерції, з урахуванням сучасних тенденцій у сфері проєктування, розробки та впровадження програмних рішень. Визначено ключові проблеми, пов’язані з ефективною організацією управління конфігураціями продукції, а також здійснено критичний огляд суміжних задач, існуючих комп’ютерних систем і компонентів, що виконують аналогічні функції.

Розроблено концепцію та визначено призначення прототипу програмної екосистеми, що забезпечує підтримку сумісності продуктів. У межах дослідження деталізовано функціональні можливості системи для різних категорій користувачів, включаючи адміністраторів і кінцевих споживачів. Архітектура прототипу враховує принципи масштабованості, модульності й безпеки, що досягається завдяки використанню сучасних технологій — зокрема, J2EE, Spring MVC, Hibernate, Spring Security, Bootstrap і Apache Solr.

Проєктна частина роботи містить розробку архітектури програмної системи, побудову моделі взаємодії користувачів (акторів), а також впровадження шаблонів проєктування (MVC, DAO, Factory), що забезпечують структурованість, повторне використання компонентів і зниження зв’язаності між модулями. Також сформовано модель бази даних із використанням підходів об'єктно-реляційного відображення та метаданих.

У межах реалізаційного етапу створено прототип екосистеми з підтримкою повнофункціонального адміністративного інтерфейсу, модуля управління користувачами, а також реалізовано механізми автентифікації та розмежування прав доступу. Здійснено інсталяцію програмного забезпечення, налаштування серверного середовища, міграцію бази даних та повноцінне розгортання системи.

Таким чином, результати роботи демонструють ефективність запропонованого підходу до проєктування й реалізації програмної екосистеми для управління конфігурацією суміжних задач, що може бути інтегрована в існуючі інформаційні середовища електронної комерції та підприємств з орієнтацією на гнучкість, масштабованість і користувацьку зручність.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Factory Design Pattern - https://www.tutorialspoint.com/design_pattern/factory_pattern.htm
2. Benefit of using MVC | GeeksforGeeks - https://www.geeksforgeeks.org/benefit-of-using-mvc/
3. J. Preece, Y. Rogers, H. Sharp, Interaction Design: Beyond Human-Computer Interaction. 5th ed. Wiley, 2020.
4. B. Shneiderman, C. Plaisant, Designing the User Interface: Strategies for Effective Human-Computer Interaction. 6th ed. Pearson, 2016.
5. A. Felfernig, L. Hotz, C. D. Zanker, Configurator Technology: Principles, Practice, and Opportunities. Cambridge University Press, 2014.
6. D. L. McGuinness, "Product Configuration Systems," IEEE Intelligent Systems, vol. 15, no. 4, pp. 16-19, 2000.
7. M. Fowler, Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.
8. R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures. Ph.D. dissertation, University of California, Irvine, 2000.
9. J. Bloch, Effective Java. 3rd ed. Addison-Wesley, 2018.
10. J. G. Davis, "Analysis of Garbage Collection Algorithms in Modern JVMs," International Journal of Software Engineering Research, vol. 8, no. 3, pp. 112-125, 2023.
11. C. Walls, Spring in Action. 5th ed. Manning Publications, 2018.
12. K. S. Lee, "Building RESTful APIs with Spring MVC," Journal of Web Development, vol. 12, no. 1, pp. 45-58, 2022.
13. C. Bauer, G. King, K. Berglund, Java Persistence with Hibernate. 2nd ed. Manning Publications, 2015.

14. F. R. Chen, "Performance Considerations for Object-Relational Mapping," ACM SIGMOD Record, vol. 34, no. 3, pp. 75-80, 2005.
15. J. M. Manila, Apache Tomcat Administration Handbook. Tech Publications, 2021.
16. A. K. Gupta, "Comparison of Java Servlet Containers," in Proc. International Conference on Enterprise Computing (ICEC 2023), 2023, pp. 301-308.
17. V. Keddy, High Performance MySQL. 4th ed. O'Reilly Media, 2021.
18. C. J. Date, An Introduction to Database Systems. 8th ed. Addison-Wesley, 2003.
19. P. Wang, "Automated Database Schema Migration Techniques," IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 8, pp. 8010-8023, 2023.
20. Spring Security. [Online]. Viewed 2025 - https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/
21. C. D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval. Cambridge University Press, 2008.
22. T. Owen, Apache Solr Essentials. Packt Publishing, 2015.
23. M. Chen, "Integrating Search Engine Technologies into Web Applications," Journal of Information Systems Research, vol. 7, no. 2, pp. 145-160, 2022.
24. I. Sommerville, Software Engineering. 10th ed. Pearson, 2015.
25. A. Humble, M. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010.
26. L. Roberts, "Optimizing Java Application Deployment on Linux Environments," in Proc. International Conference on Cloud and Distributed Systems (ICCDS 2024), 2024, pp. 188-195.
27. K. Wiegers, J. Beatty, Software Requirements. 3rd ed. Microsoft Press, 2013.

28. Apache Solr. [Online]. Viewed 2025 - http://lucene.apache.org/solr/features.html
29. D. Richards, Software Architecture Patterns. O'Reilly Media, 2015.
30. D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," Communications of the ACM, vol. 15, no. 12, pp. 1053-1058, 1972.
31. M. Scott, "Error Handling in Concurrent and Distributed Systems," in Concurrency and Distribution in Software Engineering. Springer, 2022, pp. 112-135.
32. S. Lee, "UI Design Patterns for Data Presentation: Pagination vs. Infinite Scrolling," Journal of User Experience Design, vol. 9, no. 3, pp. 70-85, 2023.
33. L. Bass, P. Clements, R. Kazman, Software Architecture in Practice. 3rd ed. Addison-Wesley, 2012.

ДОДАТКИ

Додаток А

Фрагменти програмних кодів

Представлення об'єктів бази даних

Метадані кожної таблиці представлені в наступних класах, без урахування методів отримання та встановлення

```
@Entity
@Table(name="CATEGORY")
public class Category{
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="category_id")
    private Integer categoryId;
    @Column(name="language_id")
    private char languageId;
    @Column(name="category_name")
    private String categoryName;
    @Column(name="category_date")
    private Date creationDate;
    @Column(name="update_Date")
    private Date updateDate;
    @Column(name="category_Image_Url")
    private String categoryImageUrl;
    @Column(name="category_Thumbnail_Url")
    private String categoryThumbnailUrl;
    @Column(name="state")
    private String state;
    @OneToMany(mappedBy="category", cascade=CascadeType.ALL)
    private Set<Product> product;
}
```

Сутність Product

```
@Entity
@Table(name="part_validation")
public class PartValidation {
    @EmbeddedId
    private PartValidationId partValId;
    @Column(name = "incompatible")
    @Type(type = "org.hibernate.type.NumericBooleanType")
    private boolean incompatible;
    @Column(name = "required")
    @Type(type = "org.hibernate.type.NumericBooleanType")
    private boolean required;
    @Column(name = "max_quantity")
    private int maxQuantity;
    @Column(name = "min_quantity")
    private int minQuantity;
}
```

Сутність Category

```
@Entity
@Table(name="COUNTRY")
public class Country{
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="country_id")
    private Integer countryId;
    @Column(name="iso")
    private char iso;
    @Column(name="iso3")
    private String iso3;
    @Column(name="name")
    private String name;
    @Column(name="nicename")
    private String nicename;
    @Column(name="numcode")
    private Integer numcode;
    @Column(name="phonecode")
    private Integer phonecode;
}
```

Сутність PartValidation

```
@Entity
@Table(name="menu")
public class Menu {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="menu_id")
    private Integer menuId;
    @Column(name="language_id")
    private String languageId;
    @Column(name="menu_type_id")
    private Integer menuTypeId;
    @Column(name="name")
    private String name;
    @Column(name="description")
    private String description;
    @Column(name="parent_id")
    private Integer parentId;
    @OneToMany(mappedBy="parent")
    private Set<Menu> menus;
    @Column(name="categories_category_id")
    private Integer categoryId;
    @ManyToOne(fetch=FetchType.EAGER, cascade=(CascadeType.ALL))
    @JoinColumn(name="parent_id", insertable = false, updatable = false)
    private Menu parent;
}
```

Сутність Menu

```
@Entity
@Table(name="USERS")
public class User {
    @Id
    @Column(name = "username", unique = true, nullable = false, length = 45)
    private String username;
    @Column(name = "password", nullable = false, length = 60)
    private String password;
    @Column(name = "enabled", nullable = false)
    private boolean enabled;
    @OneToOne(mappedBy="user", cascade = CascadeType.ALL)
    private Customer customer;
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private Set<UserRole> userRole = new HashSet<UserRole>(0);
}
```

Сутність User

```
@Entity
@Table(name="USER_ROLES")
public class UserRole {
    @Id
    @GeneratedValue(strategy = IDENTITY)
    @Column(name = "user_role_id", unique = true, nullable = false)
    private Integer userRoleId;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "username", nullable = false)
    private User user;
    @Column(name = "role", nullable = false, length = 45)
    private String role;
}
```

Сутність UserRole

```
@Entity
@Table(name="configuration_product")
public class ConfigurationProduct implements java.io.Serializable{
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = IDENTITY)
    @Column(name = "configuration_product_Id", unique = true, nullable = false)
    private Integer confProductId;
    @ManyToOne(cascade=CascadeType.ALL)
    @JoinColumn(name = "quote_configuration_quote_id")
    private QuoteConfiguration quoteConfiguration;
    @Column(name = "price")
    private String price;
    @Column(name = "quantity")
    private Integer quantity;
    @Column(name = "product_Part_Number", nullable=false)
    private String productPartNumber;
    public ConfigurationProduct(){
    }
    public Integer getConfProductId() {
        return confProductId;
    }
    public void setConfProductId(Integer confProductId) {
        this.confProductId = confProductId;
    }
}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “Створення прототипу екосистеми для рішення суміжних задач клієнтів ”

Обсяг пояснювальної записки: 77 аркушів.

Дата закінчення роботи: 10 червня 2025 р.

Підпис студента _____