İçindekiler

.Net Core Eğitimi Temel Kavramlar

Extension

Extension methodlara genişletilebilir method da denilebilir. Kısaca özetlemek gerekirse bir tip üzerinde herhangi bir değişikliğe uğratmadan genişletilebilir olmasını sağlayabiliriz. Önceden yazılmış class'lara farklı bir özellik eklemek istediğimizde de extensionlar'dan yararlanıyoruz. Bu işlem compile (derleme) anında gerçekleşmektedir. Ayrıca doğru özelliği doğru nesnelere aktarabilmek ve kod okunabilirliğini arttırması extension kullanımının avantajları arasında yer almaktadır.

Bir extension oluşturmak için aşağıdaki koşulların sağlanması gerekmektedir.

- Public static bir class oluşturulmalıdır.
- Oluşturmak istediğimiz method da static olarak tanımlanmalıdır.
- Methodu oluştururken hangi tip üzerinde bu metodun çalışmasını sağlayacağımızı this keyword'ü ile belirlemeliyiz. İlk parametre olarak this keyword'ü belirlenmelidir.

Basitçe bir mapping extension kullanımına ait kod parçası Şekil 1 ve Şekil 2'deki gibidir.

```
0 references
public static class MappingExtension
{

    1 reference
    public static List<BookModel> ToBookResponse(this List<Book> books)
    {
        List<BookModel> result = new List<BookModel>();

        for (int i = 0; i < books.Count; i++)
        {
            result.Add(new BookModel
            {
                Id = books[i].Id,
                name = books[i].name,
                bookType = (BookType)books[i].BookType,
                authorName = books[i].authorName,
                page = books[i].page
            });
        }

        return result;
    }
```

Şekil 1

```
1 reference
public static Book ToBook(this BookRequest bookRequest)
{
    Book book = new Book
    {
        Id = bookRequest.Id,
        name = bookRequest.name,
        authorName = bookRequest.authorName,
        BookType = bookRequest.bookType,
        page = bookRequest.page
    };
    return book;
}
```

Şekil 2

Validation

Nesnelerin istenilen özelliklere göre doldurulması için validasyon kullanılmalıdır. Örneğin bir kütüphane uygulaması yaptığımızı varsayalım. Bu uygulamada kitapların tutulduğu bir modelde kitap ismi, yazar ismi gibi gerekli bilgilerin boş girilmemesi önemli olacaktır. Boş girildiği durumda validasyon ile bu sorunun önüne geçebiliriz. Şekil 3'de bulunan kod parçasından validasyonun mantığı daha iyi anlaşılacaktır. Burada yapılan kontrol mekanizması sayesinde kitap ismi ve yazar isminin boş bırakılmaması amaçlanmıştır. Ayrıca kitabın sayfa sayısı için de bir kontrol yazılmıştır.

```
[HttpPost]
0 references
public IActionResult Post([FromBody] BookRequest bookRequest)
{
    #region validation
    List<string> validations = new List<string>();

    if (bookRequest.name == string.Empty)
    {
        validations.Add("Kitap ismi boş olarak geçilemez");
    }

    if (bookRequest.authorName == string.Empty)
    {
        validations.Add("Yazar ismi boş olarak geçilemez");
    }

    if (bookRequest.page <= 0)
    {
        validations.Add("Kitabın sayfa sayısı 0 ya da daha küçük değer alamaz.");
    }

    if (validations.Any())
        return BadRequest(validations);

    #endregion validation
```

Şekil 3

Attribute

Attributelar; classlar ve methodlar gibi çeşitli öğelerin çalışma zamanına bilgi aktarmak için kullanılan etiketlerdir. Attribute kullanarak, programa etiket özellikler eklenebilir. Bu etiketler kullanılması istenen öğrenin üzerine köşeli ([]) parantezlerle gösterilir.

EFCore InMemory

Bellekte veritabanı depolamak için kullanılan bir yöntemdir. Gerçek veritabanını yüklemek ve yapılandırmak yerine uygulamanızın kodunu bellek içi bir veritabanına karşı test edebilmek InMemory provider kullanımının avantajlarından birisidir.

InMemory veritabanı kullanmak için öncelikle Microsoft.EntityFrameworkCore.InMemory NuGet paketinin uygun sürümü yüklenmelidir. Bu yükleme işlemi tamamlandıktan sonra InMemory veritabanı kullanımına başlanabilir.

InMemory veritabanı kullanımı adım adım aşağıda anlatılmıştır.

1. Öncelikle Şekil 4'deki gibi bir Entity içeren basit bir model oluşturulmuştur. Bu modelde kitap bilgileri tutulacaktır.

```
5 references
public class Book
{
    2 references
    public int Id { get; set; }
    2 references
    public string name { get; set; }
    2 references
    public string authorName { get; set; }
    0 references
    public string type { get; set; }
    2 references
    public int page { get; set; }
    2 references
    public int BookType { get; set; }
}
```

Şekil 4

2. Daha sonra Şekil 5'deki gibi DbContext sınıfı oluşturulmuştur.

```
6 references
public class DatabaseContext : DbContext
{
    2 references
    public DatabaseContext(DbContextOptions<DatabaseContext> options) : base(options)
    {

    }
    2 references
    public DbSet<Book> Books { get; set; }
}
```

Şekil 5

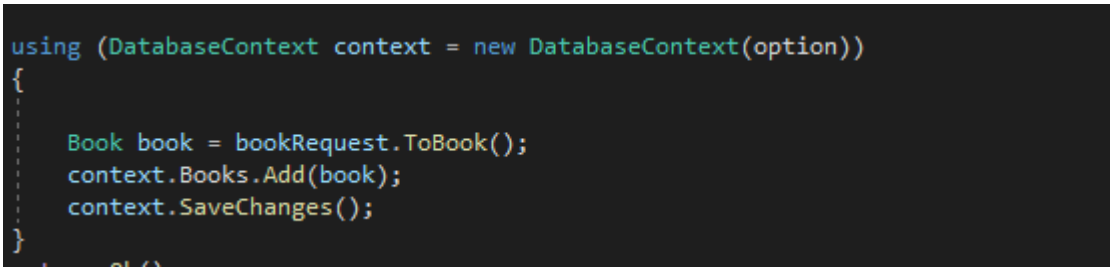
3. Son olarak InMemory veritabanını kullanmak için, bir DbContextOptions <DatabaseContext> instance oluşturulur ve In Memory olarak kullanılır.

```
private DbContextOptions<DatabaseContext> option;
0 references
public BookController()
{
    option = new DbContextOptionsBuilder<DatabaseContext>()
           .UseInMemoryDatabase(databaseName: "Book")
           .Options;

}
```

Şekil 5

Şekil 6'daki kod parçasında olduğu gibi oluşturulan instance kullanılırken, DatabaseContext sınıfının constructor kısmına verilir.



Şekil 6

Interface Class Nedir?

Bu bölümde nesneye yönelik programlamanın temel bileşenlerinden olan interface'lerden söz edeceğim. Interface'ler, kendilerini implement eden classlara rehberlik görevi üstlenirler. Bu rehberlik kavramından da anlaşılacağı gibi, interface tanımları içerisinde kod blokları bulunmamalıdır. Interface yapılar sadece içi boş tanımlamalardan ibarettir. Bu içi boş tanımlamalar Interface'in kullanılacağı class içerisinde işlevine uygun şekilde doldurulmaktadır. Ayrıca bir class'a birden çok interface eklemek mümkündür.

Abstract Class Nedir?

Abstract class'lar, nesneye yönelik programlamanın önemli bileşenlerinden bir tanesidir. Abstract Class, ortak özellikli Class'lara Base Class olma görevini üstlenir. Burada bahsedilen Base classta bulunan method ya da özelliklerin, base class'ı kalıtım ile kullanan sınıflarda ihtiyaç duyulduğu şekilde kullanılır. Base class içinde abstract ile işaretlenen metodlar ve propertyler bu sınıftan kalıtım alan her sınıfta yazılmak ve implement(uygulanmak) zorundadırlar. Ayrıca abstact tanımlamalar **private olarak yapılamazlar.**