ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ОБЛІКУ ВИТРАТНОГО МАТЕРІАЛУ САЛОНІВ КРАСИ З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ C#**»

Виконала: студентка 4 курсу, групи ПД–42
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Єздакова А.С.
(прізвище та ініціали)

Керівник Гаманюк І.М.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Негоденко О.В.
"____" _____2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

ЄЗДАКОВІЙ АНАСТАСІЇ СЕРГІЇВНІ
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка програмного забезпечення з обліку витратного матеріалу салонів краси з використанням мови програмування C#»

Керівник роботи: Гаманюк І.М., ст.викл.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «18» лютого 2022 року №
2. Строк подання студентом роботи «3» червня 2022 року
3. Вхідні дані до роботи
   3.1 Методи моделювання інформаційних систем;
   3.2 Науково-технічна література з питань, пов'язаних з програмним забезпеченням щодо обліку;
   3.3 Онлайн-редактор UML-діаграм;
   3.4 Інтегроване середовище розробки Visual Studio;
4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).
   4.1 Облік як складова бізнесу
   4.2 Аналіз наявних засобів та технологій для організації обліку
   4.3 Моделювання та проектування інформаційної системи
   4.4 Тестування системи
5. Перелік демонстраційного матеріалу (назва основних слайдів)
   5.1. Титульний слайд
   5.2. Мета, об'єкт та предмет дослідження

5.3. Актуальність роботи
5.4. Аналоги
5.5. Порівняння аналогів
5.6. Технічні завдання
5.7. Інструменти, використані для реалізації
5.8. Архітектура програми
5.9. Модель предметної галузі
5.10. Функціональність системи
5.11. Взаємодія об’єктів. Діаграма послідовності
5.12. Взаємодія об’єктів. Діаграма об’єктів
5.13. Моделювання меню
5.14. Тестування програмного забезпечення
5.15. Апробація результатів дослідження
5.16. Висновки

6. Дата видачі завдання «11» квітня 2022р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04.2022	Виконано
2	Дослідження існуючих інструментів для обліку	12.04.2022 – 14.04.2022	Виконано
3	Проектування архітектури системи	15.04.2022 – 17.04.2022	Виконано
4	Розробка системи для обліку витратних матеріалів	18.04.2022 – 27.04.2022	Виконано
5	Висновки та оформлення роботи	28.04.2022 – 01.05.2022	Виконано
6	Розробка обов‘язкових демонстраційних матеріалів	02.05.2022	Виконано
7	Попередній захист роботи	26.05.2022	
8	Здача роботи	3.06.22	

Студент _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота складається з: вступу, чотирьох розділів, висновків, списку використаних джерел із 22 найменувань. У тексті дипломної роботи міститься 66 рисунків. Загальний обсяг роботи 73 листа.

ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ РОЗРОБКИ (IDE), CRUD-ЗАПИТИ, ФОРМАТ JSON, UML-ДІАГРАМА.

Склад є важливою частиною логістичної системи. Зберігання витратних матеріалів, товарів чи інших предметів є серйозною структурною одиницею бізнесу. Облік на складі дуже важливий. Контроль над продажами та доходами зменшує кількість помилок, невідповідностей та неетичних крадіжок співробітників, дозволяючи ефективно планувати поставки, підвищуючи ефективність місця для зберігання.

Вправна організація цього процесу вкрай важлива для ведення бізнесу – невеликі помилки можуть призвести до великих збитків. Ведення обліку передбачає дуже великий обсяг даних, неефективну обробку людськими ресурсами і високий ризик помилок.

Об’єкт дослідження – облік витратних матеріалів.

Предмет дослідження – програмне забезпечення з обліку витратних матеріалів.

Мета роботи – систематизація інформації щодо матеріалів на складах та створення додатку, що має основні функції обліку.

Методи дослідження – уніфікований процес створення програмного забезпечення

На основі результатів виконаних досліджень розроблена автоматизована інформаційна система обліку витратних матеріалів, в якій реалізовано оновлення бази даних та реалізовано вхідну інформацію, що є детальною і становить основу для наступної логічної та арифметичної обробки даних.

Впровадження створеної інформаційної системи обліку матеріалів дозволить отримати достовірну та своєчасну інформацію стосовно обліку наявності косметичної продукції на складі.

ЗМІСТ

ВСТУП

На сьогоднішній день бізнес та інформаційні технології тісно пов’язані один з одним. В умовах сучасного економічного розвитку необхідно безперервно вдосконалювати поточну діяльність підприємств шляхом автоматизації бізнес-процесів. Залучення інформаційних технологій до складського обліку дозволить надійно зберігати дані, зробить їх відображення більш зручним, прискорить пошук, кардинально підвищить швидкість обробки та точність результатів.

Сьогодні існує безліч програм для управління бізнес-процесами, які також включають в себе можливості обліку на складі. У більшості випадків це громіздка частина програмного забезпечення, яка охоплює майже всі аспекти бізнесу і не зручна, коли потрібен певний модуль.

Отже, обрана тема роботи є актуальною.

Об’єктом дослідження є облік витратних матеріалів.

Предметом роботи є програмне забезпечення з обліку витратних матеріалів.

Метою роботи є створення невимогливого до технічних ресурсів модулю, який виконує функції складського обліку та може працювати з іншими системами чи модулями підприємства.

В процесі дослідження вирішувалися наступні завдання:

- вивчення методики організації процесу обліку матеріалів;
- визначення основних функцій, необхідних для обліку;
- розробка інформаційної системи відповідно до моделювання.

Методика дослідження: уніфікований процес створення програмного забезпечення

Наукова новизна роботи полягає в розподілі функцій за класами.

Практична значущість результатів: даний продукт може бути використаний у будь-яких сферах діяльності, де потребується облік матеріалів на складі.

1 ОБЛІК ЯК СКЛАДОВА БІЗНЕСУ

1.1 Організація обліку в сучасному підприємстві

Метою діяльності підприємств є отримання прибутку, основним джерелом якого є реалізація готової продукції. Процес реалізації готової продукції відіграє важливе значення, оскільки реалізація як складова загального процесу відтворення є завершальним етапом руху продукту зі сфери виробництва у сферу споживання.

Чіткий, своєчасний, правильно організований облік відвантаженої продукції сприяє посиленню контролю за наявністю матеріальних цінностей, забезпеченню підприємств коштами і прискоренню обертання оборотних коштів.

Програмне забезпечення на підприємстві дозволяє щоденно складати оборотні відомості обліку випуску продукції з виробництва і руху виробів у розрізі складів.

Процес організації обліку готової продукції та її реалізації, можна умовно розподілити на етапи, на кожному з яких важливо визначити осіб, які відповідатимуть за виконання певних розподілених обов'язків, та осіб, які контролюватимуть їх виконання (Рис. 1.1) [1].

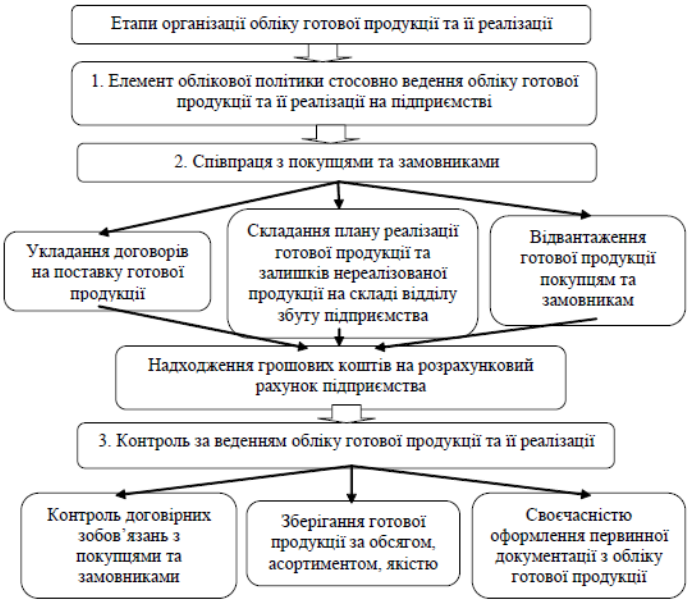


Рисунок 1.1 – Етапи організації обліку готової продукції та її реалізації

1.2 Облік в умовах цифрової економіки

Зростання ролі інформаційних технологій як фактору суспільного життя зумовило перехід до інформаційного суспільства і формування цифрової економіки визначальним трендом глобального соціально-економічного розвитку. Новітньому постіндустріальному етапу притаманні постійні технологічні інновації, збільшення зайнятості у ІТ-сфері, виробництво інформаційних продуктів і послуг, використання комп'ютерних мереж і всесвітнього інформаційного простору задля ефективної комунікації.

Значні технологічні зрушення стимулюють модернізацію бухгалтерської науки, сприяють розвитку методології та організації облікового процесу, а також актуалізують проблему позиціювання облікової системи.

Вкрай важливими є подальші наукові здобутки у цій царині – нові концепції, розробки певних видів обліку, оскільки в умовах розвитку інформаційного суспільства та цифрової економіки виникає ряд передумов для формування нової парадигми бухгалтерського обліку.

Цифрові технології наразі застосовуються у всіх напрямках суспільного життя: у системі державного управління, економіці, бізнесі, соціальній сфері. Таке перетворення пришвидшує економічні та соціальні процеси, робить їх більш якісними.

При цьому, під впливом сучасних інформаційних систем та інформаційних технологій відбуваються значні зміни в обліковій методології та практиці, що актуалізують доцільність вироблення адекватної до нових умов облікової парадигми.

Водночас протягом останніх десятиліть накопичилися проблеми, пов'язані зі зниженням функціональності обліку, зумовлені його консервативністю, недостатністю інформаційної цінності облікової інформації для зацікавлених осіб, що зумовило ряд досліджень на фундаментальному рівні, а на практичному рівні – пошук шляхів актуалізації обліку та підвищення рівня відповідності його інформації запитам користувачів [2].

2 АНАЛІЗ НАЯВНИХ ЗАСОБІВ ТА ТЕХНОЛОГІЙ ДЛЯ ОРГАНІЗАЦІЇ ОБЛІКУ

2.1 Аналіз існуючих програм з обліку складів

Програма обліку продукції є важливим інструментом для ефективного бізнесу. Автоматизовані системи можуть підвищити продуктивність бізнесу. За допомогою них можна оптимізувати не тільки процес вирішення проблем, а й бухгалтерський облік. Контроль кількості та залишків продукції допомагає оптимізувати планування.

Програмні продукти містять функціональні можливості, які полегшують виконання багатьох операцій. При їх застосуванні можна швидше вирішувати завдання та підвищити продуктивність.

Найпопулярнішими програмами у цій ніші є:

1. «BAS Комплексне управління підприємством»

Ця конфігурація використовується для створення єдиної бази даних підприємства. Програмне забезпечення надає ордерну схему для складського обліку та застосовується для отримання та доставки окремо. На складі реалізовано цільове зберігання продукції. Залишки відслідковуються там, де вони знаходяться - осередки, полиці, стелажі, а також упаковки. Відбувається керування такими процесами: розклад товару, збір, переміщення [3].

Програма має наступний інтерфейс (Рис. 2.1 – 2.4):

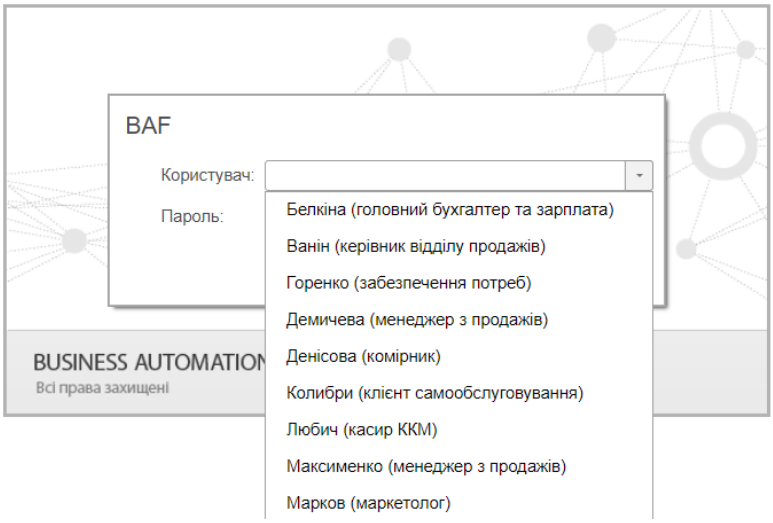


Рисунок 2.1 – інтерфейс BAS Комплексне управління підприємством

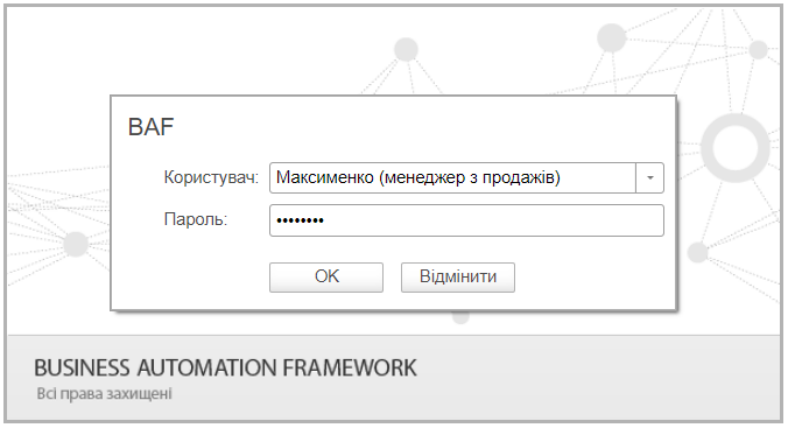


Рисунок 2.2 – інтерфейс BAS Комплексне управління підприємством

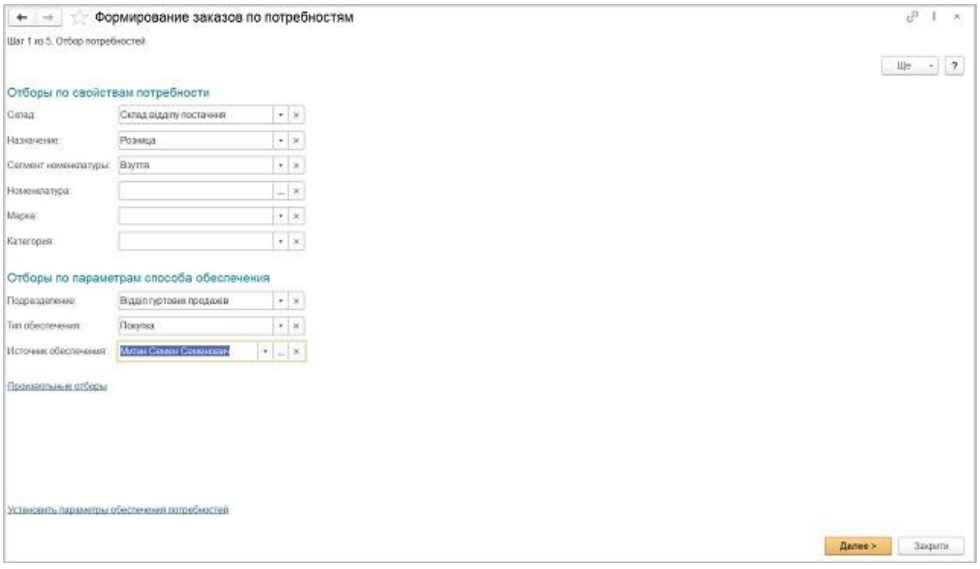


Рисунок 2.3 – інтерфейс BAS Комплексне управління підприємством

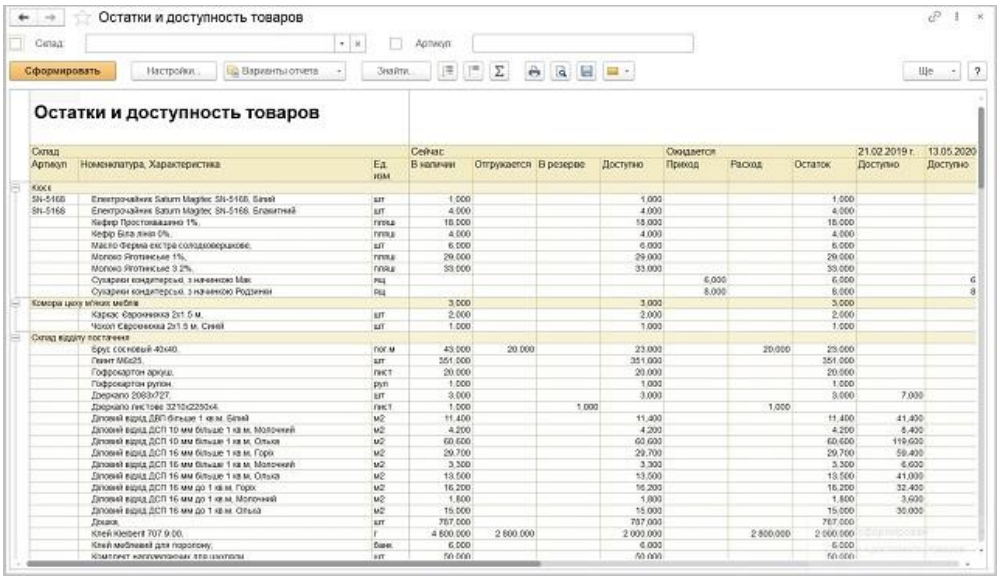


Рисунок 2.4 – інтерфейс BAS Комплексне управління підприємством

2. «ERP FOSS»

Вся інформація про продукти, які надходять на склади або відвантажуються споживачам, зберігається в хмарі. Програма також підключається до хмарного сервісу ERP FOSS та має наступні функції:

1. управління складом онлайн, повна інформація щодо організації зберігання продукції на складі;
2. повна інформація щодо маршрути руху вантажу, що надходить і виходить зі складу;
3. імпорт даних з таблиць Excel для автоматичного введення товарів у програму.

Є можливість підключити ПЗ до найбільшої в Україні служби доставки «Нова Пошта» [4].

Інтерфейс програми такий (Рис. 2.5, 2.6):

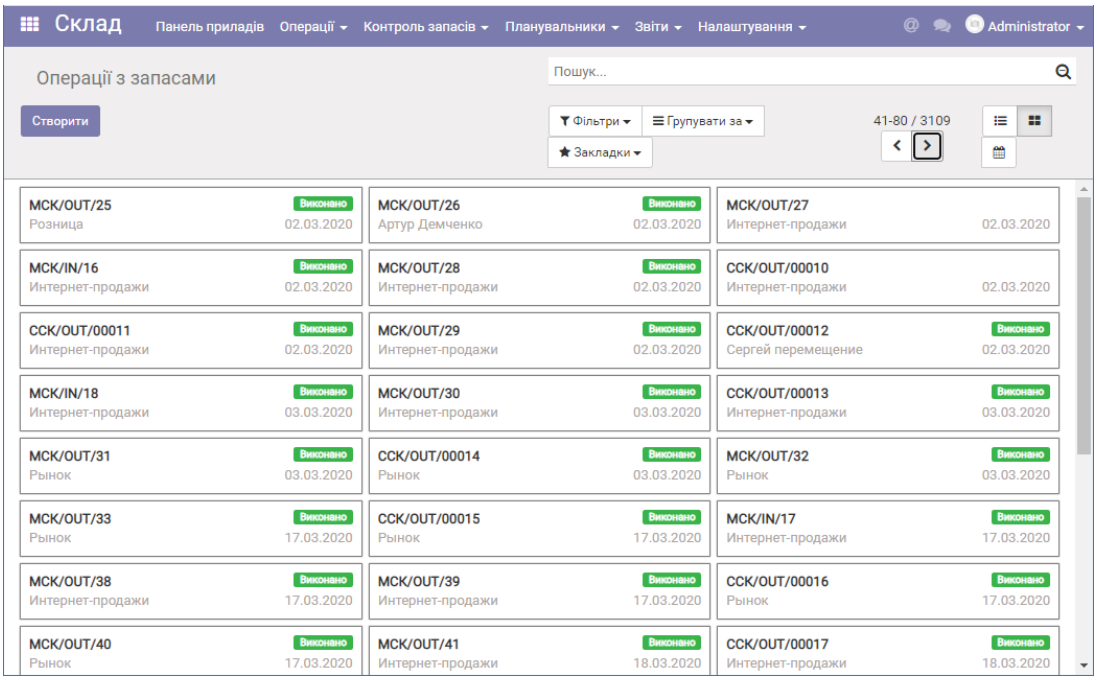


Рисунок 2.5 – інтерфейс ERP FOSS

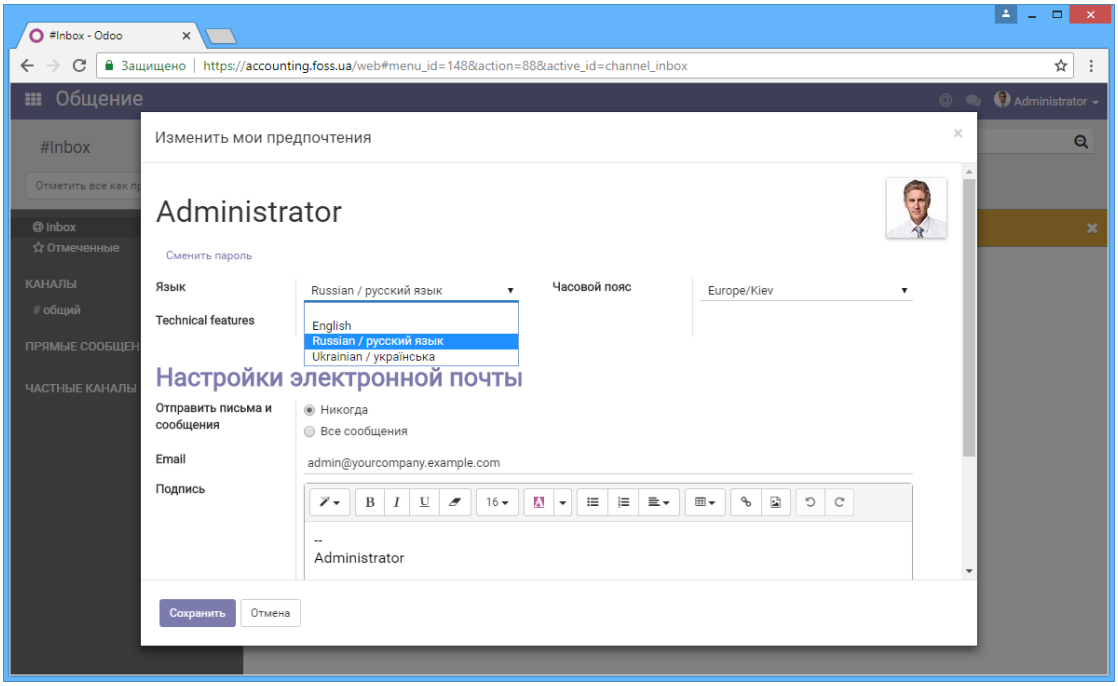


Рисунок 2.6 – інтерфейс ERP FOSS

3. «IBS Торгівля & Склад»

Програма складського обліку, яка підтримує замовлення постачальників, надходження, повернення, переміщення товарів між складами, списання (пошкодження) [5].

Вигляд програми наступний (Рис. 2.7, 2.8):

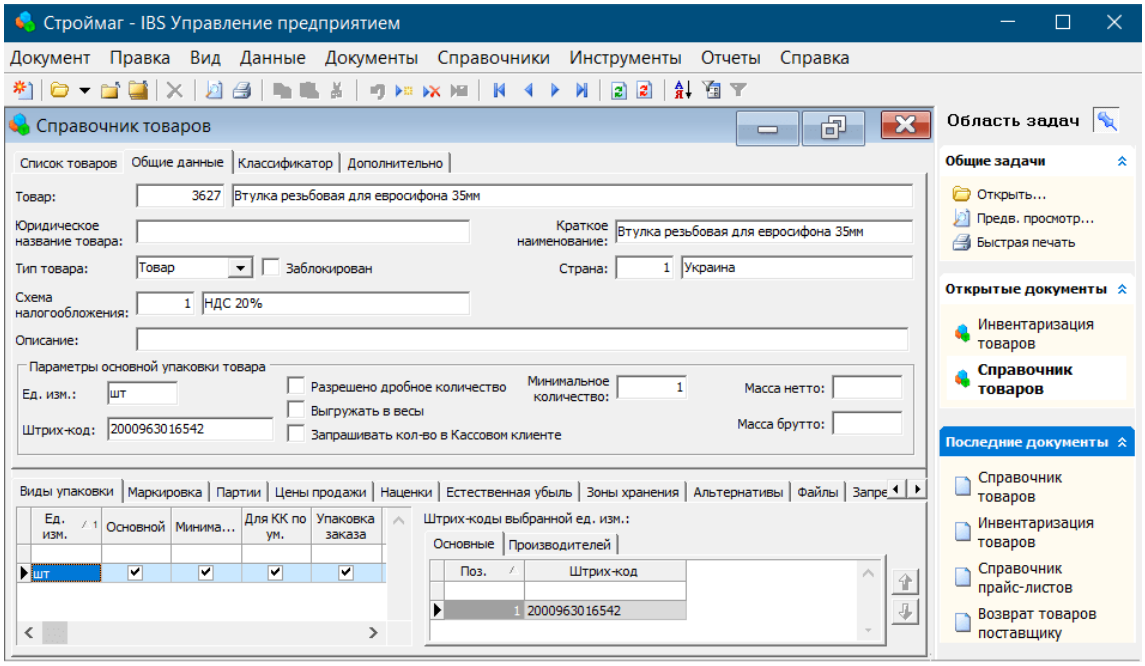


Рисунок 2.7 – інтерфейс IBS Торгівля & Склад

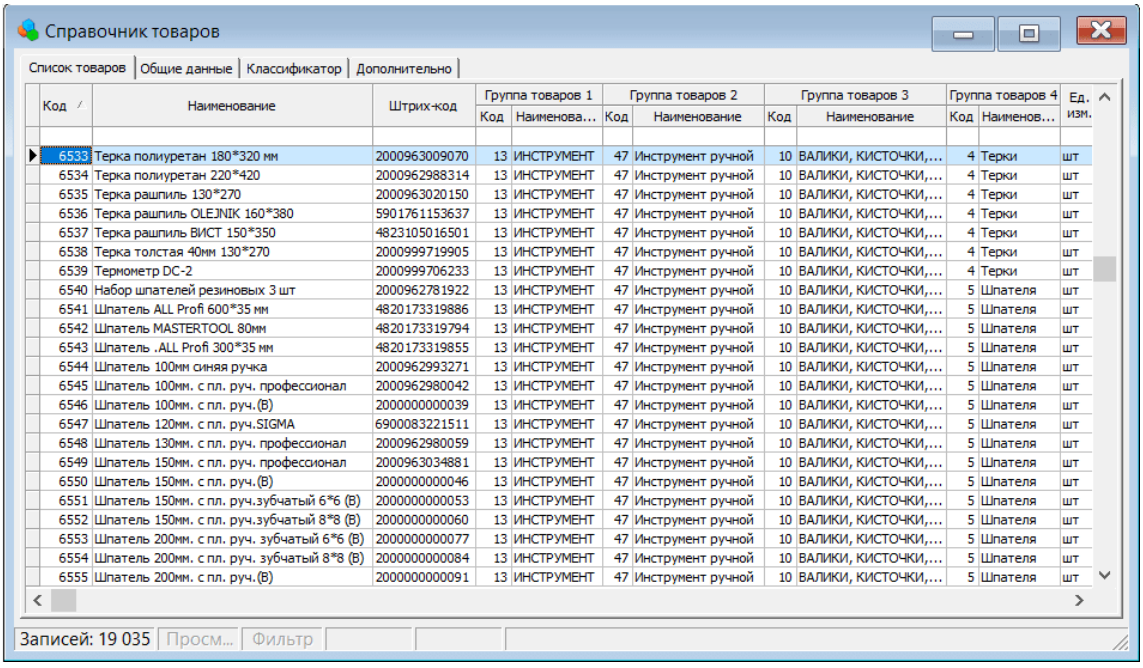


Рисунок 2.8 – інтерфейс IBS Торгівля & Склад

Отже, розглянуті вище програми мають зручний інтерфейс та корисний функціонал, але в них не вистачає взаємодії з іншими системами. Однак, створений проект це враховує.

Сенс обліку не тільки в тому, щоб визначати наявні товарні запаси на складах, а й стежити за тим, хто саме і яким чином цією продукцією керує. При створенні таблиць, усі записи про обробку вантажу мають бути пов’язані з користувачем. Це полегшить встановлення винних і притягнення їх до відповідальності у разі крадіжки.

Модуль має контроль над критичною для бізнесу інформацією, тому важливо зберігати дані в безпеці та запобігати помилковим діям користувача, які можуть призвести до серйозних наслідків.

2.2 Вибір технологій та засобів розробки програмного забезпечення

Для створення програмного забезпечення з обліку витратних матеріалів потрібно обрати засоби розробки. До засобів розробки належить: інтегроване середовище розробки (англ. IDE), мова програмування, система керування базами даних.

Також в проекті необхідно вирішити наступний набір завдань:

− розробка структур даних, що відображають сутності складського обліку;

− розробка надійних алгоритмів обчислення кількісних значень;

− проектування журналів складських операцій;

− розробка CRUD-запитів (create, read, update, delete);

− розробка зручного інтерфейсу;

− реалізація імпорту та експорту даних, для взаємодії з іншими системами;

− реалізація пошуку.

2.2.1 Огляд онлайн-редактору UML-діаграм UMLetino

UMLetino — це онлайн-інструмент для побудови UML-діаграм.

Основні можливості: веб-додаток без встановлення, збереження діаграм в сховищі браузера, підтримка багатьох типів діаграм UML, прості модифікації елементів UML на основі розмітки, експорт файлів у форматах PNG, EPS, PDF, JPG, SVG.

UMLetino підтримує різноманітні типи діаграм UML: діаграми класів, діаграми варіантів використання, діаграми послідовності, діаграми станів, діаграми розгортання, діаграми діяльності та інші [6].

Має наступний вигляд у браузері (Рис. 2.9):

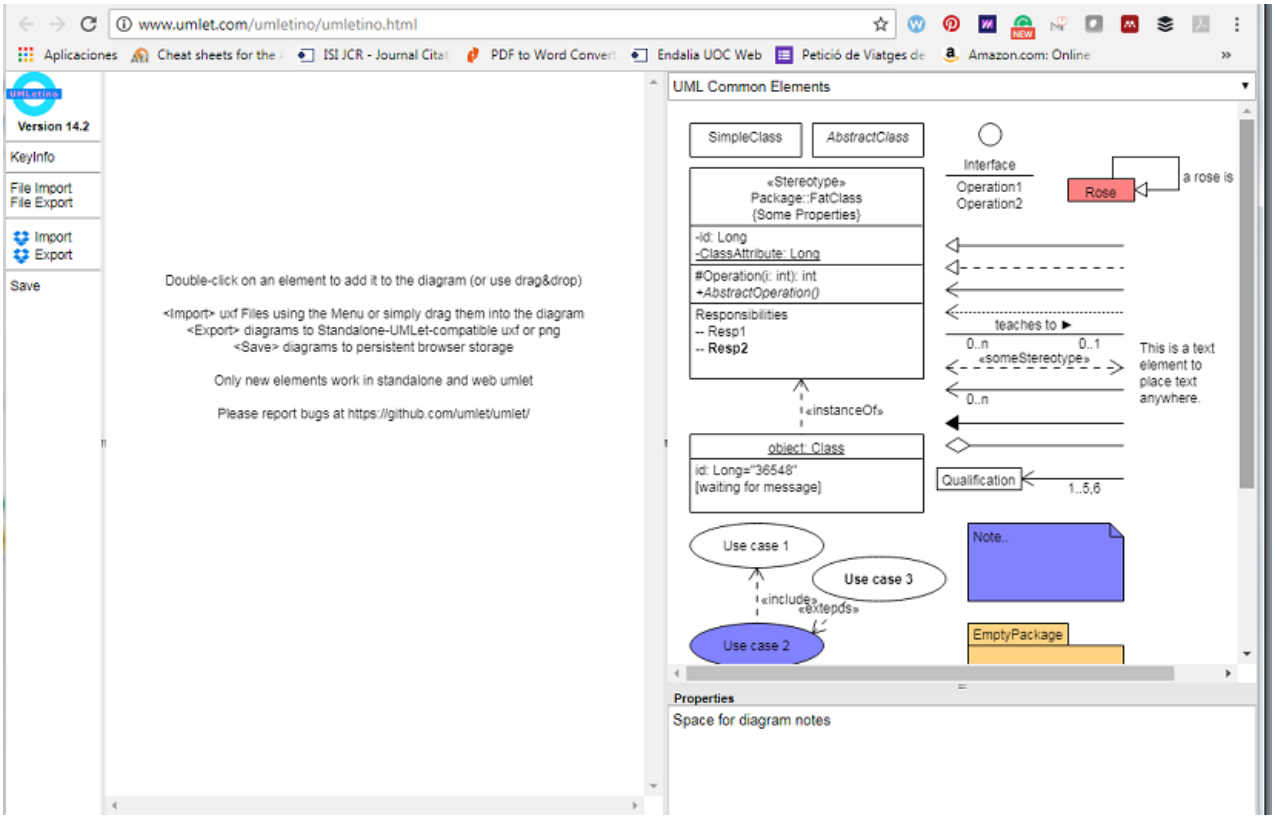


Рисунок 2.9 – інтерфейс UMLetino

2.2.2 Excel для відображення предметної галузі

Можна використовувати Microsoft Excel для зображення матеріального середовища. З його допомогою легше зрозуміти структуру побудови UML-діаграм, а згодом і написати код.

З цією метою створюється кілька таблиць (Рис. 2.10, 2.11):

BeautySalon	
Id	*SalonName*
1	Salon1
2	Salon2
3	Salon3

SalonAddress		
Id	*AddressOfSalon*	*SalonId*
1	Salon2Address	2
2	Salon3Address	3
3	Salon1Address	1

Рисунок 2.10 – таблиці BeautySalon і SalonAddress

ExpendableMaterial			
Id	*MaterialName*	*Price*	*UnitMeasurement*
1	Material1	100	шт.
2	Material2	200	л

BeautySalonExpendableMaterial			
Id	*BeautySalonId*	*ExpendableMaterialId*	*Quantity*
1	1	1	10
2	1	2	20
3	2	1	30
4	3	2	40

Рисунок 2.11 – таблиці ExpandableMaterial і BeautySalonExpandableMaterial

На даних таблицях відображаються салони краси та їх зв’язки з адресами і витратним матеріалом. Кожний салон має свій Id та з’єднується з відповідною адресою за допомогою властивості SalonId в таблиці SalonAddress. Також є таблиця ExpendableMaterial, що вказує витратний матеріал, його ціну та міру виміру. Таблиця BeautySalonExpendableMaterial показує в якому салоні знаходиться той чи інший товар та його кількість.

Аналогічно робиться зі складами (Рис. 2.12, 2.13):

Warehouse	
Id	*WarehouseName*
1	Warehouse1
2	Warehouse2

WarehouseAddress		
Id	*AddressOfWarehouse*	*WarehouseId*
1	Warehouse2Address	2
2	Warehouse3Address	3

Рисунок 2.12 – таблиці Warehouse і WarehouseAddress

WarehouseExpendableMaterial			
Id	*WarehouseId*	*ExpendableMaterialId*	*Remains*
1	1	1	40
2	1	2	50
3	2	1	60

Рисунок 2.13 – таблиця WarehouseExpandableMaterial

В свою чергу, склад теж має свою адресу. А з таблиці WarehouseExpandableMaterial видно на якому складі знаходиться матеріал та який його залишок.

Виходячи з таблиць, можемо побудувати таку діаграму предметної галузі (Рис. 2.14):

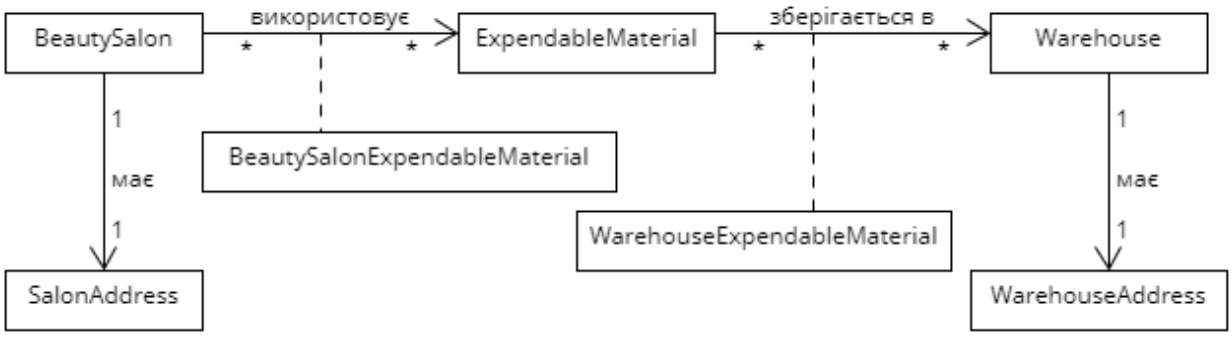


Рисунок 2.14 – Діаграма предметної галузі

2.2.3 Формат JSON

На складі багато інформації: список витратних матеріалів, список складів, список адрес складів, список салонів, список адрес салонів тощо. Питання полягає в тому, де всю цю інформацію зберігати.

Наразі популярним способом реалізувати зберігання даних у додатках є використання файлів у форматі JSON і баз даних.

JSON (Рис. 2.15) — це текстовий формат обміну даними між комп’ютерами. JSON заснований на тексті та зрозумілий людині. Цей формат дозволяє описувати об’єкти та інші структури даних. Він в основному застосовується для передачі структурованої інформації по мережі.

JSON заснований на двох правилах:

1. набір пар ім'я-значення (різними мовами втілено як об'єкт, запис, структура, словник, хеш-таблиця, список ключів або асоціативний масив);

2. упорядкований список значень (на багатьох мовах втілено як масив, вектор, список або послідовність).

Рисунок 2.15 – Приклад JSON

Вигідно використовувати json-файли, коли потрібно зберегти погано зв’язані структури, але JSON стає неефективним при збереженні структур, які тісно пов’язані одна з одною.

2.2.4 Мова C# та її особливості

Щоб реалізувати даний проект у середовищі розробки Microsoft Visual Studio використовується мова програмування C#, яка є елементом .Net Framework. Для розробки будь-якого додатку на операційній системі Windows створене середовище .Net, в той час як мова C# вживається разом з .Net Framework. Тому на сьогоднішній момент поєднання C# і .Net є найбільш продуктивним для програмістів.

C# — сучасна, універсальна, об'єктно-орієнтована мова програмування, розроблена компанією Microsoft.

Ця мова програмування дуже проста для вивчення. Важливо зазначити, що програму, написану на C#, можна розгорнути в будь-якій операційній системі, наприклад, Android, iOS, Windows або хмарній платформі.

Існує багато важливих функцій мови C#, які роблять її більш корисною та унікальною порівняно з іншими мовами:

- дуже швидка, її компіляція та час виконання не займають багато часу;

- має багатий набір бібліотечних функцій і типів даних;
- є однією з сучасних мов програмування;
- є безпечним для типу кодом, який може отримати доступ лише до розташування пам'яті та має дозвіл на виконання. Таким чином, це покращує безпеку програми;
- для вирішення великих проблем програмування на C# ділить проблему на менші модулі, які називаються функціями або процедурами, кожна з яких несе певну відповідальність, тому мова C# називається структурованою мовою програмування.

Але C# також має певні недоліки:

- повністю заснована на платформі Microsoft .Net, тому ця мова не є гнучкою;
- після внесення змін у написаному коді, необхідно його заново компілювати [7].

2.2.5 Середовище розробки Visual Studio

Інтегроване середовище розробки (англ. Integrated Development Environment, IDE) — це програмне забезпечення для створення програм, яке поєднує звичайні інструменти розробника в єдиний графічний інтерфейс користувача (англ. Graphical User Interface, GUI).

IDE зазвичай складається з:

− редактору вихідного коду: текстовий редактор, який може допомогти в написанні програмного коду з такими функціями, як підсвічування синтаксису за допомогою візуальних підказок, забезпечення автоматичного заповнення для певної мови та перевірка помилок під час написання коду.

− локальної автоматизації збірки: компіляція вихідного коду комп’ютера в двійковий код, упаковка двійкового коду та запуск автоматизованих тестів.

− налагоджувача: програма для тестування інших програм, яка може графічно відображати розташування помилки в оригінальному коді [8].

Microsoft Visual Studio — це інтегроване середовище, створене корпорацією Microsoft для розробки графічного інтерфейсу користувача, консолі, веб-додатків, веб-програм, мобільних додатків, хмарних програм і веб-сервісів тощо. За допомогою даного IDE можна зробити керований, а також написати власний код. Visual Studio (VS) використовує різні платформи програмного забезпечення, які включають в себе: Windows Store, Microsoft Silverlight, Windows API. VS застосовують для написання коду на таких мовах як: C#, C++, VB (Visual Basic), Python, JavaScript та ін [9].

2.2.6 Платформа .Net Framework

.Net Framework — це платформа для розробки програмного забезпечення, вироблена корпорацією Microsoft для створення та запуску програм Windows. Платформа складається з інструментів розробника, мов програмування та бібліотек для написання настільних і веб-додатків, а також веб-сервісів та ігор. Вона підтримує різні мови програмування. Таким чином, розробники можуть вибрати мову для розробки необхідної програми. Популярними є Visual Basic і C#.

.NET Framework має набір стандартних бібліотек класів. Бібліотека класів — це комплекс методів і функцій, які можна використовувати для поставленого завдання. Програми, що використовують фреймворк .NET можуть працювати на всіх платформах Windows [10].

3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Бачення

Дата внесення змін: 05.05.2022

Версія: Кінцевий варіант.

Дата: 05.05.2022

Опис: Кінцевий варіант. Всі уточнення враховано.

Автор: Єздакова А.С.

Вступ

Надійний застосунок з можливостями розподіленої системи та інтеграції з різними зовнішніми системами.

Позиціонування

Економічні передумови: існуючі програмні продукти не забезпечують роботу дистанційно і не інтегруються з різними зовнішніми системами.

Формулювання проблеми

Традиційні системи не забезпечують дистанційну роботу.

Традиційні системи не забезпечують інтеграцію із зовнішніми системами. При залученні зовнішніх систем до аналізу обліку витратних матеріалів існує потреба інтеграції з цими зовнішніми системами. Це стосується керівників компанії.

Місце системи

Система призначена для менеджера з обліку витратних матеріалів для обліку матеріалів, закріплення матеріалів за складами та салонами.

Система призначена для менеджера з обліку складів для обліку складів з їх адресами.

Система призначена для менеджера з обліку салонів для обліку салонів з їх адресами.

Система призначена для представлення інформації для системи аналізу процесу обліку.

Зацікавлені особи

Менеджер з обліку витратних матеріалів – для обліку матеріалів, закріплення матеріалів за складами та салонами.

Менеджер з обліку складів – для обліку складів з їх адресами.

Менеджер з обліку салонів – для обліку салонів з їх адресами.

Керівництво компанії – для представлення інформації системі аналізу процесу обліку.

Основні задачі високого рівня

- Облік витратних матеріалів, закріплення матеріалів за складами та за салонами (високий пріоритет);
- Облік складів (високий пріоритет);
- Облік салонів (високий пріоритет);
- Представлення інформації системі аналізу процесу обліку (низький пріоритет).

Задачі рівня користувача

Менеджер з обліку витратних матеріалів: вносить дані щодо матеріалів, закріплення матеріалів за складами та салонами.

Менеджер з обліку складів: вносить дані щодо складів та їх адрес.

Менеджер з обліку салонів: вносить дані щодо салонів та їх адрес.

Система аналізу процесу обліку: забезпечує аналіз процесу обліку.

Огляд

Перспективи продукту: система буде обслуговувати користувачів і взаємодіяти з іншими системами, як показано на Контекстній діаграмі (Рис. 3.1).

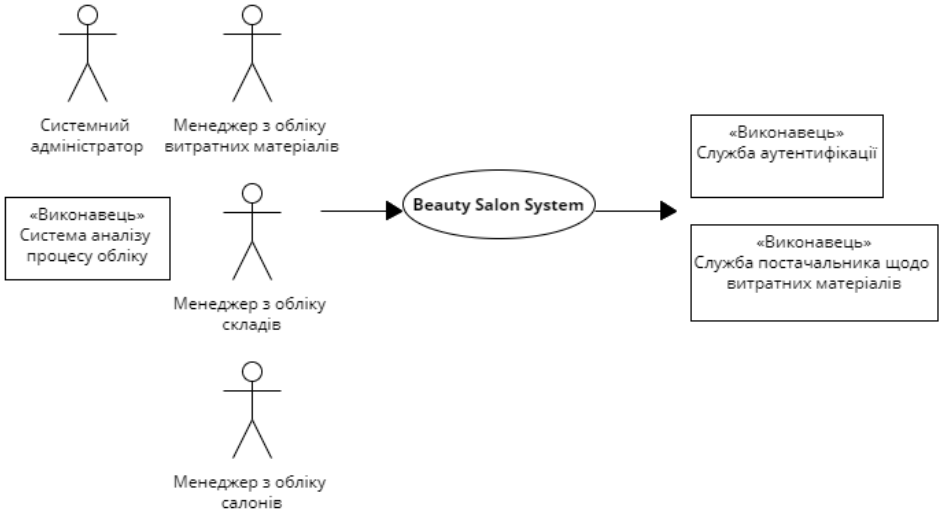


Рисунок 3.1 – Контекстна діаграма

Контекстна діаграма (англ. Context Diagram) показує взаємодію між системою та іншими акторами (зовнішніми факторами), з якими система призначена для взаємодії. Контекстна діаграма показує всю систему як єдиний процес [11].

Переваги системи

Розподілена можливість введення даних.

Інтеграція з системою аналізу процесу обліку.

Припущення і залежності

Ресурси на розробку системи будуть поступати в повному обсязі.

Вартість і ціноутворення

Вартість розробки системи: розраховується окремо.

Вартість супроводження системи 10% від вартості розробки системи щороку.

Ліцензування і встановлення

Здійснити ліцензування (сертифікацію) системи.

Встановити систему після ліцензування.

Основні властивості системи

- Облік витратних матеріалів та закріплення їх за складами та салонами;
- Облік складів та їх адрес;
- Облік салонів та їх адрес;

- Інтеграція з системою аналізу процесу обліку.

Інші вимоги та обмеження

Проектування здійснювати на матеріальній базі виконавця.

Перелік нормативно-правових актів щодо предметної галузі надається окремо.

3.1 Модель предметної галузі

Діаграма предметної галузі (зв'язків між сутностями, англ. Entity Relationship Diagram) — це тип блок-схеми, яка ілюструє, як «суб'єкти» пов'язані один з одним у системі. Діаграма найчастіше використовується для проектування або налагодження реляційних баз даних у сферах програмної інженерії, бізнес-інформаційних систем, освіти та досліджень [12].

Діаграма предметної галузі зображена на рисунку 3.2.

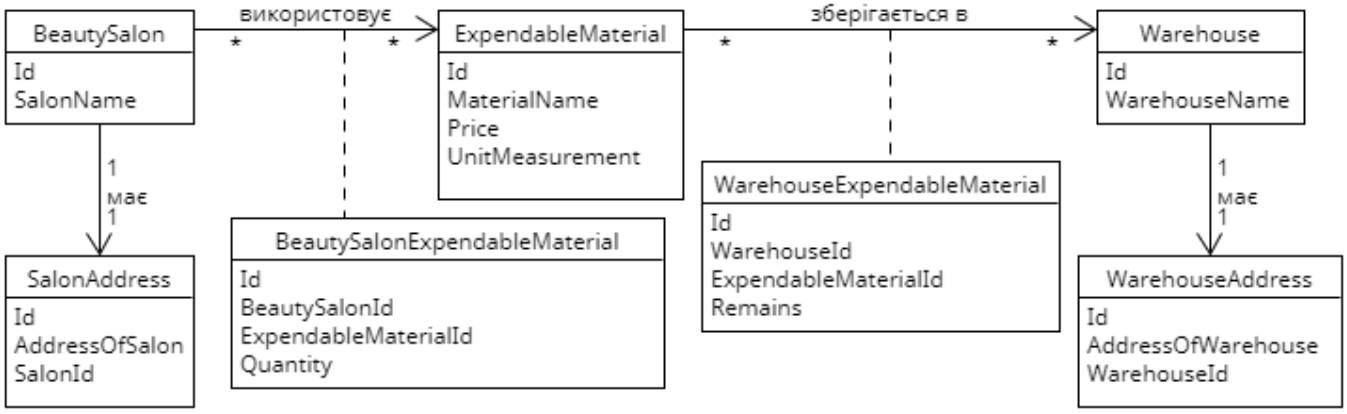


Рисунок 3.2 – Діаграма предметної галузі

На основі створеної діаграми видно, що суб'єкт (клас) BeautySalon з полями Id і SalonName має відношення «один до одного» до класу SalonAddress. Також BeautySalon використовує клас ExpandableMaterial та BeautySalonExpandableMaterial.

Суб'єкт ExpandableMaterial зберігається в суб'єкті Warehouse та використовує WarehouseExpandableMaterial.

В свою чергу, Warehouse має відношення «один до одного» до класу WarehouseAddress.

Рядки в таблицях бази даних створюються за допомогою коду (Рис. 3.3 - 3.5):

```
20 references
internal class BeautySalon : IId
{
    6 references
    public int Id { get; set; }
    2 references
    public string SalonName { get; set; }
    0 references
    public BeautySalon()
    {

    }
    4 references
    public BeautySalon(string SalonName)
    {
        this.SalonName = SalonName;
    }
```

Рисунок 3.3 – реалізація Діаграми предметної галузі

```
34 references
public class ExpendableMaterial : IId
{
    6 references
    public int Id { get; set; }
    3 references
    public string MaterialName { get; set; }
    3 references
    public float Price { get; set; }
    3 references
    public string UnitMeasurement { get; set; }
    0 references
    public ExpendableMaterial()
    {

    }
    7 references
    public ExpendableMaterial(string MaterialName, float Price, string UnitMeasurement)
    {
        this.MaterialName = MaterialName;
        this.Price = Price;
        this.UnitMeasurement = UnitMeasurement;
    }
```

Рисунок 3.4 – реалізація Діаграми предметної галузі

```
15 references
internal class BeautySalonExpendableMaterial : IId
{
    6 references
    public int Id { get; set; }
    2 references
    public int BeautySalonId { get; set; }
    2 references
    public int ExpendableMaterialId { get; set; }
    2 references
    public int Quantity { get; set; }
    0 references
    public BeautySalonExpendableMaterial()
    {

    }
    5 references
    public BeautySalonExpendableMaterial(int BeautySalonId, int ExpendableMaterialId, int Quantity)
    {
        this.BeautySalonId = BeautySalonId;
        this.ExpendableMaterialId = ExpendableMaterialId;
        this.Quantity = Quantity;
    }
```

Рисунок 3.5 – реалізація Діаграми предметної галузі

3.2 Модель прецедентів

UML-діаграма прецедентів (варіантів використання, англ. Use-Case Diagram) моделює поведінку системи. Ця діаграма визначає взаємодії між системою та її учасниками (акторами). Варіанти використання та дійові особи на діаграмах описують, що робить система і як учасники її використовують, але не те, як система працює всередині [13].

Діаграма варіантів використання менеджера з обліку витратних матеріалів представлена на рисунку 3.6.

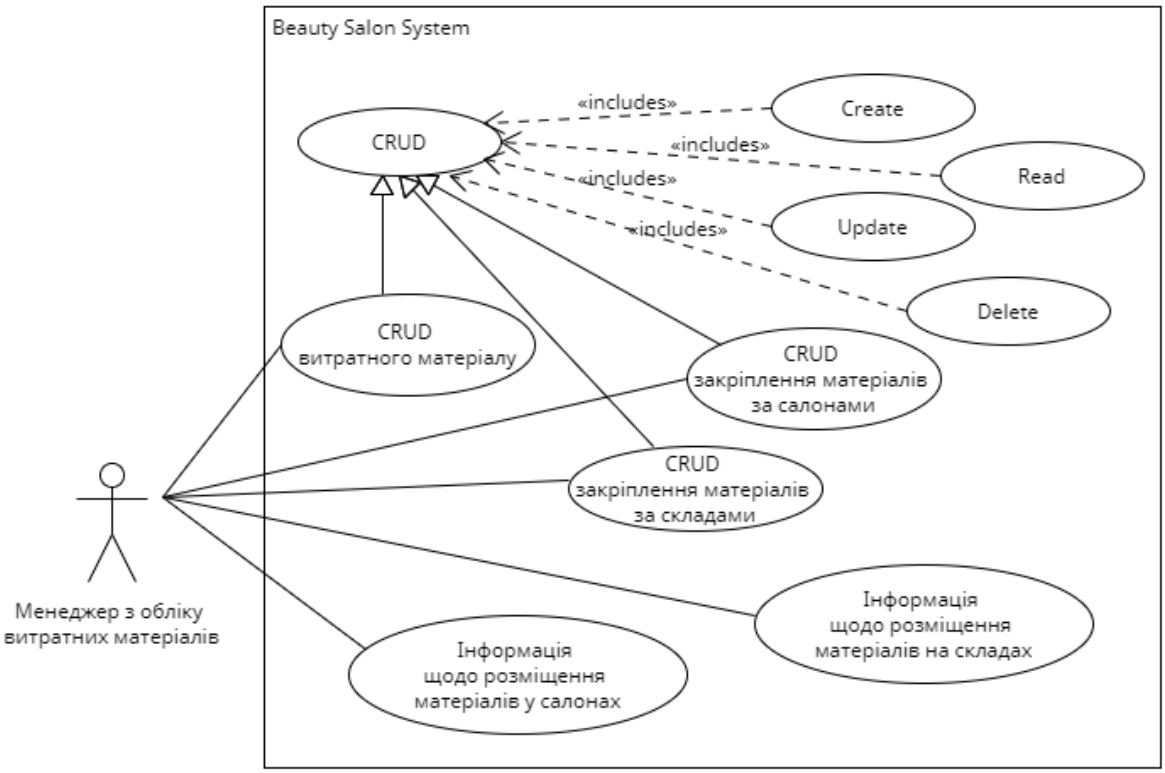


Рисунок 3.6 – Діаграма прецедентів менеджера з обліку витратних матеріалів

Виходячи з побудованої діаграми бачимо, що для сутності «менеджер з обліку витратного матеріалу» доступні п’ять варіантів використання: «CRUD витратного матеріалу», «CRUD закріплення матеріалів за складами», «CRUD закріплення матеріалів за салонами», «Інформація щодо розміщення матеріалів на складах», «Інформація щодо розміщення матеріалів у салонах».

Варіант використання «CRUD витратного матеріалу» містить розширення: створити матеріал, змінити матеріал, знайти матеріал по ID, видалити матеріал.

Для варіанту використання «CRUD закріплення матеріалів за складами» є розширення: створити матеріал, змінити матеріал, знайти матеріал по ID, видалити матеріал, закріпити матеріал за складом.

Варіант використання «CRUD закріплення матеріалів за салонами» містить розширення: створити матеріал, змінити матеріал, знайти матеріал по ID, видалити матеріал, закріпити матеріал за салоном.

Варіант використання «Інформація щодо розміщення матеріалів на складах» включає в себе перегляд вже розміщених матеріалів на складах.

Варіант використання «Інформація щодо розміщення матеріалів у салонах» включає в себе перегляд вже розміщених матеріалів у салонах.

Прецедент менеджера з обліку витратних матеріалів створюється за допомогою такого коду (Рис. 3.7):

```
5 references
internal class EM_ManagerCRUD_Service : IEM_ManagerCRUD_Service
{
    DB db;
    1 reference
    public EM_ManagerCRUD_Service(DB db)
    {
        this.db = db;
    }
    2 references
    public void ExpandableMaterialCreate(string MaterialName, float Price, string UnitMeasurement)
    {
        ExpendableMaterial expendableMaterial = new ExpendableMaterial(MaterialName, Price, UnitMeasurement);
        db.dbExpendableMaterial.AddItem(expendableMaterial);
    }
    4 references
    public ExpendableMaterial ExpandableMaterialGetById(int id)
    {
        ExpendableMaterial result = default(ExpendableMaterial);
        result = db.dbExpendableMaterial.GetItemById(id);
        return result;
    }
    2 references
    public bool ExpandableMaterialUpdate(ExpendableMaterial oldItem, string MaterialName, float Price, string UnitMeasurement)
    {
        ExpendableMaterial newItem = new ExpendableMaterial(MaterialName, Price, UnitMeasurement);
        return db.dbExpendableMaterial.UpdateItem(oldItem, newItem);
    }
    2 references
    public bool ExpandableMaterialDelete(ExpendableMaterial item)
    {
        return db.dbExpendableMaterial.DeleteItem(item);
    }
```

Рисунок 3.7 – реалізація методів CRUD менеджера з обліку витратних матеріалів

Діаграма варіантів використання менеджера з обліку складів представлена на рисунку 3.8.

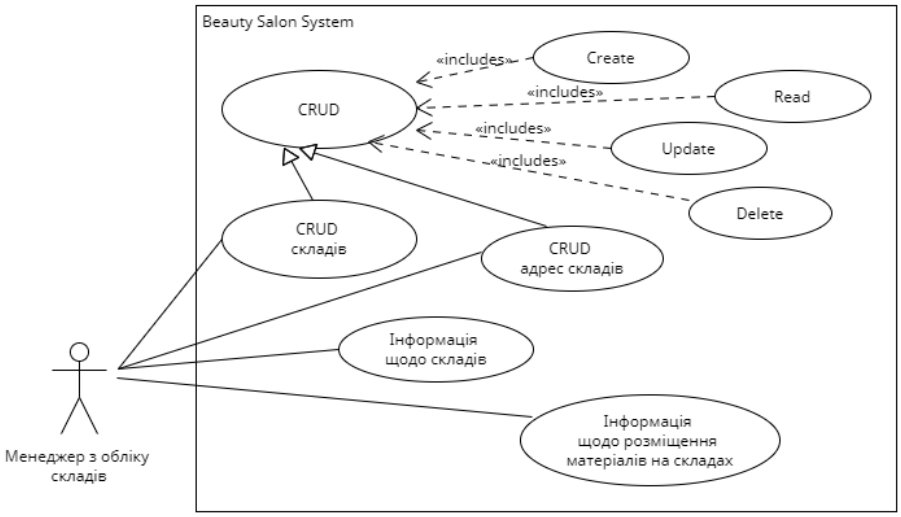


Рисунок 3.8 – Діаграма прецедентів менеджера з обліку складів

З цієї діаграми видно, що менеджеру з обліку складів доступні чотири варіантів використання: «CRUD складів», «CRUD адрес складів», «Інформація щодо складів», «Інформація щодо розміщення матеріалів на складах».

Варіант використання «CRUD складів» містить розширення: створити склад, змінити склад, знайти склад по ID, видалити склад.

Для варіанту використання «CRUD адрес складів» є розширення: створити адресу складу, змінити адресу складу, знайти адресу складу по ID, видалити адресу складу.

Варіант використання «Інформація щодо складів» включає в себе перегляд вже існуючої інформації щодо складів.

Варіант використання «Інформація щодо розміщення матеріалів на складах» включає в себе перегляд вже розміщених матеріалів на складах.

Прецедент менеджера з обліку складів створюється за допомогою наступного коду (Рис. 3.9):

```
3 references
internal class WarehouseManagerCRUD_Service
{
    DB db;
    0 references
    public WarehouseManagerCRUD_Service(DB db)
    {
        this.db = db;
    }
    1 reference
    public void WarehouseCreate(string WarehouseName)
    {
        Warehouse warehouse = new Warehouse(WarehouseName);
        db.dbWarehouse.AddItem(warehouse);
    }
    1 reference
    public Warehouse WarehouseGetById(int id)
    {
        Warehouse result = default(Warehouse);
        result = db.dbWarehouse.GetItemById(id);
        return result;
    }
```

Рисунок 3.9 – реалізація методів CREATE та READ менеджера з обліку складів

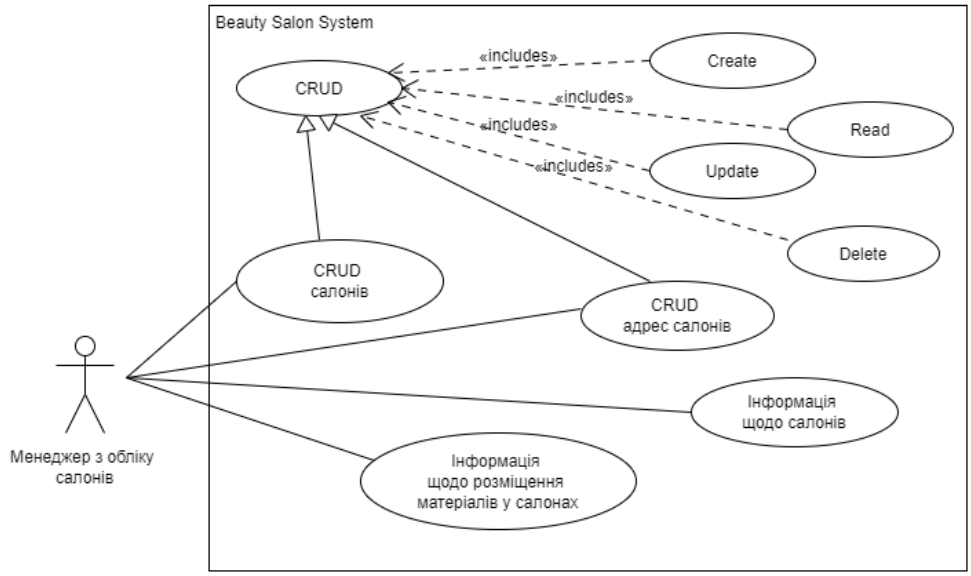


Рисунок 3.10 – Діаграма прецедентів менеджера з обліку салонів

Дана діаграма (Рис. 3.10) показує, що у менеджера з обліку салонів також чотири варіантів використання: «CRUD салонів», «CRUD адрес салонів», «Інформація щодо салонів», «Інформація щодо розміщення матеріалів у салонах».

Варіант використання «CRUD салонів» містить розширення: створити салон, змінити салон, знайти салон по ID, видалити салон.

Для варіанту використання «CRUD адрес салонів» є розширення: створити адресу салону, змінити адресу салону, знайти адресу салону по ID, видалити адресу салону.

Варіант використання «Інформація щодо салонів» включає в себе перегляд вже існуючої інформації щодо салонів.

Варіант використання «Інформація щодо розміщення матеріалів у салонах» включає в себе перегляд вже розміщених матеріалів у салонах.

Прецедент менеджера з обліку салонів створюється за допомогою такого коду (Рис. 3.11):

```
internal class SalonManagerCRUD_Service
{
    DB db;
    0 references
    public SalonManagerCRUD_Service(DB db)
    {
        this.db = db;
    }
    1 reference
    public void SalonCreate(string salonName)
    {
        BeautySalon beautySalon = new BeautySalon(salonName);
        db.dbBeautySalon.AddItem(beautySalon);
    }
    1 reference
    public BeautySalon SalonGetById(int id)
    {
        BeautySalon result = default(BeautySalon);
        result = db.dbBeautySalon.GetItemById(id);
        return result;
    }
}
```

Рисунок 3.11 – реалізація методів CREATE та READ менеджера з обліку салонів

3.3 Нефункціональні вимоги

Додаткова специфікація

Дата внесення змін: 05.05.2022

Версія: Кінцевий варіант.

Дата: 05.05.2022

Опис: Кінцевий варіант. Всі уточнення враховано.

Автор: Єздакова А.С.

Вступ

У цьому документі описано всі вимоги до системи, які не увійшли до опису прецедентів.

Наскрізна функціональність

Реєстрація подій і обробка помилок: всі події реєструються в окремому файлі, всі помилки реєструються в окремому файлі.

Бізнес правила, які можна підключити

Необхідно забезпечити можливість налаштування функціональності системи в різних точках сценаріїв деяких прецедентів (ці точки необхідно визначити) на основі заданих правил.

Безпека

Необхідно виконувати аутентифікацію усіх користувачів.

Зручність використання

Людський фактор

Користувачі системи будуть працювати з великим монітором (не з мобільним телефоном), тому необхідне таке:

Текст повинен бути видно з відстані 1 метра;

Треба уникати мерехтливих кольорів.

Швидка, проста і коректна обробка інформації - це головні принципи системи.

Повідомлення про введення хибної інформації необхідно супроводжувати звуковими сигналами. Цей пункт у програмі реалізовано наступним чином (Рис. 3.12):

```
Console.WriteLine(value: "Not Found");
Console.Beep();
```

Рисунок 3.12 – реалізація коду при введені хибного значення

Надійність

Можливість відновлення інформації: необхідно забезпечити локальне збереження даних.

Продуктивність

Наша задача - виконати запит щодо інформації не більш ніж за 10 секунд у 90% випадків.

Можливість підтримки

Адаптація системи: різні користувачі можуть встановлювати свої бізнес правила для обробки даних. Тому у визначених точках необхідно забезпечити можливість підключення бізнес-правил.

Конфігурування: система може налаштовуватися.

Обмеження

Керівництво проекту вимагає використовувати технологію .NET.

Придбані компоненти

Придбання компонентів на цей час не планується.

Безкоштовні компоненти на основі відкритого коду: використання безкоштовних компонентів не планується.

Інтерфейси

Важливі інтерфейси і апаратні засоби:

Монітор з діагоналлю не менш, ніж 15 дюймів;

Пристрій для друку.

Бізнес-правила

Правило: витратний матеріал можна видалити тільки тоді, коли відсутні закріплення його за складами чи салонами.

Програмні інтерфейси

Для зовнішніх систем необхідно забезпечити можливість підключення до нашої системи у форматі JSON.

Питання законодавства

Необхідно враховувати всі необхідні нормативно-правові акти.

Інформація з предметної галузі

Введення даних: дані вводяться окремими користувачами (менеджером з обліку витратних матеріалів, менеджером з обліку складів, менеджером з обліку салонів).

3.4 Модель проектування

3.4.1 Проектування діяльності

Діаграма діяльності (англ. Activity Diagram) — це діаграма для опису динамічних аспектів системи. Діаграма діяльності є розширеною версією блок-схеми, яка моделює перехід від однієї діяльності до іншої [14].

Діаграма нижче (Рис. 3.13) описує робочий процес створення витратного матеріалу і закріплення його за складом, який виконує менеджер з обліку витратного матеріалу. А також робочий процес створення складу, який виконує менеджер з обліку складів.

Розберемо докладніше процес роботи менеджера з обліку витратного матеріалу за кроками:

1. Початок;
2. Реєстрація витратного матеріалу: створюється об’єкт material;
3. Якщо потрібно додати ще матеріал, то знову реєстрація (крок 2);
4. Паралельно з тим відбувається закріплення матеріалу за складом;
5. Якщо потрібно додати ще закріплення, то перехід на крок 4.
6. Кінець.

Процес роботи менеджера з обліку складів за кроками можна описати наступним чином:

1. Початок;
2. Реєстрація складу: створюється об’єкт warehouse;
3. Якщо потрібно додати ще склад, то знову реєстрація (крок 2);
4. Кінець.

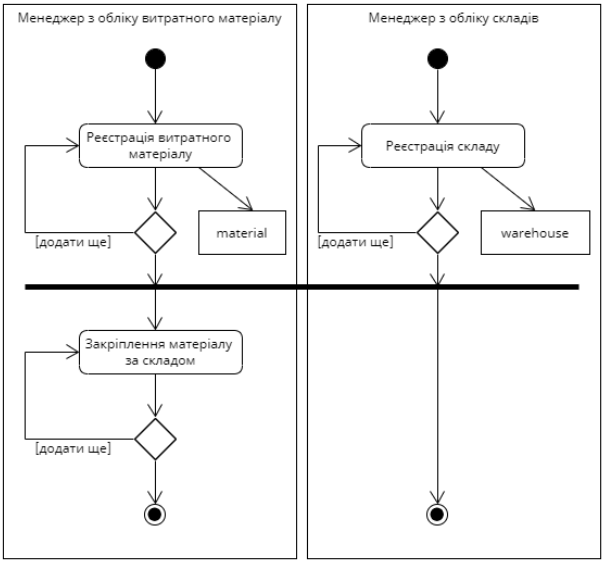


Рисунок 3.13 – Діаграма діяльності менеджерів з обліку витратного матеріалу і складів

Діяльність менеджера з обліку витратного матеріалу у коді має наступний вигляд (Рис. 3.14, 3.15):

```
IDB db;
1 reference
public EM_ManagerCRUD_Service(IDB db)
{
    this.db = db;
}
2 references
public void ExpandableMaterialCreate(string MaterialName, float Price, string UnitMeasurement)
{
    ExpendableMaterial expendableMaterial = new ExpendableMaterial(MaterialName, Price, UnitMeasurement);
    db.dbExpendableMaterial.AddItem(expendableMaterial);
}
```

Рисунок 3.14 – реєстрація витратного матеріалу

```
public List<MaterialWarehouseView> MaterialsInWarehouse()
{
    foreach (ExpendableMaterial em in db.dbExpendableMaterial.Items)
    {
        foreach (var WarehouseExpendableMaterial em_w in db.dbWarehouseExpendableMaterial.Items)
        {
            foreach (var Warehouse w in db.dbWarehouse.Items)
            {
                if (em.Id == em_w.ExpendableMaterialId && w.Id == em_w.WarehouseId)
                {
                    db.dbMaterialWarehouseView.AddItem(item: new MaterialWarehouseView(em.Id, em.MaterialName, w.Id, w.WarehouseName, em_w.Remains));
                }
            }
        }
    }
    return db.dbMaterialWarehouseView.Items;
```

Рисунок 3.15 – закріплення матеріалу за складом

Діаграма діяльності менеджерів з обліку витратного матеріалу і салонів, яка представлена на рисунку 3.16, зображує взаємодію менеджерів з питань реєстрації витратного матеріалу і салону, реєстрації матеріалу за складом, а також закріплення матеріалу за складом. Це послідовність дій двох менеджерів, результатом якої є таблиці з відповідними даними.

Розберемо докладніше процес роботи менеджера з обліку витратного матеріалу за кроками:

1. Початок;
2. Реєстрація витратного матеріалу: створюється об’єкт material;
3. Якщо потрібно додати ще матеріал, то знову реєстрація (крок 2);
4. Реєстрація матеріалу за складом: створюється об’єкт warehouseMaterial;

5. Якщо потрібно додати ще матеріал, то знову реєстрація (крок 4);

6. Паралельно з тим відбувається закріплення матеріалу за салоном;

7. Якщо потрібно додати ще закріплення, то перехід на крок 6;

8. Кінець.

Процес роботи менеджера з обліку салонів за кроками виглядає наступним чином:

1. Початок;

2. Реєстрація салону: створюється об’єкт salon;

3. Якщо потрібно додати ще салон, то знову реєстрація (крок 2);

4. Кінець.

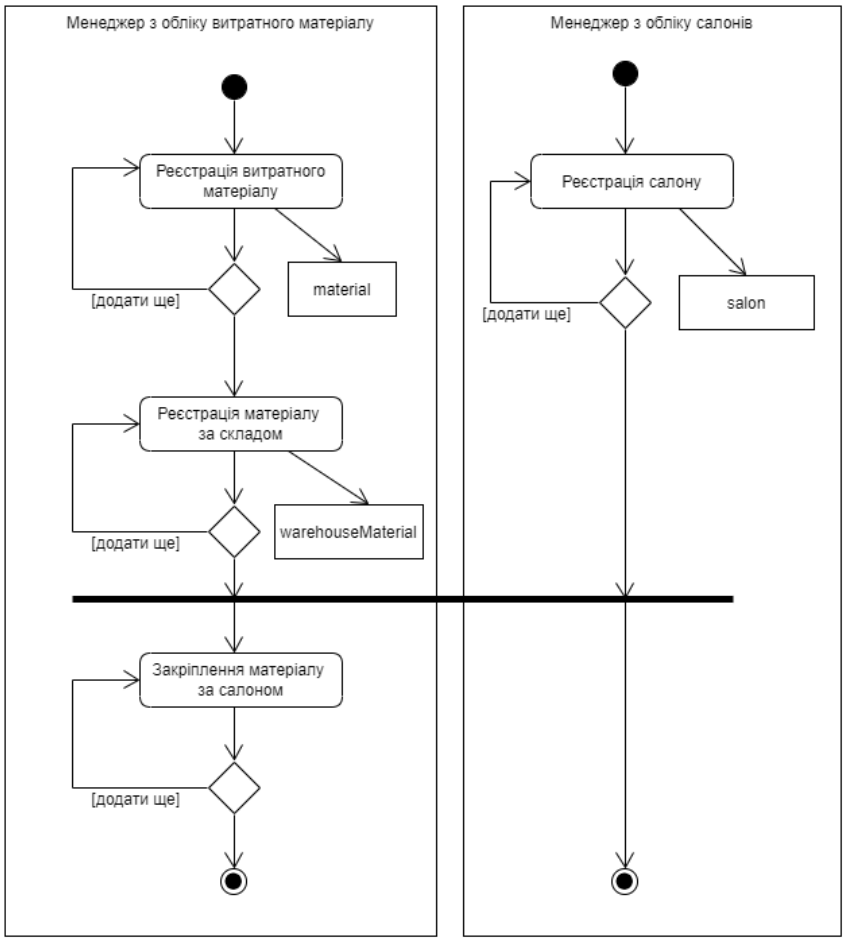


Рисунок 3.16 – Діаграма діяльності менеджерів з обліку витратного матеріалу і салонів

Реєстрація салону, якою займається менеджер з обліку салонів, створюється за допомогою коду (Рис. 3.17):

```
IDB db;
1 reference
public SalonManagerCRUD_Service(IDB db)
{
    this.db = db;
}
2 references
public void SalonCreate(string salonName)
{
    BeautySalon beautySalon = new BeautySalon(salonName);
    db.dbBeautySalon.AddItem(beautySalon);
}
```

Рисунок 3.17 – реєстрація салону

3.4.2 Проектування послідовності викликів методів та взаємодії об’єктів

Діаграма послідовності (англ. Sequence Diagram) — це діаграма, яка описує, як і в якому порядку група об’єктів працює разом. Діаграми послідовності іноді називають діаграмами подій або сценаріями подій [15].

Наведена нижче діаграма (Рис. 3.18) описує послідовний порядок взаємодій у реалізації діяльності менеджера з обліку салонів.

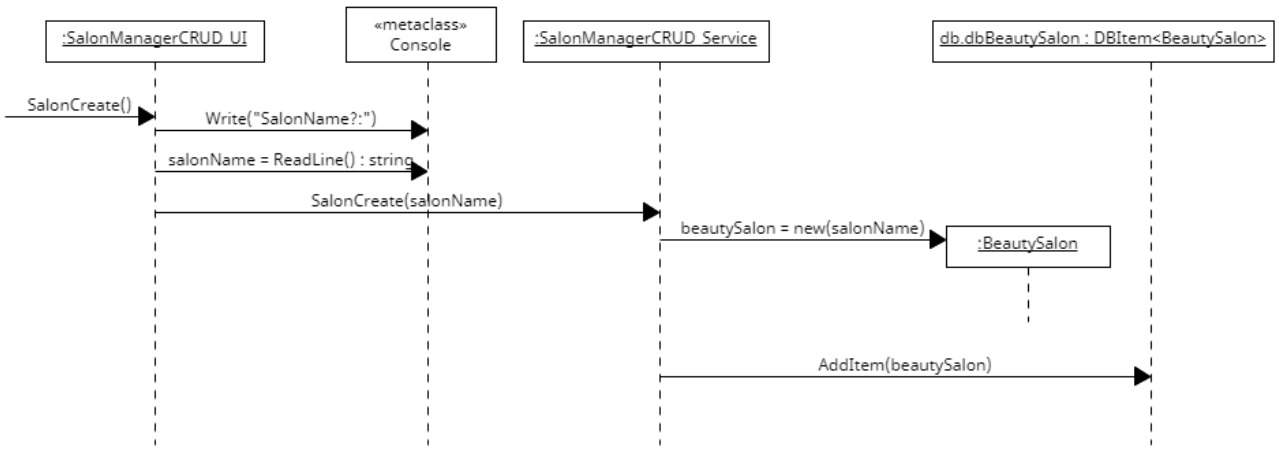


Рисунок 3.18 – Діаграма послідовності реалізації діяльності менеджера з обліку салонів

З діаграми бачимо, що коли викликається метод SalonCreate(), він відкривається в класі SalonManagerCRUD_UI, де звертається до SalonManagerCRUD_Service та виводить назву салону на екран. Цей клас також має метод SalonCreate(salonName), який створює новий салон та додає його до бази даних dbBeautySalon.

Реалізується за допомогою відповідного коду (Рис. 3.19, 3.20):

```
SalonManagerCRUD_Service salonManagerCRUD_Service;
0 references
public SalonManagerCRUD_UI(SalonManagerCRUD_Service salonManagerCRUD_Service)
{
    this.salonManagerCRUD_Service = salonManagerCRUD_Service;
}
0 references
public void SalonCreate()
{
    Console.Write(value: "SalonCreate() SalonName?: ");
    string salonName = Console.ReadLine();
    salonManagerCRUD_Service.SalonCreate(salonName);
}
```

Рисунок 3.19 – реалізація Діаграми послідовності

```
DB db;
0 references
public SalonManagerCRUD_Service(DB db)
{
    this.db = db;
}
1 reference
public void SalonCreate(string salonName)
{
    BeautySalon beautySalon = new BeautySalon(salonName);
    db.dbBeautySalon.AddItem(beautySalon);
}
```

Рисунок 3.20 – реалізація Діаграми послідовності

Результат виконання методу SalonCreate(salonName) зображений на діаграмі об’єктів (Рис. 3.21, 3.22).

До виконання метода, тобто до події (Рис. 3.21) в базі даних dbBeautySalon очікувана кількість салонів – 3. На даний момент часу в Items знаходиться 2 салони: “Purple Dreams Salon” та “Swan Salon”.

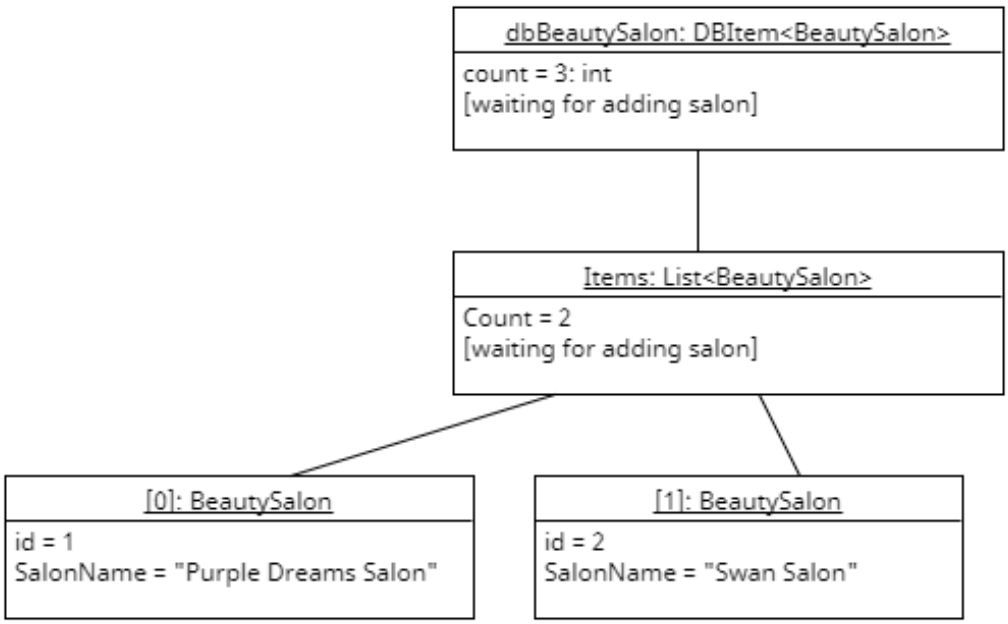


Рисунок 3.21 – до події

Після виконання метода, або після події (Рис. 3.22) в базі даних dbBeautySalon тепер очікувана кількість салонів стає 4. На даний момент часу в Items знаходяться 3 салони: “Purple Dreams Salon”, “Swan Salon” та “Blossom Salon”.

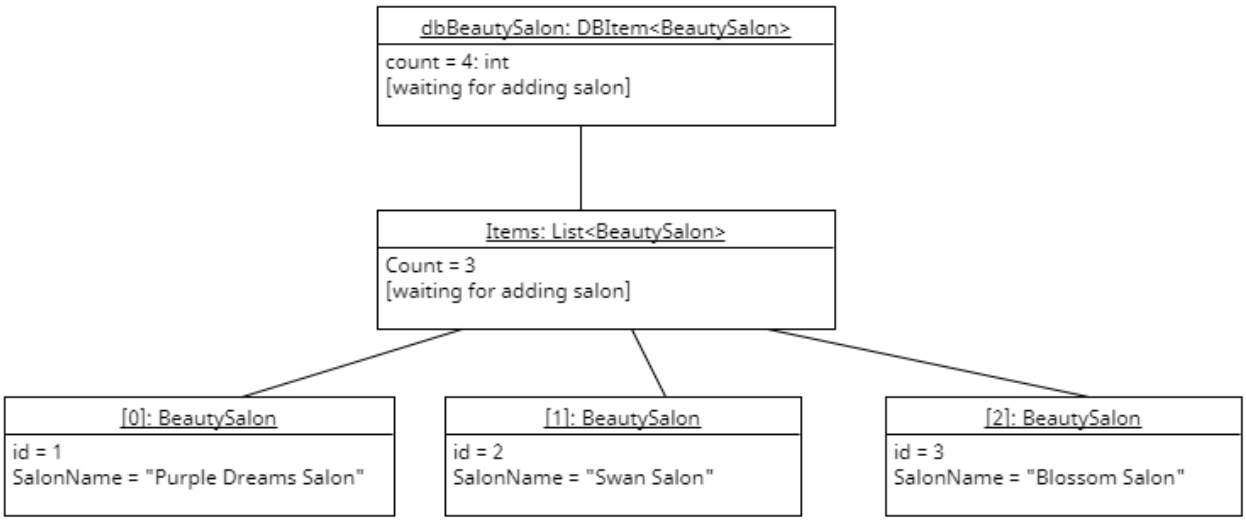


Рисунок 3.22 – після події

3.4.3 Проектування класів та їх взаємодії

Діаграма класів (англ. Class Diagram) — це діаграма, яка чітко відображає структуру певної системи шляхом моделювання її класів, атрибутів, операцій і зв’язків між об’єктами [16].

Діаграма класів головного меню (Рис. 3.23) показує, що клас MainMenu успадковується від інтерфейсу IMenu. Також клас має приватні поля em_Manager_Menu, warehouseManager_Menu, salonManager_Menu. Кожне з них теж успадковується від інтерфейсу IMenu та викликає відповідні меню. Так само MainMenu має загальнодоступний метод Run().

Далі клас EM_Manager_Menu успадковується від інтерфейсу IMenu. Має приватні поля em_CRUD_Menu, em_WarehouseCRUD_Menu, em_SalonCRUD_Menu. Кожне з них теж успадковується від інтерфейсу IMenu. Так само EM_Manager_Menu має загальнодоступний метод Run().

Аналогічно відбувається з класами WarehouseManager_Menu і SalonManager_Menu.

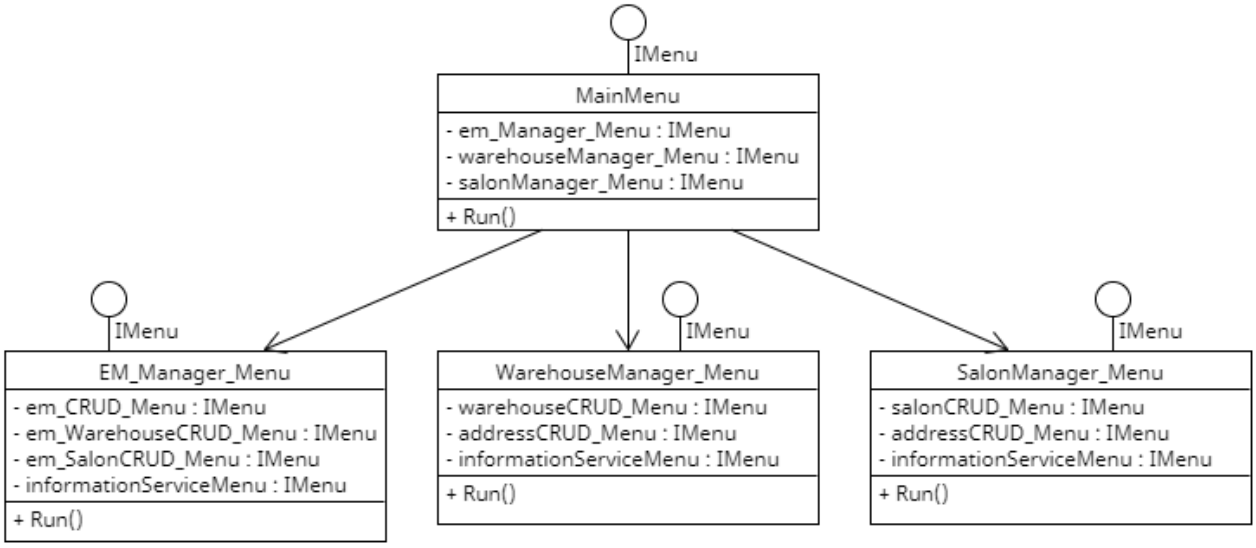


Рисунок 3.23 –Діаграма класів головного меню

У програмі реалізується за допомогою наступного коду (Рис. 3.24):

```
internal class MainMenu : IMenu
{
    IMenu em_Manager_Menu;
    IMenu salonCRUDMenu;
    1 reference
    public MainMenu(IMenu em_Manager_Menu, IMenu salonCRUDMenu)
    {
        this.em_Manager_Menu = em_Manager_Menu;
        this.salonCRUDMenu = salonCRUDMenu;
    }
    7 references
    public void Run()...
}
```

Рисунок 3.24 – реалізація класу MainMenu

На діаграмі класів меню менеджера з обліку витратного матеріалу (Рис. 3.25) можна побачити, що EM_Manager_Menu викликає класи EM_CRUD_Menu, EM_WarehouseCRUD_Menu, EM_SalonCRUD_Menu, InformationSevice_Menu, які успадковуються від інтерфейсу IMenu.

Клас EM_CRUD_Menu має приватне поле em_ManagerCRUD_UI, яке успадковується від інтерфейсу IEM_ManagerCRUD_UI та загальнодоступний метод Run().

Клас EM_WarehouseCRUD_Menu має приватне поле em_WarehouseCRUD_UI, яке успадковується від інтерфейсу IEM_WarehouseCRUD_UI та загальнодоступний метод Run().

Клас EM_SalonCRUD_Menu має приватне поле em_SalonCRUD_UI, яке успадковується від інтерфейсу IEM_SalonCRUD_UI та загальнодоступний метод Run().

Клас InformationSevice_Menu має приватне поле informationServiceCRUD_UI, яке успадковується від інтерфейсу IInformationServiceCRUD_UI та загальнодоступний метод Run().

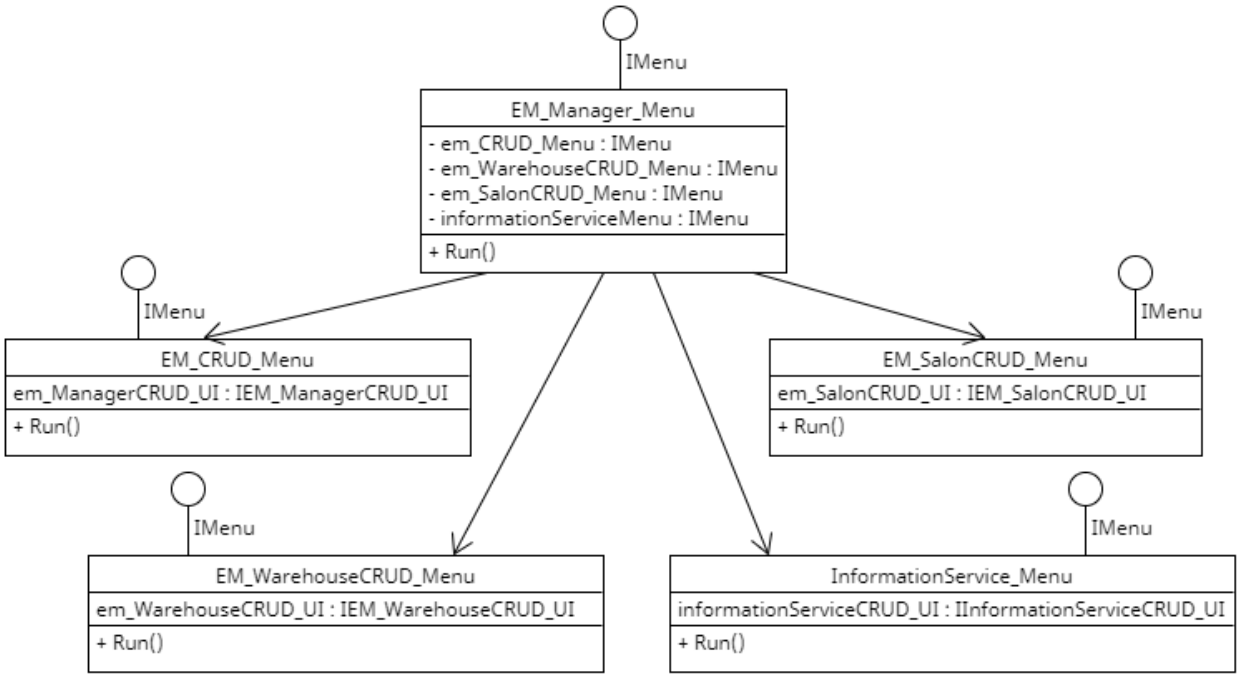


Рисунок 3.25 – Діаграма класів меню менеджера з обліку витратного матеріалу

У програмному коді має наступний вигляд (Рис. 3.26):

```
internal class EM_Manager_Menu : IMenu
{
    IMenu em_CRUD_Menu;
    IMenu em_InformationServiceMenu;
    1 reference
    public EM_Manager_Menu(IMenu em_CRUD_Menu, IMenu em_InformationServiceMenu)
    {
        this.em_CRUD_Menu = em_CRUD_Menu;
        this.em_InformationServiceMenu = em_InformationServiceMenu;
    }
    5 references
    public void Run()...
}
```

Рисунок 3.26 – реалізація класу EM_Manager_Menu

Відповідним чином робиться з меню менеджера з обліку складів (Рис. 3.27) та меню менеджера з обліку салонів (Рис. 3.28).

На діаграмі класів меню менеджера з обліку витратного складів (Рис. 3. 27) можна побачити, що Warehouse_Manager_Menu викликає класи

WarehouseCRUD_Menu, AddressCRUD_Menu та InformationSevice_Menu, які успадковуються від інтерфейсу IMenu.

Клас WarehouseCRUD_Menu має приватне поле warehouseCRUD_UI, яке успадковується від інтерфейсу IWarehouseCRUD_UI та загальнодоступний метод Run().

Клас AddressCRUD_Menu має приватне поле addressCRUD_UI, яке успадковується від інтерфейсу IAddressCRUD_UI та загальнодоступний метод Run().

Клас InformationSevice_Menu має приватне поле informationServiceCRUD_UI, яке успадковується від інтерфейсу IInformationServiceCRUD_UI та загальнодоступний метод Run().

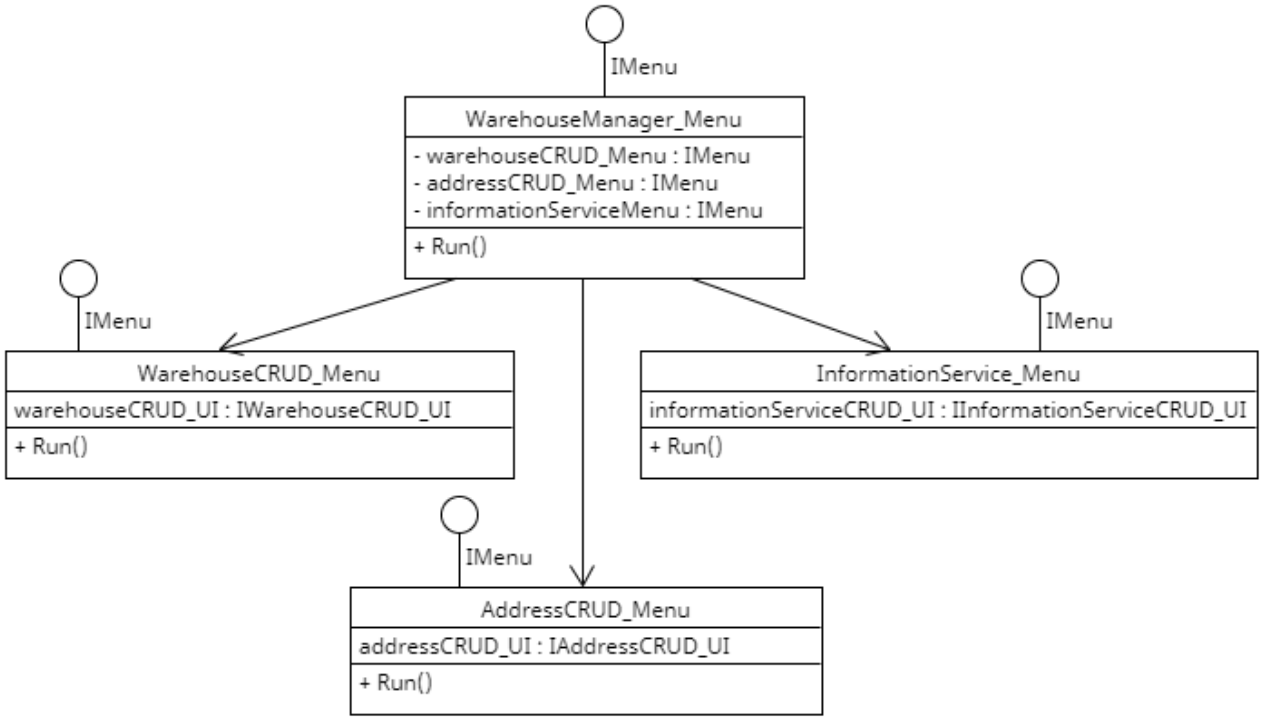


Рисунок 3.27 – Діаграма класів меню менеджера з обліку складів

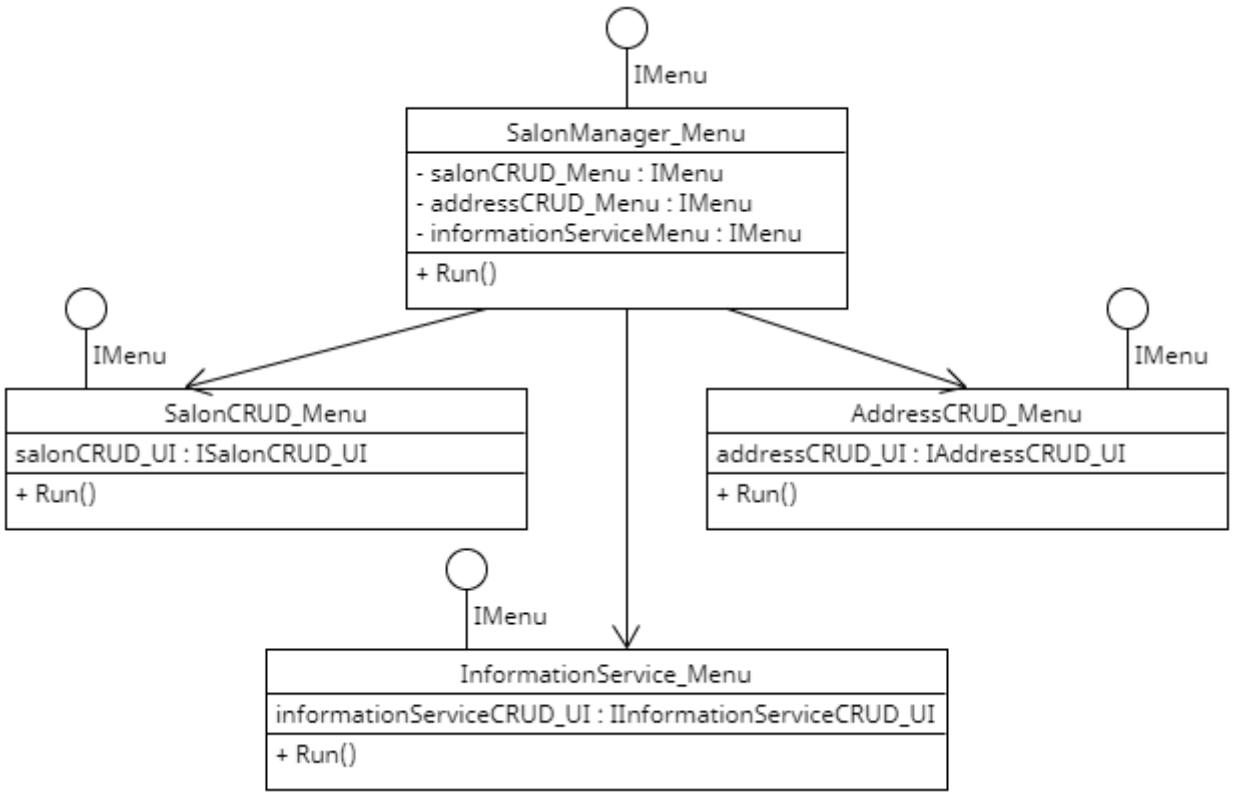


Рисунок 3.28 – Діаграма класів меню менеджера з обліку салонів

Отже, наприклад, з головного меню користувач обрав меню менеджера з обліку витратних матеріалів. Після цього перейшов до меню управління витратними матеріалами і хоче створити новий матеріал, знайти потрібний матеріал по Id або видалити обраний матеріал, тощо. Цей процес роботи описано у класі EM_Manager_CRUD.

На діаграмі класів оточення класу EM_Manager_CRUD (Рис. 3.29) представлено взаємодію класа EM_Manager_CRUD з іншими частинами системи: інтерфейсами IMenu, IID, IEM_ManagerCRUD_UI, IDB, IEM_ManagerCRUD_Service та класами EM_CRUD_Menu, ExpandableMaterial, EM_ManagerCRUD_UI, EM_ManagerCRUD_Service, DB. Також кожен елемент діаграми містить інформацію про те, які саме методи задіяні у роботі.

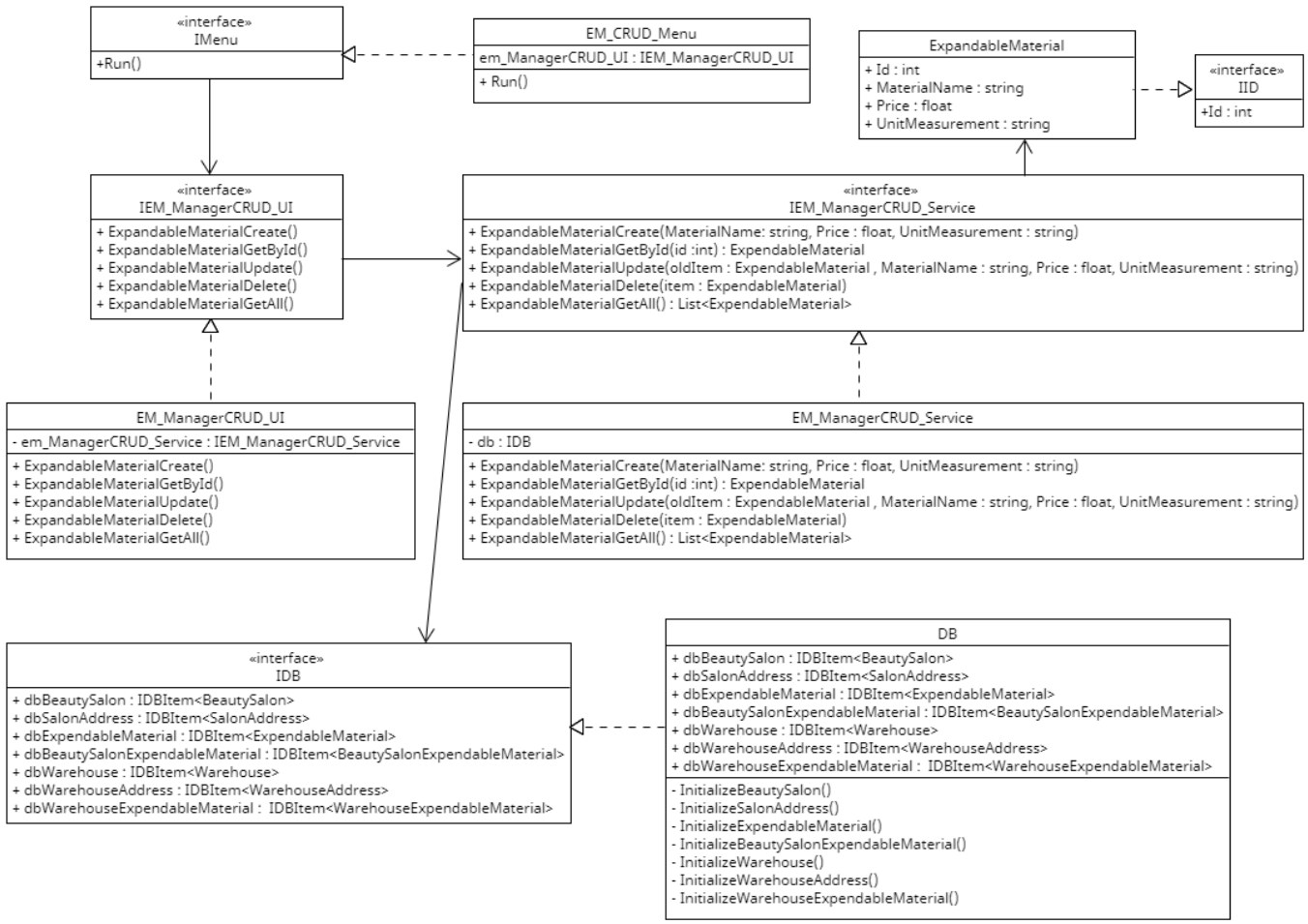


Рисунок 3.29 – Оточення класу EM_Manager_CRUD

У коді це реалізовано наступним чином (Рис. 3.30):

```
3 references
public class EM_ManagerCRUD_Service : IEM_ManagerCRUD_Service
{
    IDB db;
    1 reference
    public EM_ManagerCRUD_Service(IDB db)
    {
        this.db = db;
    }
    2 references
    public void ExpandableMaterialCreate(string MaterialName, float Price, string UnitMeasurement)
    {
        ExpendableMaterial expendableMaterial = new ExpendableMaterial(MaterialName, Price, UnitMeasurement);
        db.dbExpendableMaterial.AddItem(expendableMaterial);
    }
    4 references
    public ExpendableMaterial ExpandableMaterialGetById(int id)
    {
        ExpendableMaterial result = default(ExpendableMaterial);
        result = db.dbExpendableMaterial.GetItemById(id);
        return result;
    }
    2 references
    public bool ExpandableMaterialUpdate(ExpendableMaterial oldItem, string MaterialName, float Price, string UnitMeasurement)
    {
        ExpendableMaterial newItem = new ExpendableMaterial(MaterialName, Price, UnitMeasurement);
        return db.dbExpendableMaterial.UpdateItem(oldItem, newItem);
    }
    2 references
    public bool ExpandableMaterialDelete(ExpendableMaterial item)
    {
        return db.dbExpendableMaterial.DeleteItem(item);
    }
    2 references
    public List<ExpendableMaterial> ExpandableMaterialGetAll()
    {
        return db.dbExpendableMaterial.Items;
    }
}
```

Рисунок 3.30 – реалізація класу EM_Manager_CRUD

На рисунку 3.31 зображено взаємодію інтерфейсів IDB, IDBItem<T>, IEM_ManagerCRUD_Service і класу DBItem<T>. За допомогою них здійснюється управління даними та додавання їх до бази даних.

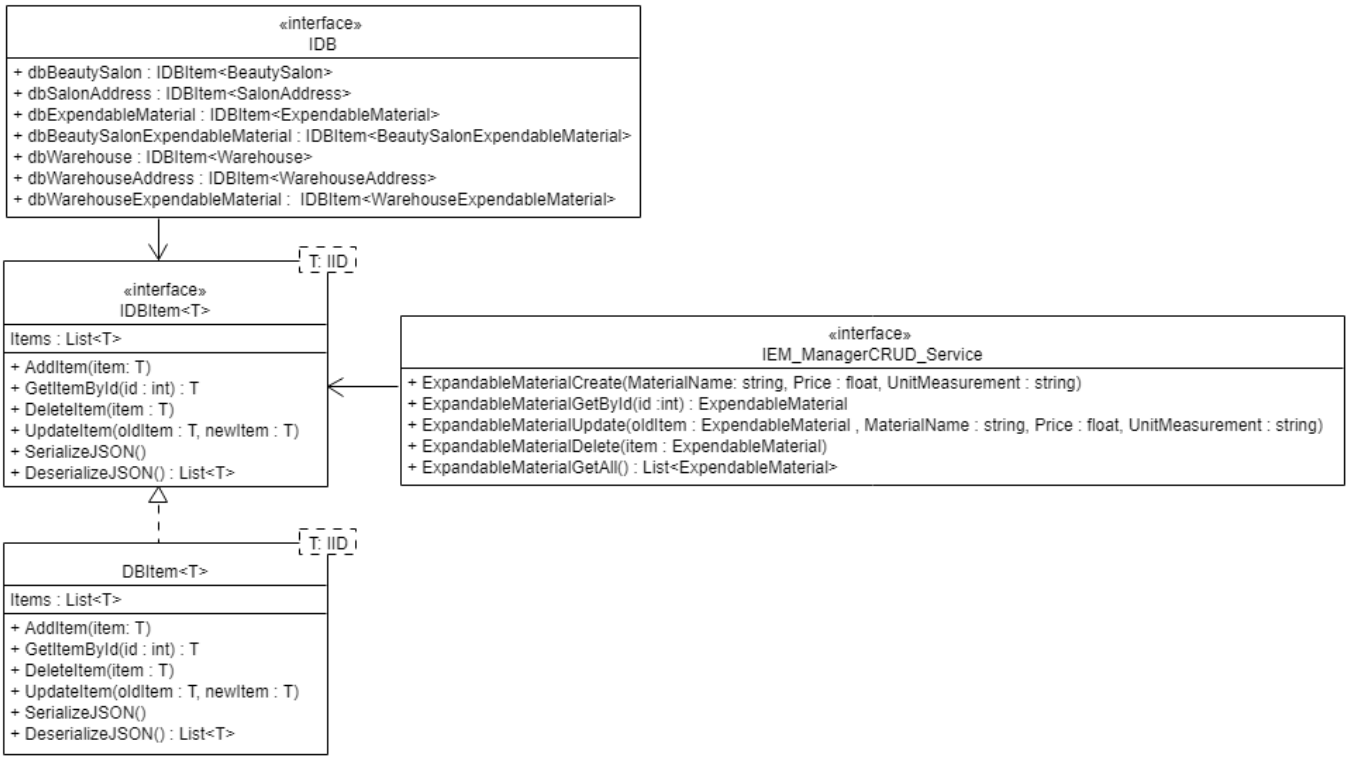


Рисунок 3.31 – Оточення класу DBItem<T>

Оточення класу DBItem<T> реалізується за допомогою такого коду (Рис. 3.32):

```
34 references
public class DBItem<T> : IDBItem<T> where T : IId
{
    private int count = 1;
    32 references | 5/5 passing
    public List<T> Items { get; set; }
    21 references | 9/9 passing
    public DBItem()
    {
        this.Items = new List<T>();
    }
    61 references | 9/9 passing
    public void AddItem(T item)...
    5 references | 1/1 passing
    public T GetItemById(int id)...
    4 references | 1/1 passing
    public bool DeleteItem(T item)...
    5 references | 3/3 passing
    public bool UpdateItem(T oldItem, T newItem)...
    3 references | 2/2 passing
    public bool SerializeJSON()...
    2 references | 1/1 passing
    public List<T> DeserializeJSON()...
}
```

Рисунок 3.32 – реалізація класу DBItem<T>

3.4.4 Проектування станів

Діаграма станів (англ. State Diagram) — це діаграма, яка підкреслює впорядковану подіями поведінку об’єкта. Вона складається зі станів, переходів, подій і дій. Використовується для ілюстрації динамічного вигляду системи. Діаграма станів особливо важлива для моделювання поведінки інтерфейсу [17].

Приведена нижче діаграма станів меню (Рис. 3.33) показує процес переходу від основного меню до меню менеджера з обліку витратних матеріалів, меню менеджера з обліку складів та меню менеджера з обліку салонів. Ці три меню також діляться на підменю. З кожного меню є вихід на рівень вище.

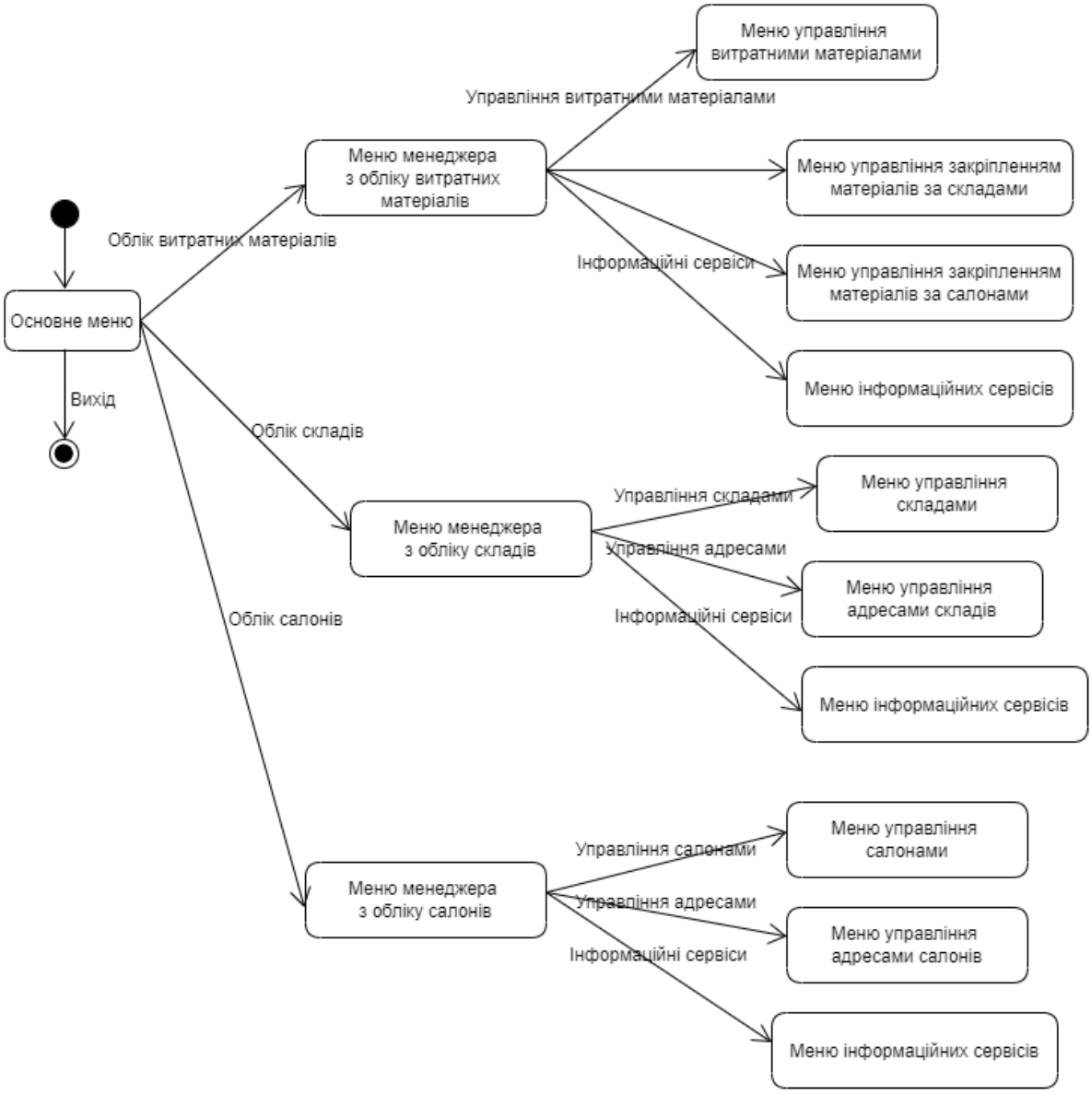


Рисунок 3.33 – Діаграма станів меню

На прикладі меню менеджера з обліку витратних матеріалів (Рис. 3.34) видно, що воно ділиться на:

1. Меню управління витратними матеріалами:
    1.1. Реєстрація витратного матеріалу;
    1.2. Пошук витратного матеріалу по Id;
    1.3. Заміна даних витратного матеріалу;
    1.4. Видалення витратного матеріалу;
    1.5. Перегляд витратних матеріалів.
2. Меню управління закріпленням матеріалів за складами;
3. Меню управління закріпленням матеріалів за салонами;
4. Меню інформаційних сервісів:
    4.1. Інформація щодо розміщення матеріалів на складах;
    4.2. Інформація щодо розміщення матеріалів у салонах.

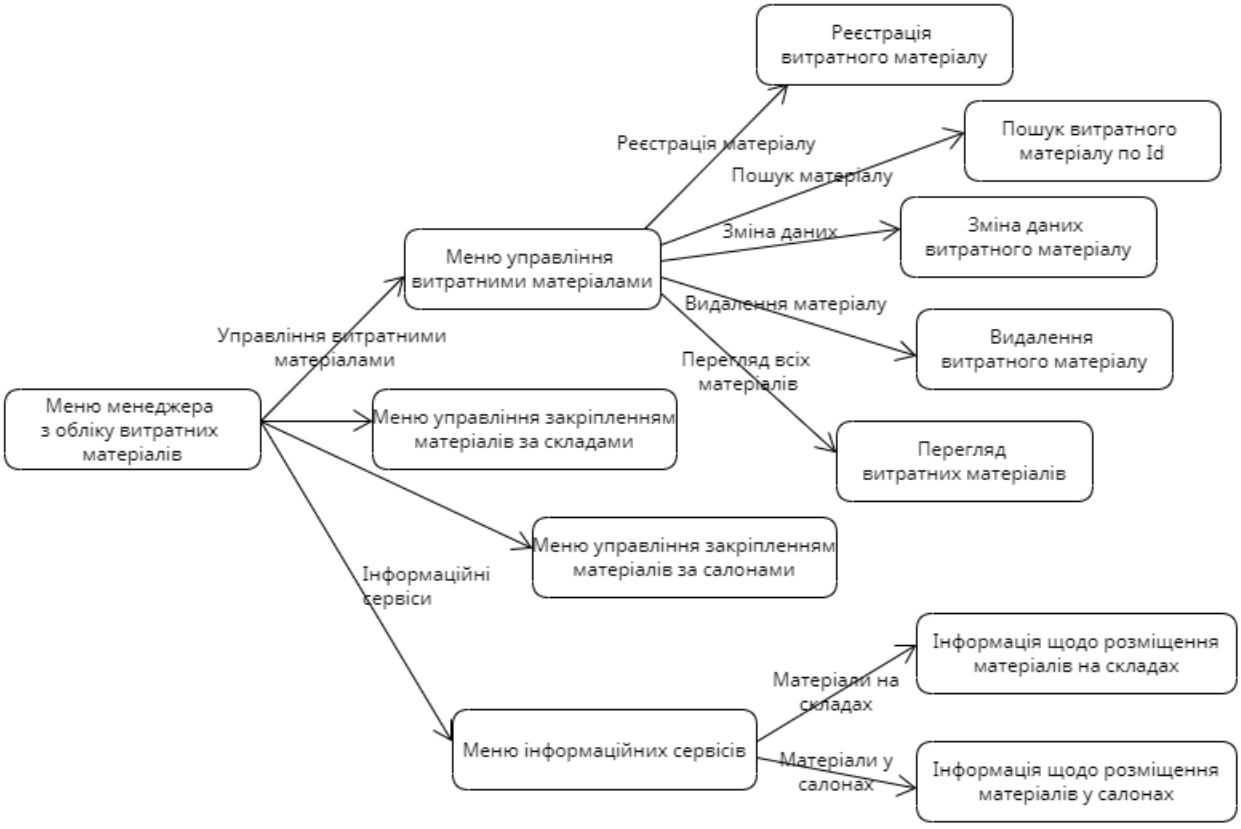


Рисунок 3.34 – Діаграма станів меню менеджера з обліку витратних матеріалів

Реалізується за допомогою такого коду (Рис. 3.35 – 3.37):

```
bool flag = true;
while (flag)
{
    Console.Clear();
    Console.WriteLine(value: "1 - Меню управління витратними матеріалами");
    Console.WriteLine(value: "2 - Меню управління закріпленням матеріалів за складами");
    Console.WriteLine(value: "3 - Меню управління закріпленням матеріалів за салонами");
    Console.WriteLine(value: "4 - Меню інформаційних сервісів");
    Console.WriteLine(value: "5 - Вихід");
    int choice = int.Parse(Console.ReadLine());
    switch (choice)
    {
        case 1:
            Console.WriteLine(value: "Меню управління витратними матеріалами");
            em_CRUD_Menu.Run();
            break;
        case 2:
            Console.WriteLine(value: "Меню управління закріпленням матеріалів за складами");
            Console.ReadKey();
            break;
        case 3:
            Console.WriteLine(value: "Меню управління закріпленням матеріалів за салонами");
            Console.ReadKey();
            break;
        case 4:
            Console.WriteLine(value: "Меню інформаційних сервісів");
            em_InformationServiceMenu.Run();
            break;
        case 5:
            Console.WriteLine(value: "Вихід");
            Console.ReadKey();
            flag = false;
            break;
```

Рисунок 3.35 – реалізація Меню менеджера з обліку витратних матеріалів

```
bool flag = true;
while (flag)
{
    Console.Clear();
    Console.WriteLine(value: "1 - Реєстрація витратного матеріалу");
    Console.WriteLine(value: "2 - Пошук витратного матеріалу по Id");
    Console.WriteLine(value: "3 - Зміна даних витратного матеріалу");
    Console.WriteLine(value: "4 - Видалення витратного матеріалу");
    Console.WriteLine(value: "5 - Перегляд витратних матеріалів");
    Console.WriteLine(value: "6 - Вихід");
    int choice = int.Parse(Console.ReadLine());
    switch (choice)
    {
        case 1:
            Console.WriteLine(value: "Реєстрація витратного матеріалу");
            em_ManagerCRUD_UI.ExpandableMaterialCreate();
            Console.ReadKey();
            break;
        case 2:
            Console.WriteLine(value: "Пошук витратного матеріалу по Id");
            em_ManagerCRUD_UI.ExpandableMaterialGetById();
            Console.ReadKey();
            break;
        case 3:
            Console.WriteLine(value: "Зміна даних витратного матеріалу");
            em_ManagerCRUD_UI.ExpandableMaterialUpdate();
            Console.ReadKey();
            break;
        case 4:
            Console.WriteLine(value: "Видалення витратного матеріалу");
            em_ManagerCRUD_UI.ExpandableMaterialDelete();
            Console.ReadKey();
            break;
        case 5:
            Console.WriteLine(value: "Перегляд витратних матеріалів");
            em_ManagerCRUD_UI.ExpandableMaterialGetAll();
            Console.ReadKey();
            break;
        case 6:
            Console.WriteLine(value: "Вихід");
            Console.ReadKey();
            flag = false;
            break;
```

Рисунок 3.36 – реалізація Меню управління витратними матеріалами

```
IEM_InformationService_UI em_InformationService_UI;
1 reference
public EM_InformationServiceMenu(IEM_InformationService_UI em_InformationService_UI)
{
    this.em_InformationService_UI = em_InformationService_UI;
}
5 references
public void Run()
{
    bool flag = true;
    while (flag)
    {
        Console.Clear();
        Console.WriteLine(value: "1 - Інформація щодо розміщення матеріалів на складах");
        Console.WriteLine(value: "2 - Інформація щодо розміщення матеріалів у салонах");
        Console.WriteLine(value: "3 - Вихід");
        int choice = int.Parse(Console.ReadLine());
        switch (choice)
        {
            case 1:
                Console.WriteLine(value: "Інформація щодо розміщення матеріалів на складах");
                em_InformationService_UI.MaterialsInWarehouse();
                Console.ReadKey();
                break;
            case 2:
                Console.WriteLine(value: "Інформація щодо розміщення матеріалів у салонах");
                em_InformationService_UI.MaterialsInSalon();
                Console.ReadKey();
                break;
            case 3:
                Console.WriteLine(value: "Вихід");
                Console.ReadKey();
                flag = false;
                break;
```

Рисунок 3.37 – реалізація Меню інформаційних сервісів

3.5 Архітектура

3.5.1 Діаграма пакетів

Діаграма пакетів (англ. Package Diagram) — це різновид структурної діаграми, що показує розташування та організацію елементів моделі в середньому та великому проекті. Діаграма може показувати як структуру, так і залежності між підсистемами або модулями, вказуючи різні уявлення про систему [18].

Архітектура програми була спроектована на трьох рівнях: UI Level, Business Logic Level, Data Level (Рис. 3.38). Стрілка «access» означає, що пакет «UI Level» вимагає допомоги від функцій пакету «Business Logic Level», а «Business Logic Level» – від функцій пакету «Data Level».

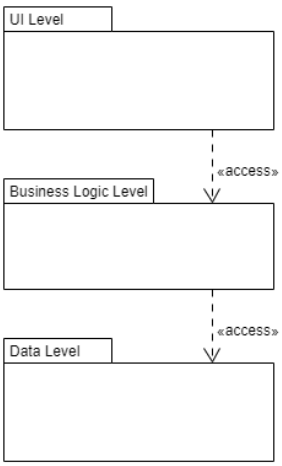


Рисунок 3.38 – трирівнева архітектура програми

Пакет «UI Level» складається з таких підрівнів (Рис. 3.39):

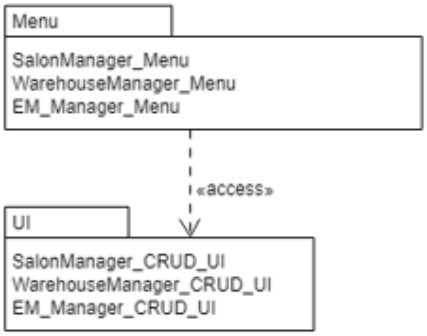


Рисунок 3.39 – структура UI Level

Пакет «Business Logic Level» складається з (Рис. 3.40):

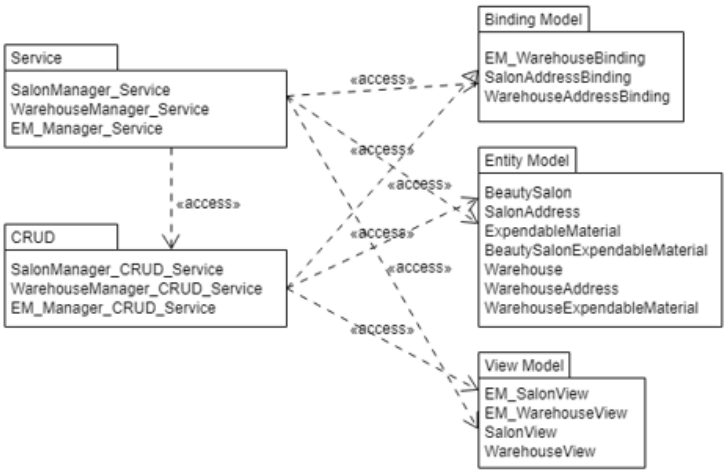


Рисунок 3.40 – структура Business Logic Level

У програмному коді діаграма має такий вигляд (Рис. 3.41, 3.42):

```
5 references
internal class SalonManagerCRUD_Service
{
    DB db;
    1 reference
    public SalonManagerCRUD_Service(DB db)
    {
        this.db = db;
    }
    1 reference
    public void SalonCreate(string SalonName)
    {
        BeautySalon beautySalon = new BeautySalon(SalonName);
        db.dbBeautySalon.AddItem(beautySalon);
    }
```

Рисунок 3.41 – реалізація класу SalonManagerCRUD_Service

```
internal class SalonManagerCRUD_UI
{
    SalonManagerCRUD_Service salonManagerCRUD_Service;
    1 reference
    public SalonManagerCRUD_UI(SalonManagerCRUD_Service salonManagerCRUD_Service)
    {
        this.salonManagerCRUD_Service = salonManagerCRUD_Service;
    }
    1 reference
    public void SalonCreate()
    {
        Console.Write(value: "SalonCreate() SalonName?: ");
        string salonName = Console.ReadLine();
        salonManagerCRUD_Service.SalonCreate(salonName);
    }
```

Рисунок 3.42 – реалізація класу SalonManagerCRUD_UI

Пакет «Data Level» складається з (Рис. 3.43):

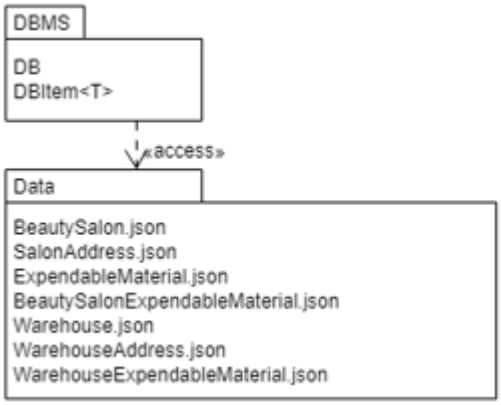


Рисунок 3.43 – структура Business Data Level

У програмному коді діаграма має такий вигляд (Рис. 3.44):

```
22 references
public class DBItem<T> where T : IId
{
    private int count = 1;
    public List<T> Items;
    7 references
    public DBItem()
    {
        this.Items = new List<T>();
    }
    32 references
    public void AddItem(T item)
    {
        item.Id = count++;
        Items.Add(item);
    }
```

Рисунок 3.44 – реалізація класу DBItem

Також у класі Program (Рис. 3.45):

```
DB db=new DB();
SalonManagerCRUD_Service salonManagerCRUD_Service = new SalonManagerCRUD_Service(db);
SalonManagerCRUD_UI salonManagerCRUD_UI = new SalonManagerCRUD_UI(salonManagerCRUD_Service);
salonManagerCRUD_UI.SalonCreate();
foreach (BeautySalon beautySalon in dbBeautySalon.Items)
{
    Console.WriteLine(beautySalon);
}
```

Рисунок 3.45 – реалізація класу Program

У програмі пакети виглядають так (Рис. 3.46):

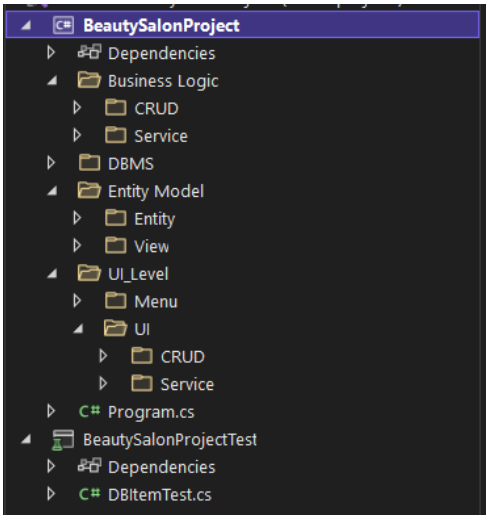


Рисунок 3.46 – вигляд пакетів в проекті

3.5.2 Діаграма артефактів

Діаграма артефактів (англ. Deployment Diagram) — це діаграма, яка показує конфігурацію вузлів обробки під час виконання та компонентів, які на них працюють. Вона часто використовується для моделювання статичного вигляду розгортання системи (топології апаратного забезпечення) [19].

Діаграма артефактів приведена нижче на рисунку 3.47.

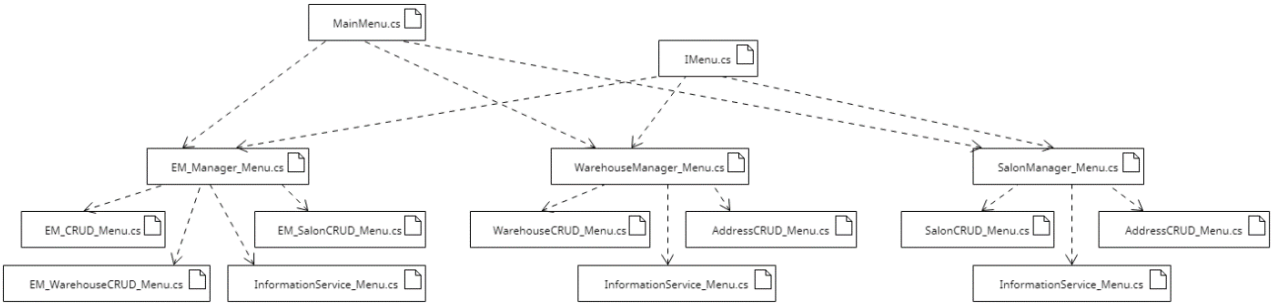


Рисунок 3.47 – Діаграма артефактів меню

У програмі виглядає наступним чином (Рис. 3.48):

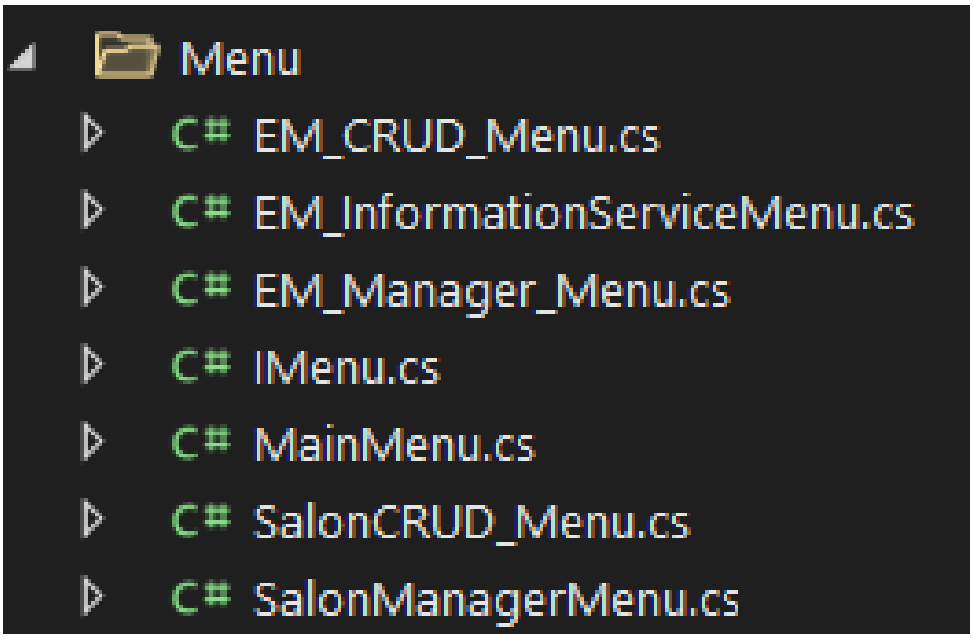


Рисунок 3.48 – Діаграма артефактів меню

ТЕСТУВАННЯ СИСТЕМИ

Тестування програмного забезпечення — це процес оцінки та перевірки того, що програмний продукт або додаток виконує те, що він повинен робити. Переваги тестування включають запобігання помилкам, зниження витрат на розробку та підвищення продуктивності [20].

Протягом усього життєвого циклу ПЗ проводиться тестування на різних рівнях.

Модульне тестування (англ. Unit testing) є базовим підходом до тестування програмного забезпечення для перевірки окремого блоку коду.

Інтеграційне тестування (англ. Integration testing) зосереджено на побудові та дизайні програмного забезпечення. Тестувальник повинен переконатися, що вбудовані блоки працюють без помилок.

Тестування системи (англ. System testing), у якому програмне забезпечення компілюється, а потім тестується як єдине ціле [21].

Також розрізняють ручне і автоматичне тестування. При ручному тестуванні людина (QA-тестувальник) виконує тести крок за кроком, без тестових сценаріїв. При автоматичному тестуванні тести виконуються автоматично через фреймворки разом з іншими інструментами та програмним забезпеченням [22].

У даному проекті виконано автоматичне тестування.

На рисунку 3.49 зображена діаграма, з якої видно, що тестування проводилося над класом DBItem<T>. Клас DBItemTest має методи, що перевіряють:

1. AddItemToItems(): чи додається створений об’єкт до бази даних;
2. AddItem_Add1ToId(): чи додається одиниця до Id;
3. GetItem_GetById(): чи можна по Id дістати потрібний об’єкт;
4. DeleteItemTest(): чи видаляється потрібний елемент;
5. UpdateItem_True(): чи змінюється вибраний елемент правильно;
6. UpdateItem_False();
7. UpdateItem_OldListAndNewList_AreEquivalent(): чи змінюється обраний об’єкт в колекції;

8. SerializeJSON_OldListAndNewList_AreEquivalent(): чи відбувається серіалізація даних в JSON;

9. DeserializeJSON_OldListAndNewList_AreEquivalent(): чи відбувається десеріалізація даних із JSON.

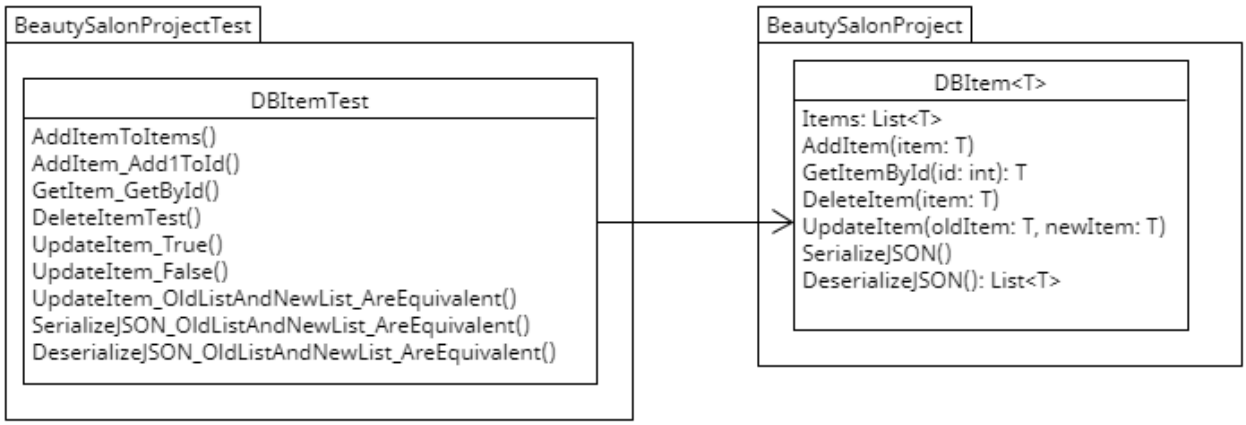


Рисунок 3.49 – модульне тестування

Результати тестування представлені нижче (Рис. 3.50):

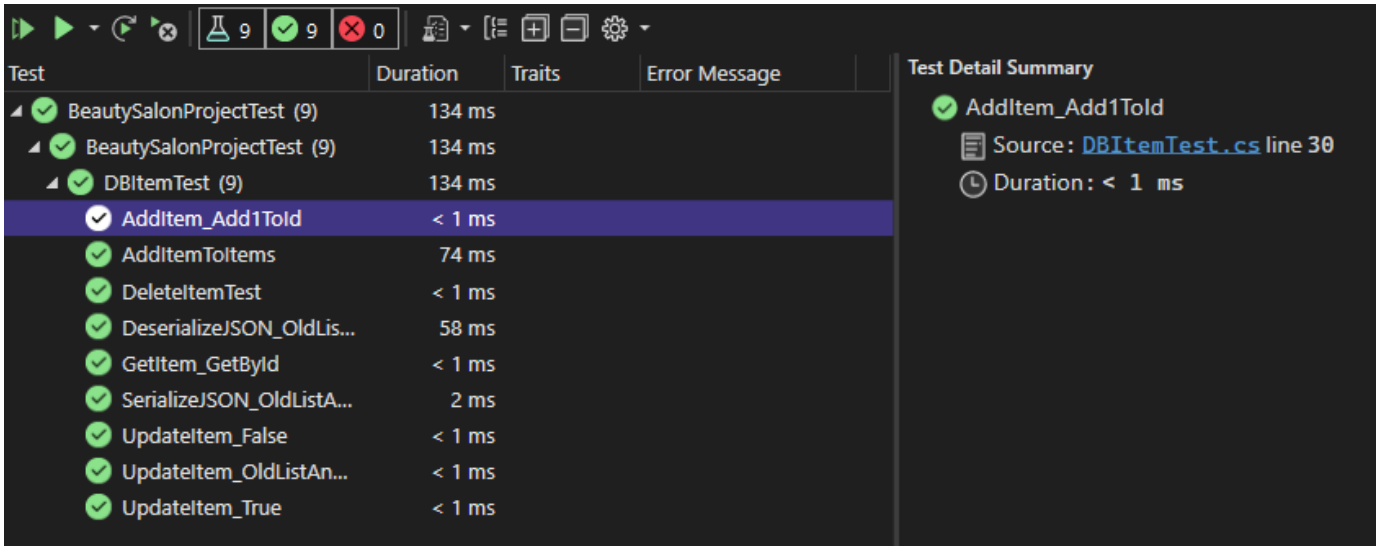


Рисунок 3.50 – результати тестування

ВИСНОВКИ

У результаті виконання роботи була розроблена інформаційна система для обліку витратного матеріалу. У процесі розробки цієї системи були досягнуті всі поставлені цілі.

1. Створена модель предметної галузі.

2. Створена модель прецедентів.

3. Проаналізовано нефункціональні вимоги в Додатковій специфікації.

4. Опрацьовано документ Бачення з визначенням місця і ролі інформаційної системи, що розробляється в загальній моделі процесу обліку витратних матеріалів, як складової моделі процесу обліку матеріалів з використанням інформаційних технологій.

5. Виконано проектування діяльності.

6. Виконано проектування послідовності викликів методів.

7. Виконано проектування класів та їх взаємодії.

8. Виконано проектування станів.

9. Опрацьована архітектура програми.

10. Реалізовано тестування системи. Результати тестування успішні, всі завдання обліку втілені без помилок.

ПЕРЕЛІК ПОСИЛАНЬ

1 Організація обліку процесу реалізації готової продукції [Електронний ресурс] – Режим доступу до ресурсу: http://www.investplan.com.ua/?op=1&z=6434&i=5.

2 Бухгалтерський облік в умовах цифрової економіки [Електронний ресурс] – Режим доступу до ресурсу: http://ibo.wunu.edu.ua/index.php/ibo/article/view/405.

3 ТОП-10 програм для обліку товару та складу [Електронний ресурс] – Режим доступу до ресурсу: https://a4.com.ua/10-program-obliku-tovaru-ta-skladu/.

4 Программа для складского учета – ERP FOSS [Електронний ресурс] – Режим доступу до ресурсу: https://erp.foss.ua/programa-dlya-skladskogo-obliku-v-malomu-i-serednomu-biznesi/.

5 IBS Торгівля і Склад [Електронний ресурс] – Режим доступу до ресурсу: http://www.ibsystems.com.ua/ua/ibs-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0-%D1%81%D0%BA%D0%BB%D0%B0%D0%B4%D1%81%D1%8C%D0%BA%D0%BE%D0%B3%D0%BE-%D0%BE%D0%B1%D0%BB%D1%96%D0%BA%D1%83/%D0%BE%D0%BF%D0%B8%D1%81.

6 UMLet - Free UML Tools for fast UML diagrams [Електронний ресурс] – Режим доступу до ресурсу: https://www.umlet.com/.

7 What Is C# Language, Advantages & Features Of C# Language [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: ttps://www.codexoxo.com/advantages-c-sharp-language/.

8 What is an IDE? [Електронний ресурс] – Режим доступу до ресурсу: https://www.redhat.com/en/topics/middleware/what-is-ide.

9 Introduction to Visual Studio [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/introduction-to-visual-studio/.

10 What is .NET Framework? Explain Architecture & Components [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/net-framework.html.

11 Context diagram | IST Project Management Office - University [Електронний ресурс] – Режим доступу до ресурсу: https://uwaterloo.ca/ist-project-management-office/tools-and-templates/tools/context-diagram#:~:text=Context%20diagrams%20show%20the%20interactions,system%20will%20be%20part%20of.

12 What is an Entity Relationship Diagram (ERD)? – Lucidchart [Електронний ресурс] – Режим доступу до ресурсу: https://www.lucidchart.com/pages/er-diagrams.

13 Use-case diagrams [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case.

14 What is Activity Diagram? - Visual Paradigm [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/.

15 UML Sequence Diagram Tutorial | Lucidchart [Електронний ресурс] – Режим доступу до ресурсу: https://www.lucidchart.com/pages/uml-sequence-diagram.

16 UML Class Diagram Tutorial | Lucidchart [Електронний ресурс] – Режим доступу до ресурсу: https://www.lucidchart.com/pages/uml-class-diagram.

17 All You Need to Know about State Diagrams - Visual Paradigm [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/about-state-diagrams/.

18 What is Package Diagram? - Visual Paradigm [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/.

19 What is Deployment Diagram? - Visual Paradigm [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/.

20 What is Software Testing and How Does it Work? | IBM [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/topics/software-testing.

21 What is Software Testing? Definition, Basics & Types [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/software-testing-introduction-importance.html.

22 Manual Testing vs. Automation Testing | Which Is Better? [Електронний ресурс] – Режим доступу до ресурсу: https://www.perfecto.io/blog/automated-testing-vs-manual-testing-vs-continuous-testing.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО- НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ОБЛІКУ ВИТРАТНОГО МАТЕРІАЛУ САЛОНІВ КРАСИ З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ C#

Виконала студентка 4 курсу
групиПД-42
Єздакова Анастасія Сергіївна
Керівник роботи:
Гаманюк Ігор Михайлович

Київ – 2022

МЕТА, ОБ ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – створення невимогливого до технічних ресурсів модулю, який виконує функції складського обліку та може працювати з іншими системами чи модулями підприємства.
- **Об’єкт дослідження** – облік витратних матеріалів.
- **Предмет дослідження** – програмне забезпечення з обліку витратних матеріалів.

АКТУАЛЬНІСТЬ РОБОТИ

- Актуальність даної роботи заключається в тому, що розроблений застосунок допомагає ефективніше працювати менеджерам з обліку та автоматизувати частину їхнього робочого процесу. Залучення інформаційних технологій до складського обліку дозволить надійно зберігати дані, зробить їх відображення більш зручним, прискорить пошук, кардинально підвищить швидкість обробки та точність результатів .
- Розроблена система обліку витратних матеріалів дозволяє оновляти базу даних та реалізовує вхідну інформацію , що є детальною і становить основу для наступної логічної та арифметичної обробки даних.
- Впровадження створеної інформаційної системи обліку матеріалів дозволить отримати достовірну та своєчасну інформацію стосовно обліку наявності косметичної продукції на складі.

АНАЛОГИ

ПОРІВНЯННЯ АНАЛОГІВ

Назва додатку	Переваги	Недоліки
BAS Комплексне управління підприємством	1. розроблена на платформі Business Automation Framework; 2. дозволяє "вмикати" або "вимикати" функціональні частини прикладного рішення без програмування; 3. організація роботи з системою через Інтернет, в режимі веб-клієнта.	Не вистачає взаємодії з іншими системами
IBS Торгівля і Склад	1. підтримує замовлення постачальників , надходження , повернення , переміщення товарів між складами; 2. надає інформацію про списання (пошкодження).	
ERP FOSS	1. хмарна система управління бізнесом, бухгалтерією та податками; 2. складські операції; 3. інтеграція з Новою Поштою, Приват24 та Prom.ua.	

ТЕХНІЧН І ЗАВДАННЯ

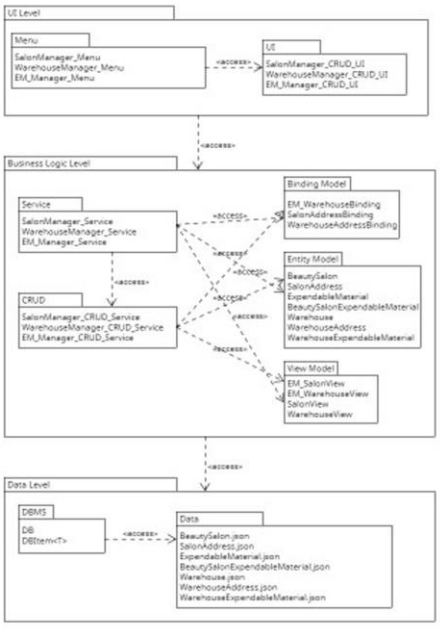
1. Опрацювати модель предметної галузі.
2. Опрацювати модель прецедентів.
3. Сформулювати нефункціональні вимоги.
4. Сформулювати документ Бачення з визначенням місця і ролі інформаційної системи, що розробляється.
5. Виконати проектування діяльності.
6. Виконати проектування послідовності викликів методів.
7. Виконати проектування класів та їх взаємодії.
8. Виконати проектування станів.
9. Опрацювати архітектуру програми.
10. Реалізувати програмне забезпечення.
11. Здійснити автоматичне тестування.

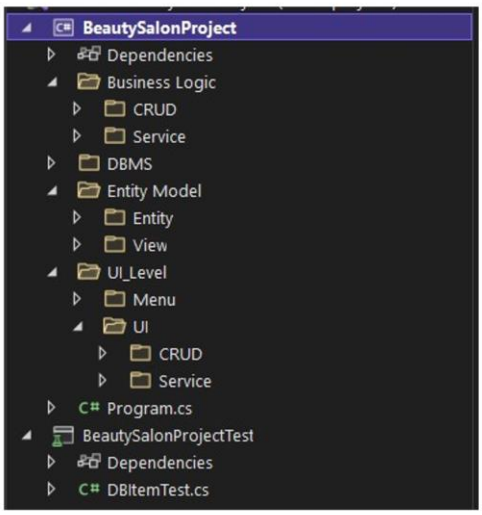
ІНСТРУМЕНТИ, ВИКОРИСТАНІ ДЛЯ РЕАЛІЗАЦІЇ

- Онлайн-редактор UML-діаграм UMLetino
- Інтегроване середовище розробки Visual Studio
- .Net Framework
- Мова програмування C#
- Файли формату JSON

АРХІТЕКТУРА ПРОГРАМИ

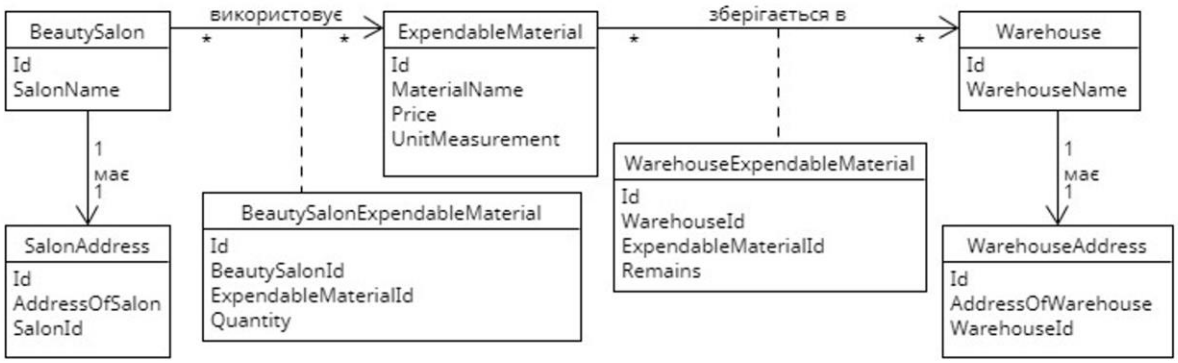
Діаграма пакетів





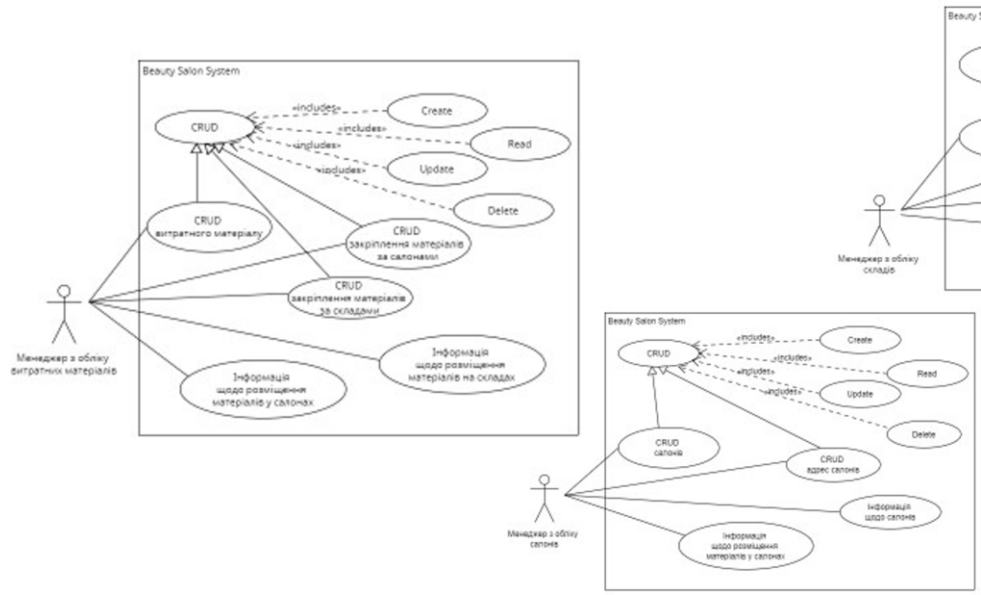
МОДЕЛЬ ПРЕДМЕТНОЇ ГАЛУЗІ

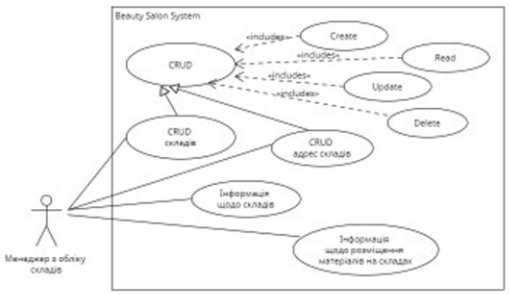
Діаграма предметної галузі



ФУНКЦІОНАЛЬНІСТЬ СИСТЕМИ

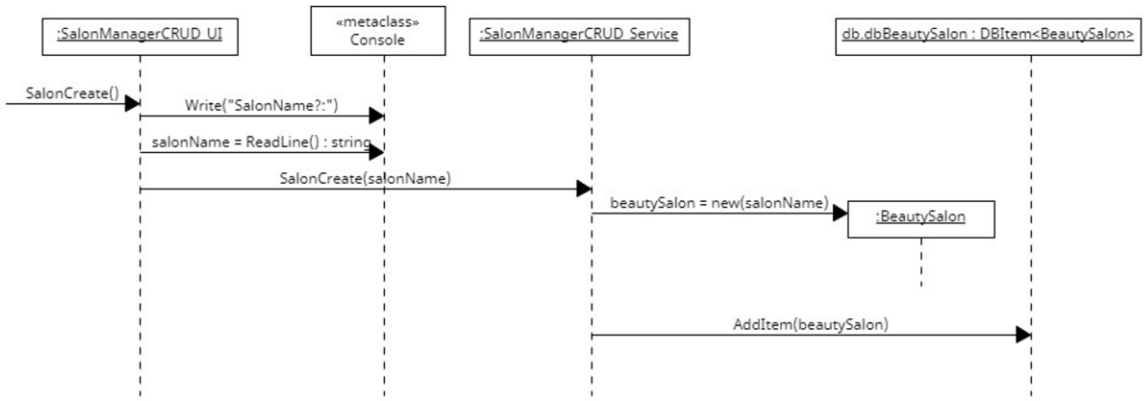
Діаграми прецедентів





ВЗАЄМОДІЯ ОБ’ЄКТІВ

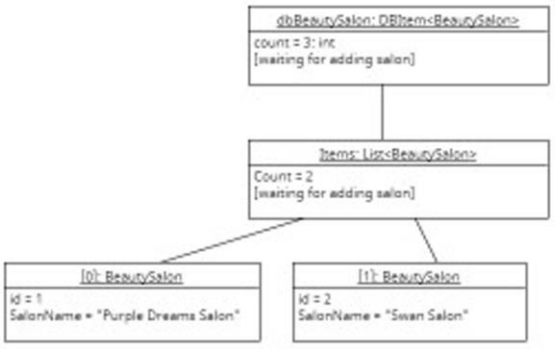
Діаграма послідовності реалізації метода реєстрації салону SalonCreate()



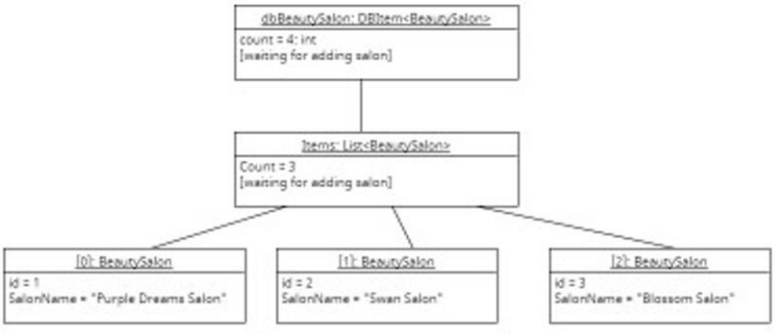
ВЗАЄМОДІЯ ОБ’ЄКТІВ

Результат діяльності методу SalonCreate(salonName) показано на діаграмі об'єктів

До події

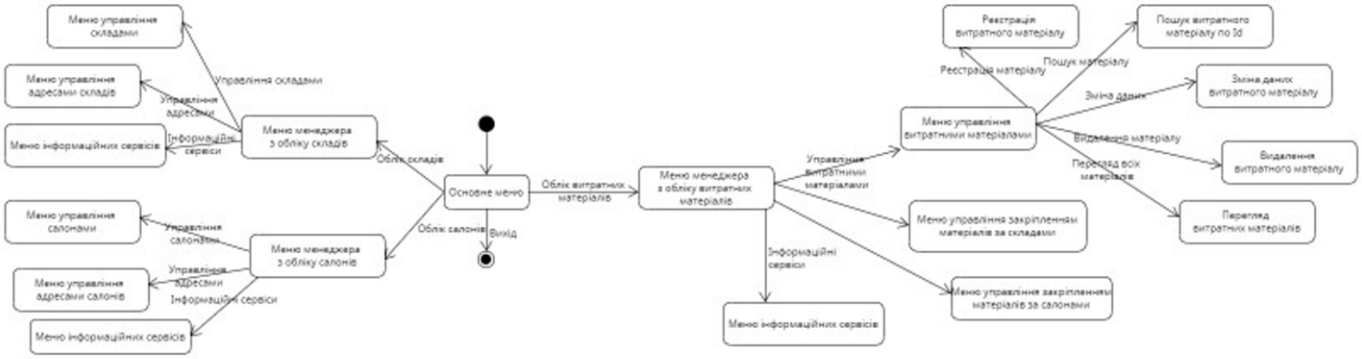


Після події



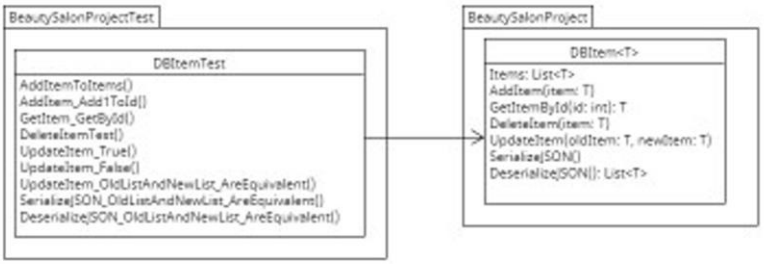
МОДЕЛЮВАННЯ МЕНЮ

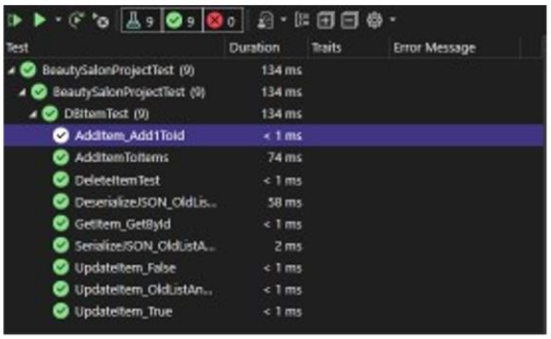
Діаграма станів меню



ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Здійснено автоматичне тестування системи. Усі тести виконані успішно.





АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Єздакова А.С. Використання електронних таблиць для відображення предметної галузі обліку витратного матеріалу салонів краси / А.С. Єздакова, І.М. Гаманюк // Застосування програмного забезпечення в інфокомунікаційних технологіях: Матеріали науково -технічної конференції. Збірник тез. 20.04.2022, ДУТ, м. Київ — К.: ДУТ, 2022. — С. 22 – 24.
2. Єздакова А.С. Використання UML-діаграми пакетів для проектування логічних рівнів інформаційної системи з обліку витратного матеріалу салонів краси / А.С. Єздакова, І.М. Гаманюк // Застосування програмного забезпечення в інфокомунікаційних технологіях: Матеріали науково - технічної конференції. Збірник тез. 20.04.2022, ДУТ, м. Київ — К.: ДУТ, 2022. — С. 48 – 50.

ВИСНОВКИ

Створено інформаційну систему з обліку витратних матеріалів, що підтверджує досягнення мети дипломного проекту. У процесі розробки цієї системи були досягнуті всі поставлені цілі.

1. Створена модель предметної галузі.
2. Створена модель прецедентів.
3. Проаналізовано нефункціональні вимоги в Додатковій специфікації.
4. Опрацьовано документ Бачення з визначенням місця і ролі інформаційної системи, що розробляється в загальній моделі процесу обліку витратних матеріалів, як складової моделі процесу обліку матеріалів з використанням інформаційних технологій.
5. Виконано проектування діяльності.
6. Виконано проектування послідовності викликів методів.
7. Виконано проектування класів та їх взаємодії.
7. Виконано проектування станів.
8. Опрацьована архітектура програми.
9. Реалізовано тестування системи. Результати тестування позитивні, всі завдання обліку втілені без помилок.

Перспективи подальших досліджень

Застосунок можна розвивати и здійснювати облік не тільки витратних матеріалів, а й облік персоналу та клієнтів.

ДЯКУЮ ЗА УВАГУ!