

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ АДМІНІСТРУВАННЯ
ЕЛЕКТРОННОЇ КОМЕРЦІЇ НА ПРИКЛАДІ ІНТЕРНЕТ ПРОДАЖІВ
НАРУЧНИХ ГОДИННИКІВ
SOFTWARE IMPLEMENTATION OF THE E-COMMERCE
ADMINISTRATION SYSTEM BY THE EXAMPLE OF
ONLINE SALES OF WATCHES

Виконав: студент 4 курсу, групи КНТ-121

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

БЕЙНИК В.А.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЕОЩЕНКО С.Д.

(ПРИЗВИЩЕ та ініціали)

Рецензент БАБЕНКО Н.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

БЕЙНИКА Владислава Анатолійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація системи адміністрування електронної комерції на прикладі інтернет продажів наручних годинників
Software Implementation of the E-commerce Administration System by the Example of Online Sales of Watches

керівник проєкту (роботи) д-р. філософ., доцент, ЛЕОЩЕНКО Сергій Дмитрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 5 травня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ЛЕОЩЕНКО С.Д., доц.		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викл.		

7. Дата видачі завдання “ 28 ” квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент

(підпис) Владислав БЕЙНИК
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Сергій ЛЕОЩЕНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 153 с., 44 рис., 31 табл., 4 дод., 16 джерел.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, АДМІНІСТРУВАННЯ, ЕЛЕКТРОННА КОМЕРЦІЯ, НАРУЧНІ ГОДИННИКИ, ІНВЕСТИЦІЯ, СТАТИСТИКА ПРОДАЖІВ, БАЗА ДАНИХ, ГРАФІЧНИЙ ІНТЕРФЕЙС.

Об'єкт дослідження – процес розроблення програмного забезпечення для адміністрування електронної комерції на прикладі інтернет-продажів наручних годинників.

Предмет дослідження – методи та інструменти реалізації взаємодії з базою даних, реалізація графічного інтерфейсу користувача, аналіз статистики продажів та зміни тенденції курсу ціни на дорогоцінні метали, з яких виготовлено наручні годинники.

Мета роботи – розробка програмного забезпечення, яке буде ефективним інструментом для керування наручними годинниками, замовленнями, аналізу продажів та курсу вартості матеріалів з яких виготовлений годинник, швидке та зручне оформлення замовлення зі сторони клієнтів.

Матеріали, методи та технічні засоби – для розробки програмного забезпечення було здійснено мовою C# на платформі .NET Framework 4.8.1 з використанням технології WPF для розробки інтуїтивно зрозумілого графічного інтерфейсу користувача. Середовище розробки – Microsoft Visual Studio Community 2022. Дані системи зберігаються у базі даних СКБД Microsoft SQL Server. Для взаємодії з базою даних використано Entity Framework. Для створення об'єктної моделі було використано метод Database First, що генерує класи на основі існуючої бази даних. Було використано

фреймворк Live Charts для створення діаграм на основі отриманих даних. Операційна система – Windows 10.

Результати – розроблене програмне забезпечення, яке автоматизує адміністрування електронної комерції інтернет-продажів наручних годинників, забезпечує зручну та ефективну взаємодію користувачів з приємним та зрозумілим графічним інтерфейсом, також програма надає функціонал для аналізу статистики продажів годинників та зміни курсу матеріалів для визначення вигідності для клієнту сервісу інвестувати у наручний годинник.

Отже, усі цілі дипломної роботи було виконано, у результаті розроблено якісне програмне забезпечення, що реалізовує ефективне адміністрування електронної комерції інтернет-продажів наручних годинників.

ABSTRACT

Explanatory note to the bachelor's diploma qualification work: 153 p., 44 fig., 31 tab., 4 append., 16 sources.

SOFTWARE, ADMINISTRATION, ELECTRONIC COMMERCE, WATCHES, INVESTMENT, SALES STATISTICS, DATABASE, GRAPHICAL INTERFACE.

The object of the study is software implementation of the e-commerce administration system by the example of online sales of watches.

The subject of the study is methods and tools for implementing interaction with the database, implementing a graphical user interface, analyzing sales statistics and changes in the price trend of precious metals from which wristwatches are made.

The purpose of the work is to develop software that will be an effective tool for managing wristwatches, orders, analyzing sales and the cost of materials from which the watch is made, and quick and convenient order processing by customers.

Materials, methods and technical means - the software was developed in the C# language on the .NET Framework 4.8.1 platform using WPF technology to develop an intuitive graphical user interface. The development environment is Microsoft Visual Studio Community 2022. The system data is stored in the Microsoft SQL Server database. The Entity Framework was used to interact with the database. The Database First method was used to create an object model, which generates classes based on an existing database. The Live Charts framework was used to create charts based on the data obtained. Operating system – Windows 10.

Results – developed software that automates the administration of e-commerce of online sales of wristwatches, provides convenient and effective user interaction with a pleasant and understandable graphical interface, the program also provides functionality for analyzing statistics of sales of watches and changes in the

exchange rate of materials to determine the profitability for the service client to invest in a wristwatch.

Thus, all the objectives of the thesis were met, and as a result, high-quality software was developed that implements effective administration of e-commerce of online sales of wristwatches.

ЗМІСТ

Перелік умовних позначень, скорочень і термінів	12
Вступ.....	13
1 Аналіз предметної області.....	15
1.1 Огляд предметної області	15
1.1.1 Сегментація ринку	16
1.1.2 Вплив смарт-годинників	16
1.1.3 Популярні бренди та виробники	16
1.1.4 Висновок з огляду ринку годинників в Україні	17
1.2 Вимоги до системи адміністрування електронної комерції	18
1.3 Існуючі програмні рішення.....	19
1.3.1 Rozetka.....	19
1.3.2 Allo.ua.....	20
1.3.3 Deka.ua	21
1.4 Необхідність розробки програмного забезпечення.....	24
1.5 Вимоги до розроблюваного програмного забезпечення.....	25
1.6 Висновки з розділу	27
2 Матеріали і методи.....	28
2.1 Огляд особливостей обраної мови програмування	28
2.2 Огляд особливостей обраного середовища розробки	31
2.3 Огляд особливостей обраної системи управління базою даних	33
2.4 Огляд особливостей технології розробки графічного інтерфейсу	35
2.5 Огляд особливостей Entity Framework для роботи з базою даних	36
2.6 Огляд особливості підходу Database First для роботи з базою даних	37
2.7 Схема розробленої бази даних.....	39
2.8 Висновки з розділу.....	40
3 Опис програми.....	42
3.1 Алгоритм функціонування розробленої програми.....	42
3.1.1 Авторизація та реєстрація користувача у системі.....	43
3.1.2 Додавання нового годинника до системи	45
3.1.3 Алгоритм додавання угоди про купівлю годинника.....	45

3.1.4 Алгоритм для підтвердження адміністратором продажу годинника	46
3.1.5 Алгоритм розрахунку особистої статистики продажів адміністратора	47
3.1.6 Алгоритм перегляду історії продажів та покупок з додаванням пагінації у список	49
3.1.7 Алгоритм обробки інформації з json-файлу для відображення зміни на протязі 10 років вартості матеріалу	50
3.1.8 Алгоритм розрахунку коефіцієнту приросту ціни матеріалу, для поради користувачеві чи варто інвестувати у цей матеріал	51
3.1.9 Алгоритм додавання годинника до списку «Обраного»	53
3.2 Вибір шаблону проектування	55
3.3 Опис модулів розроблюваної системи	57
3.3.1 Клас WatchDataBaseEntities	58
3.3.2 Клас DataBaseManage	59
3.3.3 Клас User	62
3.3.4 Клас Admin	63
3.3.5 Клас Wristwatch	64
3.3.6 Клас Agreement	65
3.3.7 Клас Material	66
3.3.8 Клас Discount	67
3.3.9 Клас AdditionalServices	67
3.3.10 Клас UserManage	68
3.3.11 Клас WristwatchManage	69
3.3.12 Клас AgreementManage	70
3.3.13 Клас CatalogManage	72
3.3.14 Клас MaterialManage	74
3.3.15 Клас DiscountManage	74
3.3.16 Клас AdditionalServiceManage	75
3.3.17 Клас LikedWristwatchManage	76
3.3.18 Клас SalesDiagramManage	76
3.3.19 Клас MaterialsStatisticManage	77
3.4 Проектування бази даних	78
3.4.1 Таблиця User	78

	10
3.4.2 Таблиця Admin	79
3.4.3 Таблиця Wristwatch.....	80
3.4.4 Таблиця LikedWatch	80
3.4.5 Таблиця Material.....	81
3.4.6 Таблиця Discount.....	81
3.4.7 Таблиця Agreement	82
3.4.8 Таблиця AdditionalServices	82
3.5 Висновки з розділу.....	83
4 Експлуатація, тестування та експериментальне дослідження програми	84
4.1 Призначення програмного забезпечення.....	84
4.2 Умови виконання програми	85
4.2.1 Апаратні вимоги програми	85
4.2.2 Вимоги до користувача	86
4.3 Виконання програми.....	86
4.3.1 Запуск програми.....	86
4.3.2 Виконання роботи програми	89
4.4 Повідомлення користувачу	106
4.5 Опис методики проведення тестування.....	108
4.6 Висновки з розділу.....	109
Висновки	111
Перелік джерел посилання	113
Додаток А Технічне завдання	115
А.1 Вступ.....	116
А.1.1 Найменування проєкту	116
А.1.2 Мета розробки та її призначення.....	116
А.1.3 Підстава для розробки програми.....	117
А.2 Основні вимоги до програми	117
А.2.1 Вимоги до функціональних характеристик.....	117
А.2.2 Вимоги до графічного інтерфейсу	118
А.2.3 Вимоги до безпеки програми.....	118
А.2.4 Вимоги до технічних та програмних засобів	119
Додаток Б Приклад формування курсу вартості матеріалів.....	121

	11
Б.1 Вміст Json-файлу з курсом матеріалів	122
Додаток В Текст програми	124
В.1 Код Wristwatch.cs.....	125
В.2 Код DataBaseManage.cs.....	125
В.3 Код AgreementManage.cs	132
В.4 Код SalesDiagramManage.cs.....	137
В.5 Код MaterialsStatisticManage.cs	139
Додаток Г Слайди презентації	143

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ООП – об’єктно-орієнтоване програмування;

СКБД – система керування базами даних;

EF – Entity Framework;

IDE – Integrated Development Environment;

JSON – JavaScript Object Notation;

LINQ – Language Integrated Query;

MVVM – Model-View-ViewModel;

SQL – Structured Query Language;

UI – User interface;

WPF – Windows Presentation Foundation;

XAML – eXtensible Application Markup Language.

ВСТУП

Розвиток цифрових технологій у наш час та стрімке зростання популярності електронної комерції вплинули на спосіб ведення бізнесу у сфері продажів. Онлайн-продажі стали важливою частиною сучасного ринку, адже вони надають покупцям можливість зручно та швидко обирати та купувати товари, а бізнесу – ефективно керувати обліком продажів, обробляти замовлення та комунікувати з клієнтами. Для забезпечення ефективної роботи торговельного бізнесу необхідне якісне програмне забезпечення, яке автоматизує адміністративні процеси та забезпечить зручний інтерфейс для взаємодії як для менеджерів магазину, так і для клієнтів.

Даний дипломний проєкт передбачає розробку програмного додатку для адміністрування електронної комерції на прикладі інтернет продажів наручних годинників. Основним завданням розроблюваної системи є створення ефективного інструменту для керування товарами, замовленнями, аналізу продажів та тенденції вартості матеріалів з яких виготовлений годинник, швидке та зручне оформлення замовлення зі сторони клієнтів.

Програмна реалізація буде здійснена мовою C# з використанням технології WPF для розробки інтуїтивно зрозумілого графічного інтерфейсу користувача [1]. Дані системи зберігатимуться у базі даних, яка працюватиме на сервері Microsoft SQL Server. Архітектура розроблюваної системи передбачає два типи користувачів: адміністраторів магазину (менеджерів) та клієнтів.

Менеджери матимуть можливість:

- додавати, редагувати та видаляти товари;
- керувати їхніми характеристиками (опис, фото, матеріал, ціна);
- підтверджувати замовлення клієнтів;
- переглядати аналітичні дані щодо продажів у вигляді діаграм;
- переглядати статистику власних продажів;

- переглядати історію продажів наручних годинників.

Клієнти зможуть:

- реєструватися у системі;
- переглядати годинники у зручному каталозі;
- додавати обрані товари у список «Обране»;
- надсилати замовлення на покупку;
- обирати додаткові послуги до замовлення;
- переглядати курс вартості матеріалу за останні 10 років.

Окремою функціональною можливістю буде реалізація модуля аналізу вартості дорогоцінного металу, з якого виготовлено механічні годинники, що дозволить точніше оцінювати тенденцію зміни їх ринкової ціни, з метою можливості придбання годинника як інвестицію.

Розробка даної програми дозволить автоматизувати процес адміністрування інтернет-магазину наручних годинників, покращити зручність використання для користувачів, підвищити ефективність управління продажами та забезпечити кращу взаємодію між адміністраторами та клієнтами. Впровадження таких систем є важливим кроком для розвитку електронної комерції, адже воно сприяє покращенню якості обслуговування продажів та зростанню ефективності роботи інтернет-магазинів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Електронна комерція є однією з найбільш перспективною галуззю сучасної торгівельної економіки, яка активно розвивається завдяки впровадженню новітніх інформаційних технологій. Онлайн-продажі дозволяють спростити та пришвидшити процес купівлі-продажу товарів, автоматизувати керування замовленнями та забезпечити гнучку та універсальну систему адміністрування бізнесу.

Однією з популярних та стабільних категорій товарів на ринку електронної комерції є наручні годинники, особливо механічні моделі з дорогих металів, що мають високий попит серед колекціонерів, інвесторів, або просто любителів точності та естетики. Продаж таких вартісних товарів вимагає детального підходу до організації онлайн-торгівлі, оскільки необхідно враховувати широкий асортимент продукції, характеристики матеріалів з яких виготовлено годинник (у тому числі дорогоцінних металів), популярність брендів на ринку та особливості взаємодії з клієнтами.

У цьому розділі буде проведено огляд предметної області, що охоплює особливості ринку наручних годинників, основні вимоги до системи адміністрування електронної комерції, існуючі програмні рішення у цій сфері та їхні особливості. Також буде визначено ключові функціональні вимоги до розроблюваної програми, що дозволять створити ефективний інструмент для керування інтернет продажами наручних годинників.

1.1 Огляд предметної області

Механічні наручні годинники вважаються важливим аксесуаром, поєднуючи функціональність та естетику. В Україні ринок наручних годинників демонструє динамічний розвиток, відображаючи як світові тенденції, так і локальні особливості обігу продукції на ринку.

1.1.1 Сегментація ринку

Ринок наручних годинників в Україні можна розділити на три сегменти [2].

Низький та середній ціновий сегмент: у цьому сегменті переважають моделі масового виробництва, доступні широкому колу споживачів [2].

Високий ціновий сегмент (преміум): цей сегмент орієнтований на преміальні та люксові моделі годинників, які підкреслюють статус власника. Згідно з дослідженням Київського національного університету імені Тараса Шевченка, пандемія COVID-19 позитивно вплинула на розвиток цього сегмента через великий попит, що спричинив стрімке зростання обсягів продажу та підвищення цін на преміум-годинники [2].

1.1.2 Вплив смарт-годинників

Поява смарт-годинників вплинула на продажі традиційних механічних моделей, особливо в нижньому та середньому цінових сегментах. У 2015 році ринок традиційних годинників в Україні зазнав помітного спаду (у середньому на 20%), тоді як інтерес до смарт-годинників зріс більш ніж удвічі [3]. Однак у преміальному та люксовому сегментах конкуренція між смарт-годинниками та класичними моделями менш відчутна, через те, що смарт-годинники є менш цікавими пропозиціями [3].

1.1.3 Популярні бренди та виробники

Серед популярних брендів наручних годинників в Україні можна виділити як міжнародні, так і вітчизняні марки.

Міжнародні бренди:

- «Casio»: японський бренд, відомий своєю надійністю та широким асортиментом моделей [4];

- Daniel Klein: турецький бренд, що поєднав класику та стиль у відносно недорогому сегменті [4];
- Q&Q: дочірня компанія корпорації Citizen представляє великий асортимент моделей та доступність цін [4];
- Orient: дешева механіка та широкий вибір кварцових моделей [4];
- Timex: перевірений часом американський виробник, що надає асортимент як класичних так і спортивних моделей наручних годинників [4].

Вітчизняні виробники:

- Kleynod: найстаріший український виробник наручних годинників, який використовує швейцарські механізми із трирічною гарантією. В асортименті представлені колекції із національною символікою [5];
- Slava: відроджена 2012 року марка годинників, що використовує японські механізми Miyota. Представляє класичний український годинник у доступному ціновому сегменті [5];
- ZIZ: молодіжний бренд з яскравим дизайном годинників. ZIZ заснований у 2008 році та орієнтований на молоду вікову категорію клієнтів. Відрізняється яскравими авторськими принтами на циферблатах та продається вже у 150 країнах світу [5];
- AndyWatch: експерт у персоналізації зовнішнього вигляду годинників. З 2009 року спеціалізується на оригінальних ремінцях та можливості нанесення власного дизайну на циферблат годинника [5].

1.1.4 Висновок з огляду ринку годинників в Україні

Ринок наручних годинників в Україні характеризується різноманітністю сегментів та суттєвим впливом сучасних технологій. Сегментація ринку вказує на різні вподобання споживачів залежно від

цінового діапазону. Вплив смарт-годинників відчутний у низькому та середньому ціновому сегменті, тоді як преміум-моделі мають перевагу у високому ціновому сегменті ринку [2], [3]. Вітчизняні виробники, сприяють розвитку національного годинникового виробництва, пропонуючи споживачам якісні та унікальні продукти наручних механічних годинників [5].

1.2 Вимоги до системи адміністрування електронної комерції

Система адміністрування електронної комерції повинна відповідати загальним вимогам, які забезпечують її ефективність, безпеку, зручність використання та масштабованість.

Однією з основних вимог є функціональність, яка включає можливість керування користувачами, товарами, замовленнями та фінансовими операціями. Система має підтримувати реєстрацію та авторизацію користувачів, контроль доступу до функціоналу відповідно до їхніх ролей, а також забезпечувати зручне керування каталогом товарів. Важливою є інтеграція механізмів пошуку, що дозволить швидко знаходити необхідні товари або інформацію.

Не менш важливою є зручність використання системи (юзабіліті). Інтерфейс повинен бути інтуїтивно зрозумілим і доступним для різних категорій користувачів. Важливою є швидкість виконання операцій, щоб забезпечити комфортний користувацький досвід від використання програми.

Система повинна відповідати вимогам безпеки, включаючи захист персональних даних (особливо пароля від акаунта користувача), безпечне збереження та передачу інформації, запобігання несанкціонованому доступу та забезпечення цілісності даних клієнтів.

Важливою характеристикою будь-якої системи є масштабованість. Вона має бути здатною обробляти зростаючу кількість користувачів, замовлень і взаємодію з базою даних без втрати продуктивності. Це

передбачає використання ефективної архітектури бази даних та оптимізованих алгоритмів обробки інформації у програмі.

Окрім цього, система повинна підтримувати аналітику та звітність, що дозволить адміністраторам отримувати актуальну інформацію про продажі, переваги користувачів та ефективність використання маркетингових стратегій. Графічне представлення аналітичних даних у вигляді діаграм допоможе швидко та легко приймати обґрунтовані рішення.

Таким чином, для будь-якої системи адміністрування електронної комерції необхідні функціональність, зручність, безпека, масштабованість та аналітичні можливості. Це забезпечить ефективну роботу з програмою як для клієнтів, так і для адміністраторів, сприяючи зростанню прибутку бізнесу та покращенню взаємодії з клієнтами.

1.3 Існуючі програмні рішення

В Україні також є кілька платформ та програм, які спеціалізуються на продажу наручних годинників в електронній комерції. Ось деякі з них, а також їх переваги та недоліки.

1.3.1 Rozetka

Переваги сервісу наведено далі.

Широкий асортимент. Rozetka – одна з найбільших онлайн-платформ в Україні, що пропонує великий вибір наручних годинників від різних брендів, як класичних, так і спортивних чи розкішних [6].

Безпека покупок. Платформа забезпечує захист покупців, зокрема можливість повернення товару, що є важливим аспектом для онлайн-покупок.

Масштабованість. Платформа обслуговує не тільки великі міста, а й віддалені населені пункти, забезпечуючи доставку по всій території України.

Недоліки сервісу наведено далі.

Нерідко високі ціни. На деякі моделі годинників ціни можуть бути вищими порівняно з іншими платформами або фізичними магазинами, особливо в сезон акцій чи знижок.

Перевантаження асортименту. Велика кількість товарів може створювати складнощі для пошуку саме потрібного товару, оскільки користувачі можуть зіткнутися з перевантаженням інформацією.

Результат пошуку та фільтрації товарів з використанням сервісу Rozetka, зображено на рисунку 1.1.

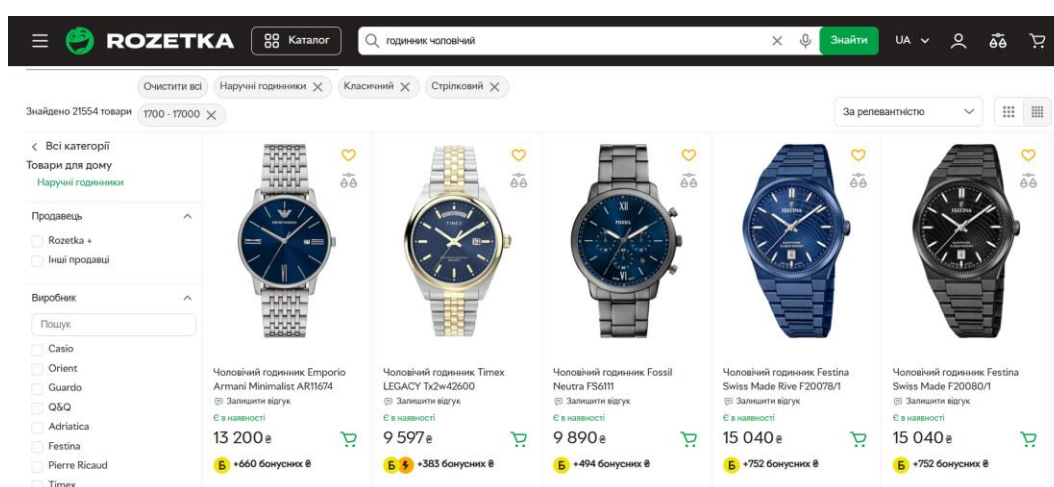


Рисунок 1.1 – Результат роботи сервісу Rozetka [6]

1.3.2 Allo.ua

Переваги сервісу наведено далі.

Добре організована навігація. Allo.ua має зручний інтерфейс, що дозволяє швидко знаходити наручні годинники за різними характеристиками, включаючи бренди, типи механізмів, кольори та інші параметри [7].

Акції та знижки. Платформа пропонує регулярні знижки та спеціальні акції, що робить покупки більш вигідними для клієнтів.

Широка доставка по Україні. Платформа здійснює доставку по всій території України, що дозволяє задовольняти потреби клієнтів з різних регіонів.

Недоліки сервісу наведено далі.

Обмежена кількість брендів. Хоча на платформі представлений широкий асортимент годинників, деякі відомі бренди можуть бути представлені лише обмежено або зовсім відсутні.

Не завжди актуальна інформація про наявність товару. У деяких випадках товари можуть бути позначені як доступні, але насправді їх немає в наявності.

Результат пошуку та фільтрації наручних годинників з використанням сервісу Allo.ua зображено на рисунку 1.2.

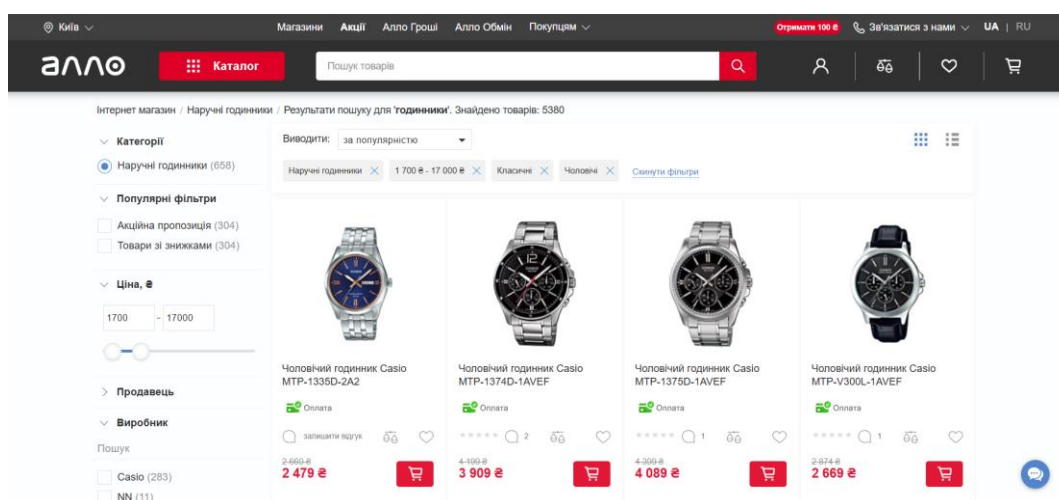


Рисунок 1.2 – Результат роботи сервісу Allo.ua [7]

1.3.3 Deka.ua

Deka.ua — це український онлайн-магазин, що спеціалізується на продажу як механічних, так і кварцових наручних годинників. На цьому сайті представлений широкий асортимент годинників від відомих міжнародних брендів, а також зручні умови для покупців, включаючи доставку та обслуговування [8].

Переваги сервісу deka.ua перелічено далі.

Широкий асортимент годинників.

Механічні годинники. Магазин пропонує різноманітні механічні моделі від таких брендів, як Seiko, Casio, Orient, Fossil, а також інші моделі класичного і спортивного стилю. Придбати механічний годинник можна в різних цінових категоріях.

Кварцові годинники. Окрім механічних моделей, на сайті є також велика кількість кварцових годинників, що користуються популярністю завдяки їх точності та простоті в обслуговуванні.

Офіційна гарантія та сертифікати. Кожен товар продається з офіційною гарантією, що підтверджує його оригінальність та відповідність всім стандартам якості. Також магазин надає сертифікати для моделей, що потребують спеціальної документації.

Зручний інтерфейс та пошук.

Фільтри для вибору. Сайт має зручний інтерфейс з фільтрами, що дозволяють швидко знайти необхідний годинник за різними параметрами, такими як тип механізму, матеріал корпусу, тип скла, колір циферблату та інші характеристики.

Інтуїтивно зрозуміле меню. Покупці можуть швидко орієнтуватися серед моделей, а також зручно порівнювати різні варіанти.

Підтримка клієнтів.

Консультації онлайн. Магазин надає послугу консультацій у режимі онлайн, де покупці можуть отримати відповіді на питання про годинники, їх характеристики та особливості.

Обслуговування після продажу. DeKa.ua також пропонує послуги з ремонту та обслуговування годинників, що додає додаткову цінність для покупців.

Недоліки сервісу deka.ua перелічено далі.

Обмежений вибір люксових брендів. Якщо ви шукаєте ексклюзивні або дуже дорогі моделі люксових брендів, вибір на цьому сайті може бути обмежений порівняно з іншими міжнародними магазинами.

Висока цінова категорія. Для покупців, які шукають бюджетні варіанти, ціни на деякі моделі можуть здатися високими, оскільки магазин орієнтований на середній та преміум-сегмент.

Deka.ua – це зручний і надійний онлайн-сервіс для тих, хто шукає механічні або кварцові годинники з офіційною гарантією, широким вибором брендів та доставкою по Україні. Магазин пропонує зручні умови покупки і добре розвинену інфраструктуру для підтримки своїх клієнтів [8].

Результат пошуку та фільтрації наручних годинників з використанням сервісу Deka.ua зображено на рисунку 1.3.

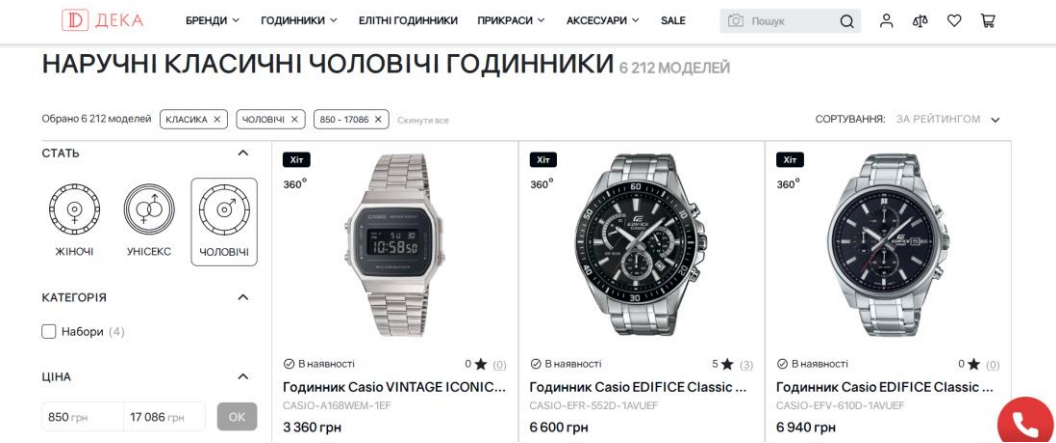


Рисунок 1.3 – Результат роботи сервісу Deka.ua [8]

Порівняння параметрів існуючих програмних рішень наведено у таблиці 1.1.

Таблиця 1.1 – Таблиця порівняння існуючих програмних рішень

Критерій	Rozetka	Allo.ua	Deka.ua
1	2	3	4
Основна спеціалізація	Універсальний маркетплейс	Універсальний маркетплейс	Спеціалізований магазин годинників
Фокус на механічних годинниках	Є, але не основна категорія	Присутній, але більший акцент на смарт-годинники	Основна спеціалізація

Продовження таблиці 1.1.

1	2	3	4
Асортимент годинників	Механічні, кварцові, смарт-годинники	Переважають смарт-годинники, але присутні і класичні моделі	Переважно механічні і кварцові годинники
Ціновий сегмент	Від бюджетного до преміального	Від бюджетного до середнього	Від середнього до преміального
Гарантія	Від продавців, не завжди надається офіційна гарантія	Залежить від продавців та бренду	Офіційна гарантія на всі годинники
Надання додаткових послуг після продажу	Немає	Немає	Є сервісне обслуговування
Наявність знижок	Наявні	Наявні	Наявні
Наявність аналітичних інструментів	Немає	Немає	Немає
Графічний інтерфейс користувача	Зрозумілий інтерфейс, але через те, що це не спеціалізований магазин, користувачеві може бути складно орієнтуватись	Гарний графічний інтерфейс, зрозумілий для користувача	Простий графічний інтерфейс

1.4 Необхідність розробки програмного забезпечення

Сучасний ринок електронної комерції вимагає ефективних та зручних інструментів для управління продажами, що дозволяють як адміністраторам, так і клієнтам зручно взаємодіяти із платформою. У зв'язку з цим виникає

необхідність розробки програмного рішення, яке забезпечить автоматизацію процесів адміністрування інтернет продажів механічних годинників.

Програмна система повинна надати менеджерам гнучкі інструменти для управління товарами, їх характеристиками та замовленнями. Зокрема, вони матимуть змогу додавати, редагувати та видаляти товари, оновлювати їхні описи, фото, матеріали, з яких виготовлені годинники, та ціни, а також підтверджувати замовлення клієнтів.

Важливою функцією буде доступ до аналітичних даних, представлених у вигляді графіків і діаграм, що дозволить ефективно відстежувати продажі та оцінювати їхні тенденції зміни. Менеджери також зможуть переглядати історію продажів, що допоможе в аналізі попиту та прийнятті стратегічних рішень щодо розвитку бізнесу.

Для клієнтів система має забезпечити інтуїтивно зрозумілий каталог, у якому вони зможуть переглядати годинники, додавати їх до списку «Обране», оформлювати замовлення та обирати додаткові послуги до угоди про продаж годинника.

Додатковою перевагою програмного забезпечення стане модуль аналізу вартості дорогоцінного металу, що дозволить оцінювати тенденції змін ринкових цін на матеріали, з яких виготовлені годинники. Ця функція буде корисною для клієнтів, які розглядають покупку годинника як довгострокову інвестицію.

Таким чином, розробка цього програмного забезпечення дозволить автоматизувати ключові бізнес-процеси, покращити взаємодію між клієнтами та менеджерами, а також забезпечити ефективний аналіз ринку наручних годинників, що є важливим фактором успіху в сфері електронної комерції.

1.5 Вимоги до розроблюваного програмного забезпечення

Розроблюване програмне забезпечення має реалізовувати наступні функціональні вимоги:

- застосунок передбачуватиме два типи користувачів, перший тип – адміністратори магазину, другий тип – клієнти магазину;
- можливість швидкої та простої реєстрації у застосунку;
- реалізація для менеджерів зручного керування товарами (додавання, видалення), з можливістю редагувати інформацію про товар;
- забезпечення зручної логіки підтвердження замовлення менеджерами;
- можливість перегляду аналітичних даних щодо продажів у вигляді діаграм для менеджерів;
- можливість у адміністраторів переглядати статистику власних продажів;
- можливість у адміністраторів переглядати історію як загальних, так і особистих продажів наручних годинників;
- реалізація перегляду клієнтами годинники у зручному каталозі;
- можливість у клієнтів додавати обрані годинники у список «Обране»;
- можливість надсилати замовлення на покупку годинника;
- можливість обирати додаткові послуги до замовлення;
- можливість переглядати курс вартості матеріалу за останні 10 років;
- реалізація модуля аналізу вартості дорогоцінного металу, з якого виготовлено механічні годинники, що дозволить точніше оцінювати тенденцію зміни їх ринкової ціни, з метою можливості придбання годинника як інвестицію.

Вимоги до графічного інтерфейсу програми:

- розробка інтерфейсу з використанням технології WPF;
- дружній та інтуїтивно зрозумілий графічний інтерфейс;
- проста та чітка навігація між вікнами користувача;
- візуалізація статистики продажів у вигляді графічних діаграм.

Вимоги до безпеки програми:

- збереження даних у базі даних для забезпечення цілісності та безпеки даних;
- приховування паролів при авторизації користувача;
- збереження конфіденційності користувачів при взаємодії з програмою.

1.6 Висновки з розділу

Аналіз предметної області та існуючих програмних рішень у сфері електронної комерції показав, що більшість платформ орієнтовані на широкий спектр товарів, тоді як спеціалізований магазин механічних годинників повинен надавати додаткові можливості, такі як аналітика вартості матеріалів, історія змін цін та детальну інформацію про годинник. Виявлені під час аналізу переваги та недоліки таких програмних рішень дозволили сформулювати основні вимоги до системи адміністрування інтернет продажів наручних годинників.

Запропоноване програмне забезпечення повинне забезпечити менеджерам ефективне керування товарним асортиментом, замовленнями, аналітичними даними та взаємодією з клієнтами. Водночас, клієнти матимуть змогу зручно переглядати годинники, формувати власні списки обраних товарів, оформлювати замовлення та отримувати доступ до аналітики вартості дорогоцінних металів. Результати аналізу підтвердили необхідність розробки спеціалізованого програмного забезпечення, яке дозволить автоматизувати управління інтернет продажами механічних годинників, підвищити ефективність обслуговування клієнтів і покращити процес аналізу ринку.

Реалізація таких функціональних можливостей сприятиме підвищенню конкурентоспроможності магазину та створенню зручної і сучасної програми для продажу механічних годинників.

2 МАТЕРІАЛИ І МЕТОДИ

У даному розділі розглянуті основні особливості обраної мови програмування, програмних засобів, фреймворків та бібліотек за допомогою яких реалізовано програмне забезпечення.

2.1 Огляд особливостей обраної мови програмування

C# (C-Sharp) – це об'єктно-орієнтована мова програмування, розроблена компанією Microsoft у рамках платформи .NET [9]. Вона поєднує в собі елементи C++, Java та інших мов, що робить її зручною для широкого кола розробників. Основні особливості C# наведено далі.

Об'єктно-орієнтована парадигма. C# підтримує такі концепції ООП:

- інкапсуляція (приховування реалізації через модифікатори доступу `private`, `protected`, `public`) [10];
- наслідування (`class Derived : Base`) [10];
- поліморфізм (використання `virtual`, `override`, `abstract` методів) [10];
- інтерфейси (`interface` для реалізації множинного наслідування) [10].

Автоматичне управління пам'яттю. Використовує збір сміття (Garbage Collector, GC) для очищення непотрібних об'єктів.

Немає потреби в явному звільненні пам'яті, як у C++ (`delete` або `free`).

Типізація та безпека.

Статична типізація – змінні мають визначений тип (`int`, `string`, `bool` тощо).

Безпека типів – відсутні покажчики (але можна використовувати `unsafe` код).

Nullable типи (`int?`, `double?`) для безпечної роботи зі значеннями `null`.

Розширена робота з колекціями.

- Масиви (`Array`);

- Generic-колекції (List<T>, Dictionary<TKey, TValue>);
- LINQ (мова запитів для колекцій: var result = myList.Where(x => x > 17)).

Асинхронне програмування. C# підтримує асинхронні операції через async/await, що дозволяє створювати ефективні програми.

Кросплатформність. Раніше C# працював тільки на Windows, але з появою .NET Core мова стала кросплатформною:

- можливість запуску на Windows, Linux, macOS;
- створення веб-додатків, мобільних додатків (Xamarin), ігор (Unity).

Безпека та сучасні можливості:

- атрибути ([Obsolete], [Serializable]);
- динамічні типи (dynamic);
- патерни проєктування (Singleton, Factory, Dependency Injection).

Популярні фреймворки:

- ASP.NET Core – для веб-розробки;
- Entity Framework (EF Core) – для роботи з базами даних [13];
- Blazor – для веб-додатків із C# замість JavaScript;
- MAUI – для мобільної розробки.

Порівняння обраної мови програмування з іншими мовами за визначеними критеріями наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування

Критерій	C#	C++	Python
1	2	3	4
Типізація	Статична, яка є безпечною	Статична	Динамічна
Складність	Середня	Висока	Низька

Продовження таблиці 2.1.

1	2	3	4
Парадигма	Об'єктно-орієнтована, імперативна	Об'єктно-орієнтована, процедурна, функціональна	Об'єктно-орієнтована, процедурна, функціональна
Автоматичне управління пам'яттю	Так (Garbage Collector)	Ні	Так (Garbage Collector)
Швидкість роботи	Висока	Дуже висока	Низька
Кросплатформність	Так	Так	Так
Основне призначення	Веб, десктоп, мобільні додатки, ігри	Системне програмування, ігри, високопродуктивні програми	Веб, наукові розрахунки, машинне навчання

C# – це потужна, безпечна та зручна мова програмування, яка підходить як для новачків, так і для досвідчених розробників. Вона активно розвивається, підтримується Microsoft і широко застосовується в різних сферах розробки програм.

Використання C# для розробки програмної системи інтернет-продажів наручних годинників має низку переваг. Завдяки об'єктно-орієнтованому підходу мови можна зручно моделювати товари, замовлення та користувачів, що спрощує розширення функціоналу. Платформа .NET забезпечує високу продуктивність, кросплатформність та ефективну взаємодію з базами даних через Entity Framework Core. Використання технології WPF дозволить створити зручний та сучасний UI.

C# також надає потужні засоби для аналітики та візуалізації даних, що важливо для відстеження продажів та ринкових тенденцій. Завдяки можливості масштабування та доповнення новими модулями, C# стане оптимальним вибором для створення надійної та гнучкої системи управління товарами та замовленнями.

2.2 Огляд особливостей обраного середовища розробки

Visual Studio Community 2022 – це безкоштовне інтегроване середовище розробки від Microsoft, яке забезпечує потужний інструментарій для створення додатків на C#, C++, Python, JavaScript та інших мовах програмування. Воно підтримує розробку для Windows, вебу, мобільних платформ та хмарних сервісів.

Серед головних особливостей Visual Studio Community 2022 варто відзначити зручний інтерфейс із покращеною підтримкою високої роздільної здатності, 64-бітну архітектуру для швидкого завантаження та роботи з великими проєктами, а також глибоку інтеграцію з .NET 7/8, Blazor, ASP.NET та Entity Framework. Завдяки IntelliCode забезпечується розумне автозаповнення коду на основі штучного інтелекту, а вбудовані засоби відлагодження дозволяють виконувати покрокову відладку, профілювати продуктивність і аналізувати використання пам'яті.

Інструмент підтримує вбудовану інтеграцію з Git та GitHub для зручного керування версіями, а також надає можливість кросплатформної розробки для Windows, Linux, macOS та мобільних пристроїв через Xamarin і .NET MAUI. Додатково Visual Studio пропонує засоби аналізу коду для оптимізації та виправлення помилок, а також можливість розширення функціоналу через Visual Studio Marketplace.

Visual Studio Community 2022 є ефективним середовищем розробки, що підходить для навчання, фрілансу та малого бізнесу, надаючи професійні інструменти безкоштовно.

Порівняння інтегрованих середовищ розробки за визначеними критеріями наведено у таблиці 2.2.

Таблиця 2.2 – Порівняння інтегрованих середовищ розробки

Критерій	Visual Studio Community 2022	Visual Studio Code	SharpDevelop
Продуктивність	Потужне IDE, але може сповільнюватися при великих проєктах	Легке і швидке IDE, можна розширити функціональність	Легке IDE, швидке, але обмежене у функціональності
Функціональність	Вбудований налагоджувач, інструменти для тестування, DevOps	Підтримує плагіни, має вбудований термінал, інструменти для DevOps	Базові інструменти для .NET розробки, обмежена функціональність
Ціна	Безкоштовне, але для комерційного використання є платна версія	Безкоштовне	Безкоштовне
Додаткові інструменти	Інтеграція з GitHub, розширення Marketplace	Велика кількість розширень у Marketplace, інтеграція з Git	Підтримка розширень, інтеграція з Git
Особливості	Підтримка .NET Framework, .NET Core, DevOps	Гнучке налаштування середовища, підтримка роботи у хмарі	Підходить для невеликих проєктів та навчання

Visual Studio Community 2022 є оптимальним середовищем для розробки системи адміністрування електронної комерції інтернет продажів, замовленнями та аналітикою, оскільки забезпечує зручний та ефективний процес створення застосунків, серверної логіки та інтеграції з базами даних.

Завдяки глибокій інтеграції з C# та .NET, Visual Studio надає вбудовані засоби для розробки додатків з використанням технології WPF, що дозволяє створювати швидку, безпечну та масштабовану систему. Інструменти для

роботи з Entity Framework Core забезпечують зручну взаємодію з базою даних, що важливо для управління товарами та замовленнями.

Інтелектуальне автозаповнення IntelliCode та потужні засоби відлагодження спрощують розробку складної бізнес-логіки, зокрема для реалізації аналітики продажів і моніторингу вартості дорогоцінних металів. Вбудована підтримка Git та GitHub дозволяє ефективно керувати версіями коду, що особливо важливо для командної роботи.

Оскільки система повинна мати зручний користувацький інтерфейс для менеджерів і клієнтів, Visual Studio підтримує технологію WPF, що дає змогу створювати функціональні графічні інтерфейси без використання сторонніх технологій.

Крім того, завдяки 64-бітній архітектурі та оптимізованій продуктивності середовище забезпечує швидку компіляцію, що пришвидшує розробку та тестування системи. Visual Studio також має вбудовані засоби аналізу продуктивності та безпеки, що гарантує стабільність та захист даних користувачів.

Таким чином, використання Visual Studio Community 2022 у цьому проєкті дозволить швидко розробити надійну, продуктивну та масштабовану систему з сучасним інтерфейсом і широкими можливостями аналітики статистики продажів.

2.3 Огляд особливостей обраної системи управління базою даних

Microsoft SQL Server – це реляційна система керування базами даних (СКБД), яка забезпечує високу продуктивність, безпеку та масштабованість для роботи з великими обсягами даних.

Однією з ключових особливостей є розширені можливості аналітики та бізнес-інтелекту, що дозволяє ефективно обробляти дані та будувати звіти за допомогою інтегрованих інструментів, таких як SQL Server Reporting Services (SSRS) та SQL Server Analysis Services (SSAS).

SQL Server підтримує мову запитів T-SQL, яка дозволяє виконувати складні операції з даними, оптимізуючи їхнє зберігання та обробку. Вбудовані механізми оптимізації запитів та кешування покращують продуктивність, що особливо важливо для високонавантажених систем.

Важливою перевагою є високий рівень безпеки, включно з підтримкою шифрування даних, автентифікацією через Active Directory та функціями контролю доступу на рівні рядків (Row-Level Security).

Microsoft SQL Server має інструменти для резервного копіювання та відновлення, що забезпечують надійність збереження даних і мінімізують ризики втрати інформації.

Система також підтримує масштабованість, що дозволяє використовувати її як для невеликих проєктів, так і для великих корпоративних рішень.

Таким чином, SQL Server є потужною, надійною та функціональною СКБД, яка підходить для розробки бізнес-додатків, обробки великих даних та побудови складних аналітичних рішень.

Порівняння СКБД за визначеними критеріями наведено у таблиці 2.3.

Таблиця 2.3 – Порівняння систем управління базою даних

Критерій	Microsoft SQL Server	MySQL	Access
1	2	3	4
Тип СКБД	Реляційна	Реляційна	Файлова, реляційна
Розробник	Microsoft	Oracle Corporation	Microsoft
Призначення	Високонавантажені корпоративні системи, аналітика, бізнес-рішення	Веб-додатки, середні та малі проєкти	Невеликі локальні бази даних, офісні рішення
Мова запитів	T-SQL	SQL	SQL (обмежений) + VBA
Продуктивність	Висока	Висока	Низька
Безпека	Високий рівень	Високий рівень	Низький рівень

Продовження таблиці 2.3.

1	2	3	4
Масштабованість	Висока, підтримка кластеризації та реплікації	Висока	Обмежена
Підтримка ACID	Так	Так	Часткова
Підтримка транзакцій	Так	Так	Часткова

За результатами порівняння можна сказати, що Microsoft SQL Server найбільше підходить для програмного проєкту адміністрування інтернет-продажів, де потрібна продуктивність, безпека та аналітика.

2.4 Огляд особливостей технології розробки графічного інтерфейсу

Windows Presentation Foundation (WPF) – це сучасна технологія для розробки графічного інтерфейсу користувача в середовищі Windows [1]. Вона входить до складу платформи .NET і використовує мову розмітки XAML для опису інтерфейсу, що дозволяє відокремити логіку програми від її візуальної частини.

Основні особливості WPF перелічено далі.

Гнучка система розмітки (XAML). WPF використовує XAML (Extensible Application Markup Language) для опису UI, що дозволяє легко структурувати інтерфейс і змінювати його без редагування основного коду [11].

Підтримка апаратного прискорення. Завдяки використанню DirectX для рендерингу графіки, WPF забезпечує плавну анімацію та ефективне відображення складних візуальних елементів.

Зв'язування даних (Data Binding). WPF підтримує MVVM (Model-View-ViewModel), що дозволяє автоматично синхронізувати дані між UI та бізнес-логікою, спрощуючи розробку динамічних застосунків [12].

Стили та шаблони (Styles & Templates). Інтерфейс у WPF легко налаштовується завдяки CSS-подібним стилям, що дозволяють централізовано змінювати вигляд елементів.

Графіка та анімація. Підтримка векторної графіки, 2D і 3D рендерингу, анімацій дозволяє створювати сучасний, інтерактивний UI.

Масштабованість і адаптивність. Завдяки автоматичному масштабуванню DPI інтерфейс виглядає чітко незалежно від роздільної здатності екрану.

Підтримка мультимедіа. WPF дозволяє легко вбудовувати аудіо, відео та 3D-графіку, що робить його ідеальним для мультимедійних застосунків.

WPF є потужним інструментом для створення сучасних десктопних застосунків із розширеними можливостями кастомізації, продуктивною обробкою UI та підтримкою архітектури MVVM. Він підходить для складних бізнес-додатків, аналітичних панелей та мультимедійних застосунків у середовищі Windows.

2.5 Огляд особливостей Entity Framework для роботи з базою даних

Entity Framework (EF) – це ORM (Object-Relational Mapping) фреймворк від Microsoft, який спрощує взаємодію .NET-додатків із реляційними базами даних [13]. Він дозволяє працювати з базою за допомогою об'єктів C#, без написання SQL-запитів вручну.

Основні особливості Entity Framework наведено далі.

Підтримка різних підходів до роботи з базою:

- Code First – спочатку створюється модель у коді C#, а потім генерується база даних;
- Database First – використовується вже існуюча база даних, а модель створюється автоматично;
- Model First – спочатку створюється модель візуально, після чого генерується база.

Автоматичне відображення об'єктів на таблиці (ORM). EF дозволяє працювати з базою даних через C#-класи (Entity-класи), уникаючи ручного написання SQL-коду.

Зв'язування та навігаційні властивості. Фреймворк підтримує один-до-одного, один-до-багатьох та багато-до-багатьох зв'язки між таблицями через об'єктні моделі.

Підтримка LINQ. EF дозволяє писати запити до бази через LINQ to Entities, що спрощує роботу з даними та робить код більш читабельним.

Автоматичне відстеження змін (Change Tracking). Фреймворк автоматично визначає зміни в об'єктах і формує відповідні SQL-запити при збереженні змін.

Міграції бази даних. Інструмент EF Migrations дозволяє легко оновлювати структуру бази даних без втрати даних при змінах у моделях.

Кросплатформність. Entity Framework Core підтримує роботу не тільки з SQL Server, а й з MySQL, PostgreSQL, SQLite та іншими СУБД, а також працює у середовищах Windows, Linux, macOS.

Entity Framework значно спрощує роботу з базами даних у .NET-додатках, дозволяючи писати чистий та підтримуваний код без прямої роботи з SQL-запитами. Він підходить як для малих, так і для великих проєктів, забезпечуючи ефективну взаємодію з реляційними базами даних.

2.6 Огляд особливості підходу Database First для роботи з базою даних

Database First – це підхід у Entity Framework, який дозволяє спочатку створити базу даних у СКБД, а потім автоматично згенерувати C#-класи об'єктної моделі для роботи з нею. Він ідеально підходить для випадків, коли база вже існує або її розробляють окремо від застосунку.

Основні особливості підходу Database First перелічено далі.

Автоматична генерація моделей. За допомогою Entity Framework Tools можна створити Entity-класи моделі та контекст роботи з базою даних DbContext (клас що прив'язує модель до БД) без написання коду вручну, що дозволяє зекономити багато часу, спрощуючи інтеграцію з базою.

Синхронізація зі схемою бази. При зміні структури бази (наприклад, додавання нових таблиць) можна оновити модель без редагування вручну, єдиний недолік цього підходу – це те, що модель згенерується заново, відкинувши усі поточні зміни у коді програми моделі.

Підтримка складних реляційних структур. Database First дозволяє легко працювати з зовнішніми ключами, зв'язками «один-до-багатьох» і «багато-до-багатьох», які вже визначені у базі даних.

Збереження контролю за SQL-структурою. Оскільки спочатку створюється база, розробники можуть оптимізувати SQL-запити, налаштовувати індекси, тригери та процедури, перш ніж вони будуть використані в застосунку.

Можливість використання з великими системами. Цей підхід добре підходить для великих проєктів, де базу даних розробляє окрема команда, а застосунок має адаптуватися до готової структури.

Використання LINQ та ORM. Після генерації моделей можна працювати з базою через LINQ-запити та використовувати всі переваги Entity Framework, як у Code First.

Менше ризиків помилок у моделі. Оскільки схема бази створюється окремо, знижується ризик несумісності або логічних помилок у структурі даних.

Database First – це ідеальний варіант для програмного проєкту адміністрування електронної комерції інтернет-продажів наручних годинників, де база даних уже створена у СКБД Microsoft SQL Server. Він дає змогу зосередитися на бізнес-логіці застосунку, використовуючи переваги Entity Framework для роботи з даними.

2.7 Схеми розробленої бази даних

Схеми розробленої бази даних у СКБД Microsoft SQL Server з усіма зв'язками між таблицями та їх полями зображена на рисунку 2.1.

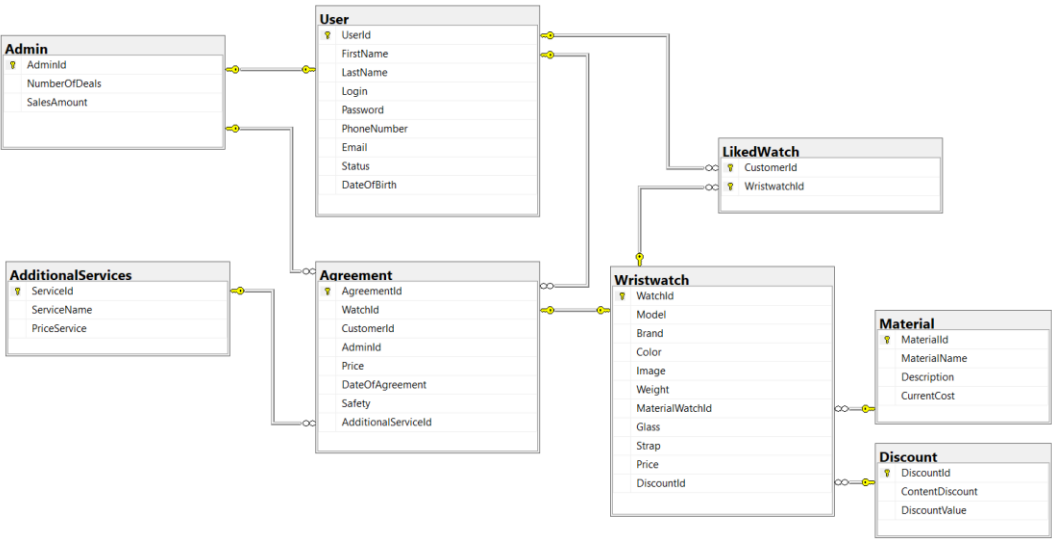


Рисунок 2.1 – Схеми розробленої бази даних

Результат створення об'єктної моделі зі збереженням усіх зв'язків у схемі на основі розробленої бази даних використовуючи Entity Framework та підхід Database First зображено на рисунку 2.2.

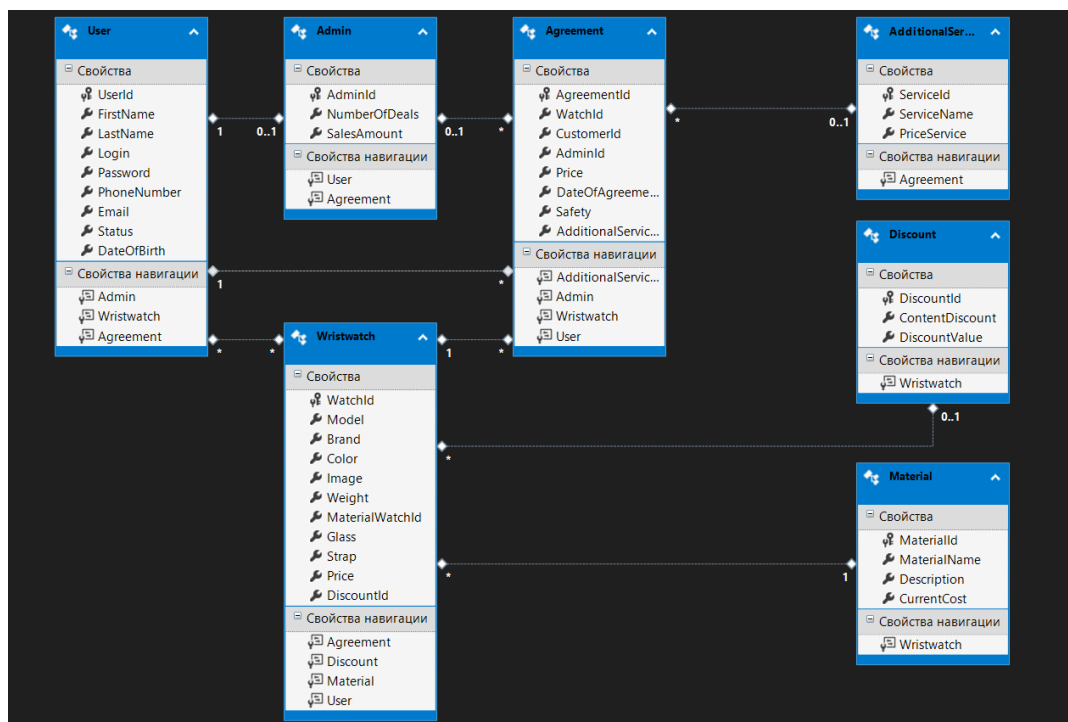


Рисунок 2.2 – Схема об'єктної моделі у програмі

2.8 Висновки з розділу

Розглянуті засоби розробки демонструють потужний інструментарій для створення сучасного програмного забезпечення для електронної комерції інтернет-продажів наручних годинників. Використання мови програмування C# у поєднанні з .NET забезпечує високу продуктивність та гнучкість у розробці. Інтегроване середовище Visual Studio Community 2022 надає широкий набір інструментів для ефективного написання, тестування та налагодження коду програми.

Для роботи з базою даних Microsoft SQL Server гарантує надійність, масштабованість та безпеку даних, а Entity Framework значно спрощує взаємодію з нею через об'єктно-реляційне відображення. Підхід Database First дозволяє легко інтегруватися з розробленою базою даних та забезпечує узгодженість структури даних.

Розробка графічного інтерфейсу за допомогою WPF дає змогу створювати гнучкі та привабливі UI-рішення з підтримкою гарного інтерфейсу, анімацій та сучасних архітектурних підходів, таких як MVVM.

Створення діаграм статистики продажів та зміни курсу ціни матеріалу, з якого виготовлено наручний годинник, буде реалізовано за допомогою фреймворку LiveCharts [14].

Таким чином, поєднання цих технологій дозволяє розробити масштабовану, продуктивну та зручну програмну систему, що відповідає вимогам технічного завдання.

3 ОПИС ПРОГРАМИ

Для розробки програмного застосунку було використано об'єктно-орієнтований підхід, який використовує класи та методи. Клас є основним компонентом C#, що виконує роль представлення конкретної моделі сутності об'єкта, який використовується у програмі.

3.1 Алгоритм функціонування розробленої програми

Основні алгоритми, що можуть виконуватися при роботі із розробленою програмою зображено на рисунку 3.1.

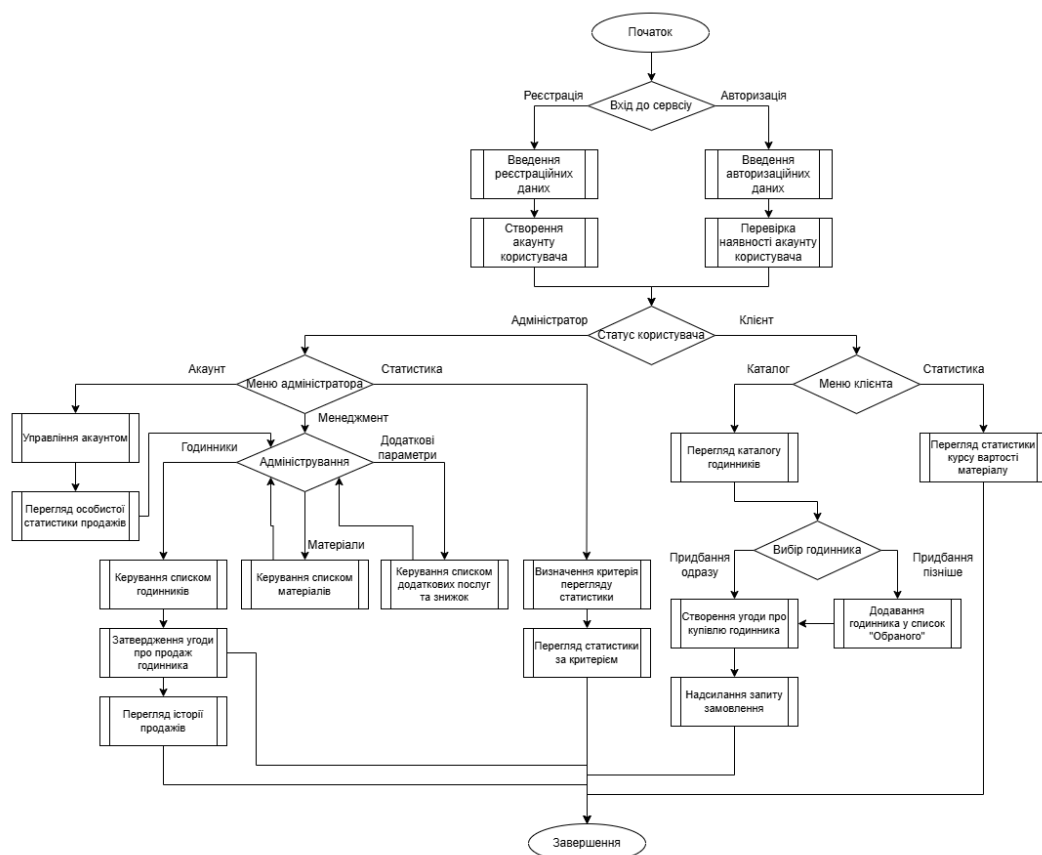


Рисунок 3.1 – Алгоритм роботи розробленої програми

Для функціонування програмного забезпечення на тему «Програмна реалізація системи адміністрування електронної комерції на прикладі інтернет продажів наручних годинників» було розроблено такі основні алгоритми:

- авторизація та реєстрація користувача у системі;
- додавання нового годинника до системи;
- алгоритм додавання угоди про купівлю годинника;
- алгоритм для підтвердження адміністратором продажу годинника;
- алгоритм розрахунку особистої статистики продажів адміністратора;
- алгоритм перегляду історії продажів та покупок з додаванням пагінації у список;
- алгоритм обробки інформації з json-файлу для відображення зміни на протязі 10 років вартості матеріалу;
- алгоритм розрахунку коефіцієнту приросту ціни матеріалу, для поради користувачеві чи варто інвестувати у цей матеріал;
- алгоритм додавання годинника до списку «Обраного».

3.1.1 Авторизація та реєстрація користувача у системі

Реєстрація користувача (Registration_User).

Мета: Додати нового користувача у базу даних та, якщо він адміністратор, зареєструвати його як адміністратора.

Отримання даних від користувача: введення логіна, пароля, імені, статусу (User або Admin).

Додавання користувача в базу (Add_User_To_DB): відкривається підключення до бази через WatchDataBaseEntities.

Перевіряється, чи існує користувач із таким логіном.

Якщо користувача немає, додається новий запис у таблицю User.

Викликається context.SaveChanges() для збереження змін.

Виводиться повідомлення про успішну реєстрацію або помилку.

Перевірка статусу користувача: якщо `user.Status == "Admin"`, необхідно зареєструвати його як адміністратора.

Викликається `Get_User(login)`, щоб отримати ID новоствореного користувача.

Виконується `Registration_Admin(admin.UserId)`, де створюється запис в таблиці `Admin`.

Реєстрація адміністратора (`Registration_Admin`).

Мета: Створити запис про адміністратора в базі даних.

Створення об'єкта `Admin`: `AdminId = adminId` (ідентифікатор користувача).

`NumberOfDeals = 0`, `SalesAmount = 0` (початкові значення).

Додавання в базу (`Add_Admin_To_DB`): відкривається підключення до бази (`WatchDataBaseEntities`).

Об'єкт `Admin` додається до таблиці `Admin`.

Викликається `context.SaveChanges()`.

Виводиться повідомлення про успішну реєстрацію або помилку.

Авторизація користувача (`Authorization_User`).

Мета: Перевірити введені логін і пароль, щоб дозволити доступ до системи.

Отримання введених даних (логін і пароль).

Отримання користувача з бази (`Get_User(login)`).

Перевірка відповідності даних:

- якщо користувача немає – виводиться повідомлення "Please enter the correct login", повертається `false`;
- якщо логін і пароль співпадають – повертається `true` (успішний вхід);
- якщо пароль неправильний – виводиться повідомлення "Enter the correct password", повертається `false`.

3.1.2 Додавання нового годинника до системи

Мета: Додавання нового наручного годинника (Wristwatch) до бази даних.

Відкриття підключення до бази даних.

Використовується using (var context = new WatchDataBaseEntities()), що гарантує автоматичне закриття підключення після завершення роботи методу.

WatchDataBaseEntities – це об'єкт контексту, який взаємодіє з базою даних через Entity Framework.

Виконання операції додавання (try-блок).

context.Wristwatch.Add(wristwatch) – додає новий екземпляр Wristwatch у відповідну таблицю бази даних.

context.SaveChanges() – зберігає зміни у базі даних, фізично додаючи запис.

Виводиться повідомлення "Wristwatch was added" для підтвердження успішного додавання.

Обробка винятків (catch-блок).

Якщо під час операції виникає помилка (наприклад, порушення цілісності даних, проблеми з підключенням до бази), її текст відображається через MessageBox.Show(ex.Message).

3.1.3 Алгоритм додавання угоди про купівлю годинника

Мета:

Метод створює угоду (Agreement) між користувачем (User) і наручним годинником (Wristwatch), додає її в базу даних, а потім перевіряє, чи цей годинник був у списку "улюблених" користувача, і, якщо так, видаляє його звідти.

Послідовність виконання алгоритму наведена далі.

Присвоєння значень полям угоди (Agreement): `Agreement.WatchId = Wristwatch.WatchId`.

Присвоюється ідентифікатор годинника, який купує користувач: `Agreement.CustomerId = User.UserId`.

Встановлюється ідентифікатор користувача, який укладає угоду: `Agreement.DateOfAgreement = DateTime.Now.Date`.

Фіксується поточна дата угоди.

Додавання угоди до бази даних: викликається метод `dataBaseManage.Add_Agreement_To_DB(agreement)`, який додає угоду в таблицю `Agreement`.

Перевірка, чи годинник був у списку "улюблених": викликається `dataBaseManage.Check_LikedWristwatch_In_DB(user.UserId, Wristwatch.WatchId)`.

Якщо годинник був доданий у список "улюблених", метод повертає `true`.

Видалення годинника з "улюблених", якщо він там є: якщо `Check_LikedWristwatch_In_DB` повернув `true`, викликається `dataBaseManage.Delete_LikedWristwatch_From_DB(user.UserId, Wristwatch)`.

Видаляється запис про те, що користувач додав цей годинник у список "улюблених".

3.1.4 Алгоритм для підтвердження адміністратором продажу годинника

Мета:

Метод підтверджує угоду (Agreement) адміністратором, оновлює дані угоди в базі, а також оновлює статистику адміністратора (кількість угод і загальну суму продажів).

Послідовність виконання алгоритму наведено далі.

Призначення адміністратора угоди: `agreement.AdminId = user.UserId`.

Призначається AdminId як UserId поточний користувач (передбачається, що це адміністратор, який підтверджує угоду).

Оновлення угоди в базі даних:
dataBaseManage.Update_Agreemet_For_Submit(agreement).

Викликається метод для оновлення статусу угоди в БД, що, означає зміну її статусу на "підтверджена".

Отримання даних адміністратора: var admin =
dataBaseManage.Get_Admin_From_DB(user.UserId).

Викликається метод отримання адміністратора за його UserId.

Це необхідно для подальшого оновлення статистики адміністратора.

Оновлення кількості угод адміністратора:
dataBaseManage.Set_Number_Of_Deals_For_Admin(admin).

Викликається метод, який збільшує кількість укладених угод (NumberOfDeals) адміністратора.

Оновлення загальної суми продажів адміністратора:
dataBaseManage.Set_Sales_Amount_For_Admin(admin, agreement).

Метод оновлює загальну суму продажів (SalesAmount) адміністратора, додаючи вартість угоди (agreement.Amount).

3.1.5 Алгоритм розрахунку особистої статистики продажів адміністратора

Мета методу Set_Number_Of_Deals_For_Admin(Admin admin):

Збільшити лічильник угод (NumberOfDeals) для адміністратора в базі даних.

Послідовність виконання алгоритму наведено далі.

Відкриття підключення до бази даних: using(var context = new WatchDataBaseEntities()) – створює контекст для взаємодії з БД.

using забезпечує автоматичне закриття підключення після завершення операції.

Пошук адміністратора в БД: `var adminUpdate = context.Admin.Find(admin.AdminId).`

Використовується `Find()`, що знаходить адміністратора за `AdminId`.

Перевірка, чи знайдено адміністратора: `if(adminUpdate != null)` – якщо `adminUpdate` не `null`, можна оновлювати дані.

Збільшення кількості угод: `adminUpdate.NumberOfDeals++` – збільшує кількість угод на 1.

Збереження змін у БД: `context.SaveChanges()` – зберігає оновлені дані в БД.

Обробка можливих винятків: якщо виникає помилка (наприклад, адміністратора немає в БД або проблеми з підключенням), вона виводиться у `MessageBox.Show(ex.Message).`

Мета методу `Set_Sales_Amount_For_Admin(Admin admin, Agreement agreement):`

Додати до загальної суми продажів адміністратора (`SalesAmount`) вартість підтвердженої угоди (`agreement.Price`).

Послідовність виконання коду наведено далі.

Відкриття підключення до бази даних: `using(var context = new WatchDataBaseEntities())` – ініціалізується контекст бази даних.

Пошук адміністратора в БД: `var adminUpdate = context.Admin.Find(admin.AdminId).`

Отримуємо адміністратора за його `AdminId`.

Перевірка, чи знайдено адміністратора: `if(adminUpdate != null)` – переконуємось, що такий адміністратор існує.

Додавання суми угоди до загальної суми продажів: `adminUpdate.SalesAmount += agreement.Price.`

До `SalesAmount` додається вартість підтвердженої угоди.

Збереження змін у БД: `context.SaveChanges()` – зберігає оновлені дані.

Обробка можливих винятків: якщо виникає помилка, її текст відображається у `MessageBox.Show(ex.Message).`

3.1.6 Алгоритм перегляду історії продажів та покупок з додаванням пагінації у список

Алгоритм роботи та мета методу `Get_AgreementHistory_From_DB`:

Отримати історію угод (`Agreement`) з бази даних для адміністратора (`Admin`) з пагінацією.

Послідовність виконання коду наведено далі.

Відкриття підключення до бази даних: `using(var context = new WatchDataBaseEntities())` – створює контекст для взаємодії з базою даних.

Обчислення загальної кількості сторінок: викликається метод `Get_TotalPages_AgreementHistory(pageSize, admin)`, щоб визначити максимальну кількість сторінок.

Перевірка, чи `pageNumber` не перевищує `totalPages`: якщо запитуваний номер сторінки (`pageNumber`) більший за загальну кількість сторінок (`totalPages`), він змінюється на `totalPages`, щоб уникнути виходу за межі даних.

Формування запиту до бази даних: отримуються записи з таблиці `Agreement`, при цьому:

- використовується `Include("Wristwatch")` та `Include("AdditionalServices")` для завантаження зв'язаних об'єктів (годинники та додаткові послуги);
- фільтруються тільки ті записи, де `AdminId` не `null` (тобто угоди, підтверджені адміністратором);
- угоди сортуються за `AgreementId`;
- використовуються `Skip((pageNumber - 1) * pageSize)` та `Take(pageSize)`, щоб реалізувати пагінацію.

Перетворення вибірки у список та повернення результату: `ToList()` формує список об'єктів `Agreement`.

Якщо вибірка пройшла успішно, список повертається як результат.

Обробка винятків (catch-блок): якщо під час запиту виникає помилка, її текст виводиться через `MessageBox.Show(ex.Message)`.

У разі помилки повертається порожній список (`new List<Agreement>()`).

Алгоритм роботи та мета методу `Get_TotalPages_AgreementHistory`:

Обчислення загальної кількості сторінок для пагінації історії угод.

Послідовність виконання коду наведено далі.

Відкриття підключення до бази даних: `using(var context = new WatchDataBaseEntities())` – створює контекст для взаємодії з базою даних.

Обчислення загальної кількості угод (`totalRecords`).

Якщо `user.Status == "Admin"`: отримується загальна кількість угод, які підтвердив будь-який адміністратор (`context.Agreement.Where(rec => rec.AdminId != null).Count()`).

Якщо `user.Status == "Customer"`: отримується кількість угод, укладених цим користувачем (`context.Agreement.Where(rec => rec.CustomerId == user.UserId && rec.AdminId != null).Count()`).

Обчислення загальної кількості сторінок: використовується `Math.Ceiling(((double)totalRecords / pageSize)`, щоб округлити число сторінок у більшу сторону.

Повернення загальної кількості сторінок: якщо все пройшло успішно, повертається `totalPages`.

Якщо виникає помилка, її текст виводиться у `MessageBox.Show(ex.Message)`, а метод повертає 0.

3.1.7 Алгоритм обробки інформації з json-файлу для відображення зміни на протязі 10 років вартості матеріалу

Алгоритм роботи та мета методу `Get_MaterialsPrice_From_JSON`:

Метод зчитує дані про матеріали та їхню ціну з JSON-файлу, перетворює їх у список об'єктів `MaterialDto`, а також встановлює перший елемент як вибраний матеріал.

Послідовність виконання коду наведено далі.

Зчитування JSON-файлу: `File.ReadAllText(filePath)`.

Читає вміст JSON-файлу за шляхом `filePath` і зберігає його у змінну `json` у вигляді рядка.

Десеріалізація JSON-даних у C#-об'єкти: `JsonConvert.DeserializeObject<Dictionary<string, List<MaterialsJsonModel>>>(json)`.

Десеріалізує JSON-рядок у словник (`Dictionary<string, List<MaterialsJsonModel>>`), де:

- ключ – це `materials`;
- значення – список об'єктів `MaterialsJsonModel`.

Перевірка, чи JSON-дані коректні: `if (jsonData != null && jsonData.TryGetValue("materials", out var materialsList))`.

Переконується, що JSON-файл містить необхідні дані (ключ `materials`).

Якщо `materialsList` існує, метод продовжує обробку.

Перетворення `MaterialsJsonModel` у `MaterialDto`: `Materials = materialsList.Select(m => new MaterialDto { ... }).ToList()`.

Кожен об'єкт `MaterialsJsonModel` конвертується у `MaterialDto`.

Поле `PriceByYears` (ціни по роках) заповнюється за допомогою `ToDictionary(price => price.Year, price => price.Price)`.

Перевірка, чи список `Materials` не порожній: `if (Materials.Count > 0)`.

Якщо після обробки JSON є хоча б один матеріал, перший елемент списку (`Materials.First()`) призначається у `SelectedMaterial`.

3.1.8 Алгоритм розрахунку коефіцієнту приросту ціни матеріалу, для поради користувачеві чи варто інвестувати у цей матеріал

Мета:

Метод аналізує зміну ціни вибраного матеріалу за роками та формує рекомендацію щодо інвестування.

Послідовність виконання коду наведено далі.

Очищення повідомлення про рекомендацію: `Advice_Message = null`.

Видаляє попередні рекомендації перед виконанням нового аналізу.

Перевірка наявності необхідних даних: `if(SelectedMaterial != null && SelectedMaterial.PriceByYears != null && SelectedMaterial.PriceByYears.Count > 1)`.

Переконається, що вибраний матеріал (`SelectedMaterial`) існує та існує словник `PriceByYears` із цінами за роки.

Даних щонайменше два роки (`PriceByYears.Count > 1`), щоб можна було побудувати трендову модель.

Підготовка даних для обчислення лінійної регресії: визначення базового року (`baseYear`), тобто найменшого значення ключа (першого року у вибірці).

Формування списків:

- `years` – список років у вигляді (`рік - baseYear`), щоб зробити розрахунки відносно першого року;
- `prices` – список відповідних цін матеріалу за кожен рік.

Обчислення статистичних величин:

- `n = years.Count` – кількість точок у вибірці;
- `sumX` – сума всіх значень `years`;
- `sumY` – сума всіх значень `prices`;
- `sumXY` – сума добутків `years[i] * prices[i]`;
- `sumX2` – сума квадратів `years[i]`.

Обчислення коефіцієнтів рівняння лінійної регресії.

Використовується метод найменших квадратів для знаходження рівняння прямої:

$$Y = a + bX, \quad (3.1)$$

де b – нахил трендової прямої;

a – вільний член рівняння.

$$b = (n \cdot \text{sum}XY - \text{sum}X \cdot \text{sum}Y) / (n \cdot \text{sum}X_2 - \text{sum}X \cdot \text{sum}X), \quad (3.2)$$

де n – кількість точок у вибірці;

$\text{sum}X$ – сума всіх значень `years`;

$\text{sum}Y$ – сума всіх значень `prices`;

$\text{sum}XY$ – сума добутків `years[i] * prices[i]`;

$\text{sum}X_2$ – сума квадратів `years[i]`.

$$a = (\text{sum}Y - b \cdot \text{sum}X) / n \quad (3.3)$$

Формується рівняння тренду: `Advice_Message += $"Рівняння тренду: Y = {a} + {b}X\n"`.

Формування рекомендації щодо інвестування.

Якщо $b > 0$, тренд показує зростання ціни матеріалу: виводиться повідомлення "Ціна на матеріал зростає". Рекомендується інвестування ("Рекомендується інвестувати").

Якщо $b \leq 0$, тренд показує спад ціни: виводиться повідомлення "Ціна на матеріал спадає". Інвестування не рекомендується.

Обробка випадку недостатньої кількості даних.

Якщо `PriceByYears` містить 1 або 0 записів, або `SelectedMaterial` дорівнює `null`, виводиться повідомлення: "Недостатньо даних для аналізу".

3.1.9 Алгоритм додавання годинника до списку «Обраного»

Мета:

Метод додає годинник (`Wristwatch`) до списку вподобаних (`liked list`) користувачем (`customer`).

Послідовність виконання коду наведена далі.

Відкриття з'єднання з базою даних: використовується `using (var context = new WatchDataBaseEntities())`, що забезпечує автоматичне закриття підключення після завершення операцій.

Обробка помилок через `try-catch`: якщо виникне виняток під час виконання операцій (наприклад, база даних недоступна або `customerId/wristwatchId` некоректні), повідомлення про помилку буде виведене користувачеві.

Отримання користувача з його вподобаними годинниками: `var user = context.User.Include("Wristwatch").FirstOrDefault(u => u.UserId == customerId)`.

Використовується `Include("Wristwatch")`, щоб одразу завантажити список годинників, які вподобав користувач.

`FirstOrDefault(u => u.UserId == customerId)` шукає користувача за `customerId`.

Отримання годинника за `wristwatchId`: `var watch = context.Wristwatch.Find(wristwatchId)`.

Пошук годинника в базі даних за унікальним ідентифікатором.

Перевірка, чи знайдені об'єкти: якщо `watch != null && user != null`, тобто і користувач, і годинник існують у базі даних, виконуються наступні кроки.

Додавання годинника у список вподобаних користувача: `user.Wristwatch.Add(watch)`.

Додає знайдений годинник у колекцію вподобаних об'єктів користувача.

Визначено, що між користувачем (`User`) і годинниками (`Wristwatch`) є зв'язок типу `many-to-many`.

Збереження змін у базі даних: `context.SaveChanges()`.

Фактично зберігає доданий годинник у списку вподобаних.

Виведення повідомлення про успішну операцію: `MessageBox.Show("Wristwatch was added into liked list")`.

3.2 Вибір шаблону проєктування

Для розробки програмного забезпечення було використано шаблон подібний до архітектури проєктування MVVM.

Шаблон проєктування MVVM (Model-View-ViewModel) – це архітектурний шаблон, який використовується в програмуванні для розділення обов'язків між компонентами додатка [12]. Він спеціально розроблений для розробки програмних додатків з графічним інтерфейсом користувача, таких як десктопні застосунки.

Шаблон MVVM складається з трьох основних компонентів Model, View, ViewModel.

Model (Модель):

- представляє бізнес-логіку або дані додатка;
- не залежить від представлення (View) або ViewModel;
- може містити логіку для доступу до даних (наприклад, отримання даних з бази даних);
- зазвичай містить класи, структури або інші об'єкти, що представляють сутності програми.

View (Представлення):

- відповідає за відображення інтерфейсу користувача (UI);
- може бути реалізований у вигляді декларативного розмітки (наприклад, XAML в WPF або XML в Android);
- не має власної логіки і намагається бути максимально простим інтерфейсом для взаємодії з користувачем.

ViewModel (Модель-подання):

- посередник між Model та View;
- містить логіку, яка пов'язує дані Model з представленням View;
- не містить безпосередньої залежності від View, що робить його тестування і повторне використання більш простим;

- використовує механізми спостереження або прив'язки для оновлення представлення при зміні даних в Model;
- може містити команди, які виконують дії відповідно до дій користувача.

Основна ідея MVVM – розділення бізнес-логіки та інтерфейсу користувача для полегшення розробки, тестування та підтримки коду. Крім того, цей шаблон дозволяє робити різні представлення для одних і тих же даних (наприклад, десктопний інтерфейси) без зміни логіки додатка.

Принцип взаємодії компонентів у архітектурному шаблоні MVVM зображено на рисунку 3.2.

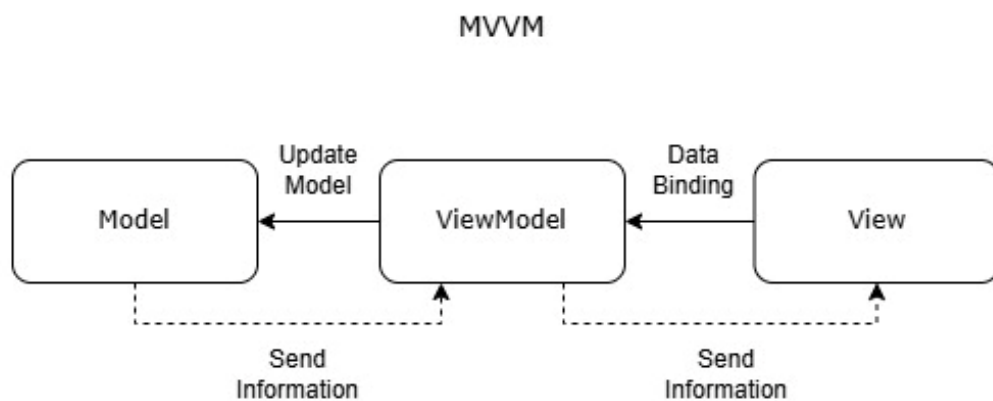


Рисунок 3.2 – Взаємодія компонентів у архітектурному шаблоні MVVM

Для прив'язки даних (Data Binding) у класах моделі-представлення було використано та реалізовано інтерфейс `INotifyPropertyChanged` та його метод `OnPropertyChanged()` [15].

`INotifyPropertyChanged` – це інтерфейс у .NET, який використовується для повідомлення про зміни властивостей об'єкта. Він особливо корисний у шаблонах MVVM (Model-View-ViewModel), де зміни в моделі (або ViewModel) повинні автоматично відображатися у прив'язаному до неї інтерфейсі користувача.

Ключові особливості інтерфейсу `INotifyPropertyChanged` перелічено далі.

Подія `PropertyChanged`:

- визначає, що в об'єкта змінилося значення певної властивості;
- використовується механізмом прив'язки (Binding) у WPF та WinForms для автоматичного оновлення UI при зміні значень.

Метод `OnPropertyChanged`:

- викликає подію `PropertyChanged`, передаючи їй назву властивості, що змінилася;
- забезпечує механізм сповіщення про зміни у об'єкті програми.

3.3 Опис модулів розроблюваної системи

Для реалізації програмного забезпечення було використано класи, щоб побудувати структуру програми, що містять поля, властивості та методи [16]. Структурну схему класів програми побудовану за допомогою засобів Visual Studio Community 2022 зображено на рисунку 3.3.

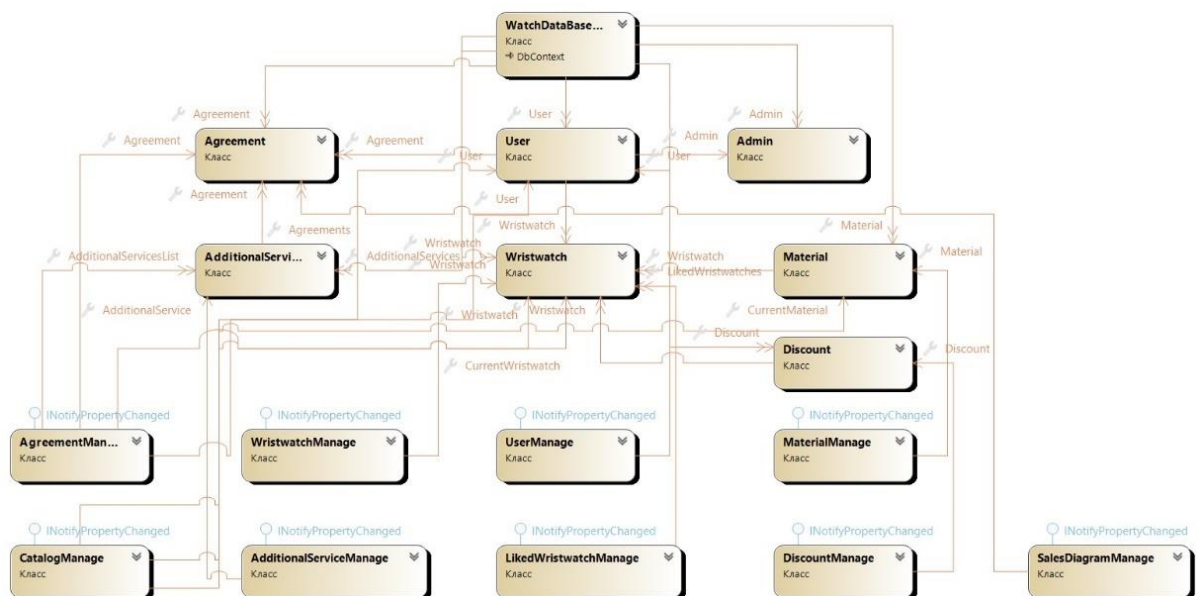


Рисунок 3.3 – Структурна схема класів програми

3.3.1 Клас WatchDataBaseEntities

Клас WatchDataBaseEntities відповідає за контекст бази даних в додатку, що працює з Entity Framework. Він визначає набори сутностей (таблиці) і конфігурації моделі бази даних. Нижче наведено його опис на основі наведеного прикладу.

Поля та методи класу наведено у таблиці 3.1.

Таблиця 3.1 – Опис полів та методів класу WatchDataBaseEntities

Поля і методи класу	Опис
Public:	
DbSet<AdditionalServices> AdditionalServices	Набір сутностей, що представляє таблицю додаткових послуг до угоди продажу годинників
DbSet<Admin> Admin	Набір сутностей, що представляє таблицю адміністраторів сервісу
DbSet<Agreement> Agreement	Набір сутностей, що представляє таблицю угод про продаж годинників
DbSet<Discount> Discount	Набір сутностей, що представляє таблицю знижок на ціну годинників
DbSet<Material> Material	Набір сутностей, що представляє таблицю матеріалів з яких виготовлені годинники
DbSet<User> User	Набір сутностей, що представляє таблицю користувачів системи
DbSet<Wristwatch> Wristwatch	Набір сутностей, що представляє таблицю наручних годинників
WatchDataBaseEntities()	Конструктор контексту бази даних, що ініціалізує контекст
Protected:	
override void OnModelCreating(DbModelBuilder modelBuilder)	Метод, що ініціалізує моделі сутностей та їх ключі таблиць у програмі

3.3.2 Клас DataBaseManage

Клас DataBaseManage містить методи для роботи з базою даних, включаючи додавання, зчитування, оновлення та видалення даних про користувачів, наручних годинників, угод продажу, матеріалів, знижок, додаткових послуг, списку обраних користувачами годинників.

Методи класу наведено у таблиці 3.2.

Таблиця 3.2 – Опис методів класу DataBaseManage

Методи класу	Опис
1	2
Public:	
void Add_User_To_DB(User user)	Метод, що додає користувача до БД після його реєстрації
User Get_User_From_DB(string login)	Метод, що виконує пошук користувача у БД та повертає його
void Update_User_In_DB(User userUpdate)	Метод, що виконує оновлення користувача у БД
void Delete_User_From_DB(User user)	Метод, що виконує видалення користувача з БД
void Add_Admin_To_DB(Admin admin)	Метод, що виконує додавання адміністратора у БД
Admin Get_Admin_From_DB(int adminId)	Метод, що виконує пошук адміністратора у БД
void Add_Wristwatch_To_DB(Wristwatch wristwatch)	Метод, що додає новий наручний годинник у БД
List<Wristwatch> Get_SoldWristwatches_From_DB()	Метод, що повертає усі годинники, які наявні у БД
List<Wristwatch> Get_ActualWristwatches_From_DB()	Метод, що виконує пошук непроданих годинників та повертає їх

Продовження таблиці 3.2.

1	2
List<Wristwatch> Get_CatalogWristwatches_From_DB()	Метод, що повертає годинники із БД для каталогу
public void Update_Wristwatch_In_DB(Wristwatch updateWristwatch)	Метод, що оновлює необхідний годинник у БД
void Delete_Wristwatch_From_DB(Wristwatch wristwatch)	Метод, що видаляє необхідний годинник із БД
void Add_Material_To_DB(Material material)	Метод, що додає новий матеріал до БД
List<Material> Get_Materials_From_DB()	Метод, що повертає матеріали з БД
void Update_Material_In_DB(Material updateMaterial)	Метод, що оновлює необхідний матеріал у БД
void Delete_Material_From_DB(Material material)	Метод, що видаляє необхідний матеріал з БД
void Add_Discount_To_DB(Discount discount)	Метод, що додає нову знижку до БД
Discount Get_Discount_From_DB(int? id)	Метод, що отримує необхідну знижку по Id з БД
List<Discount> Get_Discounts_From_DB()	Метод, що отримує усі знижки з БД
void Edit_Discount_In_DB(Discount updateDiscount)	Метод, що оновлює необхідну знижку у БД
void Delete_Discount_From_DB(Discount discount)	Метод, що видаляє необхідну знижку з БД
void Add_LikedWatch_To_DB(int wristwatchId, int customerId)	Метод, що додає Id годинника до таблиці «Обране» користувача
List<Wristwatch> Get_LikedWristwatches_From_DB(int customerId)	Метод, що виконує пошук «обраних» годинників користувачем
bool Check_LikedWristwatch_In_DB(int customerId, int wristwatchId)	Метод, що перевіряє наявність обраного годинника у БД

Продовження таблиці 3.2.

1	2
void Delete_LikedWristwatch_From_DB(int userId, Wristwatch wristwatch)	Метод, що видаляє необхідний годинник зі списку «Обране»
void Add_Agreement_To_DB(Agreement agreement)	Метод, що додає нову угоду про продаж годинника у БД
List<Agreement> Get_Agreements_From_DB()	Метод, що повертає список угод, підтверджених адміністратором
List<Agreement> Get_Agreements_For_Submit_From_DB()	Метод, що повертає список непідтверджених угод
List<Agreement> Get_AgreementsHistory_For_User_From_DB(User customer, int pageNumber, int pageSize)	Метод, що повертає список підтверджених угод для відображення у історії продажів годинників, для конкретного користувача
List<Agreement> Get_AgreementHistory_From_DB(User admin, int pageNumber, int pageSize)	Метод, що повертає список усіх підтверджених угод
int Get_TotalPages_AgreementHistory(int pageSize, User user)	Метод, що розраховує кількість сторінок для пагінації списку історії угод
int Get_TotalPages_AgreementHistory_For_Orders(int pageSize, User user)	Метод, що розраховує кількість сторінок для списку замовлень
void Set_Number_Of_Deals_For_Admin(Admin admin)	Метод, що збільшує кількість угод адміністратора, при підтвердженні нової угоди продажу
void Set_Sales_Amount_For_Admin(Admin admin, Agreement agreement)	Метод, що збільшує суму продажів годинників адміністратором
List<Agreement> Get_Orders_For_User_From_DB(User customer, int pageNumber, int pageSize)	Метод, що повертає усі замовлення для конкретного користувача

Продовження таблиці 3.2

1	2
List<Agreement> Get_Admin_AgreementHistory_From_DB(User admin, int pageNumber, int pageSize)	Метод, що повертає історію продажів годинників для конкретного адміністратора
void Update_Agreement_For_Submit(Agreement agreementUpdate)	Метод, що оновлює угоду, як підтверджену адміністратором
void Add_AdditionalService_To_DB(AdditionalServices additionalService)	Метод, що додає додаткову послугу у БД
List<AdditionalServices> Get_AdditionalServices_From_DB()	Метод, що повертає список додаткових послуг до угоди про продаж з БД
void Update_AdditionalService_In_DB(AdditionalServices additionalServiceUpdate)	Метод, що оновлює необхідну додаткову послугу до угоди у БД
void Delete_AdditionalService_From_DB(AdditionalServices additionalService)	Метод, що видаляє необхідну додаткову послугу з БД

3.3.3 Клас User

Клас User представляє користувача системи та є частиною об'єктної моделі бази даних.

Поля класу наведені у таблиці 3.3.

Таблиця 3.3 – Опис полів класу User

Поля	Опис
1	2
Public:	
int UserId	Поле, що визначає Id користувача
string FirstName	Поле, що визначає ім'я користувача

Продовження таблиці 3.3

1	2
string LastName	Поле, що визначає прізвище користувача
string Login	Поле, що визначає логін користувача
string Password	Поле, що визначає пароль користувача
string PhoneNumber	Поле, що визначає номер телефону користувача
string Email	Поле, що визначає електронну пошту користувача
string Status	Поле, що визначає роль користувача
DateTime DateOfBirth	Поле, що визначає дату народження користувача
Admin Admin	Навігаційна властивість, що визначає зв'язок з таблицею Admin
List<Wristwatch> Wristwatch	Навігаційна властивість, що визначає зв'язок з таблицею Wristwatch
List<Agreement> Agreement	Навігаційна властивість, що визначає зв'язок з таблицею Agreement

3.3.4 Клас Admin

Клас Admin представляє адміністратора системи та є частиною об'єктної моделі бази даних.

Поля класу наведені у таблиці 3.4.

Таблиця 3.4 – Опис полів класу Admin

Поле	Опис
1	2
Public:	
int AdminId	Поле, що визначає Id адміністратора
int NumberOfDeals	Поле, що визначає кількість угод про продаж годинників адміністратором

Продовження таблиці 3.4

1	2
double SalesAmount	Поле, що визначає вартість продажів угод підтверджених адміністратором
User User	Навігаційна властивість, що визначає зв'язок з таблицею User
ICollection<Agreement> Agreement	Навігаційна властивість, що визначає зв'язок з таблицею Agreement

3.3.5 Клас Wristwatch

Клас Wristwatch представляє наручний годинник та є частиною об'єктної моделі бази даних.

Поля класу наведені у таблиці 3.5.

Таблиця 3.5 – Опис полів класу Wristwatch

Поля	Опис
1	2
Public:	
int WatchId	Поле, що визначає Id наручного годинника
string Model	Поле, що визначає модель годинника
string Brand	Поле, що визначає бренд годинника
string Color	Поле, що визначає колір годинника
string Image	Поле, що визначає назву зображення годинника
double Weight	Поле, що визначає вагу годинника
int MaterialWatchId	Поле, що визначає Id матеріалу з якого виготовлено годинник
string Glass	Поле, що визначає скло годинника
string Strap	Поле, що визначає ремінець годинника
double Price	Поле, що визначає вартість годинника

Продовження таблиці 3.5.

1	2
int DiscountId	Поле, що визначає знижку на ціну годинника
ICollection<Agreement> Agreement	Навігаційна властивість, що визначає зв'язок з таблицею Agreement
Discount Discount	Навігаційна властивість, що визначає зв'язок з таблицею Discount
Material Material	Навігаційна властивість, що визначає зв'язок з таблицею Material
ICollection<User> User	Навігаційна властивість, що визначає зв'язок з таблицею User

3.3.6 Клас Agreement

Клас Agreement представляє угоду про продаж наручного годинника та є частиною об'єктної моделі бази даних.

Поля класу наведені у таблиці 3.6.

Таблиця 3.6 – Опис полів класу Agreement

Поля	Опис
1	2
Public:	
int AgreementId	Поле, що визначає Id угоди про продаж годинника
int WatchId	Поле, що визначає Id годинника
int CustomerId	Поле, що визначає Id клієнта
int AdminId	Поле, що визначає Id адміністратора
double Price	Поле, що визначає вартість угоди
DateTime DateOfAgreement	Поле, що визначає дату укладення угоди
string Safety	Поле, що визначає наявність гарантії на годинник

Продовження таблиці 3.6.

1	2
int AdditionalServiceId	Поле, що визначає Id додаткових послуг до угоди
AdditionalServices AdditionalServices	Навігаційна властивість, що визначає зв'язок з таблицею AdditionalServices
Admin Admin	Навігаційна властивість, що визначає зв'язок з таблицею Admin
Wristwatch Wristwatch	Навігаційна властивість, що визначає зв'язок з таблицею Wristwatch
User User	Навігаційна властивість, що визначає зв'язок з таблицею User

3.3.7 Клас Material

Клас Material представляє матеріал з якого виготовлений наручний годинник та є частиною об'єктної моделі бази даних.

Поля класу наведені у таблиці 3.7.

Таблиця 3.7 – Опис полів класу Material

Поле	Опис
Public:	
int MaterialId	Поле, що визначає Id матеріалу
string MaterialName	Поле, що визначає назву матеріалу
string Description	Поле, що визначає опис матеріалу
double CurrentCost	Поле, що визначає вартість матеріалу
ICollection<Wristwatch> Wristwatch	Навігаційна властивість, що визначає зв'язок з таблицею Wristwatch

3.3.8 Клас Discount

Клас Discount представляє знижку на ціну годинника та є частиною об'єктної моделі бази даних.

Поля класу наведені у таблиці 3.8.

Таблиця 3.8 – Опис полів класу Discount

Поле	Опис
Public:	
int DiscountId	Поле, що визначає Id знижки
string ContentDiscount	Поле, що визначає опис знижки
int DiscountValue	Поле, що визначає відсоток знижки
ICollection<Wristwatch> Wristwatch	Навігаційна властивість, що визначає зв'язок з таблицею Wristwatch

3.3.9 Клас AdditionalServices

Клас AdditionalServices представляє додаткові послуги до угоди про продаж годинника та є частиною об'єктної моделі бази даних.

Поля класу наведені у таблиці 3.9.

Таблиця 3.9 – Опис полів класу AdditionalServices

Поле	Опис
Public:	
int ServiceId	Поле, що визначає Id додаткової послуги
string ServiceName	Поле, що визначає назву додаткової послуги
double PriceService	Поле, що визначає вартість додаткової послуги
ICollection<Agreement> Agreement	Навігаційна властивість, що визначає зв'язок з таблицею Agreement

3.3.10 Клас UserManage

Клас UserManage представляє функціонал для управління користувачами сервісу.

Поля та методи класу наведено у таблиці 3.10.

Таблиця 3.10 – Опис полів та методів класу UserManage

Поле та метод	Опис
Public:	
User User	Поле, що визначає користувача системи
string Login	Поле, що визначає логін користувача
string Password	Поле, що визначає пароль користувача
bool IsPasswordHidden	Поле, що визначає чи приховувати пароль
User CurrentUpdateUser	Поле, що визначає поточного користувача для оновлення
List<string> StatusList	Список, що визначає доступні ролі користувачів (адміністратор або клієнт)
DataBaseManage dataBaseManage	Об'єкт, що визначає взаємодію з базою даних
UserManage()	Конструктор класу
void Registration_User()	Метод, що реєструє користувача у системі
User Get_User()	Метод, що повертає об'єкт користувача
User Get_User(string login)	Метод, що повертає об'єкт користувача за логіном
Admin Get_Admin(User user)	Метод, що повертає об'єкт адміністратора, для відображення його функціоналу
bool Authorazation_User()	Метод, що виконує авторизацію користувача
void Edit_User()	Метод, що редагує користувача
void DeleteUser()	Метод, що видаляє користувача
void Registration_Admin(int adminId)	Метод, що додає адміністратора у таблицю адміністраторів

3.3.11 Клас WristwatchManage

Клас WristwatchManage представляє функціонал для управління наручними годинниками.

Поля та методи класу наведено у таблиці 3.11.

Таблиця 3.11 – Опис полів та методів класу WristwatchManage

Поле та метод	Опис
Public:	
Wristwatch Wristwatch	Поле, що визначає об'єкт наручного годинника
List<Wristwatch> Wristwatches	Список, що визначає наручні годинники
List<Material> MaterialsList	Список, що визначає матеріали з яких виготовлені годинники
List<Discount> DiscountsList	Список, що визначає знижки на ціну годинників
DataBaseManage dataBaseManage	Об'єкт, що визначає взаємодію з базою даних
WristwatchManage()	Конструктор класу
void Add_Wristwatch()	Метод, що виконує додавання наручного годинника
void Show_Actual_Wristwatches()	Метод, що відображає список непроданих наручних годинників
void Show_Sold_Wristwatches()	Метод, що відображає усі наручні годинники
void Edit_Wristwatch()	Метод, що редагує необхідний наручний годинник
void Delete_Wristwatch(Wristwatch wristwatch)	Метод, що видаляє наручний годинник
void Open_Image()	Метод, що відкриває необхідне зображення при додаванні нового наручного годинника

3.3.12 Клас AgreementManage

Клас AgreementManage представляє функціонал для управління угодами про продаж наручних годинників.

Поля та методи класу наведено у таблиці 3.12.

Таблиця 3.12 – Поля та методи класу AgreementManage

Поле та метод	Опис
1	2
Public:	
Agreement Agreement	Поле, що визначає об'єкт угоди про продаж годинника
Wristwatch Wristwatch	Поле, що визначає об'єкт наручного годинника
User User	Поле, що визначає об'єкт користувача сервісу
BitmapImage WristwatchImage	Поле, що визначає об'єкт зображення наручного годинника
List<Agreement> Agreements	Список, що визначає угоди про продаж годинників
int PageNumber	Поле, що визначає номер поточної сторінки для пагінації
int PageSize	Поле, що визначає кількість елементів на сторінці
double CurrentWristwatchPrice	Поле, що визначає поточну вартість годинника
double SafetyPrice	Поле, що визначає вартість страховки годинника
double AdditionalServicePrice	Поле, що визначає вартість додаткового сервісу до угоди
List<string> SafetyList	Список, що визначає доступні варіанти страховки

Продовження таблиці 3.12

1	2
List<AdditionalService> AdditionalServicesList	Список, що визначає доступні додаткові сервіси
Discount Discount	Поле, що визначає об'єкт знижки на ціну годинника
DataBaseManage dataBageManage	Об'єкт, що визначає взаємодію з базою даних
AgreementManage(User customer, Wristwatch wristwatch)	Конструктор класу для клієнта
AgreementManage(User user)	Конструктор класу
void Add_Agreement()	Метод, що додає новий годинник
void Safety_Wristwatch_Price(string safety)	Метод, що розраховує ціну угоди зі страховкою годинника
void AdditionalService_Wristwatch_Price(AdditionalServices additionalService)	Метод, що розраховує ціну угоди з додатковими послугами
void Get_Agreements_For_Submit()	Метод, що відображає угоди для підтвердження адміністратором
void Submit_Agreement(Agreement agreement)	Метод, що підтверджує угоду
void Show_AgreementHistory()	Метод, що відображає історію продажів годинників
void Show_AgreementHistory_For_Admin()	Метод, що відображає історію продажів для адміністратора
void Show_Orders()	Метод, що відображає замовлення користувача
void Show_AgreementHistory_By_Admin()	Метод, що відображає історію продажів за адміністратором
void Back_AgreementHistory_User_Page()	Метод, що переходить на попередню сторінку при перегляді історії користувачем
void Next_AgreementHistory_User_Page()	Метод, що переходить на наступну сторінку при перегляді історії користувачем

Продовження таблиці 3.12.

1	2
<code>void Back_AgreementHistory_Page()</code>	Метод, що переходить на попередню сторінку при перегляді історії адміністратором
<code>void Next_AgreementHistory_Page()</code>	Метод, що переходить на наступну сторінку при перегляді історії адміністратором
<code>void Back_AgreemntHistory_Orders_Page()</code>	Метод, що переходить на попередню сторінку при перегляді замовлень клієнтом
<code>void Next_AgreementHistory_Orders_Page()</code>	Метод, що переходить на наступну сторінку при перегляді замовлень клієнтом
<code>void Back_Admin_AgreementHistory_Page()</code>	Метод, що переходить на попередню сторінку при перегляді історії продажів за адміністратором
<code>void Next_Admin_AgreementHistory_Page()</code>	Метод, що переходить на наступну сторінку при перегляді історії продажів за адміністратором

3.3.13 Клас CatalogManage

Клас CatalogManage представляє функціонал для управління каталогом доступних для продажу годинників.

Поля та методи класу наведено у таблиці 3.13.

Таблиця 3.13 – Опис полів та методів класу CatalogManage

Поле та метод	Опис
Public:	
public User User	Поле, що визначає об'єкт користувача, що переглядає каталог
List<Wristwatch> wristwatches	Список, що визначає доступні для продажу наручні годинники
Wristwatch CurrentWristwatch	Поле, що визначає об'єкт поточного наручного годинника, що відображається на формі
Material CurrentMaterial	Поле, що визначає об'єкт матеріалу з якого виготовлений поточний годинник
BitmapImage CurrentImage	Поле, що визначає об'єкт зображення наручного годинника
int indexWristwatch	Поле, що визначає індекс поточного годинника
DataBaseManage dataBaseManage	Об'єкт, що визначає взаємодію з базою даних
CatalogManage(User user)	Конструктор класу для відображення каталогу починаючи з першого доступного годинника
CatalogManage(User user, Wristwatch wristwatch)	Конструктор класу для відображення каталогу з конкретного годинника
void Get_Catalog()	Метод, що повертає список доступних для продажу годинників
static BitmapImage getImage(string name)	Статичний метод, що повертає зображення для наручного годинника за його назвою
void Next_Wristwatch()	Метод, що переходить до наступного годинника у каталозі
void Back_Wristwatch()	Метод, що переходить до попереднього годинника у каталозі
void Add_Liked_Wristwatch()	Метод, що додає необхідний годинник до списку «Обране» користувача

3.3.14 Клас MaterialManage

Клас MaterilaManage представляє функціонал для управління матеріалами з яких виготовлено наручні годинники.

Поля та методи класу наведено у таблиці 3.14.

Таблиця 3.14 – Опис полів та методів класу MaterialManage

Поле та метод	Опис
Public:	
Material Material	Поле, що визначає об'єкт матеріалу, з якого виготовлено годинник
List<Material> Materials	Список, що визначає матеріали
DataBaseManage dataBaseManage	Об'єкт, що визначає взаємодію з базою даних
MaterialManage()	Конструктор класу
void Add_Material()	Метод, що додає новий матеріал
void Edit_Material()	Метод, що редагує матеріал
void Delete_Material()	Метод, що видаляє матеріал

3.3.15 Клас DiscountManage

Клас DiscountManage представляє функціонал управління знижками на наручні годинники.

Поля та методи класу наведено у таблиці 3.15.

Таблиця 3.15 – Опис полів та методів класу DiscountManage

Поле та метод	Опис
1	2
Public:	
Discount Discount	Поле, що визначає об'єкт знижки на годинник

Продовження таблиці 3.15.

1	2
List<Discount> Discounts	Список, що визначає знижки на ціну
DataBaseManage dataBaseManage	Об'єкт, що визначає взаємодію з базою даних
DiscountManage()	Конструктор класу
void Add_Discount()	Метод, що додає знижку
void Edit_Discount()	Метод, що редагує знижку
void Delete_Discount()	Метод, що видаляє знижку

3.3.16 Клас AdditionalServiceManage

Клас AdditionalServiceManage представляє функціонал для управління додатковими послугами до угоди про продаж наручного годинника.

Поля та методи класу наведено у таблиці 3.16.

Таблиця 3.16 – Опис полів та методів класу AdditioanalServiceManage

Поле та метод	Опис
Public:	
AdditionalServices AdditionalService	Поле, що визначає об'єкт додаткової послуги до угоди про продаж годинника
List<AdditionalServices> AdditionalServices	Список, що визначає додаткові послуги
DataBaseManage dataBaseManage	Об'єкт, що визначає взаємодію з базою даних
AdditionalServiceManage()	Конструктор класу
void Add_AddtionalService()	Метод, що додає додаткову послугу
void Edit_AdditionalService()	Метод, що редагує додаткову послугу
void Delete_AdditionalService(AdditionalServices additionalService)	Метод, що видаляє додаткову послугу

3.3.17 Клас LikedWristwatchManage

Клас LikedWristwatchManage представляє функціонал для управління наручними годинниками у списку «Обране» користувача.

Поля та методи класу наведено у таблиці 3.17.

Таблиця 3.17 – Опис полів та методів класу LikedWristwatchManage

Поле та метод	Опис
Public:	
List<Wristwatch> LikedWristwatches	Список, що визначає «обрані» користувачем годинники
DataBaseManage dataBaseManage	Об'єкт, що визначає взаємодію з базою даних
LikedWristwatchManage()	Конструктор класу
void Show_LikedWristwatches(User user)	Метод, що відображає список «обраних» користувачем годинників
void Delete_LikedWristwatch(User user, Wristwatch wristwatch)	Метод, що видаляє годинник зі списку «обраного»

3.3.18 Клас SalesDiagramManage

Клас SalesDiagramManage представляє функціонал для управління статистикою продажів наручних годинників.

Поля та методи класу наведено у таблиці 3.18.

Таблиця 3.18 – Опис полів та методів класу SalesDiagramManage

Поле та метод	Опис
1	2
Public:	
List<Agreement> Agreements	Список, що визначає список угод про продаж годинників

Продовження таблиці 3.18.

1	2
SeriesCollection WristwatchSales	Колекція для відображення статистики у вигляді діаграми
DataBaseManage dataBaseManage	Об'єкт, що визначає взаємодію з базою даних
SalesDiagramManage()	Конструктор класу
void Get_Statistics_For_Sales_By_Brand()	Метод, що розраховує статистику продажів за брендом годинників
void Get_Statistics_For_Sales_By_Material()	Метод, що розраховує статистику продажів за матеріалом з якого виготовлено годинник
void Get_Statistics_For_Sales_By_DateAgreement()	Метод, що розраховує статистику продажів за місяцем укладання угоди про продаж годинника

3.3.19 Клас MaterialsStatisticManage

Клас MaterialsStatisticManage представляє функціонал для управління статистикою курсу вартості матеріалів, з яких виготовлені годинники, та аналізу чи варто інвестувати у цей матеріал.

Поля та методи класу наведено у таблиці 3.19.

Таблиця 3.19 – Опис полів та методів класу MaterialsStatisticManage

Поле та метод	Опис
1	2
Public:	
List<MaterialDto> Materials	Список, що визначає матеріали, курс яких буде відображено у статистиці
MaterialDto SelectedMaterial	Поле, що визначає об'єкт обраного матеріалу для відображення діаграми його курсу вартості

Продовження таблиці 3.19.

1	2
SeriesCollection ChartSeries	Колекція для відображення статистики курсу вартості матеріалу на діаграмі
List<string> Labels	Список, що визначає підписи для матеріалів на діаграмі
string Advice_Message	Поле, що визначає повідомлення про те, чи варто інвестувати у матеріал
string File_Path	Поле, що визначає шлях до файлу з курсом вартості матеріалів
MaterialsStatisticManage()	Конструктор класу
void Get_MaterialsPrice_From_JSON()	Метод, що зчитує інформацію про вартість матеріалів з json-файлу
void UpdateChart()	Метод, що оновлює діаграму
void Calculate_Statistic()	Метод, що розраховує коефіцієнти рівняння лінійної регресії на фоні даних про зміну вартості матеріалу з json-файлу та визначає чи варто інвестувати у обраний користувачем матеріал

3.4 Проєктування бази даних

Для функціонування програмного забезпечення було розроблено базу даних на сервері Microsoft SQL Server, яка зберігає усю інформацію для роботи сервісу електронної комерції інтернет-продажів наручних годинників. База даних містить вісім таблиць, кожна з яких має стовпці різних типів даних.

3.4.1 Таблиця User

Таблиця User представляє сутність користувачів системи та містить поля опису особистих даних користувача. Пов'язана з таблицею Admin зв'язком один до одного за полем UserId. Також має зв'язок один до багатьох

з таблицею Agreement та зв'язок багато до багатьох з таблицею Wristwatch через проміжну таблицю LikedWatch.

Опис полів сутності наведено у таблиці 3.20.

Таблиця 3.20 – Опис полів сутності User

№	Назва поля	Тип даних	Розмір
1	UserId	int	-
2	FirstName	nvarchar	75
3	LastName	nvarchar	75
4	Login	nvarchar	75
5	Password	nvarchar	75
6	PhoneNumber	nvarchar	27
7	Email	nvarchar	75
8	Status	nvarchar	25
9	DateOfBirth	date	-

3.4.2 Таблиця Admin

Таблиця Admin представляє сутність адміністраторів системи та містить поля особистої статистики продажів адміністратора. Пов'язана з таблицею User зв'язком один до одного та зв'язком один до багатьох з таблицею Agreement.

Опис полів сутності наведено у таблиці 3.21.

Таблиця 3.21 – Опис полів сутності Admin

№	Назва поля	Тип даних	Розмір
1	AdminId	int	-
2	NumberOfDeals	int	-
3	SalesAmount	float	-

3.4.3 Таблиця Wristwatch

Таблиця Wristwatch представляє сутність наручних годинників для продажу у системі та містить поля їх опису та характеристик. Пов'язана з таблицею User зв'язком багато до багатьох через проміжну таблицю LikedWatch та зв'язком один до одного з таблицею Agreement, зв'язком багато до одного з таблицями Material та Discount.

Опис полів сутності наведено у таблиці 3.22.

Таблиця 3.22 – Опис полів сутності Wristwatch

№	Назва поля	Тип даних	Розмір
1	WatchId	int	-
2	Model	nvarchar	75
3	Brand	nvarchar	75
4	Color	nvarcahr	75
5	Image	nvarchar	175
6	Weight	float	-
7	MaterialWatchId	int	-
8	Glass	nvarchar	75
9	Strap	nvarchar	125
10	Price	float	-
11	DiscountId	int	-

3.4.4 Таблиця LikedWatch

Таблиця LikedWatch представляє сутність списку «Обраних» користувачем годинників. Є проміжною таблицею для реалізації зв'язку багато до багатьох між таблицями User та Wristwatch.

Опис полів сутності наведено у таблиці 3.23.

Таблиця 3.23 – Опис полів сутності LikedWatch

№	Назва поля	Тип даних	Розмір
1	CustomerId	int	-
2	WristwatchId	int	-

3.4.5 Таблиця Material

Таблиця Material представляє сутність матеріалу з якого виготовлений наручний годинник. Пов'язана з таблицею Wristwatch зв'язком один до багатьох.

Опис полів сутності наведено у таблиці 3.24.

Таблиця 3.24 – Опис полів сутності Material

№	Назва поля	Тип даних	Розмір
1	MaterialId	int	-
2	MaterialName	nvarchar	75
3	Description	nvarchar	275
4	CurrentCost	float	-

3.4.6 Таблиця Discount

Таблиця Discount представляє сутність знижки на ціну наручного годинника. Пов'язана з таблицею Wristwatch зв'язком один до багатьох.

Опис полів сутності наведено у таблиці 3.25.

Таблиця 3.25 – Опис полів сутності Discount

№	Назва поля	Тип даних	Розмір
1	DiscountId	int	-
2	ContentDiscount	nvarchar	275
3	DiscountValue	int	-

3.4.7 Таблиця Agreement

Таблиця Agreement представляє сутність угоди про продаж наручного годинника. Пов'язана з таблицею Wristwatch зв'язком один до одного та зв'язком багато до одного з таблицями User, Admin та AdditionalService.

Опис полів сутності наведено у таблиці 3.26.

Таблиця 3.26 – Опис полів сутності Agreement

№	Назва поля	Тип даних	Розмір
1	AgreementId	int	-
2	WatchId	int	-
3	CustomerId	int	-
4	Price	float	-
5	DateOfAgreement	date	-
6	Safety	nvarchar	35
7	AdditionalServiceid	int	-

3.4.8 Таблиця AdditionalServices

Таблиця AdditionalServices представляє сутність додаткової послуги до угоди про продаж наручного годинника. Пов'язана з таблицею Agreement зв'язком один до багатьох.

Опис полів сутності наведено у таблиці 3.27.

Таблиця 3.27 – Опис полів сутності AdditionalServices

№	Назва поля	Тип даних	Розмір
1	ServiceId	int	-
2	ServiceName	nvarchar	75
3	PriceService	float	-

3.5 Висновки з розділу

У розділі опис програми було описано модулі програмного забезпечення у вигляді класів програми, структуру їх побудови, розроблені основні алгоритми функціонування системи, обраний шаблон проєктування застосунку, а також розроблену модель бази даних для зберігання інформації про сервіс електронної комерції інтернет-продажів наручних годинників.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

У даному розділі розглянуто призначення програми, умови її виконання, процес виконання програми та повідомлення для користувача.

4.1 Призначення програмного забезпечення

Програмне забезпечення адміністрування електронної комерції інтернет-продажів наручних годинників надає менеджерам гнучкі інструменти для управління товарами, їх характеристиками та замовленнями. Вони мають змогу додавати, редагувати та видаляти товари, оновлювати їхні описи, фото, матеріали, з яких виготовлені годинники, та ціни, а також підтверджувати замовлення клієнтів. Важливою функцією є доступ до аналітичних даних, представлених у вигляді графіків і діаграм, що дозволяє ефективно відстежувати продажі та оцінювати їхні тенденції. Менеджери можуть переглядати історію продажів, що допомагає в аналізі попиту та прийнятті стратегічних рішень щодо розвитку бізнесу.

Для клієнтів система надає інтуїтивно зрозумілий каталог, у якому вони можуть переглядати годинники, додавати їх до списку «Обране», оформлювати замовлення та обирати додаткові послуги до угоди про продаж годинника.

Додатковою перевагою програмного забезпечення є модуль аналізу вартості дорогоцінного металу, що дозволяє оцінювати тенденції змін ринкових цін на матеріали, з яких виготовлені годинники. Ця функція є корисною для клієнтів, які розглядають покупку годинника як довгострокову інвестицію.

4.2 Умови виконання програми

4.2.1 Апаратні вимоги програми

Для забезпечення зручної роботи десктопного застосунку, створеного на WPF .NET Framework 4.8.1, апаратні вимоги до клієнтських комп'ютерів можуть бути такими як наведено далі.

Процесор (CPU):

- а) тип процесора: сучасний багатоядерний процесор з архітектурою x86 або x64;
- б) тактова частота: рекомендується процесор з тактовою частотою 2.0 GHz або вище;
- в) кількість ядер: процесор з 4 або більше ядрами для забезпечення багатозадачності та швидкого виконання складних операцій.

Оперативна пам'ять (RAM):

- а) мінімальний обсяг: 4 GB RAM для базової роботи;
- б) рекомендований обсяг: 8 GB або більше для забезпечення плавної роботи, особливо при використанні ресурсозатратних функцій або великих обсягів даних.

Графічна підсистема:

- а) відеокарта: сучасна відеокарта з підтримкою DirectX 11 або вище;
- б) пам'ять відеокарти: мінімум 1 GB відеопам'яті, але для складніших графічних інтерфейсів рекомендується 2 GB і більше;
- в) роздільна здатність екрану: підтримка роздільної здатності 1920x1080 (Full HD) або вище для комфортного перегляду.

Дискова підсистема:

- а) тип накопичувача: твердотільний накопичувач (SSD) для швидкого завантаження операційної системи та застосунків.
- б) вільний простір: мінімум 3 GB вільного дискового простору для встановлення та роботи застосунку. Більше місця може знадобитися для зберігання даних.

Операційна система:

- а) сумісні ОС: Windows 10 (64-bit) або Windows 11 (64-bit);
- б) оновлення ОС: операційна система повинна бути оновлена.

4.2.2 Вимоги до користувача

Для запуску програмного забезпечення користувачу потрібно дотриматись усіх апаратних вимог. Далі треба запустити виконуваний файл з розширенням .exe для роботи програми. Користувач повинен авторизуватись використовуючи свої авторизаційні дані (логін та пароль), або, якщо він ще не має акаунту створити його.

Адміністратор може додавати та редагувати інформацію про сервіс, тому йому треба бути уважним при введенні інформації у відповідні елементи керування.

Клієнт може укласти угоди про покупку наручного годинника та надання додаткових послуг до угоди, отримувати розраховану статистику курсу вартості матеріалу з якого виготовлено годинник.

Програмне забезпечення має інтуїтивно зрозумілий графічний інтерфейс, тому користувач зможе опанувати зручну та комфортну роботу з програмним забезпеченням адміністрування електронної комерції інтернет-продажів наручних годинників.

4.3 Виконання програми

4.3.1 Запуск програми

Для запуску програмного забезпечення треба відкрити виконуваний файл «Shop_Wristwatch.exe» у папці «Shop_Wristwatch/bin/Debug». Запустивши програму користувач побачить авторизаційне вікно, де відображено назву програми, поля для введення авторизаційних даних, кнопки

авторизації та реєстрації. Зовнішній вигляд форми авторизації зображено на рисунку 4.1.

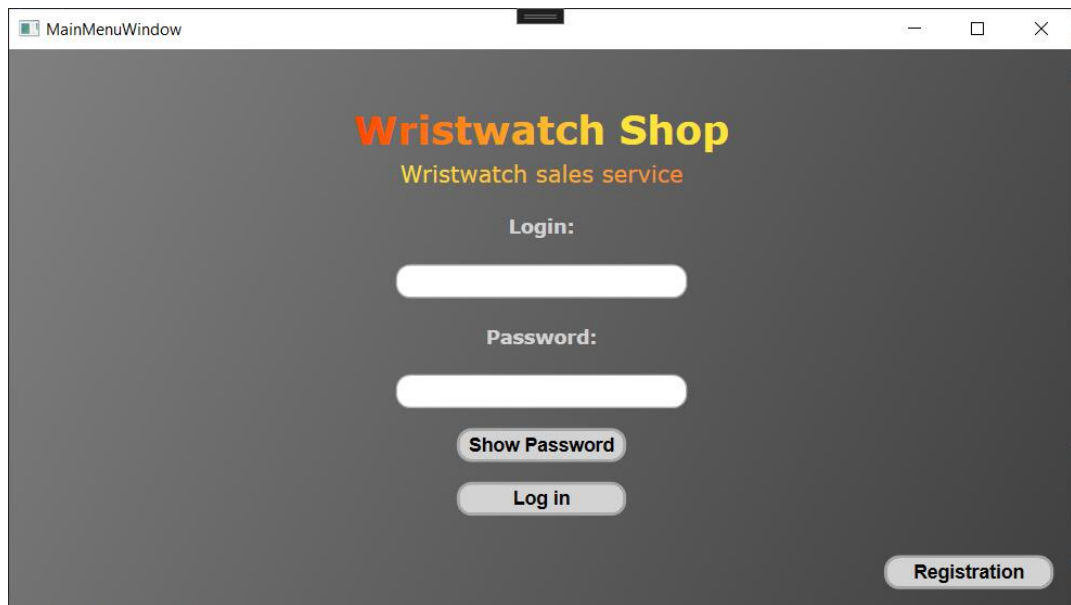
The image shows a software window titled 'MainMenuWindow'. The background is dark gray. At the top center, the text 'Wristwatch Shop' is displayed in a bold, orange-yellow font, with 'Wristwatch sales service' in a smaller, lighter orange font below it. In the center, there is a login section. It starts with the label 'Login:' in white. Below it is a white rectangular input field. Underneath the input field is the label 'Password:' in white. Below that is another white rectangular input field. To the right of this second input field is a small, rounded button labeled 'Show Password' in gray. Below the password input field is a rounded button labeled 'Log in' in gray. In the bottom right corner of the window, there is a rounded button labeled 'Registration' in gray.

Рисунок 4.1 – Зовнішній вигляд авторизаційної форми програми

Далі адміністратору треба перейти на форму реєстрації та внести свої реєстраційні дані для запису їх у системі, процес реєстрації зображено на рисунку 4.2.

The image shows a software window titled "RegistrationWindow" with a dark gray background. In the top-left corner, there is a "Back" button. The main heading "Registration" is centered at the top in a large, bold, orange font. Below the heading, there are several input fields, each preceded by a label: "First Name:" with the value "Dmytro", "Last Name:" with "Gray", "Login:" with "dimGray", "Password:" with "dim777gray", "Phone number:" with "+380975755757", "Email:" with "dimgray@gmail.com", and "Date of birth:" with a date picker showing "17.03.2004" and "15". Below these is a "Status:" label with a dropdown menu showing "Admin". At the bottom center is a "Registration" button.

Рисунок 4.2 – Процес реєстрації адміністратора у програмі

Далі адміністратор може авторизуватись у системі, використовуючи свої авторизаційні дані. Результат авторизації зображено на рисунку 4.3.

The image shows a software window titled "MainMenuWindow" with a dark gray background. The heading "Wristwatch Shop" is centered at the top in a large, bold, orange font, with the subtitle "Wristwatch sales service" below it in a smaller orange font. Below the heading, there are two input fields: "Login:" with the value "dimGray" and "Password:" with a masked password "••••••••". Below the password field is a "Show Password" button. At the bottom center is a "Log in" button. In the bottom-right corner is a "Registration" button.

Рисунок 4.3 – Результат авторизації адміністратора

За необхідності можна вимкнути приховування паролю, результат зображено на рисунку 4.4.

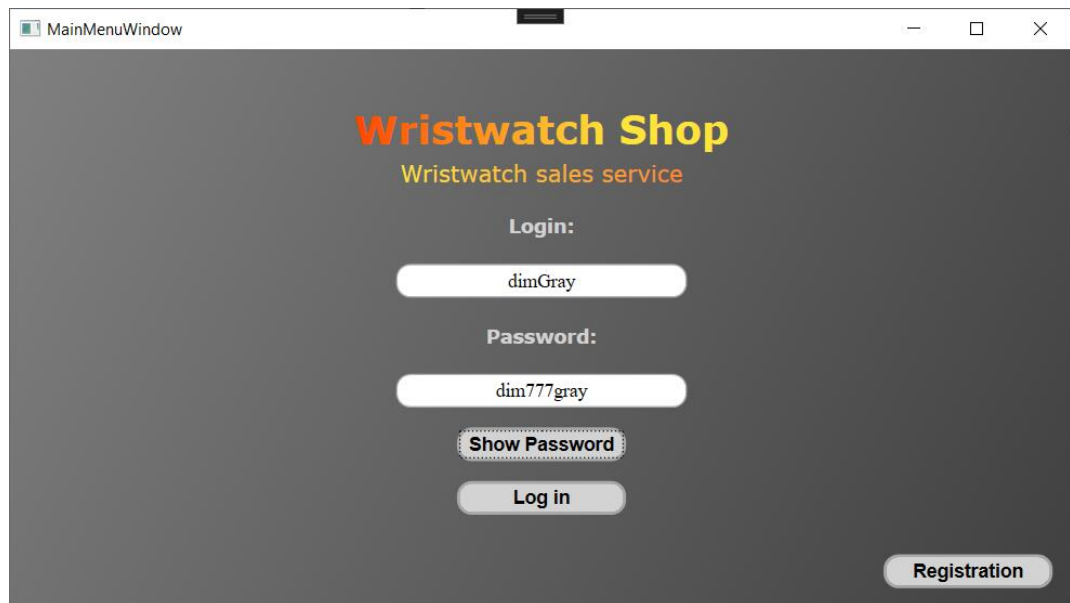


Рисунок 4.4 – Результат відображення паролю у неприхованому вигляді

4.3.2 Виконання роботи програми

Після авторизації менеджер опиниться на головній формі програми, де знаходяться кнопки для переходу у розділи функціональності програмного забезпечення (акаунт адміністратора, управління наручними годинниками системи, управління матеріалами системи, управління знижками на ціну годинника, підтвердження угод про продаж годинників, історія продажів годинників, управління додатковими послугами до угоди, розрахунок статистики продажів годинників). Зовнішній вигляд форми зображено на рисунку 4.5.

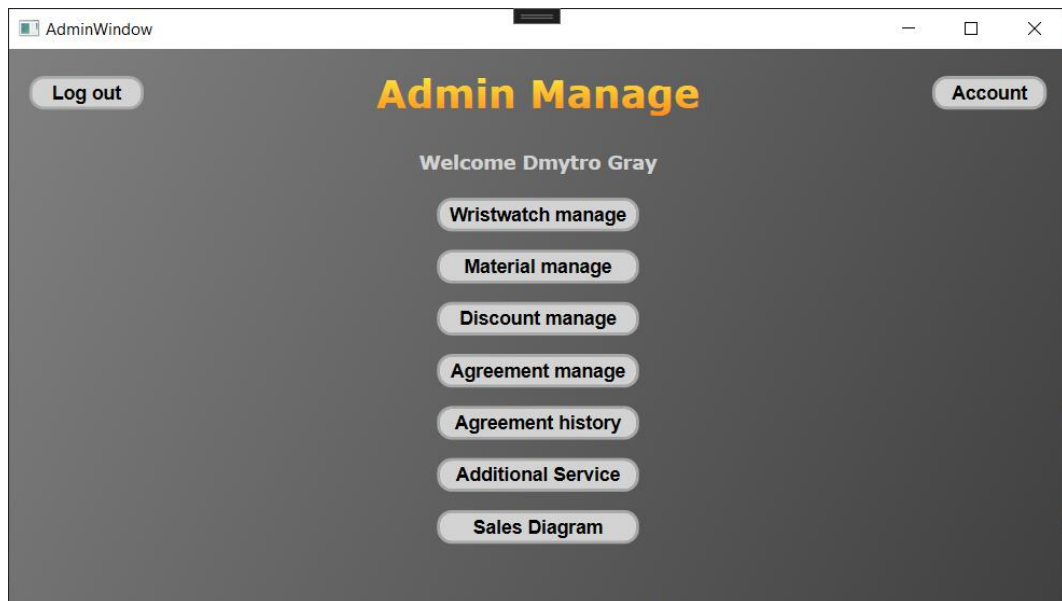


Рисунок 4.5 – Зовнішній вигляд головної форми адміністратора

Далі адміністратор може перейти у особистий акаунт для перегляду своєї інформації, зовнішній вигляд форми зображено на рисунку 4.6.

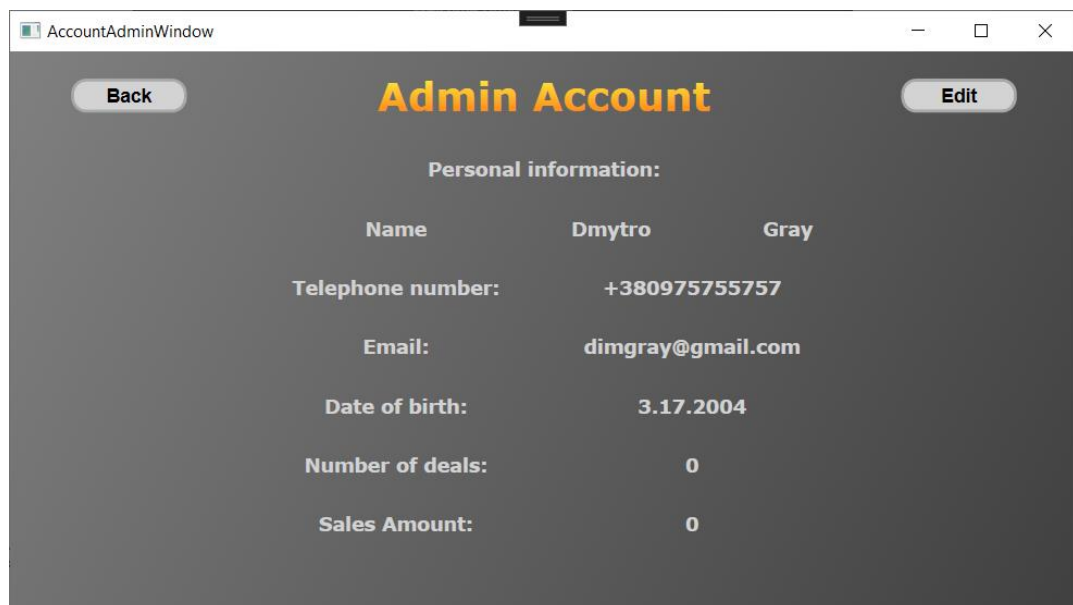


Рисунок 4.6 – Зовнішній вигляд форми акаунту користувача

Адміністратор може редагувати особисту інформацію, процес редагування зображено на рисунку 4.7.

The screenshot shows a window titled 'EditAccountWindow' with a dark gray background. At the top left is a 'Back' button. The title 'Edit Account' is centered in orange. Below it are several input fields for personal information: 'First Name' (Dmytro), 'Last Name' (Gray), 'Login' (dimGray), 'Password' (dim777gray), 'Phone Number' (+380975755757), 'Email' (dimgray_update@gmail.com), and 'Date of birth' (17.03.2004). At the bottom are 'Edit' and 'Delete' buttons.

Field	Value
First Name	Dmytro
Last Name	Gray
Login	dimGray
Password	dim777gray
Phone Number	+380975755757
Email	dimgray_update@gmail.com
Date of birth	17.03.2004

Рисунок 4.7 – Процес редагування особистої інформації адміністратора

Результат редагування особистої інформації адміністратора зображено на рисунку 4.8.

The screenshot shows a window titled 'AccountAdminWindow' with a dark gray background. At the top left is a 'Back' button and at the top right is an 'Edit' button. The title 'Admin Account' is centered in orange. Below it is the section 'Personal information:' followed by a table of administrator details.

Personal information:		
Name	Dmytro	Gray
Telephone number:	+380975755757	
Email:	dimgrayupdate@gmail.con	
Date of birth:	3.17.2004	
Number of deals:	0	
Sales Amount:	0	

Рисунок 4.8 – Результат редагування інформації адміністратора

Далі адміністратор може перейти у розділ управління наручними годинниками сервісу. Зовнішній вигляд форми управління годинниками зображено на рисунку 4.9.

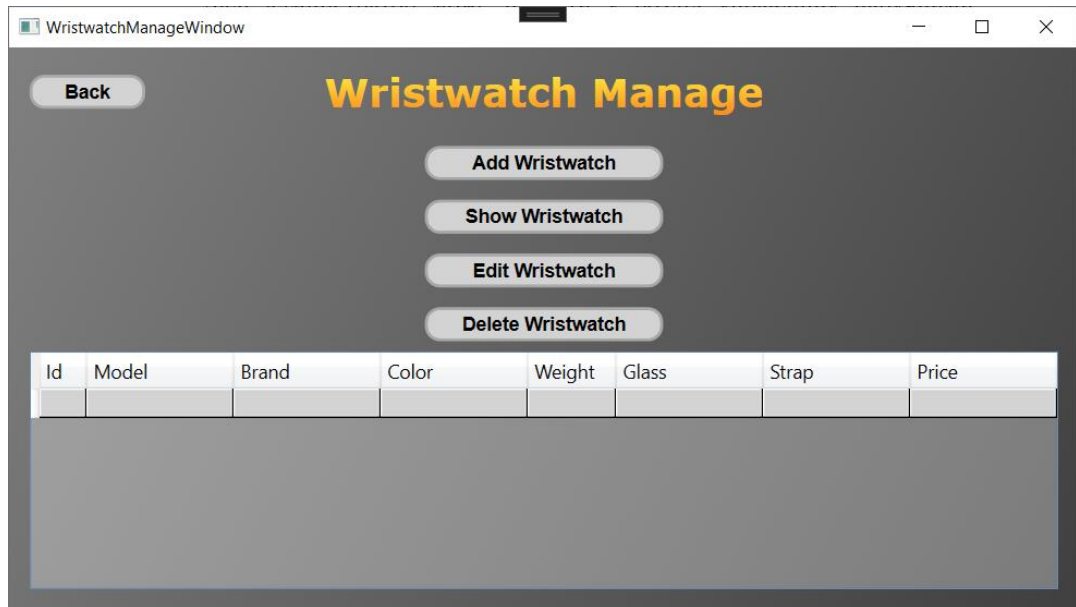


Рисунок 4.9 – Зовнішній вигляд форми управління наручними годинниками

Далі адміністратор може додати новий годинник до системи, перейшовши на форму додавання. Процес додавання зображено на рисунку 4.10.

Add Wristwatch

Model: EDFICE Classic

Brand: Casio

Color: Silver

Image: \Casio.png

Weight: 175

Material: Gold update

Glass: Mineral

Strap: Steel

Price: 6940

Discount: None

Add

Рисунок 4.10 – Процес додавання наручного годинника

Результат додавання нового годинника до системи зображено на рисунку 4.11.

Wristwatch Manage

Add Wristwatch

Show Wristwatch

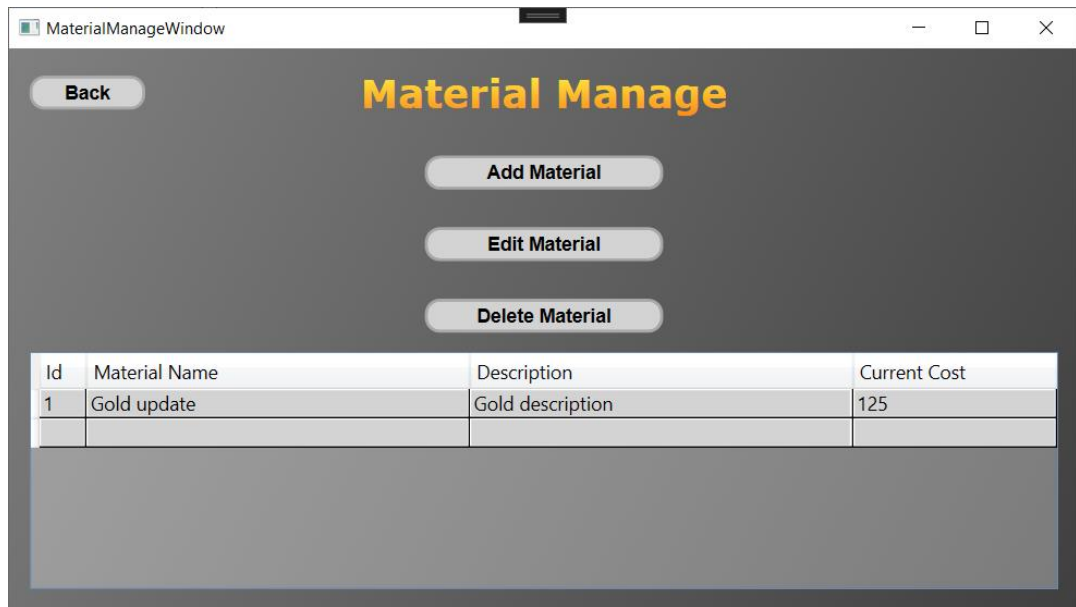
Edit Wristwatch

Delete Wristwatch

Id	Model	Brand	Color	Weight	Glass	Strap	Price
17	EDFICE Classic	Casio	Silver	175	Mineral	Steel	6940

Рисунок 4.11 – Результат додавання наручного годинника

Далі адміністратор може перейти на форму управління матеріалами з яких виготовлені наручні годинники, зовнішній вигляд форми зображено на рисунку 4.11.

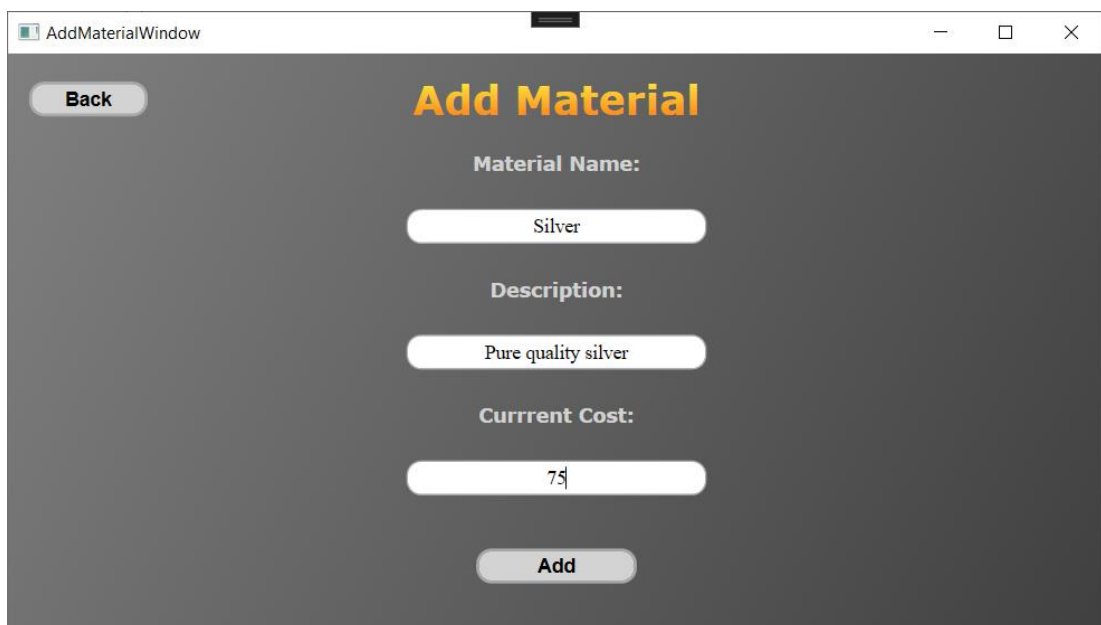


The screenshot shows a window titled "MaterialManageWindow". It features a "Back" button in the top left. The main heading is "Material Manage" in orange. Below the heading are three buttons: "Add Material", "Edit Material", and "Delete Material". At the bottom is a table with the following data:

Id	Material Name	Description	Current Cost
1	Gold update	Gold description	125

Рисунок 4.11 – Зовнішній вигляд форми управління матеріалами

Далі користувач може додати новий матеріал до системи, процес додавання матеріалу зображено на рисунку 4.12.



The screenshot shows a window titled "AddMaterialWindow". It features a "Back" button in the top left. The main heading is "Add Material" in orange. Below the heading are three input fields with labels: "Material Name:" (containing "Silver"), "Description:" (containing "Pure quality silver"), and "Current Cost:" (containing "75"). At the bottom is an "Add" button.

Рисунок 4.12 – Процес додавання нового матеріалу

Результат додавання матеріалу зображено на рисунку 4.13.

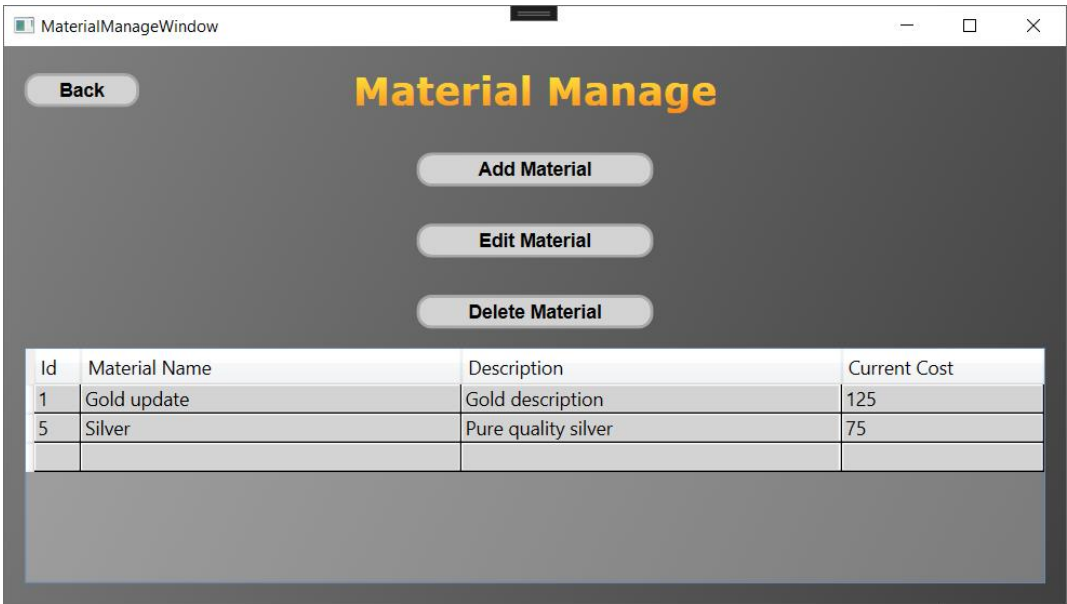


Рисунок 4.13 – Результат додавання матеріалу до системи

Далі адміністратор може перейти на форму управління знижками на ціну наручних годинників, зовнішній вигляд форми зображено на рисунку 4.14.

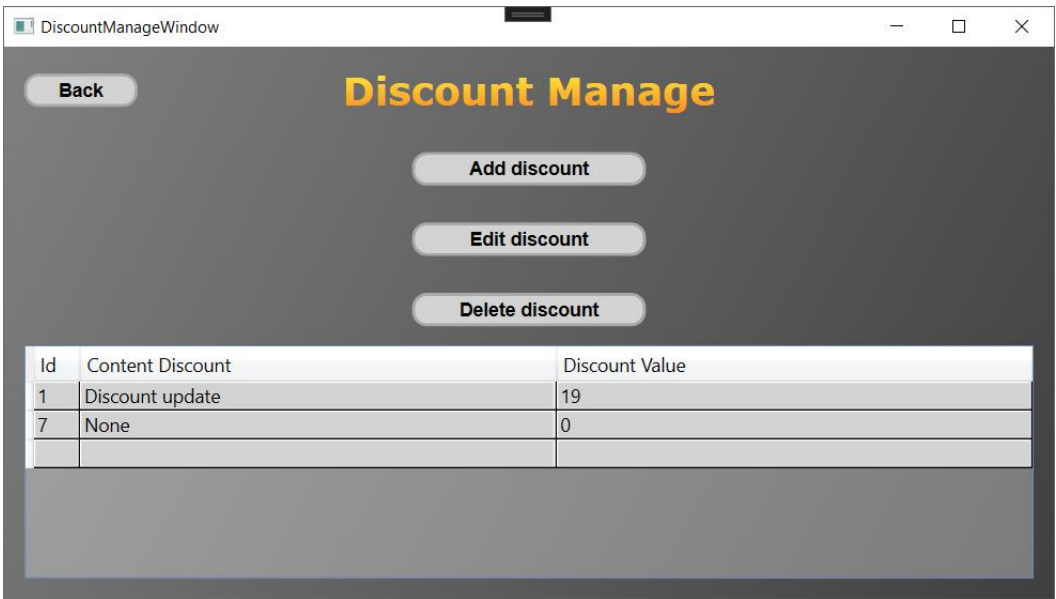


Рисунок 4.14 – Зовнішній вигляд форми управління знижками

Далі адміністратор може додати нову знижку, процес додавання зображено на рисунку 4.15.

Back

Add Discount

Content Discount:

Sale discount

Discount Value:

25

Add

Рисунок 4.15 – Процес додавання знижки

Результат додавання знижки зображено на рисунку 4.16.

Back

Discount Manage

Add discount

Edit discount

Delete discount

Id	Content Discount	Discount Value
1	Discount update	19
7	None	0
8	Sale discount	25

Рисунок 4.16 – Результат додавання знижки

Далі адміністратор може перейти на форму управління додатковими послугами до угоди про продаж наручного годинника. Зовнішній вигляд форми зображено на рисунку 4.17.

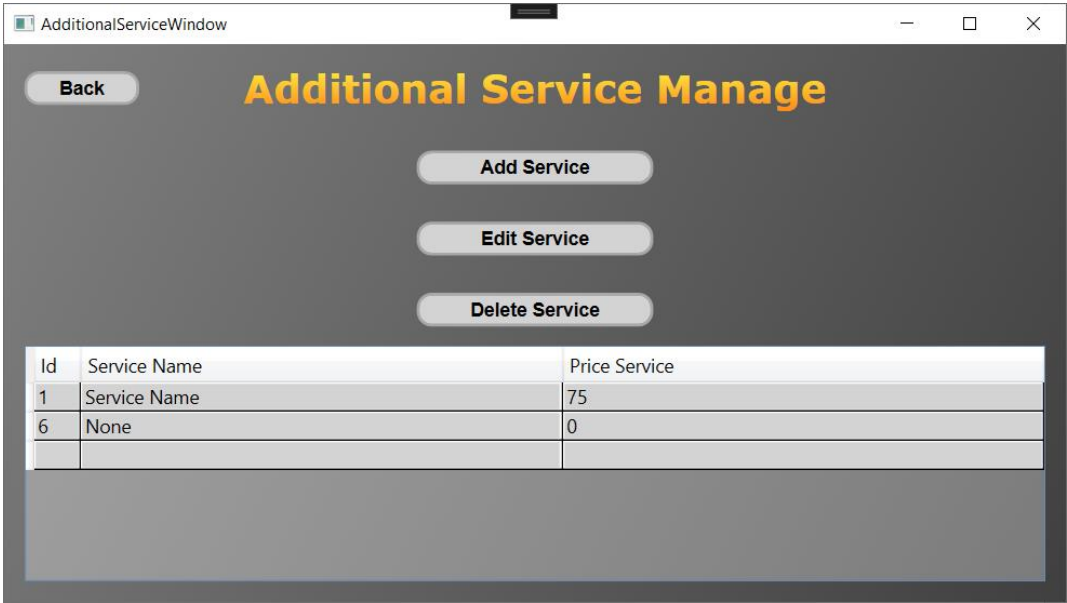


Рисунок 4.17 – Зовнішній вигляд форми управління додатковими послугами

Далі адміністратор може додати нову додаткову послугу, процес додавання додаткової послуги зображено на рисунку 4.18.

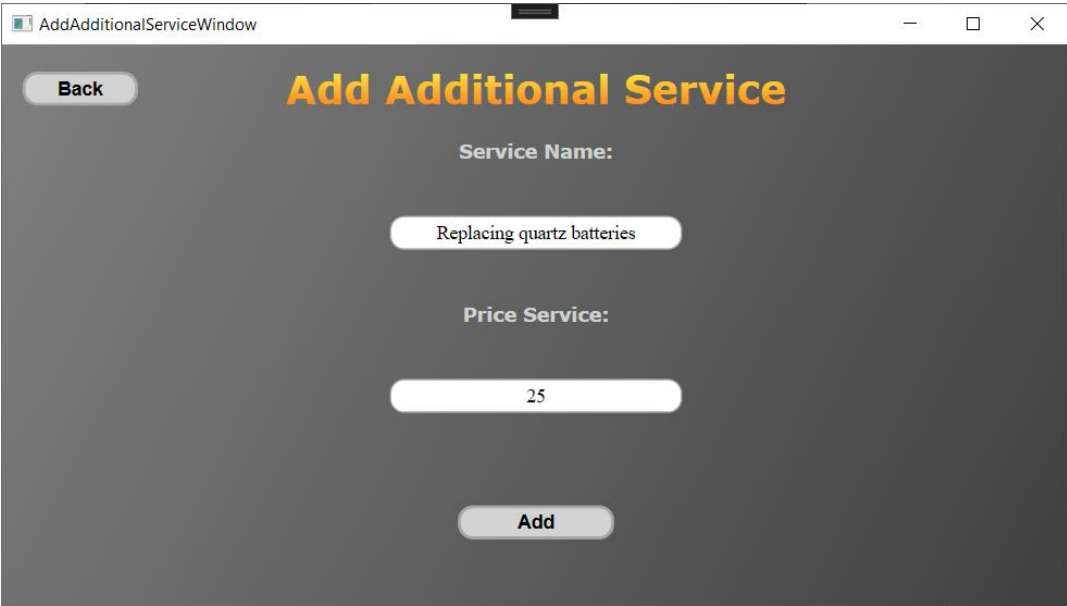


Рисунок 4.18 – Процес додавання додаткової послуги

Результат додавання додаткової послуги зображено на рисунку 4.19.

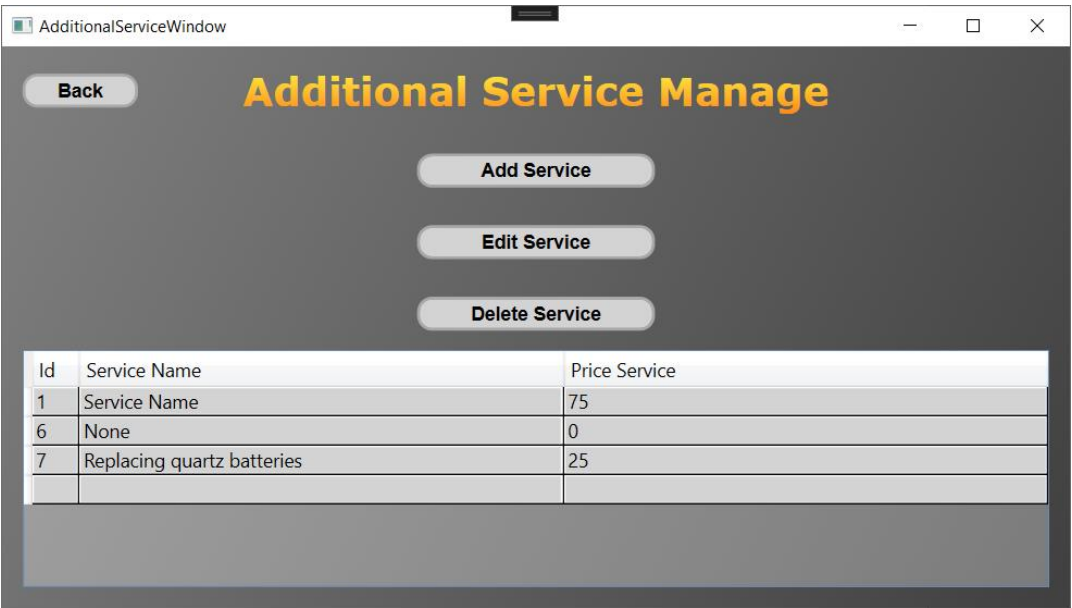


Рисунок 4.19 – Результат додавання додаткової послуги

Для клієнта необхідно спочатку авторизуватись у системі, далі користувач потрапить на головну форму зображену на рисунку 4.20.

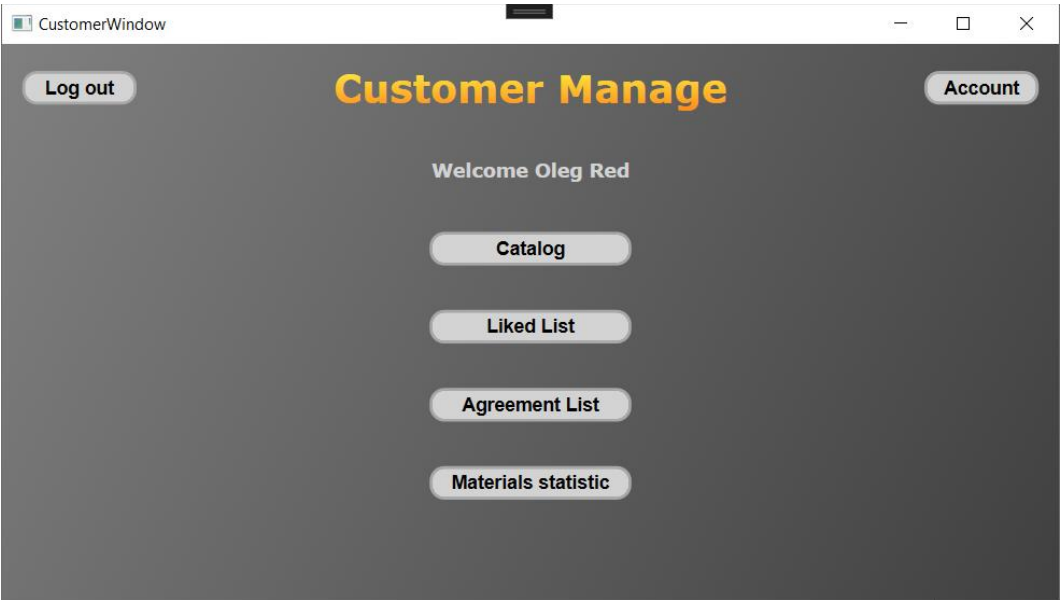
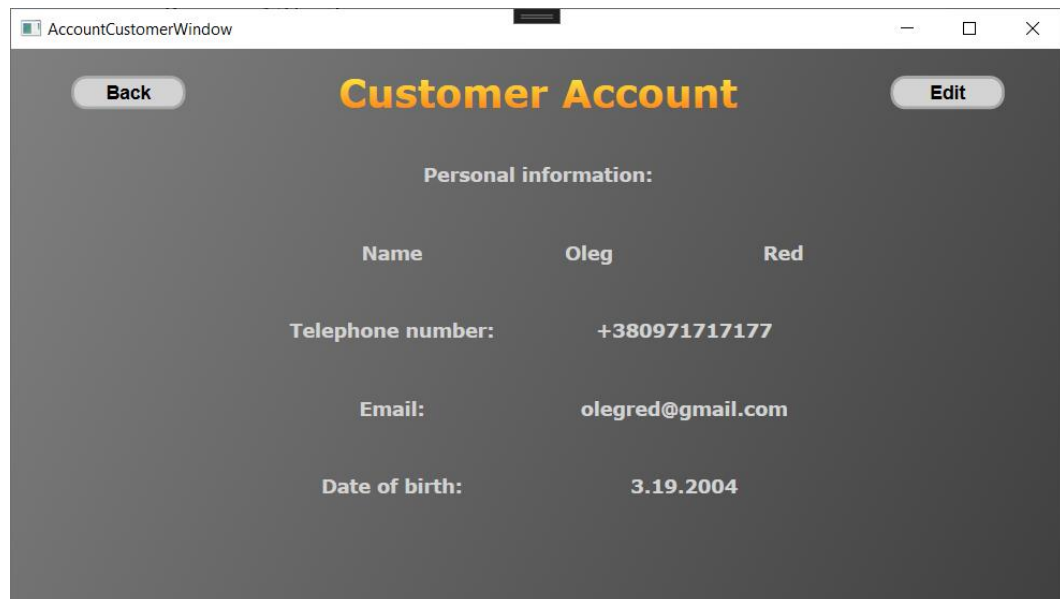


Рисунок 4.20 – Головна форма клієнта

Далі користувач може переглянути свій акаунт, форма акаунту користувача зображена на рисунку 4.21.



The screenshot shows a window titled 'AccountCustomerWindow'. Inside, there is a 'Customer Account' header with 'Back' and 'Edit' buttons. Below the header, the section 'Personal information:' contains a table of user details.

Personal information:		
Name	Oleg	Red
Telephone number:	+380971717177	
Email:	olegred@gmail.com	
Date of birth:	3.19.2004	

Рисунок 4.21 – Зовнішній вигляд акаунту користувача

Далі користувач може перейти до каталогу з доступними годинниками, зовнішній вигляд каталогу зображено на рисунку 4.22.



Рисунок 4.22 – Зовнішній вигляд каталогу з доступним годинником

Далі клієнт може додати наручний годинник, який йому сподобався до списку «Обране», результат відображення годинника у списку зображено на рисунку 4.23.

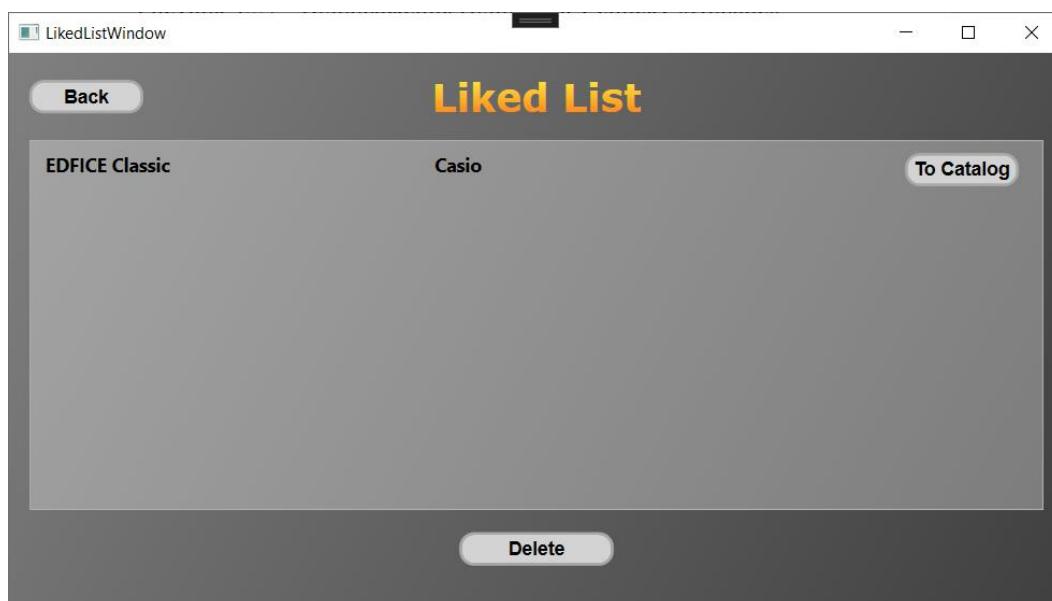


Рисунок 4.23 – Відображення годинника у списку «Обране»

Далі користувач може створити угоду про купівлю годинника, процес створення угоди, з додаванням до ціни страховки та додаткової послуги зображено на рисунку 4.24.

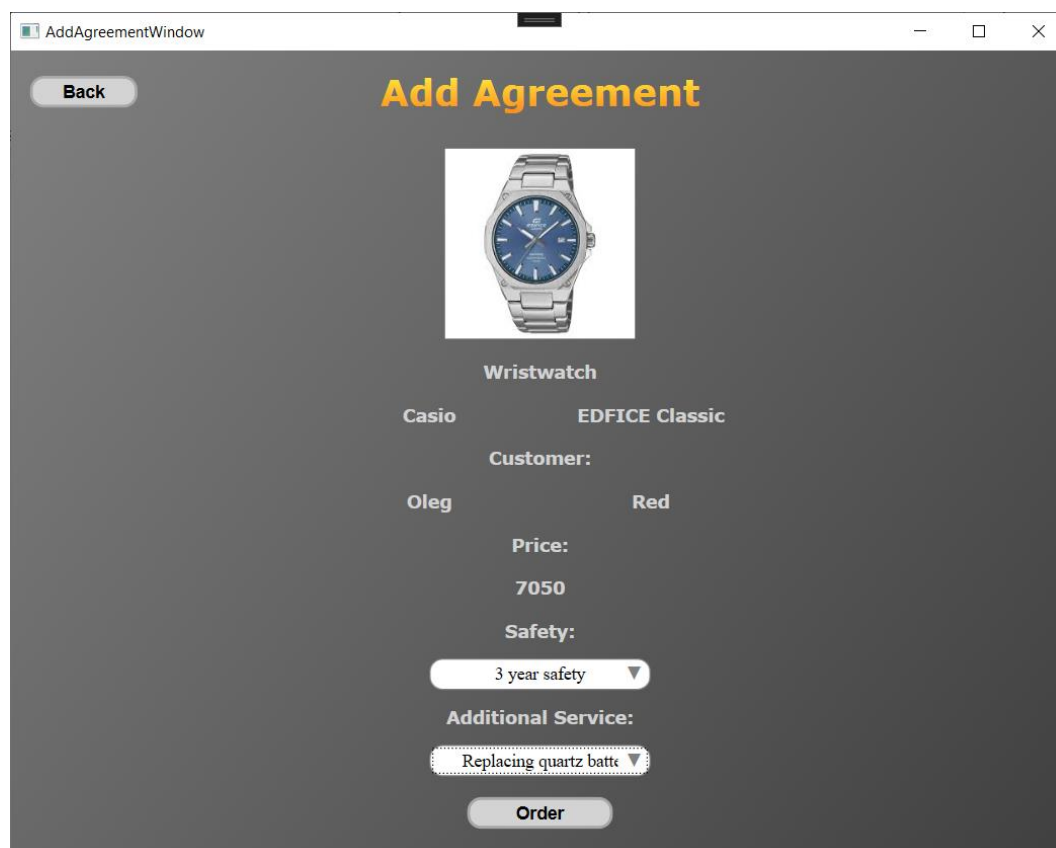


Рисунок 4.24 – Процес додавання угоди про купівлю годинника

Після додавання угоди, наручний годинник видаляється зі списку «Обране» та більше не доступний у каталозі годинників.

Далі користувач може переглянути свої замовлення на формі перегляду замовлень, зовнішній вигляд форми зображено на рисунку 4.25.



Рисунок 4.25 – Результат перегляду замовлення користувача

Далі адміністратору треба підтвердити замовлення клієнта на формі підтвердження замовлень, процес підтвердження угоди зображено на рисунку 4.26.

The screenshot shows a window titled 'SubmitWristwatchWindow' with a dark background. At the top left is a 'Back' button. The main title 'Agreement Manage' is in large orange text. Below it is a table with 8 columns: Id, Watch Id, Watch Model, Customer Id, Price, Date Agreement, Safety, and Additional Service. The first row contains the values: 20, 17, EDFICE Classic, 15, 7050, 4/9/2025, 3 year safety, and 7. Below the table is a large gray rectangular area. At the bottom center is a 'Submit' button.

Id	Watch Id	Watch Model	Customer Id	Price	Date Agreement	Safety	Additional Service
20	17	EDFICE Classic	15	7050	4/9/2025	3 year safety	7

Рисунок 4.26 – Процес підтвердження угоди про купівлю годинника

Після підтвердження угода вважається завершеною і адміністратор може переглянути історію продажів на відповідній формі, результат перегляду історії зображено на рисунку 4.27.

The screenshot shows a window titled 'AgreementHistoryWindow' with a dark background. At the top left is a 'Back' button. The main title 'Agreement History' is in large orange text. Below it are two tabs: 'Agreements' (selected) and 'My agreements'. Below the tabs is a table with 7 columns: Id, Watch, Price, Date Of Agreement, Safety, Additional Service, and Customer. The first row contains the values: 20, EDFICE Cla, 7050, 4/9/2025, 3 year safety, Replacing quartz batt, and 15. Below the table is a large gray rectangular area. At the bottom left is a 'Back Page' button, and at the bottom right is a 'Next Page' button.

Id	Watch	Price	Date Of Agreement	Safety	Additional Service	Customer
20	EDFICE Cla	7050	4/9/2025	3 year safety	Replacing quartz batt	15

Рисунок 4.27 – Результат перегляду історії продажу адміністратора

Далі адміністратор може переглянути діаграму розрахованої статистики продажів за різними критеріями, результат розрахунку продажів за брендом наручного годинника зображено на рисунку 4.28.

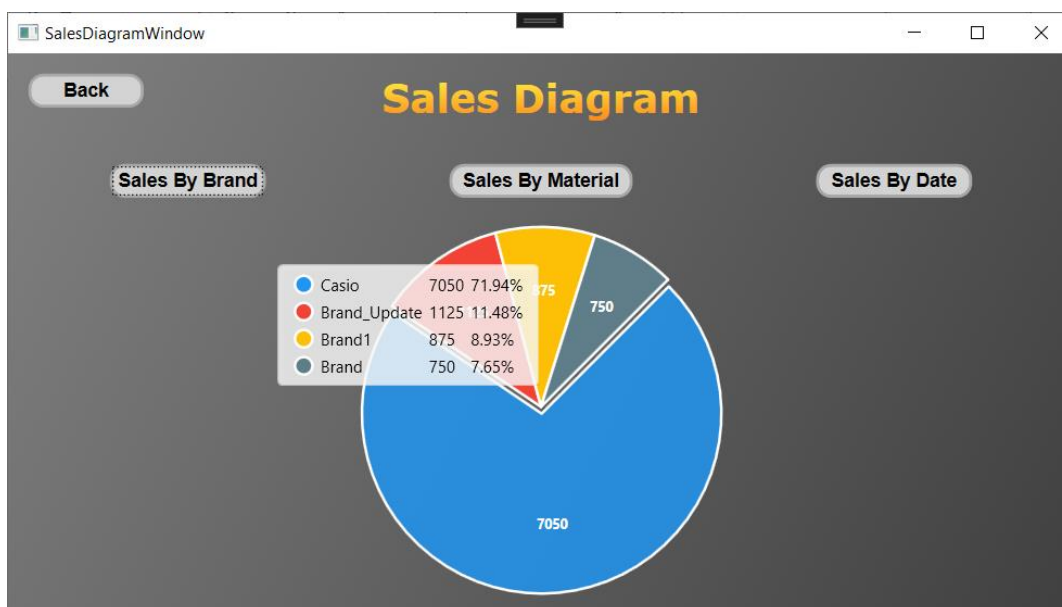


Рисунок 4.28 – Відображення статистики продажів за брендом

Клієнт може переглянути статистику зміни ціни на матеріал, з якого виготовлено годинник, що розраховується аналізуючи json-файл з ціною матеріалу по рокам та, використовуючи рівняння лінійної регресії, визначає чи варто клієнту інвестувати у цей матеріал. Результат відображення статистики та рекомендації зображено на рисунках 4.29-4.31.

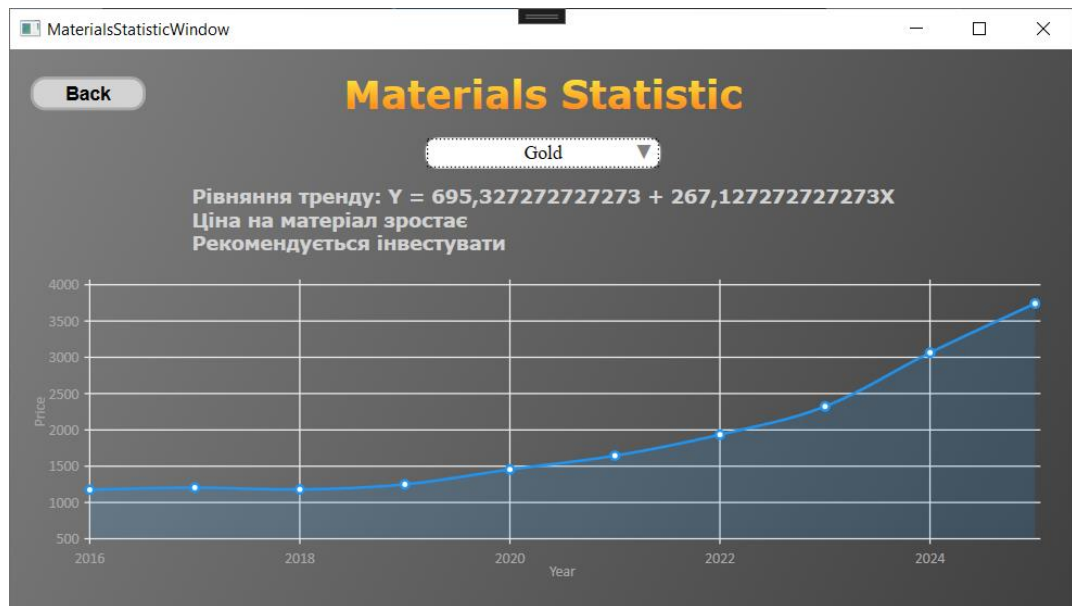


Рисунок 4.29 – Результат розрахунку статистики зміни ціни на золото

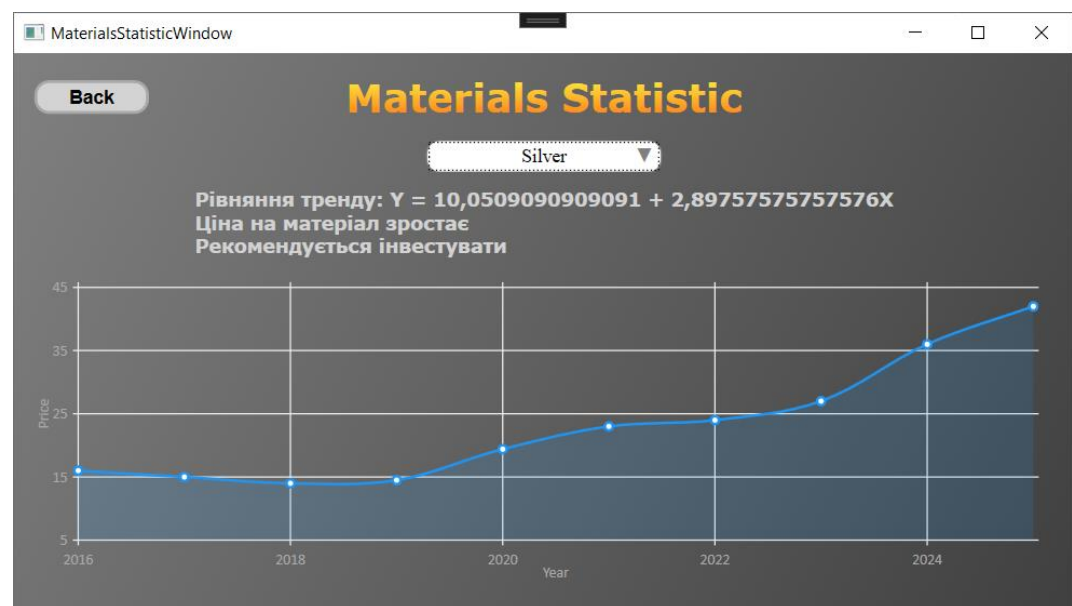


Рисунок 4.30 – Результат розрахунку статистики зміни ціни на срібло

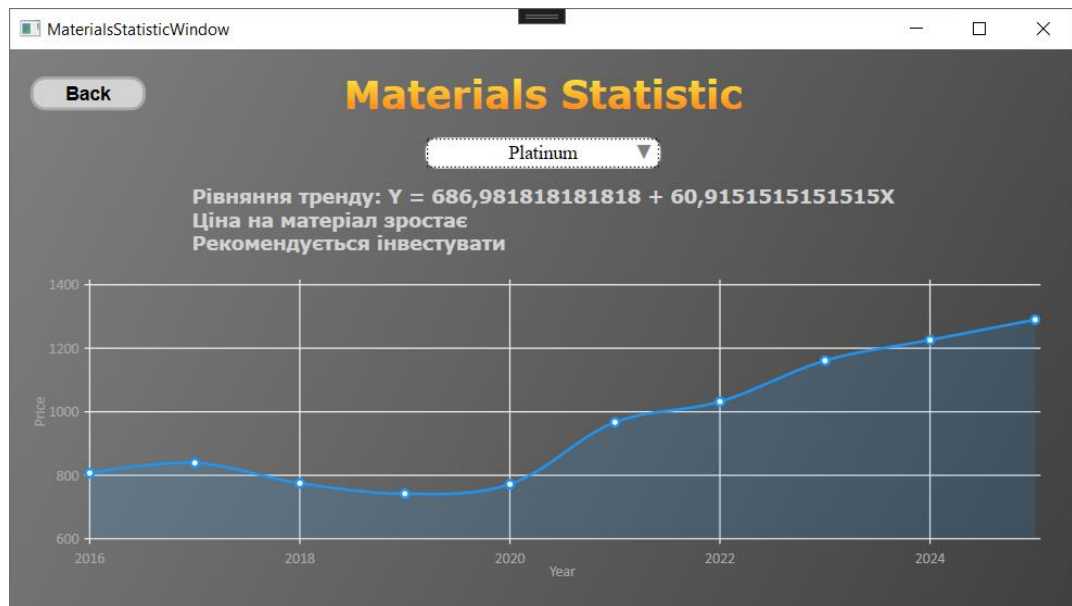


Рисунок 4.31 – Результат розрахунку статистики зміни ціни на платину

4.4 Повідомлення користувачу

Розроблена програма має повідомлення, що інформують користувача про помилки авторизації у системі. Менеджера інформують при редагуванні списку наручного годинника та іншої інформації у програмі. Повідомлення виводяться у вигляді вікон з текстом повідомлення. Результат виведення повідомлення при неправильному введенні логіна при авторизації зображено на рисунку 4.32.

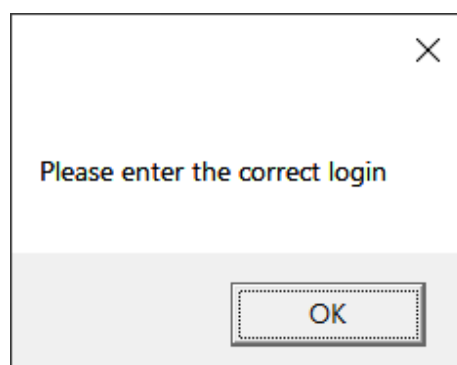


Рисунок 4.32 – Повідомлення користувачу при неправильному введенні логіну

При неправильному введенні пароля користувачеві буде виведено повідомлення про помилку авторизації, повідомлення зображено на рисунку 4.33.

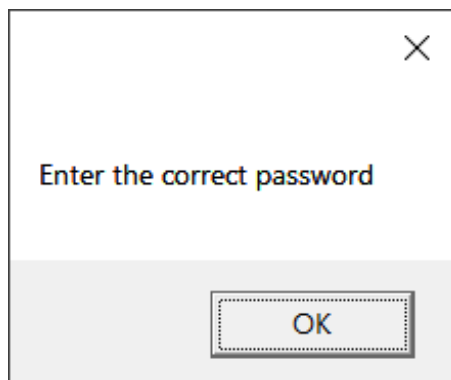


Рисунок 4.33 – Повідомлення користувачу при неправильному введенні пароля

При адмініструванні (додаванні, редагуванні, видаленні) інформації про годинники, матеріали, знижки, додаткові послуги у системі адміністратору виводяться повідомлення про успішність виконання операції або, у разі помилки, виводиться відповідне повідомлення.

Після успішного додавання наручного годинника у систему адміністратору виведеться повідомлення зображене на рисунку 4.34.

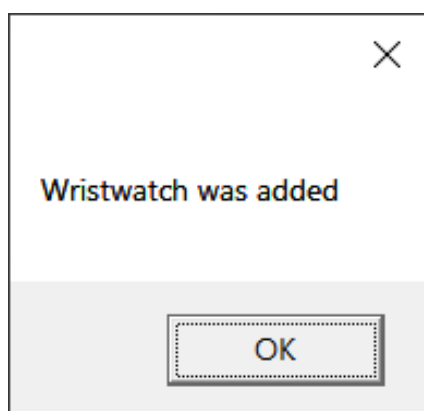


Рисунок 4.34 – Повідомлення при успішному додаванні годинника

Далі після успішного редагування наручного годинника адміністратору виведеться повідомлення зображене на рисунку 4.35.

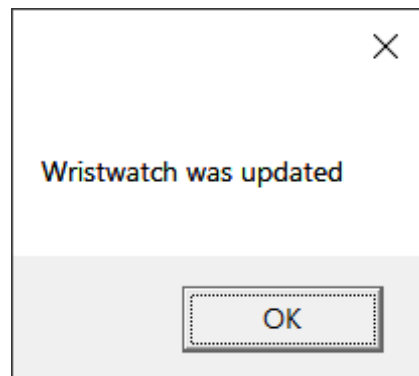


Рисунок 4.35 – Повідомлення при успішному оновленні годинника

Далі після видалення наручного годинника адміністратору виводиться повідомлення зображене на рисунку 4.36.

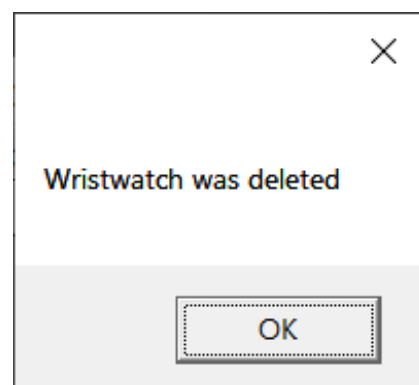


Рисунок 4.36 – Повідомлення при успішному видаленні годинника

4.5 Опис методики проведення тестування

Тестування програмного забезпечення було мануальним, тобто без використання автоматизованих тестів. Було проведено тестування функціональності, швидкості роботи та зручності користувацького інтерфейсу програми.

За результатами проведення мануального тестування можна стверджувати, що програма виконує усі передбачені функціональні вимоги: реєстрація та авторизація користувачів у системі, управління даними програми адміністратором, перегляд каталогу та оформлення угоди про покупку користувачем, розрахунок статистики продажів у адміністратора та розрахунок зміни курсу ціни на матеріал, з якого виготовленні годинники у клієнта.

Також програма працює швидко та є оптимізованою, адже програмна мова C# має автоматичні інструменти управління пам'яттю та її очищення у разі не використання.

Графічний інтерфейс користувача є інтуїтивно зрозумілим та простим для зручного використання, не потребує додаткового навчання для його розуміння. Реалізований у приємних кольорах, що не відволікає користувача від основного функціоналу програми.

4.6 Висновки з розділу

У результаті експериментального дослідження розробленого програмного забезпечення було мануально протестовано відповідність виконання усіх передбачених функцій програми:

- програмне забезпечення адміністрування електронної комерції інтернет-продажів наручних годинників надає менеджерам гнучкі інструменти для управління товарами, їх характеристиками та замовленнями;
- менеджери мають змогу додавати, редагувати та видаляти товари, оновлювати їхні описи, фото, матеріали, з яких виготовлені годинники, та ціни, а також підтверджувати замовлення клієнтів;
- важливою функцією є доступ до аналітичних даних, представлених у вигляді графіків і діаграм, що дозволяє ефективно відстежувати продажі та оцінювати їхні тенденції;

– менеджери можуть переглядати історію продажів, що допомагає в аналізі попиту та прийнятті стратегічних рішень щодо розвитку бізнесу.

Отже, програмне забезпечення виконує весь необхідний функціонал та має приємний та інтуїтивно зрозумілий графічний інтерфейс користувача, що робить програму зручною та корисною для використання.

ВИСНОВКИ

Під час виконання дипломної роботи було розроблено програмне забезпечення для адміністрування електронної комерції на прикладі інтернет-продажів наручних годинників. Десктопна програма призначена для ефективного керування товарами, замовленнями, аналізу продажів та зміни тенденції вартості матеріалів з яких виготовлений годинник, швидке та зручне оформлення замовлення для клієнтів сервісу.

Програмна реалізація здійснена мовою C# з використанням технології WPF .NET Framework 4.8.1 для забезпечення інтуїтивно зрозумілого графічного інтерфейсу користувача. Дані системи зберігаються у базі даних, яка працює на сервері Microsoft SQL Server. Для підвищення ефективності запитів було використано інструменти Entity Framework для взаємодії з базою даних.

Архітектура розробленої системи передбачає два типи користувачів: адміністраторів магазину (менеджерів) та клієнтів.

Менеджери мають можливість:

- додавати, редагувати та видаляти інформацію у системі;
- керувати характеристиками наручних годинників (опис, фото, матеріал, ціна);
- підтверджувати замовлення на покупку годинника клієнтів;
- переглядати аналітичні дані щодо продажів у вигляді діаграм;
- переглядати статистику власних продажів;
- переглядати історію продажів наручних годинників.

Клієнти мають можливість:

- реєструватися та авторизуватися у системі;
- переглядати годинники у зручному каталозі доступних до продажу годинників;
- додавати обрані годинники у список «Обране»;

- надсилати замовлення на покупку наручного годинника;
- обирати додаткові послуги до замовлення;
- переглядати зміни курсу вартості матеріалу за останні 10 років.

Окремою функціональною можливістю є реалізація модуля аналізу вартості дорогоцінного металу, з якого виготовлено наручні годинники, що дозволяє точніше оцінювати тенденцію зміни їх ринкової ціни та радить користувачеві звернути увагу на матеріал, з метою можливості придбання годинника як інвестицію.

Розробка даної програми дозволила автоматизувати процес адміністрування інтернет-магазину наручних годинників, покращила зручність використання для користувачів, підвищила ефективність управління продажами та забезпечила кращу взаємодію між адміністраторами та клієнтами.

Розроблений програмний продукт успішно реалізує всі поставлені задачі та відповідає функціональним вимогам. Впроваджена інтеграція з існуючими системами забезпечує високу продуктивність та стабільність роботи програми, а також відповідає стандартам безпеки та зручності користування.

Розроблення даного програмного забезпечення є важливим кроком для розвитку електронної комерції, адже воно сприяє покращенню якості обслуговування продажів та зростанню ефективності роботи сервісів які спеціалізуються на продажах наручних годинників.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Get started (WPF) [Electronic resource]. – Access mode: <https://learn.microsoft.com/uk-ua/dotnet/desktop/wpf/getting-started/?view=netframeworkdesktop-4.8>.
2. Кочкіна Н., Харченко Б. Споживчі мотивації на ринку luxury годинників України та Італії: допандемійні і довоєнні тенденції та перспективи. / Н. Кочкіна, Б. Харченко – Вісник Київського національного університету імені Тараса Шевченка. 2023. Економіка, № 1 (222), С. 69-75. DOI: <https://doi.org/10.17721/1728-2667.2023/222-1/9>.
3. Бачуріна Інна. Чи вбили смарт-годинники український годинниковий ринок. / Інна Бачуріна / [Електр. ресурс] – Режим доступу: <https://mind.ua/openmind/20176558-chi-vbili-smart-godinniki-ukrayinskij-godinnikovij-rinok>.
4. Топ марок наручних годинників 2020. [Електр. ресурс] – Режим доступу: <https://joy-pup.com/ua/techno-ua/top-marok-naruchnih-godinnikov-2020/>.
5. Кращі годинники українського виробництва: огляд топ-моделей та брендів. [Електр. ресурс] – Режим доступу: <https://odessa-life.od.ua/uk/article-uk/promo-uk/krashhi-godynnyky-ukrayinskogo-vyrobnyczstva-oglyad-top-modelej-ta-brendiv>.
6. Маркетплейс Rozetka.com.ua [Електрон. ресурс] – Режим доступу: <https://rozetka.com.ua/>.
7. Маркетплейс Allo.ua [Електрон. ресурс] – Режим доступу: <https://allo.ua/>.
8. Сервіс продажу годинників Deka.ua [Електрон. ресурс] – Режим доступу: <https://deka.ua/>.
9. Get started with .NET Framework [Electronic resource]. – Access mode: <https://learn.microsoft.com/uk-ua/dotnet/framework/get-started/>.

10. Object-Oriented programming (C#) [Electronic resource]. – Access mode: <https://learn.microsoft.com/uk-ua/dotnet/csharp/fundamentals/tutorials/oop>.
11. XAML overview (WPF .NET) [Electronic resource]. – Access mode: <https://learn.microsoft.com/uk-ua/dotnet/desktop/wpf/xaml/?view=netdesktop-8.0>.
12. Model-View-ViewModel (MVVM) [Electronic resource]. – Access mode: <https://learn.microsoft.com/uk-ua/dotnet/architecture/maui/mvvm>.
13. Entity Framework Core [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/ef/core/>.
14. LiveCharts – framework for creating graphics and diagrams [Electronic resource]. – Access mode: <https://v0.lvcharts.com/>.
15. Binding declarations overview (WPF .NET) [Electronic resource]. – Access mode: <https://learn.microsoft.com/uk-ua/dotnet/desktop/wpf/data/binding-declarations-overview?view=netdesktop-8.0>.
16. Overview of object-oriented techniques in C# [Electronic resource]. – Access mode: <https://learn.microsoft.com/uk-ua/dotnet/csharp/fundamentals/object-oriented/>.

ДОДАТОК А
Технічне завдання

A.1 Вступ

A.1.1 Найменування проєкту

Програмна реалізація системи адміністрування електронної комерції на прикладі інтернет продажів на прикладів наручних годинників.

A.1.2 Мета розробки та її призначення

Сучасний ринок електронної комерції вимагає ефективних та зручних інструментів для управління продажами, що дозволяють як адміністраторам, так і клієнтам зручно взаємодіяти із платформою. У зв'язку з цим виникає необхідність розробки програмного рішення, яке забезпечить автоматизацію процесів адміністрування інтернет продажів механічних годинників.

Програмна система повинна надати менеджерам гнучкі інструменти для управління товарами, їх характеристиками та замовленнями. Зокрема, вони матимуть змогу додавати, редагувати та видаляти товари, оновлювати їхні описи, фото, матеріали, з яких виготовлені годинники, та ціни, а також підтверджувати замовлення клієнтів. Важливою функцією буде доступ до аналітичних даних, представлених у вигляді графіків і діаграм, що дозволить ефективно відстежувати продажі та оцінювати їхні тенденції зміни. Менеджери також зможуть переглядати історію продажів, що допоможе в аналізі попиту та прийнятті стратегічних рішень щодо розвитку бізнесу.

Для клієнтів система має забезпечити інтуїтивно зрозумілий каталог, у якому вони зможуть переглядати годинники, додавати їх до списку «Обране», оформлювати замовлення та обирати додаткові послуги до угоди про продаж годинника.

Додатковою перевагою програмного забезпечення стане модуль аналізу вартості дорогоцінного металу, що дозволить оцінювати тенденції змін ринкових цін на матеріали, з яких виготовлені годинники. Ця функція

буде корисною для клієнтів, які розглядають покупку годинника як довгострокову інвестицію.

Таким чином, розробка цього програмного забезпечення дозволить автоматизувати ключові бізнес-процеси, покращити взаємодію між клієнтами та менеджерами, а також забезпечити ефективний аналіз ринку наручних годинників, що є важливим фактором успіху в сфері електронної комерції.

A.1.3 Підстава для розробки програми

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмна реалізація системи адміністрування електронної комерції на прикладі інтернет продажів наручних годинників», затверджене наказом Національного університету «Запорізька політехніка» № 209 від 28 квітня 2025 р.

A.2 Основні вимоги до програми

A.2.1 Вимоги до функціональних характеристик

Розроблюване програмне забезпечення має реалізовувати наступні функціональні вимоги:

- застосунок передбачатиме два типи користувачів, перший тип – адміністратори, другий тип – клієнти сервісу;
- можливість швидкої та простої реєстрації у застосунку;
- реалізація для адміністраторів зручного керування наручними годинниками (додавання, видалення), з можливістю редагування інформації про товар;
- забезпечення зручної логіки підтвердження замовлення менеджерами;

- можливість перегляду аналітичних даних щодо продажів у вигляді діаграм для адміністраторів;
- можливість у адміністраторів переглядати статистику власних продажів;
- можливість у адміністраторів переглядати історію як загальних, так і особистих продажів наручних годинників;
- реалізація зручного каталогу для перегляду годинників клієнтами;
- можливість у клієнтів додавати уподобані годинники у список «Обране»;
- можливість надсилати замовлення на покупку годинника;
- можливість обирати додаткові послуги до замовлення;
- можливість переглядати курс вартості матеріалу за останні 10 років;
- реалізація модуля аналізу вартості дорогоцінного металу, з якого виготовлено годинник, що дозволить точніше оцінювати тенденцію зміни їх ринкової ціни, з метою можливості придбання годинника як інвестицію.

A.2.2 Вимоги до графічного інтерфейсу

Вимоги до графічного інтерфейсу програми:

- розробка інтерфейсу з використанням технології WPF;
- дружній та інтуїтивно зрозумілий графічний інтерфейс;
- проста та чітка навігація між вікнами користувача;
- візуалізація статистики продажів у вигляді графічних діаграм.

A.2.3 Вимоги до безпеки програми

Вимоги до безпеки програми:

- збереження даних у базі даних для забезпечення цілісності та безпеки даних;
- приховування паролів при авторизації користувача;
- збереження конфіденційності користувачів при взаємодії з програмою.

A.2.4 Вимоги до технічних та програмних засобів

Для забезпечення зручної розробки десктопного застосунку, створеного на WPF .NET Framework 4.8.1, апаратні вимоги можуть бути такими як перелічено далі.

Процесор (CPU):

- а) тип процесора: сучасний багатоядерний процесор з архітектурою x86 або x64;
- б) тактова частота: рекомендується процесор з тактовою частотою 2.0 GHz або вище;
- в) кількість ядер: процесор з 4 або більше ядрами для забезпечення багатозадачності та швидкого виконання складних операцій.

Оперативна пам'ять (RAM):

- а) мінімальний обсяг: 4 GB RAM для базової роботи;
- б) рекомендований обсяг: 8 GB або більше для забезпечення плавної роботи, особливо при використанні ресурсозатратних функцій або великих обсягів даних.

Графічна підсистема:

- а) відеокарта: сучасна відеокарта з підтримкою DirectX 11 або вище;
- б) пам'ять відеокарти: мінімум 1 GB відеопам'яті, але для складніших графічних інтерфейсів рекомендується 2 GB і більше;
- в) роздільна здатність екрану: підтримка роздільної здатності 1920x1080 (Full HD) або вище для комфортного перегляду.

Дискова підсистема:

а) тип накопичувача: твердотільний накопичувач (SSD) для швидкого завантаження операційної системи та застосунків.

б) вільний простір: мінімум 3 GB вільного дискового простору для встановлення та роботи застосунку. Більше місця може знадобитися для зберігання даних.

Операційна система:

а) сумісні ОС: Windows 10 (64-bit) або Windows 11 (64-bit);

б) оновлення ОС: операційна система повинна бути оновлена.

ДОДАТОК Б**Приклад формування курсу вартості матеріалів**

Б.1 Вміст Json-файлу з курсом матеріалів

```
{  
  "materials": [  
    {  
      "name": "Gold",  
      "prices": [  
        { "year": 2016, "price": 1175 },  
        { "year": 2017, "price": 1205 },  
        { "year": 2018, "price": 1180 },  
        { "year": 2019, "price": 1250 },  
        { "year": 2020, "price": 1455 },  
        { "year": 2021, "price": 1645 },  
        { "year": 2022, "price": 1935 },  
        { "year": 2023, "price": 2323 },  
        { "year": 2024, "price": 3064 },  
        { "year": 2025, "price": 3742 }  
      ]  
    },  
    {  
      "name": "Silver",  
      "prices": [  
        { "year": 2016, "price": 16 },  
        { "year": 2017, "price": 15 },  
        { "year": 2018, "price": 14 },  
        { "year": 2019, "price": 14.5 },  
        { "year": 2020, "price": 19.4 },  
        { "year": 2021, "price": 23 },  
        { "year": 2022, "price": 24 },  
        { "year": 2023, "price": 27 },  
        { "year": 2024, "price": 36 },  
        { "year": 2025, "price": 42 }  
      ]  
    },  
  ],  
}
```

```
{  
  "name": "Platinum",  
  "prices": [  
    { "year": 2016, "price": 807 },  
    { "year": 2017, "price": 839 },  
    { "year": 2018, "price": 775 },  
    { "year": 2019, "price": 742 },  
    { "year": 2020, "price": 772 },  
    { "year": 2021, "price": 967 },  
    { "year": 2022, "price": 1032 },  
    { "year": 2023, "price": 1161 },  
    { "year": 2024, "price": 1226 },  
    { "year": 2025, "price": 1290 }  
  ]  
}  
]  
}
```

ДОДАТОК В
Текст програми

B.1 Код Wristwatch.cs

```

namespace Shop_Wristwatch.DataModel
{
    using System;
    using System.Collections.Generic;

    public partial class Wristwatch
    {
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2214:DoNotCallOverridableMethodsInConstructors")]
        public Wristwatch()
        {
            this.Agreement = new HashSet<Agreement>();
            this.User = new HashSet<User>();
        }

        public int WatchId { get; set; }
        public string Model { get; set; }
        public string Brand { get; set; }
        public string Color { get; set; }
        public string Image { get; set; }
        public double Weight { get; set; }
        public int MaterialWatchId { get; set; }
        public string Glass { get; set; }
        public string Strap { get; set; }
        public double Price { get; set; }
        public Nullable<int> DiscountId { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<Agreement> Agreement { get; set; }
        public virtual Discount Discount { get; set; }
        public virtual Material Material { get; set; }
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<User> User { get; set; }
    }
}

```

B.2 Код DataBaseManage.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Remoting.Contexts;
using System.Text;
using System.Threading.Tasks;

```

```

using System.Windows;
using Shop_Wristwatch.DataModel;

namespace Shop_Wristwatch
{
    public class DataBaseManage
    {
        ...

        public User Get_User_From_DB(string login)
        {
            using(var context = new WatchDataBaseEntities())
            {
                try
                {
                    var user = context.User.FirstOrDefault(u => u.Login == login);

                    if(user != null)
                    {
                        return user;
                    }
                    else
                    {
                        MessageBox.Show("User was not found");

                        return null;
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);

                    return null;
                }
            }
        }

        ...

        public List<Wristwatch> Get_CatalogWristwatches_From_DB()
        {
            using (var context = new WatchDataBaseEntities())
            {
                try
                {
                    var wristwatches = context.Wristwatch.Where(watch =>
!watch.Agreement.Any()).ToList();

                    return wristwatches;
                }
                catch (Exception ex)
                {

```

```

        MessageBox.Show(ex.Message);

        return new List<Wristwatch>();
    }
}

...

public void Add_LikedWatch_To_DB(int wristwatchId, int customerId)
{
    using (var context = new WatchDataBaseEntities())
    {
        try
        {
            var user = context.User.Include("Wristwatch").FirstOrDefault(u =>
u.UserId == customerId);
            var watch = context.Wristwatch.Find(wristwatchId);

            if(watch != null && user != null)
            {
                user.Wristwatch.Add(watch);

                context.SaveChanges();

                MessageBox.Show("Wristwatch was added into liked list");
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}

...

public bool Check_LikedWristwatch_In_DB(int customerId, int wristwatchId)
{
    using (var context = new WatchDataBaseEntities())
    {
        try
        {
            var user = context.User.Include("Wristwatch").FirstOrDefault(u =>
u.UserId == customerId);

            if (user.Wristwatch.Find(watch => watch.WatchId == wristwatchId) != null)
            {
                return true;
            }
        }
    }
}

```

```

        else
        {
            return false;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);

        return false;
    }
}

```

...

```

public List<Agreement> Get_Agreements_For_Submit_From_DB()
{
    using(var context = new WatchDataBaseEntities())
    {
        try
        {
            var agreements =
context.Agreement.Include("Wristwatch").Where(agreement => agreement.AdminId ==
null).ToList();

            return agreements;
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message);

            return new List<Agreement>();
        }
    }
}

```

...

```

public List<Agreement> Get_AgreementHistory_From_DB(User admin, int
pageNumber, int pageSize)
{
    using(var context = new WatchDataBaseEntities())
    {
        try
        {
            int totalPages = Get_TotalPages_AgreementHistory(pageSize, admin);

            if(pageNumber > totalPages)
            {
                pageNumber = totalPages;
            }
        }
    }
}

```

```

        var agreements = context.Agreement
            .Include("Wristwatch")
            .Include("AdditionalServices")
            .Where(agreement => agreement.AdminId != null)
            .OrderBy(agreement => agreement.AgreementId)
            .Skip((pageNumber - 1) * pageSize)
            .Take(pageSize)
            .ToList();

        return agreements;
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);

        return new List<Agreement>();
    }
}

public int Get_TotalPages_AgreementHistory(int pageSize, User user)
{
    using(var context = new WatchDataBaseEntities())
    {
        try
        {
            int totalRecords = 0;

            if (user.Status == "Admin")
            {
                totalRecords = context.Agreement.Where(rec => rec.AdminId !=
null).Count();
            }
            else if (user.Status == "Customer")
            {
                totalRecords = context.Agreement.Where(rec => rec.CustomerId ==
user.UserId && rec.AdminId != null).Count();
            }

            int totalPages = (int)Math.Ceiling(((double)totalRecords / pageSize);

            return totalPages;
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message);

            return 0;
        }
    }
}

```

```

public void Set_Number_Of_Deals_For_Admin(Admin admin)
{
    using(var context = new WatchDataBaseEntities())
    {
        try
        {
            var adminUpdate = context.Admin.Find(admin.AdminId);

            if(adminUpdate != null)
            {
                adminUpdate.NumberOfDeals++;

                context.SaveChanges();
            }
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}

public void Set_Sales_Amount_For_Admin(Admin admin, Agreement agreement)
{
    using(var context = new WatchDataBaseEntities())
    {
        try
        {
            var adminUpdate = context.Admin.Find(admin.AdminId);

            if(adminUpdate != null)
            {
                adminUpdate.SalesAmount += agreement.Price;

                context.SaveChanges();
            }
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}

public List<Agreement> Get_Orders_For_User_From_DB(User customer, int
pageNumber, int pageSize)
{
    using (var context = new WatchDataBaseEntities())
    {
        try
        {

```

```

        int totalRecords = context.Agreement.Where(agreement =>
agreement.CustomerId == customer.UserId && agreement.AdminId == null).Count();

        int totalPages = (int)Math.Ceiling(((double)totalRecords / pageSize);

        if (pageNumber > totalPages)
        {
            pageNumber = totalPages;
        }

        var agreements = context.Agreement
            .Include("Wristwatch")
            .Include("AdditionalServices")
            .Where(agreement => agreement.CustomerId == customer.UserId &&
agreement.AdminId == null)
            .OrderBy(agreement => agreement.AgreementId)
            .Skip((pageNumber - 1) * pageSize)
            .Take(pageSize)
            .ToList();

        return agreements;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);

        return new List<Agreement>();
    }
}
}

```

...

```

public void Update_Agreemet_For_Submit(Agreement agreementUpdate)
{
    using(var context = new WatchDataBaseEntities())
    {
        try
        {
            var agreement = context.Agreement.Find(agreementUpdate.AgreementId);

            if(agreement != null)
            {
                if(agreementUpdate.AdminId != null)
                {
                    agreement.AdminId = agreementUpdate.AdminId;
                }

                context.SaveChanges();
                MessageBox.Show("Agreement was updated");
            }
            else

```

```

        {
            MessageBox.Show("Agreement was not found");
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
}
...

```

B.3 Код AgreementManage.cs

```

using Shop_Wristwatch.DataModel;

...

namespace Shop_Wristwatch.Manage
{
    public class AgreementManage: INotifyPropertyChanged
    {
        ...

        private List<Agreement> agreements;
        public List<Agreement> Agreements
        {
            get { return agreements; }
            set
            {
                agreements = value;
                OnPropertyChanged(nameof(Agreements));
            }
        }

        public int PageNumber { get; set; }

        public int PageSize { get; set; }
        public double CurrentWristwatchPrice { get; set; }
        public double SafetyPrice { get; set; }
        public double AdditionalServicePrice { get; set; }

        public List<string> SafetyList { get; set; } =
            new List<string>() { "None", "1 year safety", "2 years safety", "3 year safety" };

        public List<AdditionalServices> AdditionalServicesList { get; set; }
    }
}

```

```

Discount Discount { get; set; }

DataBaseManage dataBaseManage;

public AgreementManage(User customer, Wristwatch wristwatch)
{
    dataBaseManage = new DataBaseManage();

    Agreement = new Agreement();

    User = customer;

    Wristwatch = wristwatch;

    if (Wristwatch.DiscountId != null)
    {
        Discount
dataBaseManage.Get_Discount_From_DB(Wristwatch.DiscountId);
    }

    if (Discount != null)
    {
        if (Discount.ContentDiscount != "None")
        {
            Wristwatch.Price = Wristwatch.Price - ((Wristwatch.Price / 100) *
Discount.DiscountValue);
        }
    }

    Agreement.Price = Wristwatch.Price;

    CurrentWristwatchPrice = Wristwatch.Price;

    WristwatchImage = CatalogManage.getImage(Wristwatch.Image);

    AdditionalServicesList = dataBaseManage.Get_AdditionalServices_From_DB();
}

public AgreementManage(User user)
{
    dataBaseManage = new DataBaseManage();

    User = user;

    PageNumber = 1;

    PageSize = 3;
}

public void Add_Agreement()
{
    Agreement.WatchId = Wristwatch.WatchId;

```

```

        Agreement.CustomerId = User.UserId;

        Agreement.DateOfAgreement = DateTime.Now.Date;

        dataBaseManage.Add_Agreement_To_DB(agreement);

        if (dataBaseManage.Check_LikedWristwatch_In_DB(user.UserId,
Wristwatch.WatchId))
        {
            dataBaseManage.Delete_LikedWristwatch_From_DB(user.UserId,
Wristwatch);
        }
    }

    public void Safety_Wristwatch_Price(string safety)
    {
        if(AdditionalServicePrice == 0)
        {
            CurrentWristwatchPrice = Wristwatch.Price;
        }
        else
        {
            CurrentWristwatchPrice = Wristwatch.Price + AdditionalServicePrice;
        }

        if (safety == "None")
        {
            Agreement.Price -= SafetyPrice;

            SafetyPrice = 0;
        }
        if (safety == "1 year safety")
        {
            CurrentWristwatchPrice += 35;

            SafetyPrice = 35;

            Agreement.Price = CurrentWristwatchPrice;
        }
        else if (safety == "2 years safety")
        {
            CurrentWristwatchPrice += 65;

            SafetyPrice = 65;

            Agreement.Price = CurrentWristwatchPrice;
        }
        else if (safety == "3 year safety")
        {
            CurrentWristwatchPrice += 85;

```

```

        SafetyPrice = 85;

        Agreement.Price = CurrentWristwatchPrice;
    }
}

public void AdditionalService_Wristwatch_Price(AdditionalServices
additionalService)
{
    if(SafetyPrice == 0)
    {
        CurrentWristwatchPrice = Wristwatch.Price;
    }
    else
    {
        CurrentWristwatchPrice = Wristwatch.Price + SafetyPrice;
    }

    if (additionalService.ServiceName == "None")
    {
        Agreement.Price -= AdditionalServicePrice;

        AdditionalServicePrice = 0;
    }
    if (additionalService != null)
    {
        CurrentWristwatchPrice += additionalService.PriceService;

        AdditionalServicePrice = additionalService.PriceService;

        Agreement.Price = CurrentWristwatchPrice;
    }
}

public void Get_Agreements_For_Submit()
{
    Agreements = dataBaseManage.Get_Agreements_For_Submit_From_DB();
}

public void Submit_Agreement(Agreement agreement)
{
    agreement.AdminId = user.UserId;

    dataBaseManage.Update_Agreemet_For_Submit(agreement);

    var admin = dataBaseManage.Get_Admin_From_DB(user.UserId);

    dataBaseManage.Set_Number_Of_Deals_For_Admin(admin);

    dataBaseManage.Set_Sales_Amount_For_Admin(admin, agreement);
}

```

```

public void Show_AgreementHistory()
{
    PageNumber = 1;

    Agreements =
dataBaseManage.Get_AgreementsHistory_For_User_From_DB(User, PageNumber, PageSize);
}

public void Show_AgreementHistory_For_Admin()
{
    PageNumber = 1;

    Agreements = dataBaseManage.Get_AgreementHistory_From_DB(User,
PageNumber, PageSize);
}

public void Show_Orders()
{
    PageNumber = 1;

    Agreements = dataBaseManage.Get_Orders_For_User_From_DB(User,
PageNumber, PageSize);
}

public void Show_AgreementHistory_By_Admin()
{
    PageNumber = 1;

    Agreements =
dataBaseManage.Get_Admin_AgreementHistory_From_DB(User, PageNumber, PageSize);
}

...

public void Back_AgreementHistory_Page()
{
    if(PageNumber > 1)
    {
        PageNumber--;

        Agreements = dataBaseManage.Get_AgreementHistory_From_DB(User,
PageNumber, PageSize);
    }
}

public void Next_AgreementHistory_Page()
{
    int totalPage = dataBaseManage.Get_TotalPages_AgreementHistory(PageSize,
User);

    if (PageNumber < totalPage)
    {

```

```

        PageNumber++;

        Agreements = dataBaseManage.Get_AgreementHistory_From_DB(User,
        PageNumber, PageSize);
    }
}

...

}
}

```

B.4 Код SalesDiagramManage.cs

```

using LiveCharts;
using LiveCharts.Wpf;
using Shop_Wristwatch.DataModel;

...

namespace Shop_Wristwatch.Manage
{
    class SalesDiagramManage: INotifyPropertyChanged
    {
        public List<Agreement> Agreements { get; set; }

        private SeriesCollection wristwatchSales;
        public SeriesCollection WristwatchSales
        {
            get
            {
                return wristwatchSales;
            }
            set
            {
                wristwatchSales = value;
                OnPropertyChanged(nameof(WristwatchSales));
            }
        }

        DataBaseManage dataBaseManage;

        public SalesDiagramManage()
        {
            dataBaseManage = new DataBaseManage();
        }

        public void Get_Statistics_For_Sales_By_Brand()
        {

```

```

        Agreements = dataBaseManage.Get_Agreements_From_DB();

        var salesByBrand = Agreements.GroupBy(agreement =>
agreement.Wristwatch.Brand).Select(item =>
        new
        {
            Brand = item.Key,
            TotalSales = item.Sum(agreement => agreement.Price)
        })
        .OrderByDescending(i => i.TotalSales).ToList();

        WristwatchSales = new SeriesCollection();

        foreach(var item in salesByBrand)
        {
            WristwatchSales.Add(new PieSeries
            {
                Title = item.Brand,
                Values = new ChartValues<double> { item.TotalSales },
                DataLabels = true
            });
        }
    }

    public void Get_Statistics_For_Sales_By_Material()
    {
        Agreements = dataBaseManage.Get_Agreements_From_DB();

        var salesByMaterial = Agreements.GroupBy(agreement =>
agreement.Wristwatch.Material.MaterialName).Select(item =>
        new
        {
            Material = item.Key,
            TotalSales = item.Sum(agreement => agreement.Price)
        })
        .OrderByDescending(i => i.TotalSales).ToList();

        WristwatchSales = new SeriesCollection();

        foreach (var item in salesByMaterial)
        {
            WristwatchSales.Add(new PieSeries
            {
                Title = item.Material,
                Values = new ChartValues<double> { item.TotalSales },
                DataLabels = true
            });
        }
    }

    public void Get_Statistics_For_Sales_By_DateAgreement()
    {

```

```

        Agreements = dataBaseManage.Get_Agreements_From_DB();

        var salesByDateAgreement = Agreements.GroupBy(agreement =>
agreement.DateOfAgreement.Month).Select(item =>
            new
            {
                DateAgreement = item.Key,
                TotalSales = item.Sum(agreement => agreement.Price)
            })
            .OrderByDescending(i => i.TotalSales).ToList();

        WristwatchSales = new SeriesCollection();

        foreach(var item in salesByDateAgreement)
        {
            WristwatchSales.Add(new PieSeries
            {
                Title = item.DateAgreement.ToString(),
                Values = new ChartValues<double> { item.TotalSales },
                DataLabels = true
            });
        }
    }
    ...
}

```

B.5 Код MaterialsStatisticManage.cs

```

using LiveCharts;
using LiveCharts.Wpf;
using Newtonsoft.Json;
using Shop_Wristwatch.DataModel;

...

namespace Shop_Wristwatch.MaterialsStatistic
{
    public class MaterialsStatisticManage : INotifyPropertyChanged
    {
        private List<MaterialDto> materials;

        public List<MaterialDto> Materials
        {
            get
            {
                return materials;
            }
        }
    }
}

```

```

        set
        {
            materials = value;
            OnPropertyChanged(nameof(Materials));
        }
    }

    private MaterialDto selectedMaterial;
    public MaterialDto SelectedMaterial
    {
        get
        {
            return selectedMaterial;
        }
        set
        {
            selectedMaterial = value;
            OnPropertyChanged(nameof(SelectedMaterial));
        }
    }

    private SeriesCollection chartSeries;
    public SeriesCollection ChartSeries
    {
        get
        {
            return chartSeries;
        }
        set
        {
            chartSeries = value;
            OnPropertyChanged(nameof(ChartSeries));
        }
    }

    ...

    const string filePath = "J:\\Дипломна робота\\Shop_Wristwatch\\Materials.json";

    public MaterialsStatisticManage()
    {
        Materials = new List<MaterialDto>();

        ChartSeries = new SeriesCollection();

        Labels = new List<string>();
    }

    public void Get_MaterialsPrice_From_JSON()
    {
        string json = File.ReadAllText(filePath);
    }

```

```

        var jsonData = JsonConvert.DeserializeObject<Dictionary<string,
List<MaterialsJsonModel>>>(json);

        if(jsonData != null && jsonData.TryGetValue("materials", out var materialsList))
        {
            Materials = materialsList.Select(m => new MaterialDto
            {
                MaterialName = m.Name,
                PriceByYears = m.Prices.ToDictionary(price => price.Year, price =>
price.Price)
            })
            .ToList();
        }

        if(Materials.Count > 0)
        {
            SelectedMaterial = Materials.First();
        }
    }

    public void UpdateChart()
    {
        if(SelectedMaterial != null && SelectedMaterial.PriceByYears != null)
        {
            Labels = SelectedMaterial.PriceByYears.Keys.Select(year =>
year.ToString()).ToList();

            ChartSeries.Clear();

            ChartSeries.Add(new LineSeries
            {
                Title = SelectedMaterial.MaterialName,
                Values = new
ChartValues<double>(SelectedMaterial.PriceByYears.Values)
            });
        }
    }

    public void Calculate_Statistic()
    {
        Advice_Message = null;

        if(SelectedMaterial != null && SelectedMaterial.PriceByYears != null &&
SelectedMaterial.PriceByYears.Count > 1)
        {
            var baseYear = SelectedMaterial.PriceByYears.Keys.Min();
            var years = SelectedMaterial.PriceByYears.Keys.Select(y => (double)(y -
baseYear)).ToList();
            var prices = SelectedMaterial.PriceByYears.Values.ToList();

            int n = years.Count;
            double sumX = years.Sum();

```

```

double sumY = prices.Sum();
double sumXY = years.Zip(prices, (x, y) => x * y).Sum();
double sumX2 = years.Sum(x => x * x);

double b = (n * sumXY - sumX * sumY) / (n * sumX2 - sumX * sumX);
double a = (sumY - b * sumX) / n;

Advice_Message += $"Рівняння тренду: Y = {a} + {b}X\n";
if(b > 0)
{
    Advice_Message += "Ціна на матеріал зростає\n";
    Advice_Message += "Рекомендується інвестувати";
}
else
{
    Advice_Message += "Ціна на матеріал спадає\n";
    Advice_Message += "Не рекомендується інвестувати";
}
}
else
{
    Advice_Message += "Недостатньо даних для аналізу";
}
}

public event PropertyChangedEventHandler PropertyChanged;
protected void OnPropertyChanged(string propertyName)
{
    PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
}
}

```

ДОДАТОК Г
Слайди презентації

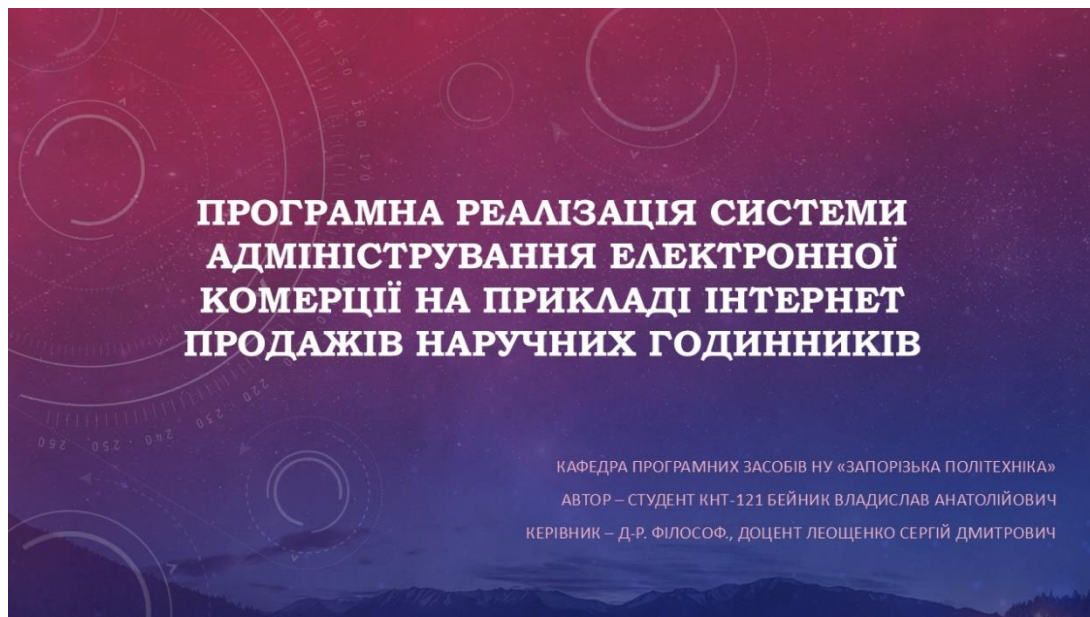


Рисунок Г.1 – Слайд 1



Рисунок Г.2 – Слайд 2

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Механічні наручні годинники вважаються важливим аксесуаром, поєднуючи функціональність та естетику. В Україні ринок наручних годинників демонструє динамічний розвиток, відображаючи як світові тенденції, так і локальні особливості обігу продукції на ринку.

Ринок наручних годинників в Україні можна розділити на три сегменти.

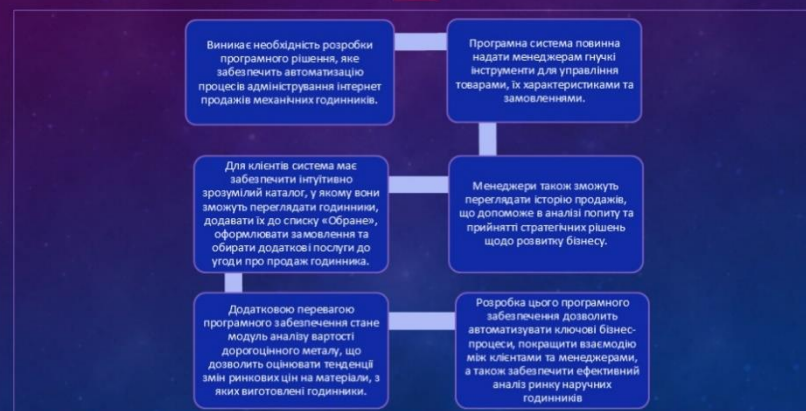
- Низький та середній ціновий сегмент: у цьому сегменті переважають моделі масового виробництва, доступні широкому колу споживачів.
- Високий ціновий сегмент (преміум): цей сегмент орієнтований на преміальні та люксові моделі годинників, які підкреслюють статус власника.

Ринок наручних годинників в Україні характеризується різноманітністю сегментів та суттєвим впливом сучасних технологій. Сегментація ринку вказує на різні вподобання споживачів залежно від цінового діапазону. Вплив смарт-годинників відчутний у низькому та середньому ціновому сегменті, тоді як преміум-моделі мають перевагу у високому ціновому сегменті ринку. Вітчизняні виробники, сприяють розвитку національного годинникового виробництва, пропонуючи споживачам якісні та унікальні продукти наручних механічних годинників.

3

Рисунок Г.3 – Слайд 3

НЕОБХІДНІСТЬ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



4

Рисунок Г.4 – Слайд 4

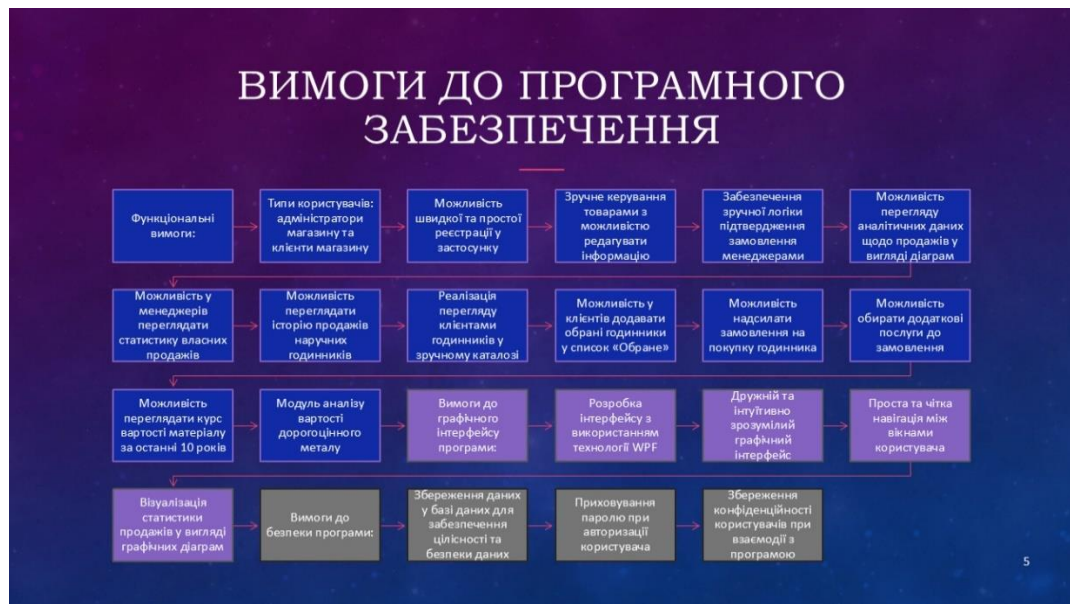


Рисунок Г.5 – Слайд 5

ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

Критерій	Rozetka	Allo.ua	Deka.ua
Основна спеціалізація	Універсальний маркетплейс	Універсальний маркетплейс	Спеціалізований магазин годинників
Фокус на механічних годинниках	Є, але не основна категорія	Присутній, але більший акцент на смарт-годинники	Основна спеціалізація
Асортимент годинників	Механічні, кварцові, смарт-годинники	Переважають смарт-годинники, але присутні і класичні моделі	Переважно механічні і кварцові годинники
Ціновий сегмент	Від бюджетного до преміального	Від бюджетного до середнього	Від середнього до преміального
Гарантія	Від продавців, не завжди надається офіційна гарантія	Залежить від продавців та бренду	Офіційна гарантія на всі годинники
Надання додаткових послуг після продажу	Немає	Немає	Є сервісне обслуговування
Наявність знижок	Наявні	Наявні	Наявні
Наявність аналітичних інструментів	Немає	Немає	Немає
Графічний інтерфейс користувача	Зрозумілий інтерфейс, але через те, що це не спеціалізований магазин, користувачеві може бути складно орієнтуватись	Гарний графічний інтерфейс, зрозумілий для користувача	Простий графічний інтерфейс

6

Рисунок Г.6 – Слайд 6

ВИБІР МОВИ ПРОГРАМУВАННЯ

Критерій	C#	C++	Python
Типізація	Статична, яка є безпечною	Статична	Динамічна
Складність	Середня	Висока	Низька
Парадигма	Об'єктно-орієнтована, імперативна	Об'єктно-орієнтована, процедурна, функціональна	Об'єктно-орієнтована, процедурна, функціональна
Автоматичне управління пам'яттю	Так (Garbage Collector)	Ні	Так (Garbage Collector)
Швидкість роботи	Висока	Дуже висока	Низька
Кросплатформність	Так	Так	Так
Основне призначення	Веб, десктоп, мобільні додатки, ігри	Системне програмування, ігри, високопродуктивні програми	Веб, наукові розрахунки, машинне навчання

Рисунок Г.7 – Слайд 7

ВИБІР СЕРЕДОВИЩА РОЗРОБКИ

Критерій	Visual Studio Community 2022	Visual Studio Code	SharpDevelop
Продуктивність	Потужне IDE, але може сповільнюватися при великих проектах	Легке і швидке IDE, можна розширити функціональність	Легке IDE, швидке, але обмежене у функціональності
Функціональність	Вбудований налагоджувач, інструменти для тестування, DevOps	Підтримує плагіни, має вбудований термінал, інструменти для DevOps	Базові інструменти для .NET розробки, обмежена функціональність
Ціна	Безкоштовне, але для комерційного використання є платна версія	Безкоштовне	Безкоштовне
Додаткові інструменти	Інтеграція з GitHub, розширення Marketplace	Велика кількість розширень у Marketplace, інтеграція з Git	Підтримка розширень, інтеграція з Git
Особливості	Підтримка .NET Framework, .NET Core, DevOps	Гнучке налаштування середовища, підтримка роботи у хмарі	Підходить для невеликих проектів та навчання

Рисунок Г.8 – Слайд 8

ВИБІР СИСТЕМИ УПРАВЛІННЯ БАЗОЮ ДАНИХ

Критерій	Microsoft SQL Server	MySQL	Access
Тип СКБД	Реляційна	Реляційна	Файлова, реляційна
Розробник	Microsoft	Oracle Corporation	Microsoft
Призначення	Високонавантажені корпоративні системи, аналітика, бізнес-рішення	Веб-додатки, середні та малі проекти	Невеликі локальні бази даних, офісні рішення
Мова запитів	T-SQL	SQL	SQL (обмежений) + VBA
Продуктивність	Висока	Висока	Низька
Безпека	Високий рівень	Високий рівень	Низький рівень
Масштабованість	Висока, підтримка кластеризації та реплікації	Висока	Обмежена
Підтримка ACID	Так	Так	Часткова
Підтримка транзакцій	Так	Так	Часткова

9

Рисунок Г.9 – Слайд 9

ПІДХІД ПРОЄКТУВАННЯ DATABASE FIRST

Database First – це підхід у Entity Framework, який дозволяє спочатку створити базу даних у СКБД, а потім автоматично згенерувати C#-класи об'єктної моделі для роботи з нею. Він ідеально підходить для випадків, коли база вже існує або її розробляють окремо від застосунку.

Основні особливості підходу Database First:

- Автоматична генерація моделей
- Синхронізація зі схемою бази
- Підтримка складних реляційних структур
- Збереження контролю за SQL-структурою
- Можливість використання з великими системами
- Використання LINQ та ORM
- Менше ризиків помилок у моделі

Database First – це ідеальний варіант для програмного проєкту адміністрування електронної комерції інтернет-продажів на ручних годинниках, де база даних уже створена у СКБД Microsoft SQL Server. Він дає змогу зосередитися на бізнес-логіці застосунку, використовуючи переваги Entity Framework для роботи з даними.

10

Рисунок Г.10 – Слайд 10

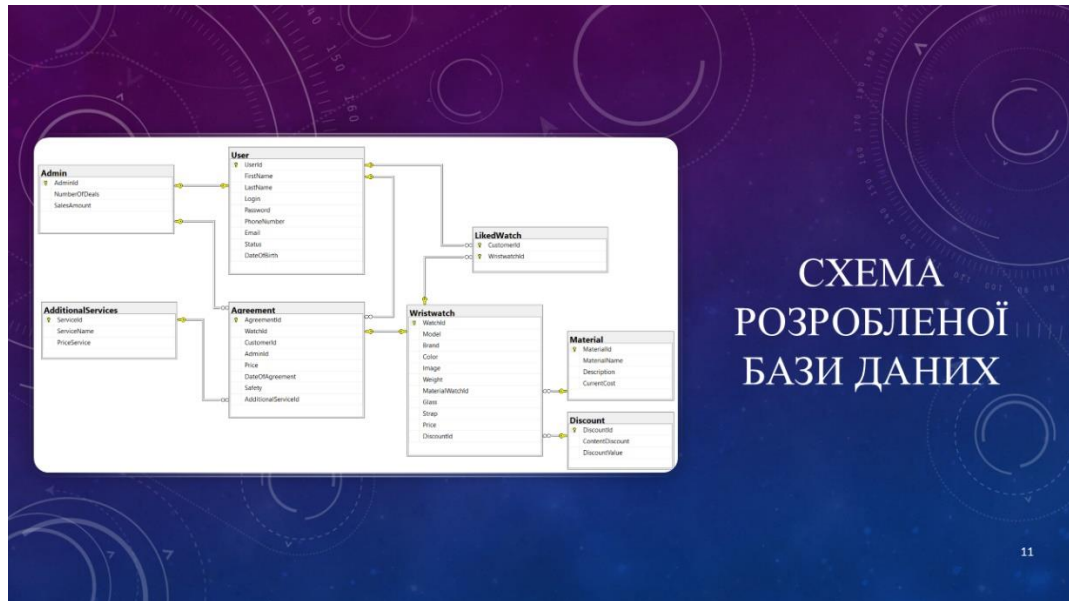


Рисунок Г.11 – Слайд 11

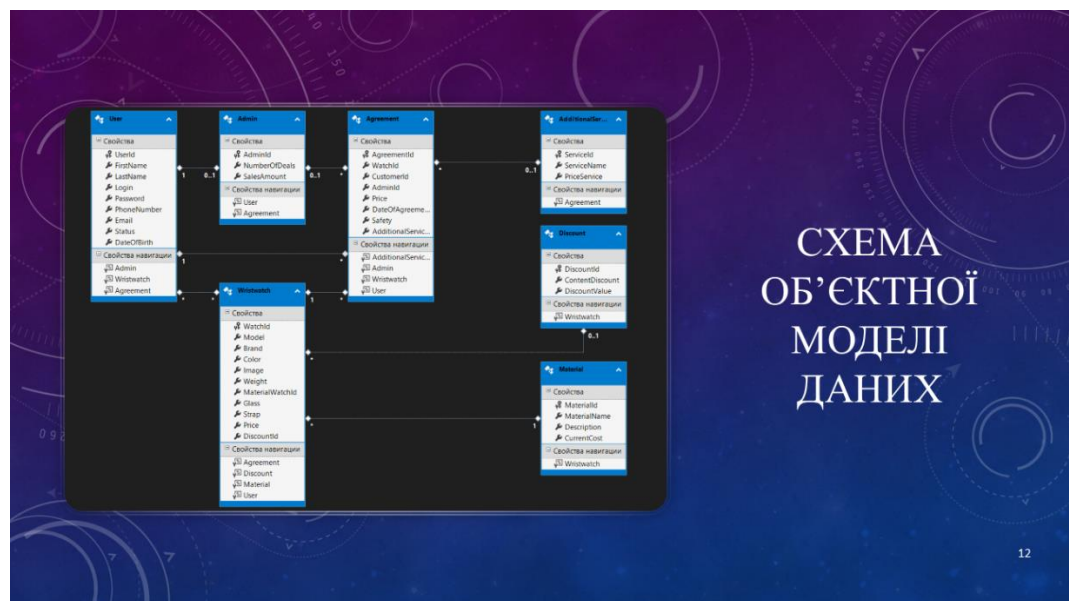
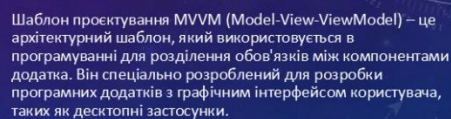


Рисунок Г.12 – Слайд 12

Для розробки програмного забезпечення було використано шаблон подібний до архітектури проектування MVVM.



Основна ідея MVVM – розділення бізнес-логіки та інтерфейсу користувача для полегшення розробки, тестування та підтримки коду. Крім того, цей шаблон дозволяє робити різні представлення для одних і тих же даних (наприклад, десктопний інтерфейс) без зміни логіки додатка.

Рисунок Г.13 – Слайд 13



Рисунок Г.14 – Слайд 14



Рисунок Г.15 – Слайд 15

РОЗРОБЛЕНІ ПРОГРАМНІ АЛГОРИТМИ

Для функціонування програмного забезпечення було розроблено такі основні алгоритми:

- авторизація та реєстрація користувача у системі;
- додавання нового годинника до системи;
- алгоритм додавання угоди про купівлю годинника;
- алгоритм для підтвердження адміністратором продажу годинника;
- алгоритм розрахунку особистої статистики продажів адміністратора;
- алгоритм перегляду історії продажів та покупок з додаванням пагінації у список;
- алгоритм обробки інформації з json-файлу для відображення зміни на протязі 10 років вартості матеріалу;
- алгоритм розрахунку коефіцієнту приросту ціни матеріалу, для поради користувачеві чи варто інвестувати у цей матеріал;
- алгоритм додавання годинника до списку «Обраного».

16

Рисунок Г.16 – Слайд 16

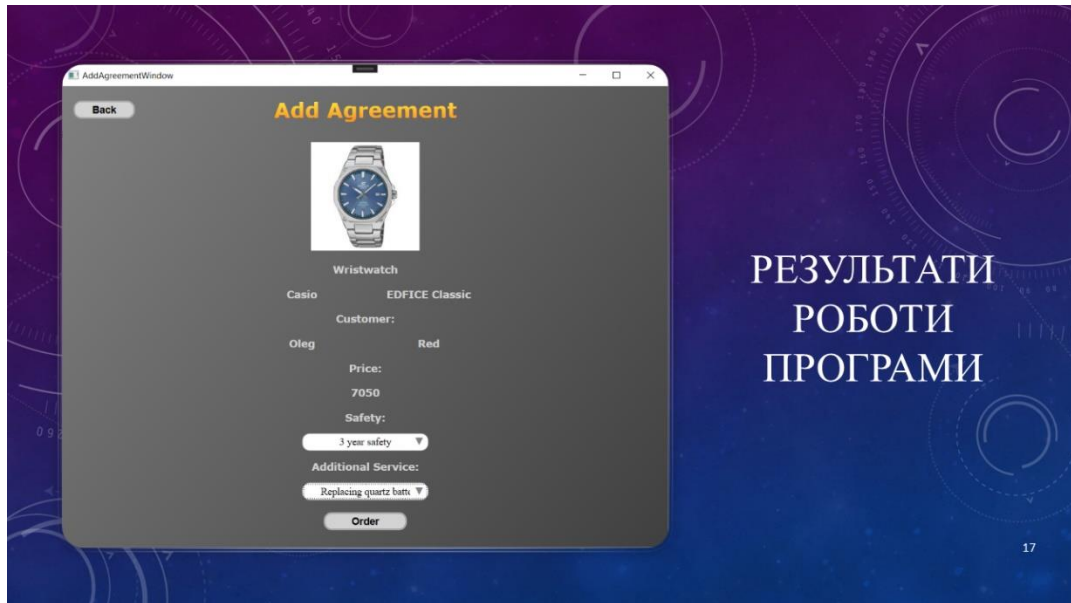


Рисунок Г.17 – Слайд 17

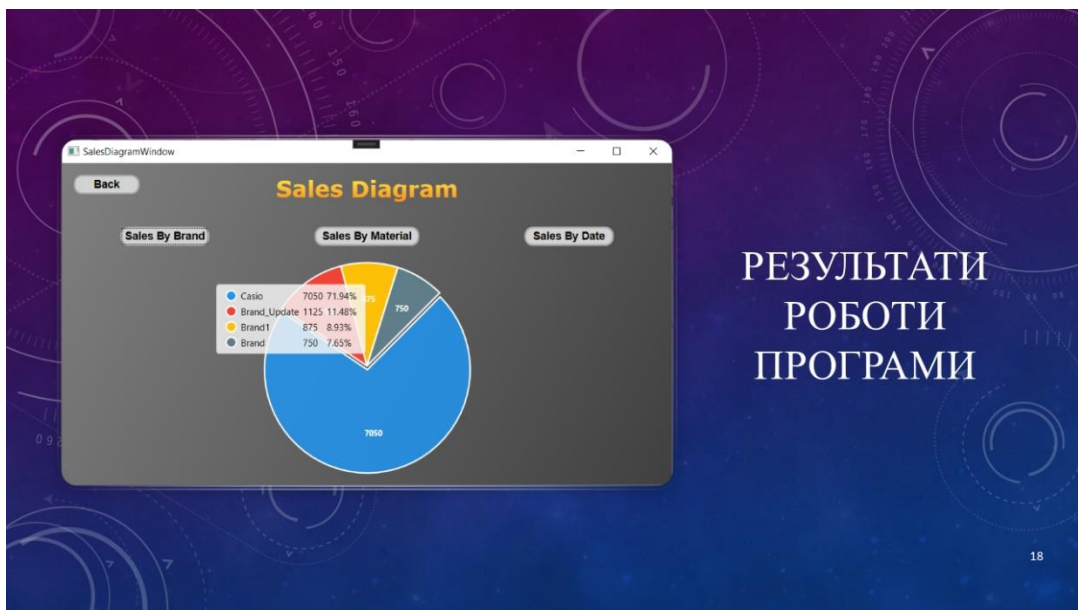


Рисунок Г.18 – Слайд 18

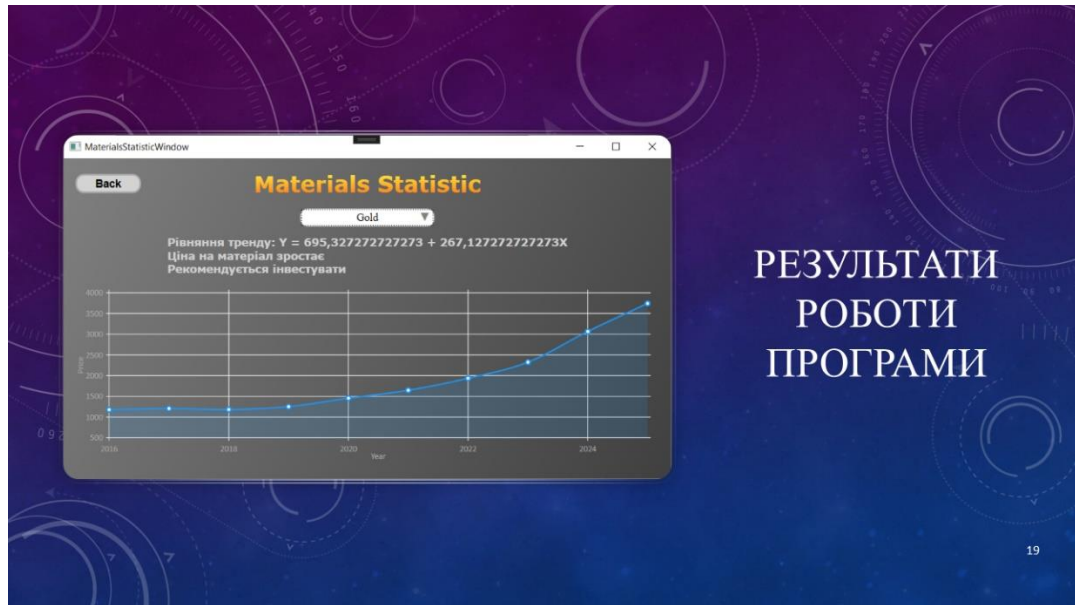


Рисунок Г.19 – Слайд 19

ВИСНОВКИ

Під час виконання дипломної роботи було розроблено програмне забезпечення для адміністрування електронної комерції на прикладі інтернет-продажів наручних годинників. Десктопна програма призначена для ефективного керування товарами, замовленнями, аналізу продажів та зміни тенденції вартості матеріалів з яких виготовлений годинник, швидке та зручне оформлення замовлення для клієнтів сервісу.

Програма реалізована мовою C# з використанням технології WPF, .NET Framework 4.8.1 для забезпечення інтуїтивно зрозумілого графічного інтерфейсу користувача. Дані системи зберігаються у базі даних, яка працює на сервері Microsoft SQL Server. Для підвищення ефективності запитів було використано інструменти Entity Framework для взаємодії з базою даних.

Архітектура розробленої системи передбачає два типи користувачів: адміністраторів магазину (менеджерів) та клієнтів.

Окремою функціональною можливістю є реалізація модуля аналізу вартості дорогоцінного металу, з якого виготовлено наручні годинники, що дозволяє точніше оцінювати тенденцію зміни їх ринкової ціни та радить користувачеві звернути увагу на матеріал, з метою можливості придбання годинника як інвестицію.

Розробка даної програми дозволила автоматизувати процес адміністрування інтернет-магазину наручних годинників, покращила зручність використання для користувачів, підвищила ефективність управління продажами та забезпечила кращу взаємодію між адміністраторами та клієнтами.

Розроблений програмний продукт успішно реалізує всі поставлені задачі та відповідає функціональним вимогам. Впровадження інтеграції з існуючими системами забезпечує високу продуктивність та стабільність роботи програми, а також відповідає стандартам безпеки та зручності користування.

Розроблення даного програмного забезпечення є важливим кроком для розвитку електронної комерції, адже воно сприяє покращенню якості обслуговування продажів та зростанню ефективності роботи сервісів які спеціалізуються на продажах наручних годинників.

20

Рисунок Г.20 – Слайд 20