

Mariusz Wieczorek

mariusz.wieczorek@kabat.pl

Repozytorium, UnitOfWork, Serwisy

Jak w praktyce użyć tych wzorców

asp.net Core

Warstwy dostępu do danych

**Repozytorium**

Pracujemy na wstrzykniętym przez DI kontekście.  
  
Jest to klasa zależna od frameworka, który zapewnia nam fizyczny dostęp do danych.  
Traktujemy źródło danych jak kolekcje, aby móc w celach testowych podstawić zamiast repozytorium łączącego się z konkretną bazą lub web serwisem, udawanego repozytorium na kolekcji.  
Dokonujemy więc takich operacji jak pobieranie danych, edycja, dodawanie, usuwanie ale bez zapisu zmian.

|  |
| --- |
| public interface ICategoryRepository  {  IEnumerable<Category> GetCategories(string userId);  void AddCategory(Category category);  Category GetCategory(int id, string userId);  void UpdateCategory(Category category);  void DeleteCategory(int id, string userId);  } |

|  |
| --- |
| public class CategoryRepository : ICategoryRepository  {  private readonly IApplicationDbContext \_context;  public CategoryRepository(IApplicationDbContext context)  {  \_context = context;  }  public void AddCategory(Category category)  {  \_context.Categories.Add(category);  }  public Category GetCategory(int id, string userId)  {  var category = \_context.Categories  .Single(x => x.Id == id && ( x.UserId == userId || String.IsNullOrEmpty(x.UserId) ));  return category;  } |

**UnitOfWork**

Jest to klasa, którą opakowujemy wszystkie repozytoria w aplikacji.  
Do konstruktora wstrzykujemy kontekst oraz repozytoria.  
Dodajemy metodę zapisującą dane, ponieważ w klasach repozytoryjnych takich metod już nie ma.  
W przykładzie poniżej metodą zapisującą dane jest **Complete()**, w której wywołujemy zapisanie zmian w kontekście na którym pracujemy **context.SaveChanges()**  
Wyodrębnienie metody zapisującej dane pozwala nam na uzyskanie transakcyjności.  
Metoda zapisu jest uruchomiona na końcu, więc albo zapamiętane są wszystkie modyfikacje danych albo żadne.

|  |
| --- |
| public interface IUnitOfWork  {  ITaskRepository Task { get; }  ICategoryRepository Category { get; }  void Complete();  } |

|  |
| --- |
| public class UnitOfWork : IUnitOfWork  {  // readonly przy polu oznacza, że jego wartość  // możemy zmienić tylko w konstruktorze  private readonly IApplicationDbContext \_context;  public UnitOfWork(IApplicationDbContext context,  ICategoryRepository category,  ITaskRepository task)  {  \_context = context;  Task = task; // new TaskRepository(context);  Category = category; // new CategoryRepository(context);  }  // obiekty repozytoryjne  public ITaskRepository Task { get; }  public ICategoryRepository Category { get; }  // na koniec metoda zapisująca zmiany  public void Complete()  {  \_context.SaveChanges();  }  } |

**Serwisy**

Klasa, którą opakowujemy klasę **UnitOfWork**, oraz jest to miejsce w którym możemy zaimplementować jakąś dodatkową logikę biznesową.

|  |
| --- |
| Public interface IcategoryService  {  IEnumerable<Category> GetCategories(string userId);  void AddCategory(Category category);  Category GetCategory(int id, string userId);  void UpdateCategory(Category category);  void DeleteCategory(int id, string userId);  } |

|  |
| --- |
| public class CategoryService : ICategoryService  {  private readonly IUnitOfWork \_unitOfWork;  public CategoryService(IUnitOfWork unitOfWork)  {  \_unitOfWork = unitOfWork;  }  public void AddCategory(Category category)  {  \_unitOfWork.Category.AddCategory(category);  \_unitOfWork.Complete();  }  public void DeleteCategory(int id, string userId)  {  \_unitOfWork.Category.DeleteCategory(id, userId);  \_unitOfWork.Complete();  }  public IEnumerable<Category> GetCategories(string userId)  {  return \_unitOfWork.Category.GetCategories(userId);  }  public void UpdateCategory(Category category)  {  \_unitOfWork.Category.UpdateCategory(category);  \_unitOfWork.Complete();  }  public Category GetCategory(int id,string userId)  {  return \_unitOfWork.Category.GetCategory(id, userId);  }  } |

**Kontroler**

W tym modelu serwisy, to klasy, których instancji używamy w akcjach kontrolera.

|  |
| --- |
| [Authorize]  public class TaskController : Controller  {  private readonly ITaskService \_taskService;  private readonly ICategoryService \_categoryService;  public TaskController(ITaskService task, ICategoryService category)  {  \_taskService = task;  \_categoryService = category;  }  #region Tasks ----------------------------------------------------------------  public IActionResult Tasks()  {  var userId = User.GetUserId();  var vm = new TasksViewModel()  {  FilterTasks = new FilterTasks(),  Tasks = \_taskService.Get(userId, new FilterTasks()),  Categories = \_categoryService.GetCategories(userId)  };  return View(vm);  } |