ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської кваліфікаційної роботи

на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА WEB-ДОДАТКУ ДЛЯ СКЛАДСЬКОГО ОБЛІКУ НА ПЛАТФОРМІ ASP.NET CORE ТА BLAZOR МОВОЮ C#**»

Виконав: студент 4 курсу, групи ПД–44
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Чернявський Ж.А.
(прізвище та ініціали)

Керівник Гаманюк І.М.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Нормконтроль _____
(прізвище та ініціали)

Київ –2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність - 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри
Інженерії програмного
забезпечення
_____О.В. Негоденко

«____»_____2023 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Чернявський Ждан Анатолійович

(прізвище, ім'я, по батькові)

1. Тема роботи: «**Розробка web-додатку для складського обліку мовою C# на платформі ASP.NET Core та Blazor**»

Керівник роботи: Гаманюк Ігор Михайлович, старший викладач

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

затверджені наказом вищого навчального закладу від «24» лютого 2023року №26.

5. Строк подання студентом роботи «1» червня 2023 року
6. Вихідні дані до роботи:

Способи складського обліку, технічна література програмного забезпечення з приводу складського обліку, технічна література .NET, технічні засоби розробки такі як Visual Studio 2022 – середа розробки, методи та бібліотеки C#, MS SQL SERVER, RAZOR PAGES, браузер для керування і тестування.

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

4.1 Опис предметної області автоматизації складського обліку.

4.2 Аналіз наявних засобів та технологій для організації складського обліку.

4.3 Моделювання та проектування застосунку для автоматизації складського обліку.

4.4 Реалізація застосунку для автоматизації складського обліку.

5. Перелік демонстраційного матеріалу (назва основних слайдів)

5.1. Титульний слайд

5.2. Мета, об’єкт та предмет дослідження

5.3. Задачі дипломної роботи

5.4. Аналіз аналогів

5.5. Вимоги до програмного забезпечення

5.6. Засоби реалізації

5.7. Архітектура застосунку

5.8. Діаграми варіантів використання

5.9. Діаграми діяльності

5.10. – 5.11. Схема БД

5.15. Апробація результатів дослідження

5.16. Висновки

5.17. Кінцевий слайд

6. Дата видачі завдання «25» лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	14.03.23-16.04.23	Виконано
2	Розробка вимог до системи	16.04.23-21.04.23	Виконано
3	Проектування системи	21.04.23-02.05.23	Виконано
4	Концепція та архітектура програмного забезпечення	02.05.23-04.05.23	Виконано
5	Програмне реалізація системи, написання коду	04.05.23-18.05.23	Виконано
6	Вступ, висновки, реферат	18.05.23-19.05.23	Виконано
7	Розробка обов‘язкових демонстраційних матеріалів	20.05.23-21.05.23	Виконано
8	Попередній захист роботи	19.05.23-26.05.23	Виконано
9	Здача роботи	29.05.23	

Студент _____ Чернявський Ж.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Гаманюк І.М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи с. 43, рис. 39, джерел 14

Об'єкт дослідження – процес ведення складського обліку.

Предмет дослідження – програмне забезпечення для ведення складського обліку.

Мета роботи – автоматизація ведення складського обліку шляхом впровадження web-додатку для складського обліку.

Методи дослідження – методи створення бази даних товарів та керування ними за допомогою користувацького інтерфейсу (додавання, видалення, редагування товару).

При створені було проведено аналіз існуючих рішень, таких як FishbowlInventory, OdooInventory та ін. систем складського обліку. Такі програмні продукти використовують HTTP методи для проведення маніпуляцій з товарами

Особливістю є зручний інтерфейс та швидка відповідь серверу. Всі ці потреби враховані у програмному забезпеченні. Веб-додаток написано за допомогою ASP.NET Core (MVC) .NET 6, Blazor та RazorPages з використанням бази даних MS SQL Server.

Таким чином було розроблено та описано серверну, інтерфейсну частини та базу даних яка, виконує потреби користувача в управлінні обліком товарів та створенні звітності.

Галузь використання – початковий бізнес, підприємство, складський облік.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 10
ВСТУП 11
1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ СКЛАДСЬКОГО ОБЛІКУ 13
1.1 Загальні відомості про облік товарів 13
1.2 Система контролю запасів у сфері торгівлі 14
1.3 Облік товарів 14
1.4 Переваги та недоліки 15
1.5 Схема автоматизованого варіанту складського обліку 17
2 АНАЛІЗ ІСНУЮЧИХ ЗАСОБІВ ТА ТЕХНОЛОГІЙ ДЛЯ СКЛАДСЬКОГО ОБЛІКУ 18
3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗАЦІЇ СКЛАДСЬКОГО ОБЛІКУ 27
3.1 Модель предметної галузі 27
3.2 Модель прецедентів 28
3.3 Нефункціональні вимоги. Бачення з контекстною діаграмою 30
3.4 Проектування діяльності. Діаграма діяльності 30
3.5 Проектування послідовності викликів методів та взаємодії об'єктів. Діаграми послідовностей 33
3.6 Проектування послідовності. Діаграми станів 36
3.7 Діаграми комунікацій (кооперацій) 37
3.8 Схема БД 39
4 РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗАЦІЇ СКЛАДСЬКОГО ОБЛІКУ 40
4.1 Рефлексія 40
4.2 Асинхронні операції 41
4.3 Лямбда-вирази 44
4.4 Атрибути 45
4.5 JSON-конфігурація 47

4.6 Razor Pages **48**
4.7 Entity Framework Core **50**
4.8 Blazor **51**
ВИСНОВКИ **54**
ПЕРЕЛІК ПОСИЛАНЬ **55**
ДОДАТКИ **57**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ORM – Object-Relational Mapping (Об'єктно-реляційна проєкція).

ПК – персональний комп’ютер.

ПЗ – програмне забезпечення.

IDE – Integrated Development Environment (Інтегроване середовище розробки).

БД – База даних.

HTTP – HyperText Transfer Protocol (протокол передачі гіпертекстових документів).

TP – Thread Pool (басейн потоків).

LINQ – Language Integrated Query (мова інтегрованих запитів).

API – Application Programming Interface.

SQL – Structured query language (мова структурованих запитів).

RFID – Radio Frequency Identification (радіочастотна ідентифікація);

ВСТУП

У сучасному динамічному бізнес-середовищі, де швидкість і точність є ключовими факторами успіху, ефективний складський облік стає незамінним для компаній у всіх галузях. Забезпечення належного контролю запасів, моніторинг руху товарів і моніторинг стану запасів є ключовими питаннями для досягнення високої продуктивності та конкурентоспроможності. У цьому контексті все більшого значення набуває розробка веб-додатків для складського обліку, які можуть автоматизувати та оптимізувати процеси, пов’язані з управлінням складськими запасами. Веб-додатки є потужним інструментом для ефективного управління запасами, надають точну та актуальну інформацію, аналітичні звіти та інтуїтивно зрозумілі інтерфейси. Важливість такого програмного забезпечення полягає в його здатності спростити складські процеси, знизити ризик помилок, підвищити продуктивність праці та забезпечити безперебійний ланцюг поставок. Компанії, які використовують веб-додаток для складського обліку, можуть автоматизувати складні та трудомісткі завдання, такі як розміщення замовлень, отримання товарів, управління запасами, відстеження руху товарів і надсилання звітів. Це дозволяє їм зосередитися на стратегічному розвитку бізнесу, покращенні клієнтського досвіду та наданні якісних послуг.

Веб-додаток для складського обліку дозволить підприємствам забезпечити високу ефективність і точність управління запасами, знизити ризик втрати товару. Це дозволить в режимі реального часу отримувати інформацію про стан запасів, прогнозувати потреби в запасах і вчасно реагувати на зміни попиту. Крім того, такий додаток підвищить задоволеність клієнтів, забезпечуючи швидку обробку замовлень, точну інформацію про наявність товару та своєчасну доставку.

Об’єкт дослідження: сервер та інтерфейс для обліку товарів.

Предмет роботи: веб-застосунок для складського обліку.

Мета роботи: полегшення процесу обліку товарів та формування звітності.

Завдання роботи: розробка програмного забезпечення для складського обліку товарів та ведення звітності, а також бази даних, серверу та користувацького інтерфейсу для нього.

Методика дослідження:

1. Обрати оптимальний метод обробки та зберігання даних за допомогою серверу та бази даних;
2. Обрати архітектуру для веб-системи складського обліку;
3. Обрати модель розробки програмного забезпечення.

Враховуючі потреби системи та існуючи рішення, було зроблено висновок, що найактуальнішим варіантом буде створеня застосунку на клієнт-серверній архітектурі та використанням БД з обміном даними по протоколу HTTPS

Наукова новизна роботи: наукова новизна роботи полягає в створенні клієнт-серверного коду поєднаного між собою в одному застосунку та бази даних до нього, а також розробка веб-системи, яка дозволить зберігати складські дані на сервері, формувати звітність у форматі xlsx. Також варто звернути увагу на використання MS SQL Server, який допомагає значно пришвидшити пошук по базі даних.

Практична значущість результатів: полягає в тому що даний продукт може бути використаний у всіх сферах діяльності яка потребує вести облік товару, де потрібен сервер з стабільним інтернет з'єднанням для обліком (додавання, редагування та видалення товарів, додавання аккаунтів працівників до бази даних.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ СКЛАДСЬКОГО ОБЛІКУ

1.1 Загальні відомості про облік товарів

Ми живемо у часі коли додатки для автоматизації бізнес процесів стрімко набирають популярність, адже відкривається дуже багато бізнесів які стрімко зростають, і у цій гонці бізнесів стало дуже важливо не стояти на місці та все робити швидко. Для бізнесів які орієнтовані на торгівлю є дуже важливим пришвидшити та полегшити облік того, шо у них є на складах і для цього на допомогу приходять додатки для складського обліку.

Реалізація належного складського обліку є життєво важливим для успіху будь-якого бізнесу, особливо тих, хто займається купівлею, продажем і виробництвом товарів. Існує багато різних методів ведення обліку запасів, включаючи традиційні паперові методи. Проте все більше компаній переходять на автоматизовані системи, щоб спростити цей процес.

Складська діяльність є невід'ємною частиною організації, яка має зв'язок з складським обліком. Погане управління складськими операціями може негативно вплинути на ефективність всього бізнесу.

В умовах зростання конкуренції рівень технічної оснащеності складу впливає на такі важливі аспекти, як час і оперативність вантажно-розвантажувальних робіт. Ефективна організація роботи складу може знизити логістичні витрати компанії. Низка факторів, таких як глобалізація та розвиток ланцюгів постачання, визначає вимоги до складського програмного забезпечення:

— зберігання та постійне оновлення великої кількості різних видів товарів;

— надання послуг високої якості;

— можливість легко та швидко вносити та редагувати товари;

— можливість легко формувати звіти.

1.2 Система контролю запасів у сфері торгівлі

Існує багато видів торгових організацій, всі вони можуть займатися переробкою, продажом, постачанням товарів.

Всі ці організації об'єднує те, що вони мають вести облік товарів у продовж всього існування бізнесу.

1.3 Облік товарів

Процес обліку товарів є важливою частиною фінансового менеджменту. Це включає систематичне відстеження руху товарів від надходження на склад до продажу або використання у виробничих процесах. Облік запасів допомагає компаніям контролювати свої запаси, забезпечувати своєчасне поповнення та вимірювати фінансові результати. До основних етапів процесу обліку товарів відносяться: Надходження товару, як тільки товар надходить на склад, він реєструється в системі бухгалтерського обліку. Це включає визначення кількості товарів, їх вартості, ідентифікаторів (наприклад, код або номер товару) та інших необхідних даних. Зберігання та контроль запасів, Товари зберігаються на складі з урахуванням їх типових характеристик і вимог безпеки. Система обліку дозволяє контролювати кількість товарів на складі за допомогою регулярних перевірок, інвентаризації та систем автоматичного поповнення. Класифікація та каталогізація, товари можна класифікувати за різними критеріями, такими як тип, розмір, модель або виробник. Це дозволяє компанії швидко знаходити ідентифіковані товари, вести статистику продажів і забезпечувати належне управління запасами. Відстеження руху товарів: кожен рух товарів відображається в системі обліку. Це може включати продаж товарів клієнтам, переміщення товарів між складами або виробничими відділами, повернення товарів, амортизацію тощо. Кожна операція супроводжується відповідним записом у бухгалтерських документах (наприклад, накладних, рахунках-фактурах тощо). . Визначення витрат на зберігання. Знання витрат на зберігання є життєво важливим для фінансового аналізу компанії.

Витрати на запаси можна визначити за допомогою різних методів, наприклад FIFO (First In, First Out), LIFO (Last In, First Out) або середньозваженого методу. Це допоможе визначити точну вартість проданого товару та залишок на складі. Аналіз і звітність: система обліку продукції надає дані для аналізу продажів, поведінки клієнтів, ефективності управління запасами та інших аспектів бізнес-операцій. Ці дані використовуються для прийняття рішень, планування поповнення запасів, ціноутворення на продукти та фінансової звітності. Загалом, процес обліку товарів є важливим і складним аспектом управління підприємством. Це допомагає забезпечити належний контроль запасів, оптимізувати витрати та спланувати рух товарів для досягнення максимальної ефективності та прибутковості бізнесу.

1.4 Переваги та недоліки

Складське програмне забезпечення може значно пришвидшити роботу бізнесу та має багато переваг таких, як:

— Автоматизація та ефективність: раніше працівники підприємств які мають склади були вимушені робити облік та звіти в ручну в такому ПЗ як Excel, наразі автоматизація складського обліку вирішує цю проблему зручним, зрозумілим, та інтуїтивним інтерфейсом. Такі ПЗ надають широкий спектр функцій, в тому числі і генерування звіту по товарам.

— Підвищення точності та надійності: такі додатки допомогають уникати помилок які люди допускають при мануальному введені інформації. Вони допомогають працівникам не розточувати концентрацію, а чим більше концентрація працівника тим вище ефективність та тим нижчий відсоток допустити помилку, у такому бізнесі кожна помилка може коштувати величезних грошових ресурсів.

— Мобільність: подібні додатки які працюють у веб середовищі надають можливість отримувати до них доступ не встановлюючи на кожний комп’ютер підприємства.

— Масштабованість та зниження витрат: підхід роботи додатку обліку як веб-додатку економить місце на пристроях зберігання інформації та не потребує закупівлі дорогих комп’ютерів з високою обчислювальною потужністю, це призводить до того, що підприємці можуть зекономити на жорстких дисках високого об’єму, дорогих процесорах, тощо. Такі додатки розгортаються на хмарі і до них можуть отримати доступ усі працівники локальної корпоративної мережи або засобами VPN, через те, що веб- додатки легко інтегруються з хмарами вони можуть рости разом із розширенням бізнесу, також сучасні хмари дають можливість.

— Економити на використанні додатку, адже коли ваш додаток не використовується гроші за нього не платяться, це стало можливим завдяки таким хмарним технологіям як AWS Lambda.

— Скорочення часу обслуговування: програми для складського обліку дозволяють прискорити процес прийому та відпуску товарів на складі. Завдяки автоматизації та точному веденню записів працівники можуть швидко знаходити потрібні їм деталі, підвищуючи продуктивність і скорочуючи час виконання завдань.

— Відстежування у реальному часі: при додаванні товару одним працівником, усі зміни будуть побачені іншими працівниками, такий підхід разом із логуванням дає прозорість обігу товарів;

Але у таких додатках є і мінуси які можуть негативно впливати на стан бізнесу, при аналізі були виявлені наступні недоліки:

— Залежність від Інтернету: використання веб-додатку потребує постійного доступу до Інтернету. При відсутності або ненадійному підключенні до Інтернету можуть виникнути проблеми з доступом до даних і роботою з системою складського обліку, а це означає що працівники не зможуть працювати.

— Потреба в навчанні персоналу: використання нових веб-додатків складського обліку може потребувати навчання персоналу. Це може забрати час і

ресурси компанії, а також вимагати зміни робочих процесів і адаптації до нової системи;

Такі додатки є в край важливими для усіх підприємств які хоч якось пов’язані із торгівлею, такі як : Виробничі компанії, дистриб'ютори, роздрібні магазини та інші бізнеси, які мають потребу в управлінні запасами. Вони допомагають забезпечити точність та ефективність ведення обліку, знижують витрати як через помилку працівників так і на закупівлі дорогих комп'ютерів з високою обчислювальною потужністю. Додатки для інвентаризації та складського обліку є важливим інструментом для багатьох компаній, які прагнуть ефективно керувати запасами, зменшити витрати та підвищити конкурентоспроможність.

1.5 Схема автоматизованого варіанту складського обліку

Рисунок 1.5.1 - Схема автоматизованого варіанту складського обліку

2 АНАЛІЗ ІСНУЮЧИХ ЗАСОБІВ ТА ТЕХНОЛОГІЙ ДЛЯ СКЛАДСЬКОГО ОБЛІКУ

Отже розглянемо існуючі рішення для виконання задач складського обліку. Першим аналого це «Fishbowl Inventory». Fishbowl Inventory — це програмне забезпечення для керування запасами та виробництвом, розроблене Fishbowl. Призначений для підприємств будь-якого розміру, які потребують ефективного контролю над своїми запасами та виробництвом. (рис. 2.1).

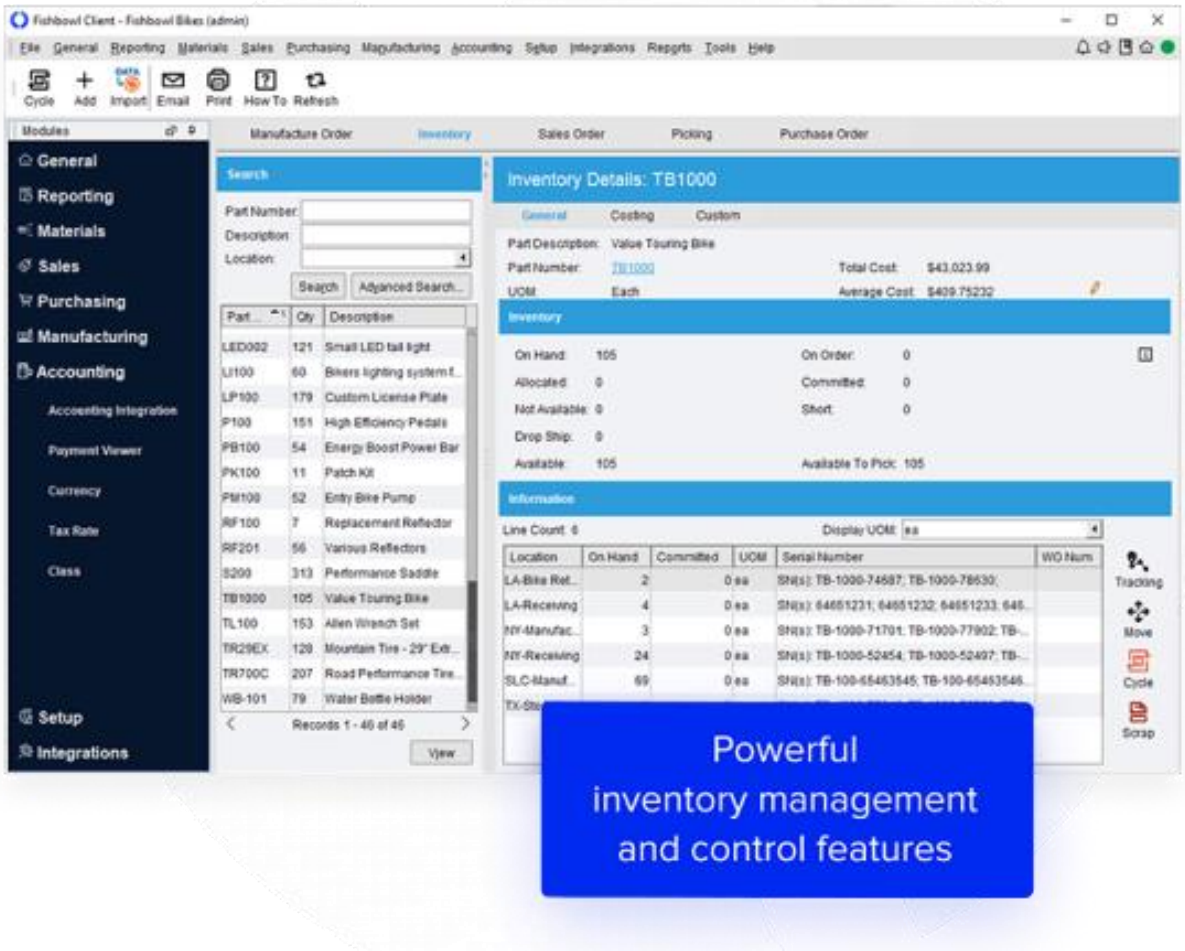


Рисунок 2.1 - Fishbowl Inventory

Основні можливості «Fishbowl Inventory»:

— Управління запасами: Fishbowl дозволяє контролювати всі аспекти запасів, включаючи прихід, відвантаження, переміщення, інвентаризацію і повернення. Ви можете стежити за кількістю наявних одиниць, вартістю запасів, серійними номерами і датами придбання.

— Управління замовленнями: програма дозволяє створювати, відстежувати і керувати замовленнями клієнтів. Ви можете стежити за статусами замовлень, виконувати розрахунок строків виконання, керувати доставкою та використовувати інструменти для оптимізації процесу замовлення.

— Управління виробництвом: Fishbowl дозволяє створювати робочі накази і керувати виробництвом продукції. Ви можете стежити за процесом виробництва, включаючи стадії виробництва, витрати на робочу силу, використання матеріалів і управління розкладом виробництва.

— Управління постачальниками: програма дозволяє створювати і керувати базою даних постачальників, включаючи контактну інформацію, умови постачання і історію замовлень. Ви можете стежити за вартістю закупок, термінами поставок і управляти відносинами з постачальниками.

— Звітність і аналітика: Fishbowl надає широкі можливості для створення звітів і аналізів;

Розглянемо Zoho Inventory — це інтегрована система управління запасами та управління виробництвом, розроблена Zoho Corporation. Цей застосунок дозволяє підприємствам ефективно контролювати та оптимізувати свої запаси та виробничі операції (рис. 2.2).

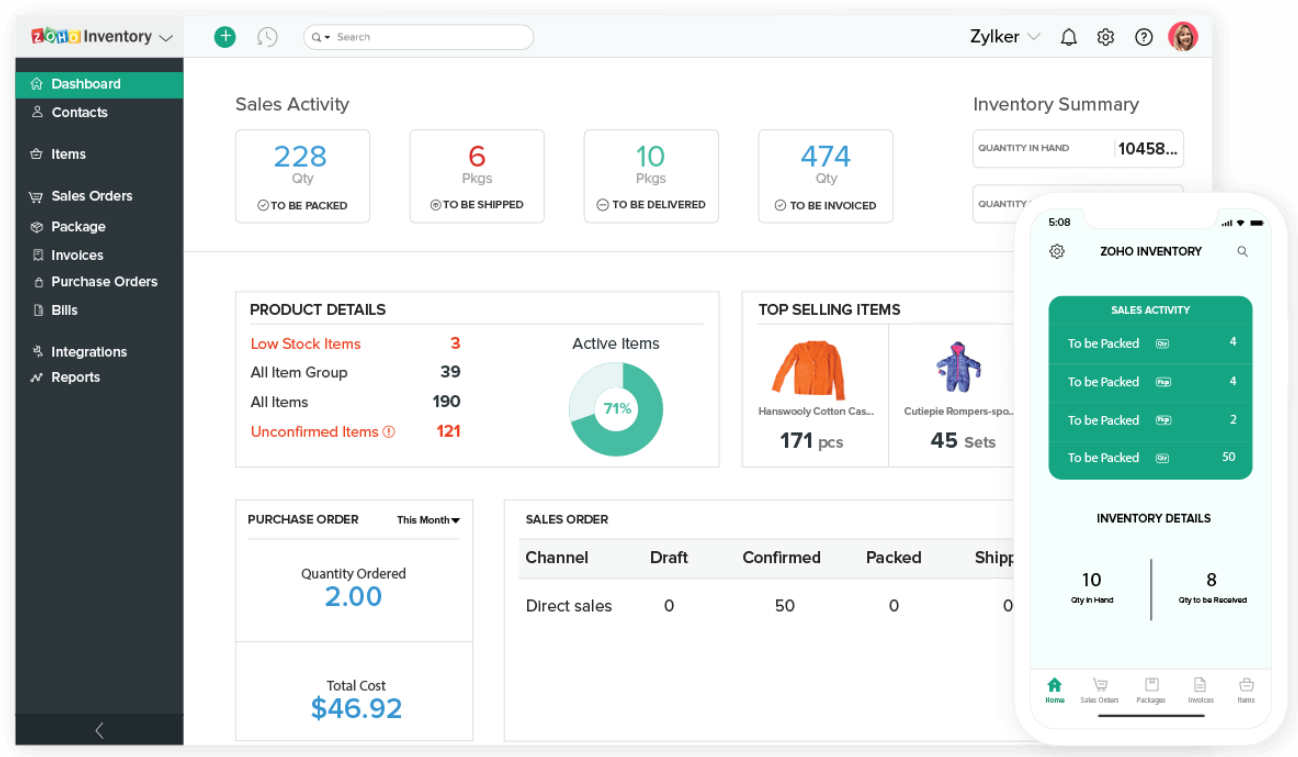


Рисунок 2.2 - Zoho Inventory

Основні можливості «Zoho Inventory»:

— Управління запасами: Zoho Inventory надає можливість стежити за всіма аспектами запасів, включаючи прихід, відвантаження, переміщення, інвентаризацію та повернення. Ви можете відслідковувати кількість наявних одиниць, вартість запасів, серійні номери та дати придбання.

— Управління замовленнями: програма дозволяє створювати, відстежувати та керувати замовленнями клієнтів. Ви можете відстежувати статуси замовлень, розраховувати строки виконання, керувати доставкою та використовувати інструменти для оптимізації процесу замовлення.

— Управління виробництвом: Zoho Inventory дозволяє створювати робочі накази та керувати виробництвом продукції. Ви можете відстежувати процес виробництва, включаючи стадії виробництва, витрати на робочу силу, використання матеріалів та управління графіком виробництва.

— Управління постачальниками: Zoho Inventory дозволяє створювати та керувати базою даних постачальників, включаючи контактну інформацію, умови

постачання та історію замовлень. Ви можете відстежувати вартість закупок, терміни поставок та управляти взаєминами з постачальниками.

— Звітність і аналітика: Zoho Inventory надає розширені засоби;

Ще одним представником застосунків для керування складом є програмне забезпечення «Odoo Inventory» - це модуль системи управління ресурсами підприємства (ERP) під назвою Odoo. Він надає комплексні можливості для управління запасами, виробництвом і постачанням. (рис. 2.3).

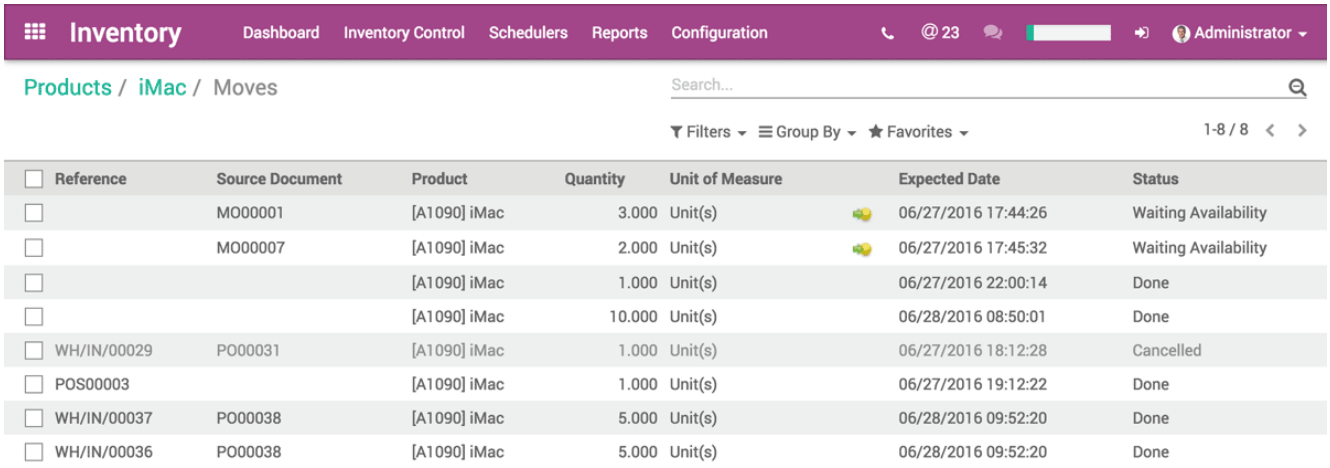


Рисунок 2.3 - Odoo Inventory

Основні функції Odoo Inventory включають:

— Управління запасами: Odoo Inventory дозволяє вам контролювати всі аспекти запасів, включаючи надходження, відправлення, переміщення, інвентаризацію та повернення. Ви можете відстежувати кількість доступних одиниць, інвентарну вартість, серійні номери, дати покупки та різні характеристики продукту.

— Управління замовленнями: ви можете створювати та керувати замовленнями клієнтів у системі Odoo Inventory. Це дозволяє відстежувати статуси замовлень, керувати доставками, розраховувати терміни доставки, контролювати запаси та забезпечувати ефективну обробку замовлень.

— Управління виробництвом: Odoo Inventory надає функціональність для планування та управління виробництвом. Ви можете установлювати графіки

виробництва, керувати використанням матеріалів, відстежувати хід виробництва та оптимізувати процес.

— Управління постачальниками: система дозволяє створювати та керувати базою даних постачальників, включаючи контактну інформацію, умови постачання та історію замовлень. Ви можете контролювати вартість покупок, контролювати час доставки, керувати партнерськими відносинами та використовувати інструменти для оптимізації поставок.

— Звітування та аналітика: Odoo Inventory надає широкі можливості звітності та аналітики, які дозволяють отримати детальну інформацію про ваші запаси, виробництво та постачання. Ви можете створювати звіти про рівень запасів, вартість і товарообіг, аналізувати ефективність виробництва, ідентифікувати популярні продукти та клієнтів. За допомогою інтегрованих інструментів аналітики ви можете визначати ключові показники ефективності, визначати тенденції та вдосконалювати процес прийняття стратегічних рішень для оптимізації процесів управління запасами та виробництвом;

Ще одним представником застосунків для обліку є «TradeGecko» - це хмарна платформа управління запасами та поставками для компаній. Він надає розширені функції для ефективного контролю та оптимізації запасів, керування замовленнями та керування постачальниками (рис. 2.4).

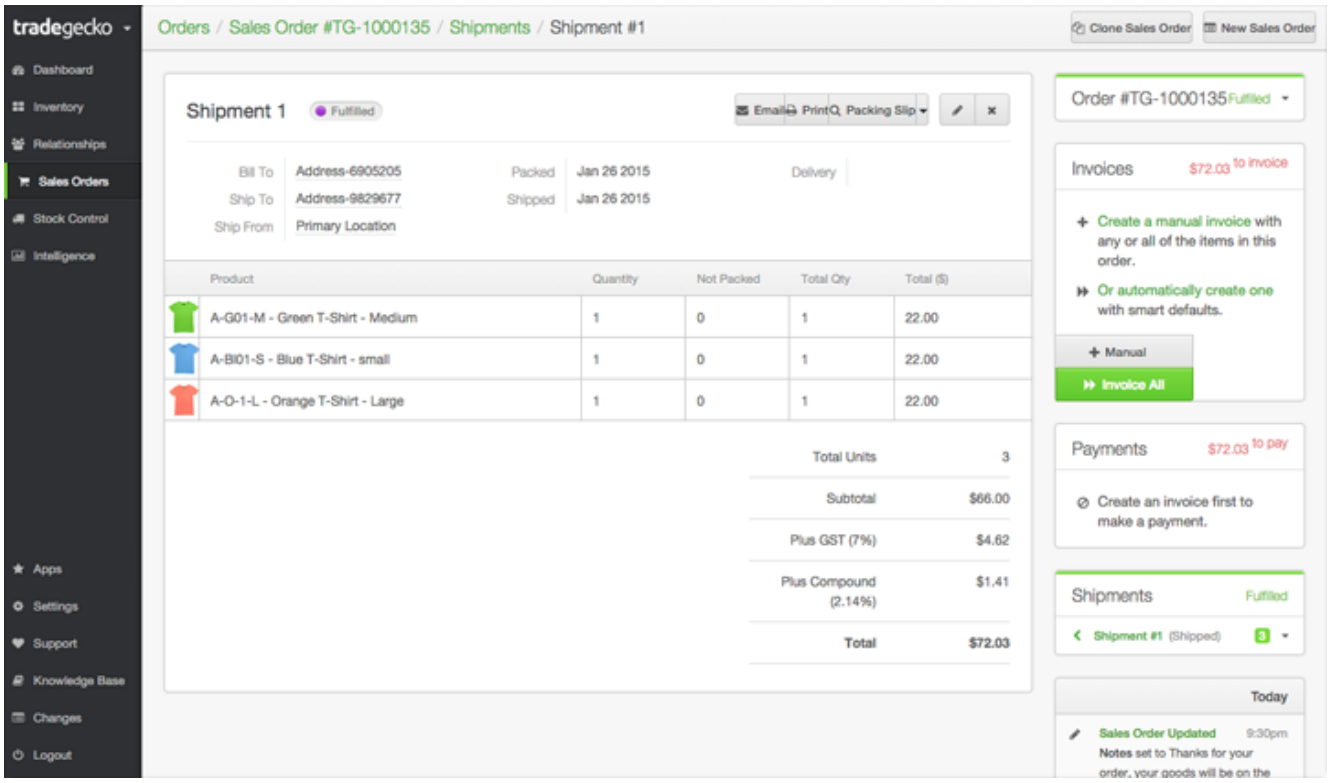


Рисунок 2.4.- TradeGecko

Основні функції TradeGecko включають:

— Управління запасами: TradeGecko дозволяє контролювати запаси продуктів у режимі реального часу. Ви можете відстежувати рівень запасів, вартість, серійні номери, специфікації та розташування кожного товару. Передбачені функції прибуття, відвантаження, руху запасів та інвентаризації.

— Управління замовленнями: програма дозволяє створювати замовлення клієнтів, відстежувати їхні статуси та керувати виконанням. Ви можете контролювати запаси, оцінювати терміни доставки, керувати виконанням замовлень і забезпечувати задоволення потреб клієнтів.

— Керування постачальниками: TradeGecko дозволяє створювати та керувати базою даних постачальників, включаючи контактну інформацію, умови доставки, історію замовлень і цінові угоди. Ви можете ефективно керувати відносинами з постачальниками, контролювати час доставки та оптимізувати процеси поставок.

— Управління виробництвом: TradeGecko забезпечує функціонал для планування та керування виробництвом. Ви можете створювати робочі накази, встановлювати розклади виробництва, контролювати використання матеріалів, стежити за прогресом виробництва та підтримувати ефективну роботу;

Ще одним представником є «SKULabs» - це хмарна платформа для управління запасами та виконання замовлень, розроблена спеціально для підприємств роздрібної торгівлі та електронної комерції. Він пропонує широкий спектр функціональних можливостей, які дозволяють ефективно керувати запасами, обробляти замовлення та покращувати операції з продажу (рис. 2.5).

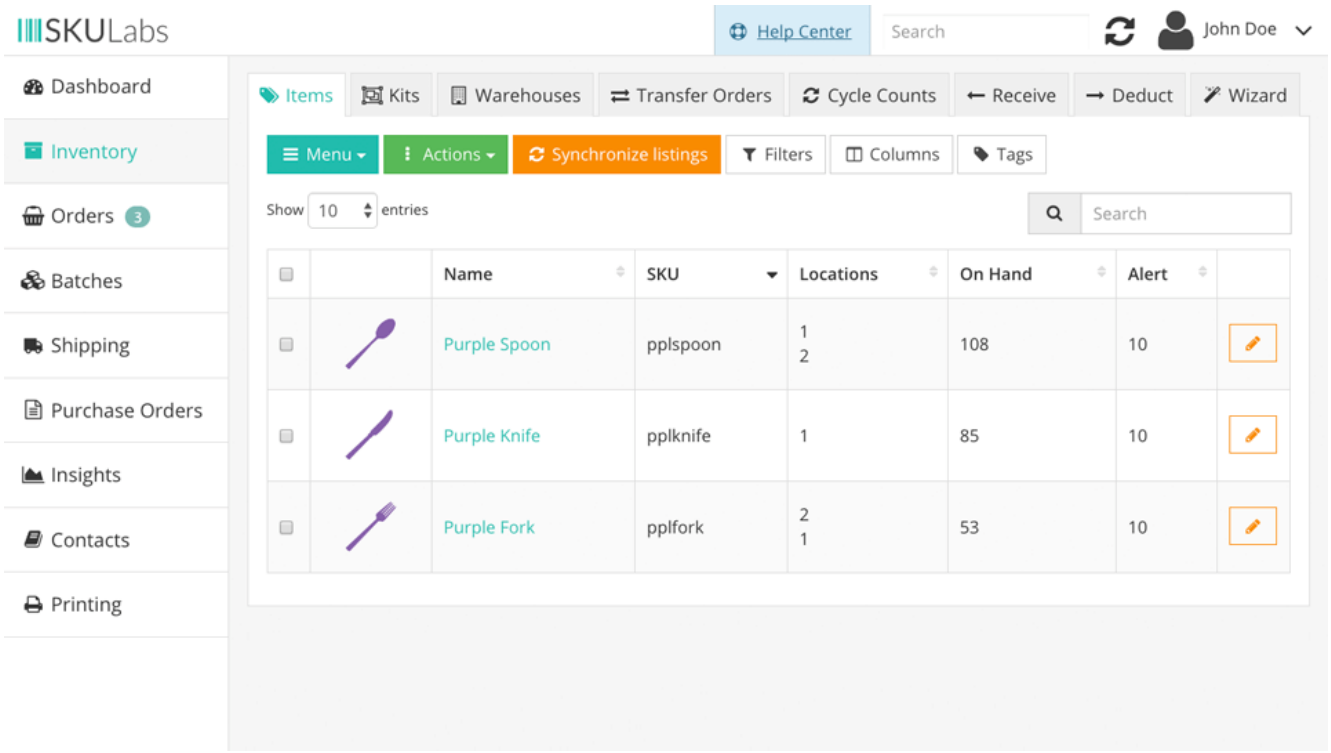


Рисунок 2.5 - SKULabs

Основні функції SKULabs включають:

— Управління запасами: SKULabs надає повний контроль над запасами товарів. Ви можете стежити за кількістю товарів на складі, вартістю запасів, рухами товарів та зберігати деталізовану інформацію про кожен SKU. Забезпечується відстеження запасів в реальному часі, сповіщення про низький рівень запасів та автоматичне поповнення.

— Обробка замовлень: SKULabs дозволяє легко обробляти замовлення з різних онлайн платформ, таких як Amazon, eBay, Shopify тощо, з однієї централізованої платформи. Ви можете автоматизувати процеси підтвердження, виконання та доставки замовлень, відстежувати статус замовлень в режимі реального часу та спрощувати процес виконання.

— Управління доставкою: SKULabs надає інтеграцію з провідними службами доставки, такими як UPS, FedEx, USPS та іншими. Ви можете встановлювати пріоритети доставки, генерувати етикетки та відстежувати відправлення. Платформа також дозволяє автоматично розраховувати вартість доставки та встановлювати правила відправлення.

— Управління складськими операціями: SKULabs пропонує широкий набір інструментів для оптимізації складських операцій. Ви можете встановлювати оптимальні розміщення товарів на складі, організовувати пікірні та пакувальні процеси, а також використовувати технології, такі як сканування штрих-кодів та радіочастотне ідентифікування (RFID), для точного відстеження товарів та ефективного управління запасами.

— Інтеграція з електронною комерцією: SKULabs надає можливості інтеграції з популярними платформами електронної комерції, такими як Shopify, Magento, WooCommerce та іншими. Це дозволяє автоматично синхронізувати товари, запаси, замовлення та клієнтську інформацію між SKULabs і вашими електронними магазинами, забезпечуючи єдиний пункт контролю та покращену ефективність;

Таблиця 2.1 – Порівняння існуючих застосунків

Застосунки	Fishbowl Inventory	Zoho Inventory	Odoo Inventory	TradeGecko	SKULabs	Warehouse Engine
Управління запасами	+	+	+	+	+	+

Управління постачальниками	+	+	+	+	+	+
Звітність і аналітика	+	+	+	+	+	+
Можливість розгорнути застосунок на своїх серверах	-	-	-	-	-	+
Можливість інтеграції з хмарними сервісами	-	-	-	-	-	+
Управління працівниками	-	+	+	-	-	+

Таблиця 2.1 – Порівняння існуючих застосунків

3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗАЦІЇ СКЛАДСЬКОГО ОБЛІКУ

3.1 Модель предметної галузі

Модель предметної галузі – це абстрактна модель, що описує певну галузь бізнесу. Наше дослідження зосереджене на складі, який безпосередньо пов'язаний з конкретними бізнес-сферами. Після аналізування цієї галузі, було побудовано модель «Складський облік» (рис. 3.1.1).

Рисунок 3.1.1 - Діаграма предметної галузі

3.2 Модель прецедентів

На основі аналізу існуючих рішень було зроблено вивід, що ПЗ повинно мати змогу швидко проводити різні види операцій з товарами. Так як у складу є і власник і працівник ми матимемо декілька варіантів використання ПЗ. Перший можна побачити на Рис. 3.2.1:

Рисунок 3.2.1 - Модель прецедентів №1 – Admin

Актором тут виступає власник складу, який може виконувати дії за допомогою UI. Також власник має більше функціоналу ніж звичайний працівник складу.

Наступна діаграма показує можливості звичайного користувача, тобто працівника складу (Рис. 3.2.2), як ви можете побачити відрізняються вони тим, що звичайний користувач не має ніякого доступу до керування користувачами.

Рисунок 3.2.2 - Модель прецедентів №2 – User

3.3 Нефункціональні вимоги. Бачення з контекстною діаграмою

Контекстна діаграма дозволяє описати зовнішні властивості системи (рис. 3.3.1).

Рисунок 3.1.1 - Контекстна діаграма

На діаграмі зображено об’єкт проектування – сервіс для автоматизації складського обліку. Елементами які проводять взаємодію із об’єктом проектування є певні групи користувачів, наприклад: власник, адміністратор та користувач складу(працівник). Також можна побачити різні потоки даних для елементів системи.

Наприклад можна побачити що для адміністратора системи вхідними даними є поточна конфігурації системи та інформація про помилки.

3.4 Проектування діяльності. Діаграма діяльності

На діаграмах зображених нижче можна побачити логіку роботи сервера на прикладі такого функціоналу як експорт товару (Рис 3.4.1).

Рисунок 3.4.1 – Діаграма діяльності для функції «Екпорт товару»

На наступній діаграмі можна побачити логіку пошуку (Рис 3.4.2) яка реалізована у даному додатку.

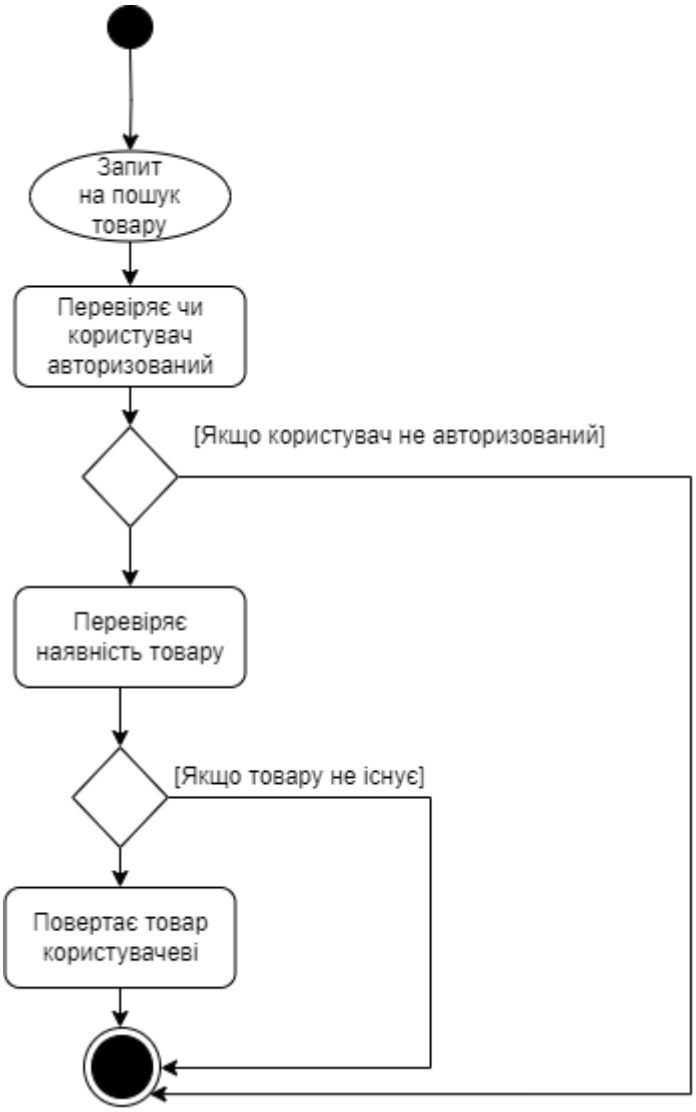


Рисунок 3.4.2 - Діаграма діяльності для функції «Пошук товару»

На останній діаграмі діяльності можна побачити логіку авторизації (Рис. 3.4.3):

Рис 3.4.3 - Діаграма діяльності для функції «Авторизація»

3.5 Проектування послідовності викликів методів та взаємодії об'єктів. Діаграми послідовностей

Були використані діаграми послідовності для зображення внутрішньої логіки роботи програми у відношенні до наступних функцій: експорт товару (рис. 3.5.1), видалення товару (рис. 3.5.2), пошук товару (рис. 3.5.3).

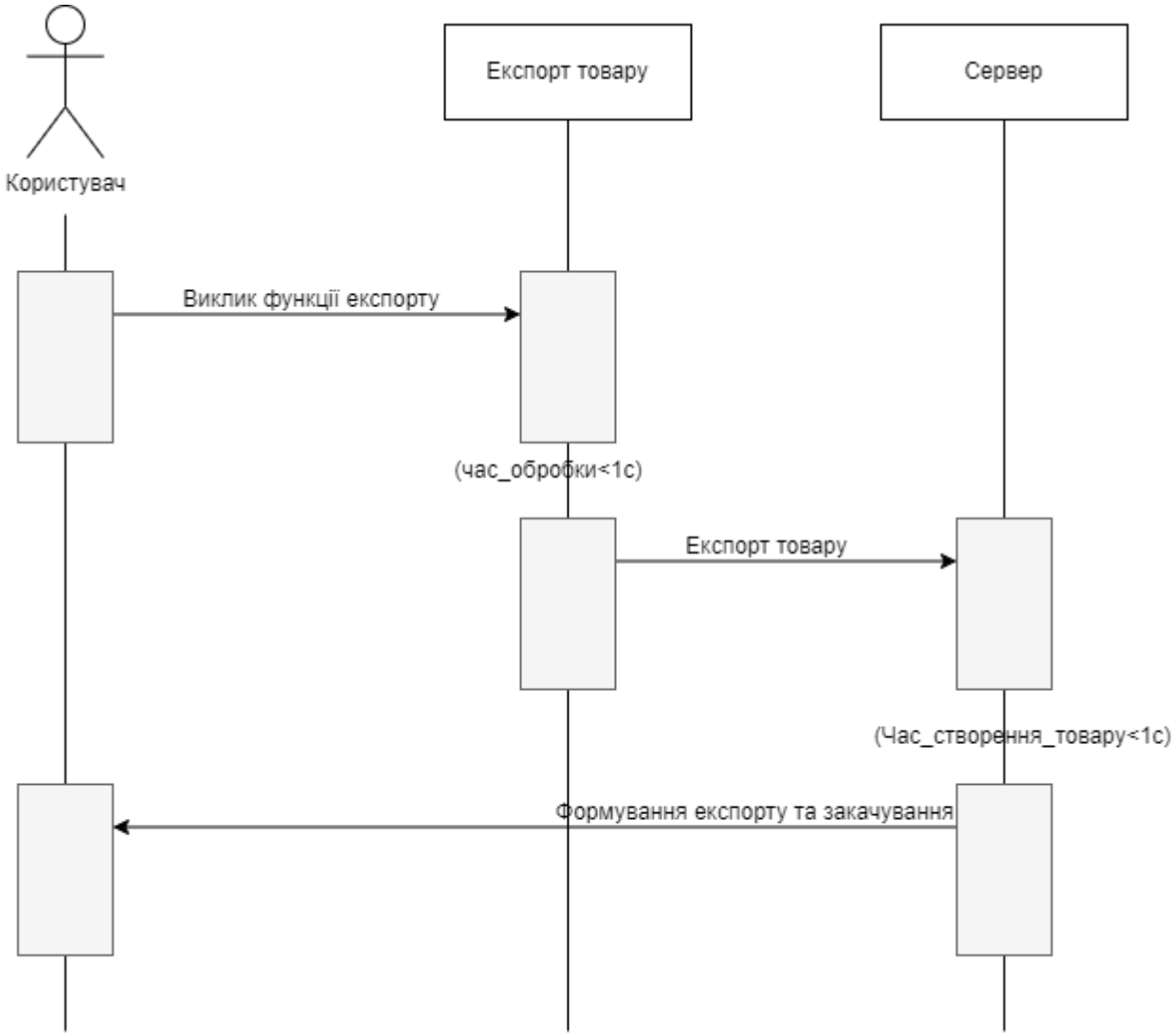


Рисунок 3.5.1 - Діаграма послідовності для функції «Експорт товару»

З діаграм можна зрозуміти, що середній час обробки інформації на етапі передачі потоків даних становить менше ніж 1 секунда. Зменшення часу обробки може бути здійснене завдяки покращення характеристик серверу на якому виконується застосунок.

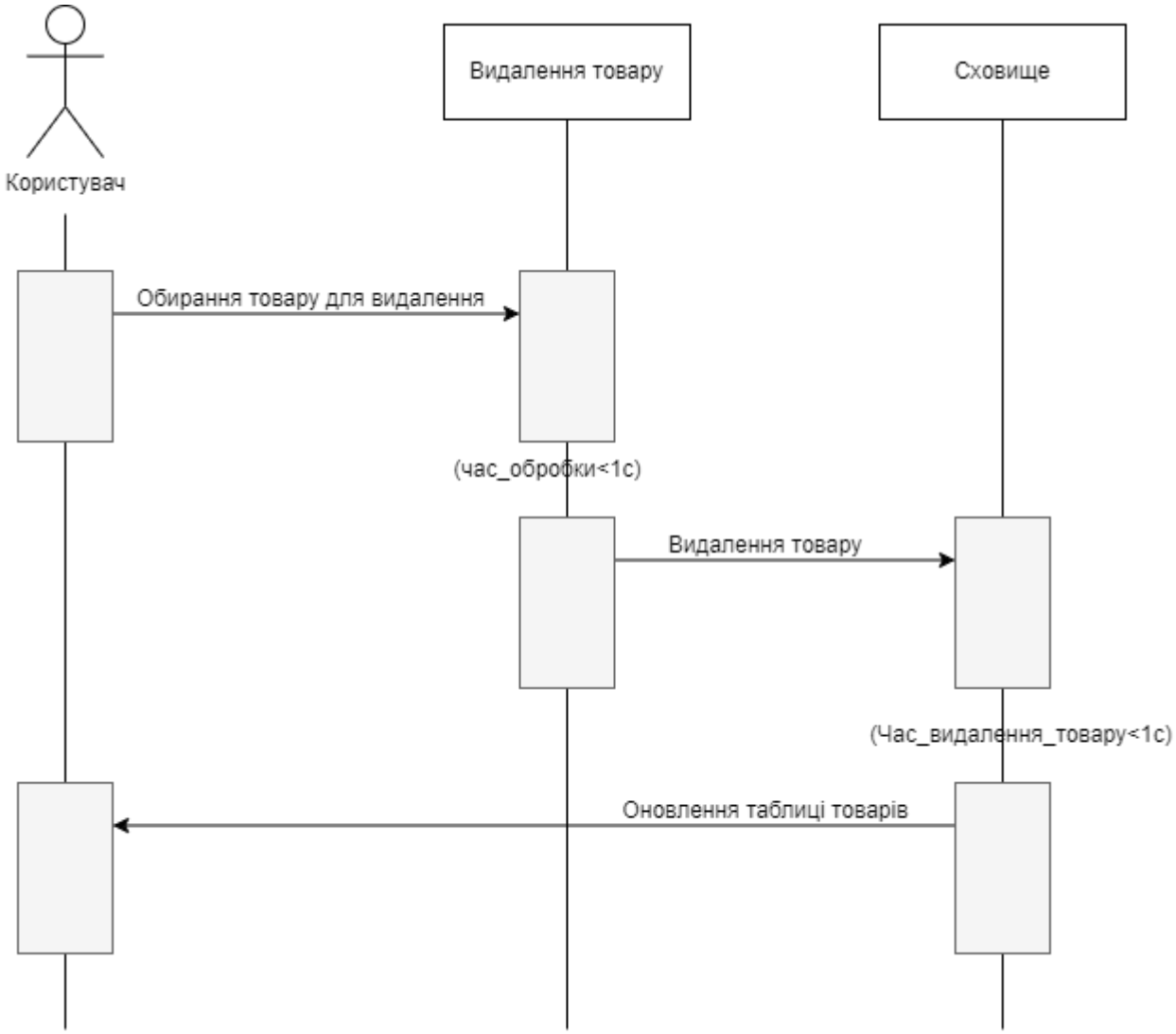


Рис 3.5.2 - Діаграма послідовності для функції «Видалення товару»

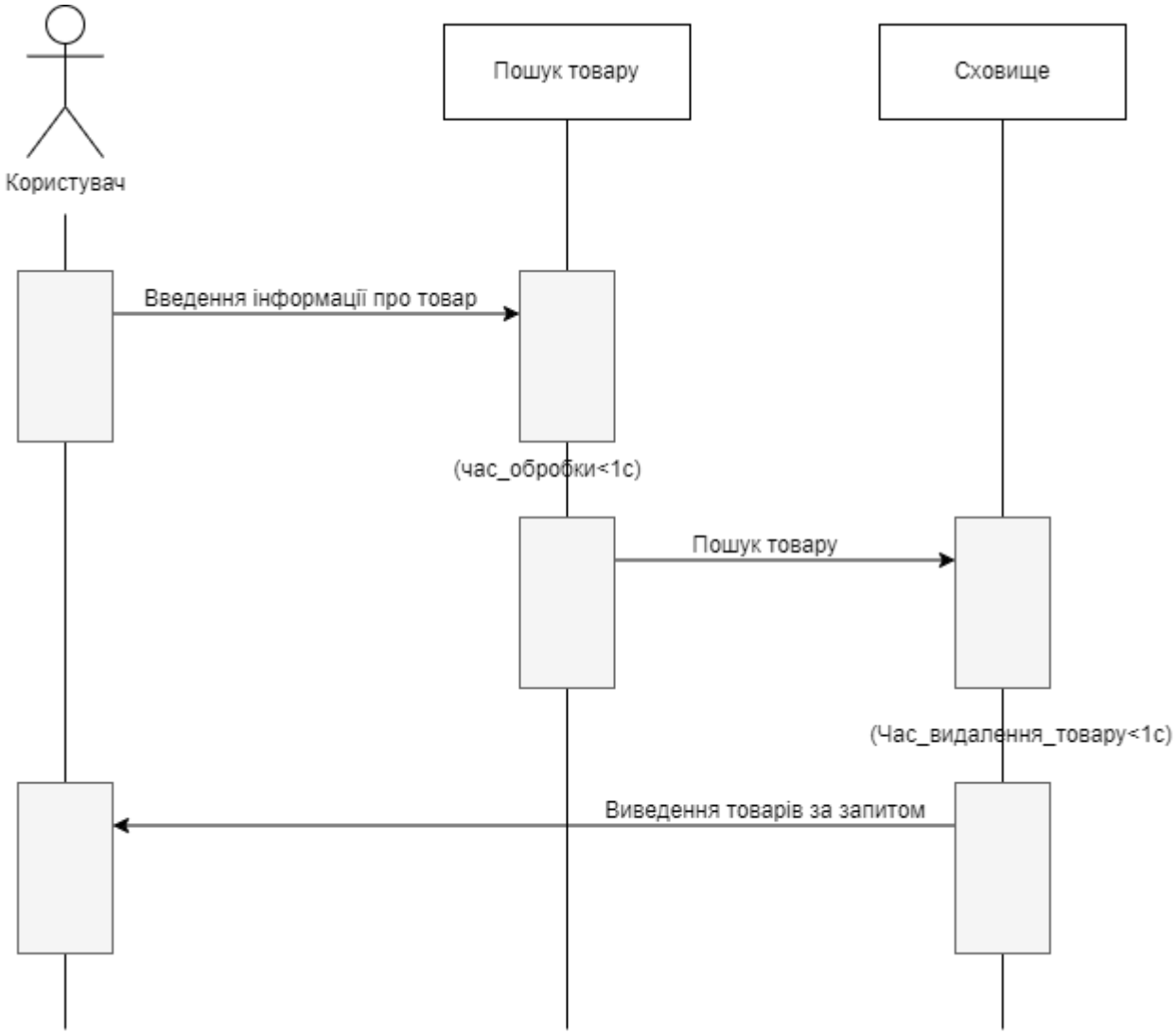


Рис 3.5.3 - Діаграма послідовності для функції "Пошук товару"

Як можна побачити швидкість пошуку займає менше однієї секунди, але це дуже сильно залежить від кількості товарів у базі даних.

3.6 Проектування послідовності. Діаграми станів

Діаграми станів для операцій над товарами (рис. 3.6.1, 3.6.2, 3.6.3).

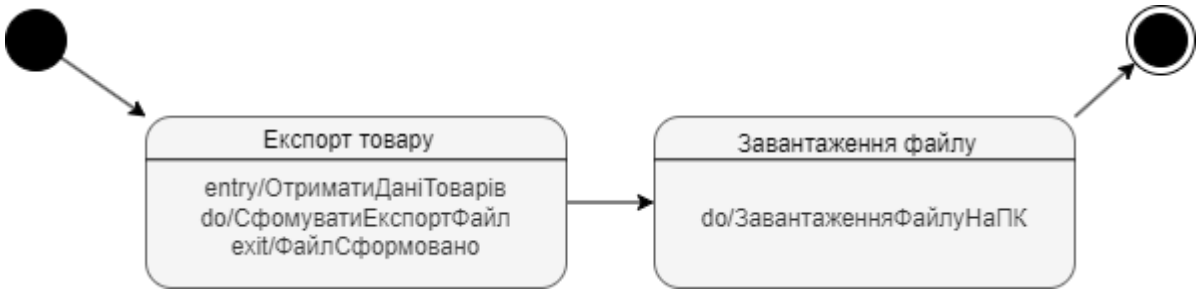


Рисунок 3.6.1 - Діаграма станів для функції "Екпорт товару"

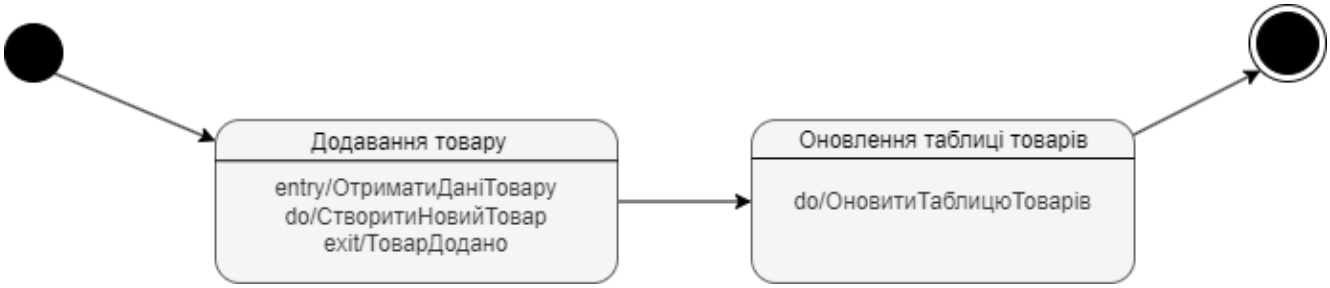


Рисунок 3.6.2 - Діаграма станів для функції "Додавання товару"

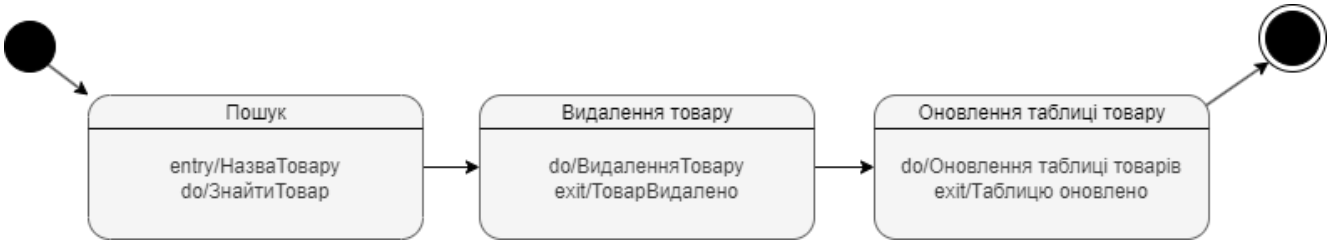


Рисунок 3.6.3 - Діаграма станів для функції "Видалення товару"

3.7 Діаграми комунікацій (кооперацій)

Нижче наведені діаграми комунікацій функцій пов’язаних з операціями керування товаром (рис. 3.7.1, 3.7.2, 3.7.3).

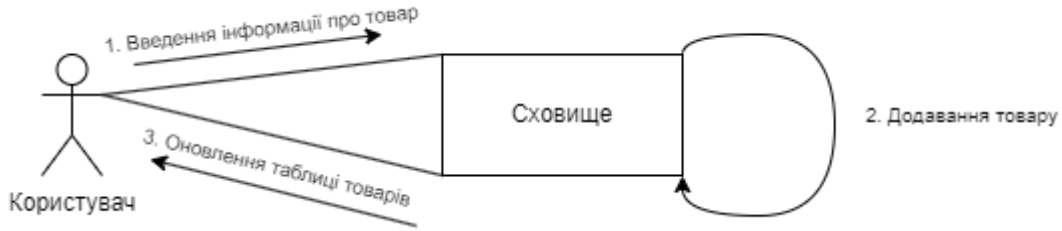


Рисунок 3.7.1 - Діаграма комунікації для функції "Додавання товару"

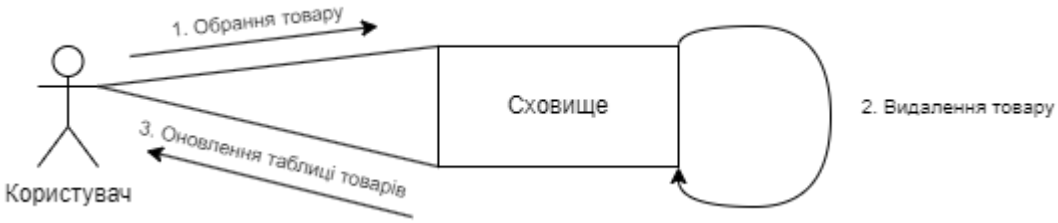


Рисунок 3.7.1 - Діаграма комунікації для функції "Видалення товару"

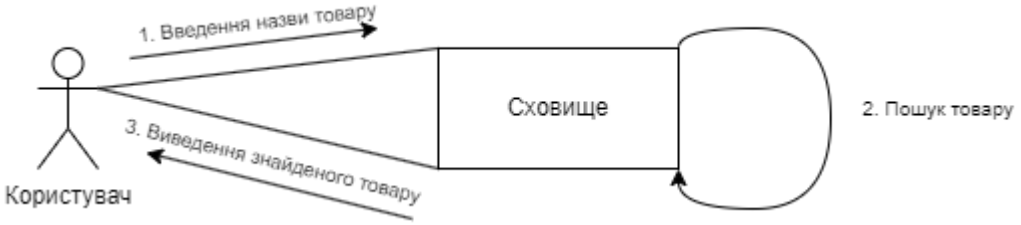


Рисунок 3.7.1 - Діаграма комунікації для функції "Пошук товару"

3.8 Схема БД

Далі буде наведена схема бази даних (рис. 3.8.1).

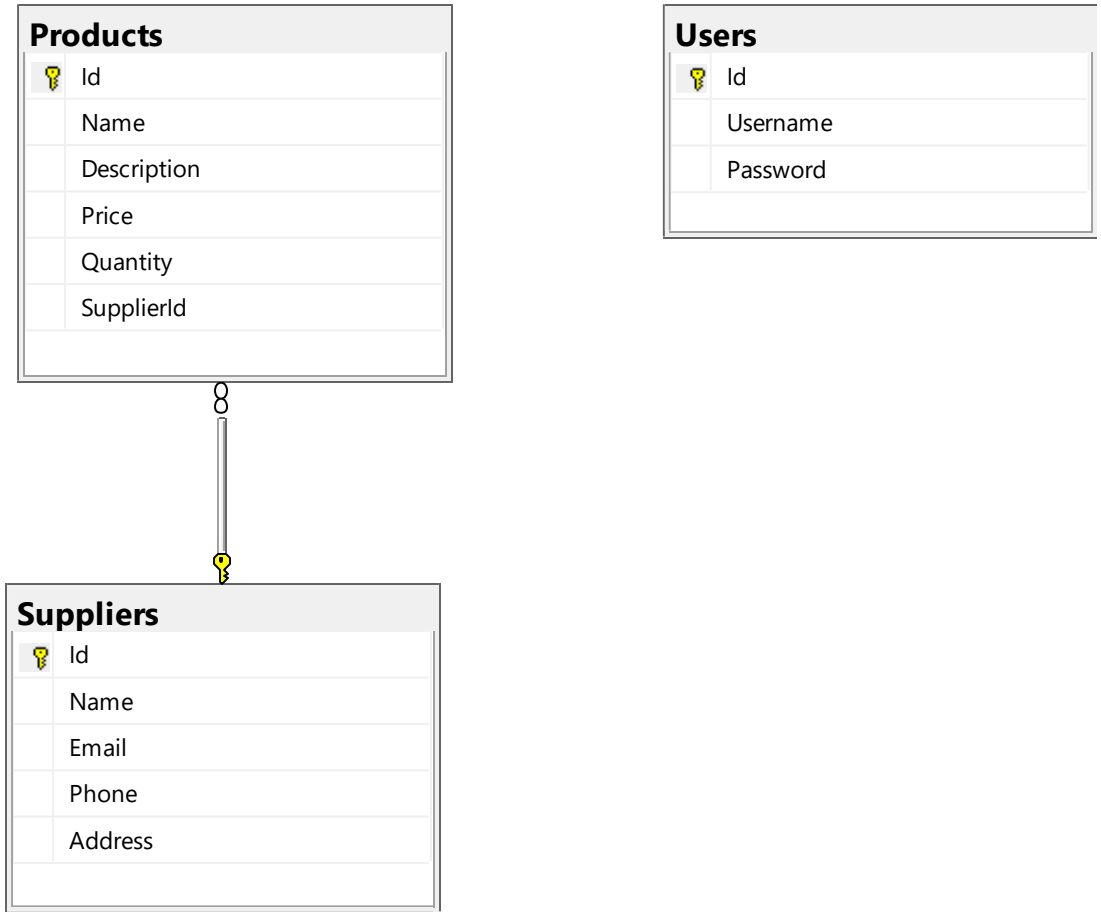


Рисунок 3.8.1 – Схема БД

4 РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗАЦІЇ СКЛАДСЬКОГО ОБЛІКУ

4.1 Рефлексія

Рефлексія – являється механізмом, що дає можливість досліджувати, аналізувати структуру, функціональність, поведінку типів даних у коді застосунку під час виконання программи. Цей потужний та важливий інструмент надає програмісту доступ до метаданих програми, таких як класи, інтерфейси, методи, властивості та інше. Ця складова C# дозволяє створювати гнучкий функціонал та не прив'язуватись до типів даних. На скріншоті (рис. 4.1.1) можна побачити клас ExcelExportService який використовує рефлексію для отримання назви для колонок ексель через атрибути класу, наприклад класу товару (рис 4.1.2).

```
1 reference
public class ExcelExportService : IExcelExportService
{
    5 references
    public byte[] ExportToExcel<T>(IEnumerable<T> data, string sheetName = "Sheet1")
    {
        using (var workbook = new XLWorkbook())
        {
            var worksheet = workbook.Worksheets.Add(sheetName);
            var properties = typeof(T).GetProperties();

            var displayNames = properties.Select(prop => prop.GetCustomAttributes(typeof(DisplayNameAttribute), false)
                                .Cast<DisplayNameAttribute>().FirstOrDefault()?.DisplayName ?? prop.Name).ToList();
            for (int i = 0; i < properties.Length; i++)
            {
                worksheet.Cell(1, i + 1).Value = displayNames[i];
            }

            for (int i = 0; i < data.Count(); i++)
            {
                var item = data.ElementAt(i);
                for (int j = 0; j < properties.Length; j++)
                {
                    worksheet.Cell(i + 2, j + 1).Value = properties[j].GetValue(item)?.ToString();
                }
            }

            using (var stream = new MemoryStream())
            {
                workbook.SaveAs(stream);
                return stream.ToArray();
            }
        }
    }
}
```

Рисунок 4.1.1 – Клас ExcelExportService

```
22 references
public class ProductViewModel
{
    10 references
    public int? Id { get; set; }

    [DisplayName("Назва")]
    [Required(ErrorMessage = "Поле 'Назва' є обов'язковим.")]
    [StringLength(50, MinimumLength = 3, ErrorMessage = "Поле 'Назва' повинно містити від 3 до 50 символів.")]
    12 references
    public string Name { get; set; }

    [DisplayName("Опис")]
    [Required(ErrorMessage = "Поле 'Опис' є обов'язковим.")]
    [StringLength(200, MinimumLength = 10, ErrorMessage = "Поле 'Опис' повинно містити від 10 до 200 символів.")]
    11 references
    public string Description { get; set; }

    [DisplayName("Кількість")]
    [Required(ErrorMessage = "Поле 'Кількість' є обов'язковим.")]
    [Range(0, int.MaxValue, ErrorMessage = "Поле 'Кількість' повинно бути додатнім числом.")]
    11 references
    public int Quantity { get; set; }

    [DisplayName("Ціна")]
    [Required(ErrorMessage = "Поле 'Ціна' є обов'язковим.")]
    [Range(0, double.MaxValue, ErrorMessage = "Поле 'Ціна' повинно бути додатнім числом.")]
    11 references
    public decimal Price { get; set; }

    [DisplayName("Постачальник")]
    [Required(ErrorMessage = "Поле 'Постачальник' є обов'язковим.")]
    13 references
    public int SupplierId { get; set; }
}
```

Рисунок 4.1.2 – Клас ExcelExportService

Якщо подивитись на те як дістаються display names ExcelExportService(Рис. 4.1.1) можна побачити, що з якогось класу дістаються поля і з них інформація про кастомні атрибути наприклад атрибут DisplayName з ProductViewModel (Рис. 4.1.2.)

4.2 Асинхронні операції

Така потужна мова програмування як C# підтримує такий потужний інструмент як асинхронне програмування. Асинхронне програмування у C# здійснюється завдяки двум ключовим словам таким як async та await та дозволяє виконувати завдання асинхронно, тобто не блокуючи основний потік. Ключове слово async вказує, що метод може бути викликаний асинхронно, а оператор await викликикає цей метод асинхронно у іншому потоці, що надає змогу оптимізувати застосунок.

При асинхронному виклику метода із ТР береться потік, який і буде супроводжувати викликаємий метод, головний бонус цього являється тим, що як тільки потік виконав те що мусив, наприклад HTTP виклик, він може повернутися назад до ТР та обслужити інші методи, а по завершенню виконання попереднього метода його може обслужити інший потік із ТР. Приклад асинхронного виклику метода звертання до бази даних можна побачити на скріншоті (Рис 4.2.1)

```
public class UserService : IUserService
{
    private readonly WarehouseEngineDbContext _dbContext;

    0 references
    public UserService(WarehouseEngineDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    2 references
    public async Task<bool> FindUserAsync(string username, string password)
    {
        return await _dbContext.Users.FirstOrDefaultAsync(x => x.Username == username && x.Password == password) == null ? false : true;
    }
}
```

Рисунок 4.2.1 – Приклад асинхронного виклику запиту до БД

При створенні асинхронних методів непомітно від ока програміста створюється машина станів, яка і відповідає за розумне регулювання потоків, приклад цьої машини можна побачити через дизасемблювання C# коду (Рис 4.2.2)

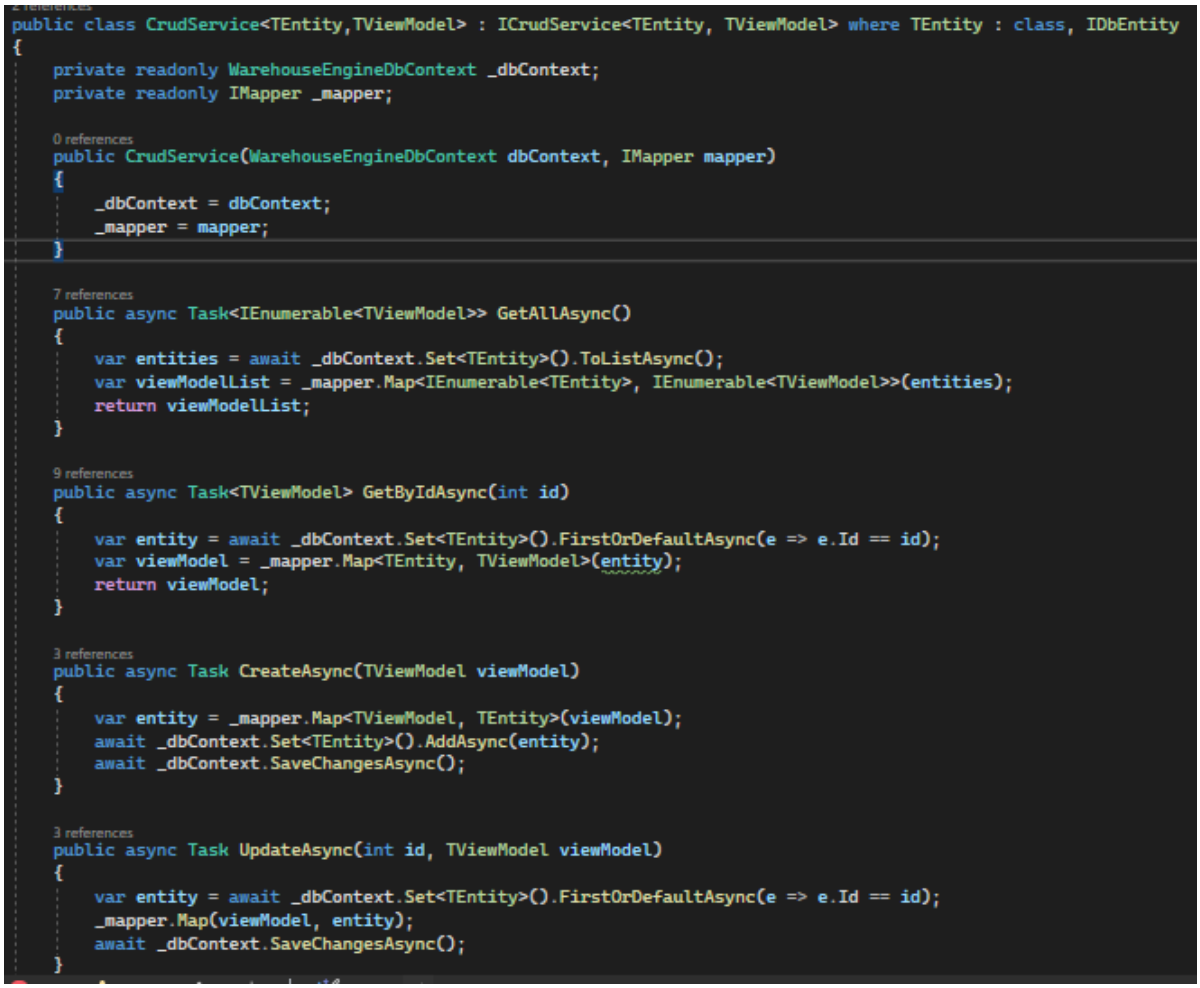
```
using System.Diagnostics;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;
using EfCoreDataAccessLayer;
using EfCoreDataAccessLayer.Models;
using WarehouseEngineBusinessLayer.Interfaces;

namespace WarehouseEngineBusinessLayer.Services
{

    [NullableContext(, ]
    [Nullable()]
    public class UserService : IUserService
    {

        [CompilerGenerated]
        private sealed class <>c__DisplayClass2_0
        {

            [Nullable()]
            public string password;
            [Nullable()]
            public string username;

            public <>c__DisplayClass2_0()
            {
            }

        } // class <>c__DisplayClass2_0

        [CompilerGenerated]
        private sealed class <FindUserAsync>d__2 : IAsyncStateMachine
        {

            public int <>1__state;
            [Nullable()]
            public UserService <>4__this;
            [Nullable()]
            private UserService.<>c__DisplayClass2_0 <>8__1;
            [Nullable()]
            private User <>s__2;
            [Nullable()]
            public AsyncTaskMethodBuilder<bool> <>t__builder;
            [Nullable(,  = ,  = )]
            private TaskAwaiter<User> <>u__1;
            [Nullable()]
            public string password;
            [Nullable()]
            public string username;

            public <FindUserAsync>d__2()
            {
            }

            void IAsyncStateMachine.MoveNext()
            {
                // trial
            }

            [DebuggerHidden]
            void IAsyncStateMachine.SetStateMachine(IAsyncStateMachine stateMachine)
            {
            }

        } // class <FindUserAsync>d__2

        private readonly WarehouseEngineDbContext _dbContext;

        public UserService(WarehouseEngineDbContext dbContext)
        {
            _dbContext = dbContext;
        }

        [AsyncStateMachine(typeof(WarehouseEngineBusinessLayer.Services.UserService.<FindUserAsync>d__2))]
        [DebuggerStepThrough]
        public Task<bool> FindUserAsync(string username, string password)
        {
            UserService.<FindUserAsync>d__2 <findUserAsync>d__2 = new UserService.<FindUserAsync>d__2();
            <findUserAsync>d__2.<>t__builder = Create();
            <findUserAsync>d__2.<>4__this = this;
            <findUserAsync>d__2.username = username;
            <findUserAsync>d__2.password = password;
            <findUserAsync>d__2.<>1__state = -1;
            <findUserAsync>d__2.<>t__builder.Start<UserService.<FindUserAsync>d__2>(ref <findUserAsync>d__2);
            return <findUserAsync>d__2.<>t__builder.get_Task();
        }

    } // class UserService

}
```

Рисунок 4.2.2 – Дизасембльований UserService

На скріншоті (Рис. 4.2.2) можна побачити як компілятор непомітно від ока програміста згенерував код машини станів, яка і виконує функції курування потоками

4.3 Лямбда-вирази

У мові програмування C# є такий зручний інструмент як лямбда-вирази, це короткий синтаксис який дозволяє оголошувати анонімні функції або методи без оголошення окремого коду, вони надають зручний спосіб передачі функціональності як параметрів методів та використовувати їх у різноманітних операціях з колекціями. Складаються лямбда вирази із простого оператора "=>" та самого виразу або блоку коду. Найчастіше лямбда вирази використовуються у зв'язку з LINQ для лінійних операцій над колекціями у таких методах як Where, FirstOrDefault, OrderBy, Select та інших. Такий приклад використання можна побачити у класі CrudService (Рис. 4.3.1), який виконує різні операції над колекціями з інформацією з БД.



```
public class CrudService<TEntity,TViewModel> : ICrudService<TEntity, TViewModel> where TEntity : class, IDbEntity
{
    private readonly WarehouseEngineDbContext _dbContext;
    private readonly IMapper _mapper;

    0 references
    public CrudService(WarehouseEngineDbContext dbContext, IMapper mapper)
    {
        _dbContext = dbContext;
        _mapper = mapper;
    }

    7 references
    public async Task<IEnumerable<TViewModel>> GetAllAsync()
    {
        var entities = await _dbContext.Set<TEntity>().ToListAsync();
        var viewModelList = _mapper.Map<IEnumerable<TEntity>, IEnumerable<TViewModel>>(entities);
        return viewModelList;
    }

    9 references
    public async Task<TViewModel> GetByIdAsync(int id)
    {
        var entity = await _dbContext.Set<TEntity>().FirstOrDefaultAsync(e => e.Id == id);
        var viewModel = _mapper.Map<TEntity, TViewModel>(entity);
        return viewModel;
    }

    3 references
    public async Task CreateAsync(TViewModel viewModel)
    {
        var entity = _mapper.Map<TViewModel, TEntity>(viewModel);
        await _dbContext.Set<TEntity>().AddAsync(entity);
        await _dbContext.SaveChangesAsync();
    }

    3 references
    public async Task UpdateAsync(int id, TViewModel viewModel)
    {
        var entity = await _dbContext.Set<TEntity>().FirstOrDefaultAsync(e => e.Id == id);
        _mapper.Map(viewModel, entity);
        await _dbContext.SaveChangesAsync();
    }
```

Рисунок 4.3.1 – Клас CrudService

На скріншоті (Рис 4.3.1) у методі GetByIdAsync на першій строці метода можна побачити як елегантно та легко передаються критерії пошуку потрібної інформації, тобто у виклику FirstOrDefaultAsync передається лямбда вираз, який обумовлює собою пошук моделі з певним ID.

4.4 Атрибути

Ще один зручний інструмент C# - атрибути. Атрибути являють собою метадані, які можуть бути використані для забезпечення спеціальної під час компіляції або під час виконання самого коду. Такий функціонал дозволяє доповнювати члени класу спеціальною інформацією яка потім може бути використана за допомгою рефлексії, вони можуть бути накладені на такі члени як

методи, властивості, поля, параметри, зборки, класи тощо. Оголошуються атрибути завдяки квадратним скобкам "[]" у які поміщається сама назва атрибути та параметри якщо необхідно. Мова програмування C# дозволяє створення власних атрибутів, які являють собою клас, який успадковується від класу Attribute. Атрибути можуть бути використані у зв'язці з різноманітними бібліотеками таких як Blazor та RazorPages, таким прикладом є атрибут Required у ProductViewModel(Рис.4.4.1)

```
public class ProductViewModel
{
    10 references
    public int? Id { get; set; }

    [DisplayName("Назва")]
    [Required(ErrorMessage = "Поле 'Назва' є обов'язковим.")]
    [StringLength(50, MinimumLength = 3, ErrorMessage = "Поле 'Назва' повинно містити від 3 до 50 символів.")]
    12 references
    public string Name { get; set; }

    [DisplayName("Опис")]
    [Required(ErrorMessage = "Поле 'Опис' є обов'язковим.")]
    [StringLength(200, MinimumLength = 10, ErrorMessage = "Поле 'Опис' повинно містити від 10 до 200 символів.")]
    11 references
    public string Description { get; set; }

    [DisplayName("Кількість")]
    [Required(ErrorMessage = "Поле 'Кількість' є обов'язковим.")]
    [Range(0, int.MaxValue, ErrorMessage = "Поле 'Кількість' повинно бути додатнім числом.")]
    11 references
    public int Quantity { get; set; }

    [DisplayName("Ціна")]
    [Required(ErrorMessage = "Поле 'Ціна' є обов'язковим.")]
    [Range(0, double.MaxValue, ErrorMessage = "Поле 'Ціна' повинно бути додатнім числом.")]
    11 references
    public decimal Price { get; set; }

    [DisplayName("Постачальник")]
    [Required(ErrorMessage = "Поле 'Постачальник' є обов'язковим.")]
    13 references
    public int SupplierId { get; set; }
}
```

Рисунок 4.4.1 – Клас ProductViewModel

На скріншоті (Рис 4.4.1) можна побачити атрибут Required та передаваймий параметр ErrorMessage, таким чином UI фреймворк RazorPages виходячи із метаданих буде розуміти, що ці поля мають бути заповнені та не дозволить продовжити операцію доки вони не будуть заповнені, якщо спроба продовжити операцію з пустими полями буде виконана, цей фреймворк додасть червоні надписи під полями вводу згідно з параметром ErrorMessage у атрибуті (Рис. 4.4.2).

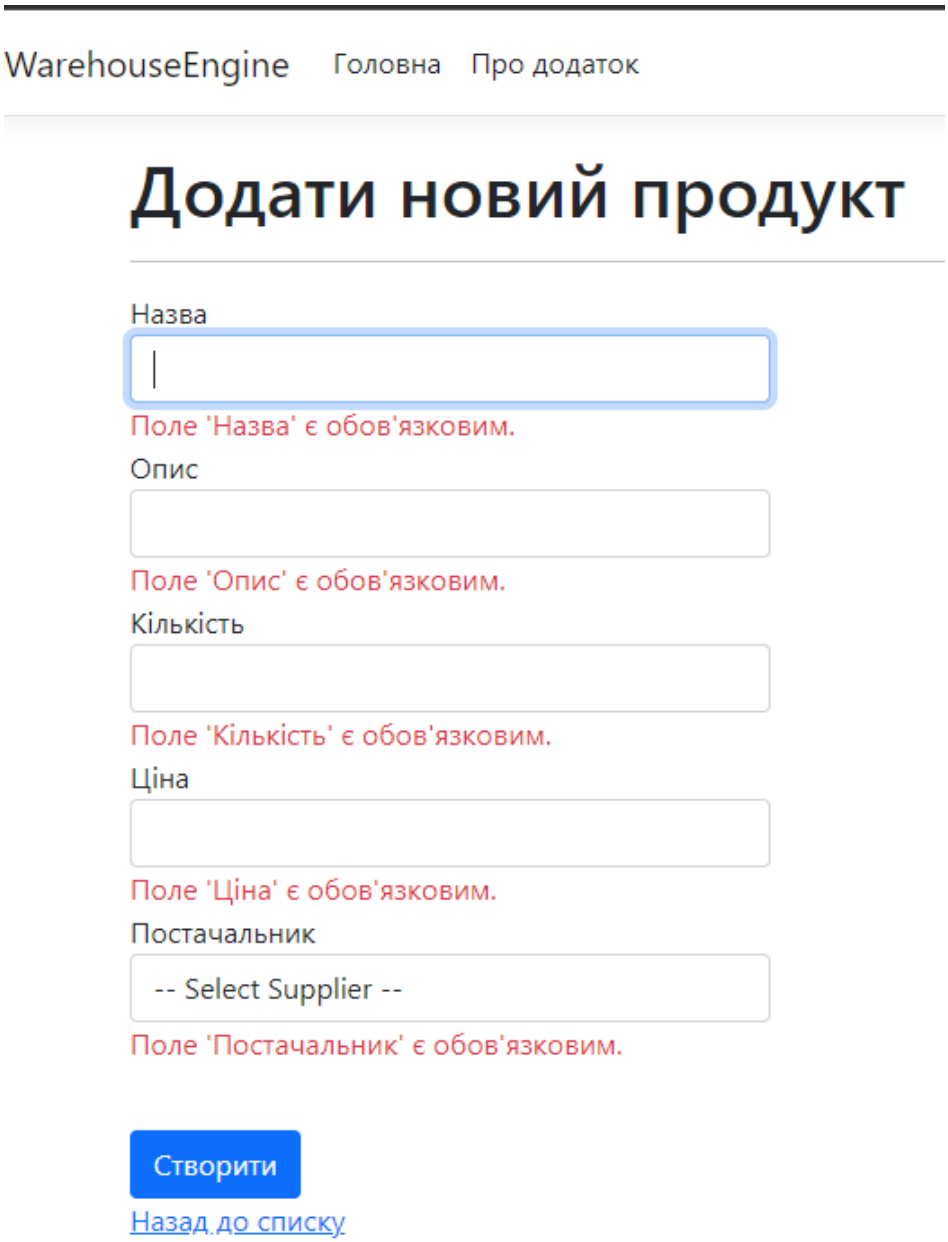


Рисунок 4.4.2 – Приклад використання атрибутів у RazorPages

4.5 JSON-конфігурація

У ASP.NET Core конфігурації у форматі JSON використовуються дуже часто, адже це дозволяє динамічно підставляти різні конфігурації без змін основного коду. Такий метод конфігурування може бути використаний для налаштування підключення до бази даних, логування тощо. У ASP.NET Core це робиться завдяки файлу appsettings.json з якого і беруться конфігураційні JSON-об’єкти. Введення

такого способу конфігурацій робиться завдяки класу WebApplicationBuilder який і має доступ до appsettings.json. Таким чином можна налаштувати строку підключення до БД (Рис 4.5.1) та потім використати цю секцію у коді (Рис 4.5.2).

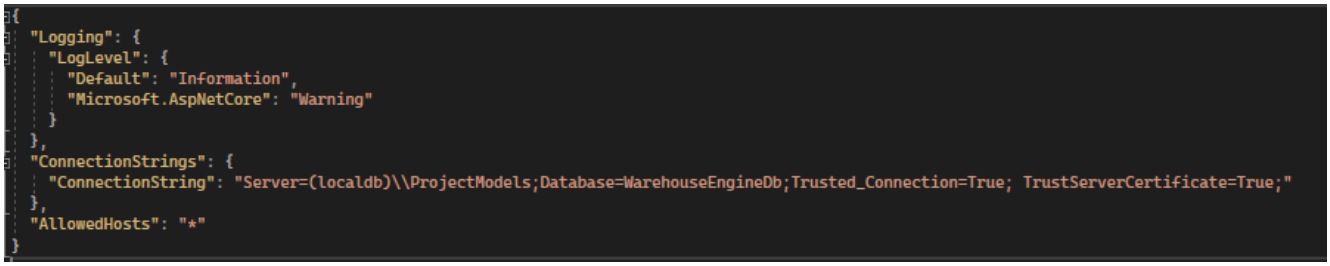


Рисунок 4.5.1 – Задання строки підключення у appsettings.json

Рисунок 4.5.2 – Використання appsettings.json у коді

Звертатися до такої конфігурації можна завдяки методу GetSection (Рис. 4.5.2), де у параметрі передається назва зовнішнього JSON-об’єкту та слідом у квадратних скобках назва поля у цьому об’єкті.

4.6 Razor Pages

Razor Pages це технологія завдяки який можна вбудовувати C# код у HTML сторінку, тобто код вмонтовується прямо у RazorPages(HTML) розмітку. Такий підхід створення веб-застосунків дозволяє не створювати клієнт та сервер як різні

застосунки, а навпаки поєднувати код клієнту та серверу. Таким чином веб-застосунки можуть бути розроблені набагато швидше.

Такий підхід дозволяє не використовувати API, як у традиційних веб-застосунках, де сервер відправляє JSON-об’єкти на клієнт. Все працює завдяки поєднанню з ASP.NET Core фреймворком який виступає у ролі серверу, тож у такому підході C# об’єкти можуть бути передані одразу у Razor Pages представлення (Рис. 4.6.1) і там бути оброблені, завдяки цьому інструменту, який обумовлює собою використання C# прямо у HTML розмітці (Рис 4.6.2).

```
public class ProductController : Controller
{
    private readonly ICrudService<Product, ProductViewModel> _productService;
    private readonly ICrudService<Supplier, SupplierViewModel> _supplierService;
    private readonly IExcelExportService _excelExportService;
    const string exportAll = "AllProductExport";
    const string exportPart = "PartProductExport";
    const string excelFormat = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";

    0 references
    public ProductController(ICrudService<Product, ProductViewModel> productService, ICrudService<Supplier, SupplierViewModel> supplierService, IExcelExportService excelExportService)
    {
        _productService = productService;
        _supplierService = supplierService;
        _excelExportService = excelExportService;
    }

    0 references
    public async Task<IActionResult> Index(string searchString, int? pageNumberParam)
    {
        ViewData["CurrentFilter"] = searchString;

        var product = await _productService.GetAllAsync();
        ViewBag.supService = _supplierService;

        if (!String.IsNullOrEmpty(searchString))
        {
            product = product.Where(d => d.Name.Contains(searchString));
        }

        int pageSize = 10;
        int pageNumber = (pageNumberParam ?? 1);

        return View(await PaginatedList<ProductViewModel>.CreateAsync(product, pageNumber, pageSize));
    }
```

Рисунок 4.6.1 – Передавання C# об’єкту до RazorPages представлення

```
<div class="d-flex justify-content-center">
    <nav aria-label="Page navigation">
        <ul class="pagination">
            @if (Model.HasPreviousPage)
            {
                <li class="page-item">
                    <a class="page-link" href="@Url.Action("Index", new { pageNumberParam = Model.PageIndex - 1, searchString = ViewData["CurrentFilter"] })">Previous</a>
                </li>
            }
            @if (Model.TotalPages <= 20)
            {
                @for (int i = 1; i <= Model.TotalPages; i++)
                {
                    if (i == Model.PageIndex)
                    {
                        <li class="page-item active">
                            <span class="page-link">@i</span>
                        </li>
                    }
                    else
                    {
                        <li class="page-item">
                            <a class="page-link" href="@Url.Action("Index", new { pageNumberParam = i, searchString = ViewData["CurrentFilter"] })">@i</a>
                        </li>
                    }
                }
```

Рисунок 4.6.2 – Вмонтований C# у HTML розмітку

Як можна побачити у методі Index у класі ProductController (Рис. 4.6.1), останньою строчкою повертається метод результат метода View із результатом іншого метода у середині, який являється C# об’єктом, завдяки методу View сам C# об’єкт відправиться у відповідний файл з розширенням .cshtml який і є razor pages файлом. Як можна побачити на скріншоті (Рис. 4.6.2) HTML розмітка може бути легко доповнена C# логікою завдяки оператору, так і поєднується статичний та динамічний контент завдяки цім двум фреймворкам.

4.7 Entity Framework Core

EF Core – являється легкою кроссплатформовою ORM бібліотекою, яка надає можливість отримати доступ до бази даних безспосередньо із C# коду. Цей фреймворк дозволяє мапування із бази даних на об’єкти C# і автоматично генерує SQL запити.

EF Core дозволяє пришвидшити розробку завдяки тому, що розробник може не писати SQL запити, вони будуть згенеровані автоматично цім фреймворком. Завдяки цьому фреймворку розробка стає більш гнучкою так, як більше немає прив’язки до однієї єдиної БД, все що змінюється це строка підключення, весь код який націлений на взаємодію із однією БД залишиться працюючим і для іншої БД, тобто написавши код один раз він може бути використаний з будь-якою базою даних. Також функціонал цього фреймворку дозволяє користуватися підходом CodeFirsrt, за допомогою якого можна не створювати таблиці БД перед тим як писати сам код та моделі на C#, у такому підході розробник пише класи-моделі на основі яких генеруються таблиці у БД. EF Core дозволяє зручно на гнучко сконфігурувати таблиці БД завдяки методам-розширення у методі контексту БД OnModelCreating (Рис. 4.7.1) даного фреймворку та(або) атрибутам написаних над полями моделей (Рис 4.7.2).

```
12 references
public class WarehouseEngineDbContext : DbContext
{
    0 references
    public DbSet<Product> Products { get; set; }
    1 reference
    public DbSet<Supplier> Suppliers { get; set; }
    1 reference
    public DbSet<User> Users { get; set; }

    1 reference
    public WarehouseEngineDbContext(DbContextOptions<WarehouseEngineDbContext> options) : base(options)
    {
    }

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Product>()
            .HasOne(d => d.Supplier)
            .WithMany(s => s.Products)
            .HasForeignKey(d => d.SupplierId);
    }
}
```

Рисунок 4.7.1 – Приклад налаштування зв’язків між таблицями EF Core

```
1 reference
public class User
{
    [Key]
    0 references
    public int Id { get; set; }
    1 reference
    public string Username { get; set; }
    1 reference
    public string Password { get; set; }
}
```

Рисунок 4.7.2 – Приклад налаштування індексів БД завдяки атрибутам EF Core

4.8 Blazor

Blazor – це фреймворк для розробки веб інтерфейсів, який дозволяє писати поєднувати HTML розмітку та C# код як RazorPages, основною особливістю є те, що він використовує технологію WebAssembly, що дозволяє виконувати веб-застосунки у браузері. На відміну від Razor Pages цей фреймворк використовує SignalR за умовчуванням для забезпечення зв’язку клієнту та серверу у реальному часі наприклад годинник у реальному часі (Рис 4.8.1), завдяки якому користувачі застосунку зможуть відстежувати час.

Blazor та RazorPages можна легко поєднувати між собою завдяки тегу <component> (Рис 4.8.3), це дозволить досягти більшої гнучкості застосунку.

```razor
@page "/Clock"


<ul class="navbar-nav ml-auto">
                        <li class="nav-item">
                                <form method="post" asp-action="Logout" asp-controller="User">
                                 <p class="nav-link btn btn-link px-3">Години у реальному часі @CurrentTime</p>
                            </form>
                        </li>
 </ul>

@code {
    int currentCount = 0;
    private string CurrentTime { get; set; }

    protected override void OnInitialized()
    {
        UpdateTime();
    }

    private async void UpdateTime()
    {
        while (true)
        {
            CurrentTime = DateTime.Now.ToString("HH:mm:ss");
            StateHasChanged();
            await Task.Delay(1000);
        }
    }
    void IncrementCount()
    {
        currentCount++;
    }
}
```

Рисунок 4.8.1 – Годинник у реальному часі Blazor реалізація

Завдяки тому, що Blazor використовує SignalR годинник (Рис. 4.8.2) буде оновлюватись без цілого оновлення сторінки.

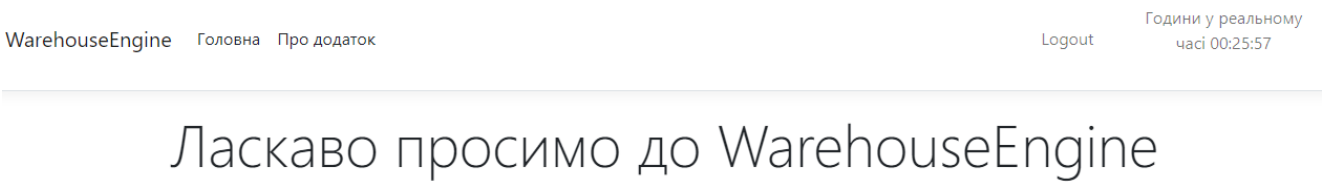


Рисунок 4.8.2 – Годинник у реальному часі

```razor
            }
            <component>@(await Html.RenderComponentAsync<Clock>(RenderMode.Server))</component>
        </nav>
    </header>
    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>
```

Рисунок 4.8.3 – Використання Blazor у RazorPages

Тож завдяки гнучкості та зворотній сумісності фреймворків у .NET можна побачити як легко вони можуть бути комбіновані один з одним без перешкод. Blazor дуже гарна альтернатива іншим UI фреймворкам таким як Angular та нічим йому не уступає, а завдяки підтримці C# програміст може сфокусуватись на одній мові програмування і вести швидку розробку.

ВИСНОВКИ

1. Під час виконання роботи було проведено аналіз предметної галузі. Було виявлено основні потреби користувачів та створено модель предметної галузі.

2. Досліджено та проаналізовано ринок аналогів веб-застосунків для автоматизації складського обліку.

3. Завдяки даним виявленим в ході аналізу предметної галузі та ринку аналогів веб-застосунків для автоматизації складського обліку, було сформовано необхідний для користувачів функціонал та створено модель прецедентів із зображенням різного використання продукту в залежності від прав.

4. Розроблено сервер, БД, інтерфейс користувача для полегшення складського обліку, використовуючи сучасні потужні технології розробки. У процесі створення ПЗ для автоматизації складського обліку було використано : .NET Core, ASP.NET Core MVC, EF Core, Razor Pages, Blazor, HTML, CSS. Використано клієнт серверну архітектуру.

5. Під час створення веб-застосунку було використано рефлексію, асинхронні операції, лямбда-вирази, атрибути, JSON-конфігурацію, класи, методи.

6. У даному веб-застосунку реалізовано систему аутентифікації та авторизації на базі кукі, що не вимагає створення додаткового застосунку який виступає сервером аутентифікації користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Overview of ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/ef/core/.
2. Daniel R. Overview of ASP.NET Core [Електронний ресурс] / R. Daniel, A. Rick, L. Shaun – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0.
3. What is Visual Studio? [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022.
4. A tour of the C# language [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.
5. ASP.NET Razor [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/ASP.NET_Razor.
6. Fast & powerful cross-platform .NET IDE [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/rider/.
7. Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Visual_Studio_Code.
8. Fishbowl Inventory [Електронний ресурс] – Режим доступу до ресурсу: https://www.fishbowlinventory.com/.
9. Zoho Inventory [Електронний ресурс] – Режим доступу до ресурсу: https://www.zoho.com/inventory/.
10. Odoo Inventory [Електронний ресурс] – Режим доступу до ресурсу: https://www.odoo.com /app/inventory.
11. TradeGecko [Електронний ресурс] – Режим доступу до ресурсу: https://commerce.intuit.com.
12. SKULabs [Електронний ресурс] – Режим доступу до ресурсу: https://www.skulabs.com/.
13. ASP.NET Core Blazor [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/blazor/?WT.mc_id=dotnet-35129-

website&view=aspnetcore-7.0

14. Attributes [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/advanced-topics/reflection-and-attributes/.

ДОДАТКИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА WEB-ДОДАТКУ ДЛЯ СКЛАДСЬКОГО ОБЛІКУ НА ПЛАТФОРМІ ASP.NET CORE ТА BLAZOR МОВОЮ C#

Виконав студент 4 курсу
групи ПД-44
Чернявський Ждан Анатолійович
Керівник роботи
Гаманюк Ігор Михайлович, ст. викладач кафедри ІПЗ

Київ – 2023

Мета, об’єкт та предмет роботи

- **Мета роботи** – автоматизація ведення складського обліку шляхом впровадження web-додатку для складського обліку.
- **Об'єкт дослідження** – процес ведення складського обліку.
- **Предмет дослідження** – програмне забезпечення для ведення складського обліку.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз предметної галузі та виявити основні потреби користувачів.
2. Дослідити та проаналізувати ринок аналогів веб-застосунків для автоматизації складського обліку.
3. Сформулювати вимоги до застосунку.
4. Спроектувати базу даних для застосунку.
5. Спроектувати архітектуру веб-застосунку .
6. Розробити серверну частину.
7. Розробити систему аутентифікації на базі cookie.
8. Розробити UI частину.

3

АНАЛІЗ АНАЛОГІВ

Застосунки	Fishbowl Inventory	Zoho Inventory	Odoo Inventory	TradeGecko	SKULabs	**Warehouse Engine**
Управління запасами	+	+	+	+	+	**+**
Управління постачальниками	+	+	+	+	+	**+**
Звітність і аналітика	+	+	+	+	+	**+**
Можливість розгорнути застосунок на своїх серверах	-	-	+	-	-	**+**
Можливість інтеграції з хмарними сервісами	-	-	+	-	-	**+**
Управління працівниками	-	+	+	-	-	**+**

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Можливість експортування даних у excel
2. Можливість додавати, редагувати та видаляти товари
3. Можливість додавати, редагувати та видаляти постачальників
4. Можливість пошуку товарів та постачальників
5. Авторизація

Не функціональні вимоги

1. Багатоплатформність
2. Хмарна сумісність
3. Безпека

5

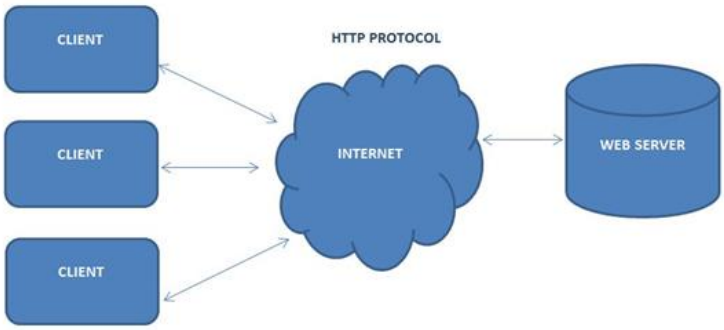
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

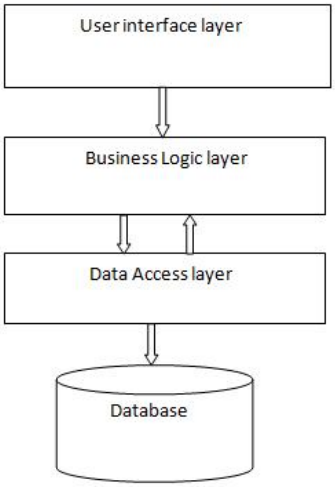
ДІАГРАМИ ВАРІАНТІВ ВИКОРИСТАННЯ

7

АРХІТЕКТУРА ЗАСТОСУНКУ



Зовнішня архітектура застосунку



Внутрішня архітектура застосунку

8

ДІАГРАМИ ДІЯЛЬНОСТІ

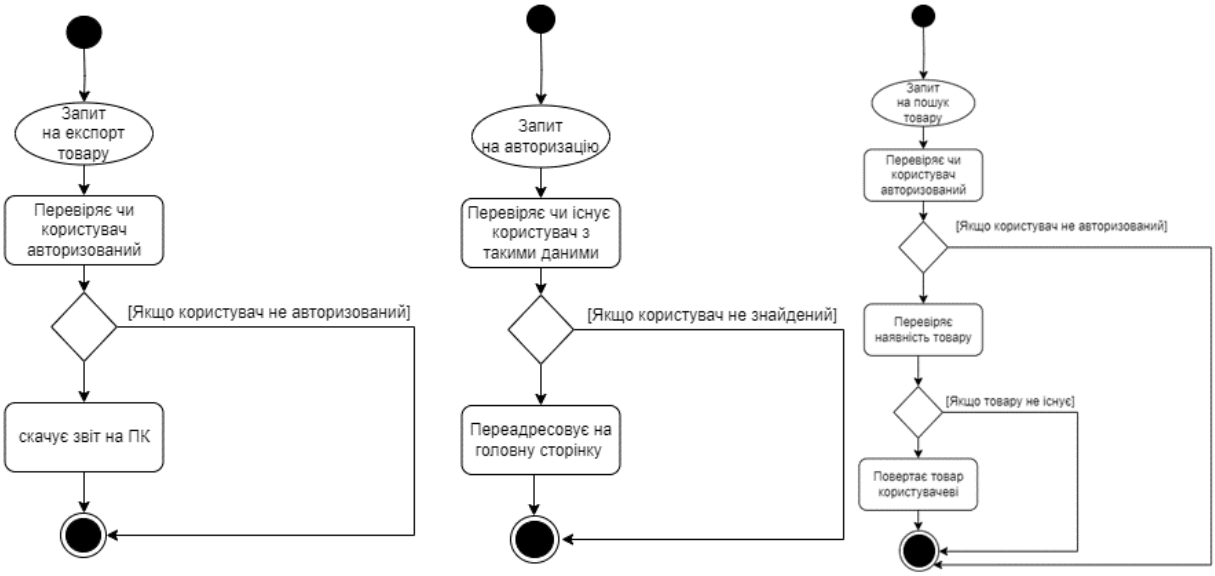
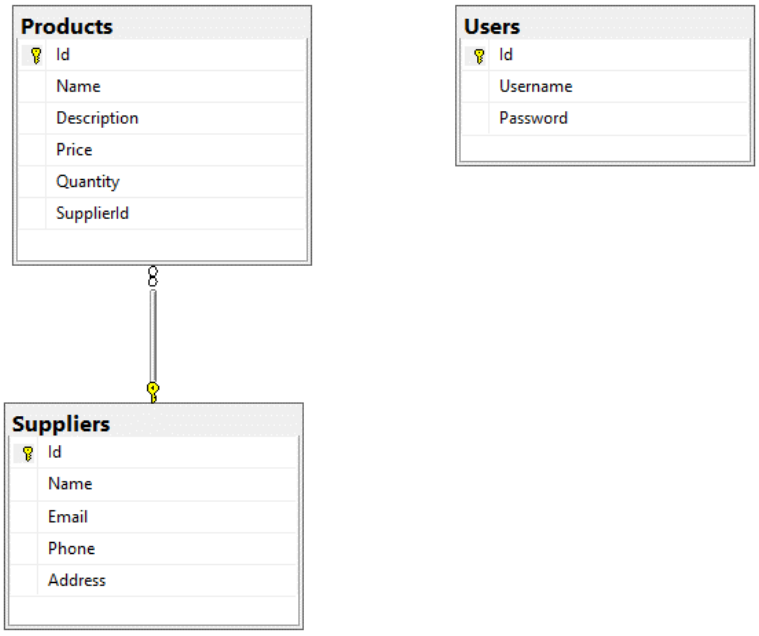
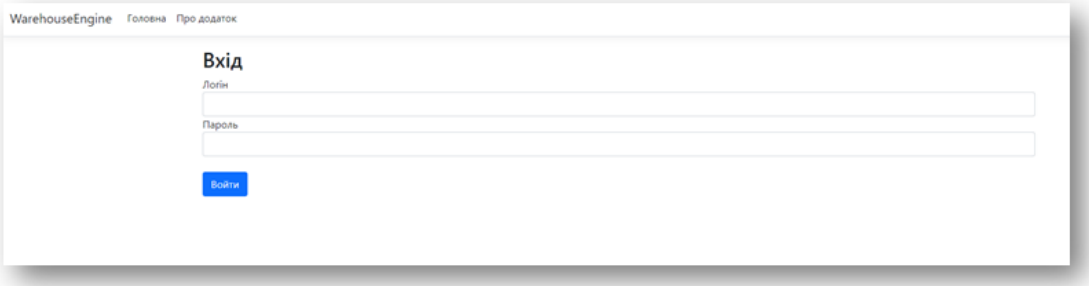


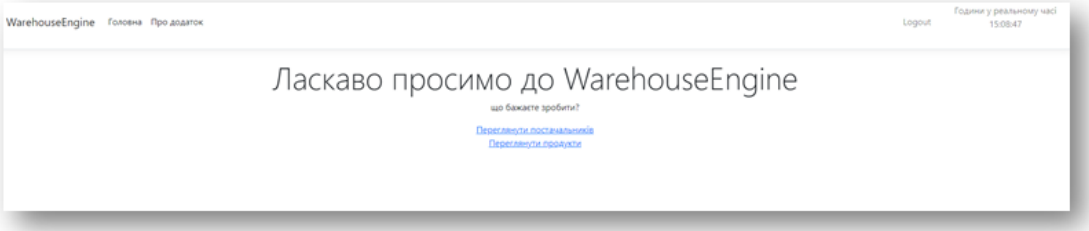
СХЕМА БД



ЕКРАННІ ФОРМИ



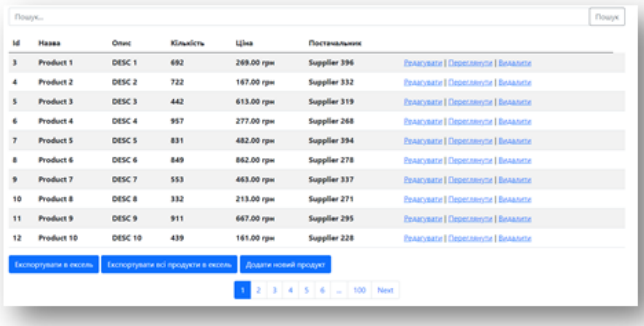
Сторінка авторизації



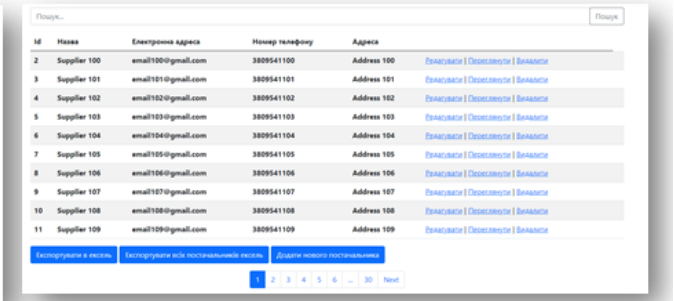
Головна сторінка застосунку

11

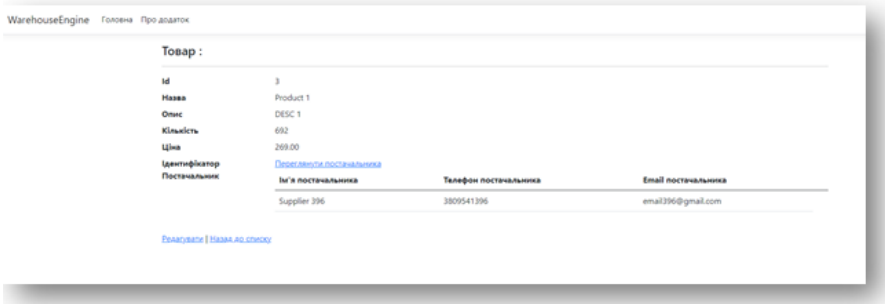
ЕКРАННІ ФОРМИ



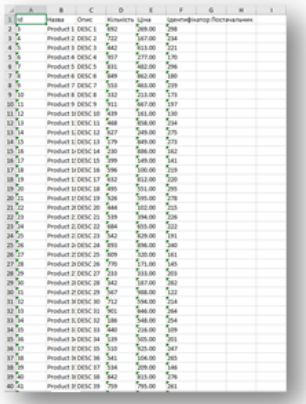
Перегляд товарів



Перегляд постачальників



Детальний перегляд товару



Експорт товарів у excel

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Чернявський Ж. А. Аналіз переваг використання ui-фреймворку razor pages у розробці web-додатку для складського обліку на платформі asp.net core / Чернявський Ж.А., Гаманюк І.М. // Застосування програмного забезпечення в інфокомунікаційних технологіях: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 20.04.2023, ДУТ, м. Київ — К.: ДУТ, 2023. — С. 148.

2. Чернявський Ж. А. Розробка web-додатку для складського обліку мовою c# на платформі asp.net core та blazor core / Чернявський Ж.А., Гаманюк І.М. // Застосування програмного забезпечення в інфокомунікаційних технологіях: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 20.04.2023, ДУТ, м. Київ — К.: ДУТ, 2023. — С. 146.

ВИСНОВКИ

1. Проведено аналіз предметної галузі. Було виявлено основні потреби користувачів та створено модель предметної галузі.
2. Досліджено та проаналізовано ринок аналогів веб-застосунків для автоматизації складського обліку.
3. Сформовано необхідний для користувачів функціонал та створено модель прецендентів із зображенням різного використання продукту в залежності від прав.
4. Розроблено сервер, БД, інтерфейс користувача для полегшення складського обліку. У процесі створення ПЗ для автоматизації складського обліку було використано : .NET Core, ASP.NET Core MVC, EF Core, Razor Pages, Blazor, HTML, CSS. Використано клієнт серверну архітектуру.
5. Реалізовано систему аутентифікації та авторизації на базі cookie, що не вимагає створення додаткового застосунку який виступає сервером аутентифікації користувачів.

ДЯКУЮ ЗА УВАГУ!