НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
Інформаційних систем і технологій

_____ Швиденко М.З

“___” _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Інформаційна система автоматизованого робочого місця працівника складської інфраструктури»

Спеціальність 122 – «Комп’ютерні науки»

Гарант освітньої програми
Д.е.н., професор _____ Руденський Р.А.

Керівник бакалаврської кваліфікаційної роботи
К.е.н., доцент _____ Рогоза К.Г.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Паровенко Іван Олександрович
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інформаційних систем і технологій

_____ Швиденко М. З.

"___" _____ 2025 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Паровенко Іван Олександрович

Спеціальність 122 – «Комп'ютерні науки»

1. Тема бакалаврської кваліфікаційної роботи «Інформаційна система автоматизованого робочого місця працівника складської інфраструктури» затверджена наказом ректора НУБіП України від 25.04.2025 № 699 «С»
2. Термін подання завершеної роботи на кафедру 02.06.2025
(рік, місяць, число)
3. Вихідні дані до роботи: надання візуальної інформації про кількість, розміщення та перелік товарів підприємства у складських приміщеннях.
4. Перелік питань, що розглядаються:
   - Аналіз проблемної області
   - Моделювання предметної області
   - Проектування програмної системи
   - Впровадження та експлуатація системи

Дата видачі завдання "25" квітня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ / Рогоза К.Г./
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання: _____ / Паровенко І.О./
(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД	–	База даних
ОС	–	Операційна система
ПЗ	–	Програмне забезпечення
СКБД	–	Система керування баз даних
API	–	Application Programming Interface
Bash	–	Bourne Again SHell
MVVM	–	Model-View-ViewModel
SQL	–	Structed Query Language
UI	–	User Interface

ВСТУП

В умовах стрімкого технологічного прогресу та зростаючої конкуренції в секторах логістики та ланцюгів постачання компанії все більше зосереджуються на оптимізації своїх внутрішніх процесів. Склади відіграють вирішальну роль в організації зберігання, переміщення та управління товарними запасами. Проте ефективність роботи складу багато в чому залежить від якості та швидкості обробки інформації, чому часто перешкоджають застарілі ручні методи або відсутність спеціалізованих програмних засобів.

Важливим аспектом оптимізації складу є автоматизація завдань, які виконують працівники складської інфраструктури, в обов’язки яких входить прийом, зберігання, відправка та облік товарів. На практиці ці завдання передбачають обробку великих обсягів даних, що вимагає точності, оперативності та своєчасного доступу до актуальної інформації.

Дана бакалаврська робота присвячена розробці інформаційної системи автоматизованого робочого місця працівника складської інфраструктури. Запропонована система реалізована як настільна програма з використанням C# та Windows Forms, дані зберігаються та керуються через базу даних Microsoft SQL Server. Система спрямована на підтримку основної діяльності складського персоналу, такої як реєстрація продуктів, моніторинг рівня запасів, рух на складі (вхідний і вихідний) і звітність.

Актуальність обраної теми визначається у зростаючому попиті на цифрову трансформацію в логістиці та відсутністю доступних, простих у використанні та настроюваних рішень для управління складським господарством для малих і середніх підприємств. Розроблена програма пропонує практичний підхід до автоматизації складів і може слугувати прототипом для подальшого вдосконалення або інтеграції у великі корпоративні системи.

Мета роботи - покращити існуючу сферу управління складськими приміщеннями, розробивши настільну інформаційну систему, яка автоматизує діяльність працівника складської інфраструктури та підвищує ефективність роботи складу.

Цілі дослідження:

- аналіз існуючих рішень і визначення функціональних вимог;
- розробка структури бази даних для управління складськими даними;
- розробка інтерфейсу програми та бізнес-логіки;
- впровадження функції для контролю запасів і відстеження руху;
- забезпечення безпеки даних і контролю доступу користувачів;
- перевірка й оцінка продуктивності і зручності використання системи.

Методи та технології розробки. Для реалізації програмного додатку було використано сучасні інструменти та технології програмування. Основне середовище розробки – Microsoft Visual Studio 2022, мова програмування – C# з використанням технології Windows Forms для створення настільного додатку. Для проектування та управління базою даних застосовано Microsoft SQL Server 2019 та SQL Management Studio. Взаємодія програми з базою даних реалізована за допомогою технології Entity Framework, що забезпечує об'єктно-реляційне відображення та спрощує маніпуляцію даними. Також у розробці використано паттерн проектування Model-View-Controller (MVC), що дозволяє відокремити бізнес-логіку від представлення даних та підвищити модульність і підтримуваність коду.

Структура пояснювальної записки. Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 78 сторінок, включаючи 1 таблицю, 17 рисунків та 2 додатки. Список використаних джерел містить 27 найменувань.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис предметної області

Ефективне управління складом стало ключовим фактором забезпечення стабільної роботи в різних секторах бізнесу, зокрема в логістиці та ланцюгах поставок. Склади більше не розглядаються як статичні сховища — це динамічні робочі центри, де рух і відстеження товарів мають бути швидкими, точними та добре скоординованими. Незважаючи на це, багато компаній, особливо малі та середні підприємства (МСП), продовжують покладатися на застарілі ручні методи управління запасами, такі як паперові журнали або електронні таблиці Excel. Ці підходи не тільки неефективні, але й збільшують ймовірність людських помилок, затримок в обробці даних і труднощів у веденні точних записів.

Складський персонал, зокрема працівники інфраструктури, відповідають за широкий спектр щоденних завдань — від прийому поставок і впорядкування товарів на полицях до відправки товарів і підтримки точності інвентаризації. Ці процеси вимагають уваги до деталей, послідовного введення даних і доступу до інформації в реальному часі. Без відповідних програмних інструментів працівники змушені жонглювати кількома обов’язками вручну, що часто призводить до збоїв у роботі, недоречних складських запасів або неточного рівня запасів.

Хоча сьогодні існує багато комерційних систем керування складськими приміщеннями (WMS), таких як Oracle WMS Cloud, SAP EWM і Fishbowl Inventory, більшість із них розроблено для великих підприємств. Їхня складність, висока вартість ліцензування та технічні вимоги роблять їх непрактичними для невеликих організацій. Крім того, багато існуючих рішень базуються на хмарі або вимагають доступу до Інтернету, що може не відповідати інфраструктурі чи перевагам усіх компаній, особливо тих, хто

шукає автономні системи на базі робочого столу, які можуть працювати в автономному режимі та в локальних мережах.

У цьому контексті існує очевидна потреба в легкій, зручній інформаційній системі, спеціально адаптованій до потреб працівників складської інфраструктури. Відсутність таких доступних інструментів часто призводить до фрагментованого управління даними та операційної неефективності. Добре розроблена програма може слугувати централізованою платформою, де працівники можуть виконувати всі основні функції — реєструвати продукти, контролювати рівень запасів, реєструвати рух на складі та створювати звіти — швидко та організовано [1].

Розробивши спеціалізовану настільну систему, яка автоматизує основні завдання персоналу складської інфраструктури, можна значно підвищити якість обслуговування складу. Впровадження такої системи зменшує залежність від ручних процесів, знижує ризик помилок і гарантує, що дані інвентаризації постійно актуальні та безпечні. Крім того, це сприяє більш обґрунтованому прийняттю рішень, оскільки звіти та аналітика стають легко доступними.

Зрештою, інвестиції в автоматизацію складських операцій на рівні інфраструктури є стратегічним кроком до модернізації бізнес-процесів. Це забезпечує основу для підвищення продуктивності, кращого контролю над запасами та плавнішого робочого процесу — усе це має вирішальне значення в конкурентному середовищі сучасної економіки.

Предмет цієї роботи охоплює дії, робочі процеси та процеси даних, пов’язані зі складською інфраструктурою — життєво важливим компонентом логістичного ланцюга будь-якого підприємства. Склад функціонує як операційний центр, де товари приймаються, зберігаються, управляються та відправляються. Точність і ефективність цих операцій безпосередньо впливають на графік виробництва, задоволеність клієнтів і фінансові результати. Тому акцент на оптимізації робочого середовища та інструментів,

доступних працівникам складської інфраструктури, є своєчасним і актуальним.

Працівник інфраструктури, як основна фігура в складських операціях, виконує численні обов'язки, які забезпечують безперебійну циркуляцію товарів у межах об'єкта та за його межами. Це включає отримання відправлень, перевірку та реєстрацію кількості та стану товарів, організацію інвентаризації у визначених зонах зберігання, оновлення даних про запаси та обробку вихідних поставок. Крім того, працівник може бути залучений до проведення перевірок інвентаризації, маркування товарів і спілкування з іншими відділами для координації логістики [2].

Традиційно ці завдання виконувалися за допомогою ручних журналів, друкованих документів і базових офісних інструментів, що не тільки сповільнює роботу, але й збільшує ризик невідповідності даних. Із зростанням обсягів продукції та очікувань клієнтів стає все важче управляти складськими процесами без допомоги спеціалізованого програмного забезпечення. Це особливо вірно, коли мова йде про підтримку точного рівня запасів, запобігання втраті або неправильному розміщенню предметів і створення своєчасних звітів [3].

Інформаційні системи відіграють ключову роль в автоматизації та оптимізації складських робочих процесів. Добре розроблена система може забезпечити відстеження товарів у режимі реального часу, автоматизувати створення документів і підтримувати рольовий доступ до даних, забезпечуючи взаємодію кожного користувача з відповідними функціями. У випадку працівників інфраструктури спеціальна настільна програма може служити цифровим помічником, спрощуючи повторювані завдання та організовуючи потік інформації в структурованому, доступному форматі.

Впроваджуючи інформаційну систему, адаптовану до конкретних потреб працівника складської інфраструктури, підприємства можуть досягти більшої прозорості операцій, скоротити час, витрачений на рутинні процеси, і забезпечити більш надійне та безпечне управління даними. Зосередження на

настільному середовищі також забезпечує роботу в автономному режимі та локальне розгортання, чому часто віддають перевагу в компаніях, де веб-рішення непотрібні або менш безпечні.

Детальний опис класів та атрибутів предметної області наведено у таблиці 1.1.

Таблиця 1.1

Опис атрибутів класів предметної області

Клас предметної області	**Атрибут**	**Опис**
Товар	Назва товару	Повна назва продукції або матеріалу
	Код товару	Унікальний ідентифікатор товару (штрих-код або інший код)
	Одиниця виміру	Одиниця вимірювання товару (шт., кг, л тощо)
	Кількість на складі	Поточна кількість товару, доступного на складі
Постачальник	Назва компанії	Назва фірми або організації, що постачає товари
	Контактна особа	Ім’я відповідальної особи або представника постачальника
	Телефон	Контактний номер телефону для зв’язку з постачальником
	Електронна пошта	Адреса електронної пошти для листування з постачальником
Операція з товаром	Тип операції	Тип операції з товаром (наприклад: надходження, видача, переміщення)
	Дата операції	Час і дата проведення операції з товаром
	Кількість	Кількість одиниць товару, що беруть участь у операції
	Відповідальна особа	Працівник, що виконував операцію з товаром
	Коментар	Додаткова інформація або пояснення щодо операції з товаром

1.2 Огляд існуючих рішень

Розглянемо існуючі рішення предметної області.

В даному розділі розглянемо основні існуючі програмні рішення, які використовуються для автоматизації складської інфраструктури та управління складськими операціями. Такі системи значно спрощують управління товарообігом, контролем запасів і ресурсів на складах, забезпечуючи ефективність і точність.

Oracle Warehouse Management (WMS) — комплексне рішення, призначене для автоматизації та оптимізації різних складських операцій. Ця система надає підприємствам можливість ефективно керувати запасами, виконанням замовлень, доставкою та отриманням, значно покращуючи загальний процес управління ланцюгом поставок [4].

Однією з ключових сильних сторін Oracle WMS є її можливості керування запасами. Завдяки використанню передових технологій, таких як сканування штрих-кодів і RFID, система дозволяє відстежувати товари на складі в реальному часі. Це гарантує, що дані про запаси завжди точні, допомагаючи компаніям підтримувати оптимальний рівень запасів. Система підтримує різні методи управління запасами, такі як FIFO (першим прийшов, першим вийшов), LIFO (останній прийшов, першим вийшов) і відстеження на основі місцезнаходження, гарантуючи, що товари обробляються найбільш ефективним способом.

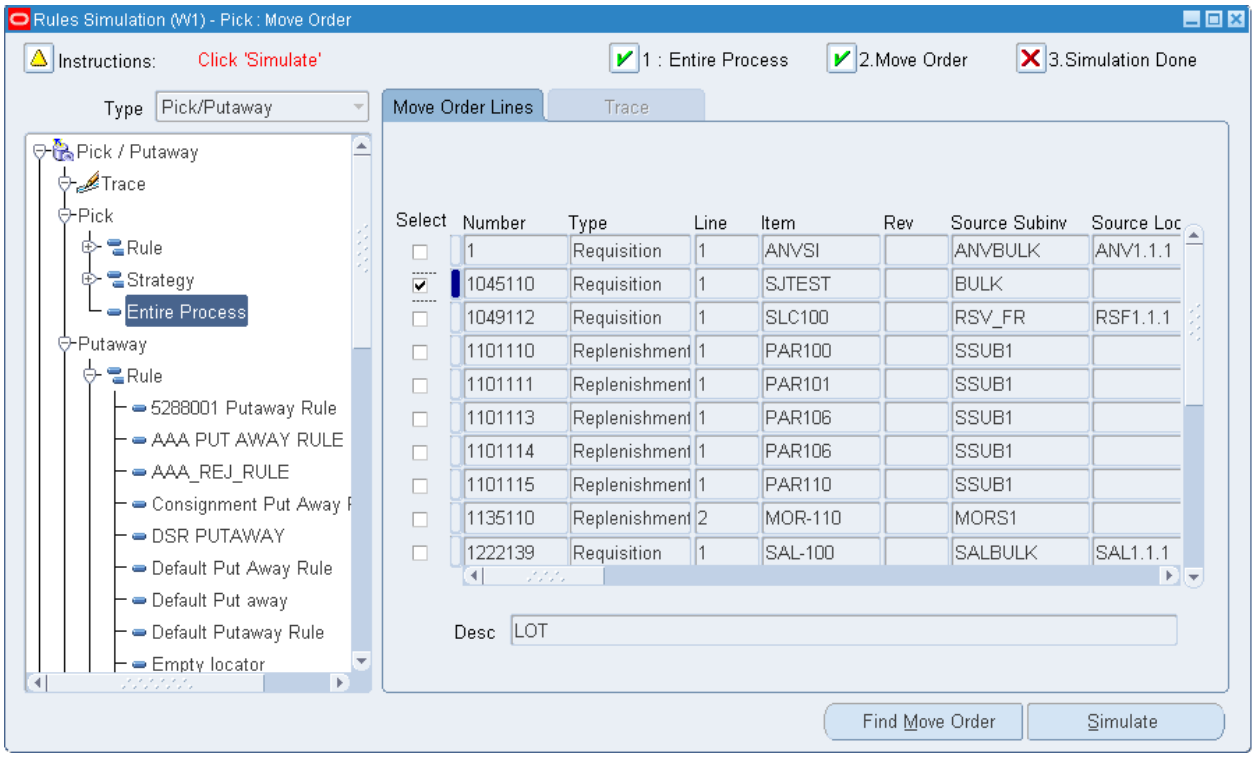


Рис. 1 Інформаційна система «Oracle Warehouse Management»

Виконання замовлень є ще однією сферою, де Oracle WMS домінує. Система автоматизує багато процесів, пов'язаних із набором замовлень, пакуванням і відправленням, що допомагає прискорити процес виконання. Програмне забезпечення бездоганно інтегрується з системами керування замовленнями, забезпечуючи плавну координацію між керуванням запасами та обробкою замовлень. Доступні різноманітні методи комплектування, включаючи хвилевий комплект, зональний комплект і пакетний комплект, що дозволяє компаніям вибрати найкращий варіант для своїх конкретних потреб [5].

Процеси доставки також оптимізовані за допомогою Oracle WMS, яка підтримує автоматизовану доставку, включаючи створення транспортних етикеток, пакувальних листів та іншої необхідної документації. Крім того, він може інтегруватися з різними перевізниками, щоб вибрати найбільш економічно ефективні та своєчасні способи доставки. Ця інтеграція допомагає зменшити витрати на доставку та підвищити точність доставки.

Система забезпечує видимість даних у режимі реального часу, дозволяючи менеджерам складів контролювати ключові показники

ефективності, такі як точність замовлень, рівень запасів і час доставки. Завдяки докладним звітам і аналітиці менеджери можуть приймати рішення на основі даних, щоб оптимізувати продуктивність складу та завчасно вирішувати будь-які проблеми.

Oracle WMS також пропонує надійні функції управління працею. Це допомагає відстежувати продуктивність співробітників, керувати графіками змін і ефективно розподіляти ресурси. Це гарантує безперебійну роботу складу та максимально ефективне використання робочої сили.

На додаток до цих функцій Oracle WMS може інтегруватися з автоматизованими системами та робототехнікою. Це включає в себе такі технології, як автоматизовані керовані транспортні засоби (AGV) і роботизовані системи комплектування, які можуть підвищити ефективність роботи за рахунок зменшення потреби в ручній праці та мінімізації помилок. Інтеграція таких технологій ще більше оптимізує потік товарів на складі, прискорюючи процеси та зменшуючи людські помилки.

Оптимізація складського простору є ще однією перевагою Oracle WMS. Система дозволяє підприємствам максимально використовувати доступний простір, направляючи товари до найбільш підходящих місць зберігання на основі таких факторів, як попит і термін придатності. Використання динамічного слотування, передової техніки для оптимізації зберігання, гарантує, що товари завжди зберігаються найбільш ефективним способом [6].

Підсумовуючи, Oracle Warehouse Management надає підприємствам потужний інструмент для вдосконалення їх складських операцій. Пропонуючи комплексні функції, видимість у реальному часі та повну інтеграцію з іншими системами Oracle, це допомагає компаніям підвищити ефективність, зменшити витрати та ефективніше задовольняти вимоги клієнтів. Oracle WMS можна налаштувати відповідно до унікальних потреб будь-якої організації, як для малого, так і для великого бізнесу, що робить його критично важливим компонентом сучасного управління ланцюгом поставок.

Розглянемо інше програмне забезпечення на ринку.

TradeGecko, тепер відомий як QuickBooks Commerce після його придбання компанією Intuit, — це комплексна хмарна платформа, призначена для спрощення управління запасами та замовленнями для компаній. Основна мета — допомогти малим і середнім підприємствам (МСП) оптимізувати свою діяльність, підвищити ефективність і масштабувати процеси свого ланцюжка поставок [7].

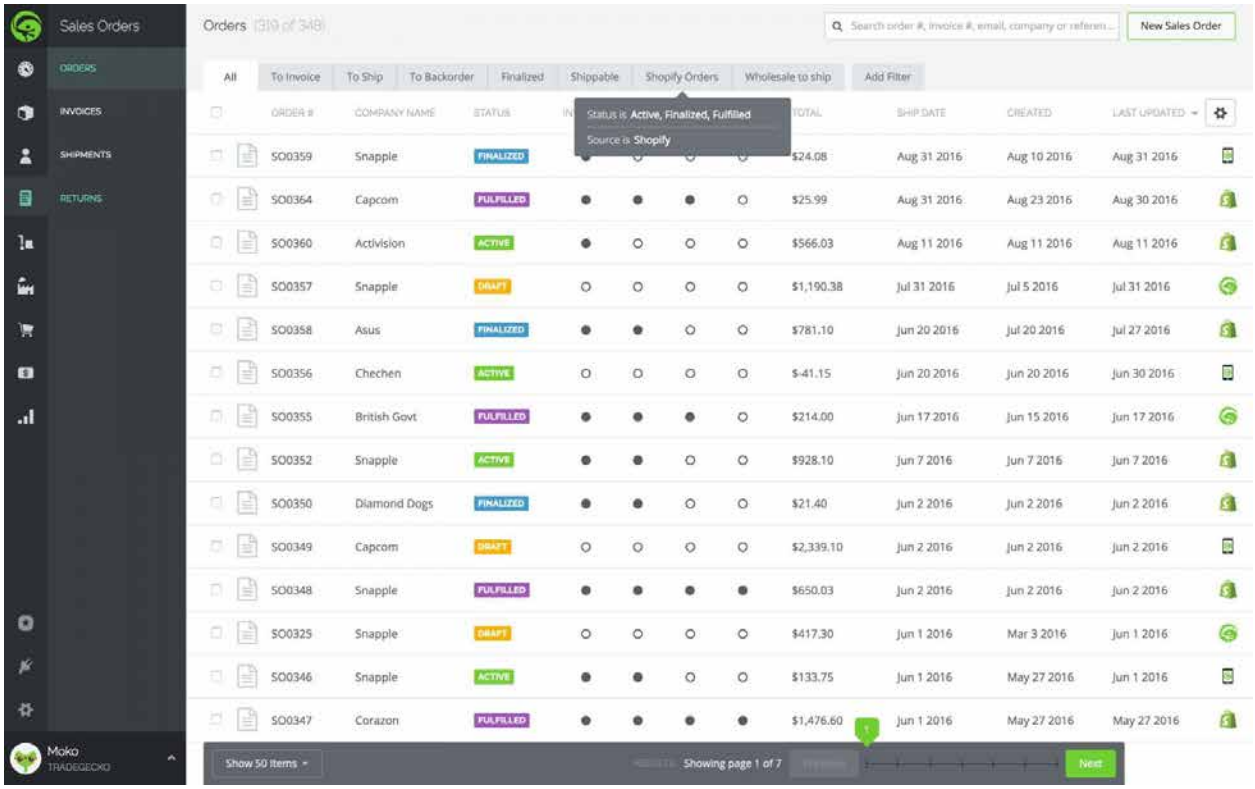


Рис. 2 Інформаційна система «TradeGecko»

Однією з видатних особливостей TradeGecko є його здатність керувати запасами в режимі реального часу. Система дає змогу компаніям точно відстежувати рівень запасів, запобігаючи таким проблемам, як відсутність або надлишок запасів. Завдяки інструментам для автоматичного оновлення запасів, сповіщень про низькі запаси та керування декількома складами, TradeGecko робить управління запасами ефективнішим і прозорішим. Підприємства також можуть класифікувати продукти, керувати різними варіантами продуктів і автоматизувати повторне замовлення, щоб завжди мати під рукою потрібну кількість товару.

Іншим ключовим аспектом TradeGecko є його функція керування замовленнями. Платформа автоматизує весь процес замовлення, від створення замовлення та виставлення рахунків до відстеження відправлення. Він легко інтегрується з різними каналами продажів, включаючи платформи електронної комерції та торгові майданчики, що дозволяє компаніям керувати замовленнями з різних джерел в одному центральному місці. Ця інтеграція не тільки зменшує ручну роботу, але й покращує точність, усуваючи необхідність повторного введення даних.

TradeGecko підтримує підприємства в управлінні продажами через кілька каналів, забезпечуючи узгодженість у списках продуктів і рівнях запасів. Незалежно від того, чи продають компанії через онлайн-магазини, ринки, такі як Amazon і eBay, або фізичні роздрібні місця, система синхронізує дані про продукти на всіх цих платформах. Це запобігає перепродажу та допомагає підприємствам підтримувати актуальні записи про запаси по всіх каналах продажу.

Платформа також включає вбудовану систему управління взаємовідносинами з клієнтами (CRM), яка допомагає компаніям відстежувати взаємодію з клієнтами та вести детальний облік замовлень і вподобань. Ця функція покращує обслуговування клієнтів, пропонуючи компаніям зрозуміти купівельну поведінку своїх клієнтів і дозволяючи їм персоналізувати свої комунікаційні та маркетингові зусилля [8].

TradeGecko об’єднує такі функції бухгалтерського обліку, як виставлення рахунків, відстеження платежів і фінансова звітність. Система генерує докладні звіти, які дають уявлення про показники продажів, оборотність запасів і загальну прибутковість. Підприємства також можуть підключити TradeGecko до бухгалтерського програмного забезпечення, такого як QuickBooks і Xero, що забезпечує безперебійне фінансове управління та усуває потребу в ручній звірці.

Крім того, TradeGecko пропонує інструменти для керування ланцюгом поставок, наприклад керування постачальниками та відстеження замовлень на

купівлю. Підприємства можуть оптимізувати свої процеси закупівель і навіть автоматизувати повторне замовлення на основі попередньо визначених порогових значень запасів. Це скорочує час, витрачений на оформлення замовлень вручну, і гарантує, що компанії ніколи не вичерпають основні продукти.

Для керування бізнесом у дорозі TradeGecko пропонує мобільний додаток, доступний як на пристроях iOS, так і на Android. Ця програма дозволяє користувачам отримувати доступ до інвентаризаційних даних у режимі реального часу, переглядати та керувати замовленнями, а також відстежувати відправлення з будь-якого місця, що робить її особливо корисною для власників бізнесу та менеджерів складів, яким потрібно залишатися на зв’язку, коли вони не в офісі.

Загалом TradeGecko є потужним інструментом для компаній, які прагнуть оптимізувати свою діяльність і підвищити рівень задоволеності клієнтів. Його повні функції, доступ до даних у режимі реального часу та можливості автоматизації роблять його цінним рішенням для підприємств електронної комерції, оптової та роздрібної торгівлі. Централізувавши інвентаризацію, замовлення та керування клієнтами, TradeGecko дозволяє компаніям працювати ефективніше та ефективно масштабуватися.

1.3 Постановка завдання

Інформаційна система автоматизованого робочого місця працівника складської інфраструктури призначена для оптимізації управління та моніторингу роботи складу. Начальник складського відділу відповідатиме за ведення точного обліку різних операційних показників, таких як рівень запасів, стан виконання замовлення, рух запасів і продуктивність праці. Щоб полегшити це, система повинна бути обладнана базою даних, яка зберігає всі відповідні дані, надаючи користувачеві доступ до персоналізованих повідомлень і звітів.

Основна мета цієї системи – забезпечити швидкий і легкий доступ до інформації щодо:

- рівень складських запасів і рух запасів;
- прогрес щодо вхідних і вихідних замовлень;
- продуктивність працівників і стан виконання завдань;
- інвентаризаційні надлишки або нестачі, які вимагають уваги або дій.

Програмна система буде структурована навколо інтуїтивно зрозумілого діалогового меню, яке запропонує користувачу різноманітний функціонал. За допомогою цього інтерфейсу керівники складів і працівники зможуть вводити дані, відстежувати показники ефективності та отримувати сповіщення про потенційні проблеми, такі як низькі запаси, затримки поставок або неефективність завдань.

Система також дозволить користувачам скасовувати або змінювати раніше зареєстровані операції, забезпечуючи гнучкість у разі помилок або змін в операційному процесі. Таке налаштування гарантує, що команда управління складом може швидко реагувати на динамічні виклики та підтримувати оптимальну продуктивність складу. Система відіграватиме ключову роль у підвищенні операційної ефективності, зменшенні помилок, що виникають вручну, і підтримці процесу прийняття рішень у складському середовищі.

1.4 Функціональні та нефункціональні вимоги

Функціональні вимоги:

1. система повинна дозволяти створення, зміну та видалення облікових записів користувачів, у тому числі працівників складу, менеджерів та адміністраторів;
2. користувачам повинні бути призначені певні ролі (наприклад, працівник складу, керівник, адміністратор) із визначеними дозволами для доступу до різних розділів системи;

3. система повинна забезпечувати відстеження рівня запасів у режимі реального часу для всіх складських продуктів;
4. користувачі повинні мати можливість додавати, видаляти або оновлювати інформацію про продукт, включаючи назву продукту, категорію, кількість на складі та розташування на складі;
5. система повинна забезпечувати моніторинг руху запасів, таких як вхідні та вихідні продукти, і надавати сповіщення про низькі рівні запасів;
6. система повинна підтримувати створення та відстеження замовлень від клієнтів;
7. вона має надавати можливість оновлювати статуси замовлення (наприклад, очікує на розгляд, завершено, відправлено);
8. користувачі повинні мати можливість відстежувати вхідні та вихідні замовлення та керувати ними, призначати продукти замовленням і оновлювати статус кожного замовлення;
9. система повинна дозволяти працівникам складу реєструвати та керувати такими завданнями, як комплектування, пакування та відправлення.

Нефункціональні вимоги:

1. система повинна мати можливість обробляти великий обсяг транзакцій і одночасних користувачів без значного зниження продуктивності;
2. час відповіді на такі дії, як додавання або оновлення запасів, обробка замовлень і створення звітів, має бути мінімальним (наприклад, менше 3 секунд для більшості операцій);
3. система повинна бути масштабованою, щоб відповідати майбутньому зростанню запасів, замовлень і користувачів;

4. вона має підтримувати розміщення додаткових складів і збільшення кількості користувачів без серйозних змін базової архітектури;
5. система повинна мати високий рівень доступності з мінімальними простоями для обслуговування або оновлень;
6. необхідно запровадити процеси резервного копіювання та відновлення, щоб забезпечити захист даних і можливість їх відновлення у разі збою;
7. система повинна забезпечувати безпеку даних за допомогою автентифікації користувача, рольового контролю доступу та шифрування даних;
8. такі конфіденційні дані, як відомості про користувачів, фінансові дані та інвентаризаційні записи, необхідно надійно зберігати та захищати від несанкціонованого доступу.

Ці функціональні та нефункціональні вимоги забезпечують повний огляд очікувань від системи управління складом, гарантуючи, що вона відповідає як операційним, так і технічним потребам для підтримки ефективного функціонування складської інфраструктури.

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Загальні відомості

Уніфікована мова моделювання (UML) — це стандартизований підхід до моделювання програмних систем, що пропонує засоби для візуалізації, специфікації та документування їх конструкції та поведінки. Це допомагає розробникам програмного забезпечення, розробникам і зацікавленим сторонам зрозуміти структуру, зв’язки та взаємодію в системі. UML містить різноманітні діаграми, які фокусуються на різних аспектах системи, розділяючи їх на дві основні категорії: структурні діаграми та діаграми поведінки. Ці діаграми допомагають протягом життєвого циклу розробки програмного забезпечення, від початкової концептуалізації до реалізації [9].

Структурні діаграми використовуються для ілюстрації статичних компонентів системи. Вони зображують організацію системи, її компоненти та зв’язки між ними. Однією з найважливіших структурних діаграм є діаграма класів, яка представляє класи в системі разом із їхніми атрибутами, методами та зв’язками. Ця діаграма особливо корисна для об'єктно-орієнтованого проєктування. Діаграма компонентів, з іншого боку, ілюструє компоненти або модулі високого рівня системи та те, як вони взаємодіють один з одним. Він зосереджений на архітектурних аспектах системи. Для більш детального перегляду на основі екземплярів діаграма об’єктів використовується для показу конкретних екземплярів об’єктів та їхніх зв’язків у певний момент часу. Крім того, діаграми розгортання мають вирішальне значення для відображення фізичних аспектів системи, таких як апаратне забезпечення та відображення компонентів програмного забезпечення на фізичних пристроях. Діаграми пакетів також використовуються для організації класів у більші, більш керовані групи, що допомагає зрозуміти загальну структуру системи.

Діаграми поведінки, навпаки, зосереджуються на динамічних аспектах системи, наприклад, як вона реагує на різні події та як процеси

розгортаються з часом. Діаграми варіантів використання відіграють важливу роль у моделюванні функціональності системи з точки зору користувачів, показуючи взаємодію між користувачами (акторами) і системою через варіанти використання. Вони забезпечують загальний огляд функціональності системи. Діаграми послідовності, ще одна діаграма критичної поведінки, описують, як об’єкти або компоненти спілкуються та взаємодіють один з одним протягом певного часу, висвітлюючи послідовність подій або повідомлень, якими обмінюються. Діаграми діяльності використовуються для моделювання робочих процесів і процесів, детально описуючи послідовність дій і моменти прийняття рішень у процесі. Вони безцінні для моделювання бізнес-процесів, взаємодії користувачів або алгоритмів. Діаграма станів або діаграма кінцевого автомата орієнтована на представлення різних станів, у яких може перебувати об’єкт, і переходів між цими станами на основі конкретних подій. Ця діаграма допомагає візуалізувати життєвий цикл об’єктів або сутностей у системі. Діаграми зв’язку, подібні до діаграм послідовності, показують, як об’єкти взаємодіють один з одним, але зосереджуються на зв’язках і механізмах передачі повідомлень між ними. Оглядові діаграми взаємодії поєднують елементи як з діаграм діяльності, так і з діаграм послідовності, забезпечуючи більш високий рівень уявлення про взаємодії в системі.

UML пропонує кілька ключових переваг, що робить його цінним інструментом у процесі розробки програмного забезпечення. Він забезпечує стандартизований підхід до моделювання, забезпечуючи послідовність і ясність для команди розробників. Візуальне представлення системи полегшує розробникам і зацікавленим сторонам розуміння дизайну, структури та поведінки системи. Гнучкість UML дозволяє використовувати його як для невеликих, так і для великих проєктів, задовольняючи потреби широкого кола програм. UML також служить життєво важливим інструментом для документування, забезпечуючи чіткий запис про дизайн системи для подальшого використання, обслуговування та оновлень. Це покращує

спілкування всередині команди розробників і з зацікавленими сторонами, оскільки пропонує спільну мову, зрозумілу кожному [10].

Підсумовуючи, UML є незамінним інструментом для розробки сучасного програмного забезпечення, що дозволяє командам моделювати та візуалізувати різні аспекти системи. Його діаграми спрощують складні системи, полегшуючи їх розуміння та спілкування. Ефективно використовуючи UML, розробники можуть гарантувати, що вони мають чітке, узгоджене розуміння структури та поведінки системи, що веде до кращого дизайну, простішого обслуговування та плавного виконання проєкту.

2.2 Об’єктне та функціональне моделювання

2.2.1 Діаграма прецедентів. Діаграма варіантів використання — це візуальний інструмент, який використовується для представлення взаємодії між системою та її користувачами, демонструючи функціональні можливості або поведінку, для виконання яких призначена система. Він є частиною Уніфікованої мови моделювання (UML) і відіграє вирішальну роль у відображенні функціональних вимог системи простим і зрозумілим способом. Цей тип діаграми є особливо цінним на початкових етапах розробки програмного забезпечення, оскільки він підкреслює основні функції, які користувачі очікують від системи [11].

Основна мета діаграми варіантів використання — визначити область дії системи та надати чіткий огляд того, як користувачі (або інші системи) взаємодіятимуть з нею. Відображаючи ці взаємодії, діаграма допомагає прояснити можливості системи та конкретні завдання, які вона виконуватиме у відповідь на дії користувача.

Діаграма зазвичай складається з трьох ключових елементів. Перший — це актор, який представляє зовнішню сутність, яка взаємодіє з системою. Це може бути людина, інша система або пристрій, який спілкується з системою для досягнення певних цілей. Другим елементом є сценарій

використання, який відноситься до певної дії або послуги, яку система надасть, коли актор взаємодіє з нею. Кожен варіант використання описує окрему функцію або поведінку, яку виконує система. Третій компонент — це зв’язок між акторами та варіантами використання, що вказує на те, як вони взаємодіють. Найпоширеніші зв’язки включають асоціації, які демонструють прямий зв’язок між актором і варіантом використання, а також спеціальні зв’язки, як-от «включати» та «розширювати», які описують, як певні варіанти використання можуть або включати інші варіанти використання, або розширювати їх функціональні можливості за певних умов.

Діаграми варіантів використання в основному використовуються на ранніх етапах проєкту, щоб окреслити функціональні вимоги системи. Вони особливо ефективні в зборі відгуків від зацікавлених сторін, у тому числі тих, хто може не мати технічного досвіду, завдяки своєму простому інтуїтивно зрозумілому дизайну. Ці діаграми надають орієнтований на користувача погляд на систему, зосереджуючись на тому, що система буде робити, а не на тому, як вона буде побудована, що допомагає гарантувати, що система відповідає потребам користувачів.

Однією з головних переваг діаграм прецедентів є те, що вони підкреслюють зовнішню поведінку системи, пропонуючи чітку картину того, як система взаємодіятиме з користувачами чи іншими системами. Це робить їх цінним інструментом для перевірки вимог і забезпечення відповідності розробленої системи цілям і очікуванням її користувачів.

Спроєктована діаграма прецедентів представлена на рис.3.

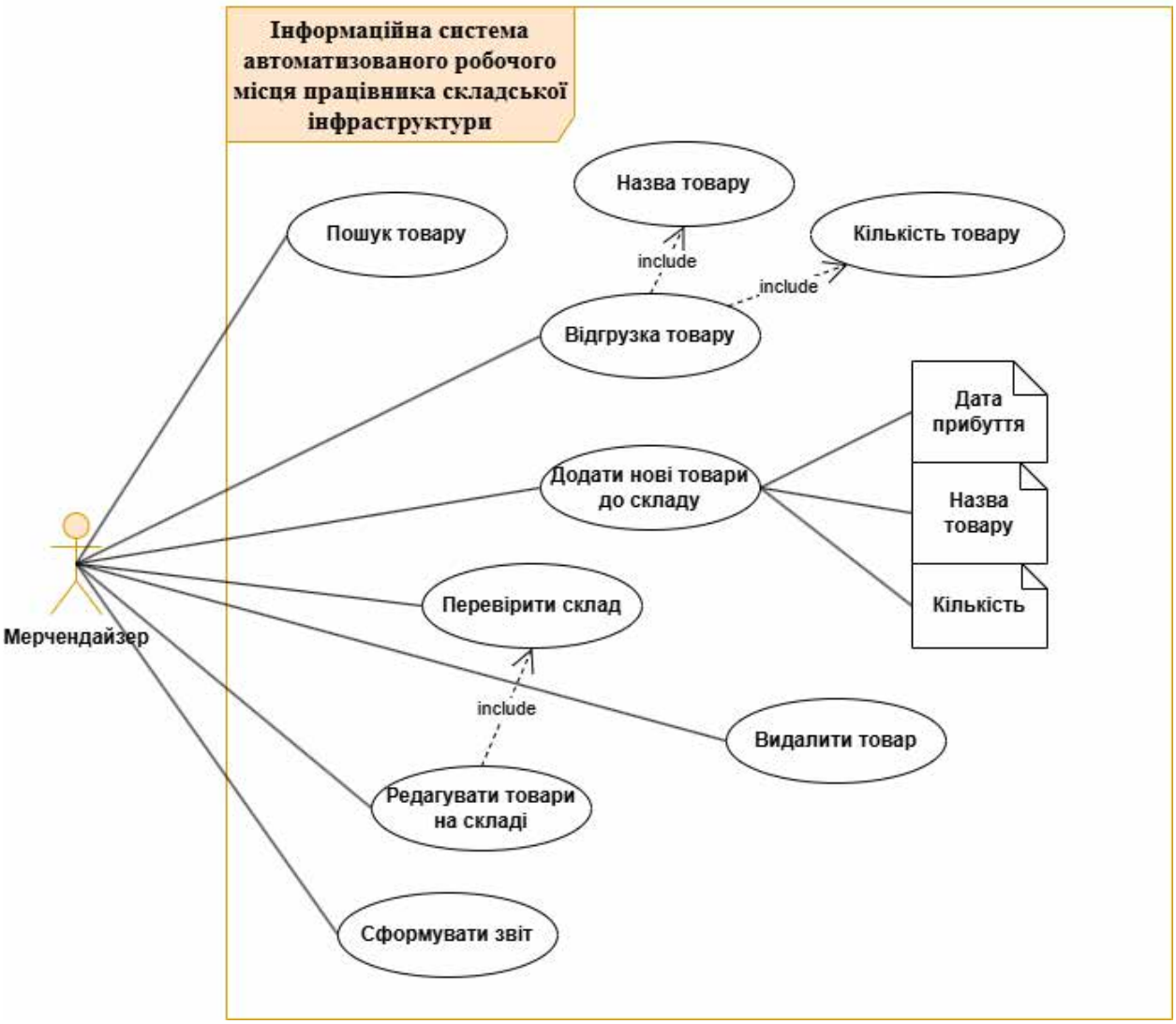


Рис. 3 Діаграма прецедентів

Створена діаграма прецедентів містить акторів:

- “Мерчендайзер”.

Актор «Мерчендайзер» включає такі прецеденти:

- пошук товару;
- відгрузка товару;
- додати нові товари до складу;
- перевірити склад;
- редагувати товари на складі;
- видалити товар;
- сформувати звіт.

Прецеденти певним чином залежать одне від одного.

Прецедент “відгрузка товару” доповнюється іншими такими прецедентами як “Назва товару” та “Кількість товару”.

Прецедент “ Додати нові товари до складу ” включає в себе такі анотації як “Дата прибуття”, “Назва товару”, “Кількість”.

Розглянемо детальніше вищеописані прецеденти.

Сценарій використання: «Додати нові товари на склад»

Актор: Мерчендайзер

Опис: Випадок використання «Додати нові товари на склад» описує процес, за допомогою якого мерчендайзер вводить нові продукти в систему управління складом (WMS). Ця дія має вирішальне значення для ведення точного обліку запасів і забезпечення відображення системою поточного рівня запасів. Процес починається, коли нова партія товарів надходить на склад або коли нові продукти вводяться в інвентар.

Основний перебіг подій:

1. мерчендайзер входить у систему складу з відповідними обліковими даними та переходить до розділу «Додати нові товари» програми;
2. система пропонує мерчендайзеру ввести детальну інформацію про новий продукт;
3. після введення необхідної інформації система перевіряє дані на повноту та правильність;
4. система перевіряє наявність відсутніх полів або недійсних записів, таких як неправильні коди продуктів або від’ємні кількості. У разі виявлення будь-яких помилок система пропонує мерчендайзеру виправити дані;
5. після перевірки даних мерчендайзер підтверджує додавання нових товарів на склад. Система зберігає інформацію про продукт у базі даних і відповідно оновлює рівень запасів;
6. система генерує квитанцію або підтвердження доданих товарів, яке може містити штрих-код або QR-код для легкої

ідентифікації на наступних етапах (таких як інвентаризація або виконання замовлення). Це підтвердження відображається продавцю, а також може бути роздруковано для фізичного зберігання;

7. система автоматично оновлює інвентарні записи на складі, щоб відобразити нові рівні запасів, гарантуючи, що продукт є в списку та готовий до використання в майбутніх замовленнях, відправках або перевірках запасів;
8. мерчендайзер може повернутися до головної панелі інструментів, виконати інші дії або вийти з системи. Нові товари тепер є частиною складських запасів і доступні для подальшої обробки.

Альтернативні потоки:

1. недійсне введення даних: якщо мерчендайзер вводить недійсну інформацію (наприклад, пропущені поля або від’ємні кількості), система відобразить повідомлення про помилку та запропонує користувачеві виправити записи;
2. тайм-аут системи: якщо система закінчилася через бездіяльність під час процесу введення даних, мерчандайзеру потрібно буде знову ввійти в систему, а всі незбережені дані буде втрачено.

Цей варіант використання гарантує, що система управління складом залишається в актуальному стані з точними даними про запаси, допомагаючи оптимізувати контроль запасів і спростити робочий процес для працівників складу.

Розглянемо інший сценарій випадку використання.

Сценарій використання: "Редагувати товари на складі"

Актор: Мерчендайзер

Опис: Випадок використання «Редагувати товари на складі» описує процес, за допомогою якого мерчендайзер змінює або оновлює наявну інформацію про продукт у системі керування складом (WMS). Це може

знадобитися в ситуаціях, коли потрібно виправити або оновити деталі продукту, наприклад, змінити кількість продукту, ціну, місце зберігання або опис продукту.

Основний перебіг подій:

1. мерчендайзер входить у систему управління складом, використовуючи свої облікові дані. Після входу вони переходять до розділу, де відображається список продуктів на складі;
2. мерчендайзер шукає конкретний продукт, який потрібно відредагувати. Це можна зробити за допомогою різних критеріїв пошуку, таких як назва продукту, код продукту, категорія або місце зберігання;
3. коли товар знайдено, мерчендайзер вибирає його зі списку. Система відображає поточні деталі продукту, такі як назва продукту, код, кількість, ціна, постачальник і місце зберігання;
4. тоді мерчендайзер зможе редагувати відповідні поля;
5. після того, як система перевірить дані, мерчендайзер переглядає внесені зміни та підтверджує, що вони правильні. Мерчендайзер натискає кнопку «Зберегти» або «Підтвердити», щоб надіслати зміни;
6. система оновлює базу даних новою інформацією про продукт. Оновлені відомості про продукт тепер відображаються в системі інвентаризації складу, а записи системи відповідно синхронізуються;
7. після успішного редагування система генерує повідомлення-підтвердження або квитанцію з оновленою інформацією про продукт. Мерчандайзер може роздрукувати це підтвердження для ведення записів або зберегти його в цифровому вигляді;

8. потім мерчендайзер повертається до розділу керування запасами системи, де він може або продовжити редагування інших продуктів, або вийти, якщо вони закінчили.

Альтернативні потоки:

1. недійсне введення даних: якщо мерчендайзер вводить недійсні дані (наприклад, від'ємну кількість або неправильний формат ціни), система відобразить повідомлення про помилку та запропонує продавцю виправити проблему, перш ніж продовжити;
2. тайм-аут системи: якщо під час процесу редагування система припинила роботу, мерчендайзеру потрібно буде повторно ввійти. Усі незбережені зміни буде втрачено.

Цей варіант використання гарантує, що система керування складом залишається точною та відображає будь-які зміни в інформації про продукт. Редагування деталей продукту допомагає підтримувати актуальні записи запасів і забезпечує безперебійну роботу на складі.

2.2.2 Діаграма послідовності. Діаграма послідовності — це тип діаграми UML (уніфікована мова моделювання), яка представляє потік взаємодій між різними компонентами або об'єктами в системі протягом певного часу. Він забезпечує детальне уявлення про те, як відбуваються процеси, показуючи, як обмінюються повідомленнями між акторами та об'єктами, як правило, у межах сценарію використання. Діаграми послідовності особливо корисні для ілюстрації покрокової послідовності операцій у системі та того, як ці операції залежать одна від одної [12].

У контексті інформаційної системи для автоматизованого робочого місця працівника складської інфраструктури діаграми послідовності можна використовувати для візуалізації конкретних взаємодій у системі управління складом (WMS). Ці взаємодії можуть включати такі дії, як

додавання нових товарів, редагування наявних деталей продукту або обробка замовлення.

Основні елементи діаграми послідовності включають акторів, які є зовнішніми суб’єктами, що взаємодіють із системою, наприклад продавцем або самою складською системою. Об’єкти або лінії життя представляють компоненти в системі, і часова шкала кожного об’єкта зображена вертикальною пунктирною лінією, яка показує його тривалість життя протягом послідовності подій. Повідомлення представлені стрілками між цими лініями життя, що вказує на зв’язок між об’єктами. Ці повідомлення можуть бути синхронними (вимагають відповіді) або асинхронними (без необхідної відповіді). Панелі активації — це вертикальні прямокутники, які з’являються на лінії життя, вказуючи, коли об’єкт активно обробляє повідомлення. Зворотні повідомлення відображаються як пунктирні стрілки, що вказує на повернення інформації або результатів із попереднього повідомлення.

Діаграма послідовності (рис. 4) читається зверху вниз, а вертикальна вісь відображає час. Потік подій рухається вниз, причому кожна горизонтальна лінія представляє об’єкт або компонент у системі.

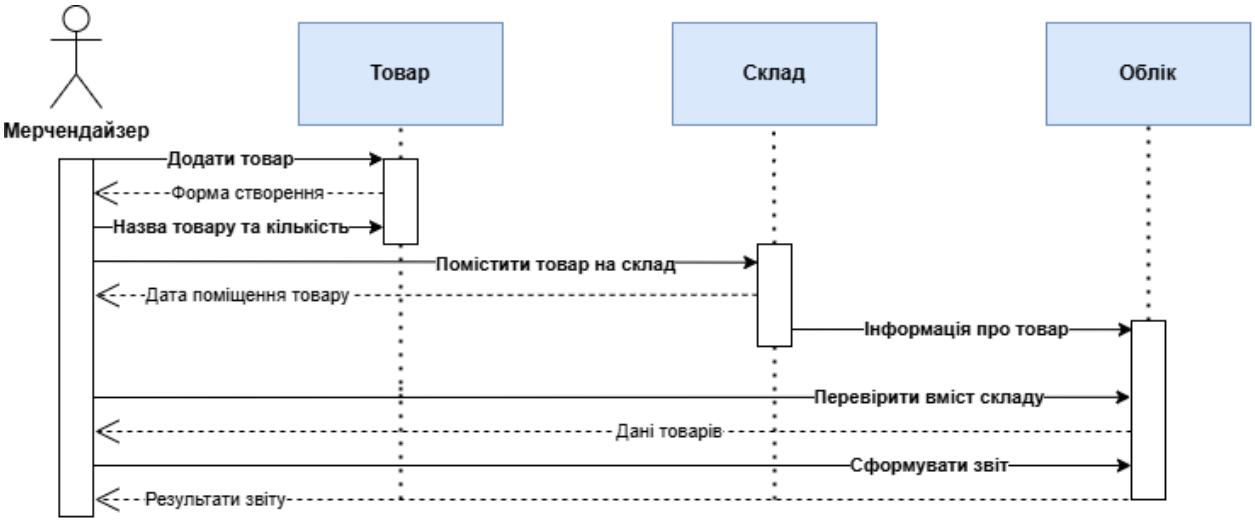


Рис. 4 Діаграма послідовності

Діаграма послідовностей, зображена на рис. 4, ілюструє процес взаємодії між кількома об'єктами в системі при виконанні завдання,

пов'язаного з додаванням нового товару на склад. У цій діаграмі беруть участь наступні об'єкти:

1. мерчендайзер: є ініціатором процесу, який додає новий товар до складу. Він взаємодіє із системою для введення даних про товар;
2. товар: об'єкт, що представляє новий товар, який додається на склад. Він містить всю необхідну інформацію про товар, таку як назва, кількість, ціна тощо;
3. склад: об'єкт, який відповідає за зберігання та управління товаром на складі. У даному випадку, склад отримує дані про товар від мерчендайзера і оновлює інформацію про наявність товару;
4. облік: система обліку товарів, яка веде реєстрацію та зберігання даних про наявні товари, їх кількість та інші важливі відомості. Вона синхронізується зі складом і забезпечує актуальність даних.

Процес, зображений на діаграмі, починається з того, що мерчендайзер ініціює дію для додавання нового товару через інтерфейс системи. Після цього він вводить необхідну інформацію про товар, таку як його назву, кількість, і місцезнаходження на складі. Дані передаються до об'єкта товар, який отримує і зберігає цю інформацію. Потім ці дані передаються до об'єкта склад, який оновлює інформацію про наявність товару на складі. Зі свого боку, облік отримує інформацію від складу та оновлює базу даних, що дозволяє підтримувати точний запис про кількість товарів.

У результаті, система автоматично оновлює всі необхідні дані, і мерчендайзер отримує підтвердження про успішне додавання товару на склад. Цей процес дозволяє забезпечити точний контроль над наявністю товарів і автоматизувати рутинні операції.

2.2.3 Діаграма активності. Діаграма діяльності — це тип діаграми UML (Unified Modeling Language), який використовується для моделювання динамічних аспектів системи. Він візуально представляє, як керування або

дані проходять через процес або робочий процес під час виконання. Діаграми діяльності особливо корисні для ілюстрації бізнес-процесів, робочих процесів і серії дій, які відбуваються в конкретному випадку використання або функції системи. Вони допомагають візуалізувати послідовність дій у процесі, особливо коли є кілька рішень, паралельні завдання або повторювані кроки [13].

Діаграма складається з кількох ключових елементів, у тому числі початкових вузлів, які позначають початок дії, представлених зафарбованим колом. Дії або дії відображаються у вигляді прямокутників із заокругленими кутами та представляють завдання, які виконуються в процесі. Вузли прийняття рішень у формі ромба вказують точки розгалуження, де потік може йти різними шляхами залежно від певних умов. Вузли злиття схожі на вузли прийняття рішень, але їх призначення полягає в об’єднанні кількох потоків в один.

Іншим важливим елементом є розгалуження та вузли з’єднання, які допомагають моделювати паралельні процеси. Вузли розгалуження розбивають процес на кілька паралельних гілок, а вузли об’єднання повертають ці гілки разом. Умови об’єктів представлені смужками або пунктирними лініями та показують, як об’єкти чи умови впливають на потік. У кінці дії або процесу кінцевий або кінцевий вузол, показаний у вигляді зафарбованого кола з рамкою, позначає завершення потоку.

Потоки керування — це стрілки, які з’єднують різні елементи на діаграмі, вказуючи напрямок потоку від однієї діяльності до іншої. Крім того, смуги можна використовувати для поділу діаграми на різні області, що представляють різних учасників, відділи або системи, які відповідають за конкретні завдання. Swimlanes допомагає з’ясувати, яка організація відповідає за кожну дію в робочому процесі.

Однією з головних переваг діаграм діяльності є їх здатність запропонувати чітке візуальне представлення робочого процесу системи. Вони особливо цінні для ілюстрації процесів, які включають паралельну

діяльність, показуючи, як різні завдання можуть виконуватися одночасно або після виконання певних умов. Ці діаграми чудово підходять для моделювання бізнес-процесів, забезпечуючи прямий погляд на послідовність завдань, необхідних для досягнення мети. Їх легко зрозуміти навіть для тих, хто не має технічних знань, що робить їх чудовим інструментом для спілкування. На рисунку 5 представлена діаграма діяльності.

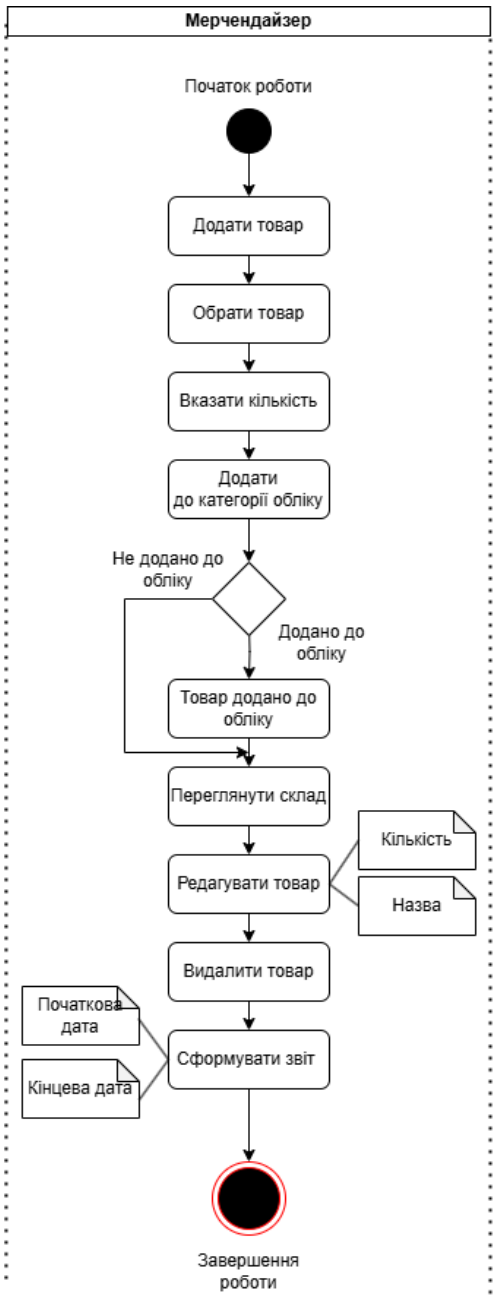


Рис. 5 Діаграма діяльності

2.3 Абстракції предметної області

Абстракція в розробці програмного забезпечення стосується спрощення складних систем шляхом приховування непотрібних деталей і зосередження на суттєвих аспектах. У контексті інформаційної системи автоматизованого робочого місця працівника складської інфраструктури абстракція використовується для розбиття складних завдань і компонентів, задіяних у складських операціях, на більш керовані та зрозумілі елементи. Такий підхід дозволяє спростити проєктування, впровадження та обслуговування системи [14].

На високому рівні абстракція допомагає у спрощеному представленні об’єктів реального світу, їх взаємодії та процесів. У випадку управління складом необхідно визначити кілька основних абстракцій, щоб ефективно моделювати систему. Ці абстракції зазвичай узгоджуються з ключовими функціональними областями системи, такими як управління запасами, обробка замовлень, надходження товарів і операції відправлення.

Наприклад, працівник складської інфраструктури є учасником системи. У абстрактному поданні завдання працівника, такі як керування запасами, отримання та відправка товарів і проведення інвентаризації, інкапсульовані в рамках конкретних робочих процесів. Система абстрагує складні завдання шляхом розподілу операцій за категоріями та розподілу ролей, гарантуючи, що кожна дія відстежується та узгоджується з цілями складських операцій.

Зосереджуючись на цих абстракціях, система може задовольнити потреби складських операцій у структурований спосіб. Взаємодії між абстрактними об’єктами розроблено таким чином, щоб відображати основні операції, дозволяючи спростити інтерфейс користувача, оптимізувати робочі процеси та оптимізувати процеси. Цей абстрактний підхід також допомагає переконатися, що користувачі можуть зосередитися на високорівневих цілях,

не зациклюючись на непотрібних деталях, що робить систему більш ефективною та зручнішою.

Зрештою, абстракція в контексті систем управління складом (рис. 6) допомагає створити ефективне, кероване та масштабоване рішення. Зменшуючи складність операцій, система покращує як досвід користувача, так і ефективність роботи, забезпечуючи злагоджену роботу всіх елементів складу.

Рис. 6 Абстракції предметної області

2.4 Діаграма класів

Діаграма класів є ключовим елементом уніфікованої мови моделювання (UML), яка представляє статичну структуру системи. Вона надає візуальне представлення класів системи, їхніх атрибутів, методів і того, як ці класи співвідносяться один з одним. Цей тип діаграми необхідний для розуміння архітектури системи та служить схемою для її реалізації. Це допомагає проілюструвати зв’язки між різними об’єктами, їхніми атрибутами даних і поведінкою, яку вони підтримують.

На діаграмі класів кожен клас зображено у вигляді прямокутника, поділеного на три частини. У верхньому розділі відображається ім’я класу, у середньому – перелік його атрибутів, а в нижньому – його методи чи операції. Атрибути представляють дані, які зберігає клас, а методи визначають дії, які

може виконувати клас. Ця структура забезпечує чітке розмежування між тим, що таке клас (його дані), і тим, що він може робити (його функціональність).

Класи поєднуються різними способами, щоб показати, як вони взаємодіють один з одним. Асоціації представлені суцільними лініями, що з'єднують класи, що вказує на те, що вони певним чином пов'язані. Множинності, такі як «1» або «0..*», можна додати до асоціацій, щоб визначити, скільки екземплярів одного класу пов'язано з екземплярами іншого. Спадкування показано лінією, яка закінчується порожнистим трикутником, ілюструючи, що один клас походить від іншого. Спадкування дозволяє класу успадковувати атрибути та методи від батьківського класу, полегшуючи повторне використання коду та встановлюючи ієрархічний зв'язок між класами [15].

Агрегація та композиція є спеціалізованими типами зв'язків, які описують зв'язки «частина-ціле». Агрегація представлена лінією з відкритим ромбом, тоді як композиція використовує заповнений ромб, щоб вказати сильніший, більш залежний зв'язок. Ці відносини пояснюють, як один клас складається з інших класів, причому композиція означає, що існування одного класу залежить від іншого.

Залежність показана як пунктирна лінія зі стрілкою, що означає, що один клас залежить від іншого. Ця залежність означає, що зміна в залежному класі може вплинути на клас, який залежить від нього, що корисно для виявлення залежностей у проєкті системи.

Діаграми класів надають статичне уявлення про структуру системи, зосереджуючись на тому, як організовані сутності та як вони взаємодіють один з одним. Вони особливо корисні на етапі проєктування розробки, оскільки дозволяють системним архітекторам і розробникам візуалізувати компоненти системи та їхні взаємозв'язки. Чітко показуючи дані та поведінку класів та їх взаємодію, діаграми класів допомагають ефективно структурувати систему.

Було створено власну діаграму класів за об'єктно орієнтованим підходом для даного застосунку (рис. 7).

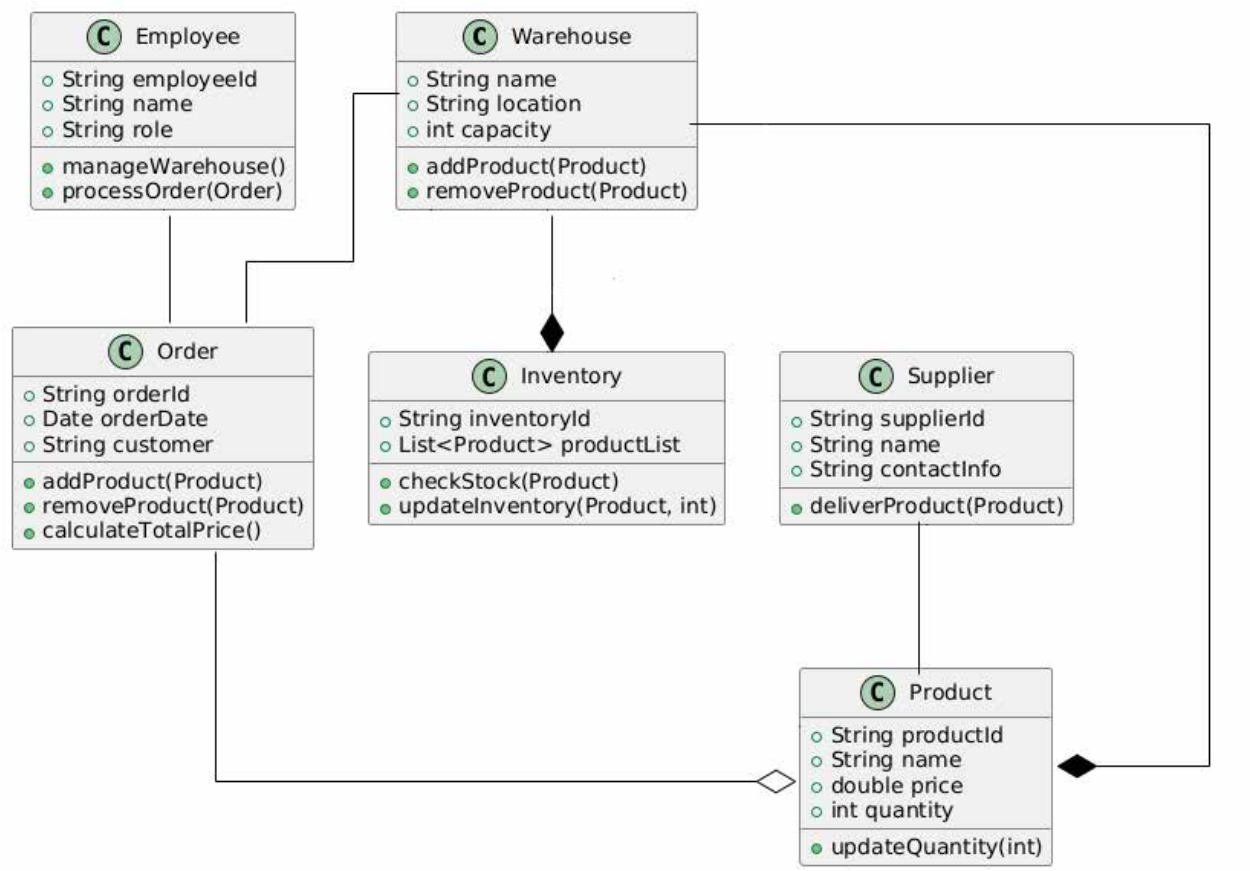


Рис. 7 Діаграма класів

Клас Warehouse є фундаментальним компонентом системи, що представляє фізичне сховище. Він містить такі ключові атрибути, як назва складу, місцезнаходження та загальна ємність зберігання. Клас складів також містить методи керування запасами, наприклад додавання та видалення продуктів на основі потреб інвентаризації. Цей клас дозволяє системі обробляти логістику підтримки доступності продукту та оптимізації простору.

Клас Product інкапсулює деталі кожного товару на складі. Він відстежує унікальний ідентифікатор продукту, назву, ціну та кількість доступних запасів. За допомогою методу updateQuantity клас гарантує, що запаси продукту оновлюються щоразу, коли відбуваються зміни, будь то нові поставки чи продажі. Цей клас є центральним у функціонуванні системи, оскільки він зберігає основні дані того, що керується на складі.

Order являє собою запит клієнта на один або кілька продуктів. Цей клас включає такі ключові атрибути, як ідентифікатор замовлення, дата розміщення та відомості про клієнта. Його методи дозволяють додавати або

видаляти продукти із замовлення, а також функцію для розрахунку загальної вартості замовлення. Це життєво важливо для керування транзакціями на складі, гарантуючи ефективну й точну обробку всіх замовлень.

Клас Employee представляє складських працівників або менеджерів. Кожен співробітник має унікальний ідентифікатор, ім’я та роль на складі. Співробітникам доручено керувати операціями, такими як нагляд за інвентаризацією та обробкою замовлень. Методи manageWarehouse і processOrder надають співробітникам інструменти, необхідні для ефективного виконання цих завдань, забезпечуючи безперебійну роботу в складському середовищі.

Клас Supplier відповідає за доставку товару на склад. Він містить такі атрибути, як унікальний ідентифікатор постачальника, ім’я постачальника та контактна інформація. Постачальники відіграють вирішальну роль у забезпеченні укомплектованості складу продуктами, а метод deliverProduct допомагає відстежувати процес доставки. Керуючи відносинами з постачальниками, склад може забезпечити своєчасне надходження продукції.

Клас Inventory служить основою системи для відстеження рівня запасів. У ньому є інвентарний ідентифікатор і список продуктів, які зараз є на складі. За допомогою таких методів, як checkStock і updateInventory, він дозволяє відстежувати рівень запасів у реальному часі та допомагає керувати коливаннями запасів. Це гарантує, що склад завжди знає про наявний запас і може приймати обґрунтовані рішення щодо повторного замовлення або поповнення запасів.

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Логічна модель даних

Логічна модель даних представляє структуру й організацію даних у системі, абстрагуючись від фізичних деталей і зосереджуючись на тому, як дані логічно взаємопов’язані. Це важлива частина проєктування бази даних, яка служить схемою для розробки фізичної моделі даних і, зрештою, керує створенням схеми бази даних.

Логічна модель даних для системи управління складом описує ключові об’єкти, задіяні в системі, та їхні взаємозв’язки. Ці сутності зазвичай включають склад, продукт, замовлення, працівника, постачальника та запаси.

Кожна сутність має власний набір атрибутів, які описують її характеристики. Наприклад, сутність Warehouse може мати такі атрибути, як warehouse_id, warehouse_name, location і ємність, тоді як сутність Product може містити такі атрибути, як product_id, product_name, description, price і quantity_in_stock.

Відносини між цими сутностями визначають, як вони взаємодіють один з одним. Наприклад, Склад містить багато Продуктів, і Продукт можна пов’язати з кількома Замовленнями. Замовлення розміщує Клієнт (який представлений окремою особою), і замовлення може містити один або кілька Продуктів. Співробітники несуть відповідальність за управління складом і обробку замовлень.

Логічна модель даних також пов’язана з цілісністю даних, гарантуючи, що кожен зв’язок між об’єктами є значущим і відповідає бізнес-правилам. Наприклад, замовлення не повинно існувати без пов’язаного клієнта, а товар завжди повинен мати дійсну ціну та кількість на складі [16].

З точки зору нормалізації, логічна модель даних спрямована на зменшення надмірності та забезпечення ефективного зберігання даних. Це часто передбачає поділ даних на окремі таблиці (або сутності) і забезпечення

правильного визначення зв’язків між сутностями за допомогою ключів і обмежень. Зовнішні ключі використовуються для встановлення зв’язків між сутностями, а первинні ключі однозначно ідентифікують записи в кожній таблиці.

Логічна модель даних не заглиблюється в технічні аспекти, такі як індексування, типи даних або оптимізація продуктивності. Він забезпечує чітке, орієнтоване на бізнес представлення вимог до даних, які потім можна перевести у фізичну модель даних для реалізації в системі баз даних, такій як MySQL, SQL Server або Oracle.

Загалом, логічна модель даних служить орієнтиром для забезпечення структурування даних у спосіб, який підтримує бізнес-процеси, а також забезпечує міцну основу для фактичного проєктування та розробки бази даних. Це гарантує, що зв’язки даних є логічно узгодженими та що система може ефективно виконувати необхідні бізнес-операції.

Створена логічна модель інформаційного забезпечення буде відповідати цим вимогам, забезпечуючи чітке та зручне представлення інформаційної системи.

Логічна модель системи представлена на рис. 8.

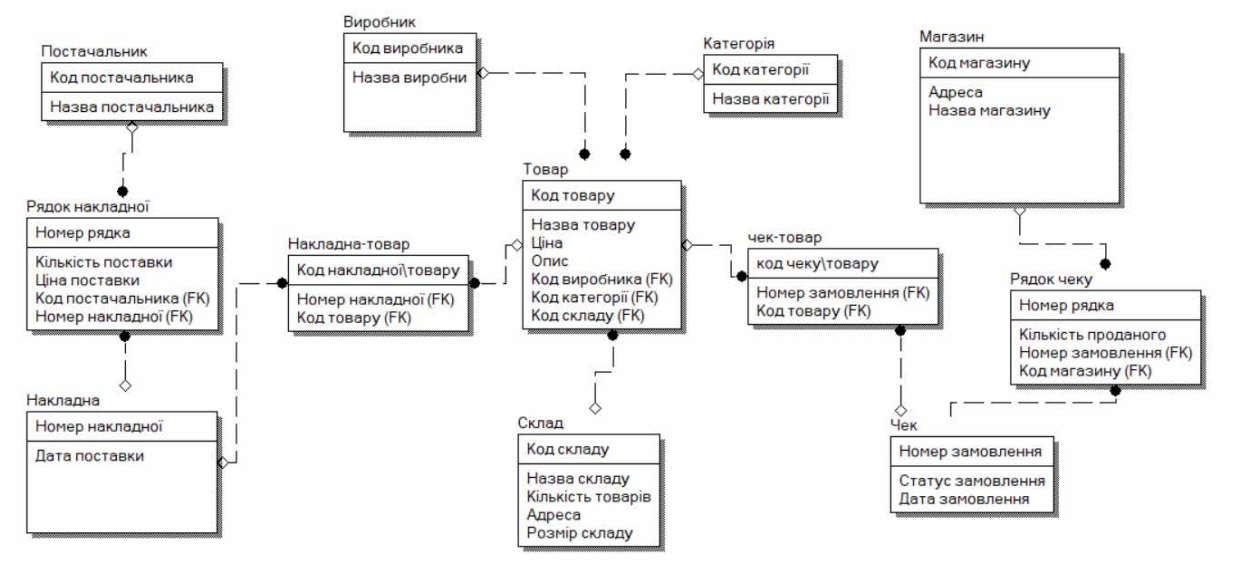


Рис. 8 ER-діаграма

Логічна модель складається з таких сутностей:

Товар - це основна одиниця обліку в системі, що представляє продукцію, яка зберігається і продається у складі чи магазині. Сутність включає такі атрибути, як назва товару, унікальний ідентифікатор (ID), опис, ціна, кількість на складі, категорія і виробник. Товари можуть бути згруповані за категоріями для зручності класифікації та пошуку.

Склад є фізичне чи віртуальне місце, де зберігається товар. Кожен склад має унікальний ідентифікатор, назву, розташування та місткість. Склад може містити кілька товарів та підтримувати дані про кількість кожного товару на складі.

Магазин - це точка продажу, яка може бути пов'язана з одним чи кількома складами. По суті магазину зберігаються такі дані, як його назва, адреса, контактна інформація, а також зв'язок з товарами, які він продає, та кількістю їх у наявності.

Постачальник — це організація чи індивідуум, що надає товари на продаж у магазин чи його постачання складу. Атрибути цієї сутності можуть включати назву компанії, контактні дані, адресу та список товарів, які постачаються цим постачальником.

Виробник - це компанія або виробник, що створює товари, які потім надходять у продаж через магазини та склади. По суті виробника містяться такі атрибути, як назва компанії, адреса, контактні дані та перелік товарів, які вона виробляє.

Категорії товарів служать для класифікації товарів залежно від їхнього типу чи призначення. Кожна категорія має унікальну назву та опис, що допомагає у систематизації та фільтрації товарів для спрощення пошуку та управління ними.

Накладна - це документ, що підтверджує факт постачання товару від постачальника на склад чи магазин. Сутність накладної може включати такі дані, як унікальний номер накладної, дата оформлення, постачальник, склад або магазин, до якого надіслано товар, та список товарів з кількістю та ціною.

Рядок накладної є окремим записом про товар, який поставляється в рамках конкретної накладної. Атрибути цієї сутності включають унікальний ідентифікатор, кількість товару, ціну та можливі знижки. Кожен товар у накладній може бути представлений окремим рядком.

Чек - це документ, що підтверджує факт продажу товарів покупцю. Чек містить інформацію про куплені товари, їх кількість, ціну, податки, а також загальну суму покупки. Атрибути можуть включати номер чека, дату продажу, інформацію про магазин та покупця.

Рядок чека є окремим рядком, у якому зазначено конкретний товар, проданий покупцю. У цій сутності зберігаються дані про товар, його кількість, ціну та підсумкову вартість по кожній позиції в чеку. Рядки чека формують повний перелік товарів, які купують покупець.

3.2 Вибір системи управління базою даних та її реалізація

При виборі системи управління базами даних (СУБД) для інформаційної системи автоматизованого робочого місця працівника складської інфраструктури необхідно враховувати кілька важливих факторів. Обрана СУБД повинна забезпечувати ефективне зберігання даних, швидкий пошук, масштабованість і безпеку, а також підтримувати інтеграцію з іншими компонентами системи. Правильна СУБД може значно вплинути на загальну продуктивність і функціональність системи. Щоб прийняти обґрунтоване рішення, важливо враховувати характер і обсяг даних, вимоги до системи та бюджет організації.

Система управління складом, швидше за все, оброблятиме великий обсяг структурованих даних, таких як деталі запасів, обробка замовлень, дані про продукт, постачальника та користувачів. Оскільки реляційні бази даних ідеально підходять для зберігання та керування структурованими даними, які вимагають зв’язків між різними об’єктами, реляційна система керування базами даних (RDBMS) є відповідним вибором для цього сценарію. РСУБД

забезпечить ефективну обробку даних, забезпечуючи цілісність даних і спрощуючи складні запити та звіти.

Крім того, система повинна підтримувати великі обсяги транзакцій, такі як оновлення запасів, обробка замовлень і управління запасами, що означає, що СУБД повинна мати можливість масштабування без шкоди для продуктивності. Він повинен обробляти зростаючі обсяги даних у міру розширення складських операцій, включаючи керування великими запасами та обробку кількох взаємодій користувачів одночасно.

Безпека також має першорядне значення, особливо тому, що складська система зберігатиме конфіденційні дані про продукти, клієнтів і постачальників. Важливо, щоб обрана СУБД забезпечувала надійні функції безпеки, такі як шифрування даних, автентифікація користувачів і контроль доступу на основі ролей, гарантуючи, що лише авторизований персонал може отримати доступ або змінити конфіденційні дані. Крім того, СУБД повинна підтримувати функції для підтримки узгодженості та цілісності даних, забезпечуючи плавну та надійну роботу бази даних.

Ще одним важливим фактором є інтеграція СУБД з іншими компонентами складської системи. Він має бути сумісним із серверними технологіями, такими як C# і WinForms, і бездоганно інтегруватися з існуючими інструментами та програмним забезпеченням, що використовується складом, таким як керування ланцюгом поставок або системи ERP.

Вартість і моделі ліцензування також відіграють важливу роль при виборі СУБД. СУБД з відкритим вихідним кодом можуть запропонувати значні переваги в ціні, тоді як пропрієтарні варіанти можуть надати більш розширені функції, але за вищою ціною. Вибір СУБД повинен відповідати як початковому бюджету, так і довгостроковим фінансовим планам організації.

Кілька варіантів RDBMS добре відповідають вимогам системи управління складом. Наприклад, Microsoft SQL Server — це RDBMS корпоративного рівня, відома своєю масштабованістю, безпекою та

продуктивністю. Він добре інтегрується з програмами .NET, що робить його природним вибором для систем, розроблених за допомогою C# і WinForms. SQL Server пропонує надійні функції, такі як шифрування, аудит даних і розширені інструменти звітності, що робить його ідеальним для великомасштабних і складних операцій [17].

Рішення обрати Microsoft SQL Server (MS SQL Server) як систему керування базами даних (СУБД) для інформаційної системи автоматизованого робочого місця працівника складської інфраструктури базувалося на кількох ключових факторах, які відповідають вимогам системи, операційним потребам і довгостроковим цілям масштабованості.

По-перше, масштабованість і продуктивність є критично важливими для системи управління складом, оскільки вона потребує обробки великих обсягів даних, таких як інвентарні записи, замовлення, відправлення та інформація про постачальників. MS SQL Server добре відомий своєю високою продуктивністю та масштабованістю, що робить його чудовим вибором для систем, які з часом будуть зростати в розмірі даних і складності. Розширене індексування MS SQL Server, функції оптимізації запитів і підтримка великомасштабних операцій гарантують, що він може впоратися зі зростаючим навантаженням транзакцій, що є типовим для завантаженого середовища складів.

Крім того, MS SQL Server відомий своїми надійними функціями безпеки, що є ще однією причиною, чому його було обрано для цього проєкту. Складські системи працюють з конфіденційними даними, такими як рівень запасів, деталі клієнта та інформація про замовлення. MS SQL Server надає повний набір інструментів безпеки, включаючи шифрування, автентифікацію користувачів, контроль доступу на основі ролей і маскування даних, які необхідні для захисту конфіденційної інформації від несанкціонованого доступу та потенційних порушень.

Іншим ключовим фактором була інтеграція з технологіями, які використовуються в проєкті. Оскільки система розробляється з

використанням C# і WinForms, MS SQL Server є природним вибором завдяки бездоганній інтеграції з платформою .NET. Ця тісна інтеграція забезпечує більш плавну розробку та полегшує зв’язок між програмою та базою даних, оптимізуючи загальний процес розробки [18].

Надійність і цілісність даних мають вирішальне значення в управлінні складом, оскільки будь-які розбіжності в даних або системні збої можуть призвести до значних операційних проблем. MS SQL Server пропонує вбудовані функції для підтримки цілісності даних, такі як відповідність ACID (Atomicity, Consistency, Isolation, Durability), що гарантує надійну обробку транзакцій. Крім того, MS SQL Server підтримує автоматичне резервне копіювання, інструменти аварійного відновлення та відмовостійкі конфігурації для забезпечення високої доступності та мінімізації часу простою, що є життєво важливим для складських операцій, які залежать від постійного потоку даних.

Знайомість і підтримка MS SQL Server також зіграли свою роль у його виборі. Microsoft SQL Server широко використовується в корпоративних програмах, і для розробників доступна велика кількість ресурсів, документації та підтримки спільноти. Це дозволяє групі розробників отримати доступ до великої кількості знань і допомоги під час вирішення технічних завдань або оптимізації системи.

Крім того, MS SQL Server надає повний набір інструментів керування, таких як SQL Server Management Studio (SSMS), який спрощує проєктування бази даних, виконання запитів та адміністрування. Ці інструменти допомагають командам розробників і операцій ефективно керувати базою даних, від розробки схеми до моніторингу та налаштування продуктивності.

Також були оцінені міркування щодо вартості. Хоча MS SQL Server є пропрієтарною системою, вартість ліцензування виправдовується її високою продуктивністю, багатим набором функцій і довгостроковою надійністю. У контексті системи управління складом корпоративного рівня інвестиції в MS SQL Server розглядаються як суттєві витрати для забезпечення того, щоб

система відповідала своїм експлуатаційним вимогам і забезпечувала необхідну підтримку для майбутнього зростання.

Підсумовуючи, MS SQL Server був обраний для цього проєкту завдяки його масштабованості, безпеці, інтеграції з C# і WinForms, надійності та наявності інструментів керування. Ці атрибути роблять його ідеальним вибором для системи управління складом, яка повинна обробляти великі обсяги даних, забезпечувати цілісність даних і підтримувати майбутнє розширення. Рішення використовувати MS SQL Server забезпечує міцну основу для створення надійної та ефективної автоматизованої системи робочого місця для працівників складської інфраструктури.

Під час проєктування таблиць у фізичній моделі основою слугували сутності з логічної моделі. Атрибути цих сутностей були використані для розробки структури таблиці. Отриману схему бази даних зображено на рис. 9.

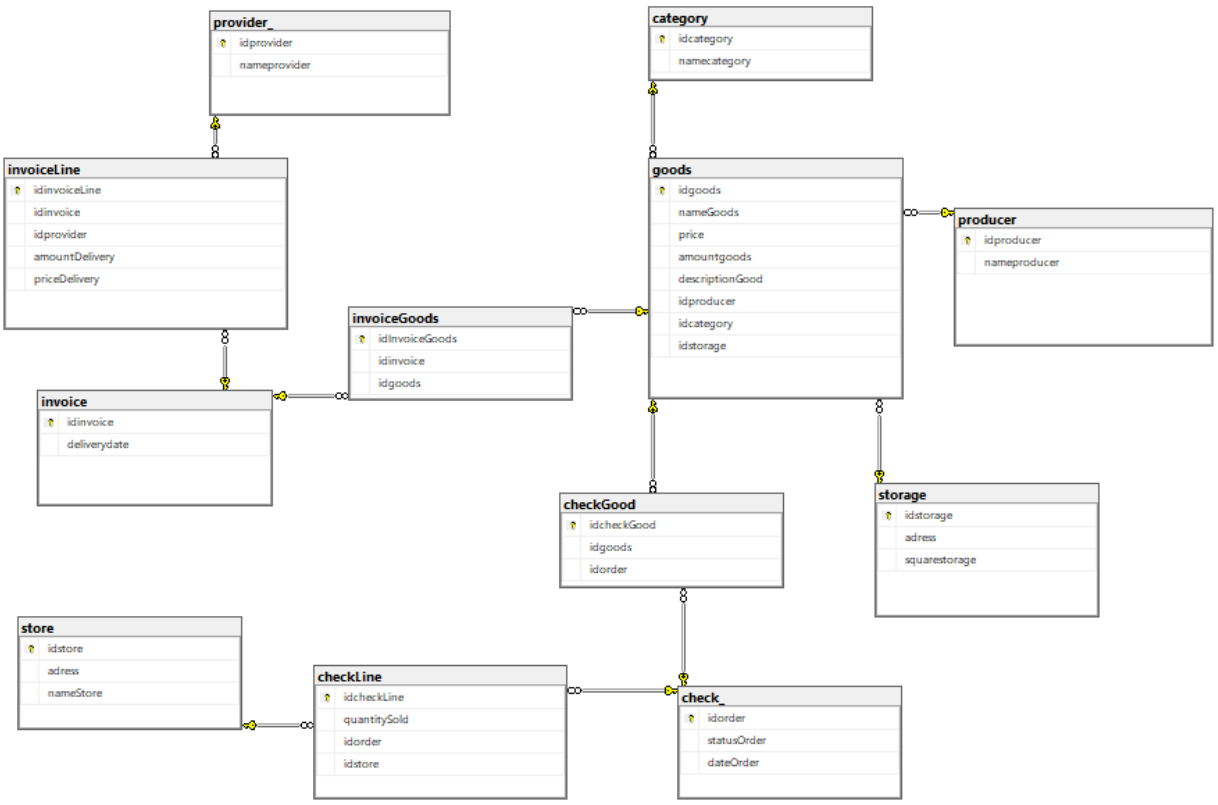


Рис. 9 База даних системи

База даних складається з кількох взаємопов’язаних таблиць, кожна з яких виконує певну мету. Таблиця «постачальник» містить інформацію про постачальників, включаючи їх унікальні ідентифікатори та імена. Категорії

товарів керуються в таблиці «категорія» з полями для ідентифікаторів категорій і імен. Товари детально описано в таблиці «товари», яка містить таку інформацію, як унікальні ідентифікатори товарів, назви, ціни, кількість, описи та пов’язані ідентифікатори для їхніх виробників, категорій і магазинів.

Інформація про виробника зберігається в таблиці «виробник», яка записує ідентифікатори та імена виробників. Таблиця "invoiceLine" відстежує деталі доставки, включаючи ідентифікатор рядка, пов’язаний з ним рахунок-фактуру, постачальника, кількість доставки та ціну доставки. Самі рахунки-фактури документуються в таблиці «рахунок-фактура», яка містить ідентифікатор рахунку-фактури та дати доставки. Щоб зіставити товари з рахунками-фактурами, таблиця «invoiceGoods» ідентифікує товари, пов’язані з кожним рахунком-фактурою, за допомогою їхніх відповідних ідентифікаторів.

Дані про зберігання та зберігання записуються в таблиці "зберігання" та "зберігання". Деталі зберігання охоплюють ідентифікатор складу, пов’язаний магазин і зону його зберігання, тоді як інформація про магазин включає ідентифікатор магазину, назву та адресу. Крім того, база даних містить таблицю "checkGood" для контролю за перевірками товарів. Ця таблиця пов’язує чеки з їхніми лініями, товарами та замовленнями. Крім того, таблиця "checkLine" відстежує окремі деталі транзакцій, такі як ідентифікатор рядка, продана кількість, а також відповідні замовлення та магазини. Таблиця «перевірка» консолідує інформацію про замовлення, включаючи ідентифікатори замовлень, статуси та дати.

Зв’язки між цими таблицями мають вирішальне значення для структури бази даних. Наприклад, постачальники пов’язані з рядками рахунків, а категорії – з товарами. Самі товари взаємопов'язані з виробниками, категоріями і магазинами. Рядки рахунків-фактур пов’язані з рахунками-фактурами, а товари в рахунках-фактурах з’єднують зв’язок між рахунками-фактурами та окремими товарами. Перевірки товарів пов’язані з відповідними рядками транзакцій, товарами та замовленнями, тоді як записи про зберігання

відповідають конкретним магазинам. Нарешті, рядки транзакцій і замовлення пов’язані з магазинами, з яких вони походять.

Ця структура бази даних організована для оптимізації керування постачальниками, товарами, категоріями, рахунками-фактурами та замовленнями, одночасно забезпечуючи ясність і легкість навігації через взаємопов’язані зв’язки.

3.3 Архітектура програмного забезпечення

Програмна архітектура Інформаційної системи для автоматизованого робочого місця працівника складської інфраструктури розроблена з урахуванням масштабованості, продуктивності та простоти інтеграції. Метою системи є оптимізація складських операцій шляхом автоматизації ключових процесів, таких як управління запасами, обробка замовлень і відстеження, що в кінцевому підсумку забезпечить працівникам складу безперебійне та ефективне робоче середовище.

Система має багаторівневу архітектуру для розподілу обов’язків і забезпечення гнучкості. В основі дизайну ділиться на кілька ключових рівнів: рівень презентації, рівень бізнес-логіки, рівень даних і додатковий рівень інтеграції для зовнішнього зв’язку. Кожен рівень відповідає за певні завдання, сприяючи загальній модульності та ремонтопридатності системи [25].

Презентаційний рівень служить інтерфейсом, з яким взаємодіють працівники складу. Це створено за допомогою Windows Forms (WinForms), що забезпечує настільну програму, яка є одночасно зручною та ефективною. Інтерфейс розроблено для спрощення взаємодії з користувачем, пропонуючи такі інструменти, як форми, кнопки та звіти, які допомагають персоналу складу ефективно керувати своїми завданнями. За допомогою цього рівня користувачі можуть вводити дані, переглядати звіти та керувати різними операціями. Він використовує кероване подіями програмування для

ініціювання внутрішніх дій, реагуючи на введення користувачів, такі як клацання або надсилання форми.

Далі рівень бізнес-логіки обробляє основні функції системи. Він обробляє запити від презентаційного рівня, гарантуючи дотримання всіх бізнес-правил і процесів. Наприклад, він перевіряє запаси перед розміщенням замовлення або оновлює рівень запасів після отримання товарів. Бізнес-логіка реалізована за допомогою C#, інкапсулюючи операції в сервіси та компоненти, що забезпечує легке обслуговування та гнучкість у внесенні змін без впливу на зовнішній інтерфейс.

Рівень даних відповідає за керування взаємодією системи з базою даних. Рівень даних безпосередньо взаємодіє з базою даних MS SQL Server, вирішуючи такі завдання, як зберігання та отримання даних, виконання SQL-запитів і підтримка узгодженості даних. Цей рівень має вирішальне значення для того, щоб дані про запаси та замовлення залишалися точними та актуальними. Він використовує такі методи, як операції CRUD (створення, читання, оновлення, видалення), щоб змінювати дані за потреби, і забезпечує надійну обробку транзакцій даних, зберігаючи цілісність у всій системі.

Якщо системі потрібно взаємодіяти із зовнішніми службами, такими як постачальники або транспортні платформи, рівень інтеграції обробляє цей зв’язок. Цей рівень дозволяє системі обмінюватися даними із зовнішніми системами через веб-сервіси (наприклад, REST або SOAP), забезпечуючи додаткову гнучкість для інтеграції, яка може знадобитися в майбутньому.

Рівень бази даних зберігає всі необхідні дані для роботи складу. Він містить інформацію про продукти, замовлення, рівень запасів, постачальників та інші відповідні бізнес-дані. Використовуючи MS SQL Server, база даних розроблена з добре організованими таблицями, зв’язками та обмеженнями для підтримки цілісності та узгодженості даних. Завдяки централізації цих даних система гарантує, що всі користувачі, будь то в різних відділах або на складах, можуть отримати доступ до однієї інформації в реальному часі [19].

Архітектура побудована на кількох шаблонах проєктування, які сприяють її модульності та масштабованості. Багаторівнева архітектура розділяє завдання на окремі рівні, що спрощує технічне обслуговування та майбутнє розширення. У системі також застосовано шаблон MVC (Model-View-Controller). Модель представляє дані, представлення представляє інтерфейс користувача, а контролер керує взаємодією між ними. Крім того, система може використовувати сервіс-орієнтовану архітектуру (SOA), яка інкапсулює основну бізнес-логіку в сервіси, які можна легко повторно використовувати або інтегрувати з іншими системами в майбутньому.

Нижче буде продемонстровано 4-шарова архітектура даного програмного забезпечення (рис. 10)

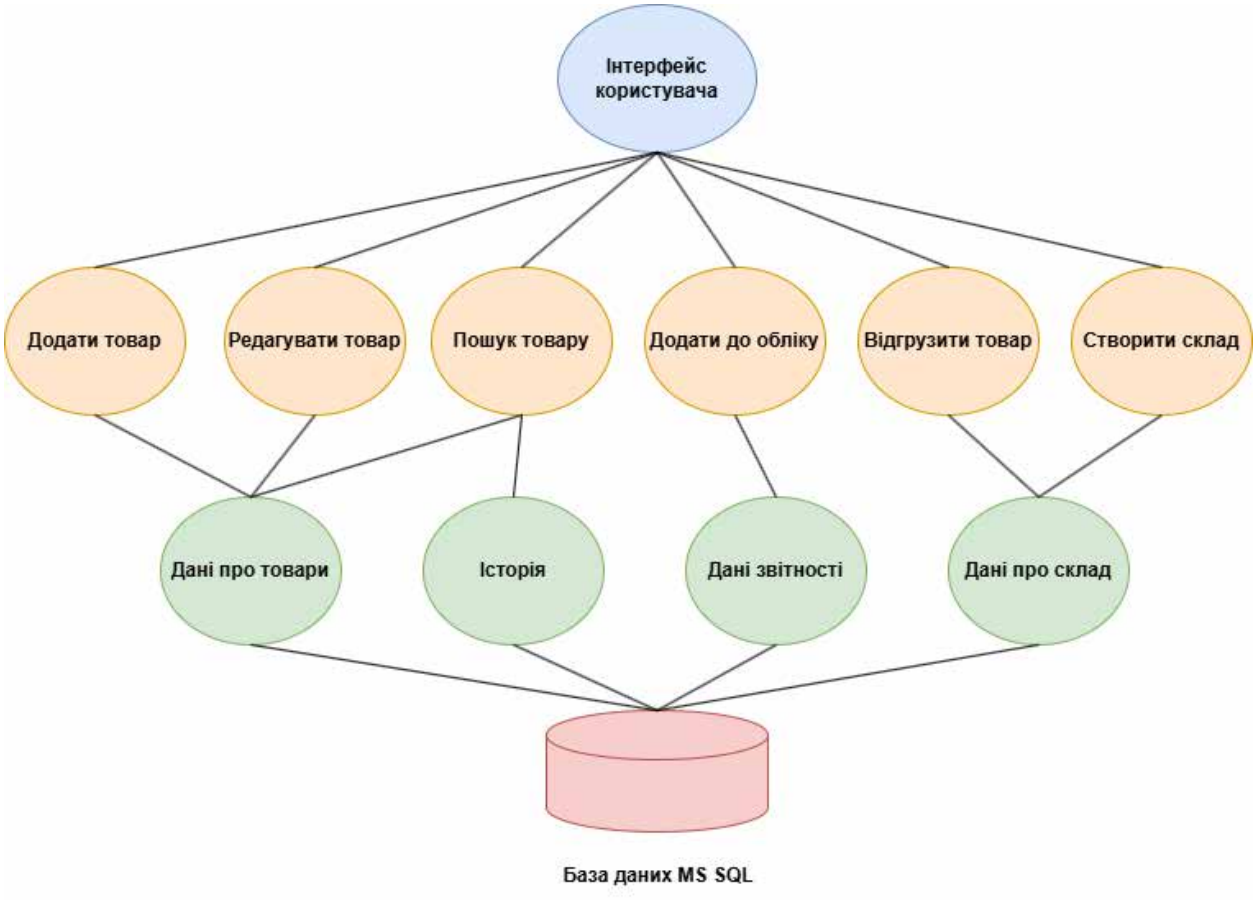


Рис. 10 Багатошарова архітектура додатку

З рисунка вище ми бачимо, що наша система побудована на 4-рівневій багатошаровій архітектурі, що складається з презентаційного рівня, бізнес-рівня, рівня збереження даних та рівня бази даних.

Інформація в системі поступово переходить від рівня до наступного рівня по черзі.

Перший рівень - це інтерфейс користувача, тобто екран комп'ютера або телефона.

Другий рівень - це основний функціонал програми, з яким працює користувач. На цьому рівні він вносить дані про товари, облік та склад.

На третьому рівні відбувається перенесення та збереження інформації від користувача до бази даних.

І четвертий рівень, безпосередньо, база даних, в який зберігається вся інформація складу.

3.4 Організаційна структура програмного забезпечення

3.4.1 Діаграма пакетів. Пакетна діаграма — це тип діаграми UML (Unified Modeling Language), яка організовує систему в окремі компоненти або пакети, щоб спростити її структуру та зробити її легшою для розуміння, особливо у великих і складних системах. Основна мета цієї діаграми — показати, як система розділена на різні функціональні області або модулі, і висвітлити зв’язки між ними. Візуалізуючи модульну організацію, діаграма пакета забезпечує чітке уявлення про те, як різні частини системи взаємодіють одна з одною [20].

У випадку інформаційної системи автоматизованого робочого місця для працівника складської інфраструктури діаграма пакета використовується для ілюстрації модульної структури системи, розбиваючи її на такі компоненти, як інтерфейс користувача, бізнес-логіка, керування базами даних та рівні інтеграції. Це допомагає отримати повне уявлення про те, як організовано програмне забезпечення та як різні компоненти працюють разом, щоб забезпечити загальну функціональність системи.

Зв’язки між цими пакетами показані через залежності, представлені стрілками на діаграмі. Наприклад, пакет бізнес-логіки може залежати від

пакета взаємодії з базою даних для отримання або оновлення даних. Ці залежності мають вирішальне значення для розуміння того, як різні частини системи впливають одна на одну [26].

На діаграмі пакетів також визначається видимість кожного пакета, яка визначає, наскільки він доступний для інших частин системи. Загальнодоступний пакет доступний іншим компонентам, тоді як приватний пакет є ізольованим, з обмеженою можливістю взаємодії з іншими пакетами.

Діаграма пакетів (рис. 11) для даного програмного забезпечення використовується для візуального представлення архітектури та організації системи. Діаграма пакетів зображує різні пакети, які складають програмне забезпечення, і ілюструє зв’язки та залежності між ними.

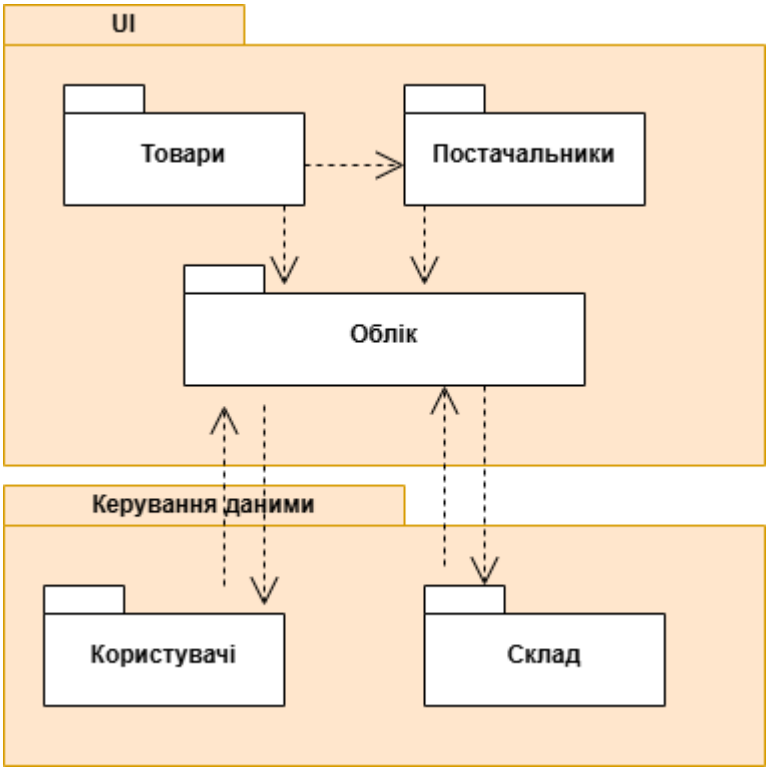


Рис. 11 Діаграма пакетів

3.5 Вибір інструментарію для створення програмного забезпечення

Для розробки Інформаційної системи автоматизованого робочого місця працівника складської інфраструктури був ретельно підібраний набір

інструментів, що забезпечує високу ефективність, масштабованість і ремонтопридатність. Ці інструменти вибираються на основі їх здатності підтримувати необхідну функціональність системи та відповідати конкретним вимогам середовища управління складом. Інструменти, вибрані для цього проєкту, включають C#, WinForms, MS SQL Server, DevExpress, EF Core та Git.

C# було обрано як основну мову програмування для цієї програми через її універсальність, простоту використання та підтримку принципів об’єктно-орієнтованого програмування. C# відомий своєю високою продуктивністю, надійністю та широким застосуванням у додатках корпоративного рівня, що робить його чудовим вибором для створення надійної та зручної програмної системи. Крім того, C# має великі бібліотеки та фреймворки, які можуть спростити процес розробки, дозволяючи швидше впроваджувати функції та скорочувати час, необхідний для налагодження та тестування [27].

WinForms було обрано як основу для розробки інтерфейсу користувача (UI). WinForms забезпечує ефективний спосіб створювати настільні програми для Windows із багатими графічним інтерфейсом. Він добре підходить для створення інтуїтивно зрозумілих інтерфейсів користувача для складських працівників, менеджерів та інших зацікавлених сторін. Завдяки компонентам інтерфейсу перетягування та скидання WinForms дозволяє швидко розробляти форми та елементи керування, що робить його кращим вибором для створення настільних програм, які вимагають високого рівня взаємодії з користувачем.

Для управління базами даних було обрано MS SQL Server як реляційну систему керування базами даних (RDBMS). SQL Server пропонує потужну та безпечну платформу для керування великими обсягами даних, гарантуючи, що система може обробляти зберігання та пошук пов’язаної зі складом інформації, такої як інвентарні записи, дані про замовлення та історії транзакцій. Потужна підтримка транзакцій, цілісності даних і

масштабованості SQL Server робить його ідеальним вибором для критично важливих програм, які вимагають надійного зберігання та керування даними.

Щоб полегшити взаємодію між додатком і базою даних, EF Core (Entity Framework Core) було обрано як структуру Object-Relational Mapping (ORM). EF Core спрощує роботу з базою даних, дозволяючи розробникам працювати з об’єктами .NET із суворим типом, а не писати необроблені запити SQL. Це спрощує доступ до даних, покращує підтримку коду та зменшує ризик помилок в операціях з базою даних. EF Core також надає такі потужні функції, як міграції, які допомагають підтримувати схему бази даних у міру розвитку системи з часом.

DevExpress використовується для вдосконалення графічного інтерфейсу та надання розширених елементів керування інтерфейсом користувача. DevExpress пропонує багатий набір готових елементів керування, таких як сітки даних, діаграми та звіти, які необхідні для керування та відображення великих наборів даних у системі керування складом. За допомогою DevExpress розробники можуть створювати адаптивні, візуально привабливі інтерфейси користувача, які покращують взаємодію з користувачем і роблять програму більш інтуїтивно зрозумілою та легкою для навігації.

Git використовується як система контролю версій для керування вихідним кодом і відстеження змін у процесі розробки. Git — це важливий інструмент для спільної розробки програмного забезпечення, що дозволяє кільком розробникам працювати над проєктом одночасно, не конфліктуючи зі змінами один одного. Використовуючи Git, команда розробників може зберігати історію змін коду, легко повертатися до попередніх версій і гарантувати, що програмне забезпечення постійно оновлюється та підтримується.

Таким чином, вибір інструментів для розробки інформаційної системи автоматизованого робочого місця працівника складської інфраструктури зосереджений на використанні надійних, ефективних і масштабованих

технологій. C# і WinForms забезпечують основу для програми, а MS SQL Server і EF Core керують даними та взаємодією. DevExpress покращує інтерфейс користувача, а Git забезпечує ефективний контроль версій і співпрацю протягом усього процесу розробки. Разом ці інструменти допоможуть створити потужну та надійну систему, адаптовану до потреб працівника складської інфраструктури.

3.6 Алгоритмізація та програмування програмних модулів

У процесі розробки інформаційної системи автоматизованого робочого місця працівника складської інфраструктури особлива увага приділялася алгоритмізації та програмній реалізації основних модулів системи. Застосований підхід базується на принципах об'єктно-орієнтованого програмування та багаторівневої архітектури, що забезпечує модульність, повторне використання коду та легкість подальшої підтримки системи.

Проект реалізовано мовою програмування C# з використанням технології Windows Forms для створення користувацького інтерфейсу. Для організації коду застосовано архітектурний патерн Model-View-Controller (MVC), який дозволяє ефективно відокремити бізнес-логіку від представлення даних. Така структура значно спрощує тестування окремих компонентів та полегшує внесення змін у систему.

База даних розроблена з використанням Microsoft SQL Server, а для взаємодії з нею застосовано технологію Entity Framework Core. Це дозволило абстрагуватися від безпосередньої роботи з SQL-запитами та зосередитися на бізнес-логіці додатку. Entity Framework Core забезпечує об'єктно-реляційне відображення (ORM), завдяки якому робота з даними відбувається на рівні

об'єктів, що значно прискорило процес розробки та знизило ймовірність помилок при роботі з базою даних.

Центральним елементом системи є модуль управління товарними запасами, алгоритми якого оптимізовані для швидкої обробки операцій з великими обсягами даних. Цей модуль відповідає за реєстрацію нових товарів, відстеження їх переміщення, контроль залишків та формування повідомлень про необхідність поповнення запасів. Фрагмент коду, що реалізує алгоритм пошуку оптимального місця для розміщення товару на складі, наведено у Додатку А.

Модуль обробки складських операцій реалізує логіку прийому, переміщення та відвантаження товарів. Для кожної операції розроблено відповідні алгоритми, які забезпечують контроль цілісності даних та автоматичне оновлення інформації про залишки. Особлива увага приділена обробці виняткових ситуацій, таких як невідповідність фактичної кількості товару зазначеній у документах або спроба відвантаження відсутнього товару. Програмний код обробки складських операцій із детальними коментарями представлено у Додатку Б.

Система звітності побудована на основі гнучких алгоритмів агрегації та аналізу даних. Для формування звітів використано технологію LINQ (Language Integrated Query), яка дозволяє ефективно виконувати складні запити до даних безпосередньо в коді C#. Це забезпечує високу продуктивність при генерації звітів різної складності — від простих списків товарів до комплексних аналітичних звітів з групуванням та фільтрацією даних за різними критеріями.

Для забезпечення безпеки даних розроблено модуль автентифікації та авторизації користувачів. У цьому модулі реалізовано алгоритми хешування паролів з використанням сучасних криптографічних методів, а також механізми контролю доступу на основі ролей. Кожен користувач системи має певний набір прав, які визначають доступні йому функції та дані. Реалізація

цього модуля ґрунтується на принципах захисту від найпоширеніших типів атак, таких як SQL-ін'єкції, міжсайтовий скриптинг та підбір паролів.

У процесі програмування було застосовано техніку асинхронного виконання операцій для забезпечення відзивчивості інтерфейсу навіть при виконанні ресурсомістких задач. Це особливо важливо при роботі з великими обсягами даних або при взаємодії з мережевими ресурсами. Асинхронні методи в C# реалізовані з використанням ключових слів async та await, що значно спрощує написання та підтримку коду.

Для покращення продуктивності системи було розроблено алгоритми кешування даних, які зменшують кількість звернень до бази даних шляхом збереження часто використовуваної інформації в оперативній пам'яті. Реалізація цих алгоритмів базується на патерні проектування "Одинак" (Singleton), який забезпечує єдину точку доступу до кешованих даних та гарантує їх консистентність.

Особливістю розробленої системи є можливість роботи в автономному режимі при тимчасовій відсутності з'єднання з сервером бази даних. Для цього реалізовано алгоритми синхронізації даних, які відстежують зміни, внесені в автономному режимі, та коректно застосовують їх до центральної бази даних після відновлення з'єднання, вирішуючи можливі конфлікти даних.

У процесі розробки активно використовувалися сучасні інструменти для відлагодження та профілювання коду, такі як Visual Studio Debugger та Performance Profiler. Це дозволило виявити та усунути потенційні проблеми продуктивності ще на етапі розробки, забезпечивши стабільну та швидку роботу системи навіть при значних навантаженнях.

Інтеграційне тестування програмних модулів проводилося з використанням фреймворку NUnit, що дозволило автоматизувати перевірку коректності роботи алгоритмів у різних сценаріях використання. Це значно

підвищило надійність системи та мінімізувало ризик появи помилок при її експлуатації.

Додаток А містить програмний код основних класів системи, включаючи реалізацію алгоритмів оптимізації розміщення товарів та управління запасами. У Додатку Б представлено код, що відповідає за взаємодію з базою даних та реалізацію бізнес-логіки обробки складських операцій. Обидва додатки супроводжуються детальними коментарями, які пояснюють принципи роботи алгоритмів та особливості їх реалізації.

Таким чином, алгоритмізація та програмування модулів інформаційної системи автоматизованого робочого місця працівника складської інфраструктури виконані з урахуванням сучасних підходів до розробки програмного забезпечення, що забезпечує надійність, ефективність та масштабованість системи.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Вимоги до апаратного та програмного забезпечення

Для забезпечення безперебійної роботи та оптимальної роботи інформаційної системи автоматизованого робочого місця працівника складської інфраструктури необхідні спеціальні апаратні та програмні компоненти. Ці вимоги ретельно підібрані для обробки обсягу даних і забезпечення ефективної, надійної та масштабованої функціональності системи в складському середовищі.

Для налаштування апаратного забезпечення сервер має бути оснащений багатоядерним процесором, таким як Intel Xeon або AMD Ryzen, щоб ефективно виконувати численні завдання, включаючи обробку даних, створення звітів і оновлення в реальному часі. Потрібно щонайменше 16 ГБ оперативної пам’яті, хоча 32 ГБ або більше рекомендується для великих операцій або великих обсягів транзакцій, оскільки це покращить здатність системи керувати складними запитами та великими наборами даних. Крім того, сервер повинен мати принаймні 500 ГБ пам’яті SSD, щоб полегшити швидкий пошук і запис даних. Щоб забезпечити високу доступність, важливо реалізувати додаткові механізми зберігання та резервного копіювання, захищаючи базу даних і дані програми. Швидкий і надійний мережевий інтерфейс, в ідеалі 1 Гбіт/с або вище, буде необхідний для підтримки безперебійного зв’язку між сервером і клієнтськими робочими станціями, особливо у великих середовищах з кількома користувачами.

На стороні клієнта робочі станції повинні бути оснащені сучасними багатоядерними процесорами, такими як Intel Core i5 або i7, щоб забезпечити плавну роботу під час взаємодії з системою. Для кожної робочої станції рекомендовано принаймні 4 ГБ оперативної пам’яті, хоча 8 ГБ забезпечать

кращу багатозадачність і більшу швидкість реагування. Для зберігання файлів програм, тимчасових даних і локальних резервних копій має бути мінімум 250 ГБ жорсткого диска або SSD. Монітор із роздільною здатністю не менше 1280x1024 пікселів забезпечить оптимальний інтерфейс користувача. Периферійні пристрої, як-от миша та клавіатура, необхідні для взаємодії із системою, і залежно від складських операцій сканери штрих-кодів і принтери також можуть знадобитися для відстеження запасів і друку звітів. Робочі станції з'єднуватимуться через стабільну та безпечну локальну мережу (LAN), яка має забезпечувати достатню пропускну здатність для підтримки одночасного доступу користувачів. У деяких випадках Wi-Fi може бути життєздатним варіантом для складів, де працівники часто пересуваються та потребують гнучкості у використанні пристроїв.

Вимоги до програмного забезпечення системи мають вирішальне значення для забезпечення сумісності, продуктивності та безпеки. Операційна система на сервері має бути Windows Server 2019 або новішої версії, або серверної ОС на базі Linux залежно від уподобань і технічного досвіду. На стороні клієнта рекомендується Windows 10 або новіша версія для забезпечення сумісності з програмою WinForms. Для керування базою даних використовуватиметься Microsoft SQL Server для зберігання та отримання даних. MS SQL Server обрано завдяки його високій продуктивності, масштабованості та функціям безпеки, що робить його ідеальним для обробки складних складських операцій.

З точки зору розробки, Microsoft Visual Studio буде інтегрованим середовищем розробки (IDE), яке використовується для проєктування системи. Visual Studio добре сумісна з C#, WinForms і MS SQL Server, що робить її найкращим вибором для розробки зручної та надійної програми. Entity Framework Core (EF Core) слугуватиме фреймворком об'єктно-реляційного відображення (ORM), який спростить взаємодію з базою даних шляхом абстрагування необроблених запитів SQL. Цей фреймворк дозволяє розробникам працювати з об'єктами та запитами LINQ замість

безпосереднього написання коду SQL, що спрощує процес розробки. Крім того, DevExpress буде використовуватися для покращення інтерфейсу користувача, надаючи розширені елементи керування та інструменти візуалізації даних, які покращать загальний досвід роботи складських працівників і менеджерів.

Щоб забезпечити плавну співпрацю між розробниками, Git буде використовуватися для контролю версій. Git дозволяє кільком розробникам одночасно працювати над кодовою базою, відстежує всі зміни та зберігає історію оновлень коду, допомагаючи організовувати процес розробки. Крім того, надійне рішення для резервного копіювання та відновлення буде необхідним для захисту як бази даних, так і даних програми. Ця система повинна включати автоматичне щоденне резервне копіювання та зовнішні (хмарні) механізми резервного копіювання для забезпечення безпеки даних у разі будь-яких системних збоїв.

З міркувань безпеки на всіх серверах і робочих станціях має бути встановлено антивірусне програмне забезпечення та програмне забезпечення для захисту від зловмисних програм, а також належним чином налаштований брандмауер, щоб запобігти несанкціонованому доступу до системи. Шифрування буде критично важливим для захисту конфіденційних даних, таких як записи транзакцій або особисті дані співробітників, забезпечення конфіденційності.

Підсумовуючи, вибрані апаратні та програмні компоненти призначені для забезпечення ефективного, безпечного та масштабованого середовища для інформаційної системи автоматизованого робочого місця працівника складської інфраструктури.

4.2 Тестування системи

При запуску програми перед користувачем відкривається вікно авторизації (Рис. 12).

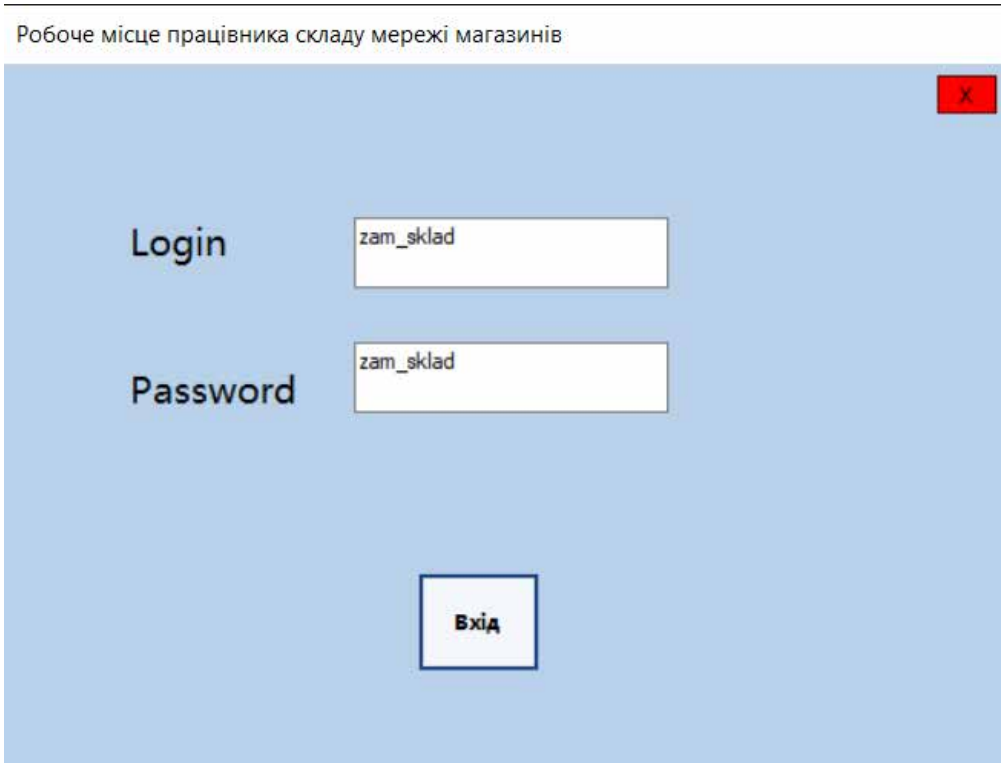


Рис. 12 Вікно авторизації

Після авторизації відкривається форма в залежності до якої ролі відноситься користувач zam_sklad чи worker. Форма для zam_sklad дає повний доступ.

Форма для worker дає доступ до перегляду інформації і додавання поставок». Перший пункт меню надає зам складу редагувати товар (Рис. 13).

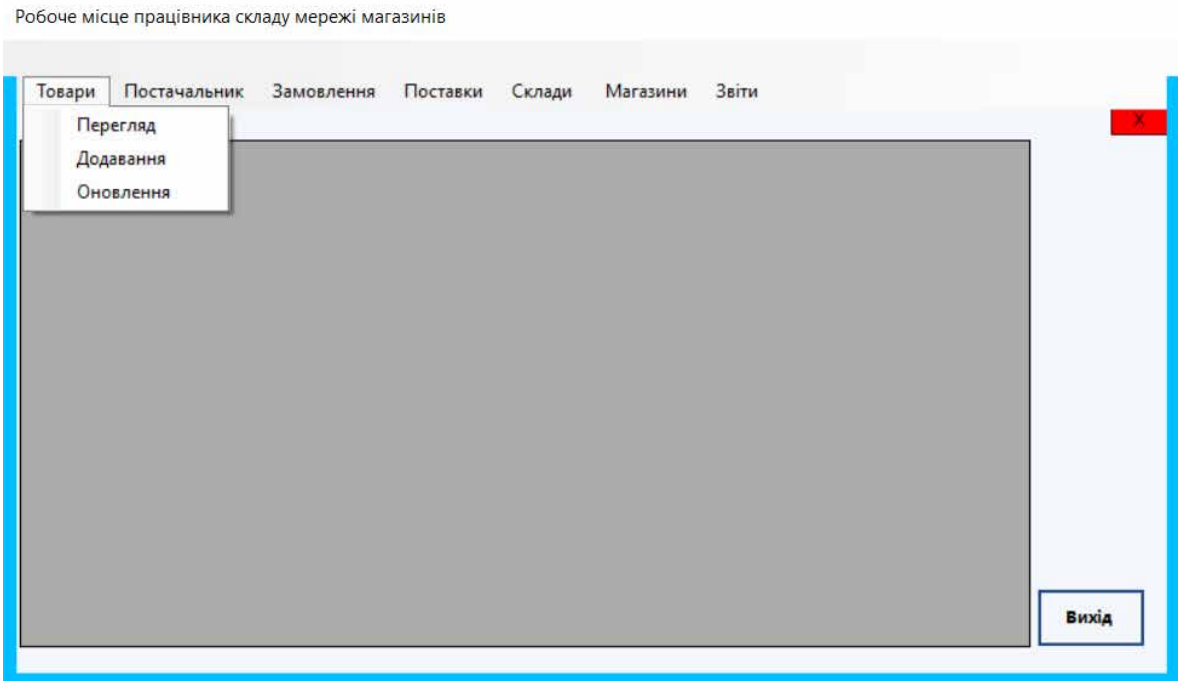


Рис. 13 Випадаюче вікно “Товари”

Натиснувши кнопку "Додавання" ми перейдемо на сторінку створення товару, де нам потрібно заповнити всі необхідні поля та успішно підтвердити свою дію (Рис. 14).

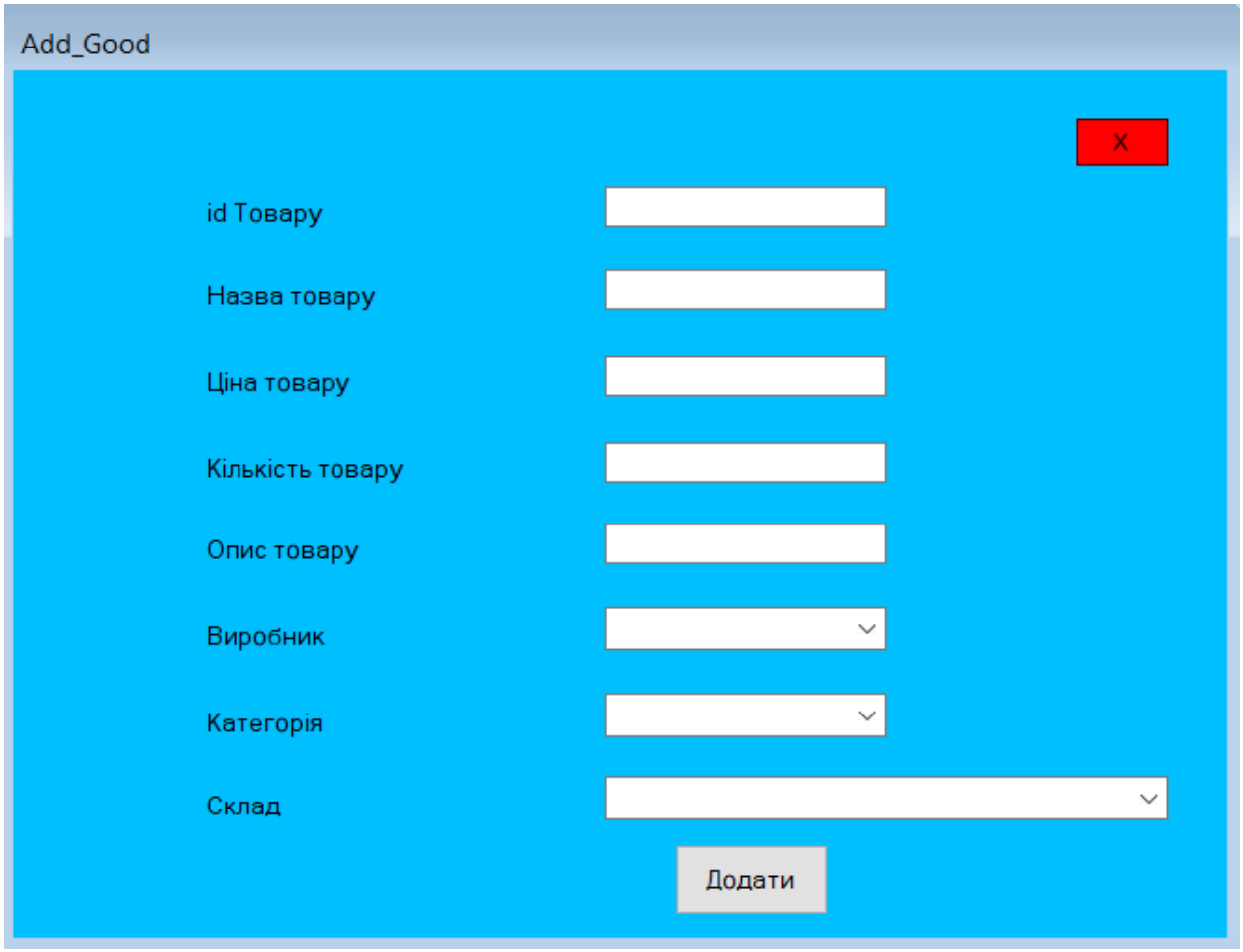


Рис. 14 Сторінка створення товару

Тепер ми можемо перейти на сторінку "Редагувати". Тут ми можемо змінити значення всіх полів щодо обраного нами товару (Рис. 15)

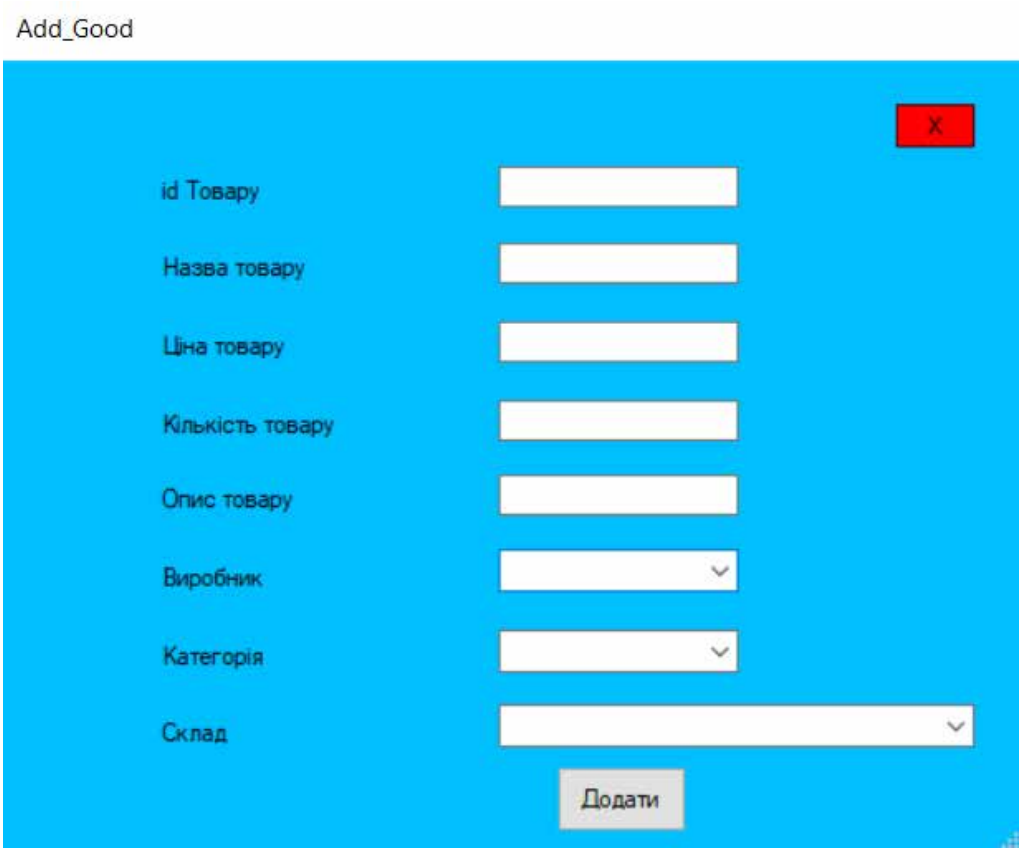
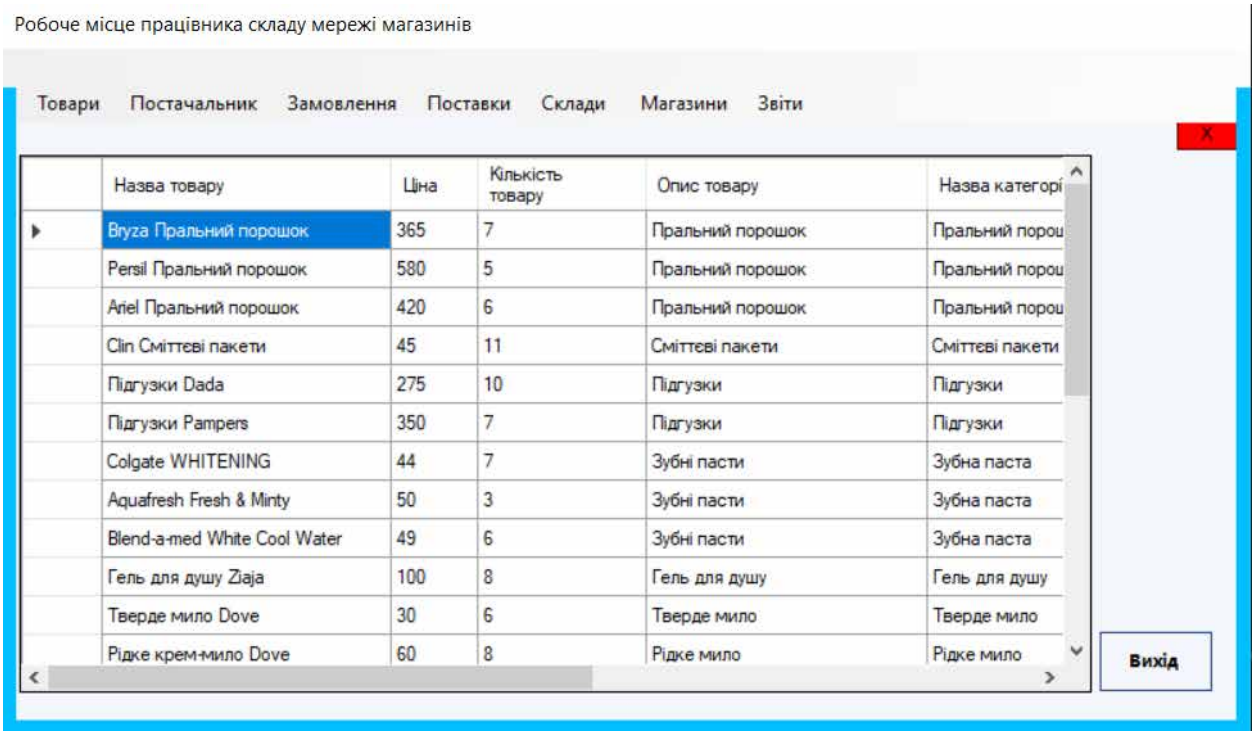


Рис. 15 Сторінка редагування товару

Переходимо на сторінку "Огляд" (Рис. 16). Тут ми бачимо список створених товарів, з якими ми можемо працювати.



	Назва товару	Ціна	Кількість товару	Опис товару	Назва категорі
▶	Bryza Пральний порошок	365	7	Пральний порошок	Пральний порош
	Persil Пральний порошок	580	5	Пральний порошок	Пральний порош
	Ariel Пральний порошок	420	6	Пральний порошок	Пральний порош
	Clin Сміттєві пакети	45	11	Сміттєві пакети	Сміттєві пакети
	Підгузки Dada	275	10	Підгузки	Підгузки
	Підгузки Pampers	350	7	Підгузки	Підгузки
	Colgate WHITENING	44	7	Зубні пасти	Зубна паста
	Aquafresh Fresh & Minty	50	3	Зубні пасти	Зубна паста
	Blend-a-med White Cool Water	49	6	Зубні пасти	Зубна паста
	Гель для душу Ziaja	100	8	Гель для душу	Гель для душу
	Тверде мило Dove	30	6	Тверде мило	Тверде мило
	Рідке крем-мило Dove	60	8	Рідке мило	Рідке мило

Рис. 16 Сторінка огляду товарів

Для перегляду звітів, розгорнутих на сервері звітів, і звітів, які існують в локальній файловій системі, було використано представлення «ChoosenSupplies», інформація виводиться через ReportView для підготовки звітів до перегляду в додатку Windows. Для даного проєкту було створено 2 звіти: «Список усіх відправок у вибрану дату», «Список усіх поставок у вибрану дату» (Рис. 17).

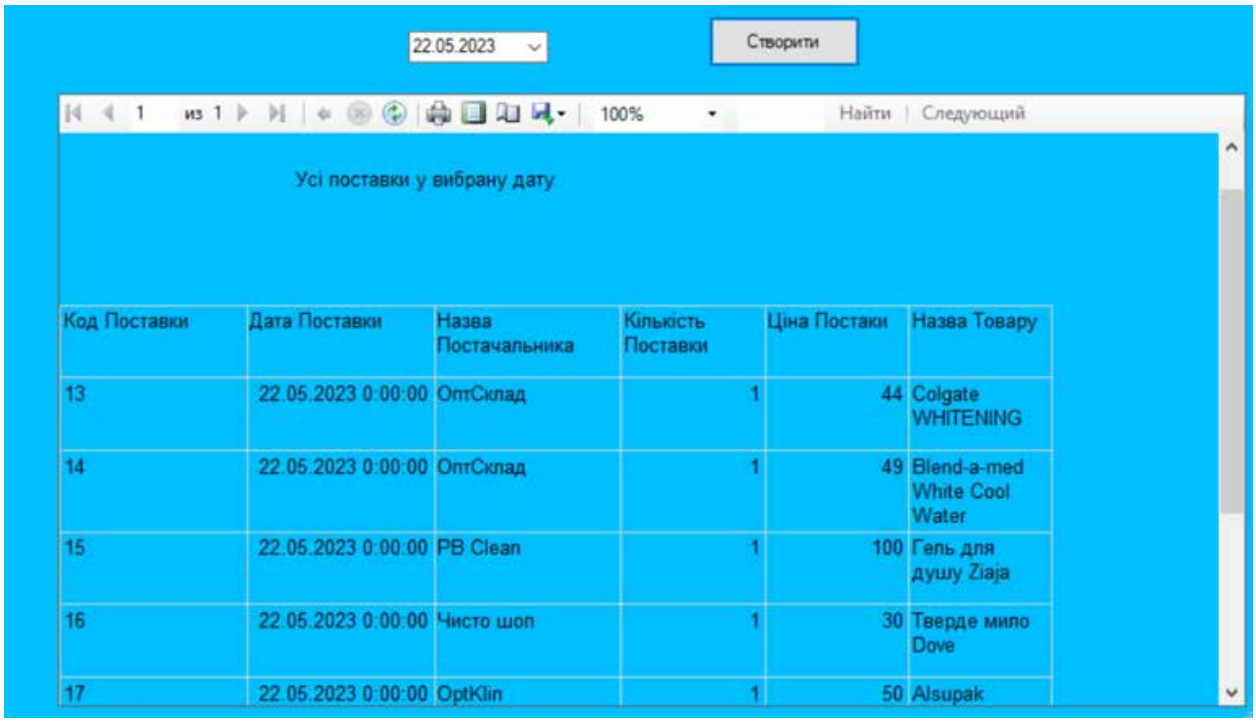


Рис. 17 Список усіх поставок у вибрану дату

4.3 Склад інсталяційного пакету

Інсталяційний пакет розробленої інформаційної системи для автоматизованого складу робочого місця працівника представляє собою комплексне рішення, створене за допомогою інструментарію Visual Studio. Цей пакет містить усі необхідні компоненти для забезпечення повноцінного функціонування системи на клієнтських комп’ютерах у складському середовищі.

У структурі пакету присутній основний виконуваний файл із розширенням .exe, який служить точкою входу в систему та ініціює її роботу.

Разом з ним постачаються бібліотеки динамічного компонування (.dll), що відповідають різним аспектам функціональності: взаємодію з базою даних через Entity Framework Core, відображення інтерфейсу користувача за допомогою компонентів DevExpress, роботу з файловою системою та формування звітності. В особливій ролі кожного конфігураційного файлу з налаштуваннями програм, де зберігаються параметри підключення до бази даних, подій входу, продуктивності та безпеки.

Важливою складовою пакетом є набір SQL-скриптів для налаштування бази даних. Ці скрипти автоматизують процес створення таблиці, встановлення індексів та зв’язків між ними, а також доповнення системи початковими даними, такими як довідники та облікові записи користувачів. Довідкова система представлена детальним керівництвом у форматі PDF, вбудованим за допомогою та інструкціями з усунення типових несправностей.

Центральним елементом пакета виступає інсталятор з розширенням .msi, який забезпечує зручне розгортання системи. Додатково включені ресурсні файли: іконки, зображення, шаблони документів та звітів, необхідні для візуального оформлення та функціонування програм.

Процес інсталяції реалізовано з використанням технологій ClickOnce та Windows Installer. На початковому етапі відбувається автоматична перевірка відповідності комп’ютера мінімальним системним вимогам – наявність потрібної версії .NET Framework, доступність вільного місця на диску та адміністративних прав для встановлення. При виявленні відсутніх компонентів система самостійно завантажує та встановлює повний вплив: фреймворк .NET, локальну версію SQL Server Express або пакети Visual C++ Redistributable.

Користувач має можливість вибору параметрів інсталяції, таких як цільова директорія, тип підключення до бази даних (локальна чи серверна) та набір компонентів для встановлення. Під час налаштування бази даних

пропонуються варіанти створення нової бази, підключення до наявної або імпорту даних з попередніх версій системи.

Після завершення інсталяції на комп'ютері формується структурований файл ієрархії. У кореневій директорії розміщуються основний виконаний файл та конфігурація системи. Окремі папки відведені для функціональних модулів (інвентаризація, звітність, управління користувачами), зовнішніх залежностей, ресурсів програм та довідкової документації. Додатково встановлюються допоміжні інструменти для управління базою даних та резервного копіювання інформації.

Система оновлення забезпечує актуальність програмного забезпечення через механізм диференційних оновлень, завантажуючи тільки змінені компоненти. Перед встановленням нової версії автоматично створюються резервні копії критичних даних, а в разі виникнення проблеми передбачена можливість відкату до попередньої стабільної версії.

Безпека інсталяційного пакету гарантується через цифрове підписання всіх виконуваних файлів та бібліотек, що дозволяє підтвердити їх автентичність. Конфіденційна інформація, включаючи паролі та ліцензійні ключі, зберігається в зашифрованому вигляді. Вбудовані механізми захисту від несанкціонованого копіювання запобігають незаконному використанню системи, а журнал безпеки фіксує всі важливі дії для подальшого аудиту.

Таким чином, інсталяційний пакет розробленої інформаційної системи для працівників відповідно представляє собою використання продуманого комплексу компонентів, що забезпечує просте розгортання, надійне функціонування та зручне оновлення програмного забезпечення в різноманітних середовищах – від невеликих складських приміщень до масштабних логістичних центрів з розподіленою інфраструктурою.

ВИСНОВКИ

Підсумовуючи, інформаційна система автоматизованого робочого місця працівника складської інфраструктури спрямована на суттєве підвищення ефективності та продуктивності роботи складу. Завдяки інтеграції сучасних технологій, таких як C#, WinForms, MS SQL Server і DevExpress, система пропонує надійну та зручну платформу для управління запасами, моніторингу робочих процесів і виконання різноманітних складських завдань. Завдяки автоматизації рутинних процесів, це зменшує кількість людських помилок, підвищує точність даних і дозволяє швидше приймати рішення.

Система розроблена для оптимізації продуктивності працівників складської інфраструктури шляхом оптимізації ключових завдань, таких як управління запасами, обробка замовлень і відстеження руху товарів. Використовуючи комплексну модель бази даних, система забезпечує ефективне зберігання, доступ і оновлення всіх відповідних даних у режимі реального часу. Впровадження сучасних інструментів розробки, таких як EF Core та Git, забезпечує безперебійну співпрацю та спрощує розробку та підтримку системи.

Вимоги до апаратного та програмного забезпечення були ретельно підібрані, щоб гарантувати масштабованість, безпеку та продуктивність системи. Завдяки надійній архітектурі система може обробляти зростаючі вимоги та великі обсяги даних, зберігаючи безпеку та цілісність даних. Крім того, використання рішень для резервного копіювання та заходів безпеки гарантує, що критичні дані сховища залишаються в безпеці та захищені від потенційних загроз.

Система має значний потенціал для майбутніх вдосконалень. З розвитком технологій систему можна інтегрувати з додатковими інструментами автоматизації, такими як сканування штрих-кодів або

відстеження RFID, для подальшого вдосконалення управління запасами та зменшення людського втручання. Крім того, масштабованість системи дозволяє розвиватися в майбутньому, дозволяючи розміщувати більші склади або розширюватися на нові бізнес-сфери. Залишаючись гнучкою та зручною, система має хороші можливості для задоволення мінливих потреб працівників складської інфраструктури, забезпечуючи довгостроковий успіх і постійне вдосконалення складських операцій.

Ця система не тільки підвищить ефективність роботи складських працівників, але й надасть менеджерам цінну інформацію завдяки точному аналізу даних і звітності. Автоматизуючи ключові завдання та покращуючи потік даних, він прокладає шлях до більш раціоналізованого, ефективного та продуктивного середовища зберігання. У результаті система сприяє досягненню ширших цілей зниження витрат, оптимізації складського простору та покращення загальної продуктивності, що робить її цінним інструментом для сучасного управління складом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Oracle Warehouse Management. URL: https://www.oracle.com/middleeast/scm/logistics/warehouse-management/what-is-warehouse-management/
2. QuickBooks Commerce. URL: https://quickbooks.intuit.com/global/
3. Голуб Б.Л. Методичний посібник до вивчення дисципліни «Програмування та алгоритмічні мови». Навчальне видання Голуб Б.Л., Щукайло Є.М.-К-, Видавничий центр НАУ. 2003. – 64 с
4. Пасічник В. В. Організація баз даних і знань. / Пасічник В. В., Резніченко В. А. – К.: BHV, 2006. – 384 с.
5. Microsoft Docs: SQL Server. URL: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver17
6. Microsoft SQL Server. URL: https://www.microsoft.com/uk-ua/sql-server
7. Діаграма варіантів використання (UseCase diagram). URL: https://flexberry.github.io/en/fd_use-case-diagram.html#content
8. Застосування UML (частина 2). Діаграма послідовності – sequence diagram. URL: https://duikt.edu.ua/ua/news-1-626-7897-zastosuvannya-uml-chastina-2-diagrama-poslidovnosti---sequence-diagram_kafedra-kompyuternih-nauk-ta-informaciynih-tehnologiy
9. Діаграма активностей (видів діяльності). URL: https://flexberry.github.io/en/fd_activity-diagram.html#content
10. UML-діаграми класів. URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/
11. What is an Entity Relationship Diagram (ERD)?. URL: https://www.lucidchart.com/pages/er-diagrams
12. How to Connect to a MySQL Database. URL: https://www.domain.com/help/article/how-to-connect-to-a-mysql-database
13. Що таке MySQL? URL: https://hyperhost.ua/ua/wiki/scho-take-mysql

14. Багаторівнева архітектура. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/n-tier-architecture?srsltid=AfmBOoq01yO1tg6udE81b1779hg8ISaYtD6TSAJFLlqrj4GQ_S7A1sVD
15. CA Technologies. Erwin Data Modeler. URL: https://www.erwin.com/products/erwin-data-modeler/
16. Rumbaugh J., Jacobson I., Booch G. Unified Modeling Language Reference Manual. Pearson Education, Limited, 2004. 752 p.
17. Microsoft. Model-View-ViewModel (MVVM) architectural pattern. URL: https://docs.microsoft.com/en-us/windows/uwp/data-binding/data-binding-and-mvvm
18. Common web application architectures. URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures
19. What is Component Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/
20. Rumbaugh J., Jacobson I., Booch G. Unified Modeling Language Reference Manual. Pearson Education, Limited, 2004. 752 стор.
21. Прессман, Р. С. Розробка програмного забезпечення: підхід спеціаліста. McGraw-Hill Education, 2014. 960 стор.
22. Соммервіль, І. Розробка програмного забезпечення. 10-е вид. Pearson Education, 2015. 928 стор.
23. Хосров-Пур, М. Енциклопедія інформаційних наук і технологій. 3-е вид. IGI Global, 2014. 4500 стор.
24. Снід, Р. Наука про склад і розподіл. 2-е вид. Університет Арканзасу, 2014. 198 стор.
25. Chen, P. P. Модель сутності-зв’язку: на шляху до єдиного перегляду даних. ACM Transactions on Database Systems, 1976, Vol. 1, № 1, С. 9-36.

26. Фаулер, М. Шаблони архітектури корпоративних додатків. Addison-Wesley, 2002. 338 стор.
27. Ambler, S. W. Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. John Wiley & Sons, 2002. 350 стор.

ДОДАТОК А

Створення бази даних. Створення таблиць та їх зв’язків

```
use shopChainDB

IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'checkLine' AND type_desc = 'USER_TABLE')
drop table checkLine
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'check_' AND type_desc = 'USER_TABLE')
drop table check_
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'store' AND type_desc = 'USER_TABLE')
drop table store
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'invoiceGoods' AND type_desc = 'USER_TABLE')
drop table invoiceGoods
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'goods' AND type_desc = 'USER_TABLE')
drop table goods
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'invoiceLine' AND type_desc = 'USER_TABLE')
drop table invoiceLine
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'invoice' AND type_desc = 'USER_TABLE')
drop table invoice
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'storage' AND type_desc = 'USER_TABLE')
drop table storage
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'category' AND type_desc = 'USER_TABLE')
drop table category
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'producer' AND type_desc = 'USER_TABLE')
drop table producer
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'provider_' AND type_desc = 'USER_TABLE')
drop table provider_

go
create table provider_(
idprovider char(5) PRIMARY KEY,
nameprovider varchar (150)
)
go
create table producer(
idproducer char(5) PRIMARY KEY,
nameproducer varchar(150)
)
go
create table category(
idcategory char(5) PRIMARY KEY,
namecategory varchar(150)
)
go
create table storage(
idstorage char(5) PRIMARY KEY,
adress varchar(max),
squarestorage float
)
go
create table invoice(
```

```
idinvoice char(5) PRIMARY KEY,
deliverydate date
)
go
create table invoiceLine(
idinvoiceLine char(5) PRIMARY KEY,
idinvoice char(5),
idprovider char(5),
FOREIGN KEY (idinvoice) REFERENCES invoice (idinvoice),
FOREIGN KEY(idprovider) REFERENCES provider_ (idprovider),
amountDelivery tinyint,
priceDelivery tinyint
)
go
create table goods(
idgoods char(5) PRIMARY KEY,
nameGoods varchar(150),
price int,
amountgoods tinyint,
descriptionGood varchar(max),
idproducer char(5),
idcategory char(5),
idstorage char(5),
FOREIGN KEY (idproducer) REFERENCES producer (idproducer),
FOREIGN KEY (idcategory) REFERENCES category (idcategory),
FOREIGN KEY (idstorage) REFERENCES storage (idstorage),
)
go
create table invoiceGoods(
idInvoiceGoods char(5) PRIMARY KEY,
idinvoice char(5),
idgoods char(5),
FOREIGN KEY (idinvoice) REFERENCES invoice (idinvoice),
FOREIGN KEY (idgoods) REFERENCES goods (idgoods),
)

go
create table store(
idstore char(5) PRIMARY KEY,
adress varchar(max),
nameStore varchar(150)
)
go
create table check_(
idorder char(5) PRIMARY KEY,
statusOrder varchar(15),
dateOrder date
)
go
create table checkGood(
idcheckGood char(5) PRIMARY KEY,
idgoods char(5),
idorder char(5),
FOREIGN KEY (idgoods) REFERENCES goods (idgoods),
FOREIGN KEY (idorder) REFERENCES check_ (idorder)
)
go
create table checkLine(
idcheckLine char(5) PRIMARY KEY,
quantitySold tinyint,
idorder char(5),
idstore char(5),
FOREIGN KEY (idstore) REFERENCES store (idstore),
FOREIGN KEY (idorder) REFERENCES check_ (idorder)
)
```

ДОДАТОК Б

Фрагменти програмного коду. Методи заповнення DataGridView даними з бази даних з використанням Select

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace lab_7_obd
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            label1.Text = "Звіти";
        }

        private void button3_Click(object sender, EventArgs e)
        {
            label1.Text = "Перегляд";
        }

        private void button4_Click(object sender, EventArgs e)
        {
            label1.Text = "Управління";
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Form form1 = Application.OpenForms.OfType<Form>().FirstOrDefault();
            this.Close();
            form1.Show();
        }

        private void Form2_Load(object sender, EventArgs e)
        {
            this.FormBorderStyle = FormBorderStyle.FixedSingle;
        }

        private void button5_Click(object sender, EventArgs e)
        {
            Form form1 = Application.OpenForms.OfType<Form>().FirstOrDefault();
            form1.Close();
        }

        private void товариToolStripMenuItem_Click(object sender, EventArgs e)
        {
```

```
        }

        private void переглядToolStripMenuItem_Click(object sender, EventArgs e)
        {

        }

        private void переглядToolStripMenuItem1_Click(object sender, EventArgs e)
        {
            dataGridView1.Show();

            DBConnection dB = new DBConnection();
            dB.Connect("", "");
            dB.openConnection();

            string query = "SELECT goods.nameGoods as [Назва товару],price as
[Ціна],amountgoods as [Кількість товару], descriptionGood as [Опис
товару],namecategory as [Назва категорії], nameproducer as [Назва виробника], adress
as [Адреса складу]\r\nFROM goods, producer, storage, category\r\nWHERE
producer.idproducer = goods.idproducer and storage.idstorage = goods.idstorage and
category.idcategory = goods.idcategory";
            SqlCommand command = new SqlCommand(query, dB.GetConnection());

            DataTable table = new DataTable();
            SqlDataAdapter adapter = new SqlDataAdapter(command);
            adapter.Fill(table);
            dataGridView1.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.AllCells;
            dataGridView1.DataSource = table;
            dB.closeConnection();

        }

        private void категоріїToolStripMenuItem_Click(object sender, EventArgs e)
        {

        }

        private void переглядToolStripMenuItem2_Click(object sender, EventArgs e)
        {
            dataGridView1.Show();

            DBConnection dB = new DBConnection();
            dB.Connect("", "");
            dB.openConnection();

            string query = "SELECT * FROM category";
            SqlCommand command = new SqlCommand(query, dB.GetConnection());

            DataTable table = new DataTable();
            SqlDataAdapter adapter = new SqlDataAdapter(command);
            adapter.Fill(table);
            dataGridView1.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.AllCells;
            dataGridView1.DataSource = table;
            dB.closeConnection();
        }

        private void переглядToolStripMenuItem3_Click(object sender, EventArgs e)
        {
            dataGridView1.Show();

            DBConnection dB = new DBConnection();
            dB.Connect("", "");
            dB.openConnection();
```

```
            string query = "SELECT provider_.nameprovider as[Назва
постачальника]\r\nFROM provider_";
            SqlCommand command = new SqlCommand(query, dB.GetConnection());

            DataTable table = new DataTable();
            SqlDataAdapter adapter = new SqlDataAdapter(command);
            adapter.Fill(table);
            dataGridView1.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.AllCells;
            dataGridView1.DataSource = table;
            dB.closeConnection();
        }

        private void переглядToolStripMenuItem4_Click(object sender, EventArgs e)
        {
            dataGridView1.Show();

            DBConnection dB = new DBConnection();
            dB.Connect("", "");
            dB.openConnection();

            string query = "SELECT * FROM producer";
            SqlCommand command = new SqlCommand(query, dB.GetConnection());

            DataTable table = new DataTable();
            SqlDataAdapter adapter = new SqlDataAdapter(command);
            adapter.Fill(table);
            dataGridView1.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.AllCells;
            dataGridView1.DataSource = table;
            dB.closeConnection();
        }

        private void переглядToolStripMenuItem7_Click(object sender, EventArgs e)
        {
            dataGridView1.Show();

            DBConnection dB = new DBConnection();
            dB.Connect("", "");
            dB.openConnection();

            string query = "SELECT storage.adress as[Адреса складу],
storage.squarestorage as[Площа складу]\r\nFROM storage";
            SqlCommand command = new SqlCommand(query, dB.GetConnection());

            DataTable table = new DataTable();
            SqlDataAdapter adapter = new SqlDataAdapter(command);
            adapter.Fill(table);
            dataGridView1.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.AllCells;
            dataGridView1.DataSource = table;
            dB.closeConnection();
        }

        private void переглядToolStripMenuItem8_Click(object sender, EventArgs e)
        {
            dataGridView1.Show();

            DBConnection dB = new DBConnection();
            dB.Connect("", "");
            dB.openConnection();

            string query = "SELECT store.adress as[Адреса магазину], store.nameStore
as[Назва магазину]\r\nFROM store";
```

```
            SqlCommand command = new SqlCommand(query, dB.GetConnection());

            DataTable table = new DataTable();
            SqlDataAdapter adapter = new SqlDataAdapter(command);
            adapter.Fill(table);
            dataGridView1.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.AllCells;
            dataGridView1.DataSource = table;
            dB.closeConnection();
        }

        private void переглядToolStripMenuItem5_Click(object sender, EventArgs e)
        {
            DBConnection dB = new DBConnection();
            dB.Connect("", "");
            dB.openConnection();

            string query = "SELECT invoice.idinvoice as[Код поставки],deliverydate
as [Дата поставки],nameprovider as [Поставщик],amountDelivery as [Кількість
поставки], priceDelivery as [Ціна постаки],nameGoods as [Назва товару]\r\nFROM
provider_, invoiceLine, invoice, invoiceGoods,goods\r\nWHERE invoiceLine.idinvoice =
invoice.idinvoice and invoiceLine.idprovider = provider_.idprovider and
invoiceGoods.idgoods = goods.idgoods and invoiceGoods.idinvoice =
invoice.idinvoice";
            SqlCommand command = new SqlCommand(query, dB.GetConnection());

            DataTable table = new DataTable();
            SqlDataAdapter adapter = new SqlDataAdapter(command);
            adapter.Fill(table);
            dataGridView1.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.AllCells;
            dataGridView1.DataSource = table;
            dB.closeConnection();
        }

        private void додаванняToolStripMenuItem1_Click(object sender, EventArgs e)
        {
            Order Order = new Order();
            Order.ShowDialog();
        }

        private void додаванняToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Add_Good addGood = new Add_Good();
            addGood.ShowDialog();
        }

        private void додатиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Add_storage Add_storage = new Add_storage();
            Add_storage.ShowDialog();
        }

        private void магазиниToolStripMenuItem_Click(object sender, EventArgs e)
        {

        }

        private void додаванняToolStripMenuItem3_Click(object sender, EventArgs e)
        {
            Add_store Add_store = new Add_store();
            Add_store.ShowDialog();
        }

        private void оновленняToolStripMenuItem_Click(object sender, EventArgs e)
```

```
        {
            update_Good update_Good = new update_Good();
            update_Good.ShowDialog();
        }

        private void додаванняToolStripMenuItem2_Click(object sender, EventArgs e)
        {
            deliveries deliveries = new deliveries();
            deliveries.ShowDialog();
        }

        private void переглядToolStripMenuItem6_Click(object sender, EventArgs e)
        {
            DBConnection dB = new DBConnection();
            dB.Connect("", "");
            dB.openConnection();

            string query = "SELECT check_.idorder as[Код поставки],dateOrder as
[Дата поставки],nameStore as [Магазин],quantitySold as [Кількість поставки],
statusOrder as [Статус постаки],nameGoods as [Назва товару]\r\nFROM store,
checkLine, check_, checkGood,goods\r\nWHERE checkLine.idorder = check_.idorder and
checkLine.idstore = store.idstore and checkGood.idgoods = goods.idgoods and
checkGood.idorder = check_.idorder";
            SqlCommand command = new SqlCommand(query, dB.GetConnection());

            DataTable table = new DataTable();
            SqlDataAdapter adapter = new SqlDataAdapter(command);
            adapter.Fill(table);
            dataGridView1.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.AllCells;
            dataGridView1.DataSource = table;
            dB.closeConnection();
        }

        private void усіПоставкиУВибрануДатуToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Form4 form4 = new Form4();
            form4.ShowDialog();
        }

        private void усіВідправкиУВибрануДатуToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Form5 form5 = new Form5();
            form5.ShowDialog();
        }

        private void panel1_Paint(object sender, PaintEventArgs e)
        {

        }
    }
}
```