Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики
Кафедра програмування

Об'єктно орієнтований аналіз і проєктування програмного забезпечення

Виконала:
студентка V курсу, групи ПМоМ-11з
нап. підгот. «Середня освіта (Інформатика)»
Папіж Христина Андріївна

Викладач:
ст. викл. Кущак П.Б.

Львів - 2023

Зміст

Вступ

Об'єктно-орієнтований аналіз і проектування програмного забезпечення (ООАПЗ) - це методика проектування програмного забезпечення, яка базується на об'єктно-орієнтованому підході. Вона дозволяє розбити програму на окремі об'єкти, які мають свої властивості та методи, і взаємодіють між собою за допомогою повідомлень.

ООАПЗ дозволяє покращити процес проектування програмного забезпечення за рахунок розбиття програми на менші, більш зрозумілі та керовані елементи. Крім того, цей підхід дозволяє полегшити розуміння програмного коду та розширення його функціональності.

ООАПЗ включає такі етапи як аналіз вимог, проектування архітектури програми, проектування деталей програми, реалізацію та тестування. Кожен з цих етапів дозволяє покращити розуміння програми та зменшити кількість помилок під час розробки.

Важливим аспектом ООАПЗ є використання патернів проектування - загальних рішень, які можуть бути використані для вирішення різних проблем під час проектування програмного забезпечення. Це дозволяє зменшити кількість помилок, спростити розробку та підтримку програмного забезпечення.

ООАПЗ є важливим інструментом для покращення процесу проектування програмного забезпечення та забезпечення високої якості програм. Цей підхід дозволяє зробити програму більш структурованою, зрозумілою та зручною для розробників та користувачів.

.NET - це платформа розробки програмного забезпечення, створена компанією Microsoft. Основою .NET є спільна мовна інфраструктура (CLI), що дозволяє розробникам використовувати різні мови програмування (такі як C#, VB.NET, F # та інші) для створення програмного забезпечення.

Один з ключових компонентів .NET - це .NET Framework, що містить набір бібліотек та інструментів для розробки десктопних та веб-додатків на мові C# або VB.NET. Однак, з'явився новий підхід для створення програмного забезпечення на базі .NET, він називається .NET Core.

MS SQL (Microsoft SQL Server) - це реляційна база даних, розроблена компанією Microsoft. Вона дозволяє зберігати та управляти даними, забезпечуючи швидкий доступ до інформації та безпеку зберігання даних.

MS SQL є одним з найбільш популярних реляційних баз даних на ринку. Вона має різні версії, включаючи Express, Standard та Enterprise, які мають різні функції та обмеження.

MS SQL має багато корисних функцій, таких як підтримка транзакцій, можливість встановлювати обмеження на дані, можливість виконувати запити для отримання даних з різних таблиць і багато іншого. Крім того, MS SQL має дуже широку підтримку для іншого програмного забезпечення, що дозволяє легко інтегрувати її з іншими системами.

В даній роботі розглянуто процес створення програмного забезпечення на платформі .NET, використовуючи паттерни проєктування та об'єктно-орієнтований підхід. Розглядається постановка задачі, опис завдання та структура проєкту. Також описано розробку бази даних зі структурою та ER-діаграмою, консольний застосунок з рівнем доступу до даних, тестування та апробація. Далі розглядається Windows Presentation Foundation застосунок з бізнес-логікою, аутентифікацією, принципом Dependency Injection, MVVM паттерном проєктування, стилізацією, тестуванням та апробацією. В цьому звіті представлені ключові елементи процесу розробки програмного забезпечення на платформі .NET з використанням паттернів проєктування та об'єктно-орієнтованого підходу.

1 Постановка задачі

1.1 Опис завдання

Зареєстрований користувач інтернет магазину – логується, потрапляє на стартову сторінку з можливістю перейти на сторінку своєї ролі за допомогою меню. Доступний перехід на сторінку історії замовлень. Може виконувати пошук та сортування. Може повторити замовлення або написати відгук на куплений товар. Реалізувано названий функціонал.

1.2 Структура проєкту

- Створено архітектуру бази даних відповідно до ролі.
- Реалізовано рівень доступу до даних за допомогою ADO.NET. Для кожної сутності додано окремий DAL інтерфейс з CRUD операціями та реалізовано відповідний клас.
- Бібліотка з тестами для DAL рівня.
- Реалізовано консольну інтерактивну програму для демонстрації роботи з DAL класами. Продемонстровано основні CRUD операції в них.
- Використовуючи DAL рівень, реалізований в попередній чистині, створено класи для бізнес логіки, які дають доступ до методів, які необхідній ролі.
- Створено окремі бізнес класи для для аутентифікації, реєстрації та зміни користувача.
- Створено проект з тестами для бізнес логіки та аутентифікації.
- Створено WPF проєкт з можливістю аутентифікації, де можна виконувати всі функції ролі. Якщо логується користувач без відповідої ролі, програма не дає доступу до функціональності.
- Використано паттерн Dependency Injection для роботи з бізнес логікою.
- Реалізовано функціональність за допомогою MVVM паттерна проєктування.
- Реалізовано дизайн за допомогою стилів та шаблонів.
- Реалізовано інтерфейс команд в класах ViewModel.
- Реалізовано валідацію даних.

2 База даних

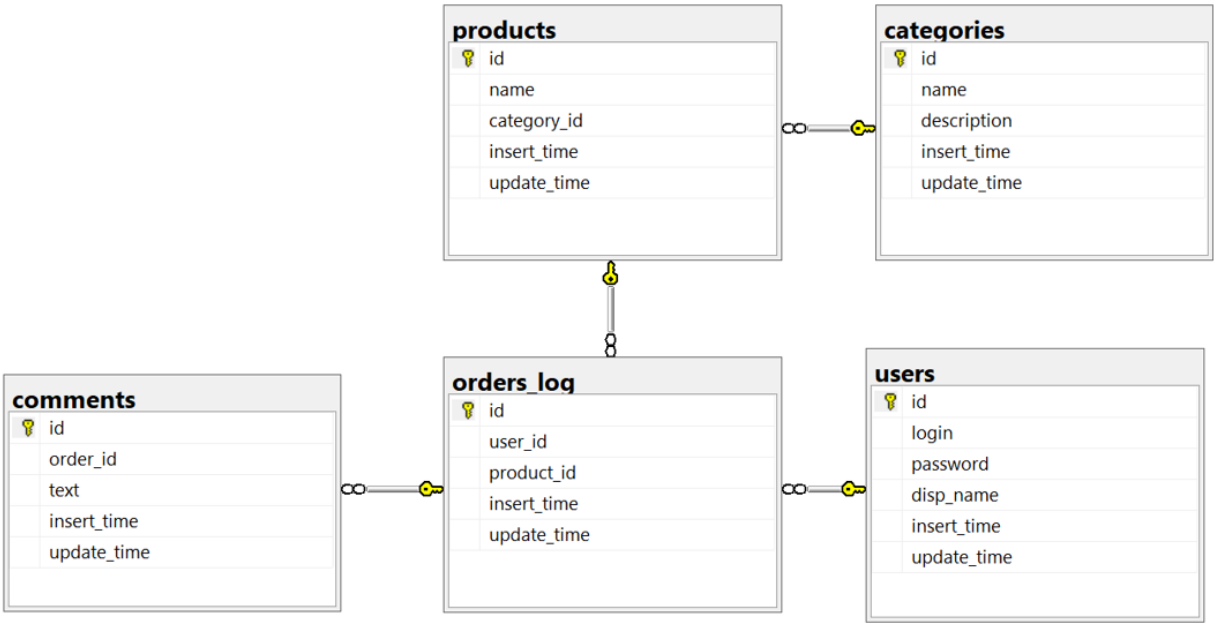
2.1 Структура

Для проєктування бази даних я використовувала систему управління базами даних «Microsoft SQL Server Management Studio».
База даних містить п'ять таблиць:

- Users
- Categories
- Products
- Orders_log
- Comments

Кожна таблиця містить колонки для відслідковування інформації про зміни даних.

2.2 ER-діаграма



3 Консольний застосунок

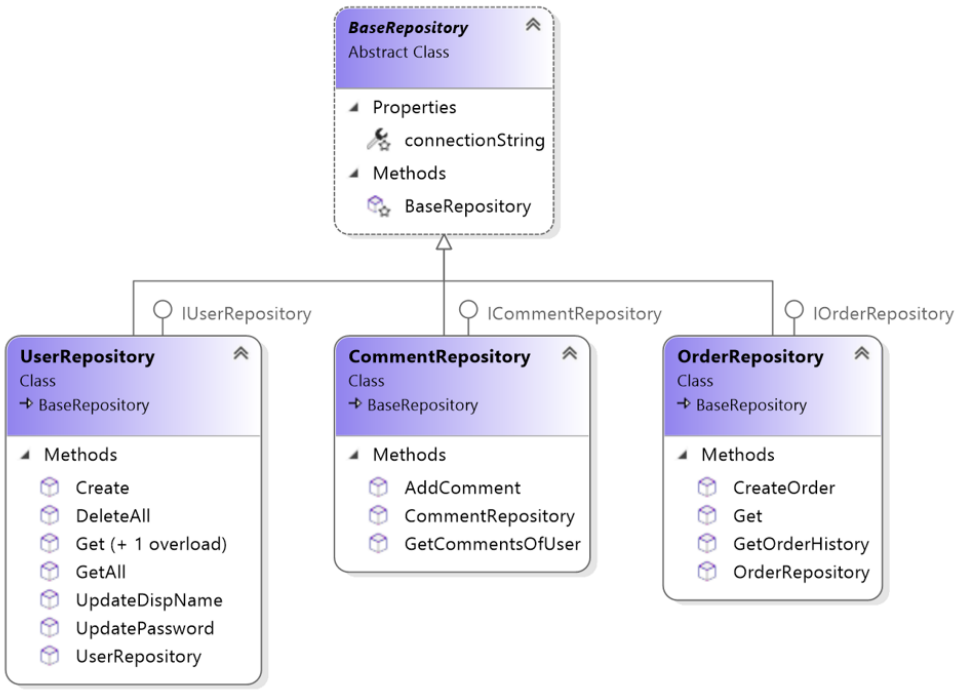
Консольний застосунок - це швидше друже, оскільки він має більш швидкодійний та легший для виконання інтерфейс командного рядка. Зазвичай, консольні застосунки не потребують великих ресурсів, тому вони можуть працювати швидше та ефективніше ніж застосунки з графічним інтерфейсом користувача (GUI).

Крім того, консольні застосунки можуть бути корисними для автоматизації та автоматичної обробки даних, оскільки їх легше скриптувати та інтегрувати з іншими інструментами. Консольні застосунки також можуть бути корисними для роботи з віддаленими серверами, оскільки вони можуть бути легкими та швидкодійними, що зменшує час пересилання даних.

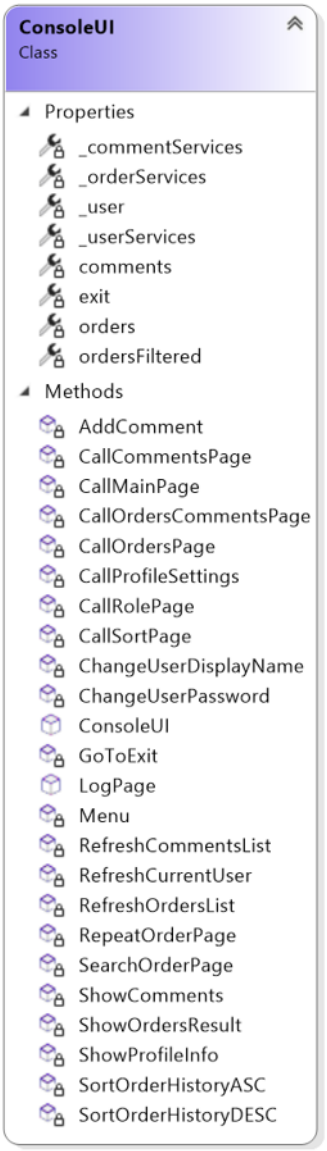
В цілому, консольні застосунки можуть бути більш ефективним та потужним інструментом, особливо якщо вам потрібно швидко та ефективно виконувати рутинні задачі або роботи з даними.

Абстракція

Імплементація



Струкрура



3.1 Рівень доступу до даних

В застосунку я реалізувала рівень доступу до даних за допомогою ADO.NET. Підключення до бази даних відбувається за допомогою «connectionString» з «App.config».

```
<connectionStrings>
  <add name="DB"
       connectionString="Server=DESKTOP-9GGT2EU;Database=Internet-shop; Integrated Security=true"
       providerName="System.Data.SqlClient" />
</connectionStrings>
```

Створила бібліотеку з класами DTO об'єктів. Для кожної сутності додала окремий DAL інтерфейс з CRUD операціями та реалізувала відповідний клас.

Приклад реалізації додавання елемента до бази даних:

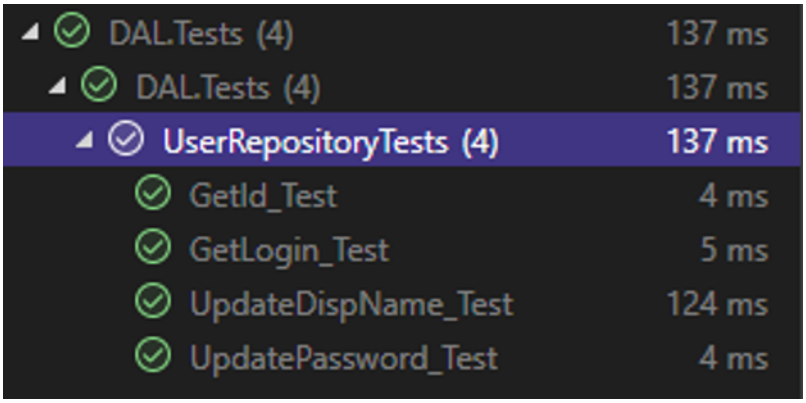
```
public void AddComment(int orderId, string commentText)
{
    using (SqlConnection connection = new(connectionString))
    {
        connection.Open();
        SqlCommand command = new(@"INSERT INTO comments (order_id, text) VALUES (@order_id, @text);", connection);
        command.Parameters.Add(new SqlParameter("@order_id", orderId));
        command.Parameters.Add(new SqlParameter("@text", commentText));
        command.ExecuteNonQuery();
    }
}
```

Приклад реалізації видалення елемента з бази даних:

```
public void DeleteAll()
{
    using (SqlConnection connection = new(connectionString))
    {
        connection.Open();
        SqlCommand command = new(@"DELETE FROM users;", connection);
        command.ExecuteNonQuery();
    }
}
```

3.2 Тестування

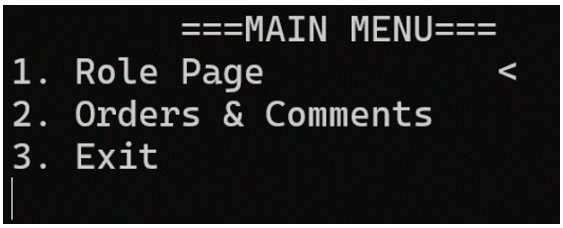
Створила бібліотеку з тестами для DAL рівня та відповідні класи для всіх сутностей:

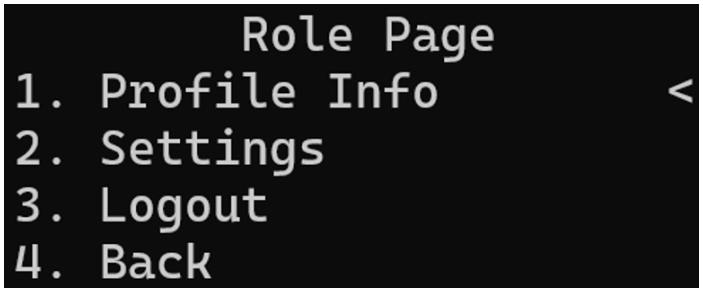


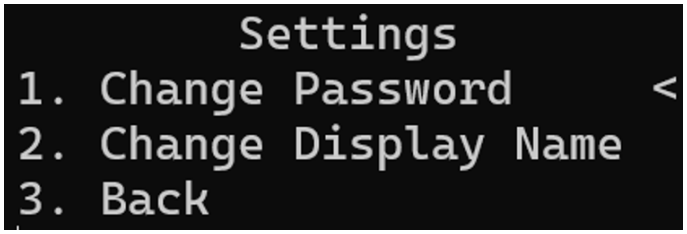
Приклад тестування:

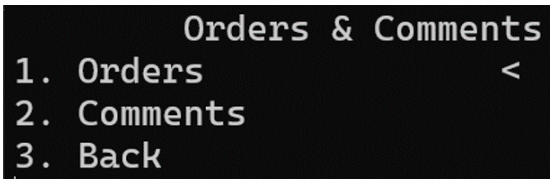
```
[Fact]
public void AddComment_Test()
{
    //Arrange
    CommentRepository commentRepository = new CommentRepository(_connectionString);
    CommentDTO expected = new CommentDTO
    {
        Id = 5,
        OrderId = 1,
        Text = "Nice Burger!"
    };
    commentRepository.AddComment(expected.OrderId, expected.Text);
    //Act
    var actual = commentRepository.GetCommentsOfUser(1)[0];
    //Assert
    Assert.Equal(actual.Text, expected.Text);
}
```

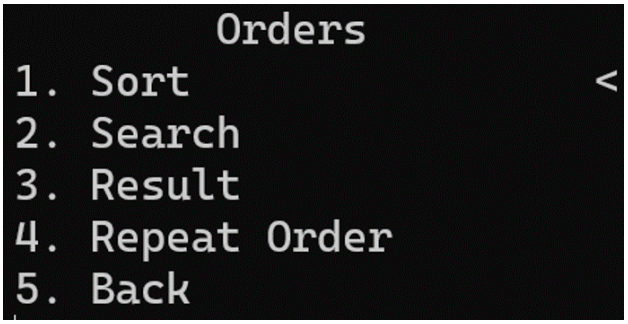
3.3 Апробація

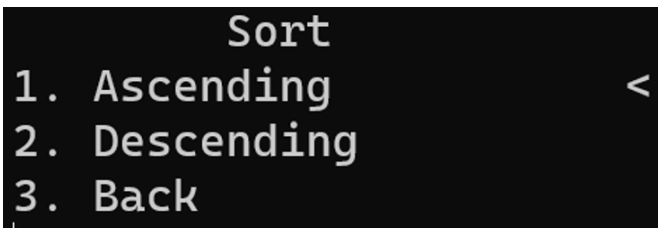


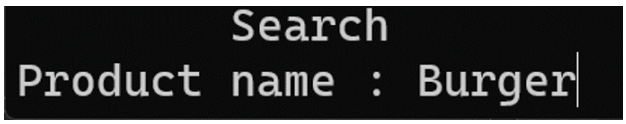


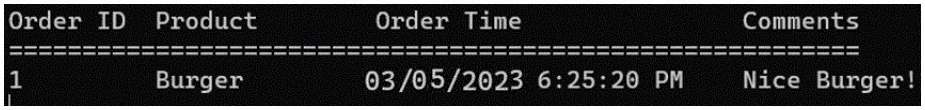


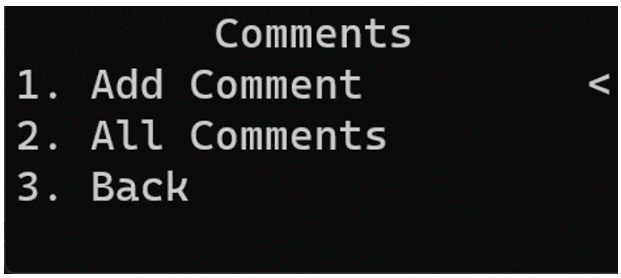


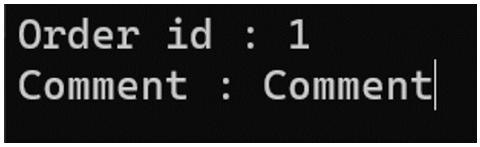












Order ID	Order Time	Text
1	03/05/2023 6:25:20 PM	Nice Burger!
2	03/05/2023 6:25:20 PM	Cool Laptop!
1	03/05/2023 12:08:03 PM	Comment

4 Windows Presentation Foundation застосунок

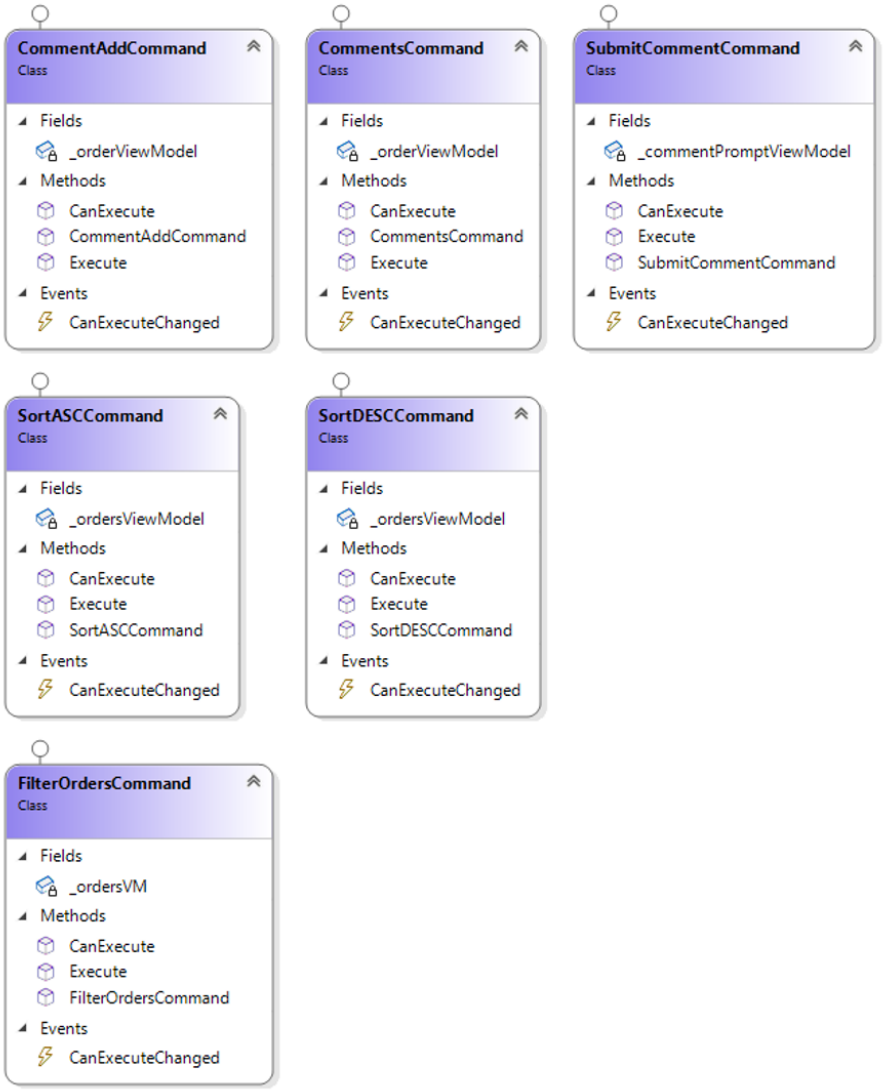
Windows Presentation Foundation (WPF) - це технологія для розробки графічних інтерфейсів користувача для десктопних застосунків у середовищі Windows. WPF була випущена компанією Microsoft у 2006 році і є частиною платформи .NET Framework.

WPF пропонує розробникам можливість створювати багатофункціональні та привабливі графічні інтерфейси користувача, що включають в себе ефекти, анімацію, 3D-зображення та інші функції. WPF використовує XAML, що є декларативною мовою розмітки для визначення інтерфейсу користувача, та має можливості для розширення.

WPF також має вбудовану підтримку для роботи з даними, включаючи зв'язування даних та валідацію, що дозволяє легко робити роботу з базами даних і іншими джерелами даних.

У цілому, WPF є потужним інструментом для розробки графічних інтерфейсів користувача в середовищі Windows з багатьма можливостями для налаштування та розширення, що дозволяє створювати привабливі та функціональні десктопні застосунки.

Структура



Приклад коду (FilterOrdersCommand)

```
public class FilterOrdersCommand : ICommand
{
    OrdersViewModel _ordersVM;
    public FilterOrdersCommand(OrdersViewModel ordersViewModel)
    {
        _ordersVM = ordersViewModel;
    }

    public event EventHandler? CanExecuteChanged;

    public bool CanExecute(object? parameter)
    {
        return true;
    }

    public void Execute(object? parameter)
    {
        _ordersVM.Orders = _ordersVM._orderServices.GetOrderHistory(Context.User.Id).Where(o => o.ProductName.Contains(_ordersVM.Search));
        if (_ordersVM.Sort == ESort.ASC)
            _ordersVM.Orders = _ordersVM.Orders.OrderBy(o => o.Id);
        else
            _ordersVM.Orders = _ordersVM.Orders.OrderByDescending(o => o.Id);

    }
}
```

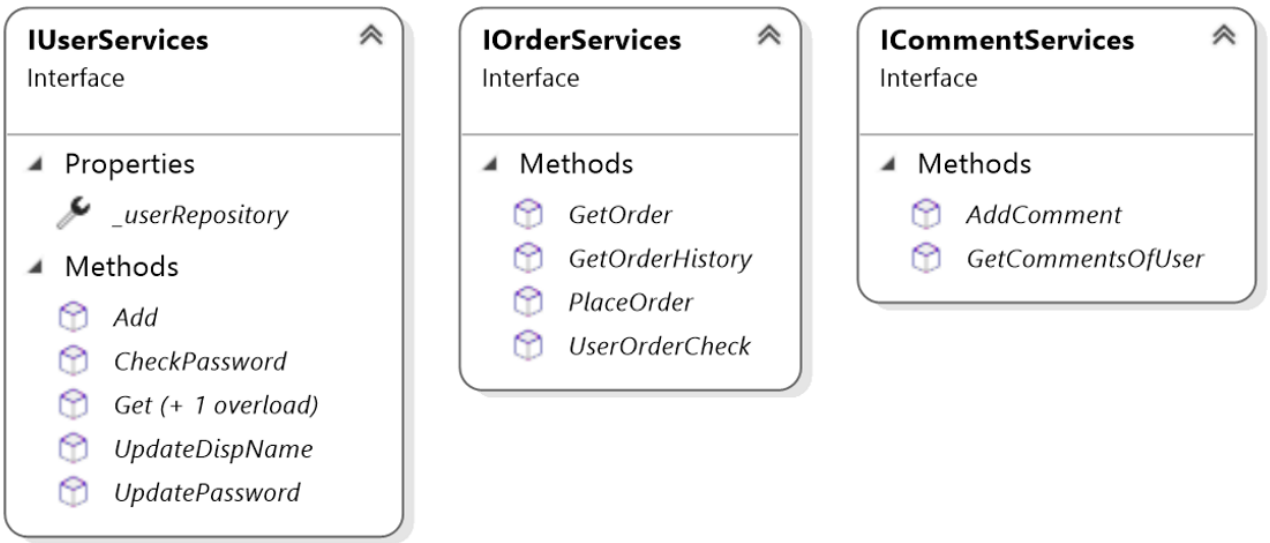
4.1 Бізнес-логіка

Використовуючи DAL рівень реалізований в попередній частині, створено класи для роботи з бізнес-логікою, які дають доступ до методів, що необхідні для ролі.

```
public bool UserOrderCheck(int userId, int orderId)
{
    OrderDTO order;
    if ((order = _orderRepository.Get(orderId)) != null)
        if (order.UserId == userId)
            return true;
    return false;
}
```

```
public void AddComment(int orderId, string commentText) => _commentRepository.AddComment(orderId, commentText);
public List<CommentDTO> GetCommentsOfUser(int id) => _commentRepository.GetCommentsOfUser(id);
```

Абстракція



4.2 Аутентифікація

Створила окремий бізнес-клас для аутентифікації:

```
public static string Hash(string password)
{
    using (var algorithm = new Rfc2898DeriveBytes(password, c_SaltSize, c_NumberOfIterations, HashAlgorithmName.SHA256))
    {
        var key = Convert.ToBase64String(algorithm.GetBytes(c_KeySize));
        var salt = Convert.ToBase64String(algorithm.Salt);
        return $"{c_NumberOfIterations}.{salt}.{key}";
    }
}

public static bool Check(string hash, string password)
{
    string[] parts = hash.Split('.', 3);
    if (parts.Length != 3)
        throw new FormatException("Unexpected hash format. " + "Should be formatted as `{iterations}.{salt}.{hash}`");
    int iterations = Convert.ToInt32(parts[0]);
    byte[] salt = Convert.FromBase64String(parts[1]);
    byte[] key = Convert.FromBase64String(parts[2]);
    using (var algorithm = new Rfc2898DeriveBytes(password, salt, iterations, HashAlgorithmName.SHA256))
        return algorithm.GetBytes(c_KeySize).SequenceEqual(key);
}
```

4.3 Принцип Dependency Injection

Dependency Injection - це принцип програмування, що дозволяє знизити залежність між компонентами програми. Цей принцип може бути особливо корисним для роботи з бізнес-логікою, оскільки дозволяє відокремити логіку від реалізації.

Зазвичай, бізнес-логіка включає в себе функції, що залежать від зовнішніх ресурсів, таких як база даних, зовнішні сервіси, файлові системи тощо. Dependency Injection дозволяє передавати ці залежності у вигляді аргументів функції або через конструктор класу, замість того, щоб вбудовувати їх безпосередньо у код.

Це дозволяє збільшити гнучкість та розширюваність програми, оскільки залежності можуть бути легко замінені або модифіковані без необхідності змінювати код бізнес-логіки. Також це сприяє полегшенню тестування, оскільки залежності можуть бути замінені на тестові варіанти без необхідності змінювати код самої логіки.

Dependency Injection може бути реалізований різними способами, наприклад, за допомогою контейнерів DI або через власноручне введення залежностей. У будь-якому випадку, використання Dependency Injection може полегшити роботу з бізнес-логікою та зробити програму більш гнучкою та розширюваною.

Приклад коду

```
protected override void Configure()
{
    _container = new SimpleContainer();

    _container.Singleton<IWindowManager, WindowManager>()
              .Singleton<IEventAggregator, EventAggregator>();

    _container.Singleton<IUserServices, UserServices>()
              .Singleton<IOrderServices, OrderServices>()
              .Singleton<ICommentServices, CommentServices>()
              .Singleton<IUserRepository, UserRepository>()
              .Singleton<IOrderRepository, OrderRepository>()
              .Singleton<ICommentRepository, CommentRepository>();

    _container.PerRequest<ShellViewModel>()
              .PerRequest<HomeViewModel>()
              .PerRequest<ProfileViewModel>()
              .PerRequest<OrdersViewModel>()
              .PerRequest<LobbyViewModel>()
              .PerRequest<LoginViewModel>()
              .PerRequest<RegisterViewModel>()
              .PerRequest<CommentPromptViewModel>();
}
```

4.4 MVVM паттерн проєктування

MVVM (Model-View-ViewModel) - це паттерн проєктування програмного забезпечення, що дозволяє розділити логіку застосунку на три основні компоненти: модель, представлення та відображення.

Модель представляє собою об'єкти даних, які використовуються в програмі. Представлення відповідає за відображення даних на екрані та взаємодію з користувачем. ViewModel - це посередник між моделлю та представленням, який забезпечує зв'язок між цими компонентами та обробляє всі необхідні операції, які пов'язані зі зміною даних чи взаємодією з користувачем.

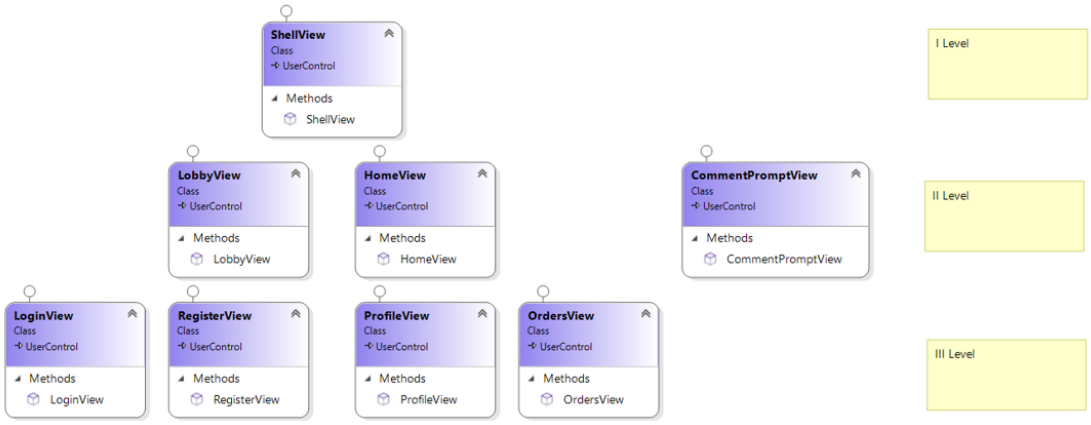
Один з основних принципів MVVM полягає в тому, що ViewModel не залежить від представлення, що дозволяє забезпечити більшу гнучкість та розширюваність програми. Крім того, ViewModel може бути легко тестованим, оскільки його можна відокремити від реального представлення та використовувати тестові дані для виконання тестів.

MVVM може бути використаний з різними технологіями та фреймворками, включаючи WPF, Xamarin та Silverlight. У випадку з WPF, MVVM забезпечує зв'язок між View та ViewModel за допомогою дата-байндингу, що дозволяє забезпечити більшу гнучкість та розширюваність програми.

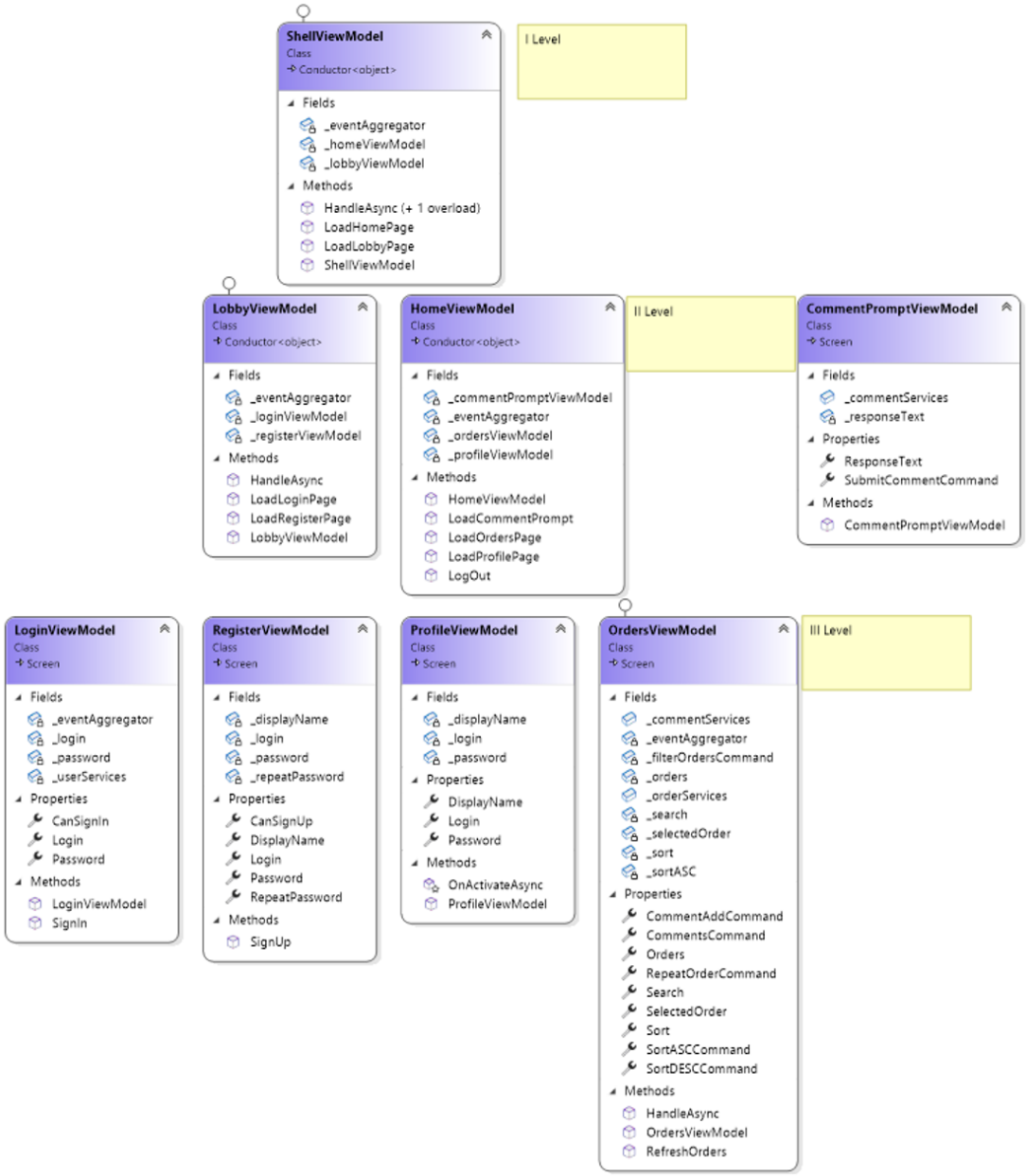
Загалом, використання паттерна проєктування MVVM дозволяє зробити програму більш гнучкою, розширюваною та тестируємою, забезпечуючи чітку розділеність між компонентами програми та зменшуючи залежності між ними.

Структура

- Views



- View Models



Приклад коду (Shell) ShellViewModel

```
public class ShellViewModel : Conductor<object>, IHandle<LogInEvent>, IHandle<LogOutEvent>
{
    private LobbyViewModel _lobbyViewModel;
    private HomeViewModel _homeViewModel;
    private IEventAggregator _eventAggregator;

    public ShellViewModel(LobbyViewModel lobbyViewModel, HomeViewModel homeViewModel, IEventAggregator eventAggregator)
    {
        _lobbyViewModel = lobbyViewModel;
        _homeViewModel = homeViewModel;
        _eventAggregator = eventAggregator;
        _eventAggregator.Subscribe(this);
        LoadLobbyPage();
    }

    public async void LoadLobbyPage()
    {
        await ActivateItemAsync(_lobbyViewModel);
    }

    public async void LoadHomePage()
    {
        await ActivateItemAsync(_homeViewModel);
    }

    public Task HandleAsync(LogInEvent message, CancellationToken cancellationToken)
    {
        LoadHomePage();
        return new Task(() => { });
    }

    public Task HandleAsync(LogOutEvent message, CancellationToken cancellationToken)
    {
        LoadLobbyPage();
        return new Task(() => { });
    }
}
```

```
<UserControl x:Class="WPFGUI.Views.ShellView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:WPFGUI.Views"
             mc:Ignorable="d"
             d:DesignHeight="500" d:DesignWidth="500" Background="Turquoise">
    <ContentControl x:Name="ActiveItem" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"/>
</UserControl>
```

4.5 Стилізація

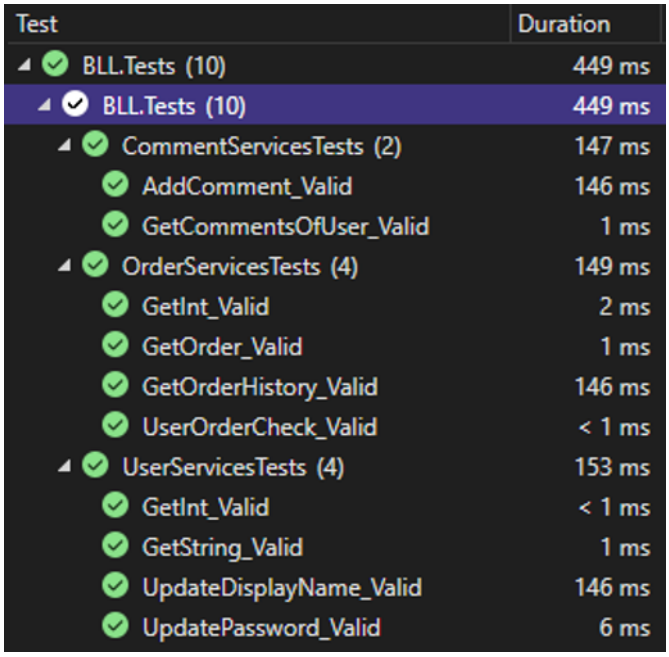
У WPF (Windows Presentation Foundation) стилі використовуються для визначення вигляду та поведінки елементів керування. Стилі в WPF дозволяють використовувати однаковий вигляд для різних елементів керування, що зменшує час, потрібний для розробки програми та дозволяє забезпечити єдність дизайну.

```
<ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Light.xaml" />
<ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Defaults.xaml" />
<ResourceDictionary Source="pack://application:,,,/MaterialDesignColors;component/Themes/Recommended/Primary/MaterialDesignColor.DeepPurple.xaml" />
<ResourceDictionary Source="pack://application:,,,/MaterialDesignColors;component/Themes/Recommended/Accent/MaterialDesignColor.Lime.xaml" />
```

```
<StackPanel>
    <TextBox x:Name="Login" materialDesign:HintAssist.Hint ="Enter Login" Style="{StaticResource MaterialDesignFloatingHintTextBox}"/>
    <TextBox x:Name="Password" materialDesign:HintAssist.Hint ="Enter Password" Style="{StaticResource MaterialDesignFloatingHintTextBox}"/>
    <Button  x:Name="SignIn" Content="Login" Margin="0 10 0 0"/>
</StackPanel>
```

4.6 Тестування

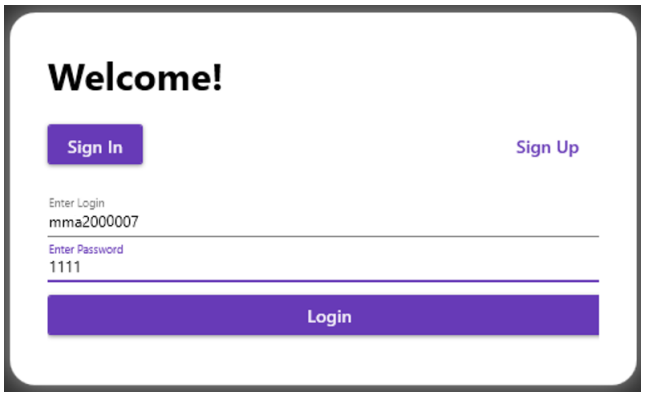
Створила проєкт для тестування бізнес-логіки та аутентифікації:

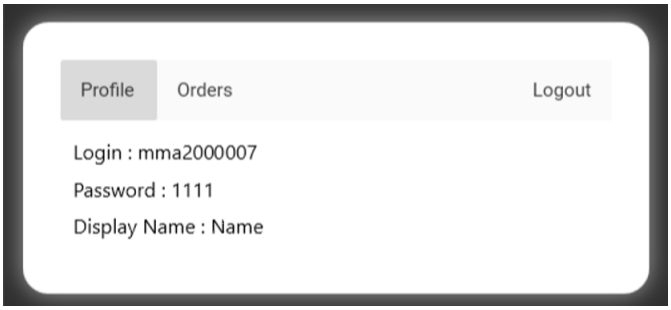


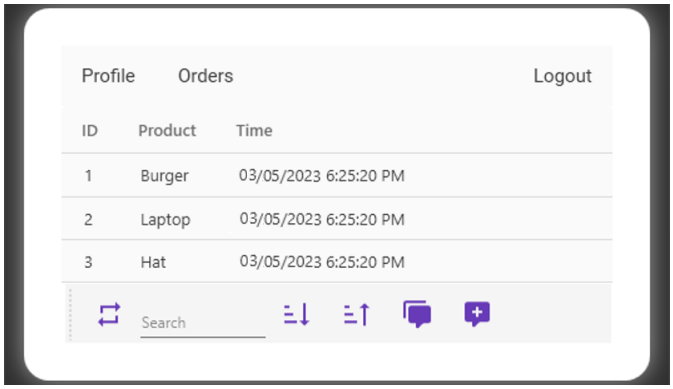
Приклад тестування:

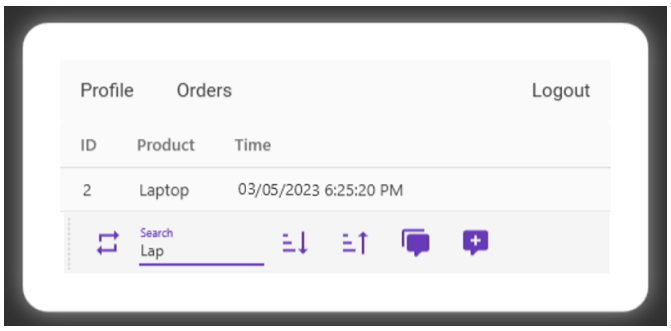
```
[Fact]
public void CreateOrder_Test()
{
    //Arrange
    OrderRepository orderRepository = new OrderRepository(_connectionString);
    OrderDTO expected = new OrderDTO
    {
        UserId = 1,
        ProductId = 1,
        ProductName = "Burger"
    };
    orderRepository.CreateOrder(expected.UserId, expected.ProductId);
    //Act
    OrderDTO actual = orderRepository.Get(1);
    //Assert
    Assert.Equal(actual.ProductId,    expected.ProductId);
    Assert.Equal(actual.ProductName, expected.ProductName);
    Assert.Equal(actual.UserId, expected.UserId);
}
```

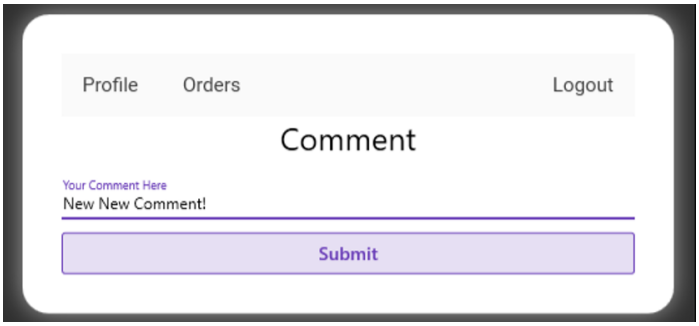
4.7 Апробація











Висновки

ADO.NET ускладнює програму, але водночас дає більшу гнучкість ніж Entity Framework. Розмежування рівня доступу до бази даних з шаром презентації дозволяє змінювати графічну складову без необхідності внесення змін у внутрішні процеси додатка, а використання інтерфейсів дозволяє цю незалежність підсилити. Код який буде використаний повторно і слугуватиме ярдом аплікації, варто тестувати навіть якщо на перший погляд він функціонує коректно і найближчим часом в ньому не плануються зміни. Тестування краще виконувати з використанням транзакції, так як це більш зручний спосіб ніж робота з окремою базою даних.
Робота з консоллю є найлегшою для розробника, але не надто орієнтована на кінцевого користувача.

Рівень бізнес логіки дає змогу об'єднати комплексні операції з базою/базами даних у вигляді методів і надає зручний доступ до них. У разі використання шару бізнес логіки, відповідні класи повинні бути організовані для кожної сутності що бере безпосередню участь в додатку. Збережені процедури дозволяють перенести логіку роботи з даними в саму базу даних.
Moq-тести надають зручний спосіб тестування при якому одиниці роботи програми звільнені від зовнішніх чинників і залежностей.

WPF надає потужний інструментарій для створення графічного інтерфейсу, даючи змогу розробникам і дизайнерам працювати незалежно. В свою чергу паттерн MVVM спрощує взаємодії між елементами додатку роблячи їх інтуїтивно зрозумілішими для розробника і підвищуючи читабельність коду. А використання стилів надає загально ствердженні рішення для створення зрозумілого і естетичного користувацького інтерфейсу.

Вдосконалила свої вміння та навички в роботі з базами даних та мовою програмування C#. Зрозуміла, що розробку застосунку варто розпочинати з детального планування, а його реалізацію виконувати поступово, стараючись писати код з поглядом на майбутні версії аплікації, роблячи його простим та читабельним використовуючи патерни проєктування.

Література

[1] Cloud Design Patterns [Електронний ресурс] / Microsoft 2023 – Режим доступу: https://learn.microsoft.com/en-us/azure/architecture/patterns/

[2] Resilience by design for cloud services [Електронний ресурс] / Microsoft 2023 – Режим доступу: http://download.microsoft.com/download/D/8/C/D8C599A4-4E8A-49BF-80EEFE35F49B914D/Resilience_by_Design_for_Cloud_Services_White_Paper.pdf

[3] Learn C# [Електронний ресурс] / Microsoft 2023 – Режим доступу:https://dotnet.microsoft.com/en-us/learn/csharp

[4] *Robert C. Martin.* "Principles Of OOD 2017.

[5] *Будай А.* Дизайн-патерни — просто, як двері. - 2012.

Додаток

З програмною реалізацією проєкту можна ознайомитись за посиланням на GitHub репозиторій.

`https://github.com/papizhkhris/internet-shop`