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

**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. 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.

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.

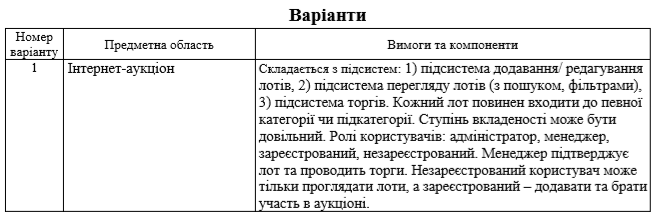
13. Перед початком розробки створити ПРИВАТНИЙ проект і відкрити для викладача доступ до нього у Visual Studio Online/ Azure DevOps Services (ніби він є ще одним членом команди).

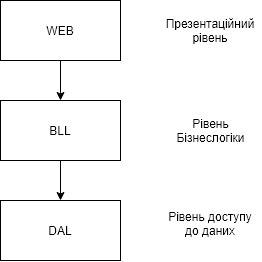
14. Назва проекту повинна бути у форматі «ГРУПА-ПІДГРУПА-ВАРІАНТ». Наприклад, PI-215-14. В підгрупі варіанти повторюватися не повинні.

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.

**Завдання варіанту № 1**



**Загальна архітектура проекту:  
 **

**Презентаційний рівень** – представлення (UI), призначений для взаємодії з користувачем. Реалізовано окремим проектом. Для створення Web-інтерфейсу була використана технологія ASP.NET Web API 2.0. На рівні UI - тільки операції взаємодії з користувачем.

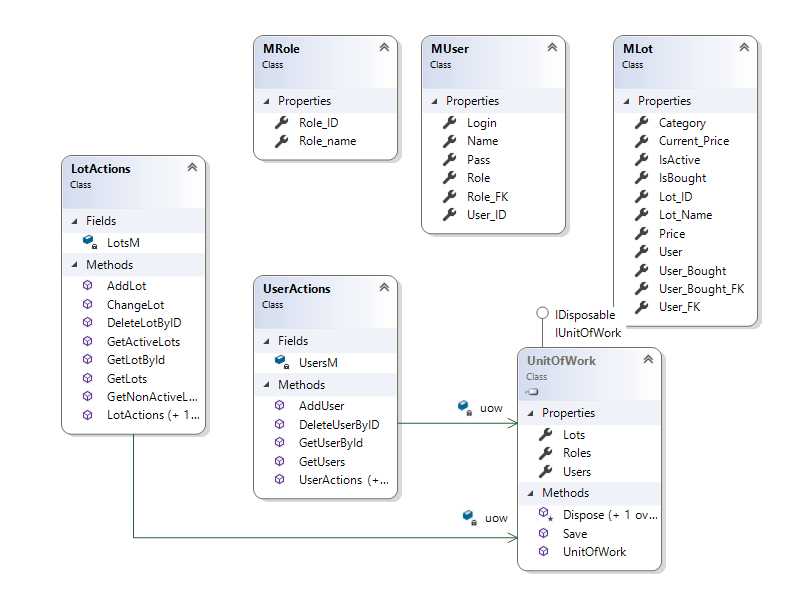
**Рівень бізнес логіки** – реалізований як динамічна бібліотека. На цьому рівні реалізований саме функціонал застосування інтернет-магазину.

**Рівень доступу до данних** – 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 було використано ORM ADO .Net Entity Framework (code first). Також застосовано шаблони проектування Репозиторій та одиниця доступу(Unit of Work).

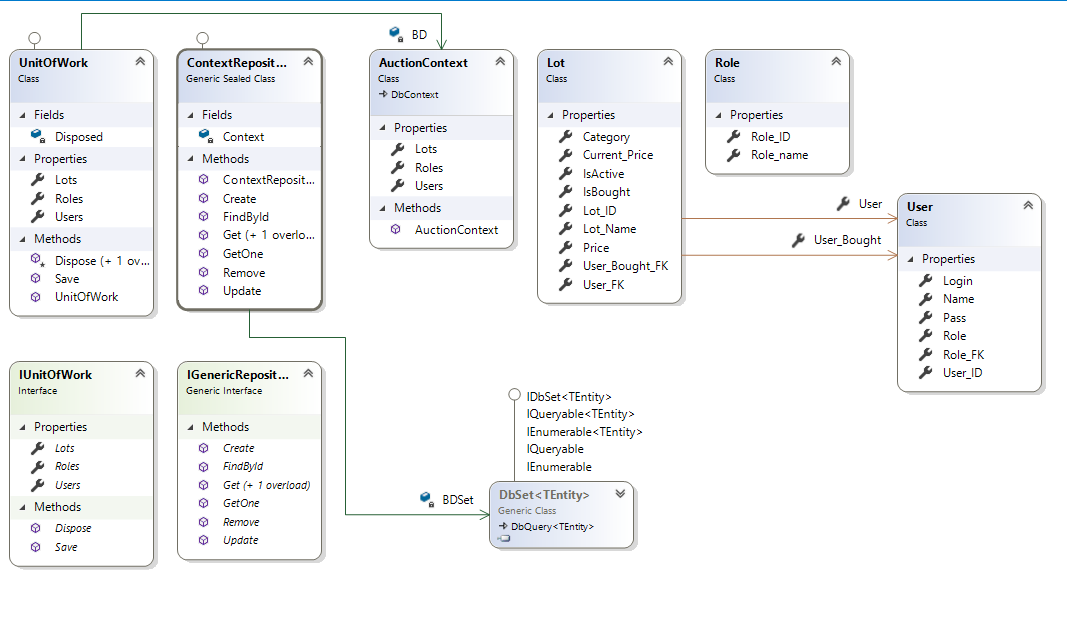
Шари взаємодіють між собою за наступним принципом: представлення використовує бізнес-логіку, бізнес-логіка – рівень доступу до даних. Для передачі даних крізь шари використовується технологія відображення (mapping).

Це забезпечує надійність, немає сильної зв’язності, програма структурована, рівні незалежні, ізольовані, можлива зміна одного рівня без впливу на інший. Принцип ізоляції рівнів полягає в тому, що рівень доступу до даних нічого не знає про рівень бізнес логіки, а бізнес логіка немає ніяких даних про рівень представлення. Завдяки цьому програма є гнучкою, готовою до легким змін. Використано принцип інверсій залежностей (Dependency Injection). Це один з SOLID-принципів об'єктно-орієнтованого проектування програм, суть якого полягає у розриві зв'язності між програмними модулями вищого та нижчого рівнів за допомогою спільних абстракцій. Принцип формується наступним чином: модулі вищого рівня не повинні залежати від модулів нижчого рівня (обидва типи модулів повинні залежати від абстракцій); абстракції не повинні залежати від деталей реалізації, деталі реалізації повинні залежати від абстракцій.

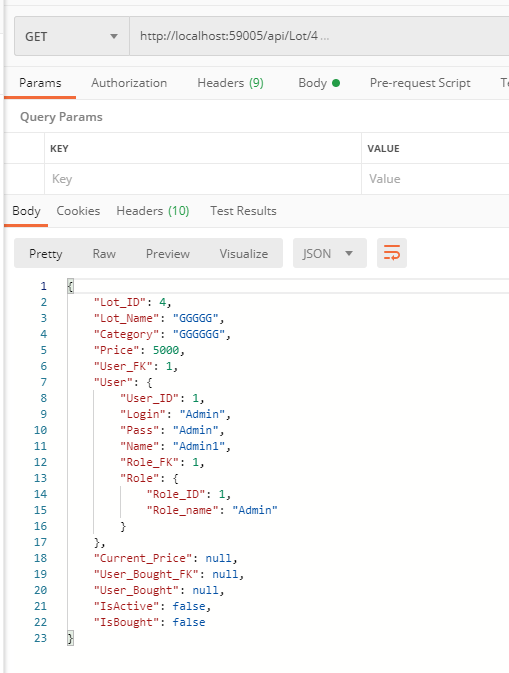
**Діаграма бізнес логіки**



**Діаграма рівню доступу до даних**



**Результат роботи програми**



**Висновки:** В ході виконання даної курсової роботи було розроблено веб-застосування з трьохшаровою системною архітектурою , було використано ORM ADO .Net Entity Framework (code first). Для розробки використовувались ASP.NET Web Api 2.0 та ORM ADO.Net Entity Framework (code first), були також застосовані шаблони проектування Репозиторій, Одиниця роботи. Репозиторії надають доступ до набору сутностей (entities) певного типу.