МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП’ЮЕТРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**Курсова робота**

З дисципліни «**Архітектура та проектування програмного забезпечення**»

Варіант №3

Виконали: студенти ПІ-218Б

Крикля В.А.

Маковський О.І

Гацоєва У.В

Прийняла: Скалова В.A.

Київ – 2020

**ЗМІСТ РОБОТИ**

**Зміст****...........................................................................................................2**

**Завдання....................................................................................................3**

**Варіант.......…….…..…….…….……………………………..….………..6**

**Загальна архітектура проекту з поясненням......................................6**

**Описання процесу розробки ..................................................................7**

**Діаграма проекту поясненнями.............................................................8**

**Інструкції з використання застосування…………………………….…9**

**Висновки………………………………………………………………...….11**

**Використані джерела……………………………………...………………12**

***Завдання***

1. Спроектувати Web-застосування у відповідності з принципами багатошарової архітектури програмних систем та скласти проектну документацію.

1.1. Описати загальну архітектуру застосування, призначення її шарів та зв’язки між шарами (рівнями).

1.2. Вихідний код програмної системи представляє сукупність проектів різного типу, поєднаних одним рішенням. Описи типів розташовуються в окремих файлах. При необхідності бажано створити додаткові папки проекту, щоб код проекту був логічно структурований.

1.3. Представити діаграми класів кожного рівня, використовуючи UML-нотації.

2. Розробити застосування на мові C#, яке відповідає вимогам у варіанті. Відокремити рівні доступу до даних, бізнес-логіки та представлення.

2.1. Верхній рівень – представлення (UI), призначений для взаємодії з користувачем. Реалізувати окремим проектом. Для створення Web-інтерфейсу використати технологію ASP.NET WebAPI. На рівні UI повинні бути тільки операції взаємодії з користувачем. Код UI повинний бути максимально простим, неперевантаженим великою кількістю операцій. Ця частина системи (Front End) може бути реалізована будь-яким способом (для стилістичного оформлення допускається використання будь-яких фреймворків та бібліотек, наприклад, Bootstrap). Дані, з якими працює рівень представлення, повинні зберігатись в окремих моделях цього рівня (маються на увазі власні класи (типи) рівня, не запозичені з інших рівнів). Передбачити контроль\перевірку даних, введених користувачем (обов’язково заповнені текстові поля, довжина введених даних, тощо).

2.2. Проміжний рівень – бізнес-логіка, реалізований як динамічна бібліотека. На цьому рівні реалізується саме функціонал застосування, описаний у варіанті. Доречним буде застосувати відомі принципи та шаблони проектування. Для виконання операцій бізнеслогіки передбачити перевірку виняткових ситуацій - виключень. При необхідності створити власні класи виключень.

2.3. Нижній рівень – шар доступу до даних у вигляді бібліотеки. Збереження даних програмної системи виконується у реляційній БД під керуванням СУБД MS SQL. Для взаємодії зі сховищем даних використати ORM ADO .Net Entity Framework (code first). Доступ до даних для шару бізнес-логіки організувати через репозиторії, поєднані у одиницю роботи (їм відповідають шаблони проектування Repository та Unit of Work (UoF). Репозиторії надають доступ до набору сутностей (entities) певного типу. Одиниця роботи (UoW) є точкою єдиного доступу до репозиторіїв та контексту Entity Framework.

3. Шари взаємодіють між собою за наступним принципом: представлення використовує бізнеслогіку, бізнес-логіка – рівень доступу до даних. Для передачі даних крізь шари використовується технологія відображення (mapping).

4. При необхідності для більшої ізоляції основних рівнів можуть вводитися додаткові рівні (наприклад, винесення Repository та UoW).

5. Для ізоляції рівнів використати DI (Ninject чи Autofac).

6. Створити модульні тести для перевірки працездатності коду

6.1. За допомогою бібліотек NUnit чи XUnit написати модульні тести до бізнес логіки. Мінімальне покриття тестами – 50%. Покриття продемонструвати відповідними засобами, наприклад AxoCover, CodeCoverage та ін. Модульні тести повинні бути окремим проектом в рішенні.

6.2. Для оформлення коду модульних тестів овоб’язково застосовувати принцип Triple A.

6.3. За допомогою DI забезпечити, щоб тести для перевірки методів роботи з даними, ніяким чином не впливали на ці дані, використавши для даних об’єкти-емулятори чи замінювачі реальних даних (mock, stub).

7. Для визначення об’єму робіт та декомпозиції задач використати систему розподілу (відслідковування) задач – Board у Visual Studio Online/ Azure DevOps Services (dev.azure.com). Для комплексної ітеративної розробки окремих компонентів застосування та спільного їх збору використовувати репозиторій – Repos у Visual Studio Online/ Azure DevOps Services.

8. Діаграма(-и) та вихідний код повинні відповідати основним принципам проектування: OOP, SOLID, Law of Demeter (LoD), DRY, YAGNI, KISS, cohesion – coupling, inheritance with caution. За невідповідність принципам оцінка за курсову може бути знижена.

9. При написанні коду притримуватися C#/.NET Code Conventions та кращих практик написання коду: іменування класів, об’єктів, властивостей, методів, інші назви повинні бути зрозумілими та відповідати їх задачам, форматувати код, не використовувати magic numbers, а також ставити мінімально необхідний модифікатор доступу (public повинен бути обґрунтований, а не використаний для всіх членів класу та класів проектів без виключення). За неохайне оформлення коду можливе зниження оцінки.

10. Зміст курсової роботи:

10.1. Титульний лист

10.2. Зміст

10.3. Загальне завдання та варіант

10.4. Загальна архітектура проекту з поясненням

10.5. Описання процесу розробки з використанням системи розподілу версій та відслідковування завдань

10.6. Діаграми кожного рівня з поясненнями

10.7. Важливі деталі реалізації проекту

10.8. Інструкції з використання застосування

10.9. Висновки

10.10. Використані джерела

11. Вихідний код додавати в пояснювальну записку курсової роботи не потрібно. Виключення можуть бути тільки для пояснення важливих деталей реалізації, але це можуть бути тільки невеликі фрагменти коду в кілька рядків.

12. Виконувати один варіант повинно не менше 2-х студентів з однієї підгрупи та не більше 3-х. В підгрупі варіанти не повинні повторюватися.

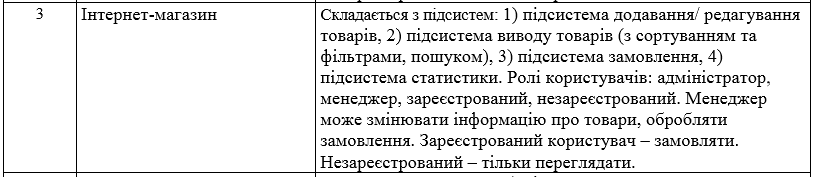
13. Перед початком розробки створити ПРИВАТНИЙ проект і відкрити для викладача доступ до нього у Visual Studio Online/ Azure DevOps Services (ніби він є ще одним членом команди).

14. Назва проекту повинна бути у форматі «ГРУПА-ПІДГРУПА-ВАРІАНТ». Наприклад, PI-215-14. В підгрупі варіанти повторюватися не повинні.

15. Під час захисту КР продемонструвати репозиторій системи контролю версій, пояснити стратегію створення гілок. А також показати систему розподілу задач та пояснити розподіл задач та процес їх виконання.

16. Для демонстрації роботи застосування допускається використовувати засоби тестування та відладки Web API (Fiddler, Postman) замість повністю реалізованого UI.

***Варіант***



***Загальна архітектура проекту з поясненням***

**Опис шарів проекту**

Концептуально проект складаєтся з трьох шарів – а саме, Data Access Layer, Business Logic та Presentation Layer.

**Data Access Layer** – шар низькорівневої роботи з MS SQL базою данних через технологію Entity Framework 6. При створенні та підключенні до бази даних використовувався принцип Database First.

**Business Logic** – шар, відповідальний за обробку даних, отриманих з Data Access Layer. Реалізований згідно з патерном узагальненого репозиторію та в рамках контрактного програмування. Містить сутності, характерні для рівня Business Logic та Presentation Layer.

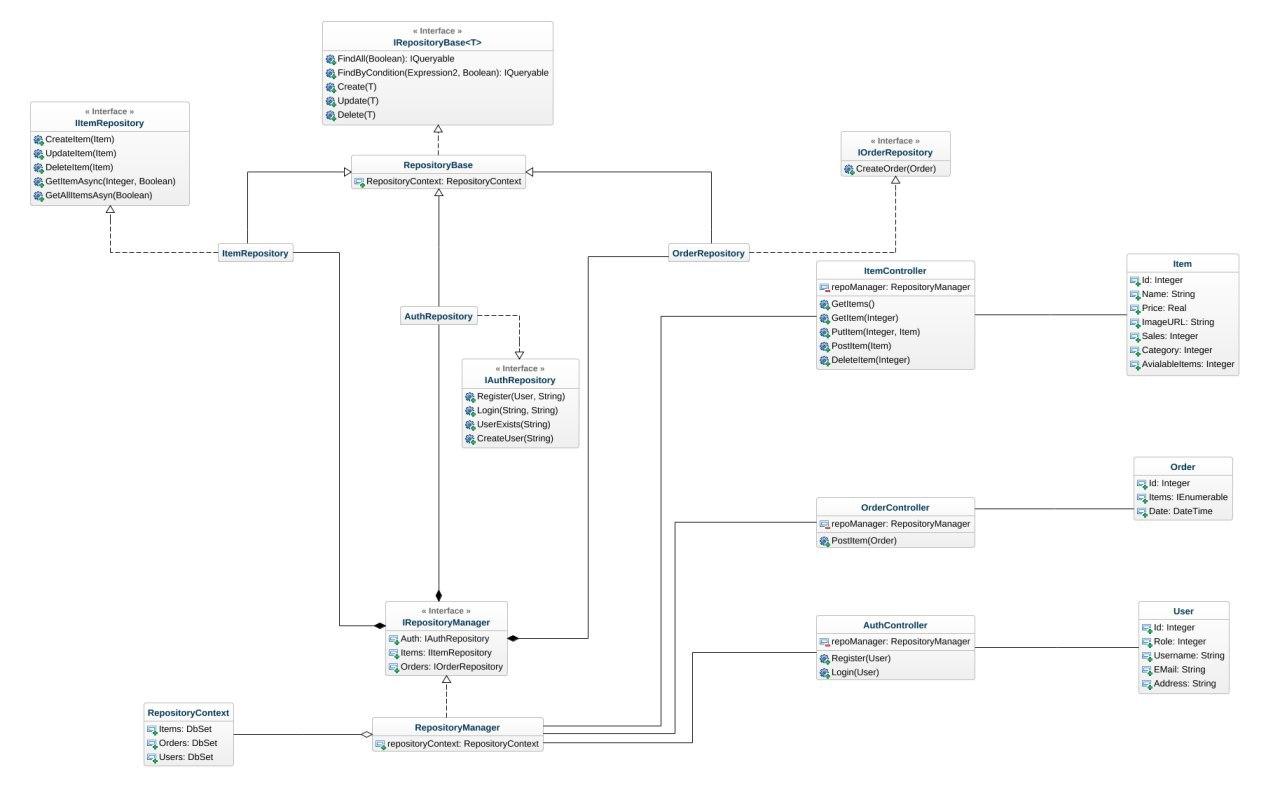
**Presentation Layer** – рівень, відновідальний за відображення інформації користовачеві та введення інформації. Взаємодіє з Business Logic. Перевіряє коректність ввенедої інформації. У разі невідповідності генерує помилку (exeption).

**Модульні тести**

Принцип Triple A був застосований до коду одиничних тестів.

AAA (Порядок, Закон, Затвердження) - це чітке розподіл у випробуванні етапу підготовки (Упорядкувати), вплив на об'єкт (Закон) та виклад результату (Затвердження)

**MsTest -** фреймворк юніт-тестування від компанії Microsoft, який за замовчуванням включений в Visual Studio і який також можна використовувати з .NET Core



***Описання процесу розробки з використанням системи розподілу версій та відслідковування завдань***

В процесі розробки ми використовувати git - розподілену систему керування версіями файлів та спільної роботи. Для зручної роботи ми створили дві гілки master і dev. На гілці master знаходилась стабільна версія проекту, на гілку dev ми коммітили останні зміни. Кожен член команди використовував локальний репозиторій для роботи і з проектом. Для синхронізації проекту ми використовували віддалений репозиторій на Github.

Git — розподілена система керування версіями файлів та спільної роботи. Проект створив Лінус Торвальдс для управління розробкою ядра Linux, а сьогодні підтримується Джуніо Хамано (англ. Junio C. Hamano). Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і комітів.

Прикладами проектів, що використовують Git, є ядро Linux, Android, LibreOffice, Cairo, GNU Core Utilities, Mesa 3D, Wine, багато проектів з X.org, XMMS2[en], GStreamer, Debian DragonFly BSD, Perl, Eclipse, GNOME, KDE, Qt, Ruby on Rails, PostgreSQL, VideoLAN, PHP, One Laptop Per Child (OLPC), АБІС Koha, GNU LilyPond та ELinks і деякі дистрибутиви GNU/Linux (див. нижче).

Програма є вільною і випущена під ліцензією GNU GPL версії 2.

Система спроектована як набір програм, спеціально розроблених з врахуванням їхнього використання у скриптах. Це дозволяє зручно створювати спеціалізовані системи управління версіями на базі Git або користувацькі інтерфейси. Наприклад, Cogito[en] є саме таким прикладом фронтенда до репозиторіїв Git. А StGit використовує Git для управління колекцією латок.

Система має ряд користувацьких інтерфейсів: наприклад, gitk та git-gui розповсюджуються з самим Git.

Віддалений доступ до репозиторіїв Git забезпечується git-демоном, SSH або HTTP сервером. TCP-сервіс git-daemon входить у дистрибутив Git і є разом з SSH найпоширенішим і надійним методом доступу. Метод доступу HTTP, незважаючи на ряд обмежень, дуже популярний в контрольованих мережах, тому що дозволяє використання існуючих конфігурацій мережевих фільтрів.

GitHub — один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Існують безкоштовні та платні тарифні плани користування сайтом. Базується на системі керування версіями Git і розроблений на Ruby on Rails і Erlang компанією GitHub, Inc (раніше Logical Awesome).

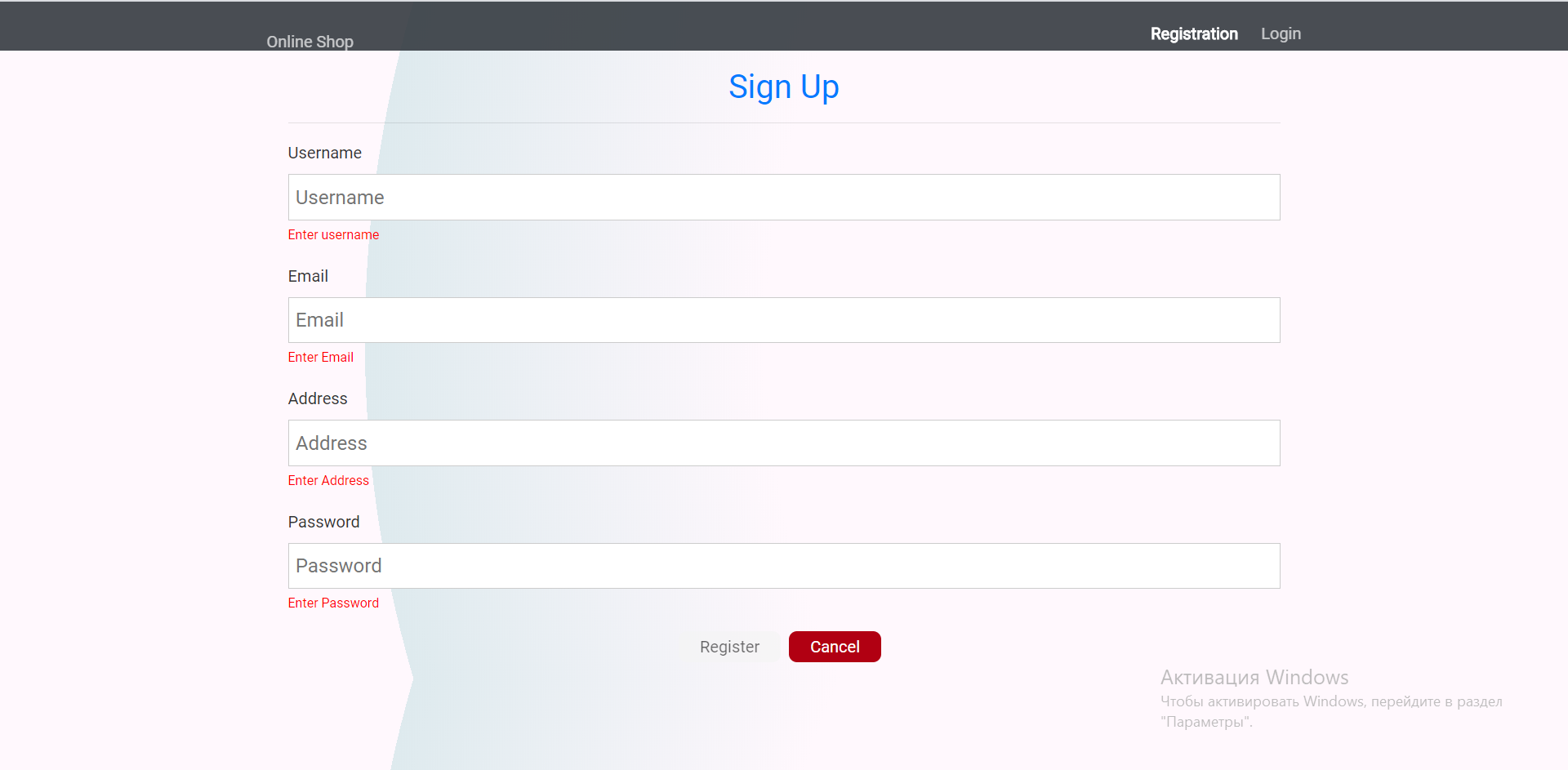
Сервіс безкоштовний для проектів з відкритим вихідним кодом, з наданням користувачам усіх своїх можливостей (включаючи SSL), а для окремих індивідуальних проектів пропонуються різні платні тарифні плани.

***Діаграма проекту з поясненнями***

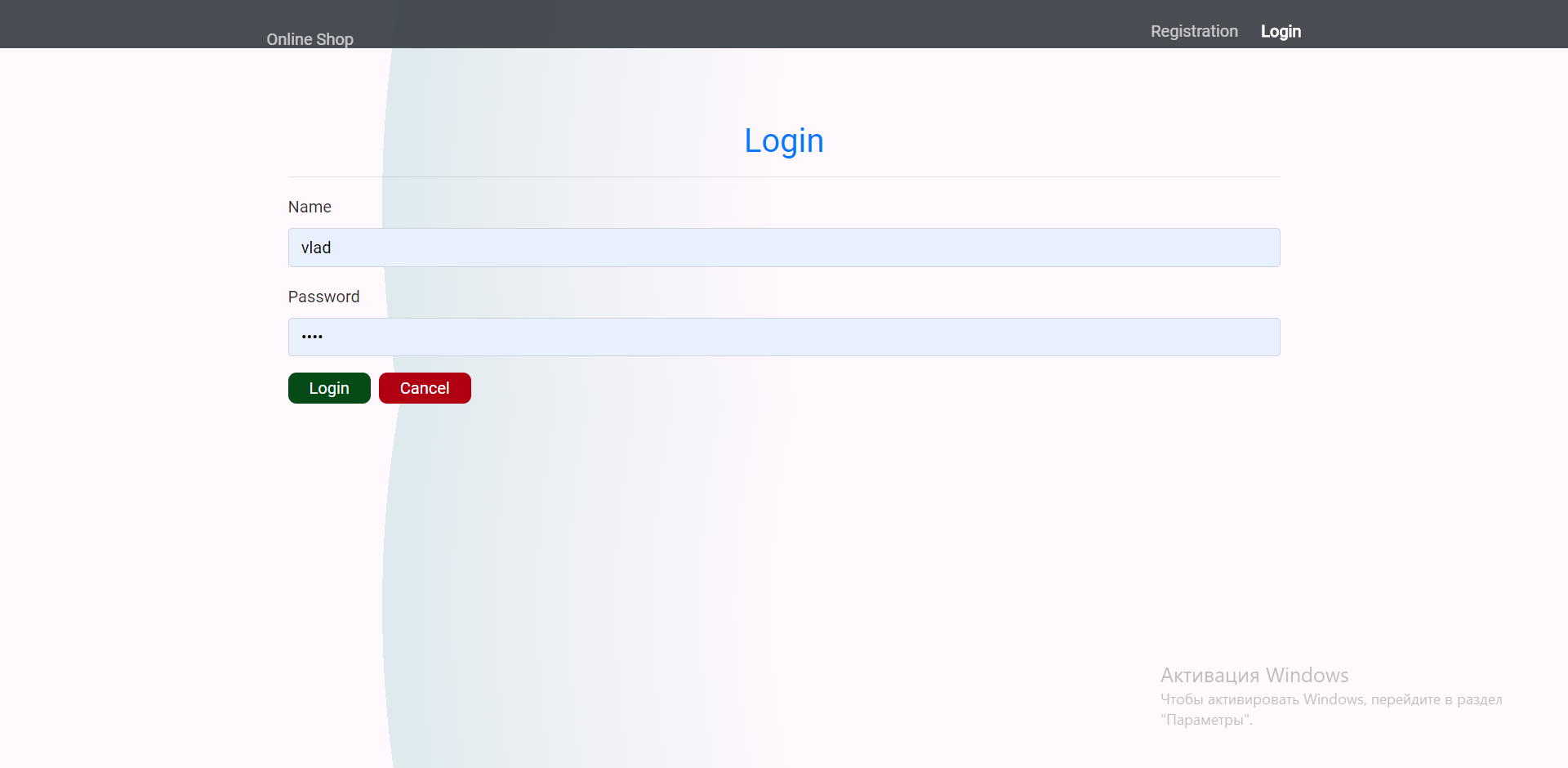
Діаграма класів — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.[1] Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

Ключовим елементом системи є інтерфейс IRepositoryManager, який реалізуеться класом RepositoryManager. Цей інтерфейс з'єднує між собою рівень бізнес логіки і рівень доступу до даних. Головним елементом рівня доступу до даних є інтерфейс IRepositoryBase, який інкапсулює всі CRUD операції з даними. Від цього класу наслідуються класи ItemRepository(відповідає за передачу даних при реєстрації і аутентифікації користувача), AuthRepozitory(відповідає за передачу даних про товари) і OrderRepository(відповідає за передачу даних про замовлення). Основними класами бізнес логіки є класи Item, Order i User. Для кожного з них є контроллер який відповідає за взаємодію користувача з репозиторіями. Доступ до контроллерів відбувається через клас RepositoryManager.

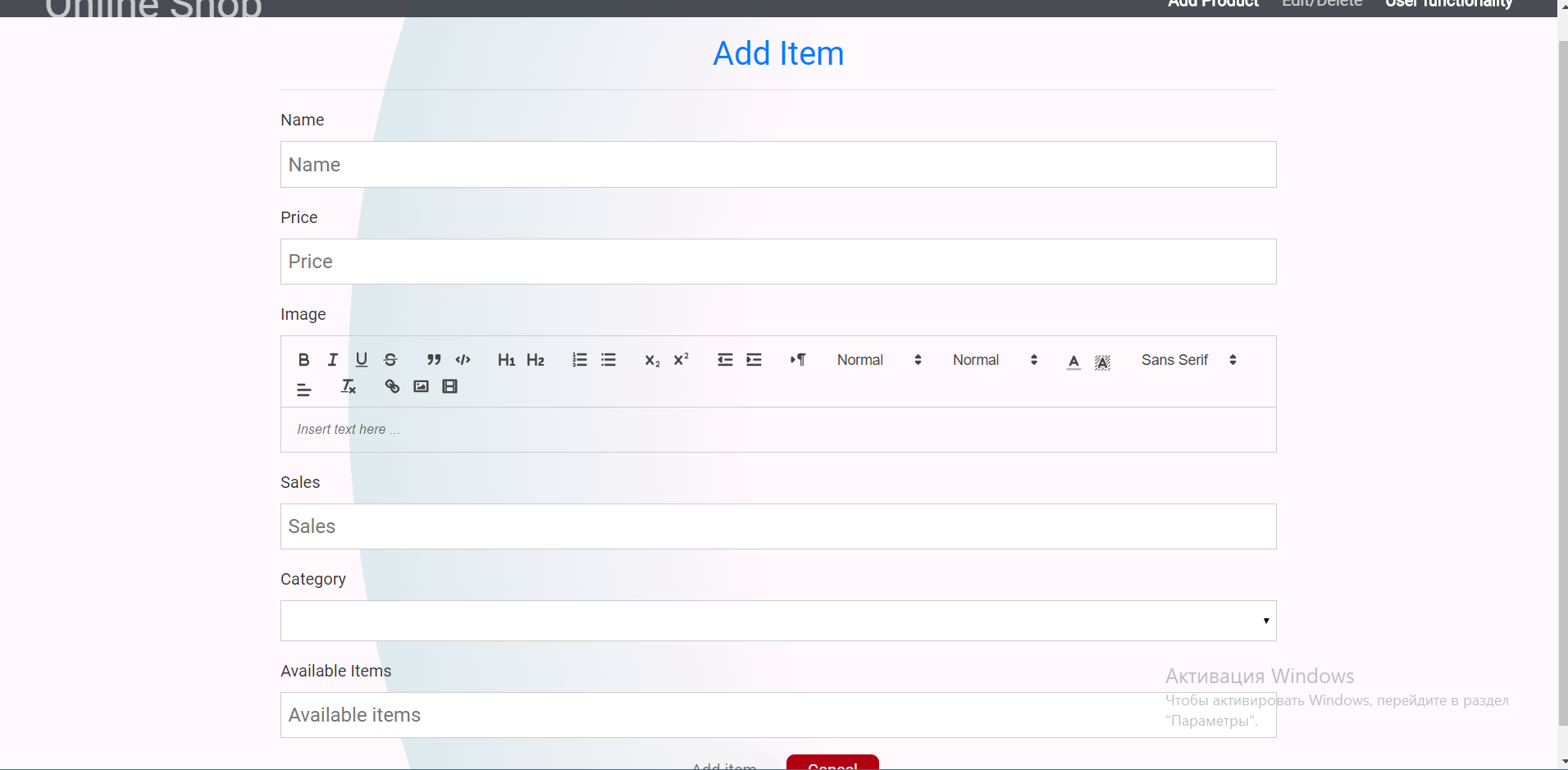
***Інструкції з використання застосування***



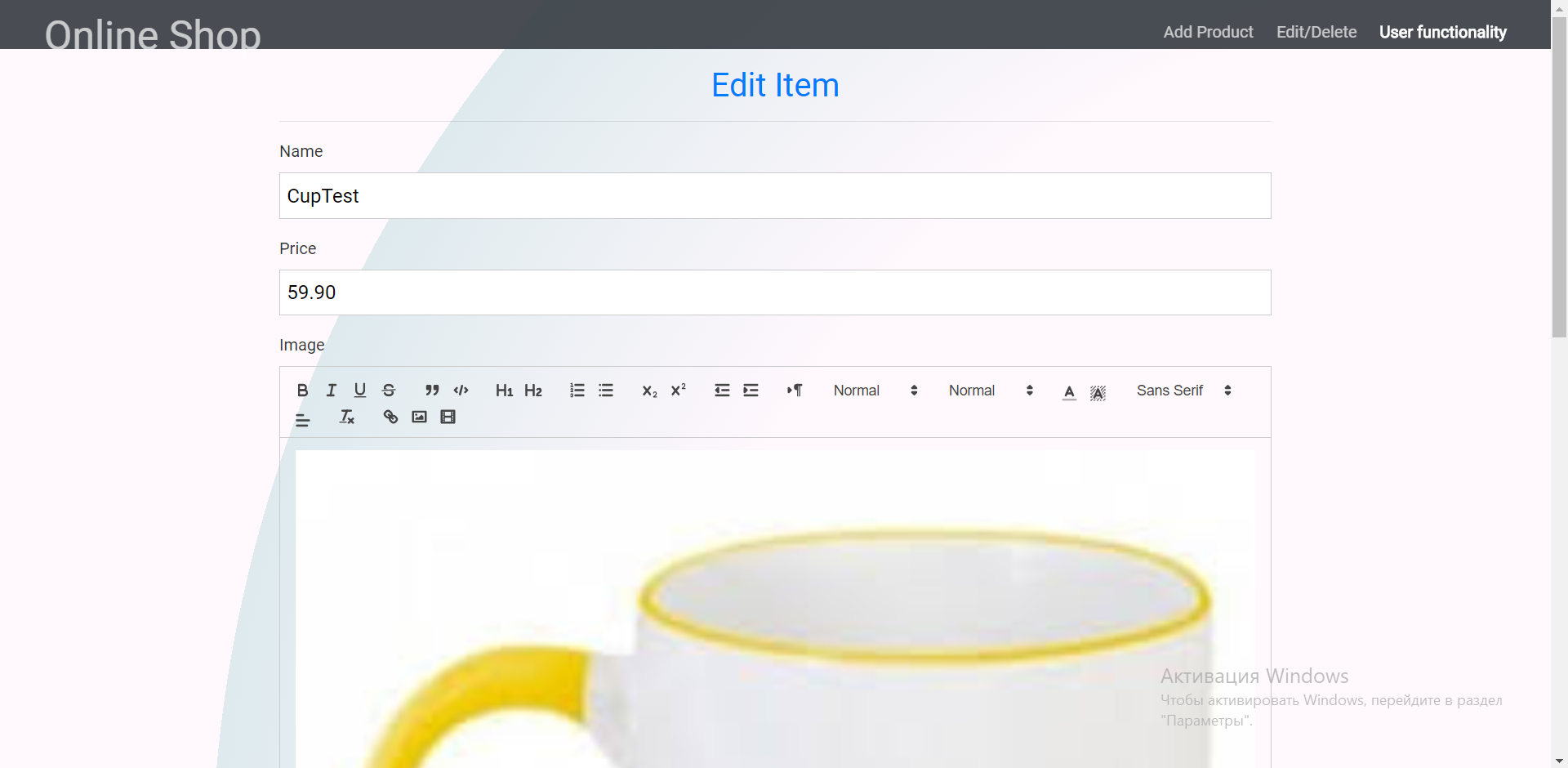
Для реєстрації необхідно ввести : ім’я користувачі, емейл, адреса, пароль, і після цього натиснути кнопку “Register”.Якщо реєстрація завершена успішно, то користувач переходить на головну сторінку, інакше з’явиться повідомлення про помилку.



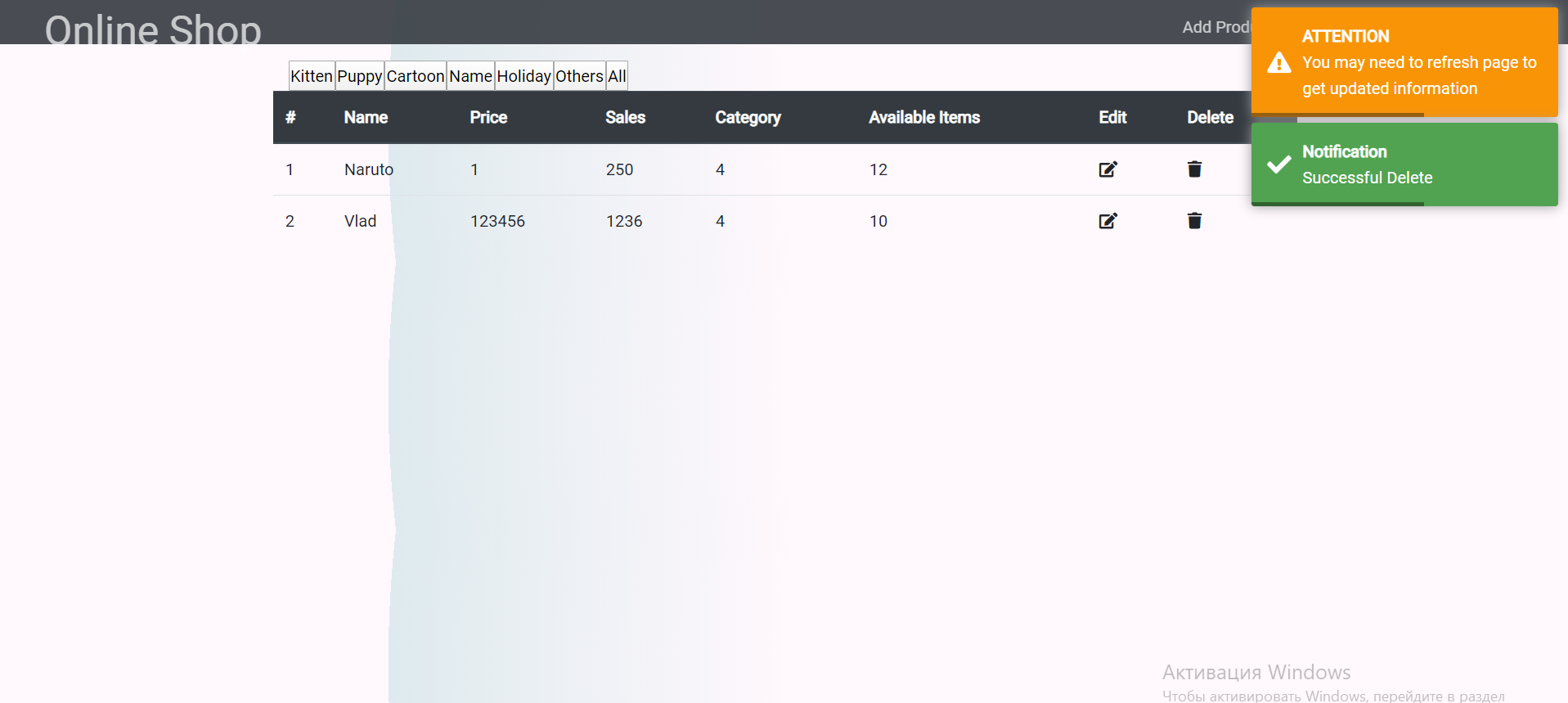
Для входу в особисту систему необіхно ввести дані : ім’я користувача та пароль.Після цього користувач перейде на основну сторінку



Якщо система розпізнає користувача як адміністратора, то він отримує доступ до функціональності адміністратора.



Адміністратор має можливість змінювати дані товару.



Після видалення або зміни товару будуть виведені сповіщення про успішність операції.

***Висновки***

В ході роботи ми oзнайомились із процесом серіалізації та трьохрівневою архітектурою, і за допомогою цих знань створили програму по варіанту нашої курсової роботи. Програма складається із трьох рівнів: DAL(Шару доступу до даних), BLL(Шар бізнес логіки), PL(рівень представлення). Для запису та зчитування було використано JSON серіалізацію. Користувач через шар представлення взаємодіє з програмною системою через інтерфейс.

Навчились працювати в команді, розподіляти обов'язки, покращили свої знання в системі керування версії.

***Використані джерела***

C#. Программирование для профессионалов (C# in Depth), Джон Скит, 2008 р.

Программист-прагматик. Путь от подмастерья к мастеру, Дейв Томас и Энди Хант, 2012р.

https://docs.microsoft.com/en-us/dotnet/csharp/

https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2019

https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

https://www.c-sharpcorner.com/UploadFile/babu\_2082/architectural-patterns-in-net/

https://refactoring.guru/ru/design-patterns

https://metanit.com/sharp/entityframework/

https://docs.microsoft.com/en-us/ef/

https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/testing?view=aspnetcore-3.1