

Mariusz Wieczorek

mariusz.wieczorek@kabat.pl

Streszczenie

Czym jest .Net Core

asp.net Core

Wstęp

**Rozszerzenie aplikacji o zarządzanie kategoriami.**

Dodajemy zarządzanie kategoriami, dodawanie  
Zalogowany użytkownik zarządza swoimi kategoriami.  
Jest też przynajmniej jedna wspólna (może mieć puste pole userId)

Po pierwsze dodajemy do modelu pole userId

Zmiany w modelu domenowym

|  |
| --- |
| Namespace MyTasks.Core.Models.Domains  {  public class Category  {  public Category()  {  Tasks = new Collection<Task>();  }  public int Id { get; set; }  [Required]  public string Name { get; set; }  public string UserId { get; set; }  public ICollection<Task> Tasks;  public ApplicationUser User { get; set; }  }  } |

|  |
| --- |
| namespace MyTasks.Core.Models.Domains  {  public class ApplicationUser : IdentityUser  {  public ApplicationUser()  {  Tasks = new Collection<Task>();  Users = new Collection<ApplicationUser>();  }    public ICollection<Task> Tasks;  public ICollection<ApplicationUser> Users;  }  } |

|  |
| --- |
| PM> add-migration add\_user\_to\_category  Build started...  Build succeeded.  To undo this action, use Remove-Migration.  PM> update-database  Build started...  Build succeeded.  Done. |

Musimy dodać metody pozwalające na zarządzanie kategoriami  
Musimy dokonać zmian w Interfejsach i klasach

Zmiany w interfejsie ICategoryRepository

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

Zmiany w klasie CategoryRepository

|  |
| --- |
| namespace MyTasks.Persistence.Repositories  {  public class CategoryRepository : ICategoryRepository  {  private readonly IApplicationDbContext \_context;  public CategoryRepository(IApplicationDbContext context)  {  \_context = context;  }  public void AddCategory(Category category)  {  \_context.Categories.Add(category);  }  public void DeleteCategory(int id, string userId)  {  var categoryToDelete = \_context.Categories.Single(x => x.Id == id && x.UserId == userId);  \_context.Categories.Remove(categoryToDelete);  }  public IEnumerable<Category> GetCategories()  {  return \_context.Categories.OrderBy(x => x.Name).ToList();  }  public void UpdateCategory(Category category)  {  var categoryToUpdate = \_context.Categories.Single(x => x.Id == category.Id);  categoryToUpdate.Name = category.Name;  }  }  } |

Zmiany w Interfejsie ICategoryService

|  |
| --- |
| namespace MyTasks.Core.Services  {  public interface ICategoryService  {  IEnumerable<Category> GetCategories();  void AddCategory(Category category);  void UpdateCategory(Category category);  void DeleteCategory(int id, string userId);  }  } |

Zmiany w klasie CategoryService

|  |
| --- |
| namespace MyTasks.Persistence.Services  {  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()  {  return \_unitOfWork.Category.GetCategories();  }  public void UpdateCategory(Category category)  {  \_unitOfWork.Category.UpdateCategory(category);  \_unitOfWork.Complete();  }  }  } |