Mariusz Wieczorek

mariusz.wieczorek@kabat.pl

Streszczenie

Tworzymy aplikacje webowej z wykorzystaniem szablonu .Net Core MVC  
Omówienie powstałej struktury plików  
Co to jest Areas  
Klasa Startup.cs i metody Configure i ConfigureServices,  
Klasa Program.cs – punkt startowy aplikacji

asp.net Core

Dependency Injection

**Dependency Injection**

Jest to wzorzec polegający na usuwaniu bezpośrednich zależności w kodzie.  
Dzięki niemu możemy uzyskać tzw. luźnie powiązania [ loose coupling ] do czego powinniśmy dążyć pisząc aplikację.

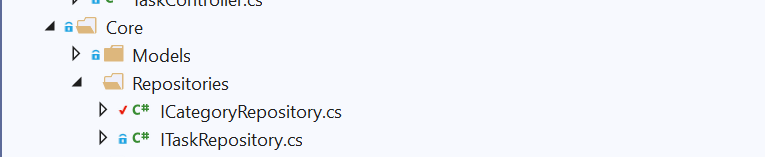
Chcemy aby nasze aplikacje były bardziej elastyczne, a ewentualne zmiany w przyszłości były jak najłatwiejsze do wprowadzania, oraz co bardzo ważne dążymy do tego aby nasza aplikacja była testowalna, czyli aby łatwo było napisać do niej testy jednostkowe.

Aby to zrobić musimy opierać nasze rozwiązania nie na konkretnych implementacjach ale na interfejsach, co sprawia, że mamy w kodzie małe zależności.

Interfejsy są stabilne, co oznacza, że jeżeli zrobimy zmianę w interfejsie, to ta zmiana jest również powiązana z implementacją tego interfejsu. Jeżeli natomiast zrobimy zmianę w konkretnej implementacji, to zazwyczaj nie musimy zmieniać tego interfejsu.

Pierwszym krokiem do wprowadzenia DI jest stworzenia abstrakcji do klas np. w postaci interfejsów, a następnie operowanie w aplikacji nie na konkretnych implementacjach ale na interfejsach.  
Konkretne implementacje zostaną właśnie dzięki DI wstrzyknięte do naszej aplikacji.  
Będziemy musieli później w klasie Startup.cs zdefiniować takie powiązania.

Interfejsy będziemy tworzyć w folderze Core. Tworzymy folder Repositories



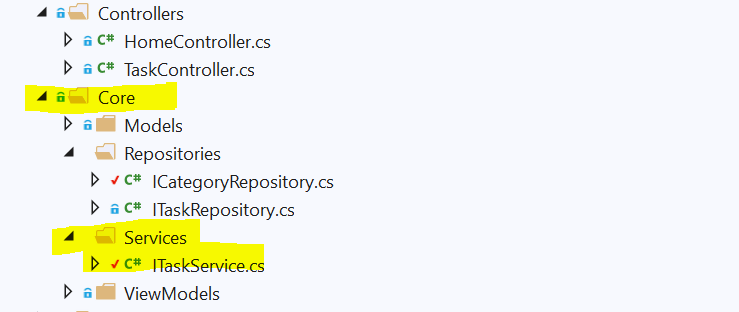
|  |
| --- |
| namespace MyTasks.Core.Repositories  {  interface ICategoryRepository  {  public IEnumerable<Category> GetCategories();  }  } |

|  |
| --- |
| namespace MyTasks.Core.Repositories  {  public interface ITaskRepository  {  IEnumerable<Task> Get(string userId,  bool isExecuted = false,  int categoryId = 0,  string title = null);  Task Get(int id, string userId);  void Add(Task task);  void Update(Task task);  void Finish(int id, string userId);  void Delete(int id, string userId);  }  } |

W klasach TaskRepository i CategoryRepository dopisujemy, że klasy te implementują interfejsy.

|  |
| --- |
| namespace MyTasks.Persistence.Repositories  {  public class CategoryRepository : ICategoryRepository  {  private readonly ApplicationDbContext \_context; |

|  |
| --- |
| namespace MyTasks.Persistence.Repositories  {  public class TaskRepository : ITaskRepository  {  private readonly ApplicationDbContext \_context; |



|  |
| --- |
| namespace MyTasks.Core  {  public interface IApplicationDbContext  {  DbSet<Task> Tasks { get; set; }  DbSet<Category> Categories { get; set; }  int SaveChanges();  }  } |

|  |
| --- |
| namespace MyTasks.Core  {  public interface IUnitOfWork  {  ITaskRepository Task { get; set; }  ICategoryRepository Category { get; set; }  void Complete();  }  } |

|  |
| --- |
| namespace MyTasks.Core.Services  {  public interface ITaskService  {  IEnumerable<Task> Get(string userId,  bool isExecuted = false,  int categoryId = 0,  string title = null);  Task Get(int id, string userId);  void Add(Task task);  void Update(Task task);  void Finish(int id, string userId);  void Delete(int id, string userId);  }  } |

|  |
| --- |
| namespace MyTasks.Core.Services  {  public interface ICategoryService  {  IEnumerable<Category> GetCategories();  }  } |