Лабораторна робота №3

Робота із базами даних в .NET. Використання бібліотеки Entity Framework.

Виконав студент 301-пТК Тараненко Олександр

Передумова

Виконайте лабораторну роботу №2. Створіть нову гілку у раніше створеному гіт репозиторії *NUPP_NET_2025_{Номер Групи}_ТК_{Прізвище}_Lab* із назвою **lab3**, яка міститиме код із другої лабораторної роботи.
Для здачі лабораторної роботи, необхідно буде створити пул реквест із гілки **lab3** у **master**, або якщо на момент здачі пул реквест із гілки lab2 у master не закритий - із гілки lab3 у lab2.

Завдання

1. Створити проєкт {Назва тематики}.Infrastructure. У новоствореному проєкті створити папку Models. У створеній папці додати модель(набір класів), яка відповідає створеній у першій лабораторній роботі моделі, та містить всередині себе властивості. Наприклад, якщо існував у першій лабораторній клас Bus, у папці Models необхідно створити клас BusModel.
   Отримана модель має обов'язково мати зв'язки один-до-одного та один-до-багатьох. За додавання зв'язків багато-до-багатьох можна отримати додаткові бали. Для наслідування більш пріоритетно застосовувати підхід Таблиця на тип (Table-per-Type).

2. Створити клас *{Назва тематики}Context*, який наслідуватиметься від класу DbContext та опише схему бази даних для обраної тематики застосовуючи Entity Framework та анотації або Fluent API на вибір, Fluent API більш пріоритетно.

3. Створити міграцію використовуючи Entity Framework. Створену міграцію застосувати до реляційної бази даних. Можна використовувати будь-яку реляційну СУБД, як MS SQL, PostgreSQL, MySQL, тощо. Також слід використати SQLite, коли немає можливості розробляти та здавати Лабораторну роботу на одній системі. При використанні SQLite файл бази даних необхідно додати до гіт репозиторія.

4. Реалізувати шаблон проєктування Репозиторій, *{Назва тематики}Context* який буде використовувати клас *{Назва тематики}Context* для досьупу до даних із БД:

   ```
   public interface IRepository<T> where T : class
   ```

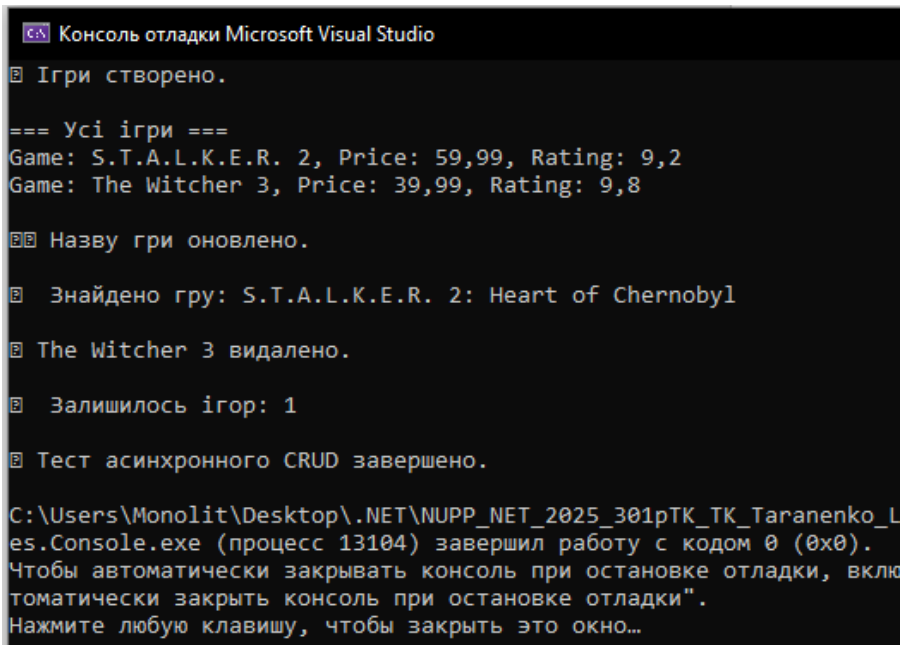
```
{
        Task<T> GetByIdAsync(int id);
        Task<IEnumerable<T>> GetAllAsync();
        Task AddAsync(T entity);
        Task Update(T entity);
        Task Delete(T entity);
 }
```

5. Оновити асинхрону версію дженерік CRUD сервісу, щоб він використовував репозиторій для доступу до даних:

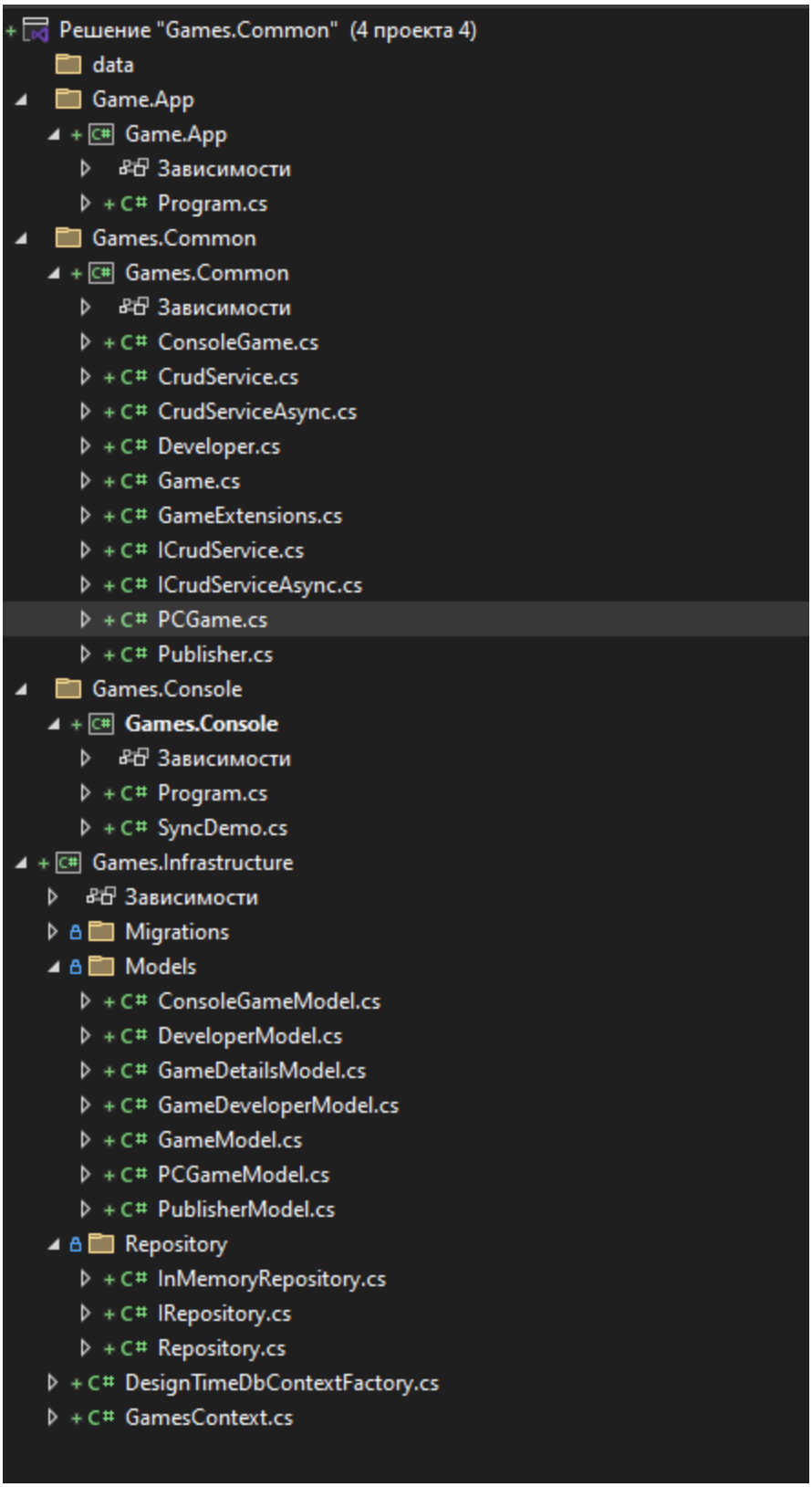
```
public interface ICrudServiceAsync<T>
{
        public Task<bool> CreateAsync(T element);
        public Task<T> ReadAsync(Guid id);
        public Task<IEnumerable<T>> ReadAllAsync();
        public Task<IEnumerable<T>> ReadAllAsync(int page, int amount);
        public Task<bool> UpdateAsync(T element);
        public Task<bool> RemoveAsync(T element);
        public Task<bool> SaveAsync();
}
```

6. Модифікуйте консольний застосунок, щоб він використовував оновлену версію CRUD сервісу та дані із бази даних.

   Результати виконання консольного застосунку.

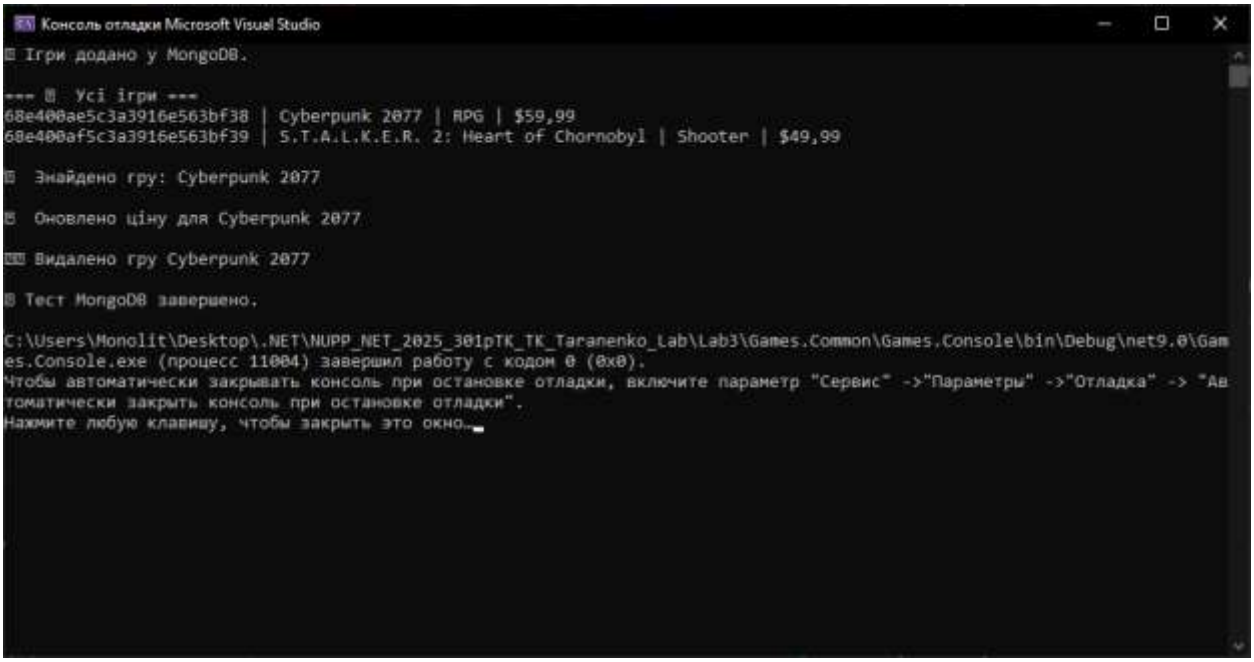


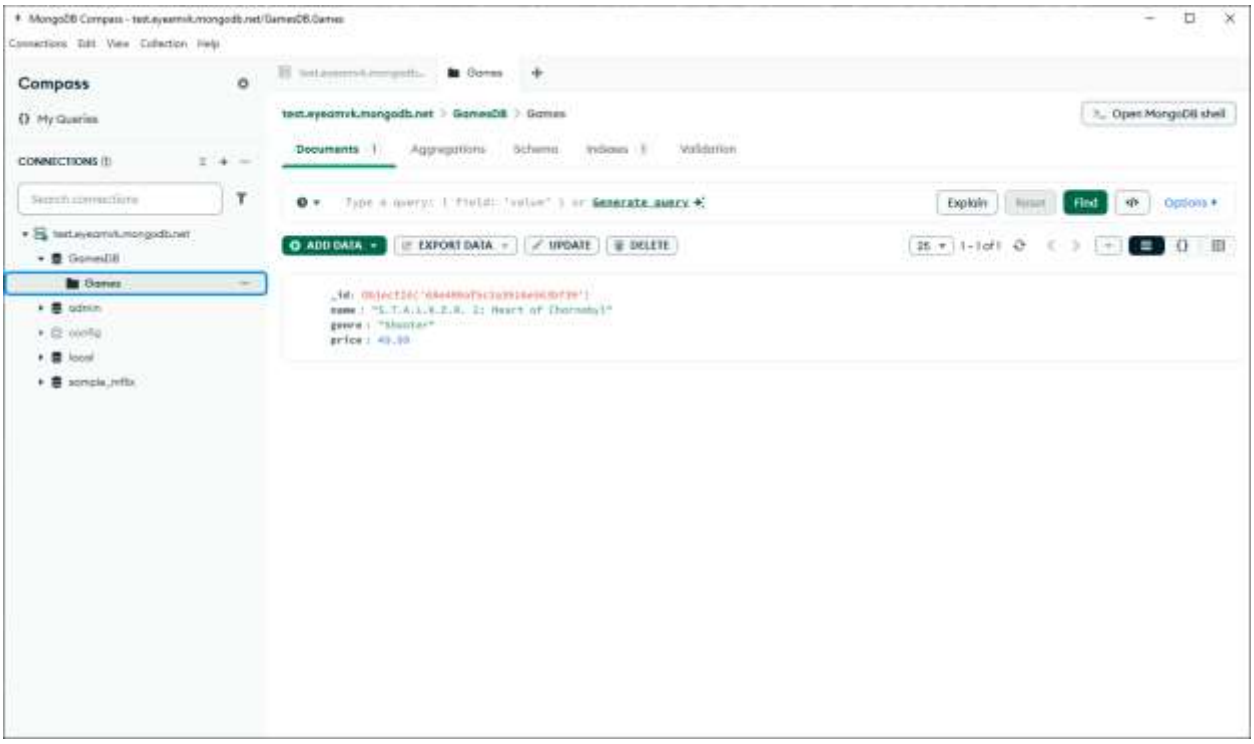
   Готова лабораторна робота має наступну файлову ієрархію:


Решение "Games.Common" (4 проекта 4)
data
Game.App
Game.App
Зависимости
Program.cs
Games.Common
Games.Common
Зависимости
ConsoleGame.cs
CrudService.cs
CrudServiceAsync.cs
Developer.cs
Game.cs
GameExtensions.cs
ICrudService.cs
ICrudServiceAsync.cs
PCGame.cs
Publisher.cs
Games.Console
Games.Console
Зависимости
Program.cs
SyncDemo.cs
Games.Infrastructure
Зависимости
Migrations
Models
ConsoleGameModel.cs
DeveloperModel.cs
GameDetailsModel.cs
GameDeveloperModel.cs
GameModel.cs
PCGameModel.cs
PublisherModel.cs
Repository
InMemoryRepository.cs
IRepository.cs
Repository.cs
DesignTimeDbContextFactory.cs
GamesContext.cs

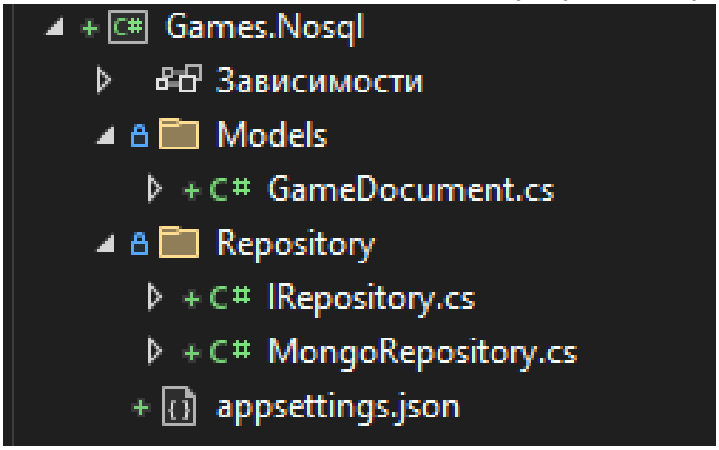
Додаткове завдання

Створіть проєкт {Назва тематики}.Nosql. У цьому проєкті реалізуйте репозиторій із 4 завдання, який буде використовувати нереляційну базу даних, як наприклад MongoDB. У звіт додайте скріншоти об’єктів із нереляційної бази даних. Використовувати Entity Framework для нереляційної БД необов’язково.





Готова лабораторна робота має наступну файлову ієрархію:



Контрольні запитання

1. Що таке реляційні бази даних? Що таке СУБД? Які СУБД ви знаєте?

Реляційні бази даних — це бази даних, у яких інформація зберігається у вигляді таблиць, що пов'язані між собою через ключі. **СУБД (система управління базами даних)** — це програмне забезпечення, яке дозволяє створювати, змінювати, зберігати та отримувати дані. **Приклади СУБД:** MySQL, PostgreSQL, Microsoft SQL Server, Oracle, SQLite.

2. Що позначає термін таблиця у реляційній БД? Які існують зв'язки у реляційній БД?

Таблиця — це структура, яка зберігає дані у вигляді рядків (записів) і стовпців (полів).

Основні типи зв'язків:

- Один-до-одного (1:1)
- Один-до-багатьох (1:N)
- Багато-до-багатьох (M:N)

3. Що таке ERD-діаграма? Як її створити та для чого вона потрібна?

ERD (Entity Relationship Diagram) — це схема, що показує сутності бази даних і зв'язки між ними.
Створюється за допомогою інструментів (наприклад, Draw.io, Lucidchart, DbDiagram.io).
Призначення: візуально показати структуру бази даних перед реалізацією.

4. Що таке DbContext і яку роль він відіграє в роботі з базою даних?

DbContext — це головний клас у Entity Framework Core, який керує з'єднанням з базою даних і дозволяє виконувати CRUD-операції через набори сутностей (`DbSet<T>`).

5. Що таке зв'язки один-до-одного, один-до-багатьох і багато-до-багатьох у контексті Entity Framework Core?

- **Один-до-одного:** одна сутність відповідає одній іншій.
- **Один-до-багатьох:** одна сутність має кілька пов'язаних.
- **Багато-до-багатьох:** обидві сутності можуть мати багато зв'язків одна з одною.

6. Як реалізувати зв'язок один-до-одного за допомогою Fluent API?

```
modelBuilder.Entity<User>()
   .HasOne(u => u.Profile)
   .WithOne(p => p.User)
   .HasForeignKey<Profile>(p => p.UserId);
```

7. Наведіть приклад реалізації зв'язку багато-до-багатьох у EF Core.

```
modelBuilder.Entity<GameDeveloper>()
    .HasKey(gd => new { gd.GameId, gd.DeveloperId });

modelBuilder.Entity<GameDeveloper>()
    .HasOne(gd => gd.Game)
    .WithMany(g => g.GameDevelopers)
    .HasForeignKey(gd => gd.GameId);
```

```
modelBuilder.Entity<GameDeveloper>()
    .HasOne(gd => gd.Developer)
    .WithMany(d => d.GameDevelopers)
    .HasForeignKey(gd => gd.DeveloperId);
```

8. У чому різниця між анотаціями (Data Annotations) та Fluent API в конфігурації моделей?

- **Data Annotations** — це атрибути, які додаються безпосередньо в код моделі (`[Key]`, `[Required]` тощо).
- **Fluent API** — це конфігурація через метод `OnModelCreating()` у `DbContext`, дає більше гнучкості та контроль.

9. Як створюється та застосовується міграція в EF Core?

1. Створення міграції:

```
dotnet ef migrations add InitialCreate
```

2. Застосування міграції:

```
dotnet ef database update
```

Міграції відстежують зміни у моделях і синхронізують їх із базою даних.

10. Яка мета проєкту {Назва тематики}.Infrastructure? Яку роль він відіграє у загальній структурі застосунку?

Проєкт **Infrastructure** відповідає за **доступ до даних** і роботу з базою (реалізація репозиторію, контекст БД, моделі). Він відокремлює логіку роботи з БД від бізнес-логіки (розділення відповідальності).

11. У чому різниця між доменною моделлю (з попередньої лабораторної) та моделлю бази даних?

- **Доменна модель** — відображає бізнес-логіку (об’єкти реального світу: Game, Developer).
- **Модель бази даних** — технічне відображення сутностей у таблицях БД (з ключами, зв’язками, атрибутами).

12. Яку проблему вирішує шаблон проектування Репозиторій?

Репозиторій ізолює бізнес-логіку від деталей доступу до даних. Дозволяє змінювати джерело даних (БД, файл, пам’ять, NoSQL) без зміни бізнес-коду.

13. У чому принципова різниця між реляційними та нереляційними базами даних?

- **Реляційні (SQL):** дані в таблицях, сувора схема, зв’язки через ключі.
- **Нереляційні (NoSQL):** дані у вигляді документів, колекцій, графів або пар «ключ-значення»; гнучка структура без фіксованої схеми.

Приклади NoSQL: MongoDB, Firebase, Cassandra, Redis.