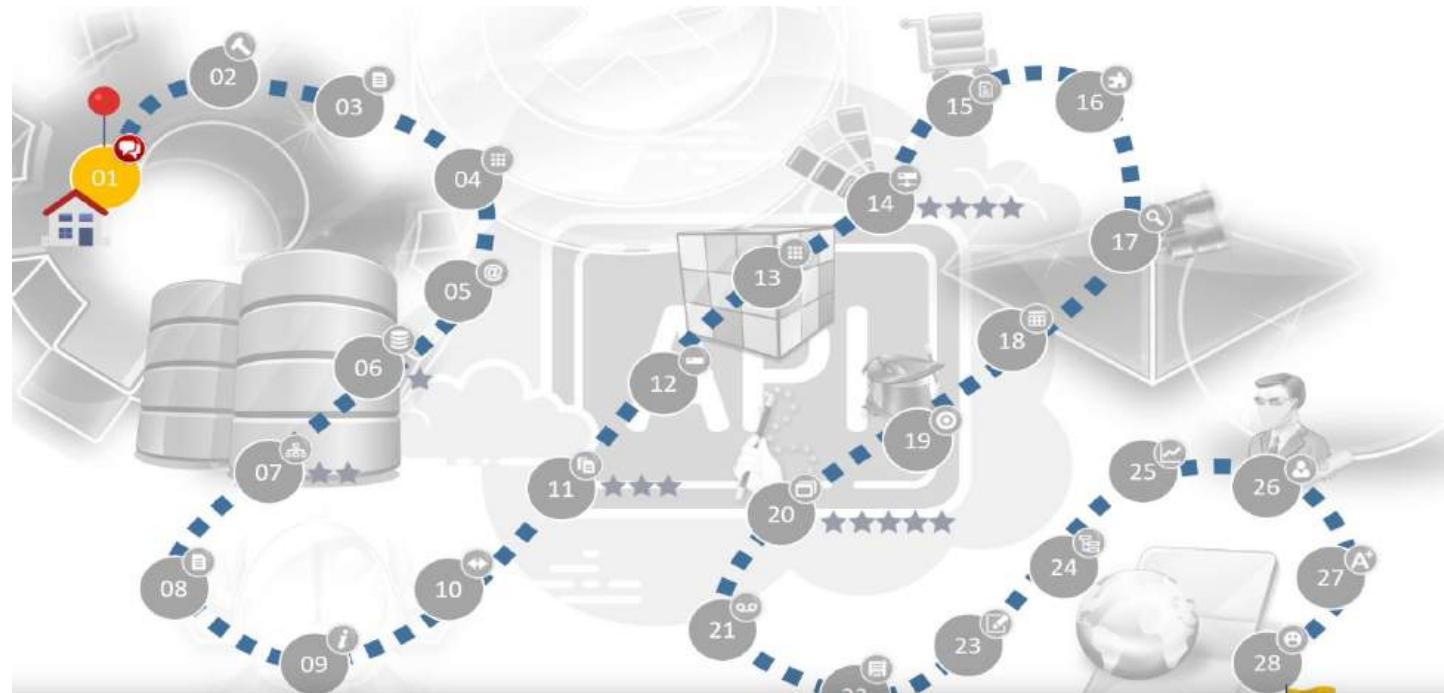




TÜRKİYE CUMHURİYETİ  
**MİLLÎ SAVUNMA BAKANLIĞI**

# **.NET CORE WEB API EĞİTİM SUNUMU**

# Eğitim Takvimi ?



# Neler Öğreneceğiz ?

- |                              |                        |                                   |
|------------------------------|------------------------|-----------------------------------|
| 1. API, HTTP, REST, RESTfull | 11. İçerik Pazarlığı   | 21. HEAD ve OPTIONS               |
| 2. Asp .Net Core             | 12. Doğrulama          | 22. Kök Belge                     |
| 3. Logging                   | 13. Asekron Kod        | 23. Versiyonlama                  |
| 4. Modeller ile Çalışma      | 14. Eylem Filtreleri   | 24. Ön Belleğe Alma               |
| 5. Postman                   | 15. Sayfalama          | 25. Hız Sınırı Kısıtlama          |
| 6. Entity Framework Core     | 16. Filtreleme         | 26. JWT Identity ve Reflesh Token |
| 7. Yazılım Mimarisi          | 17. Sıralama           | 27. API Dokümantasyon             |
| 8. Nlog Uygulaması           | 18. Arama              | 28. Bonus                         |
| 9. Global Hata Yönetimi      | 19. Veri Şekillendirme |                                   |
| 10. AutoMapper               | 20. Hateoas            |                                   |

## Ön Gereksinimler ?

- Temel Algoritma Bilgisi
- Temel C# Bilgisi
- Veri Yapılarına ve Algoritma Bilgisi
- Nesne Yönelimli Programlama Bilgisi

## Yazılımlar ?

- Visual Studio 2022 (Community)
- SQL Server
- SQL Server Management Studio
- Postman
- Git

NOT: Uygulamalarımız ASP.NET Core Runtime 6.0 versiyonu kullanacağız.

# Birinci Bölüm?

## API KAVRAMI

API, Application Programming Interface (Uygulama Programlama Arayüzü) kelimelerinin kısaltmasıdır. Kısaca, farklı yazılım uygulamalarının birbirleriyle iletişim kurmasını sağlayan bir ara yüzdür.

### Daha açıklayıcı bir şekilde:

Bir API, bir yazılımın sunduğu işlevlere başka yazılımların erişebilmesi için standart bir yol belirler. Bu, uygulamaların veri almasını, veri göndermesini veya belirli işlemleri uzaktan tetiklemesini sağlar.

### Basit bir örnek

Diyelim bir hava durumu uygulaması yapıyorsun ve güncel hava durumu verilerine ihtiyacın var. Bunun için bir hava durumu API'sine istek gönderirsın. O API, sana sıcaklık, rüzgar, yağış gibi bilgileri JSON gibi standart bir formatta geri gönderir.

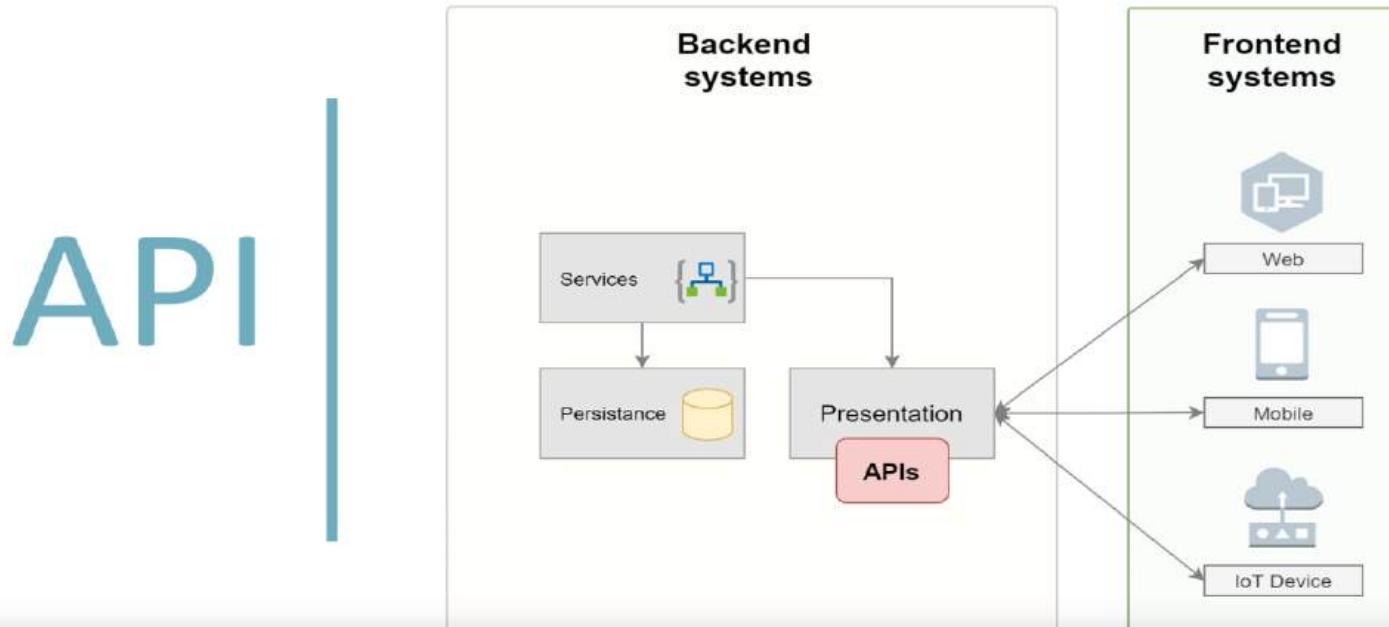
# Birinci Bölüm?

## Neden API Kullanılır?

- Tekrar kod yazmadan başkalarının sunduğu hizmetlerden yararlanmak.
- Sistemler arası entegrasyonu kolaylaştırmak.
- Güvenli ve kontrollü bir şekilde bilgi paylaşımı yapmak.

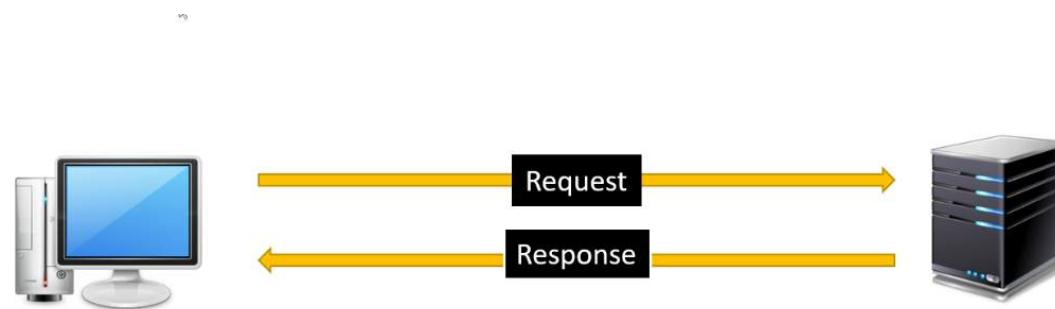
# Birinci Bölüm?

API Topoloji



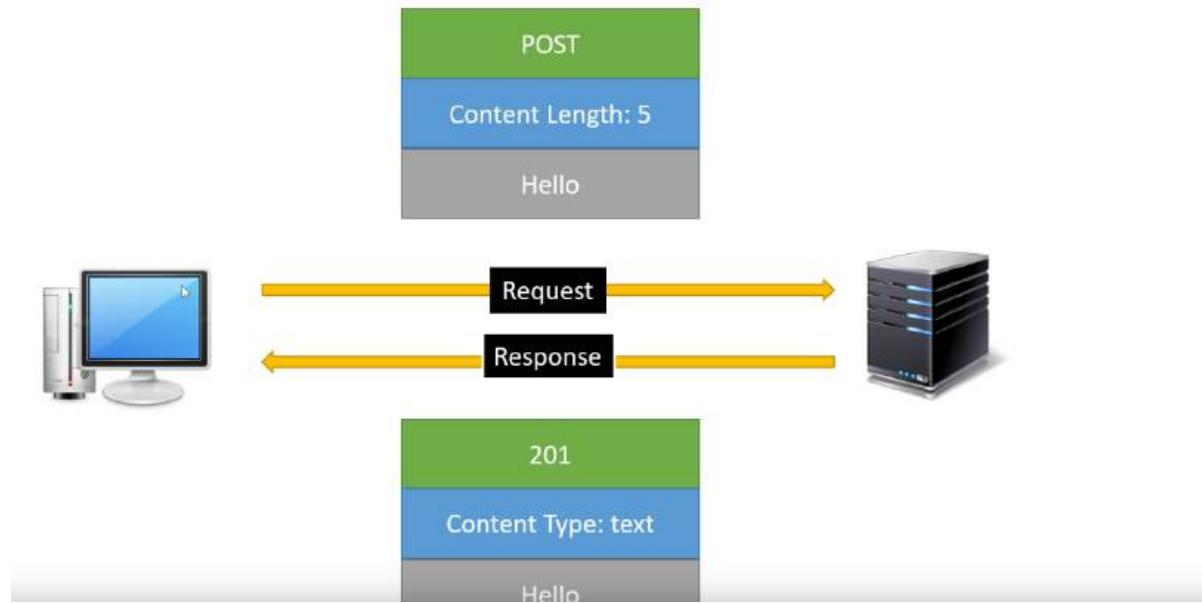
# Birinci Bölüm?

HTTP Nasıl Çalışır ?



# Birinci Bölüm?

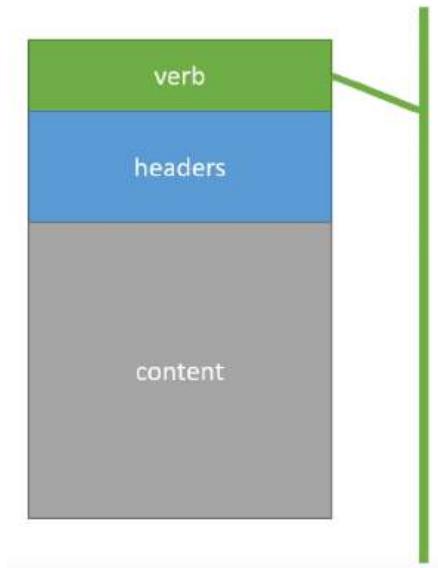
HTTP Nasıl Çalışır ?



# Birinci Bölüm?

İsteğin (Request) Yapısı?

Verb (Request) Yapısı?



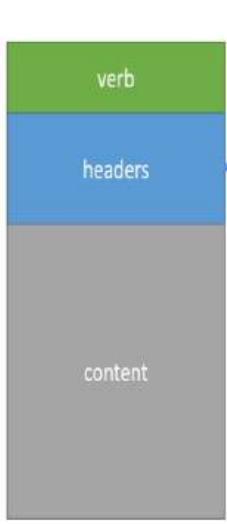
Sunucuda gerçekleştirilen eylemler:

- **GET**
  - Kaynak isteme
- **POST**
  - Kaynak oluşturma
- **PUT**
  - Kaynak güncelleme
- **PATCH**
  - Kismi kaynak güncellemesi
- **DELETE**
  - Kaynak silme

# Birinci Bölüm?

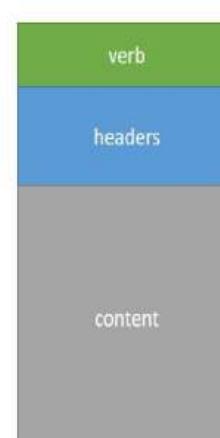
İsteğin (Request) Yapısı ?

Headers ?



İstek hakkında üst (meta) bilgiler:

- **Content Type**
  - İçeriğin formatı
- **Content Length**
  - İçeriğin boyutu
- **Authorization**
  - İsteği yapanın kimliği



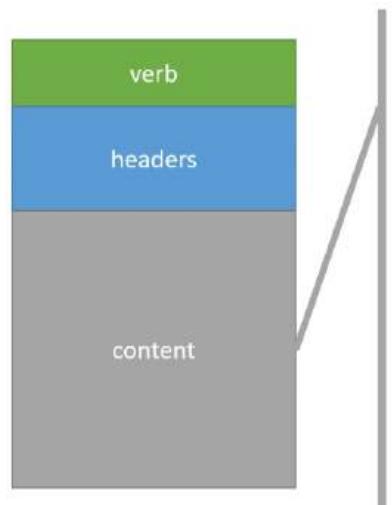
İstek hakkında üst (meta) bilgiler:

- **Accept**
  - Kabul edilen tipler
- **Cookies**
  - İstek içindeki veriler
- ve daha fazlası...

# Birinci Bölüm?

İsteğin (Request) Yapısı ?

Content ?



## İstek ile ilgili içerik:

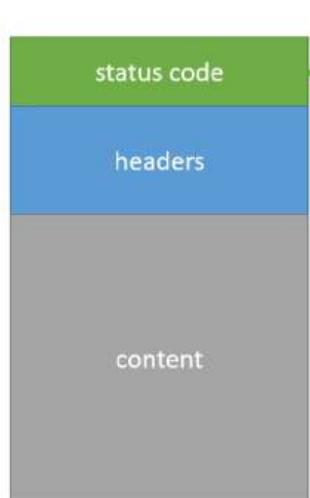
- HMTL, CSS, JavaScript, XML, JSON
- Bazı eylemler ile geçerli olmayan içerik
- İsteği gerçekleştirmeye yardımcı olmak için bilgiler
- Binary ve blobs common (.jpg gibi)



# Birinci Bölüm?

Cevabin (Response) Yapısı?

Status Code ?



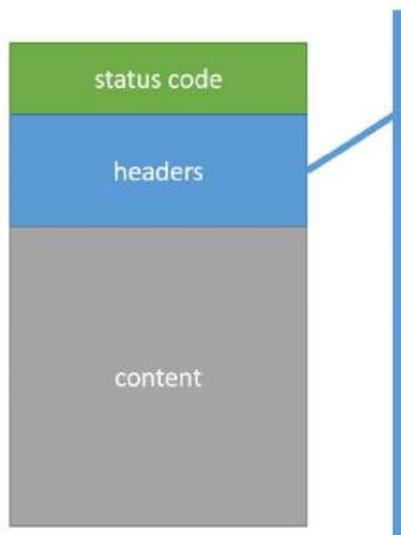
## Operasyon Durumları

- 100 – 199
  - Bilgi (Information)
- 200 – 299
  - Başarı (Success)
- 300 – 399
  - Yeniden yönlendirme (Redirection)
- 400 – 499
  - İstemci hataları (Client errors)
- 500 – 599
  - Sunucu hataları (Server errors)

# Birinci Bölüm?

Cevabin (Response) Yapısı ?

Headers ?



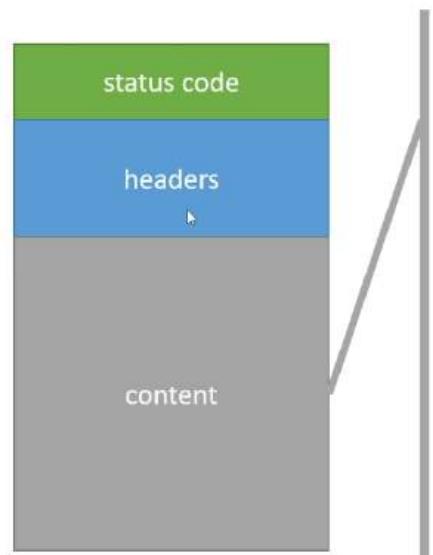
**Cevap hakkında üst (meta) bilgiler:**

- **Content Type**
  - İçeriğin formatı
- **Content Length**
  - İçeriğin boyutu
- **Expires**
  - Ne zaman geçersiz sayılır

# Birinci Bölüm?

Cevabin (Response) Yapısı ?

Content ?



İçerik:

- HTML, CSS, JavaScript, XML, JSON
- Binary ve blobs common (.jpg)

# Birinci Bölüm?

RESTfull API Tasarımı ?

(İstemci sunucu arasındaki kaynağın değiştirilmesi)

İstek atarken



# Birinci Bölüm?

RESTfull API Tasarımı ?

Sonuç

## REST API Bileşenleri



# Birinci Bölüm?

RESTfull API Tasarımı ?

En iyi  
pratikler

- Sonuçların kendini tanımlamasını sağlayınız.
- Programlı gezinmeye izin veriniz.
- Sayfalama, sıralama, filtreleme ve arama desteği sağlayınız.
- Önbellek desteği sununuz.
- Sorgu sınırı getiriniz.
- Veri şekillendirilmesini sağlayınız.
- Versiyonlama yapınız.
- Belgelendirmeye özen gösteriniz.

En iyi  
pratikler

**İsimleri (nouns) tercih edin**

- /books
- /categories
- /products
- /employees
- /authors

**Query Strings**

- Kaynak olmayan özellikler için Query String kullanın.

- /books?sort=title
- /books?page=1
- /books?pageNumber=1&pageSize=10
- /books?format=json

# Birinci Bölüm?

RESTfull API Tasarımı ?

- Lastest (Son Kitaplar)
- Monstread (En fazla okunan kitaplar)
- ./boks/latest/10

Get, Save, Update Delete olmaması  
yazım kuralları gereği önemlidir.

URIs

En iyi  
pratikler

- URI'ler kaynakların sadece bir parçasıdır.
  - ./books
  - ./books/lastest
  - ./books/mostread
- Veri olmayan öğeler için sorgu dizeleri (Query String)
  - Format
  - Sıralama

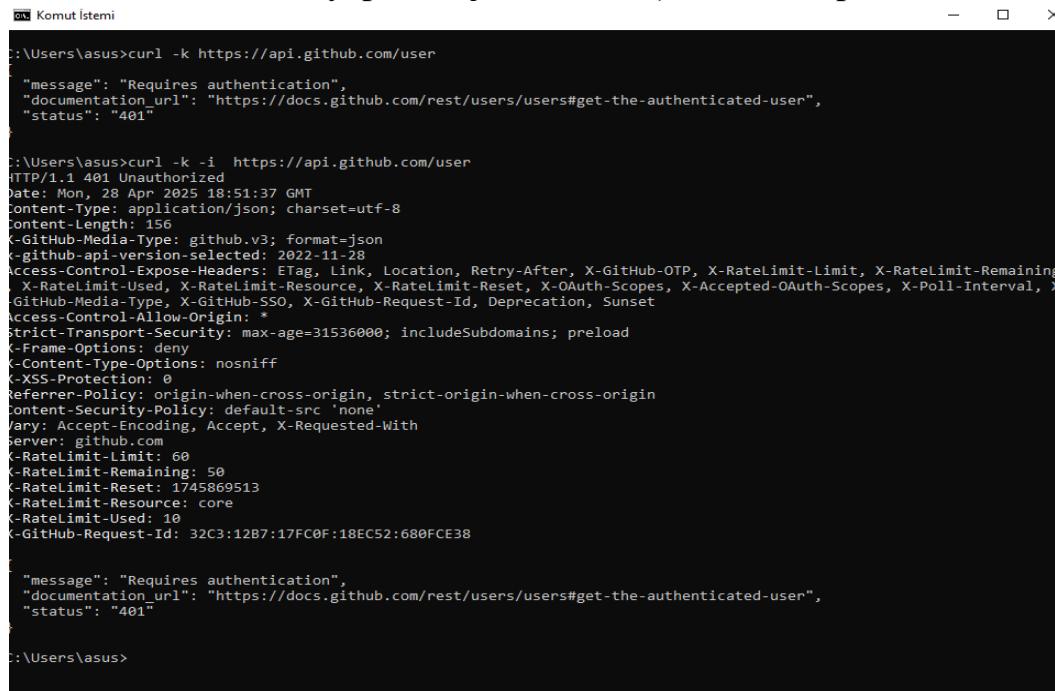
İsimler (nouns) iyi, filler (kötü)

- /getBooks
- /getBooksByTitle
- /getNewBooks
- /verifyOrders
- /saveBook
- /updateBook
- /deleteBook
- ...

# Birinci Bölüm?

RESTfull API Tasarımı ?

CURL (İnternet üzerinden veri transferi yapmak için kullanılır) (Restfull api nasıl davranış? Body Headers nasıl gördük)



```
:\Users\asus>curl -k https://api.github.com/user
{
  "message": "Requires authentication",
  "documentation_url": "https://docs.github.com/rest/users/users#get-the-authenticated-user",
  "status": "401"
}

:\Users\asus>curl -k -i https://api.github.com/user
HTTP/1.1 401 Unauthorized
Date: Mon, 28 Apr 2025 18:51:37 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 156
X-GitHub-Media-Type: github.v3; format=json
X-GitHub-API-Version-Selected: 2022-11-28
Access-Control-Expose-Headers: ETag, Link, Location, Retry-After, X-GitHub-OTP, X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Used, X-RateLimit-Resource, X-RateLimit-Reset, X-OAuth-Scopes, X-Accepted-OAuth-Scopes, X-Poll-Interval, X-GitHub-Media-Type, X-Github-SSO, X-GitHub-Request-Id, Deprecation, Sunset
Access-Control-Allow-Origin: *
Strict-Transport-Security: max-age=31536000; includeSubdomains; preload
X-Frame-Options: deny
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Referrer-Policy: origin-when-cross-origin, strict-origin-when-cross-origin
Content-Security-Policy: default-src 'none'
Vary: Accept-Encoding, Accept, X-Requested-With
Server: github.com
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 50
X-RateLimit-Reset: 1745869513
X-RateLimit-Resource: core
X-RateLimit-Used: 10
X-GitHub-Request-Id: 32C3:12B7:17FC0F:18EC52:680FCE38

{
  "message": "Requires authentication",
  "documentation_url": "https://docs.github.com/rest/users/users#get-the-authenticated-user",
  "status": "401"
}

:\Users\asus>
```

# Birinci Bölüm?

RESTfull API Tasarımı ?

CURL

```
C:\Users\asus>curl -k -i https://api.github.com/users
HTTP/1.1 200 OK
Date: Mon, 28 Apr 2025 18:53:31 GMT
Content-Type: application/json; charset=utf-8
Cache-Control: public, max-age=60, s-maxage=60
Vary: Accept,Accept-Encoding, Accept, X-Requested-With
ETag: W/"162f05f79d01914481b52960513822e56399cf139a57c54edbc762c45+086869"
X-GitHub-Media-Type: github.v3; format=json
Link: <https://api.github.com/users?since=46>; rel="next", <https://api.github.com/users/?since>; rel="first"
X-GitHub-API-Version: selected: 2022-11-28
Access-Control-Expose-Headers: Etag, Link, Location, Retry-After, X-GitHub-OTP, X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Used, X-RateLimit-Resource, X-RateLimit-Reset, X-OAuth-Scopes, X-Accepted-OAuth-Scopes, X-Poll-Interval, X-GitHub-Media-Type, X-GitHub-SSO, X-GitHub-Request-ID, Deprecation, Sunset
Access-Control-Allow-Origin: *
Strict-Transport-Security: max-age=31536000; includeSubdomains; preload
X-Frame-Options: deny
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Referrer-Policy: origin-when-cross-origin, strict-origin-when-cross-origin
Content-Security-Policy: default-src 'none'
Server: GitHub.com
Accept-Ranges: bytes
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 48
X-RateLimit-Reset: 1745809513
X-RateLimit-Resource: core
X-RateLimit-Used: 12
Transfer-Encoding: chunked
X-GitHub-Request-Id: 2C70:E408E:CF1A07:D4E3C5:680FCEAB

{
  "login": "mojombo",
  "id": 1,
  "node_id": "MDQgXXNlcIE=",
  "avatar_url": "https://avatars.githubusercontent.com/u/1?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/mojombo",
  "html_url": "https://github.com/mojombo",
  "followers_url": "https://api.github.com/users/mojombo/followers",
  "following_url": "https://api.github.com/users/mojombo/following{/other_user}",
  "gists_url": "https://api.github.com/users/mojombo/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/mojombo/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/mojombo/subscriptions",
  "organizations_url": "https://api.github.com/users/mojombo/orgs",
  "repos_url": "https://api.github.com/users/mojombo/repos",
  "events_url": "https://api.github.com/users/mojombo/events{/privacy}",
  "received_events_url": "https://api.github.com/users/mojombo/received_events",
  "type": "User",
  "user_view_type": "public",
}
```

```
C:\Users\asus>curl -k -i https://api.github.com/users/byrmrk
HTTP/1.1 200 OK
Date: Mon, 28 Apr 2025 18:53:35 GMT
Content-Type: application/json; charset=utf-8
Cache-Control: public, max-age=60, s-maxage=60
Vary: Accept,Accept-Encoding, Accept, X-Requested-With
ETag: W/"31b7bb95fcd86124abb2cd9fb649751e7feb07bb53f57345bfed7f27ede454"
Last-Modified: Sat, 28 Apr 2025 06:32:35 GMT
X-GitHub-Media-Type: github.v3; format=json
X-GitHub-Request-Id: 203E:18F613:2F4D71:30EBFD:680FCF02
X-GitHub-API-Version: selected: 2022-11-28
Access-Control-Expose-Headers: Etag, Link, Location, Retry-After, X-GitHub-OTP, X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Used, X-RateLimit-Resource, X-RateLimit-Reset, X-OAuth-Scopes, X-Accepted-OAuth-Scopes, X-Poll-Interval, X-GitHub-Media-Type, X-GitHub-SSO, X-GitHub-Request-ID, Deprecation, Sunset
Access-Control-Allow-Origin: *
Strict-Transport-Security: max-age=31536000; includeSubdomains; preload
X-Frame-Options: deny
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Referrer-Policy: origin-when-cross-origin, strict-origin-when-cross-origin
Content-Security-Policy: default-src 'none'
Server: GitHub.com
Accept-Ranges: bytes
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 47
X-RateLimit-Reset: 1745809513
X-RateLimit-Resource: core
X-RateLimit-Used: 13
Content-Length: 3083
X-GitHub-Request-Id: 203E:18F613:2F4D71:30EBFD:680FCF02

{
  "login": "byrmrk",
  "id": 56492479,
  "node_id": "MDQgXzIwOTkxNjkyN0cs",
  "avatar_url": "https://avatars.githubusercontent.com/u/56492479?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/byrmrk",
  "html_url": "https://github.com/byrmrk",
  "followers_url": "https://api.github.com/users/byrmrk/followers",
  "following_url": "https://api.github.com/users/byrmrk/following{/other_user}",
  "gists_url": "https://api.github.com/users/byrmrk/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/byrmrk/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/byrmrk/subscriptions",
  "organizations_url": "https://api.github.com/users/byrmrk/orgs",
  "repos_url": "https://api.github.com/users/byrmrk/repos",
  "events_url": "https://api.github.com/users/byrmrk/events{/privacy}",
  "received_events_url": "https://api.github.com/users/byrmrk/received_events",
  "type": "User",
  "user_view_type": "public",
  "site_admin": false,
  "name": "Bayram Gök",
  "company": null,
  "blog": "",
  "location": null,
  "hireable": null,
  "bio": null,
  "twitter_username": null,
  "public_repos": 1,
  "public_gists": 0
}
```

**Birinci Bölüm Bitti**  
**Hadi Özetleyelim?**

# İkinci Bölüm?

ASP .NET CORE TEMELLERİ

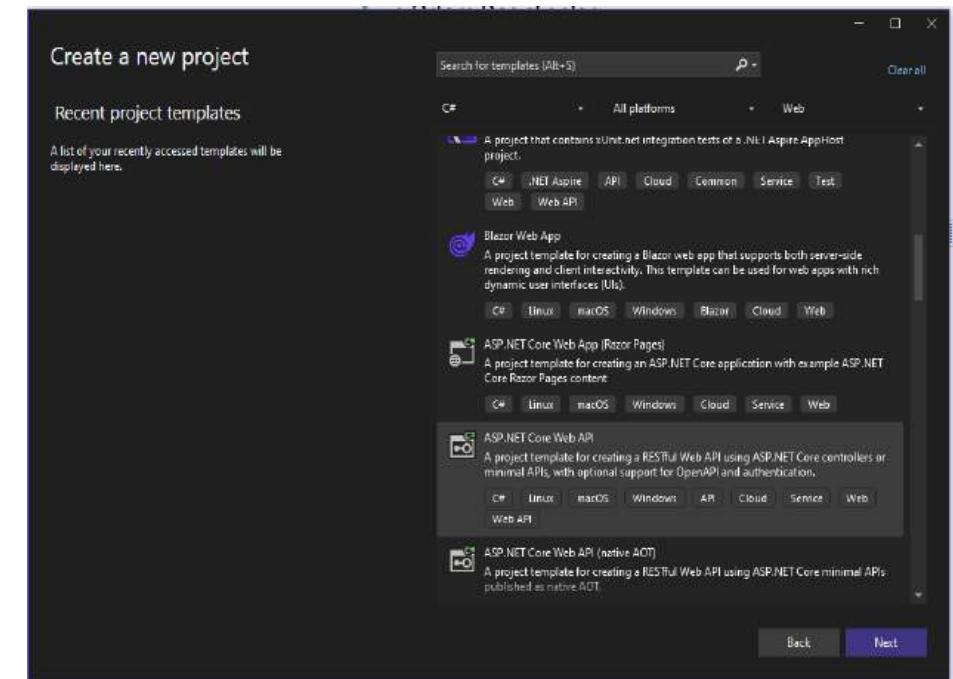
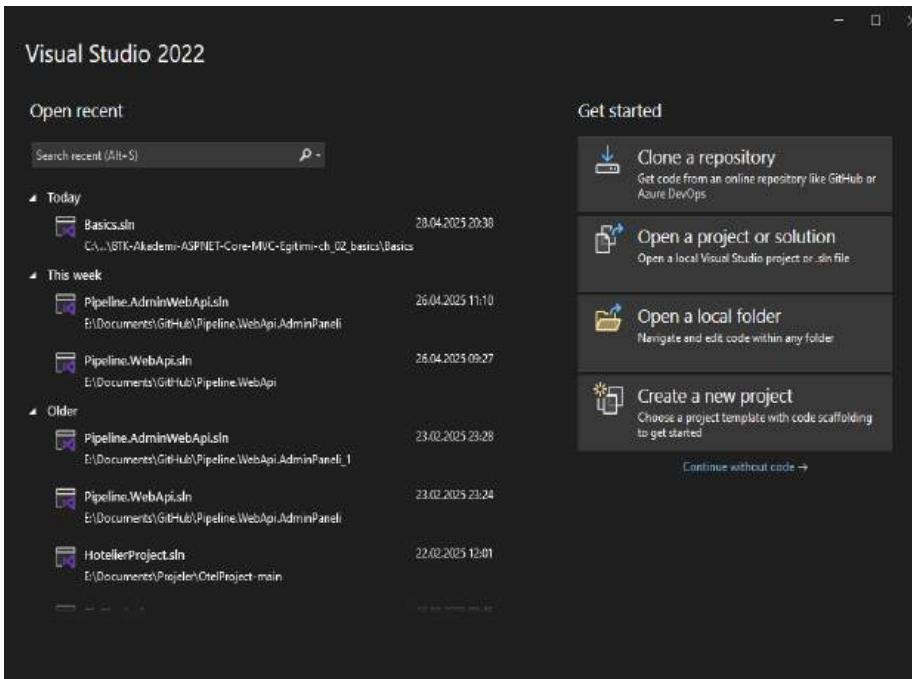
## İçerik

- dotnet core UI | CLI
- Proje Şablonunun İncelenmesi
- Proje Şablonu Uygulanması
- Ortam Değişkenleri

# İkinci Bölüm?

## ASP .NET CORE TEMELLERİ

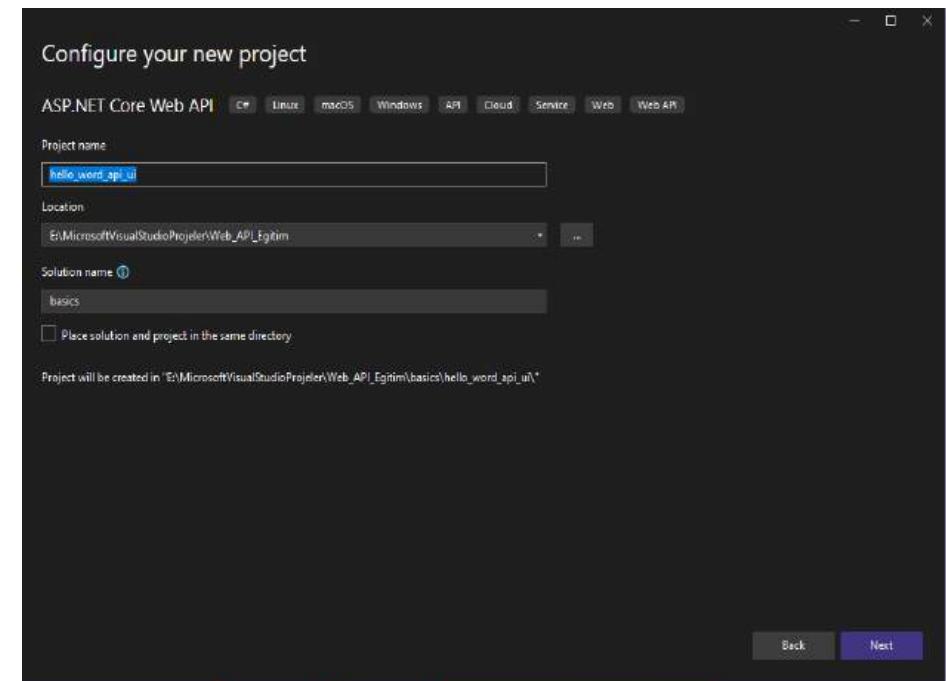
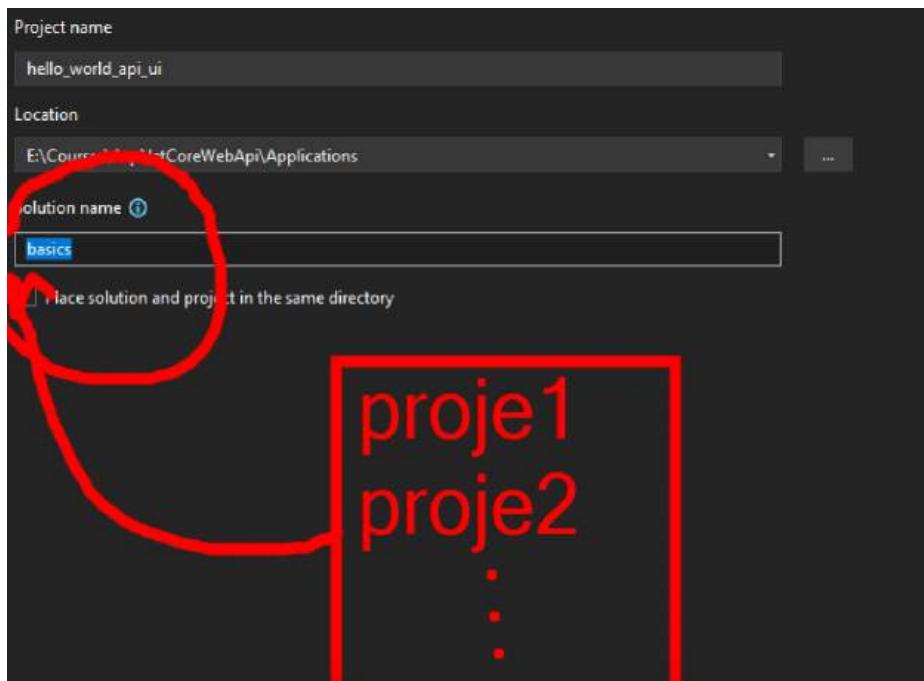
### UI | CLI ile Proje Oluşturma



# İkinci Bölüm?

## ASP .NET CORE TEMELLERİ

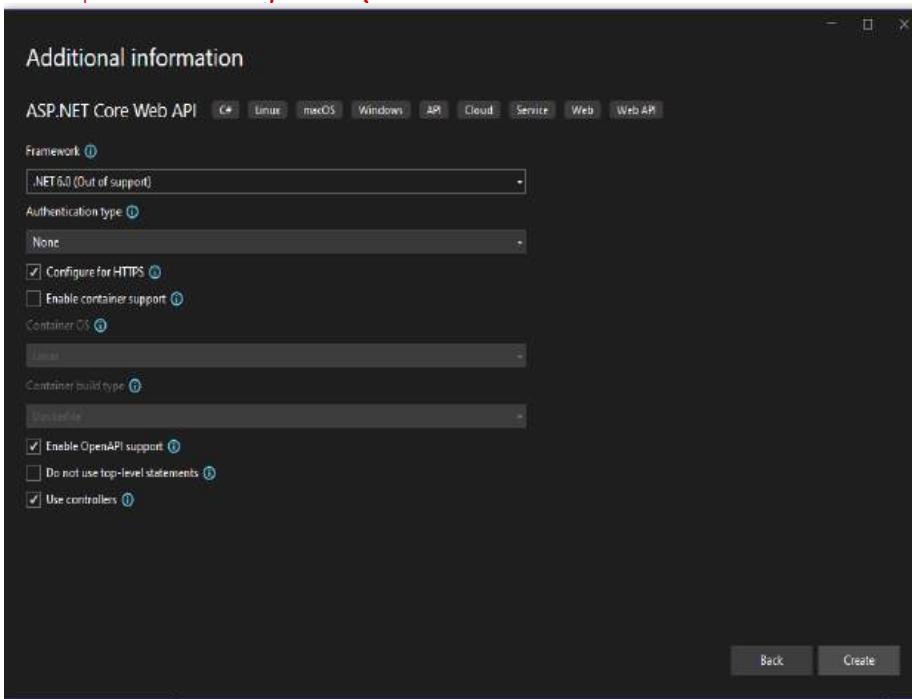
### UI | CLI ile Proje Oluşturma



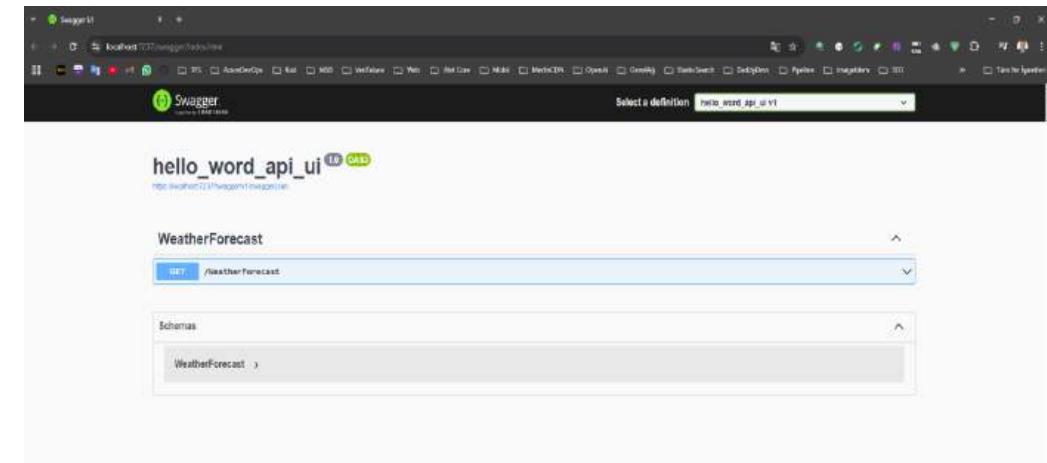
# İkinci Bölüm?

## ASP .NET CORE TEMELLERİ

### UI | CLI ile Proje Oluşturma



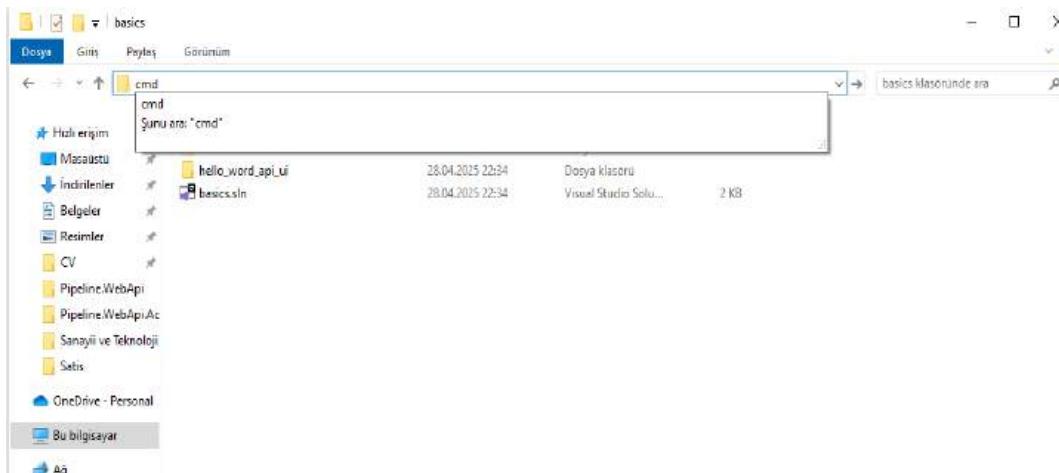
Uygulamayı Çalıştır !!!



# İkinci Bölüm?

## ASP .NET CORE TEMELLERİ

### UI | CLI ile Proje Oluşturma



CLI komut url :

<https://learn.microsoft.com/en-us/dotnet/core/tools/>

```
C:\Windows\System32\cmd.exe
::\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics>dotnet
Usage: dotnet [options]
Usage: dotnet [path-to-application]

Options:
  -h|--help      Display help.
  --info         Display .NET information.
  --list-sdks    Display the installed SDKs.
  --list-runtimes Display the installed runtimes.

Path-to-application:
  The path to an application .dll file to execute.

::\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics>dotnet new webapi -o hello_word_cli
"ASP.NET Core Web API'si" şablonu başarıyla oluşturuldu.

Oluşturma sonrası eylemleri işleniyor...
::\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics\hello_word_cli\hello_word_cli.csproj geri yükleniyor:
Geri yükleme başarılı oldu.

::\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics>dir
Volume in drive E is DATA
Volume Serial Number is 7A6D-8A0F

Directory of E:\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics

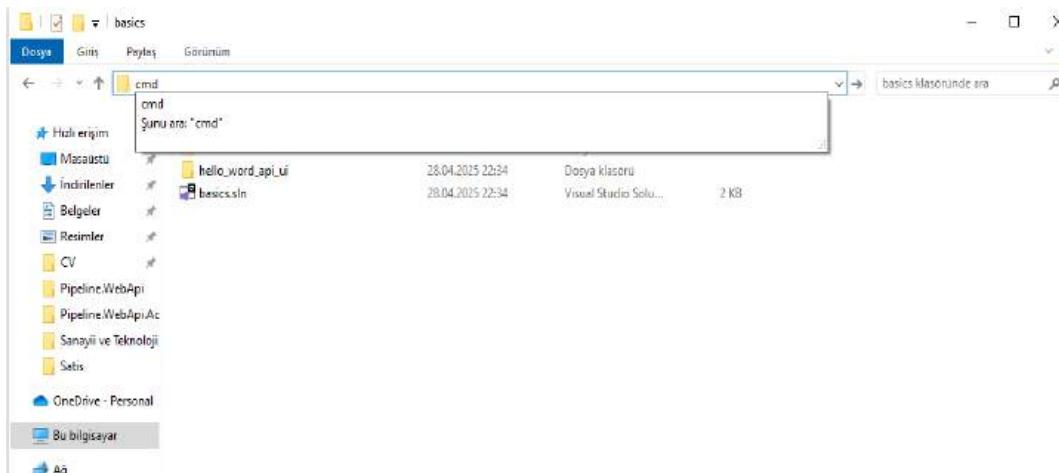
28.04.2025 22:40    <DIR>    .
28.04.2025 22:40    <DIR>    ..
28.04.2025 22:34            1.153 basics.sln
28.04.2025 22:34    <DIR>    hello_word_api_ui
28.04.2025 22:40    <DIR>    hello_word_cli
               1 File(s)     1.153 bytes
               4 Dir(s)   577.402.654.720 bytes free

::\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics>cd hello_word_cli
```

# İkinci Bölüm?

## ASP .NET CORE TEMELLERİ

### UI | CLI ile Proje Oluşturma



CLI komut url :

<https://learn.microsoft.com/en-us/dotnet/core/tools/>

```
C:\Windows\System32\cmd.exe
::\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics>dotnet
Usage: dotnet [options]
Usage: dotnet [path-to-application]

Options:
  -h|--help      Display help.
  --info         Display .NET information.
  --list-sdks    Display the installed SDKs.
  --list-runtimes Display the installed runtimes.

Path-to-application:
  The path to an application .dll file to execute.

::\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics>dotnet new webapi -o hello_word_cli
"ASP.NET Core Web API'si" şablonu başarıyla oluşturuldu.

Oluşturma sonrası eylemleri işleniyor...
::\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics\hello_word_cli\hello_word_cli.csproj geri yükleniyor:
Geri yükleme başarılı oldu.

::\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics>dir
Volume in drive E is DATA
Volume Serial Number is 7A6D-8A0F

Directory of E:\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics

28.04.2025 22:40    <DIR>    .
28.04.2025 22:40    <DIR>    ..
28.04.2025 22:34            1.153 basics.sln
28.04.2025 22:34    <DIR>    hello_word_api_ui
28.04.2025 22:40    <DIR>    hello_word_cli
               1 File(s)     1.153 bytes
               4 Dir(s)   577.402.654.720 bytes free

::\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics>cd hello_word_cli
```

# İkinci Bölüm?

## ASP .NET CORE TEMELLERİ

### UI | CLI ile Proje Oluşturma

```
CAWindows\System32\cmd.exe - dotnet run

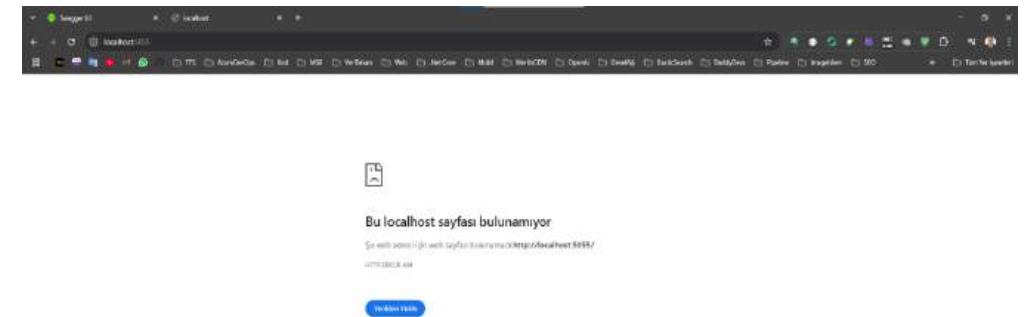
E:\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics\hello_word_cli>dir
Volume in drive E is DATA
Volume Serial Number is 7A6D-8A0F

Directory of E:\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics\hello_word_cli

28.04.2025 22:40    <DIR>        .
28.04.2025 22:40    <DIR>        ..
28.04.2025 22:40           127 appsettings.Development.json
28.04.2025 22:40           151 appsettings.json
28.04.2025 22:40           333 hello_word_cli.csproj
28.04.2025 22:40           141 hello_word_cli.http
28.04.2025 22:40    <DIR>        obj
28.04.2025 22:40           1.067 Program.cs
28.04.2025 22:40    <DIR>        Properties
               5 File(s)      1.819 bytes
               4 Dir(s)   577.402.654.720 bytes free

E:\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics\hello_word_cli>dotnet run
E:\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics\hello_word_cli\Properties\launchSettings.json icindeki başlatma ayarları kullanılıyor...
Derleniyor...
Info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5055
Info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
Info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
Info: Microsoft.Hosting.Lifetime[0]
      Content root path: E:\MicrosoftVisualStudioProjeler\Web_API_Egitim\basics\hello_word_cli
```

Çalıştır !!!  
Neden Hata Oluştı !!!

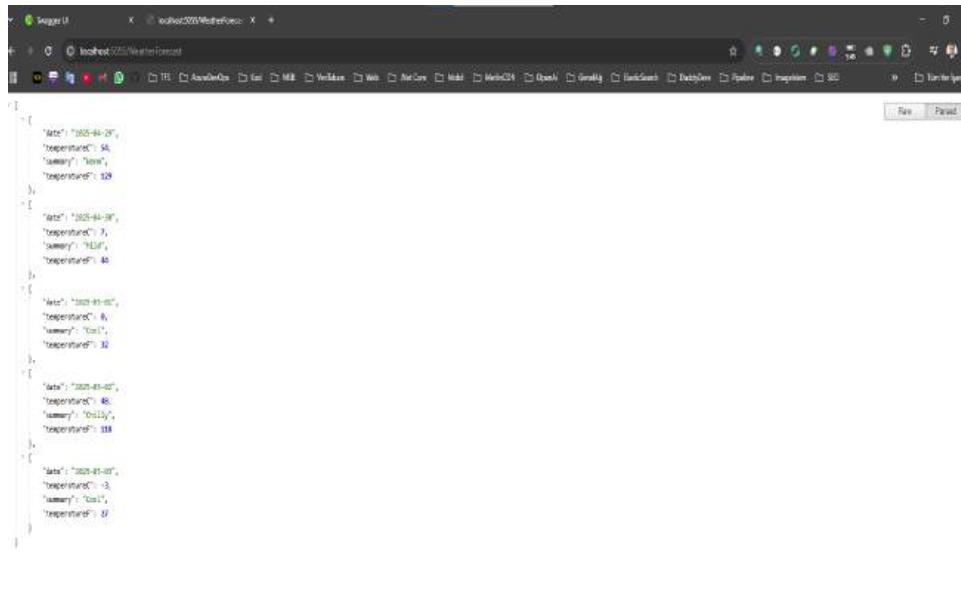


# İkinci Bölüm?

ASP .NET CORE TEMELLERİ

UI | CLI ile Proje Oluşturma

/controller yok !!!



A screenshot of a browser window titled "Sergej" showing a JSON response. The URL is "localhost:5001/weatherforecast". The JSON data consists of an array of five objects, each representing a weather forecast entry with properties: "date", "temperatureC", "summary", and "temperatureF".

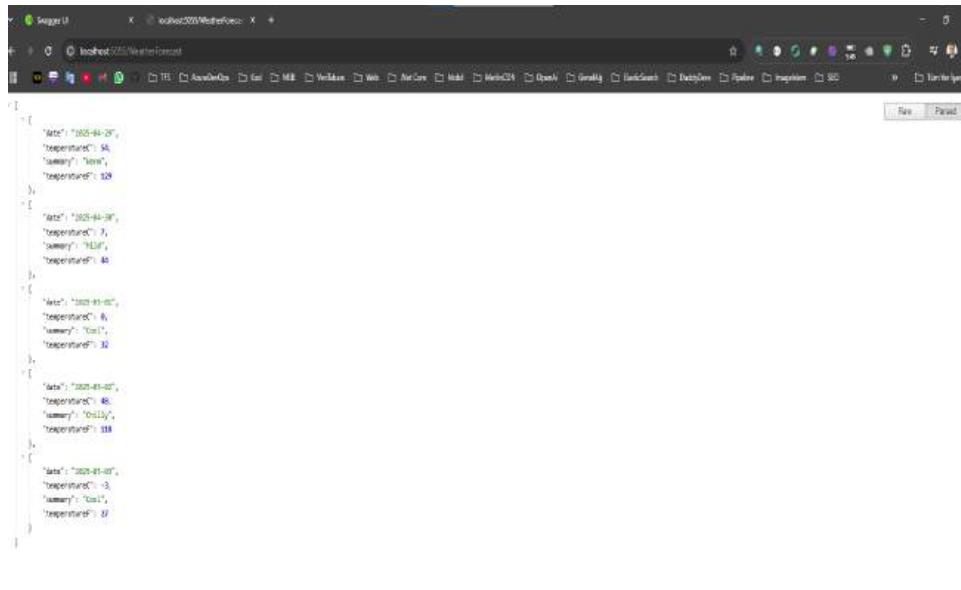
```
[{"date": "2023-01-28", "temperatureC": 15, "summary": "Very nice", "temperatureF": 59}, {"date": "2023-01-29", "temperatureC": 7, "summary": "Held", "temperatureF": 44}, {"date": "2023-01-30", "temperatureC": 6, "summary": "Cloudy", "temperatureF": 41}, {"date": "2023-01-31", "temperatureC": 18, "summary": "Cloudy", "temperatureF": 64}, {"date": "2023-02-01", "temperatureC": -3, "summary": "Cloudy", "temperatureF": 20}]
```

# İkinci Bölüm?

ASP .NET CORE TEMELLERİ

UI | CLI ile Proje Oluşturma

/controller yok !!!



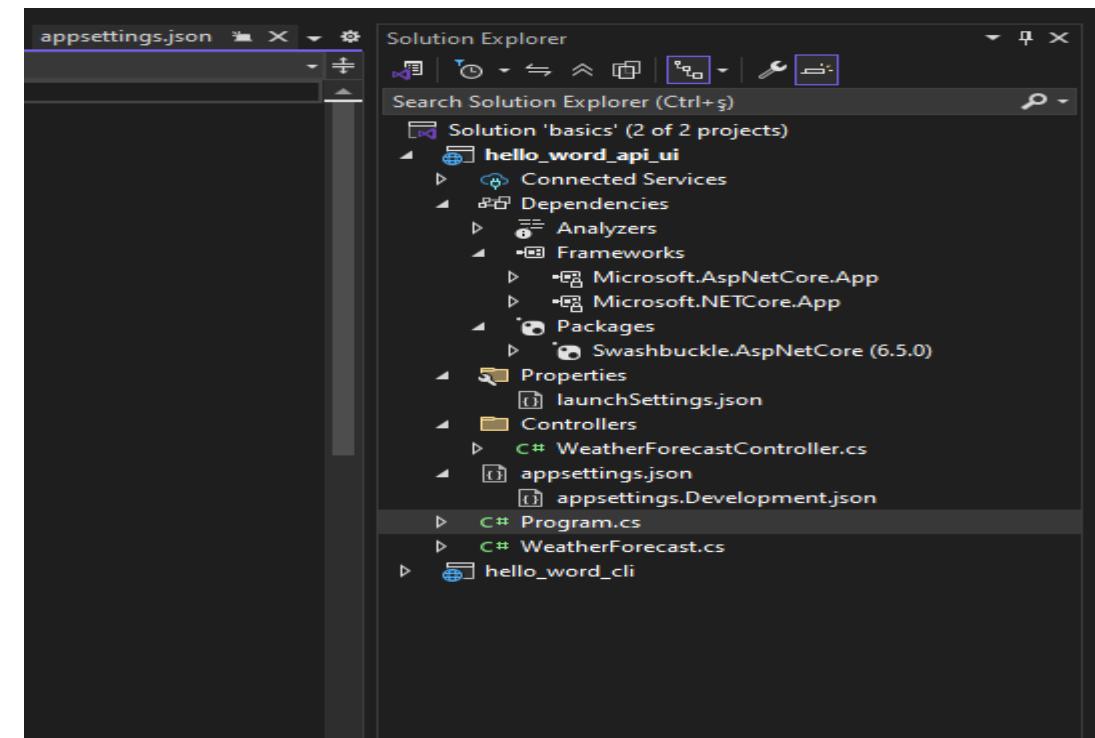
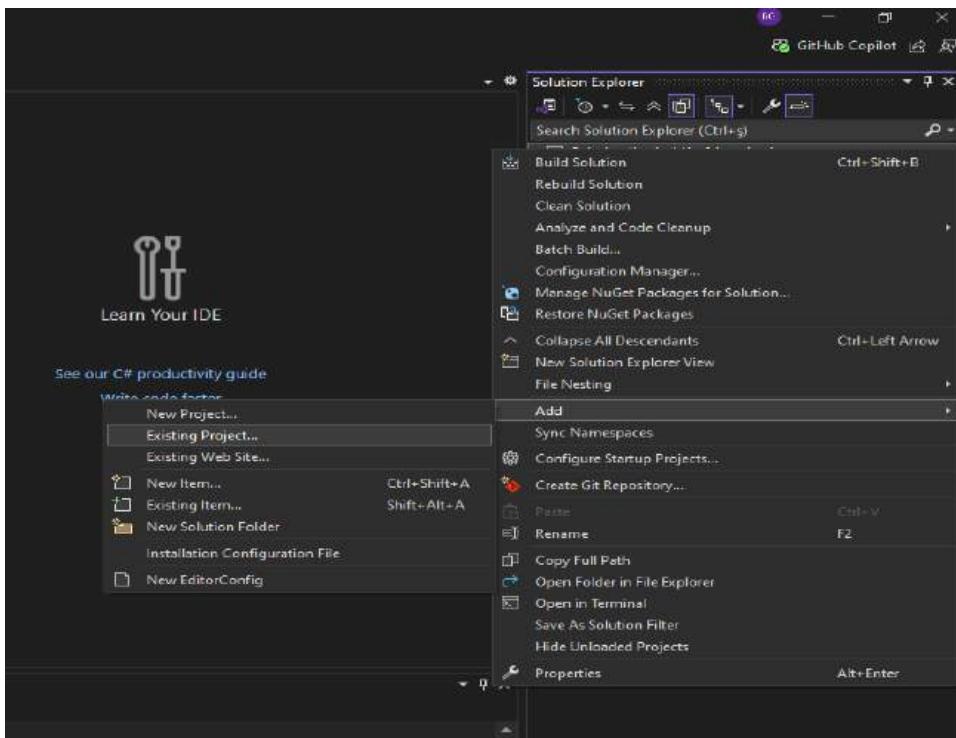
A screenshot of a browser window titled "Sergej" showing a JSON response. The URL is "localhost:5001/weatherforecast". The JSON data consists of an array of five objects, each representing a weather forecast entry with properties: Date, TemperatureC, Summary, and TemperatureF.

```
[{"Date": "2023-01-01", "TemperatureC": 15, "Summary": "Cloudy", "TemperatureF": 59}, {"Date": "2023-01-02", "TemperatureC": 10, "Summary": "Rainy", "TemperatureF": 50}, {"Date": "2023-01-03", "TemperatureC": 6, "Summary": "Cloudy", "TemperatureF": 43}, {"Date": "2023-01-04", "TemperatureC": 8, "Summary": "Cloudy", "TemperatureF": 46}, {"Date": "2023-01-05", "TemperatureC": 3, "Summary": "Cloudy", "TemperatureF": 37} ]
```

# İkinci Bölüm?

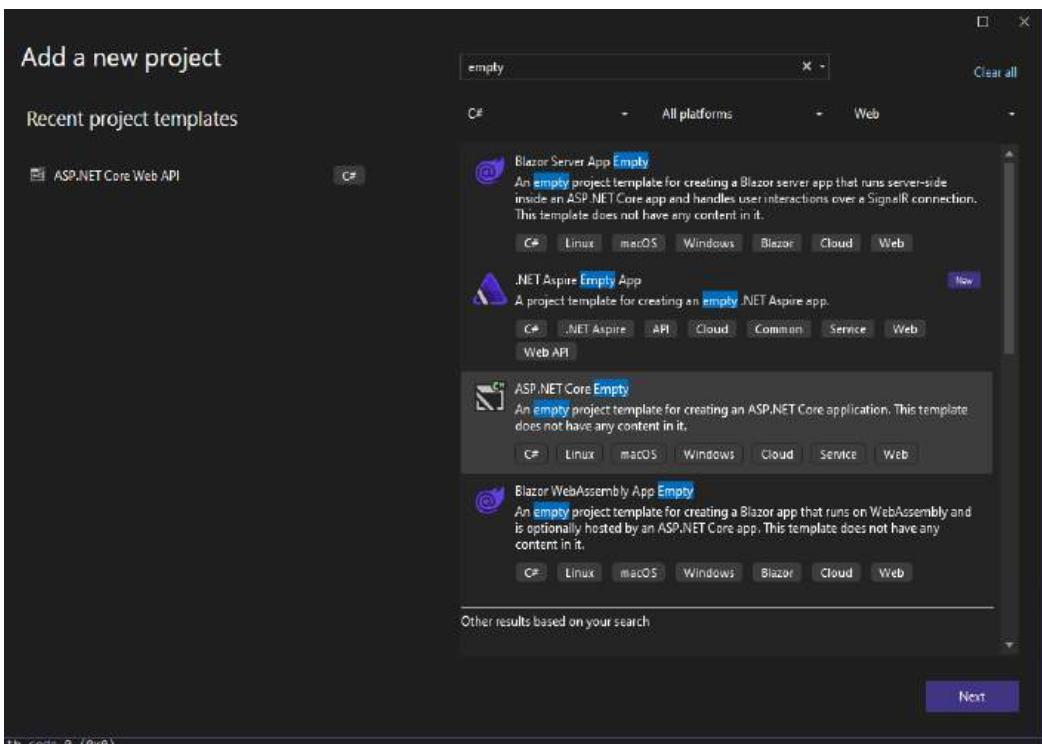
## ASP .NET CORE TEMELLERİ

### UI | CLI ile Proje Oluşturma

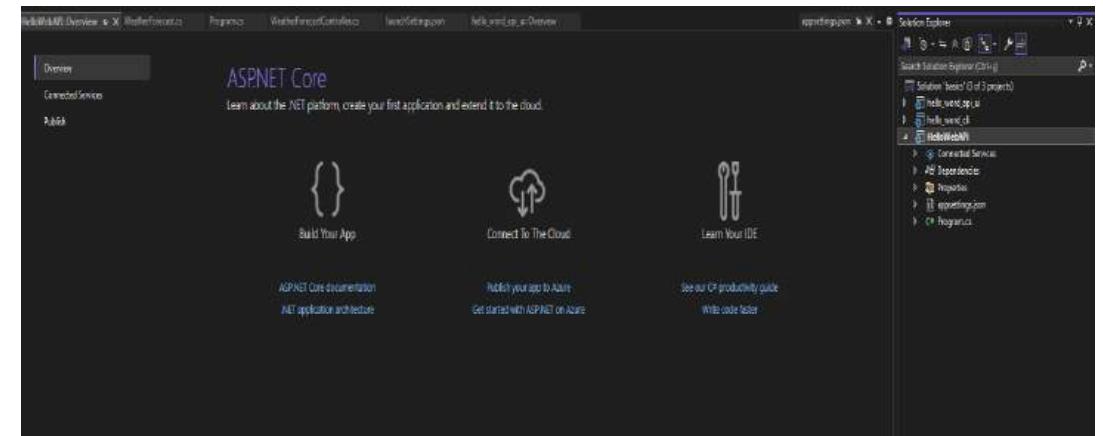


# İkinci Bölüm?

Proje Şablonun Uygulanması  
Yeni proje



Proje Adı: HelloWebAPI  
Proje ismi nasıl koyu oldu ?



# İkinci Bölüm?

Proje Şablonun Uygulanması  
Yeni proje

A screenshot of the Visual Studio IDE. The title bar shows "HelloWebAPI" and the menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extension, Window, Help, and Search. The Solution Explorer on the right shows the project structure: Solution basis (3 projekti), HelloWorldAPI, HelloWorldAPI, Connected Services, Dependencies, Properties, appsettings.json, and Program.cs. The code editor window displays the following C# code:

```
1 var builder = WebApplication.CreateBuilder(args);
2 var app = builder.Build();
3
4 //TODO: REQUEST olduğunda çalışacak middleware (Dönen RESPONSE)
5 //Kapat çalıştır ne olsak gör!
6 app.MapGet("/", () => "Hello World!");
7
8 app.Run();
9
```

Neden Hata Verdi ?

A screenshot of the Visual Studio IDE showing the same project structure and code as the first screenshot. However, the code editor now highlights line 14 with a red squiggle under "builder.Services.AddSwaggerGen();". A tooltip for the error CS1061: "IServiceCollection" does not contain a definition for "AddSwaggerGen" and no accessible extension method "AddSwaggerGen" accepting a first argument of type "IServiceCollection" could be found (are you missing a using directive or an assembly reference?) is displayed. The code editor window displays the following C# code:

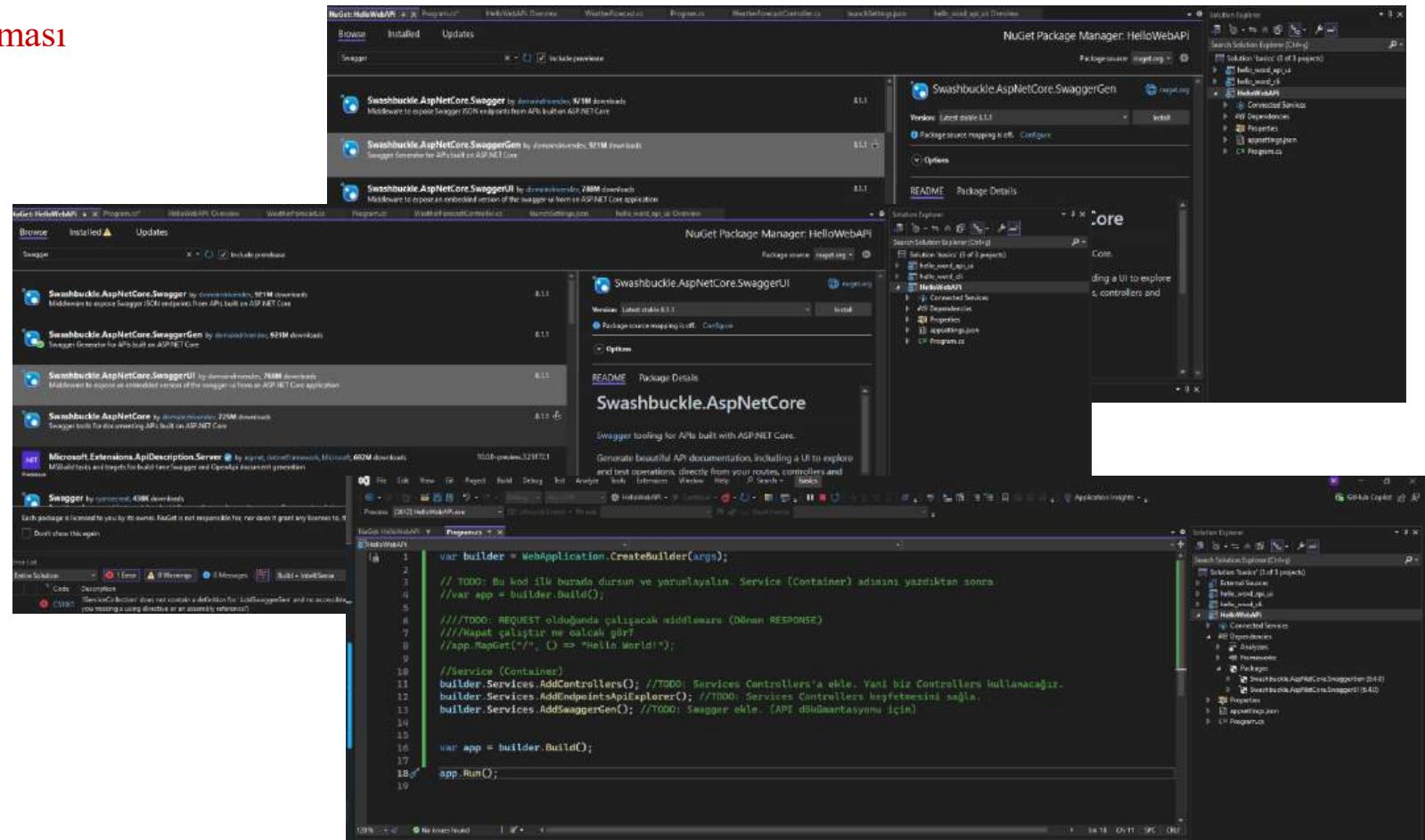
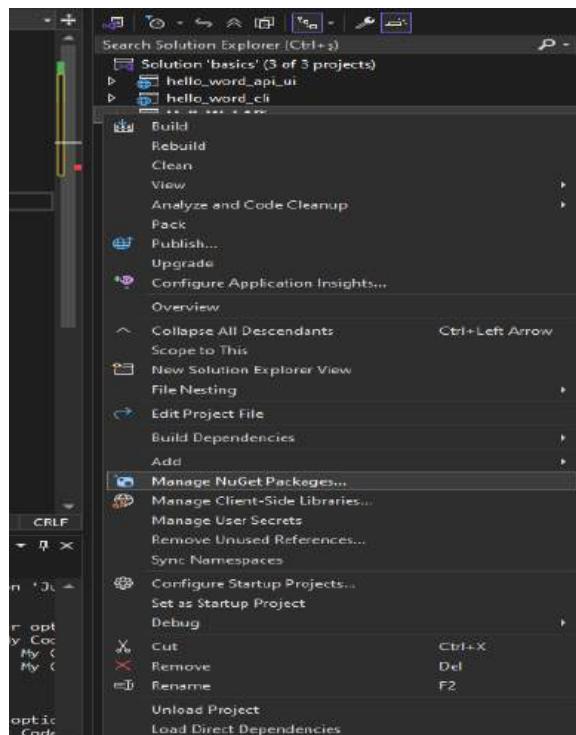
```
1 var builder = WebApplication.CreateBuilder(args);
2 var app = builder.Build();
3
4 //TODO: REQUEST olduğunda çalışacak middleware (Dönen RESPONSE)
5 //Kapat çalıştır ne olsak gör!
6 app.MapGet("/", () => "Hello World!");
7
8 //Service (Container)
9 builder.Services.AddControllers(); //TODO: Services Controllers'a ekle. Yani biz Controllers kullanacağız.
10 builder.Services.AddEndpointsApiExplorer(); //TODO: Services Controllers kesfetmesini sağla.
11 builder.Services.AddSwaggerGen(); //TODO: Swagger ekle. (API dökümantasyonu için)
12
13 app.Run();
14
```

CS1061: "IServiceCollection" does not contain a definition for "AddSwaggerGen" and no accessible extension method "AddSwaggerGen" accepting a first argument of type "IServiceCollection" could be found (are you missing a using directive or an assembly reference?)

# İkinci Bölüm?

## Proje Şablonun Uygulanması

### Yeni proje

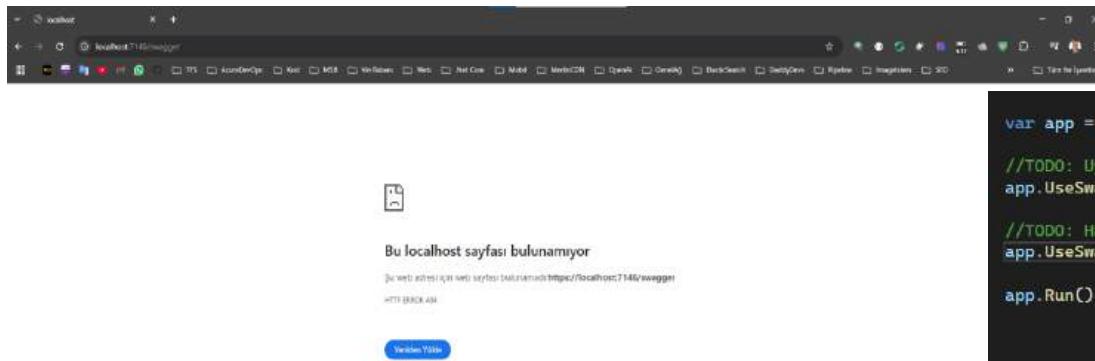


Gizlilik Derecesi [Tasnif Dışı](#)

# İkinci Bölüm?

Proje Şablonun Uygulanması  
Yeni proje

## Hatayı hatayı giderelim !!!



```
var app = builder.Build();

//TODO: Uygulamaya Swagger kullanmasını söylemedik. Ondan hata aldık.
app.UseSwagger();

//TODO: Harici olarak SwaggerUI kullan demedik.
app.UseSwaggerUI();

app.Run();
```

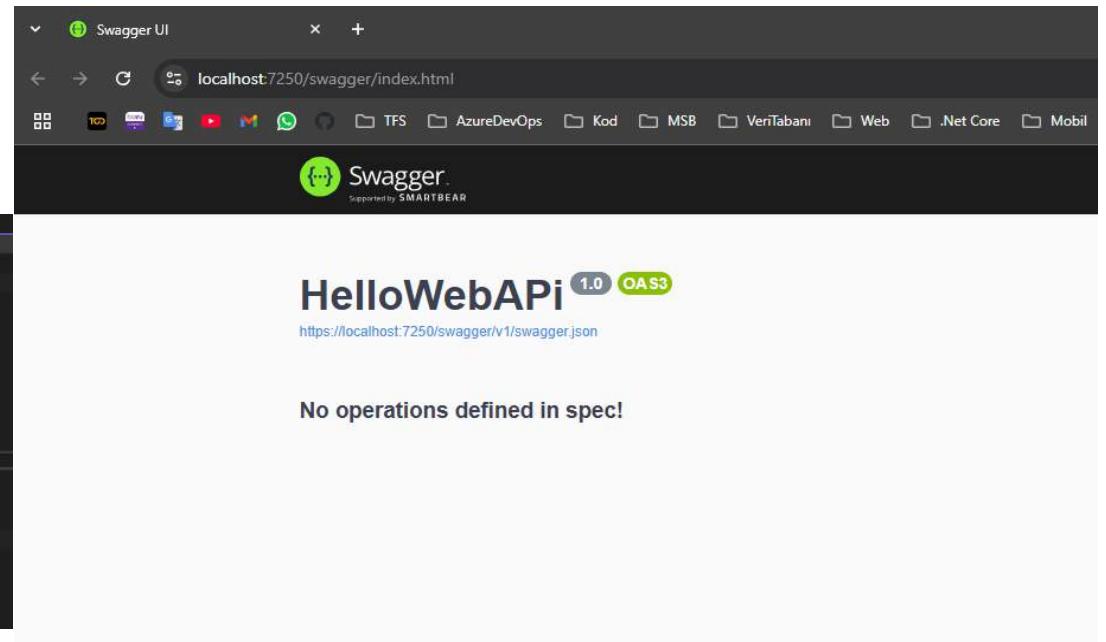
Uygulamayı çalıştır?

<https://localhost:7250/Swagger/index.html> (Neden Swagger yazıyoruz artık yazma direk açılsın? ) !!!

# İkinci Bölüm?

Proje Şablonun Uygulanması  
Yeni proje

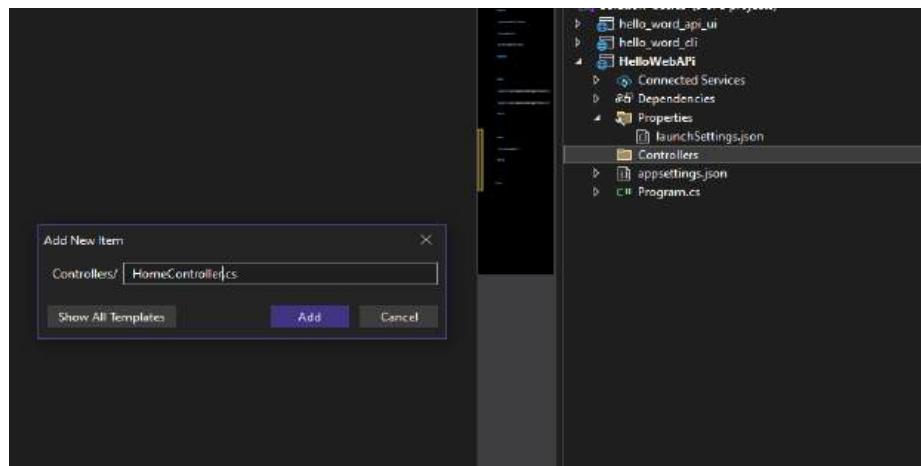
```
Program.cs # <--> launchSettings.json
Schema: https://json-schema.org/launchsettings.json
1  {
2    "iisSettings": {},
3    "profiles": {
4      "HelloWebAPI": {
5        "commandName": "Project",
6        "dotnetRunMessages": true,
7        "launchBrowser": true,
8        //TODO: Uygulama Swagger direkt gelmesi aşağıda kodu ekliyoruz.
9        "launchUrl": "swagger",
10       "applicationUrl": "https://localhost:7250;http://localhost:5212",
11       "environmentVariables": {
12         "ASPNETCORE_ENVIRONMENT": "Development"
13       }
14     },
15     "IIS Express": {}
16   }
17 }
```



Hiç Controller olmadığı için “**No operations defined in spec!**” uyarısı geldi.  
*Haydi Controller oluşturmaya geçelim ?*

# İkinci Bölüm?

Proje Şablonun Uygulanması  
Yeni proje



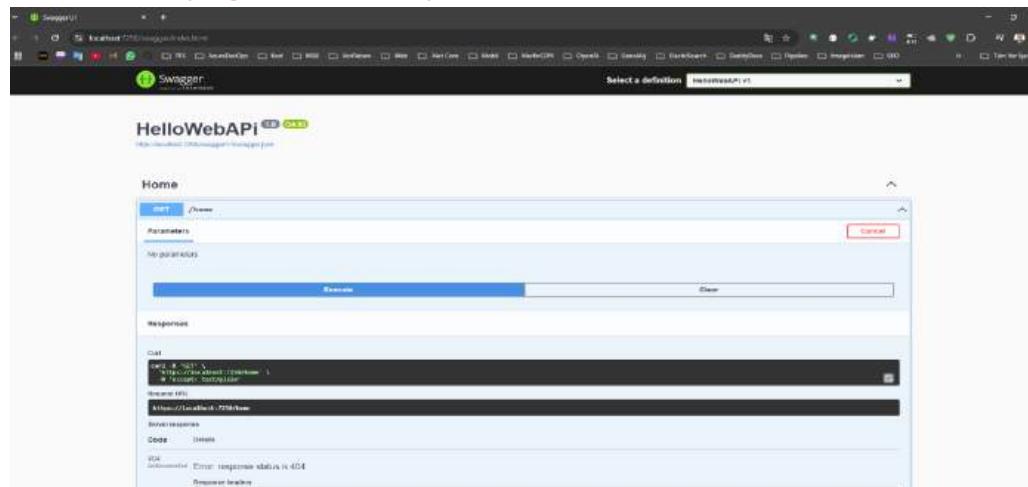
```
using Microsoft.AspNetCore.Mvc;
namespace HelloWebAPI.Controllers
{
    //TODO: Controller özellikleri kazanması için ControllerBase kalıtım ekle.
    //TODO: HomeController API yapısını kazanması için ApiController ekle.
    [ApiController]
    //TODO: HomeController içindeki eylemler için route ekle.
    [Route("home")] //TODO: HomeController içindeki eylemlere home ile erişebileceğiz.
    public class HomeController : ControllerBase
    {
        //TODO: Client için bir GET isteği oluşturalım. İlk API'mizi oluşturalım
        [HttpGet]
        public String GetMessage()
        {
            return "Hello ASP .NET Core Web API"; //TODO: Client'a dönenek mesajı belirttik.
        }
    }
}
```

# İkinci Bölüm?

Proje Şablonun Uygulanması

Yeni proje

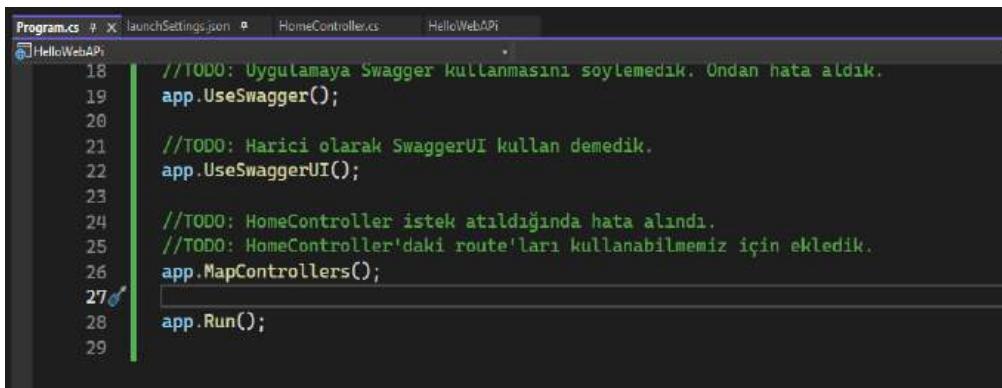
Uygulamayı çalıştır !!!



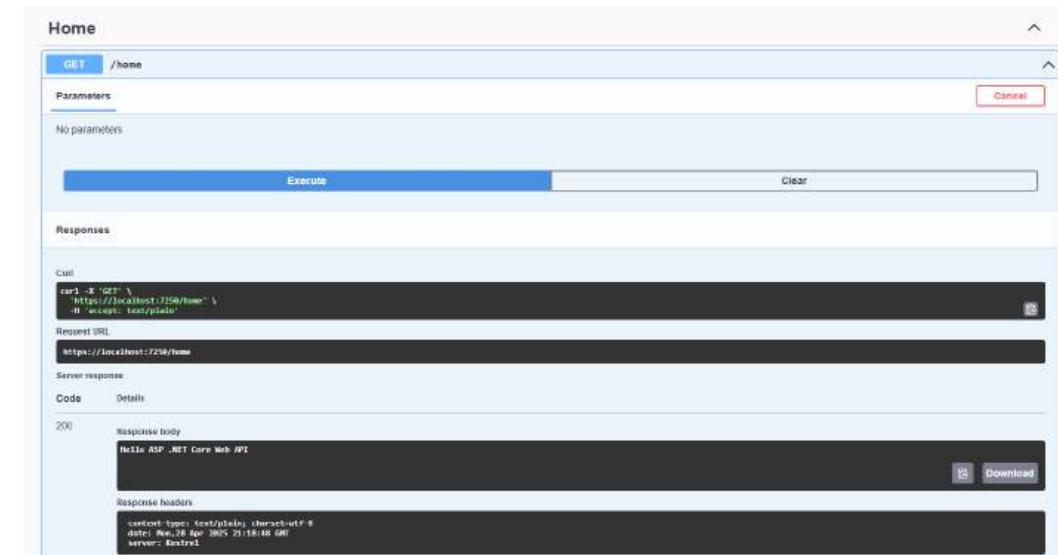
Hata alındı. Çözüm ne olabilir ?

# İkinci Bölüm?

Proje Şablonun Uygulanması  
Yeni proje



```
Program.cs
18 //TODO: Uygulamaya Swagger kullanmasını söylemedik. Ündan hata aldık.
19 app.UseSwagger();
20
21 //TODO: Harici olarak SwaggerUI kullan demedik.
22 app.UseSwaggerUI();
23
24 //TODO: HomeController istek atıldığından hata alındı.
25 //TODO: HomeController'daki route'ları kullanabilmemiz için ekledik.
26 app.MapControllers();
27
28 app.Run();
29
```



The screenshot shows the Swagger UI interface for a .NET Core Web API. The top navigation bar shows "Home" with a "GET /home" button. Below it, the "Parameters" section indicates "No parameters". In the "Responses" section, there is a "curl" example command:

```
curl -X 'GET' \
  'https://localhost:7254/home' \
  -H 'accept: text/plain'
```

The "Request URL" is listed as <https://localhost:7254/home>. The "Server response" shows a 200 status code with the following details:

**Response body**  
Hello ASP .NET Core Web API

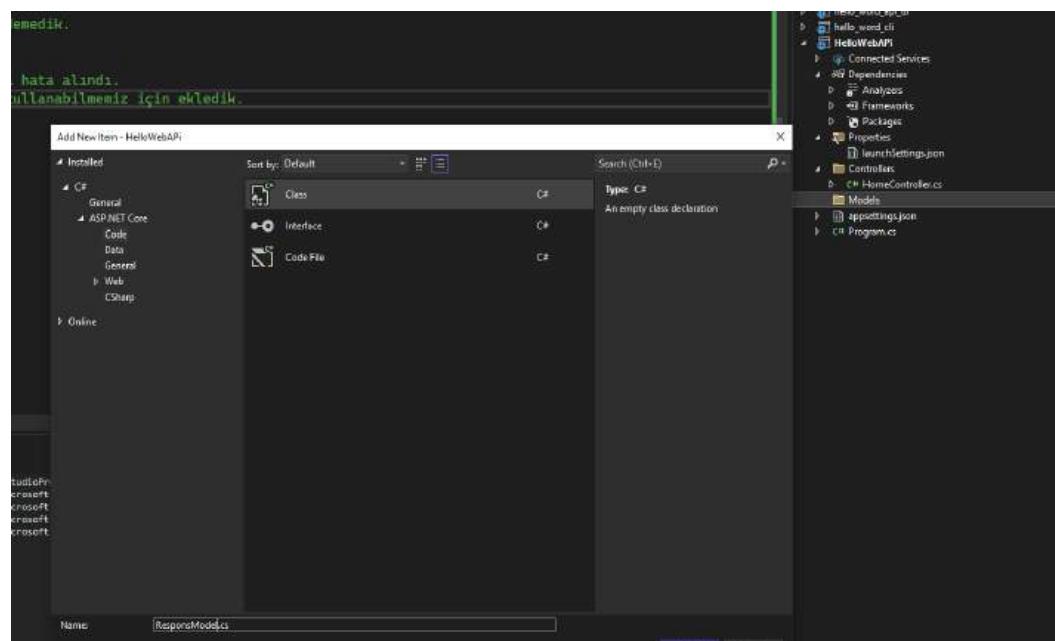
**Response headers**  
content-type: text/plain; charset=utf-8  
date: Mon, 28 Apr 2025 21:18:48 GMT  
server: Kestrel

# İkinci Bölüm?

Proje Şablonun Uygulanması

Yeni proje

*HAYDİ Model Yapalım ?*

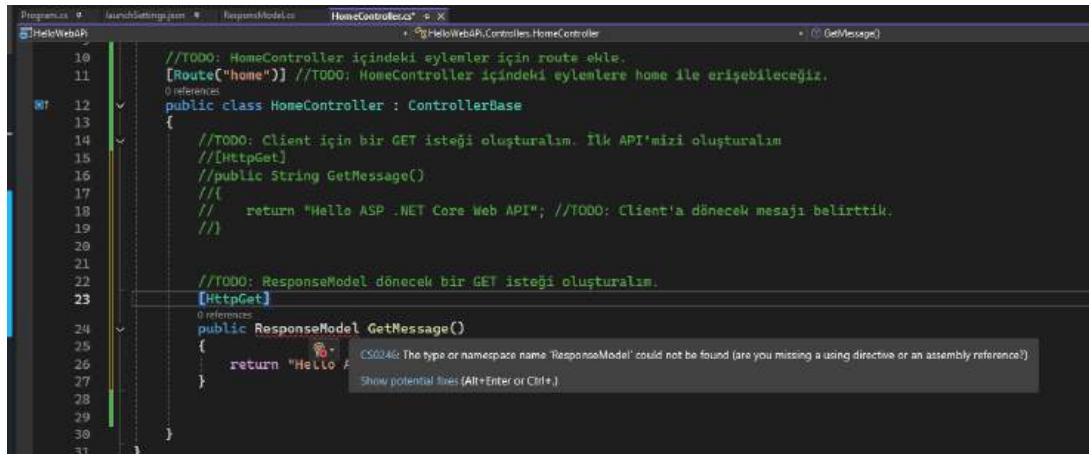


```
namespace HelloWebAPI.Models
{
    public class ResponsModel
    {
        //TODO: prop + tab tuşu ile,
        public int HttpStatusCode { get; set; }
        public String Message { get; set; }
    }
}
```

The screenshot shows the 'ResponsModel.cs' file in the 'HelloWebAPI' project. The code defines a simple class with two properties: '(HttpStatusCode)' and 'Message'. A TODO comment is present above the first property. The code editor highlights the 'String' keyword in green.

# İkinci Bölüm?

Proje Şablonun Uygulanması  
Yeni proje



The screenshot shows the HomeController.cs file in Visual Studio. The code is as follows:

```
//TODO: HomeController içindeki eylemler için route ekle.  
[Route("home")] //TODO: HomeController içindeki eylemlere home ile erişebilceğiz.  
[ApiController]  
public class HomeController : ControllerBase  
{  
    //TODO: Client için bir GET isteği oluşturalım. İlk API'ımızı oluşturalım  
    //[HttpGet]  
    //public String GetMessage()  
    //{
        //    return "Hello ASP .NET Core Web API"; //TODO: Client'a dönenek mesajı belirttiğimiz  
    //}  
  
    //TODO: ResponseModel dönenek bir GET isteği oluşturalım.  
    [HttpGet]  
    public ResponseModel GetMessage()  
    {  
        return "Hello";  
    }  
}
```

A tooltip is visible at the bottom right of the code editor, indicating a CS0246 error: "The type or namespace name 'ResponseModel' could not be found (are you missing a using directive or an assembly reference?)".

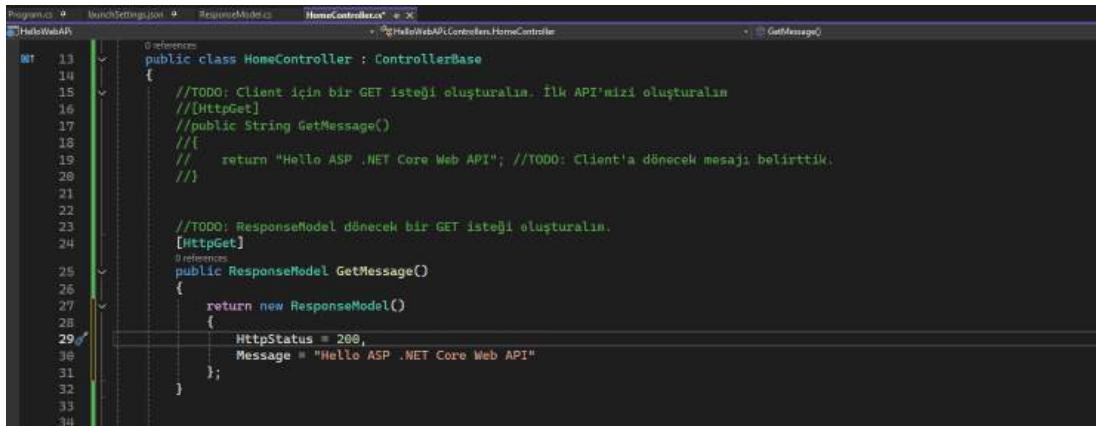
Hata aldık. Hadi çözelim.

using HelloWebAPi.Models; //TODO: ResponseModel sınıfını kullanabilmek için ekledik.

# İkinci Bölüm?

Proje Şablonun Uygulanması

Yeni proje



```
public class HomeController : ControllerBase
{
    //TODO: Client için bir GET isteği oluşturulmalı, ilk API'mizi oluşturalım
    //[HttpGet]
    //public String GetMessage()
    //{
    //    return "Hello ASP .NET Core Web API"; //TODO: Client'a dönecek mesajı belirttiğimizde
    //}

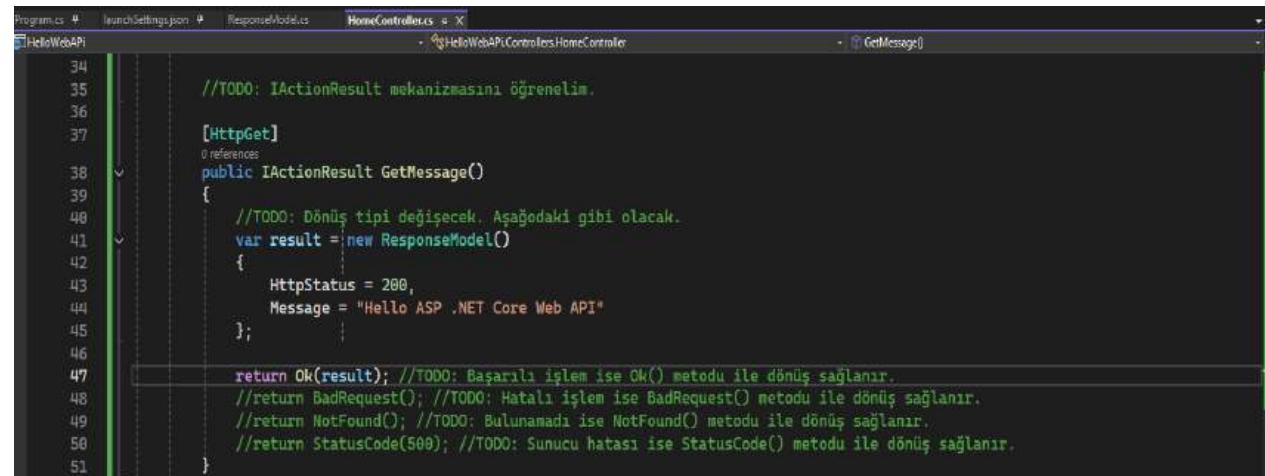
    //TODO: ResponseModel dönecek bir GET isteği oluşturulmalı.
    [HttpGet]
    public ResponseModel GetMessage()
    {
        return new ResponseModel()
        {
            HttpStatus = 200,
            Message = "Hello ASP .NET Core Web API"
        };
    }
}
```

Uygulamayı çalıştır.

# İkinci Bölüm?

Proje Şablonun Uygulanması

IActionResult Öğrenme?



The screenshot shows a Microsoft Visual Studio code editor with the following code:

```
Program.cs  launchSettings.json  ResponseModel.cs  HomeController.cs
HelloWebAPI  HelloWebAPI\Controllers\HomeController.cs  GetMessage()
34
35     //TODO: IActionResult mekanizmasını öğrenelim.
36
37     [HttpGet]
38     public IActionResult GetMessage()
39     {
40         //TODO: Dönüş tipi değişecek. Aşağıdaki gibi olacak.
41         var result = new ResponseModel()
42         {
43             HttpStatusCode = 200,
44             Message = "Hello ASP .NET Core Web API"
45         };
46
47         return Ok(result); //TODO: Başarılı işlem ise Ok() metodu ile dönüş sağlanır.
48         //return BadRequest(); //TODO: Hatalı işlem ise BadRequest() metodu ile dönüş sağlanır.
49         //return NotFound(); //TODO: Bulunamadı ise NotFound() metodu ile dönüş sağlanır.
50         //return StatusCode(500); //TODO: Sunucu hatalı ise StatusCode() metodu ile dönüş sağlanır.
51     }

```

Uygulamayı çalıştır !!!

# İkinci Bölüm?

Proje Şablonun Uygulanması  
Ortam değişkenleri



İncelencek Url:

<https://learn.microsoft.com/en-us/aspnet/core/fundamentals/environments?view=aspnet-core-9.0>

# İkinci Bölüm?

Proje Şablonun Uygulanması  
Ortam değişkenleri

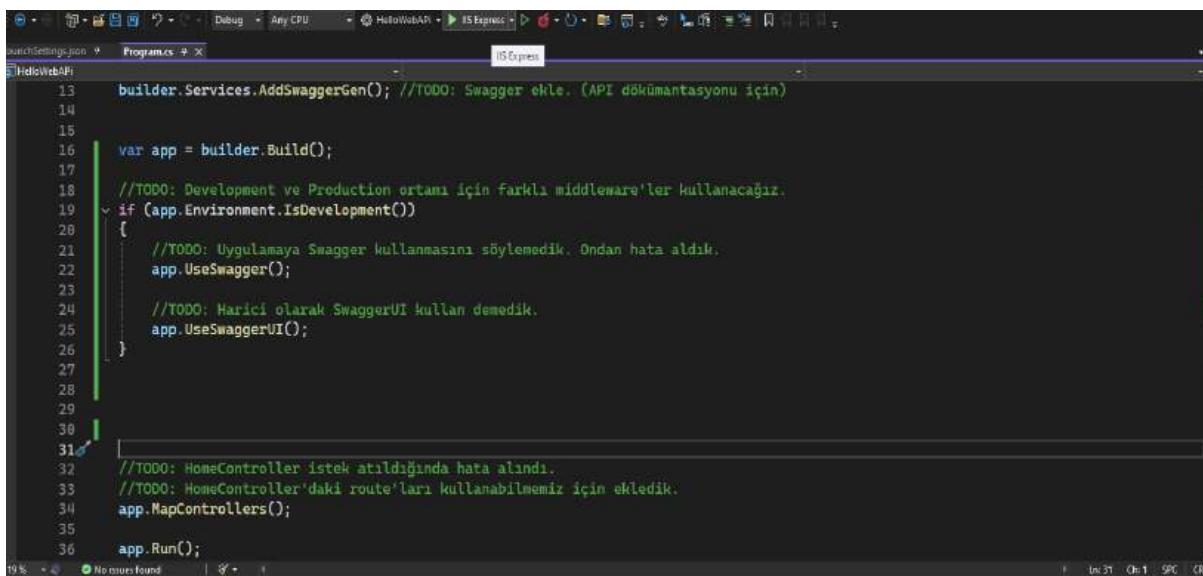
```
launchSettings.json + X
Schema: https://json.schemastore.org/launchsettings.json
14     "launchBrowser": true,
15     //TODO: Uygulama Swagger direkt gelmesi aşağıda kodu ekliyoruz.
16     "launchUrl": "swagger",
17     "applicationUrl": "https://localhost:7259;http://localhost:5212",
18     "environmentVariables": {
19         "ASPNETCORE_ENVIRONMENT": "Development"
20     }
21 },
22 "IIS Express": {
23     "commandName": "IISExpress",
24     "launchBrowser": true,
25     "environmentVariables": {
26         //TODO: IIS Express Production da çalışın. Nasıl anlayacağın Swagger gelmeyecek hata verecek
27         "ASPNETCORE_ENVIRONMENT": "Production"
28     }
29 }
30 }
31 }
32 }
```

```
launchSettings.json + Program.cs * X
HelloWebAPI
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
builder.Services.AddSwaggerGen(); //TODO: Swagger ekle. (API dökümantasyonu için)
var app = builder.Build();
//TODO: Development ve Production ortamı için farklı middleware'ler kullanacağız.
if (app.Environment.IsDevelopment())
{
    //TODO: Uygulamaya Swagger kullanmasını söylemedik. Ondan hata aldık.
    app.UseSwagger();
    //TODO: Harici olarak SwaggerUI kullanmadık.
    app.UseSwaggerUI();
}
```

# İkinci Bölüm?

Proje Şablonun Uygulanması  
Ortam değişkenleri

Yukarıdaki işlemleri yap çalıştır. (IIS Express yap çalışır. Çünkü Production modda olan o)



A screenshot of the Visual Studio IDE showing the `Program.cs` file of a .NET Core application named `HelloWebAPI`. The code is as follows:

```
builder.Services.AddSwaggerGen(); //TODO: Swagger ekle. (API dökümantasyonu için)

var app = builder.Build();

//TODO: Development ve Production ortamı için farklı middleware'ler kullanacağız.
if (app.Environment.IsDevelopment())
{
    //TODO: Uygulamaya Swagger kullanmasını söylemedik. Ondan hata aldık.
    app.UseSwagger();

    //TODO: Harici olarak SwaggerUI kullanmadık.
    app.UseSwaggerUI();
}

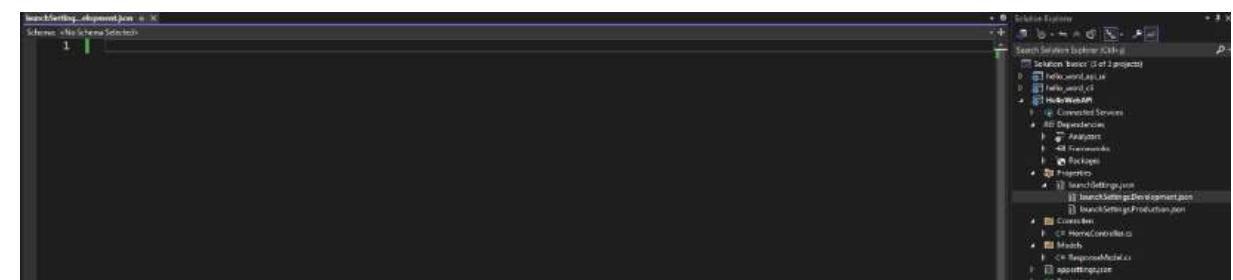
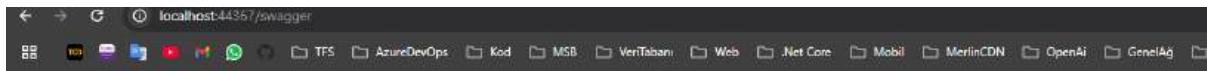
//TODO: HomeController istek atıldığından hata alındı.
//TODO: HomeController'daki route'ları kullanabilmemiz için ekledik.
app.MapControllers();

app.Run();
```

# İkinci Bölüm?

Proje Şablonun Uygulanması  
Ortam değişkenleri

Sonuç Hata Neden?  
IsDevelopment ise swagger kullan dedik,



**İkinci Bölüm Bitti**  
**Hadi Özetleyelim?**

# Üçüncü Bölüm?

## LOGGING

**Logging**, yazılım geliştirirken uygulamanın çalışması sırasında olan olayları, hataları, uyarıları veya bilgi mesajlarını bir yere kayıt etme işlemidir.

**Basit bir tanımla:**

Logging, uygulamanın içinden dışarıya yazılmış bir günlüktür.

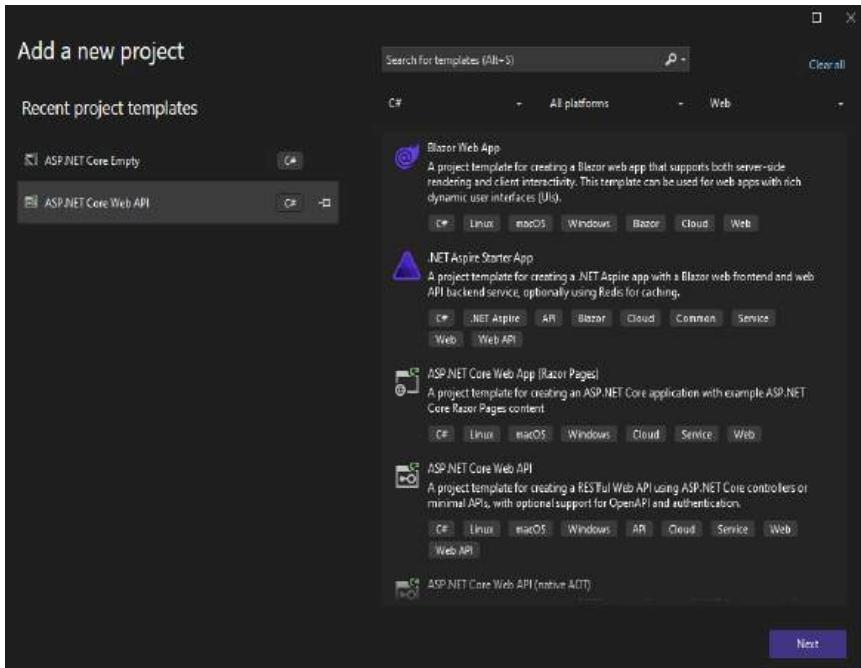
Bu günlükler sayesinde uygulamanın:

- Hangi işlemleri yaptığı,
- Ne zaman hata verdiğini,
- Hataların ya da problemlerinin neden olduğunu,
- Performans sorunlarını anlayabilirsın.

# Üçüncü Bölüm?

## LOGGING

Proje oluştur ve adını ProductApp koy !!!!



WeatherForecastController  
ve  
WeatherForecast sil!  
Sıfırdan proje olsun.

# Üçüncü Bölüm?

LOGGING

ProductApp Proje

Models klasörünü oluştur.

Altına Product classını ekle ve resimdeki kodu yaz.

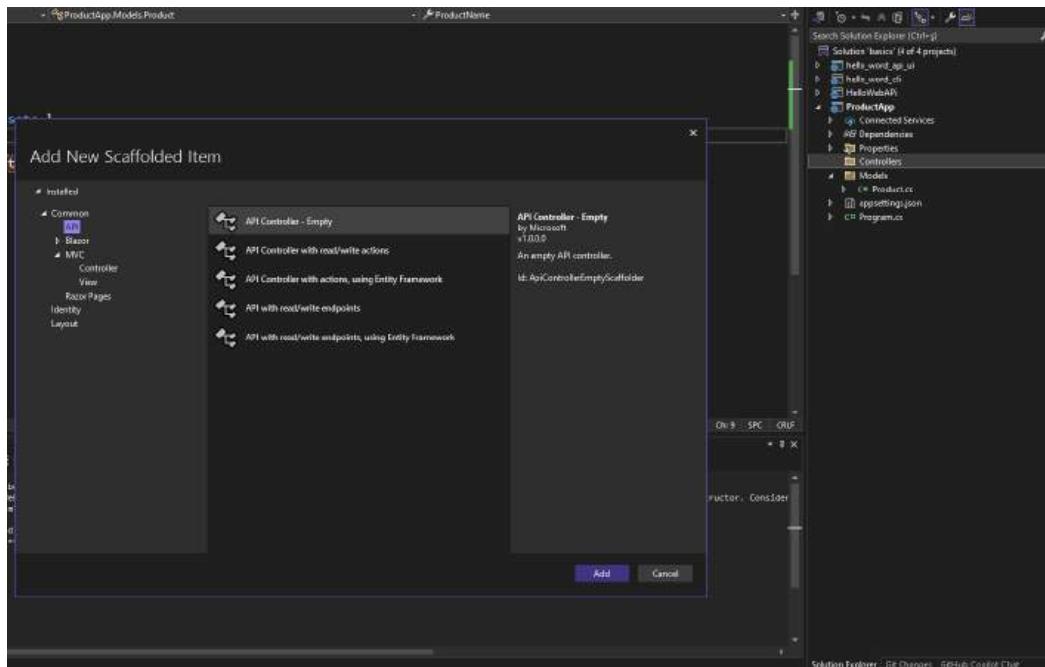
The screenshot shows the Visual Studio IDE interface. On the left, the Solution Explorer pane displays the project structure: HelloWorld, HelloWebAPI, HelloWebUI, and ProductApp. The ProductApp project is expanded, showing its subfolders: Connected Services, Dependencies, Properties, Resources, and Models. Inside the Models folder, there is a file named Product.cs. On the right, the code editor window shows the following C# code:

```
Product.cs
1  namespace ProductApp.Models
2  {
3      public class Product
4      {
5          public int Id { get; set; }
6
7          public string ProductName { get; set; }
8
9      }
10 }
11 
```

# Üçüncü Bölüm?

LOGGING

ProductApp Proje

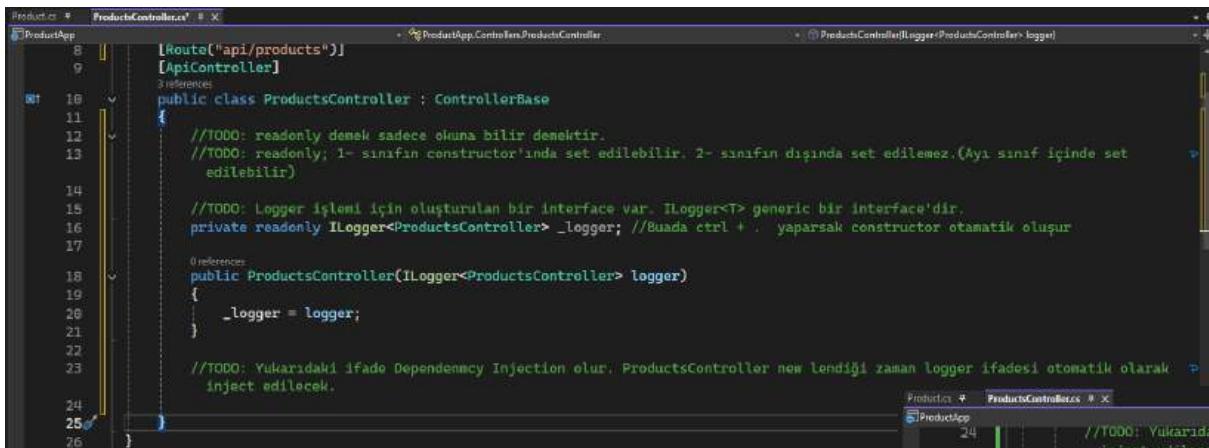


ProductsController adında Controller oluştur.  
ancak API nin altında olmasını dikkat et.

# Üçüncü Bölüm?

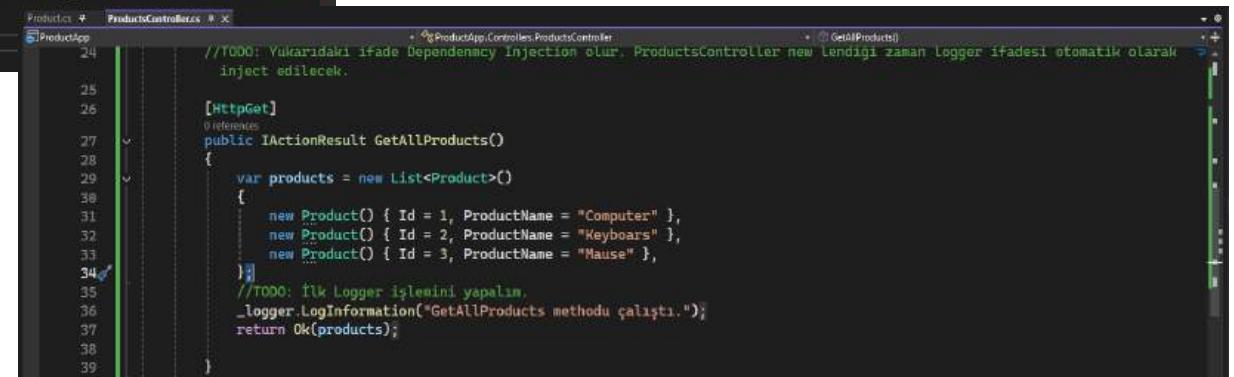
## LOGGING

### ProductApp Proje



```
ProductApp + ProductsController.cs # X
ProductApp
8  || [Route("api/products")]
9  || [ApiController]
10 || References
11 || public class ProductsController : ControllerBase
12 || {
13 ||     //TODO: readonly demek sadece okuma bilir demektir.
14 ||     //TODO: readonly; 1- sınıfın constructor'ında set edilebilir. 2- sınıfın dışında set edilemez.(Aynı sınıf içinde set edilebilir)
15 ||
16 ||     //TODO: Logger işlemi için oluşturulan bir interface var. ILogger<T> generic bir interface'dır.
17 ||     private readonly ILogger<ProductsController> _logger; //Buanda ctrl + . yaparsak constructor otomatik olusur
18 |
19 |     0 references
20 |     public ProductsController(ILogger<ProductsController> logger)
21 |     {
22 |         _logger = logger;
23 |
24 |         //TODO: Yukarıdaki ifade Dependency Injection olur. ProductsController new lendiği zaman logger ifadesi otomatik olarak inject edilecek.
25 |     }
26 }
```

Kodu yaz çalıştır  
İlk logu gör !!!!



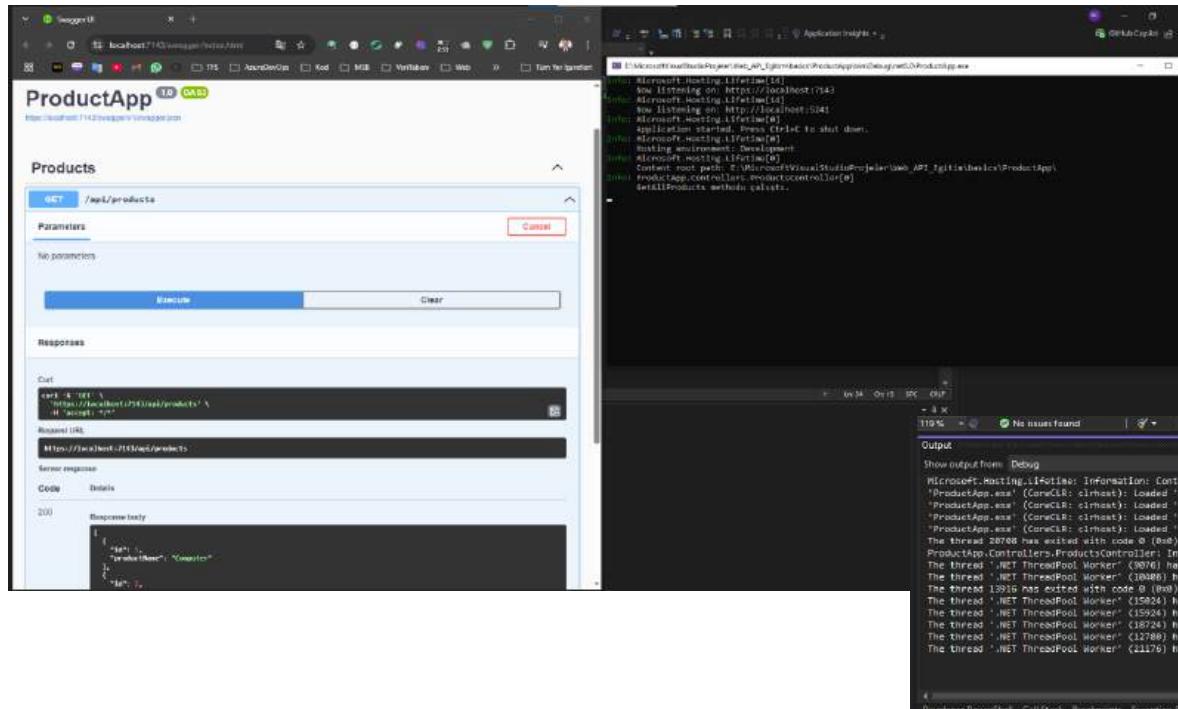
```
ProductApp + ProductsController.cs # X
ProductApp
24     //TODO: Yukarıdaki ifade Dependency Injection olur. ProductsController new lendiği zaman logger ifadesi otomatik olarak
25     //inject edilecek.
26
27     [HttpGet]
28     0 references
29     public IActionResult GetAllProducts()
30     {
31         var products = new List<Product>()
32         {
33             new Product() { Id = 1, ProductName = "Computer" },
34             new Product() { Id = 2, ProductName = "Keyboards" },
35             new Product() { Id = 3, ProductName = "Mause" },
36         };
37         //TODO: İlk Logger işlemini yapalım.
38         _logger.LogInformation("GetAllProducts methodu çalıştı.");
39     }

```

# Üçüncü Bölüm?

## LOGGING

### ProductApp Proje



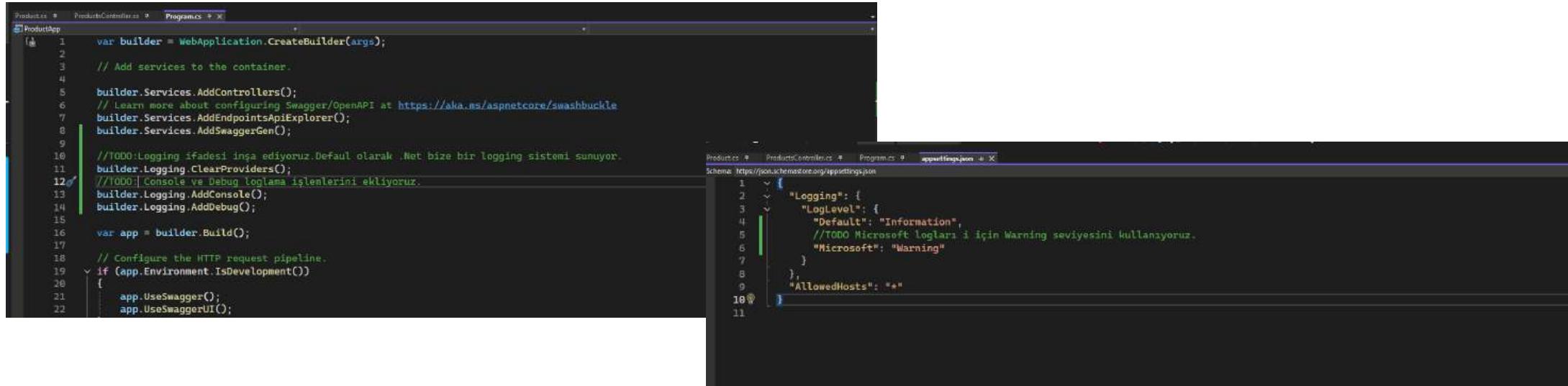
Output da da logu görebiliriz.

Bu Logu istediğimiz yere yazabiliriz.

# Üçüncü Bölüm?

## LOGGING

ProductApp Proje / Logu yapılandırıyalım !!!



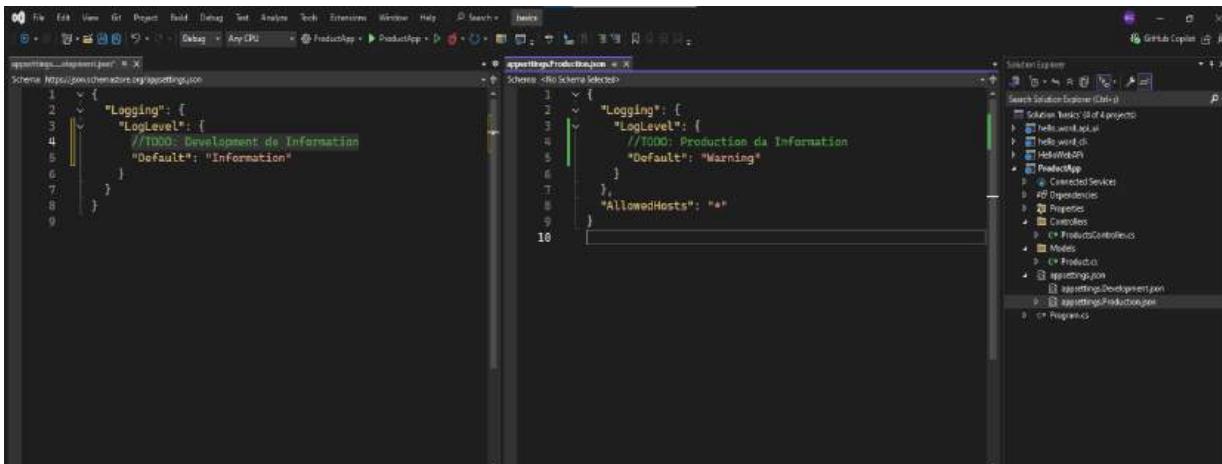
The screenshot shows two code editors side-by-side. The left editor contains `ProductController.cs` with the following code:`1 var builder = WebApplication.CreateBuilder(args);
2
3 // Add services to the container.
4
5 builder.Services.AddControllers();
6 // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
7 builder.Services.AddEndpointsApiExplorer();
8 builder.Services.AddSwaggerGen();
9
10 //TODO:Logging ifadesi inşa ediyoruz.Default olarak .Net bize bir logging sistemi sunuyor.
11 builder.Logging.ClearProviders();
12 //TODO: Console ve Debug loglama işlenelerini ekliyoruz.
13 builder.Logging.AddConsole();
14 builder.Logging.AddDebug();
15
16 var app = builder.Build();
17
18 // Configure the HTTP request pipeline.
19 if (app.Environment.IsDevelopment())
20 {
21 app.UseSwagger();
22 app.UseSwaggerUI();`The right editor contains `appsettings.json` with the following configuration:`1 {
2 "Logging": {
3 "LogLevel": {
4 "Default": "Information",
5 //TODO Microsoft logları i için Warning seviyesini kullanıyoruz.
6 "Microsoft": "Warning"
7 }
8 },
9 "AllowedHosts": "*"
10 }`

Uygulamayı çalışır ve Console daki Logları gör.  
Microsoft loglarını Warning seviyesine çekelim.

# Üçüncü Bölüm?

## LOGGING

ProductApp Proje / Loglama Ortam değişkenleri !!!



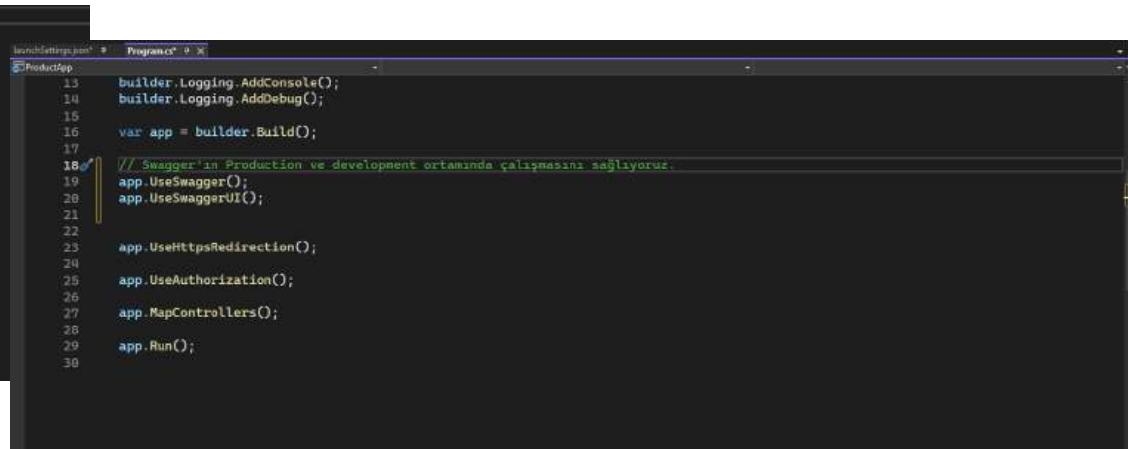
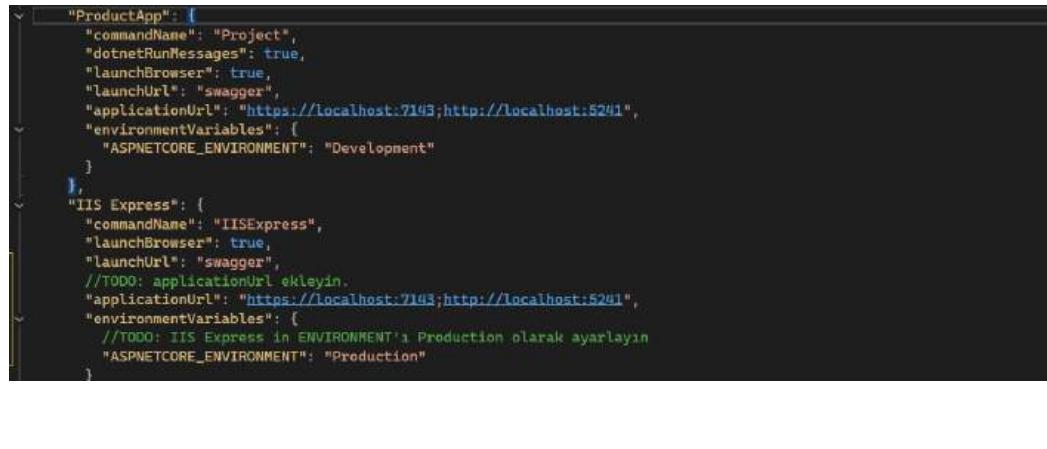
```
appsettings.json
{
  "Logging": {
    "LogLevel": {
      "Development": "Information",
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

appsettings.Production, appsettings.Development ekle.  
appsettings.Development de **Information** belirle  
appsettings.Production da **Warning** belirle

# Üçüncü Bölüm?

## LOGGING

ProductApp Proje / Loglama Ortam değişkenleri !!!



```
"ProductApp": {
    "commandName": "Project",
    "dotnetRunMessages": true,
    "launchBrowser": true,
    "launchUrl": "swagger",
    "applicationUrl": "https://localhost:7143;http://localhost:5241",
    "environmentVariables": [
        "ASPNETCORE_ENVIRONMENT": "Development"
    ],
    "IIS Express": {
        "commandName": "IISExpress",
        "launchBrowser": true,
        "launchUrl": "swagger",
        //TODO: applicationUrl ekleyin.
        "applicationUrl": "https://localhost:7143;http://localhost:5241",
        "environmentVariables": [
            //TODO: IIS Express in ENVIRONMENT'i Production olarak ayarlayın
            "ASPNETCORE_ENVIRONMENT": "Production"
        ]
    }
},
```

```
builder.Logging.AddConsole();
builder.Logging.AddDebug();
var app = builder.Build();

// Swagger'ın Production ve development ortamında çalışmasını sağlıyor.
app.UseSwagger();
app.UseSwaggerUI();

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

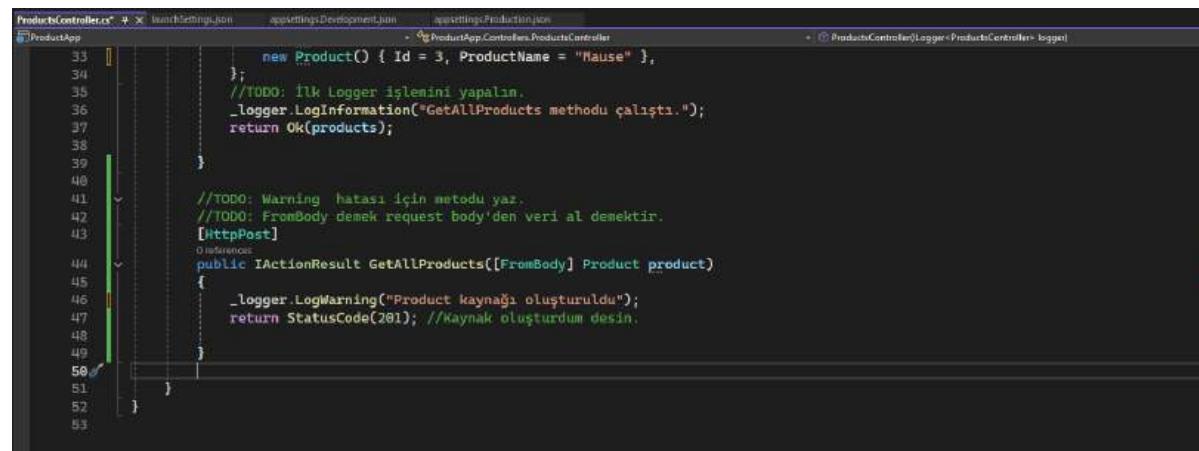
Şimdi uygulamayı ProductApp ve IIS Express de çalıştır.  
IIS Express de console açılmayacak. Output debug moduna bakalım log olmayacağı.  
Çünkü IIS Express Production da çalışıyor oda sadece Warning leri logluyor.

# Üçüncü Bölüm?

## LOGGING

ProductApp Proje / Loglama Ortam değişkenleri !!!

Şimdi Warning logu yaratalım !!!



The screenshot shows the Visual Studio IDE with the ProductsController.cs file open. The code is as follows:

```
ProductsController.cs
33     new Product() { Id = 3, ProductName = "Mause" },
34 };
35 //TODO: İlk Logger işlemini yapalım.
36 _logger.LogInformation("GetAllProducts methodu çalıştı.");
37 return Ok(products);
38 }
39
40
41 //TODO: Warning hatası için metodu yaz.
42 //TODO: FromBody demek request body'den veri al demektir.
43 [HttpPost]
44 public IActionResult GetAllProducts([FromBody] Product product)
45 {
46     _logger.LogWarning("Product kaynağı oluşturuldu");
47     return StatusCode(201); //Kaynak oluşturdu desin.
48 }
49
50 }
51
52 }
```

The code includes several TODO comments and logger statements for information, warning, and error levels.

# Üçüncü Bölüm?

## LOGGING

ProductApp Proje / Loglama Ortam değişkenleri !!!

The screenshot shows two windows side-by-side. On the left is the Visual Studio IDE with the ProductController.cs file open. The code contains several TODO comments for logging and warning handling. On the right is a screenshot of the Swagger UI interface, showing a POST request to /api/products. The request body is set to application/json with the following JSON content:

```
{"id": 1, "productName": "Mouse"}
```

The 'Responses' section shows the expected JSON response:

```
Content-Type: application/json
[{"id": 1, "productName": "Mouse"}]
```

Uygulamayı IIS Express de çalıştır.  
Outputu temizle. Warning hastasını gör.

Daha sonra ProductApp da çalıştır. Outputu temizle. Hataları gör.

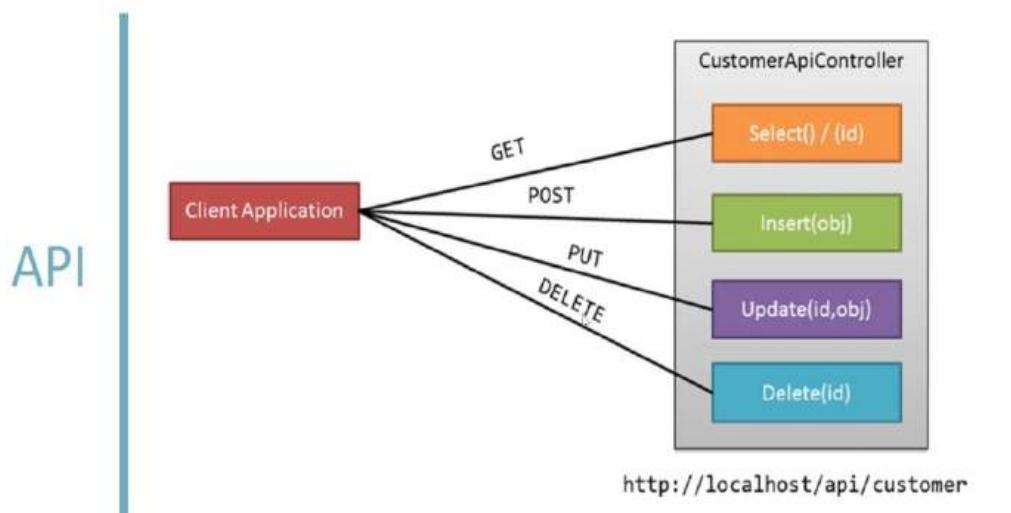
**Üçüncü Bölüm Bitti**  
**Hadi Özetleyelim?**

# Dördüncü Bölüm?

MODELLER İLE ÇALIŞMA

WEB API

Veriler ile Çalışma



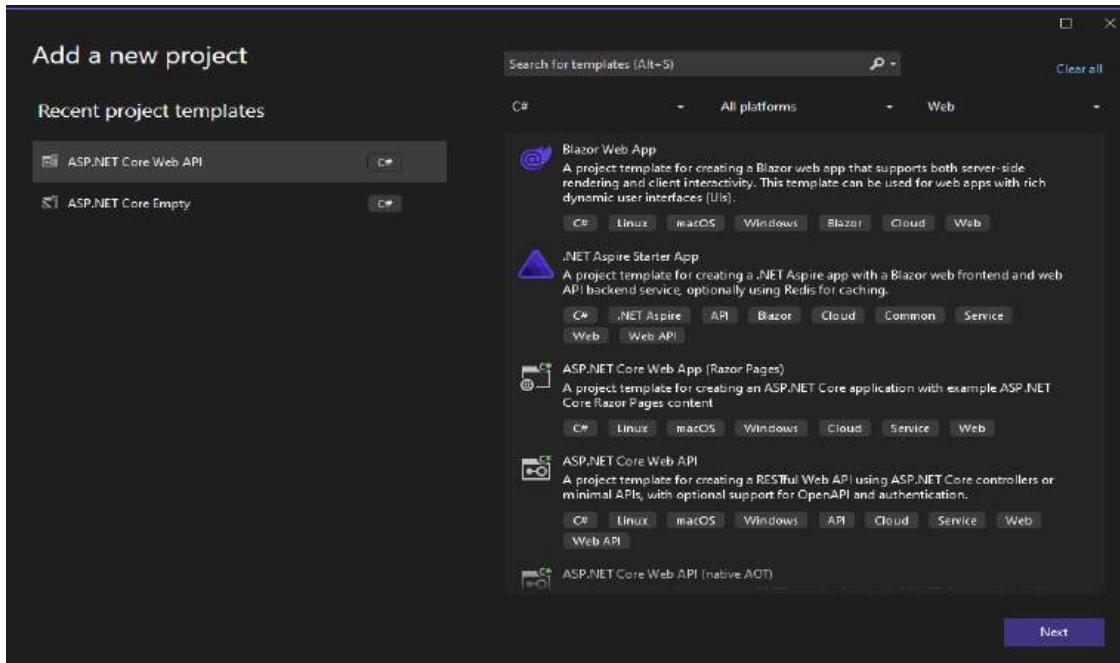
- Uygulama Oluşturma ve Veri Modelleme
- GET
- POST
- PUT
- DELETE
- PATCH

# Dördüncü Bölüm?

## MODELLER İLE ÇALIŞMA

Proje Oluşturma

Projenin adı bookDemo !!!

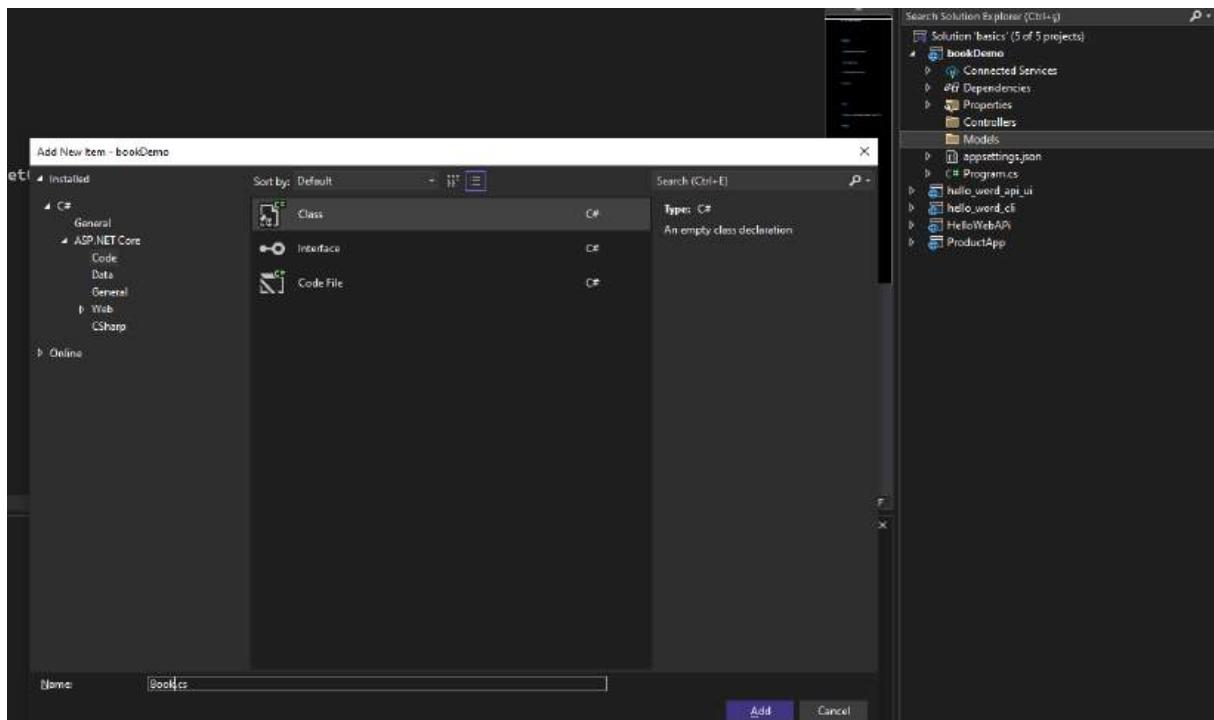


WeatherForecast.cs ve  
WeatherForecastController.cs sil

# Dördüncü Bölüm?

## MODELLER İLE ÇALIŞMA

### Proje Oluşturma



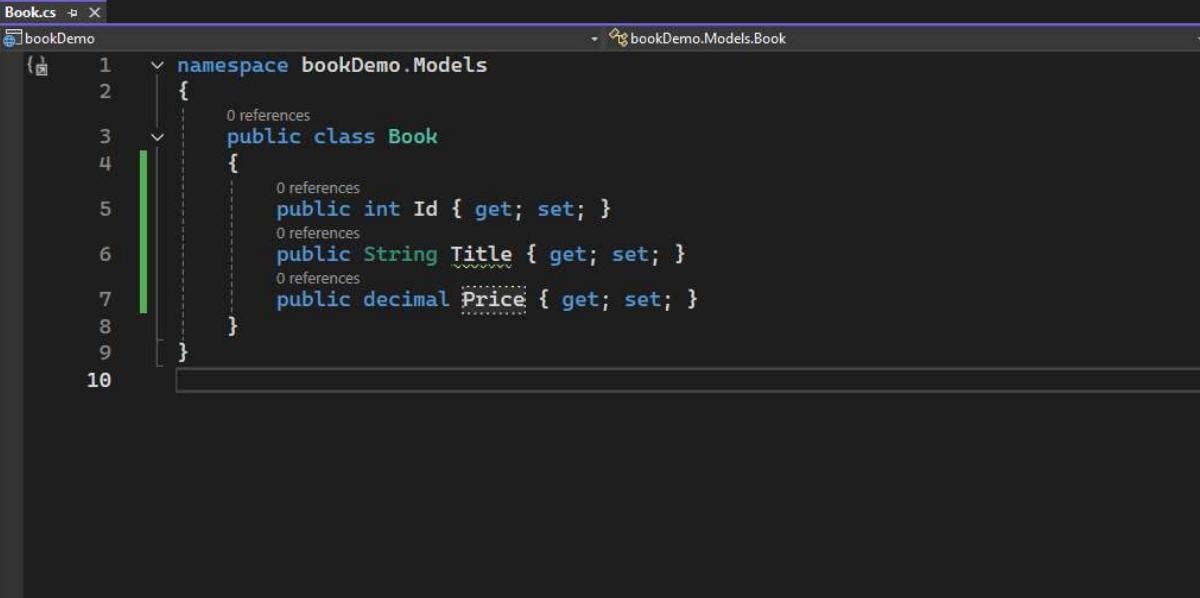
Şimdi **Models** klasörünü oluştur. İçine **Book Class**'ını oluştur !!!

# Dördüncü Bölüm?

MODELLER İLE ÇALIŞMA

Proje Oluşturma

Book Class'ını doldur. !!!

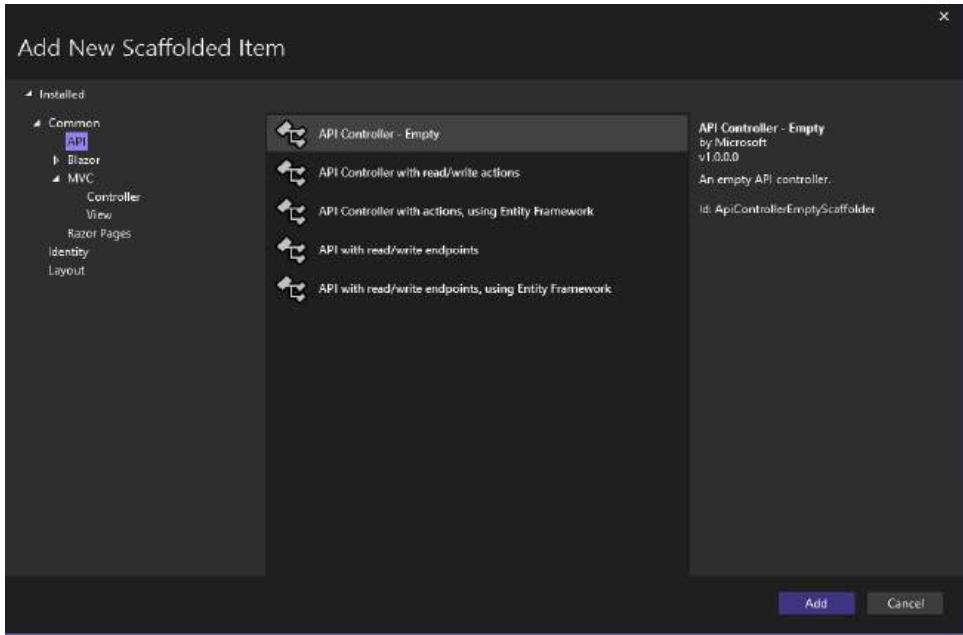


```
Book.cs 书
bookDemo          bookDemo.Models.Book
namespace bookDemo.Models
{
    public class Book
    {
        public int Id { get; set; }
        public String Title { get; set; }
        public decimal Price { get; set; }
    }
}
```

# Dördüncü Bölüm?

## MODELLER İLE ÇALIŞMA

### Proje Oluşturma

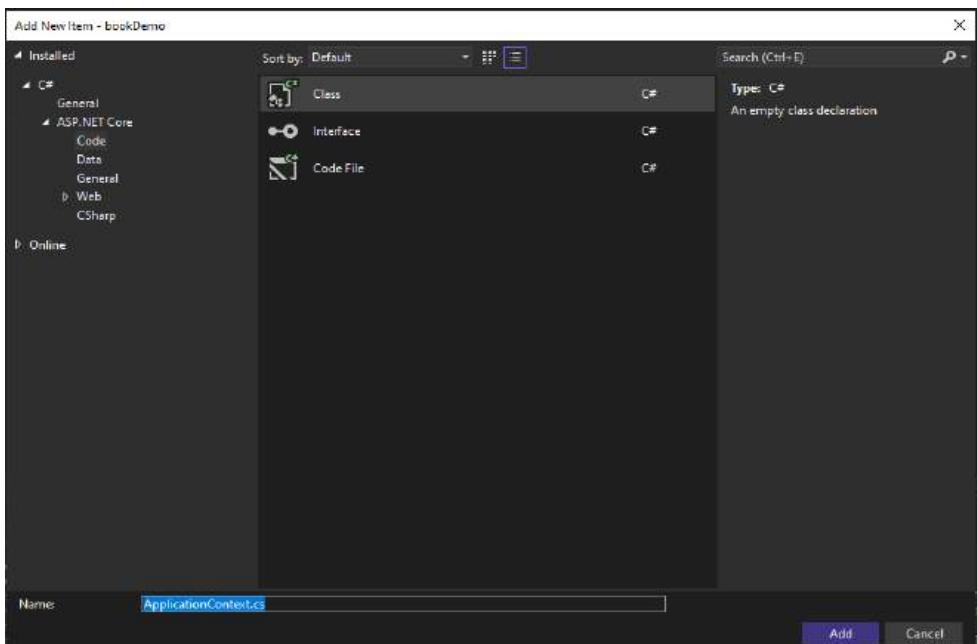


**BooksController.cs** adında Controller.cs oluştur. !!!! API olmasına dikkat et.  
Sonra uygulamayı çalıştır swagger’ı gör.

# Dördüncü Bölüm?

## MODELLER İLE ÇALIŞMA

### Proje Oluşturma



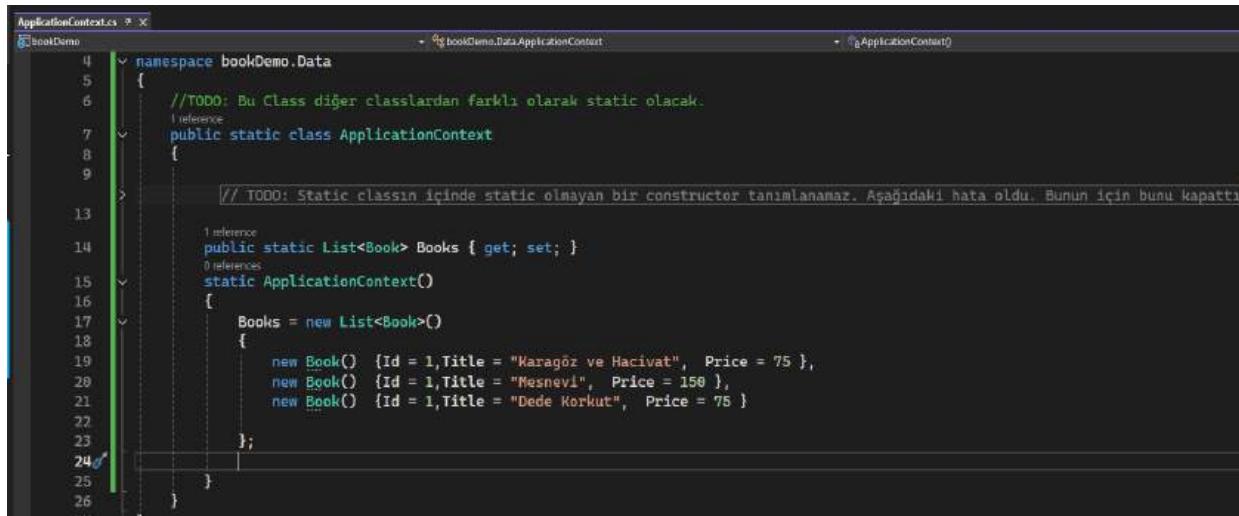
Veri katmanı için **Data** klasörünü oluştur için **ApplicationContext.cs** classını oluştur.

# Dördüncü Bölüm?

## MODELLER İLE ÇALIŞMA

### Proje Oluşturma

ApplicationContext.cs dosyasına ilgili kodu yaz.



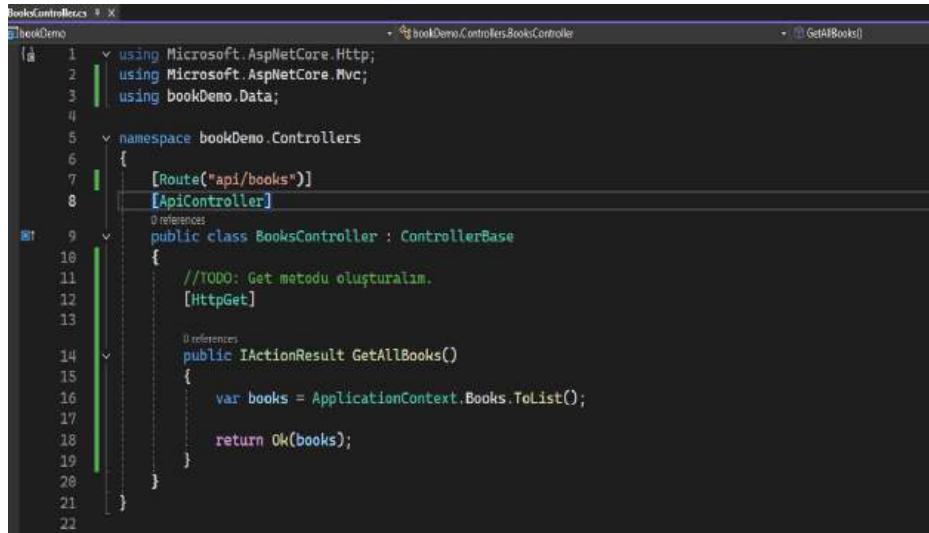
The screenshot shows a code editor window with the file 'ApplicationContext.cs' open. The code defines a static class 'ApplicationContext' within the namespace 'bookDemo.Data'. It contains a static list 'Books' and a constructor that initializes it with three book objects: 'Karagöz ve Hacivat', 'Mesnevi', and 'Dede Korkut', each with an ID of 1 and a price of 75 or 150.

```
ApplicationContext.cs
4  namespace bookDemo.Data
5  {
6      //TODO: Bu Class diğer classlardan farklı olarak static olacak.
7      public static class ApplicationContext
8      {
9          // TODO: Static classın içinde static olmayan bir constructor tanımlanamaz. Aşağıdaki hata oldu. Bunun için bunu kapattık
10         public static List<Book> Books { get; set; }
11         static ApplicationContext()
12         {
13             Books = new List<Book>()
14             {
15                 new Book() { Id = 1, Title = "Karagöz ve Hacivat", Price = 75 },
16                 new Book() { Id = 1, Title = "Mesnevi", Price = 150 },
17                 new Book() { Id = 1, Title = "Dede Korkut", Price = 75 }
18             };
19         }
20     }
21 }
```

# Dördüncü Bölüm?

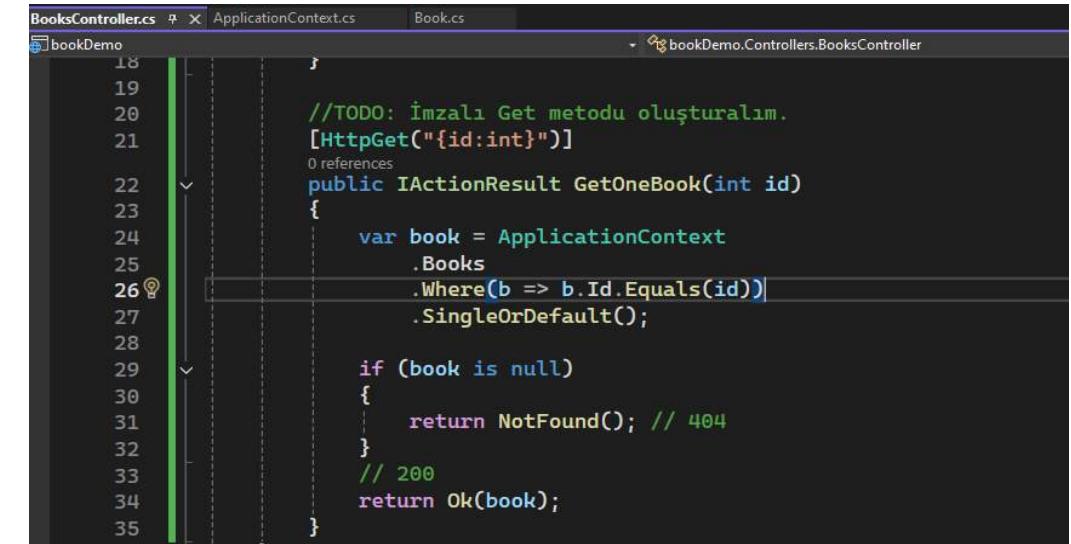
## MODELLER İLE ÇALIŞMA

### Proje Oluşturma



```
BooksController.cs
1  using Microsoft.AspNetCore.Http;
2  using Microsoft.AspNetCore.Mvc;
3  using bookDemo.Data;
4
5  namespace bookDemo.Controllers
6  {
7      [Route("api/books")]
8      [ApiController]
9      public class BooksController : ControllerBase
10     {
11         //TODO: Get metodu oluşturalım.
12         [HttpGet]
13
14         public IActionResult GetAllBooks()
15         {
16             var books = ApplicationContext.Books.ToList();
17
18             return Ok(books);
19         }
20     }
21 }
```

BooksController a **HttpGet GetAllBooks** metodunu yaz.



```
BooksController.cs
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

//TODO: imzalı Get metodu oluşturalım.
[HttpGet("{id:int}")]
public IActionResult GetOneBook(int id)
{
    var book = ApplicationContext
        .Books
        .Where(b => b.Id.Equals(id))
        .SingleOrDefault();

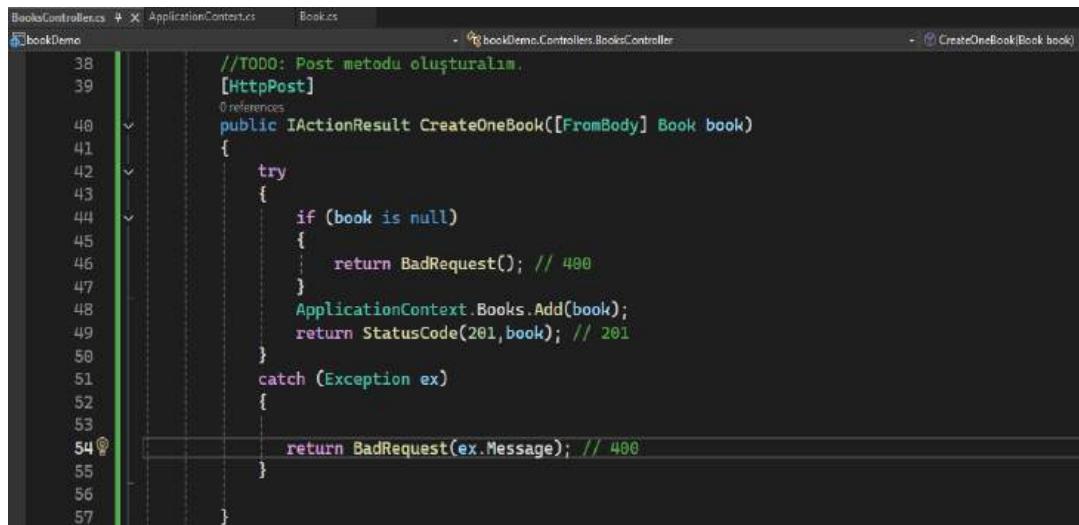
    if (book is null)
    {
        return NotFound(); // 404
    }
    // 200
    return Ok(book);
}
```

BooksController a **HttpGet GetOneBook** metodunu yaz.

# Dördüncü Bölüm?

## MODELLER İLE ÇALIŞMA

### Proje Oluşturma



```
38 //TODO: Post metodu oluşturalım.
39 [HttpPost]
40 public IActionResult CreateOneBook([FromBody] Book book)
41 {
42     try
43     {
44         if (book is null)
45         {
46             return BadRequest(); // 400
47         }
48         ApplicationContext.Books.Add(book);
49         return StatusCode(201, book); // 201
50     }
51     catch (Exception ex)
52     {
53
54         return BadRequest(ex.Message); // 400
55     }
56 }
57 }
```

BooksController a **HttpPost CreateOneBook** metodunu yaz.

Uygulama in memory çalıştığı için veriler uygulama kapatılınca silinecek.

Bunun için uygulamaya veri ekledikten sonra **GetAllBooks** metodunu çalıştır !!! Ekleneni gör !!!

**SONRA ÇALIŞTIR SONUCU GÖR !!!**

# Dördüncü Bölüm?

## MODELLER İLE ÇALIŞMA

### Proje Oluşturma

```
//TODO: Put metodu oluşturalım. (Güncelleme yapmak için)
[HttpPut("{id:int}")]
0 references
public IActionResult UpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] Book book)
{
    //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
    var bookToUpdate = ApplicationContext
        .Books
        .Find(b => b.Id.Equals(id));

    if (bookToUpdate == null)
        return NotFound(); // 404

    if (id != book.Id)
        return BadRequest(); // 400

    ApplicationContext.Books.Remove(bookToUpdate);
    bookToUpdate.Id = book.Id;
    bookToUpdate.Title = book.Title;
    bookToUpdate.Price = book.Price;

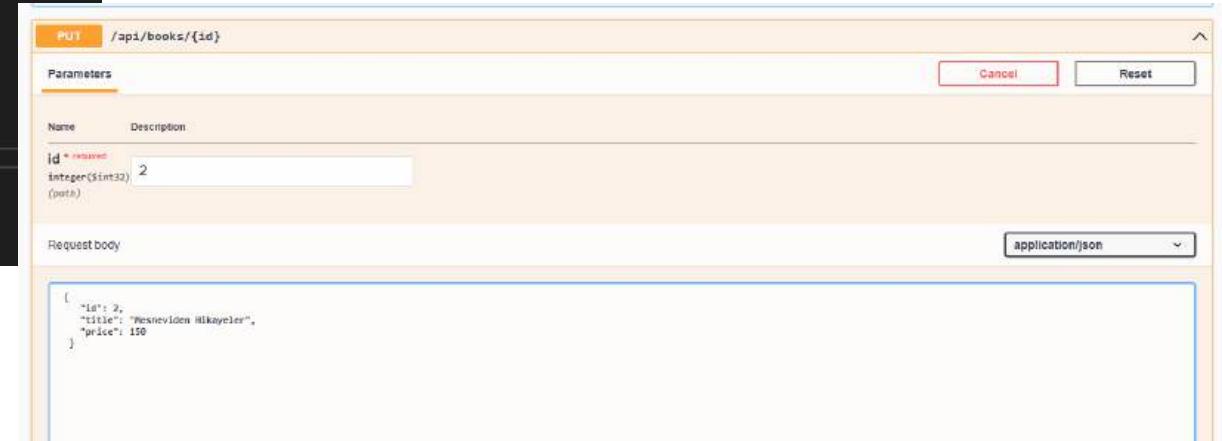
    ApplicationContext.Books.Add(bookToUpdate);

    return Ok(bookToUpdate); // 200
}
```

BooksController a **HttpPut UpdateOneBook** metodunu yaz.

Uygulama in memory çalıştığı için veriler uygulama kapatılınca silinecek.

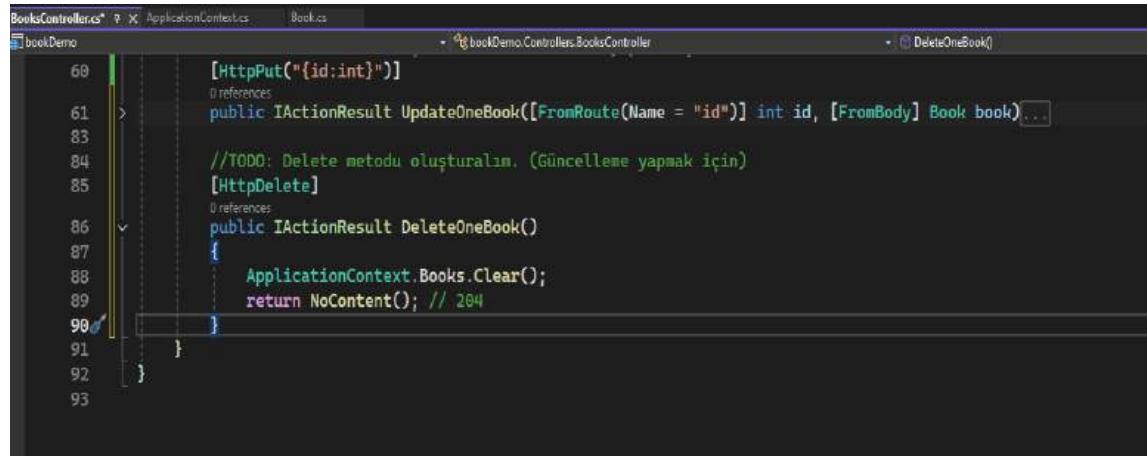
Bunun için uygulamaya veri ekledikten sonra **GetAllBooks** metodunu çalıştır !!! Ekleneni gör !!!



# Üçüncü Bölüm?

## MODELLER İLE ÇALIŞMA

### Proje Oluşturma



The screenshot shows the BooksController.cs file in Visual Studio. The code defines two actions: UpdateOneBook and DeleteOneBook. The UpdateOneBook action uses [HttpPut] and [FromRoute(Name = "id")]. The DeleteOneBook action uses [HttpDelete]. A TODO comment indicates that the Delete method needs to be implemented to handle updates.

```
BooksController.cs
[HttpPut("{id:int}")]
public IActionResult UpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] Book book)
{
    //TODO: Delete metodu oluşturalım. (Güncelleme yapmak için)
}

[HttpDelete]
public IActionResult DeleteOneBook()
{
    ApplicationContext.Books.Clear();
    return NoContent(); // 204
}
```

BooksController a **HttpDelete DeleteOneBook** metodunu yaz.

Sonra uygulamayı çalıştır. Swagger da Delete metodu gör.

İlk önce **listele** sonra **sil !!!**

Sildikten sonra tekrar **listele !!!**

# Dördüncü Bölüm?

## MODELLER İLE ÇALIŞMA

### Proje Oluşturma

```
public IActionResult DeleteOneBook([HttpDelete("{id:int}")] [FromRoute(Name = "id")] int id)
{
    var bookToDelete = ApplicationContext
        .Books
        .Find(b => b.Id.Equals(id));

    if (bookToDelete == null)
        return NotFound(new
        {
            StatusCode = 404,
            message = $"Silinecek kitap id: {id} bulunamadı."
        }); // 404

    ApplicationContext.Books.Remove(bookToDelete);
    return NoContent(); // 204
}
```

BooksController a **HttpDelete DeleteOneBook** metodunu yaz.

Sonra uygulamayı çalıştır. Swagger da Delete metodu gör.

İlk önce **listele** bir tane **Id** seç sonra o **Id'yi sil !!!**  
Sildikten sonra tekrar **listele !!!**

Daha sonra tüm kitapları sil. Sonra 1 numaralı kitabı silmeyi dene. Swagger üzerinde hatayı gör.

# Dördüncü Bölüm?

## MODELLER İLE ÇALIŞMA

### Proje Oluşturma

```
//TODO: Put metodu oluşturalım.  
[HttpPatch("{id:int}")]  
public IActionResult PartiallyUpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] JsonPatchDocument<Book>  
bookPatch)  
{  
    //TODO: Güncelleme yapmak için gelen book var mı kontrol et.  
    var bookToUpdate = ApplicationContext  
        .Books  
        .Find(b => b.Id.Equals(id));  
  
    //TODO: Güncelleme yapmak için gelen book null mı kontrol et.  
    if (bookToUpdate is null)  
        return NotFound(); // 404  
  
    bookPatch.ApplyTo(bookToUpdate);  
  
    return NoContent(); // 204  
}
```

BooksController a **HttpPatch**  
**PartiallyUpdateOneBook** metodunu yaz.

HttpPut dan farklı olarak nesnenin alan olarak  
güncellemesi

**Microsoft.AspNetCore.Mvc.NewtonsoftJson**  
**Version 6.0.10**

**Microsoft.AspNetCore.JsonPatch**  
**Version 6.0.10**

Bu paketleri indirdik şimdi program.cs de yapılandıralım.

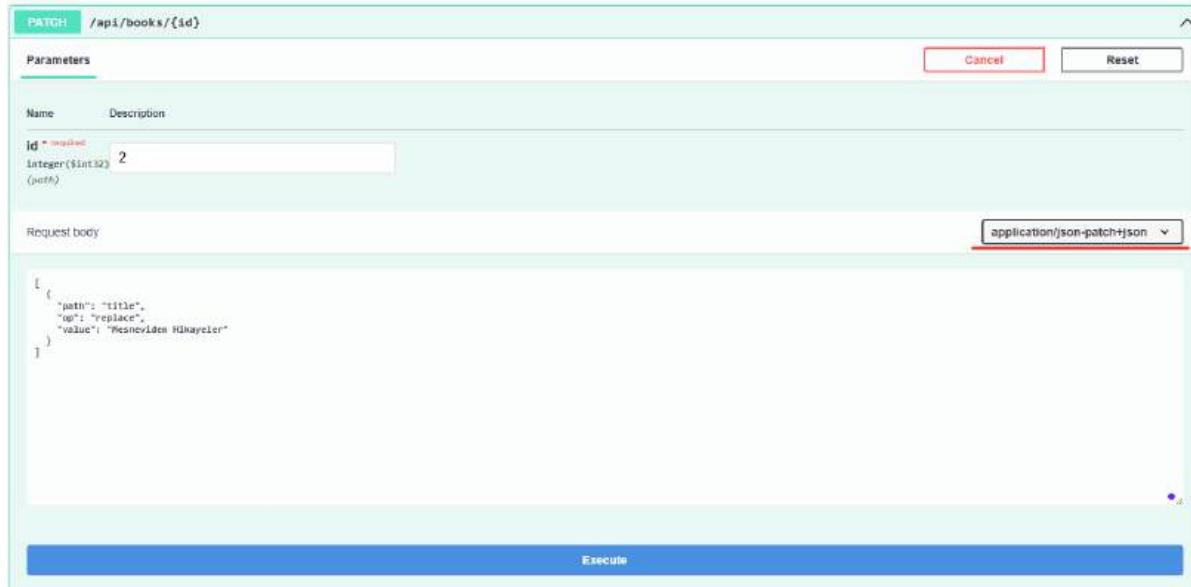
```
builder.Services.AddControllers()  
    .AddNewtonsoftJson();  
    //TODO: NewtonsoftJson paketini kullanabilmek için gerekli ayarları yapalım.
```

# Dördüncü Bölüm?

## MODELLER İLE ÇALIŞMA

### Proje Oluşturma

Kodu yazdıktan sonra **swagger** üzerinden metodu çalıştırıyalım?



# Dördüncü Bölüm?

MODELLER İLE ÇALIŞMA

Proje Oluşturma

SORU : Uygulamayı tekrar çalışın, 2 nolu kitabın **adını** ve  
**fiyatını** aynı anda güncelleyin ???

# Dördüncü Bölüm?

MODELLER İLE ÇALIŞMA  
Proje Oluşturma

CEVAP !!!

PATCH /api/books/{id}

Parameters

Name Description

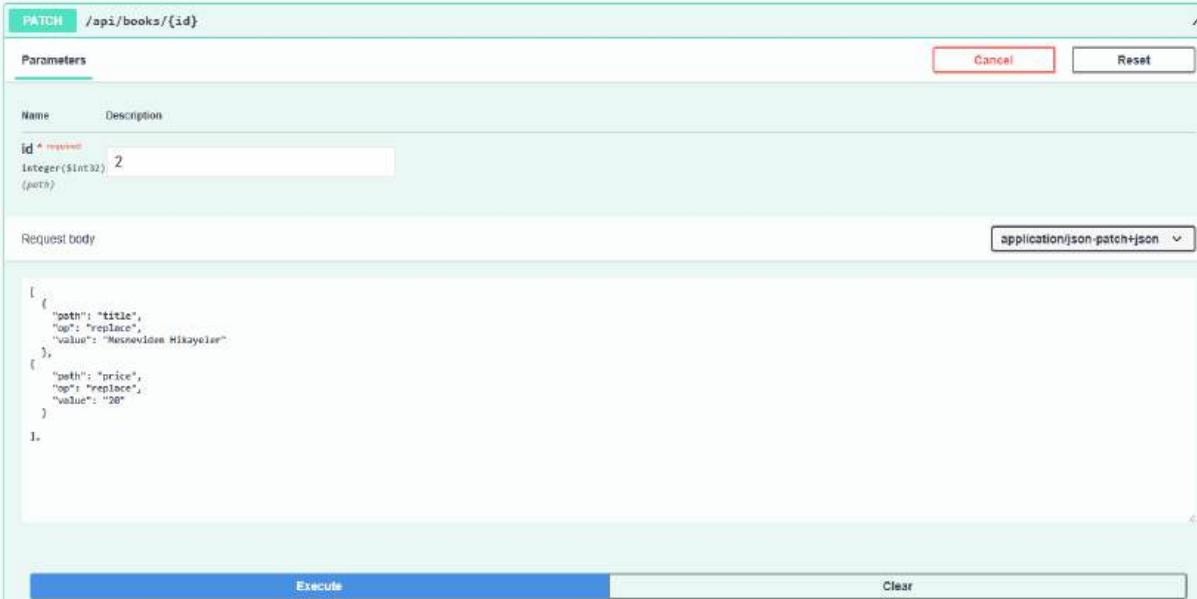
**id** required  
integer(sint32)  
(path) 2

Request body

application/json-patch+json

```
[{"path": "title", "op": "replace", "value": "Mecmueden Hikayeler"}, {"path": "price", "op": "replace", "value": "20"}]
```

Execute Clear



# **Dördüncü Bölüm Bitti**

## **Hadi Özetleyelim?**

# Beşinci Bölüm?

POSTMAN

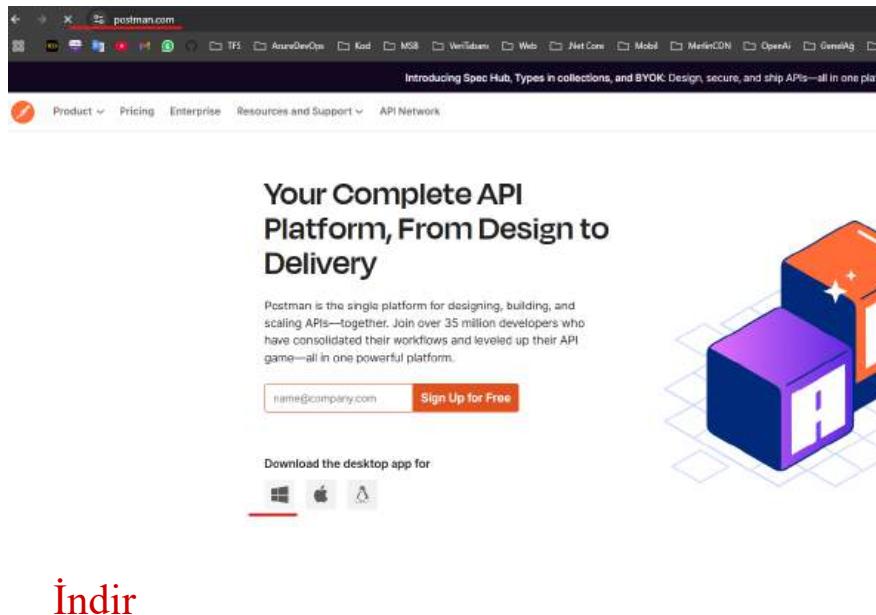
İçerik

- API Test
- HTTP Yöntemlerinin Test Edilmesi
- Global değişkenler
- Koleksiyon değişkenleri
- API Test Scriptleri
- Random Fonksiyonlar



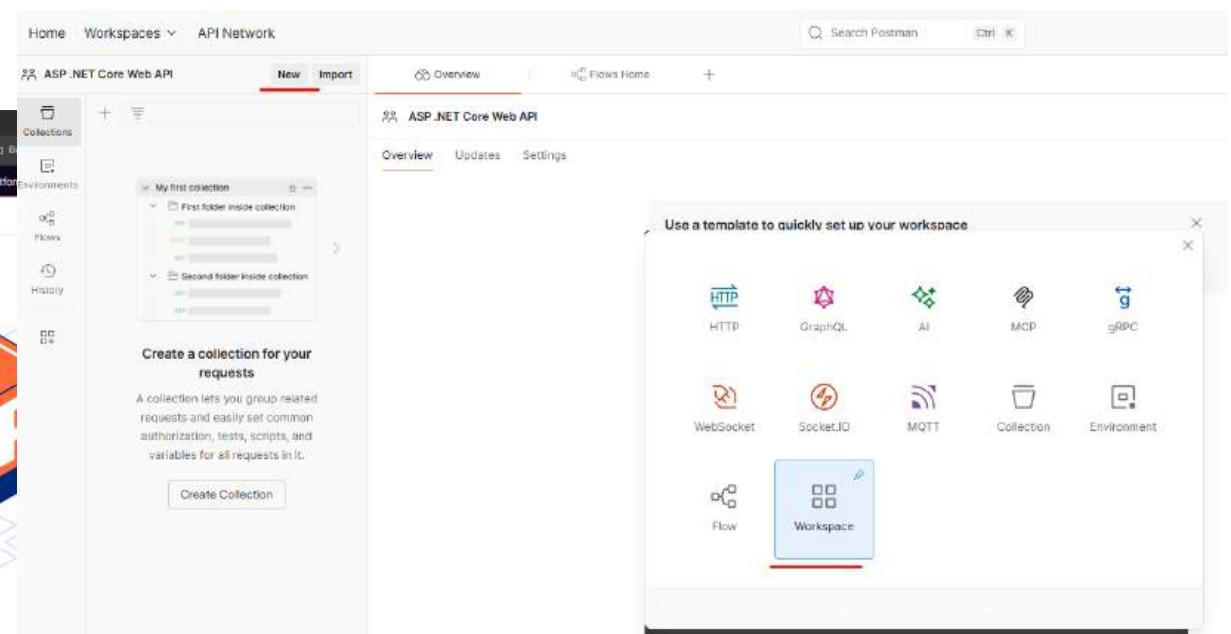
# Beşinci Bölüm?

POSTMAN  
HTTP Yöntemlerin Test Edilmesi



İndir

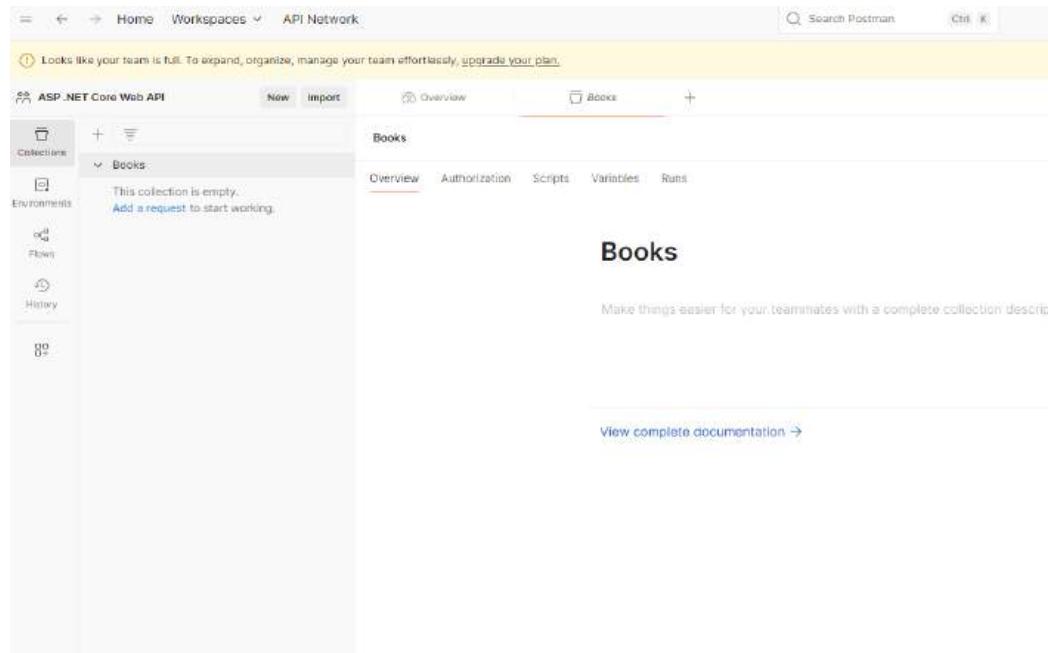
ASP .NET Core Web API isimli workspaces oluşturun



# Beşinci Bölüm?

POSTMAN

HTTP Yöntemlerin Test Edilmesi



Books adında Collection oluşturmalım !!!

# Beşinci Bölüm?

## POSTMAN

### HTTP Yöntemlerin Test Edilmesi

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (selected), 'Environments', 'Pins', and 'History'. The main area has a tree view with 'Books' expanded, showing 'GET All Books'. Below it, a request card is displayed: Method 'GET', URL 'https://localhost:7291/api/books', and a 'Send' button. Underneath the card, tabs for 'Params', 'Authorization', 'Headers (0)', 'Body', 'Scripts', and 'Settings' are visible. A table titled 'Query Params' with columns 'Key', 'Value', and 'Description' is shown, containing one entry: 'Key' and 'Value' are both empty, and 'Description' is also empty.

Şimdi uygulamamızı çalıştıralım ?

**Api/books** metoduna postman üzerinden get isteği atalım

**api/books**

# **Beşinci Bölüm?**

**POSTMAN**

**HTTP Yöntemlerin Test Edilmesi**

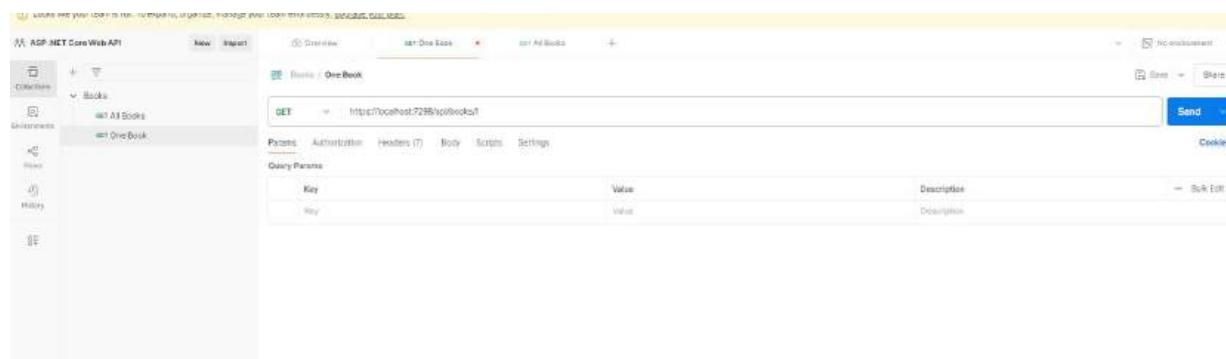
**Şimdi sıra sizde 1 nolu kitabı getiren get metodunu  
Postman üzerinden oluşturun ???**

# Beşinci Bölüm?

POSTMAN

HTTP Yöntemlerin Test Edilmesi

CEVAP !!!

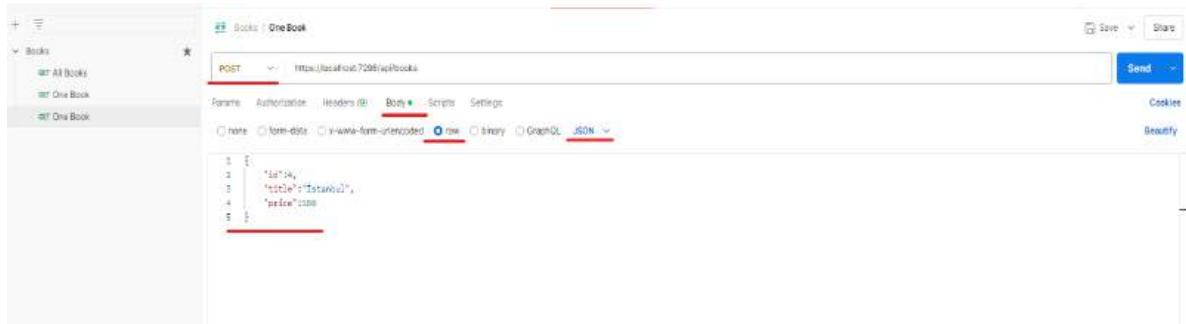


/api/books/1

# Beşinci Bölüm?

## POSTMAN

### HTTP Yöntemlerin Test Edilmesi



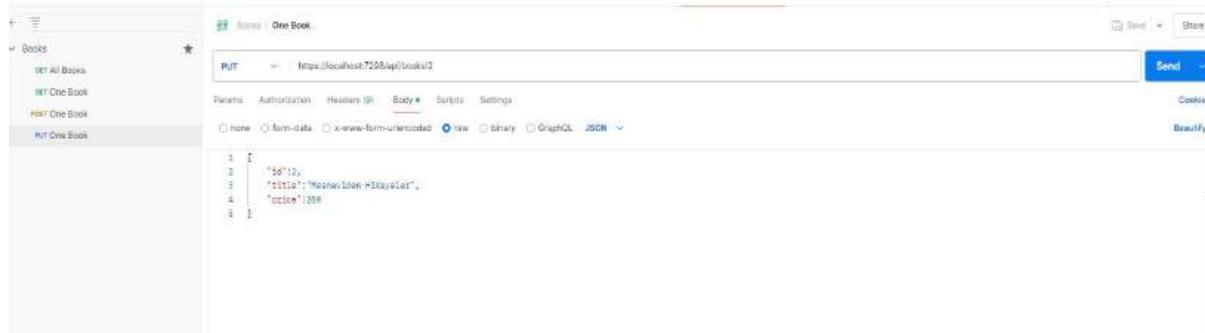
Post metodu ile postman üzerinden kitap ekleme

/api/books

# Beşinci Bölüm?

POSTMAN

HTTP Yöntemlerin Test Edilmesi



Put metodu ile postman üzerinden 2 numaralı kitabı güncelleyin !!!

<https://localhost:7298/api/books/2>

# Beşinci Bölüm?

## POSTMAN

### HTTP Yöntemlerin Test Edilmesi

The screenshot shows the Postman interface with the following details:

- Request Method:** DELETE
- URL:** <http://localhost:7298/api/books/2>
- Headers:** (Empty)
- Body:** (Empty)
- Query Params:** (Empty)
- Send** button is visible.

Delete metodu ile postman üzerinden 2 numaralı kitabı silelim !!!

<https://localhost:7298/api/books/2>

# Beşinci Bölüm?

POSTMAN

HTTP Yöntemlerin Test Edilmesi

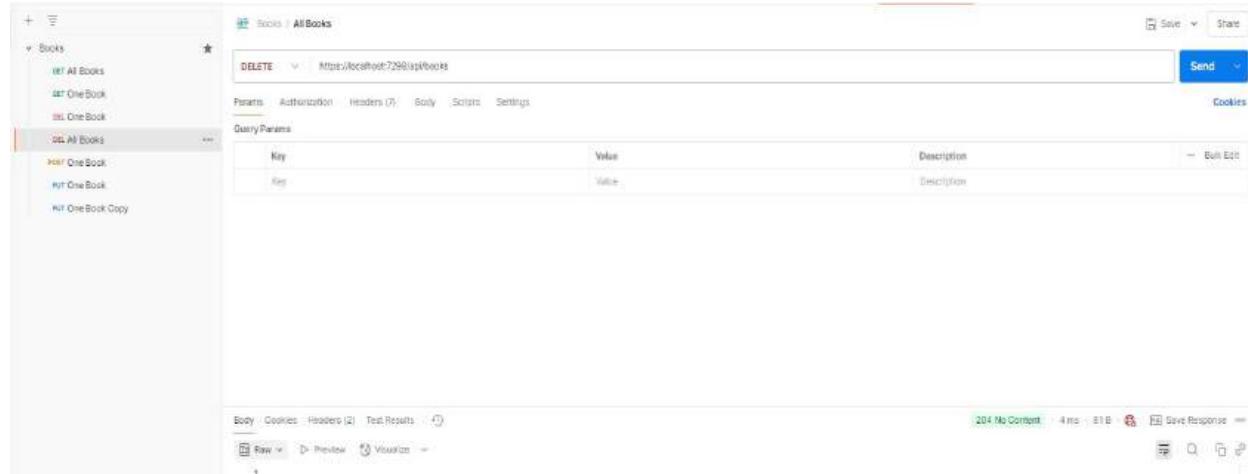
Şimdi sıra sizde **Delete** metodu ile **Postman** üzerinden tüm kitapları sile metodu oluşturun ???

# Beşinci Bölüm?

MODELLER İLE ÇALIŞMA

Proje Oluşturma

CEVAP !!!

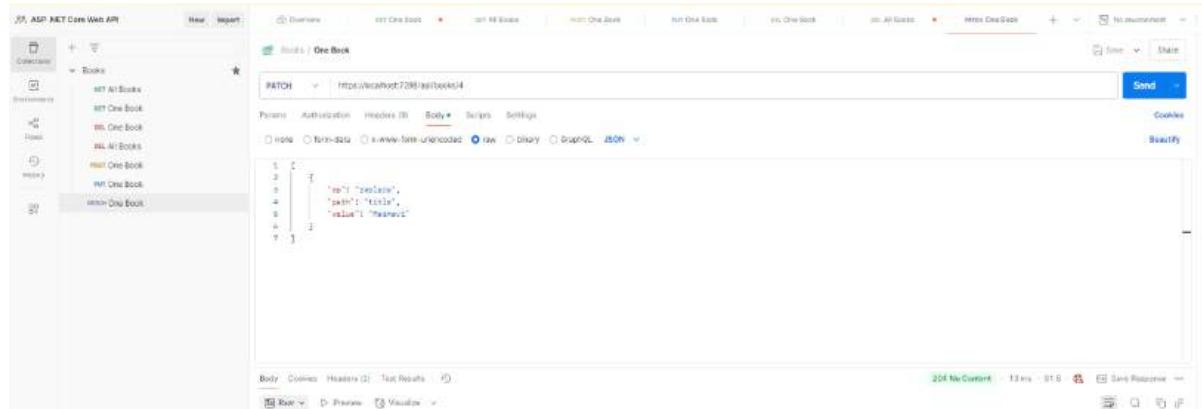


api/books

# Beşinci Bölüm?

## POSTMAN

### HTTP Yöntemlerin Test Edilmesi



Patch metodu ile postman üzerinden 4 numaralı kitabı güncelleyin !!!

<https://localhost:7298/api/books/2>

# Beşinci Bölüm?

## POSTMAN

### Global Değişkenler

The image consists of three vertically stacked screenshots of the Postman application interface, demonstrating the creation and use of global variables.

- Screenshot 1:** Shows a collection named "Books" with several items. A POST request for "All Books" is selected. In the "Authorization" tab, a "Set as variable" button is highlighted. The "Value" field contains "https://localhost:7299".
- Screenshot 2:** Shows a collection named "Books" with several items. A GET request for "Books / All Books" is selected. In the "Authorization" tab, a "Set as new variable" button is highlighted. The "Name" field is set to "baseURL", the "Value" field is set to "https://localhost:7299", and the "Scope" dropdown is set to "Global".
- Screenshot 3:** Shows a collection named "Books" with several items. A GET request for "Books / All Books" is selected. The "Authorization" tab shows the "baseURL" variable has been expanded to "https://localhost:7299".

Sunucu ve port bilgisi sürekli tekrar ediyor bunu **Global değişkene** ata !!!

# Beşinci Bölüm?

## POSTMAN

### Koleksiyon Değişkenler

The screenshot shows the Postman interface with three separate requests to the same endpoint: `/api/books/{id}`. Each request has a different value for the 'key' query parameter:

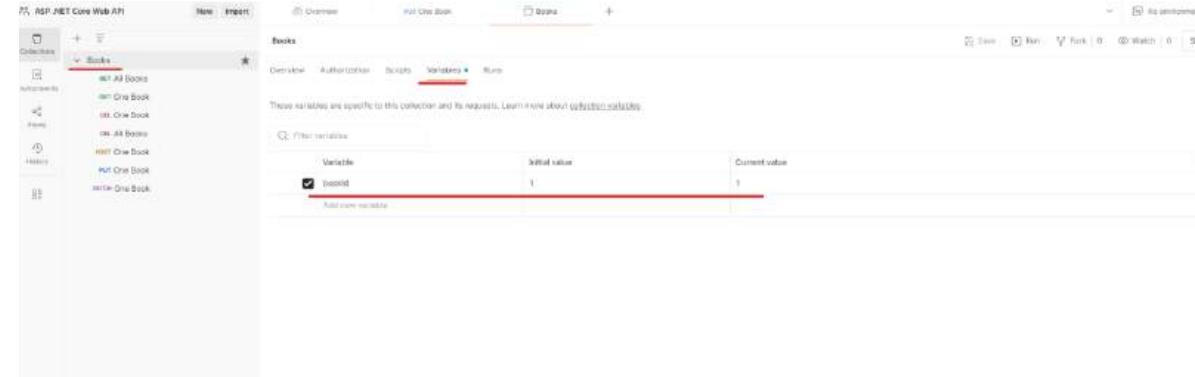
- Request 1: Key = key
- Request 2: Key = bookId
- Request 3: Key = id

**Id bilgisi sürekli tekrar ediyor bunu  
Koleksiyon değişkene ata !!!**

# Beşinci Bölüm?

## POSTMAN

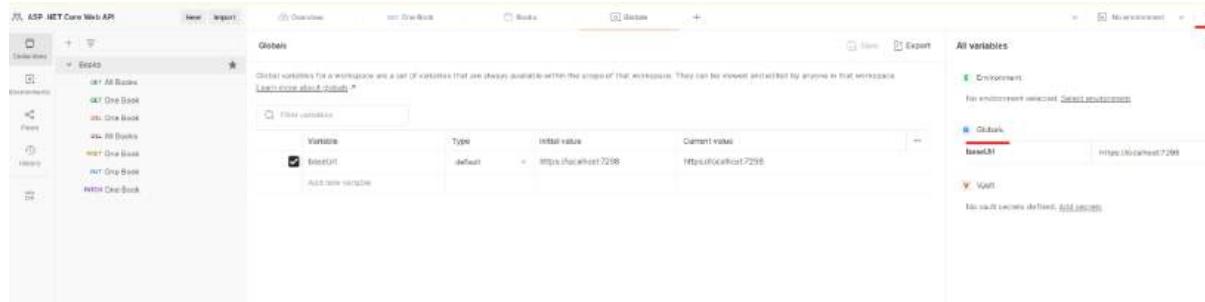
Koleksiyon Değişkeni nasıl değiştireceğiz



# Beşinci Bölüm?

POSTMAN

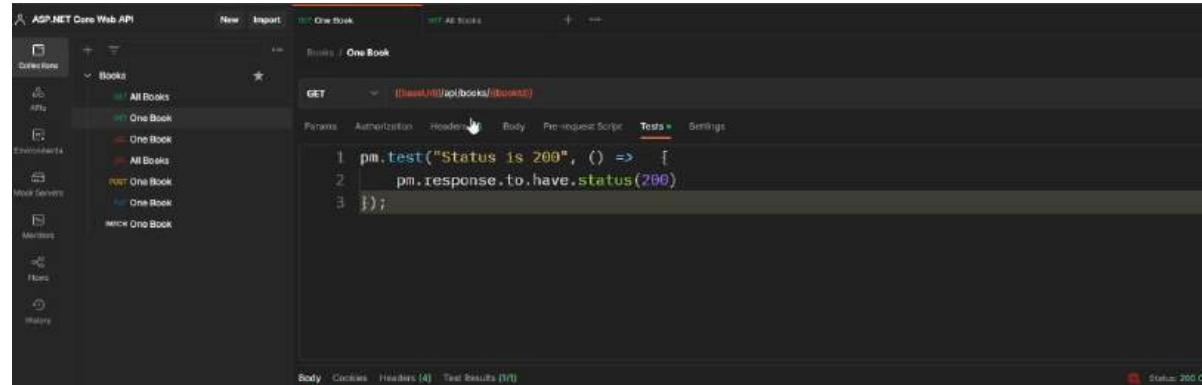
Global Değişkeni nasıl değiştireceğiz



# Beşinci Bölüm?

POSTMAN

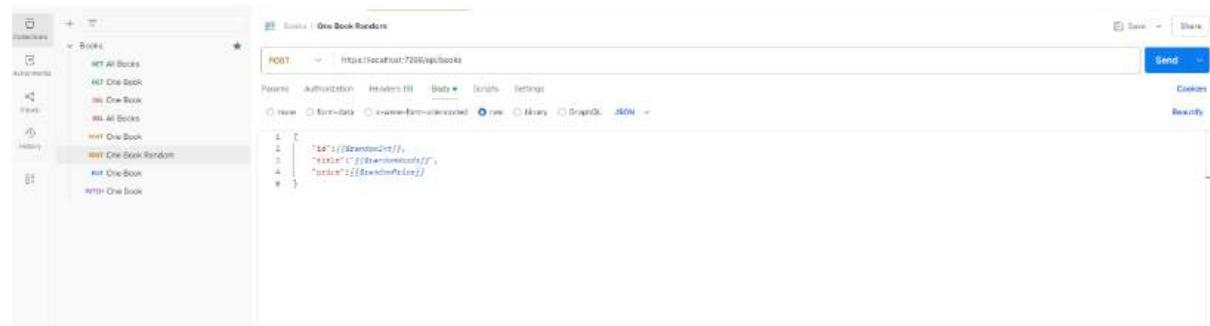
API Test Scriptleri



# Beşinci Bölüm?

## POSTMAN

### Random Foksiyonlar



Post metodu ile **One Book Random fonksiyonu** ile postman üzerinden kitap ekleme

```
{  
  "id": "{$randomInt}","  
  "title": "{$randomWords}","  
  "price": "{$randomPrice}"  
}
```

# Beşinci Bölüm?

## POSTMAN

### Döngülü Random Ekleme

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections', 'Environments', 'Flows', and 'History'. The main area is titled 'Books' and contains a list of requests:

- GET All Books
- GET One Book
- DEL One Book
- DEL All Books
- POST One Book
- POST One Book Random
- PUT One Book
- PATCH One Book

A context menu is open over the 'POST One Book Random' request, showing options like 'Add request', 'Add folder', 'Run', 'Share', 'Move', 'Fork', 'Rename', 'Duplicate', 'Delete', and 'More'. To the right, there's a 'Run Sequence' section with a tree view of requests and a 'Run configuration' panel where 'Iterations' is set to 100 and a 'Run' button is highlighted.

Gizlilik Derecesi [Tasnif Dışı](#)

Sonra All Book ile kontrol et !!!

Slayt Numarası .../...

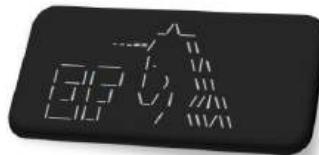
**Beşinci Bölüm Bitti**  
**Hadi Özetleyelim?**

# Altıncı Bölüm?

## EF Core

### İçerik

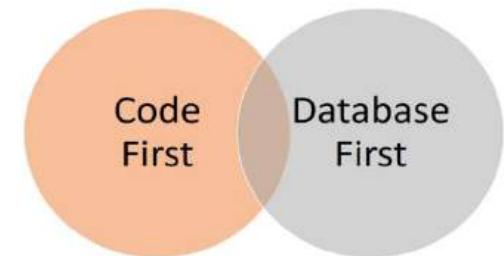
- Entities / Models
- Repository Context
- Connection String
- Migrations
- Type Configuration
- Inversion of Control
- API Testi



## EF Core

- NET geliştiricilerinin .NET nesneleri kullanarak bir veri tabanıyla çalışmasına olanak tanır.
- Genellikle yapılması gereken veri erişim kodunun çoğu ortadan kaldırır.
- Entity Framework Microsoft tarafından geliştirilmiş **ORM** çerçevelerinden biridir.

## EF Core



# Altıncı Bölüm?

EF Core

Proje Oluşturma

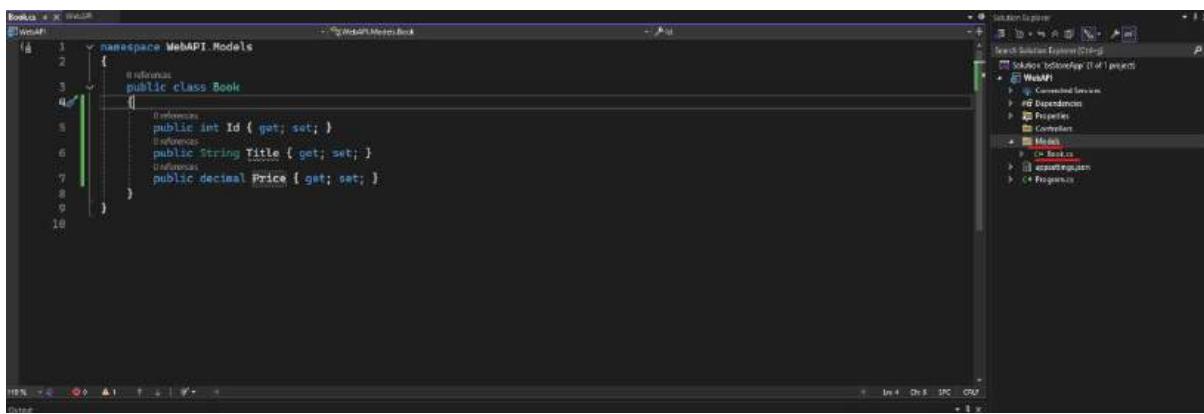
**ASP .NET Core WEB API** alanında **WebAPI** adında proje  
oluşturralım.

**WeatherForecast.cs** ve **WeatherForecastController** sil.

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Models



The screenshot shows the Visual Studio IDE. On the left, the code editor displays the `Book.cs` file with the following content:

```
1  namespace WebAPI.Models
2  {
3      public class Book
4      {
5          public int Id { get; set; }
6          public String Title { get; set; }
7          public decimal Price { get; set; }
8      }
9  }
```

On the right, the Solution Explorer shows the project structure:

- Solution Explorer: WebAPI (1 of 1 project)
  - WebAPI
    - Controllers
    - Properties
    - Models
      - Book.cs
      - Book.csproj
      - Book.csproj.user
    - Views

Models klasörünü oluştur.

İçine Book clasını oluştur ve resimdeki kodu yaz.

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Repository Context

```
//TODO: RepositoryContext class ini DbContext classından kalıtım alacak şekilde oluşturun.  
//TODO: DbContext için Microsoft.EntityFrameworkCore namespace'ini ekleyin.  
//TODO: Bunu Package Manager Console'da yapın.  
7 references  
public class RepositoryContext : DbContext  
{  
  
    //TODO: DbContextOptions nesnesi verdiğimizde base(options) ile DbContext göndermiş oluyoruz.  
    0 references  
    public RepositoryContext(DbContextOptions options) : base(options)  
    {  
  
    }  
    7 references  
    public DbSet<Book> Books { get; set; }  
  
    //TODO: Tip konfigürasyonunu için ilgili kodu ekleyin.  
    //TODO: Artık Model oluşturulurken BookConfig sınıfını kullanacağız.
```

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Connection String

The screenshot shows a Visual Studio IDE interface. On the left, the code editor displays the `appsettings.json` file with the following content:

```
Schema: https://json.schemastore.org/appsettings.json
1  {
2    >   "Logging": {},
3    >   "AllowedHosts": "*",
4    //TODO: ConnectionStrings oluştur.
5    >   "ConnectionStrings": {
6      >     "sqlConnection": "Data Source = (localdb)\\MSSQLLocalDB; Initial Catalog=bsStoreApp; Integrated Security:true;"
7      >     "sqlConnection": "Data Source = DESKTOP-C8PPKRB; Initial Catalog=bsStoreApp; Integrated Security:true;"
8      //TODO: Data Source sql server sunucusu bilgisi
9      //TODO: Initial Catalog veritabanı adı
10     //TODO: Integrated Security true ise Windows Authentication, false ise SQL Server Authentication kullanılır.
11   }
12 }
13 }
```

On the right, the Package Manager Console window shows the command:

```
PM> Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 6.0.10 -ProjectName WebAPI
```

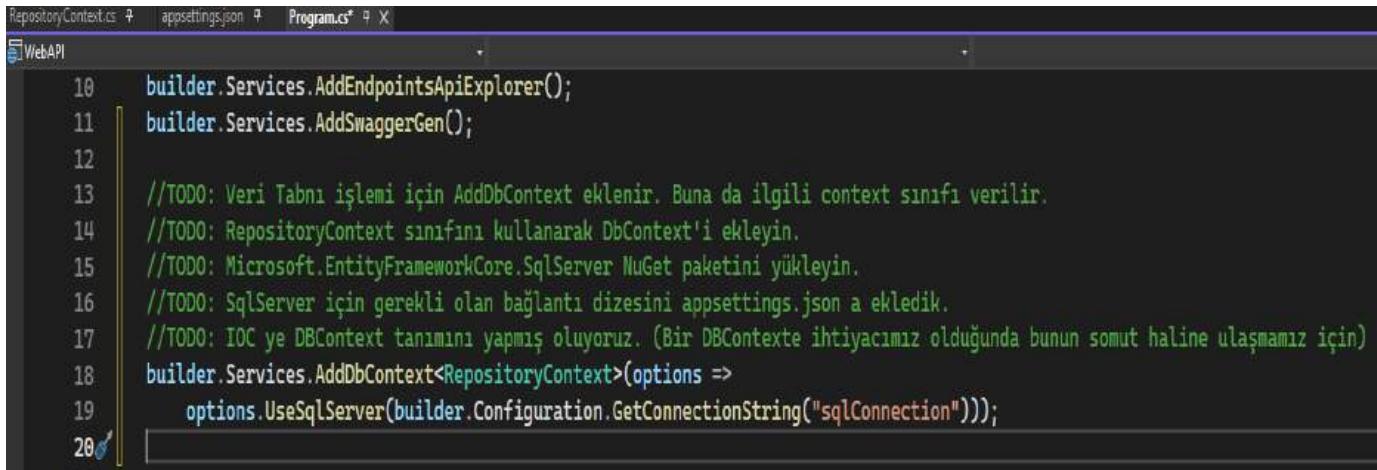
**SQLSERVER ekleyeceğiz. Register işlemi yapabilmek için.**

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Connection String

Şimdi program.cs ye servis kaydını yapalım.



```
10 builder.Services.AddEndpointsApiExplorer();
11 builder.Services.AddSwaggerGen();
12
13 //TODO: Veri Tabını işlemi için AddDbContext eklenir. Buna da ilgili context sınıfı verilir.
14 //TODO: RepositoryContext sınıfını kullanarak DbContext'i ekleyin.
15 //TODO: Microsoft.EntityFrameworkCore.SqlServer NuGet paketini yükleyin.
16 //TODO: SqlServer için gerekli olan bağlantı dizesini appsettings.json a ekledik.
17 //TODO: IOC ye DBContext tanımını yapmış oluyoruz. (Bir DBContexte ihtiyacımız olduğunda bunun somut haline ulaşmamız için)
18 builder.Services.AddDbContext<RepositoryContext>(options =>
19     options.UseSqlServer(builder.Configuration.GetConnectionString("sqlConnection")));
20
```

# Altıncı Bölüm?

EF Core

Proje Oluşturma / IoC

Basit anlatımıyla, bir nesnenin bağımlılıklarının doğrudan kendisi tarafından oluşturulması yerine, bu bağımlılıkların dışarıdan bir mekanizma tarafından verilmesidir.

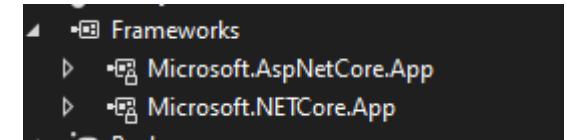
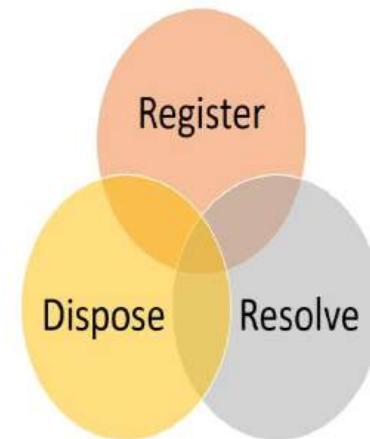
Kontrolü almamız için,

IoC üzerinden bize dbcontext lazım diyoruz.

Buda diyorki sen **RepositoryContext** kaydetmiştin bunu çözüp sana veriyorum.

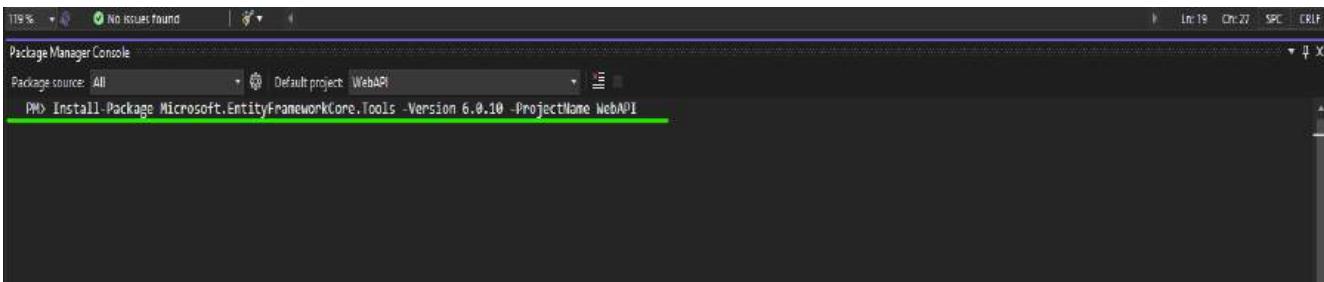
Her nesne gibi bunların da yaşam döngüsü var. IoC dispose edilmesini üstleniyor. (Bir tane mi oluşturacağım, Her istekte ayrı bir tane mi oluşturacağım, Ya da her istekte yeni nesne mi türeteceğim)

Inversion  
of Control



# Altıncı Bölüm?

EF Core  
Proje Oluşturma / Migration

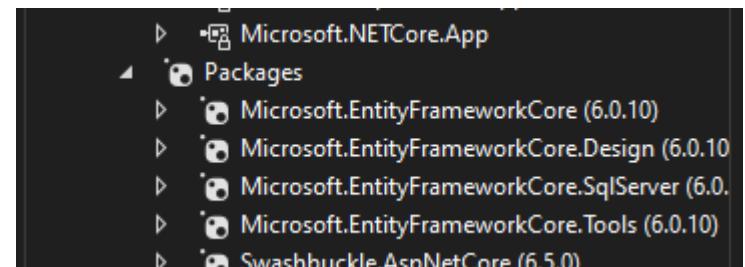


```
PM> Install-Package Microsoft.EntityFrameworkCore.Tools -Version 6.0.10 -ProjectName WebAPI
```

Migration çalıştırılması için ilgili paketi yükleyelim. Yükledikten sonra packages de kontrol edelim !!!

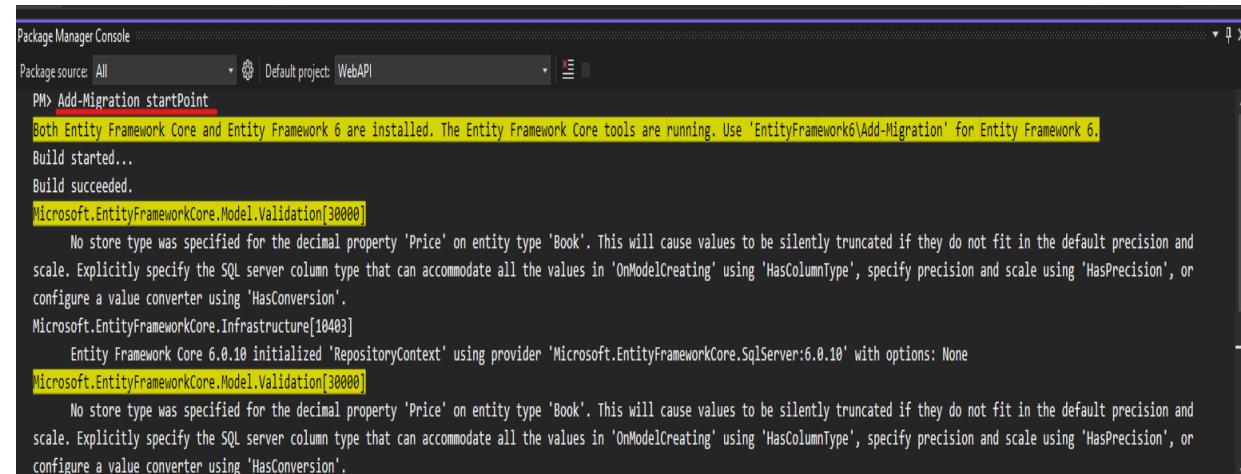
Daha sonra uygulama için ilgili paketi kuralım. (Ef Core hangi uygulama üzerinde koşacak ise buna gerek var)

Sonuç, aşağıdakilerin hepsi olması lazım



# Altıncı Bölüm?

EF Core  
Proje Oluşturma / Migration



The screenshot shows the Package Manager Console window in Visual Studio. The command entered is 'Add-Migration startPoint'. The output text is:  
Both Entity Framework Core and Entity Framework 6 are installed. The Entity Framework Core tools are running. Use 'EntityFramework6\Add-Migration' for Entity Framework 6.  
Build started...  
Build succeeded.  
Microsoft.EntityFrameworkCore.Model.Validation[30000]  
No store type was specified for the decimal property 'Price' on entity type 'Book'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value converter using 'HasConversion'.  
Microsoft.EntityFrameworkCore.Infrastructure[10403]  
Entity Framework Core 6.0.10 initialized 'RepositoryContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer:6.0.10' with options: None  
Microsoft.EntityFrameworkCore.Model.Validation[30000]  
No store type was specified for the decimal property 'Price' on entity type 'Book'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value converter using 'HasConversion'.

Şimdi Migration komutunu çalıştırıralım !!!!

Eğer projede hata yok ise projen derlenebiliyor ise bir migration scripti oluşacak

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Migration

The screenshot shows a code editor window with a dark theme. The file is named '202505011345\_startPoint.cs'. The code defines a partial class 'startPoint' that inherits from 'Migration'. It contains two methods: 'Up' and 'Down'. The 'Up' method creates a table named 'Books' with columns for Id (int, primary key, identity), Title (nvarchar(max)), and Price (decimal(18,2)). The 'Down' method is also defined.

```
202505011345_startPoint.cs # X
WebAPI
  ↑ 1 reference
  ↑ 1 reference
public partial class startPoint : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Books",
            columns: table => new
            {
                Id = table.Column<int>(type: "int", nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                Title = table.Column<string>(type: "nvarchar(max)", nullable: false),
                Price = table.Column<decimal>(type: "decimal(18,2)", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Books", x => x.Id);
            });
    }

    protected override void Down(MigrationBuilder migrationBuilder) ...
}
```

Oluşan migration script  
Up, Down diye iki tane fonksiyon oluştur.

Eğer uyguladıktan sonra migrationu geri alacak olur ise **Down** metodu çalışıyor ilgili migration siliyor.

Bunu veri tabanına yansıtacağız.

The screenshot shows the same code editor window with the 'startPoint.cs' file. It highlights the 'Up' and 'Down' methods. The 'Up' method is shown in its original state, while the 'Down' method has been modified to call 'DropTable' on the 'migrationBuilder' object, effectively dropping the 'Books' table.

```
202505011345_startPoint.cs # X
WebAPI
  ↑ 1 reference
  ↑ 1 reference
public partial class startPoint : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder) ...

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "Books");
    }
}
```

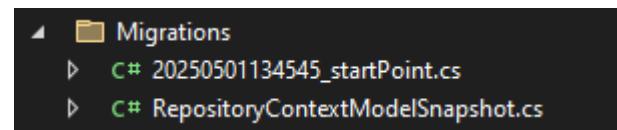
# Altıncı Bölüm?

EF Core

Proje Oluşturma / Migration

Migration dan sonra ilgili klasör oluşur.

Bu oluşum zaman-bizim\_verdiğimiz\_isim şekilde olur

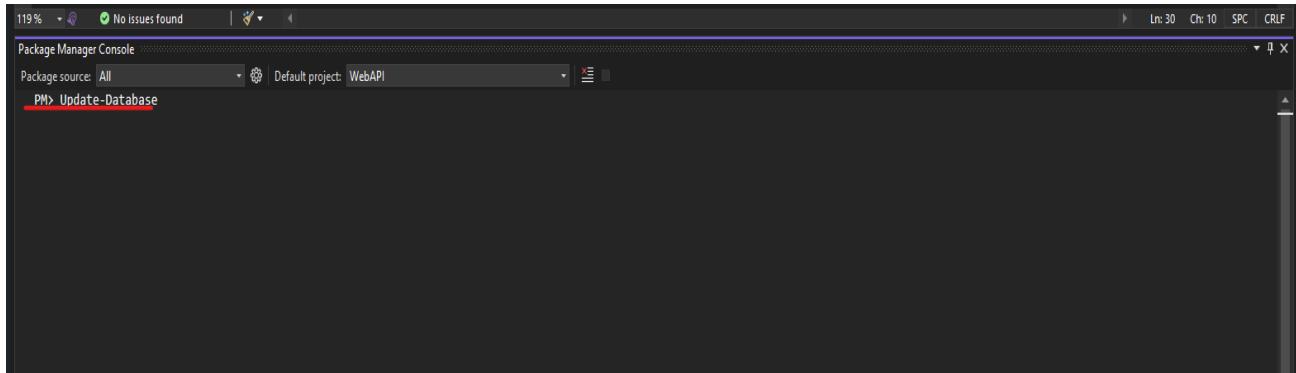


Bunu bir versiyonuda veri tabanında saklanır.

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Migration



```
119% No issues found
Package Manager Console
Package source: All | Default project: WebAPI | x
PM> Update-Database
```

Şimdi ilgili script ile veri tabanına gitmesini sağlıyoruz

Çalıştırdıktan sonra bir hata aldık sebebi  
ne olabilir?

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Migration

## Hatanın çözümü

"sqlConnection": "Data Source = DESKTOP-C8PPKRB; Initial Catalog=bsSroreApp; Integrated Security:true;"  
: yerine = olacak

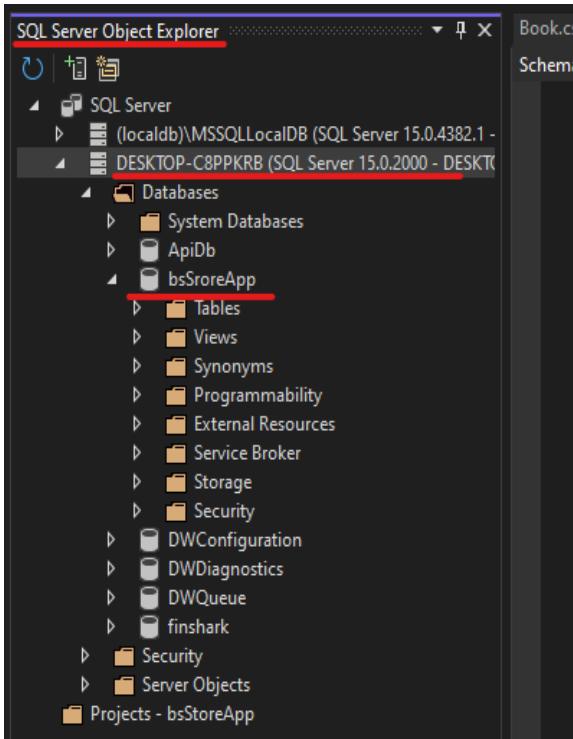
Tekrar çalıştırıldık tablonun output olduğunu gördük.

```
Executed DBCommand (4ms) [Parameters=[], CommandType=Text, CommandText=CREATE TABLE [Books] ([Id] int NOT NULL IDENTITY, [Title] nvarchar(max) NOT NULL, [Price] decimal(18,2) NOT NULL, CONSTRAINT [PK_Books] PRIMARY KEY ([Id]))];
```

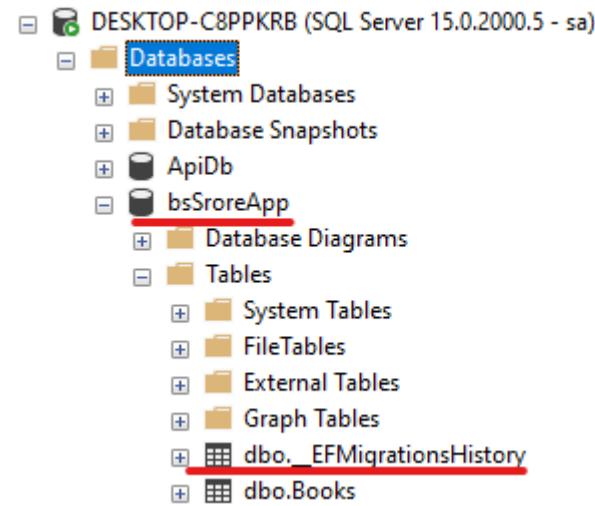
# Altıncı Bölüm?

# EF Core

## Proje Oluşturma / Migration



Migration sonucu oluşan veri tabanını ve tabloyu görelim !!!



# Altıncı Bölüm?

EF Core

Proje Oluşturma / Tip Konfigürasyonu

Tip konfigürasyonuna neden ihtiyaç var ?

- Bazı alanlara default değerler vermek için,
- Bazı alanları da kısıtlamak için,
- Çekirdek datalar girmek için

Şimdi **Repositories** altına **Config** dosyası oluşturalım. Altına **BookConfig** clasını ekle ilgili kodu yaz.

The screenshot shows the Visual Studio IDE. On the left, the code editor displays `BookConfig.cs` with the following content:

```
1  using Microsoft.EntityFrameworkCore;
2  using Microsoft.EntityFrameworkCore.Metadata.Builders;
3  using WebAPI.Models;
4
5  namespace WebAPI.Repositories.Config
6  {
7      //TODO: IEntityConfiguration dan kalıtım alıyoruz. Generic yapıda Book sınıfını alıyoruz.
8      public class BookConfig : IEntityConfiguration<Book>
9      {
10         public void Configure(EntityTypeBuilder<Book> builder)
11         {
12             //TODO: Migrasyon işlemlerinde tabloda da veri yoksa çekirdek veriler ekleyeceğiz.
13             builder.HasData(
14                 new Book { Id = 1, Title = "Karagöz ve Hacivat", Price = 75 },
15                 new Book { Id = 2, Title = "Mesnevi", Price = 175 },
16                 new Book { Id = 3, Title = "Devlet", Price = 375 }
17             );
18         }
19     }
```

On the right, the Solution Explorer pane shows the project structure:

- Solution: bStoreApp (1 of 1 project)
  - WebAPI
    - Connected Services
    - Dependencies
    - Properties
    - Controllers
    - Migrations
    - Models
    - Repositories
      - Config
        - BookConfig.cs
      - RepositoryContext.cs
    - appsettings.json
    - Program.cs

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Tip Konfigürasyonu

The screenshot shows a development environment with two windows. On the left is the 'Package Manager Console' window with the command 'PM> Add-Migration seedData' entered. On the right is the 'Code Editor' window displaying a C# file named '202505011428\_\_seedData.cs'. The code in the file is as follows:

```
1  using Microsoft.EntityFrameworkCore.Migrations;
2
3  #nullable disable
4
5  namespace WebAPI.Migrations
6  {
7      public partial class seedData : Migration
8      {
9          protected override void Up(MigrationBuilder migrationBuilder)
10         {
11         }
12     }
13
14     protected override void Down(MigrationBuilder migrationBuilder)
15     {
16     }
17 }
18
19
20 }
```

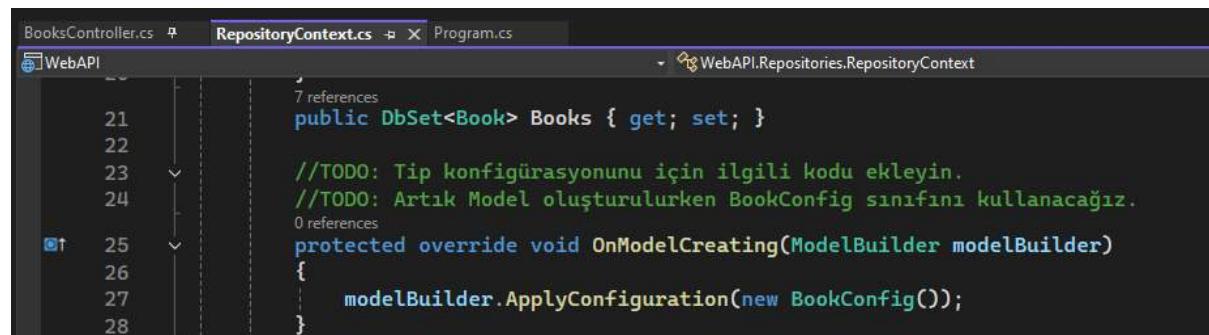
To the right of the code editor, a text overlay reads 'Şimdi tekrar migration yapacağız..'

E migration yaptık içi boş geldi. Neden ???

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Tip Konfigürasyonu



The screenshot shows a code editor with the tab 'RepositoryContext.cs' selected. The code defines a context class named 'WebAPI.Repositories.RepositoryContext'. It includes a DbSet named 'Books' and a protected override method 'OnModelCreating' that applies a configuration to the 'BookConfig' class.

```
BooksController.cs RepositoryContext.cs Program.cs
WebAPI
    ↴
    ↴ 7 references
    ↴ public DbSet<Book> Books { get; set; }
    ↴
    ↴ //TODO: Tip konfigürasyonunu için ilgili kodu ekleyin.
    ↴ //TODO: Artık Model oluşturulurken BookConfig sınıfını kullanacağız.
    ↴ 0 references
    ↴ protected override void OnModelCreating(ModelBuilder modelBuilder)
    ↴ {
    ↴     modelBuilder.ApplyConfiguration(new BookConfig());
    ↴ }
```

Veri tabanına işlemi yansıtan **dbcontext** dir.

**Tip Configürasyon** için ilgili kodu yazdık  
ama **dbcontext** ile iletişim konusunda  
herhangi bir işlem yapmadık.

Bu yüzden migration boş geldi.

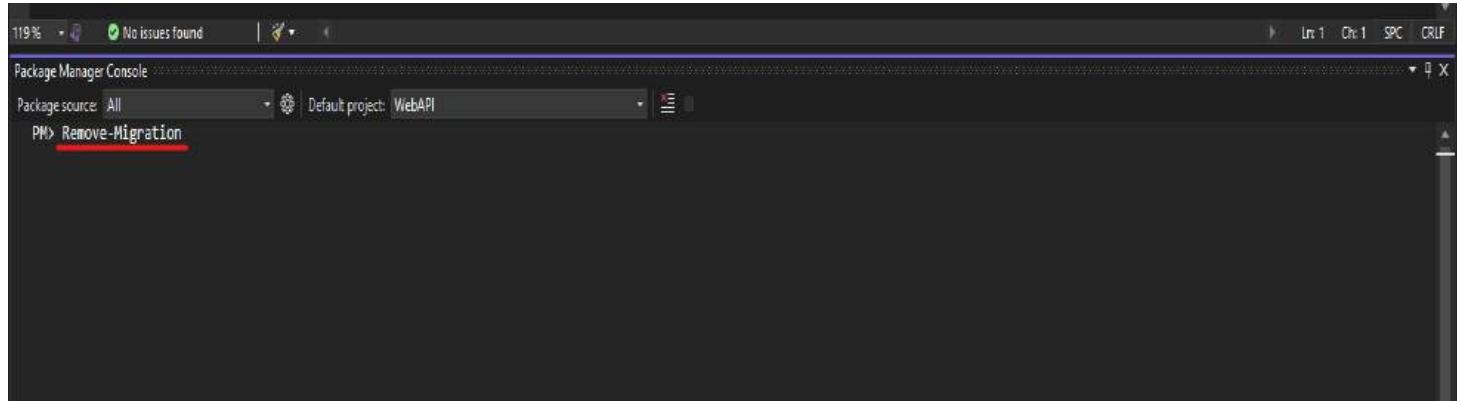
Bunun için ilgili **işlemi** yaptık

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Tip Konfigürasyonu

Şimdi en son **migration** dan kurtulalım.



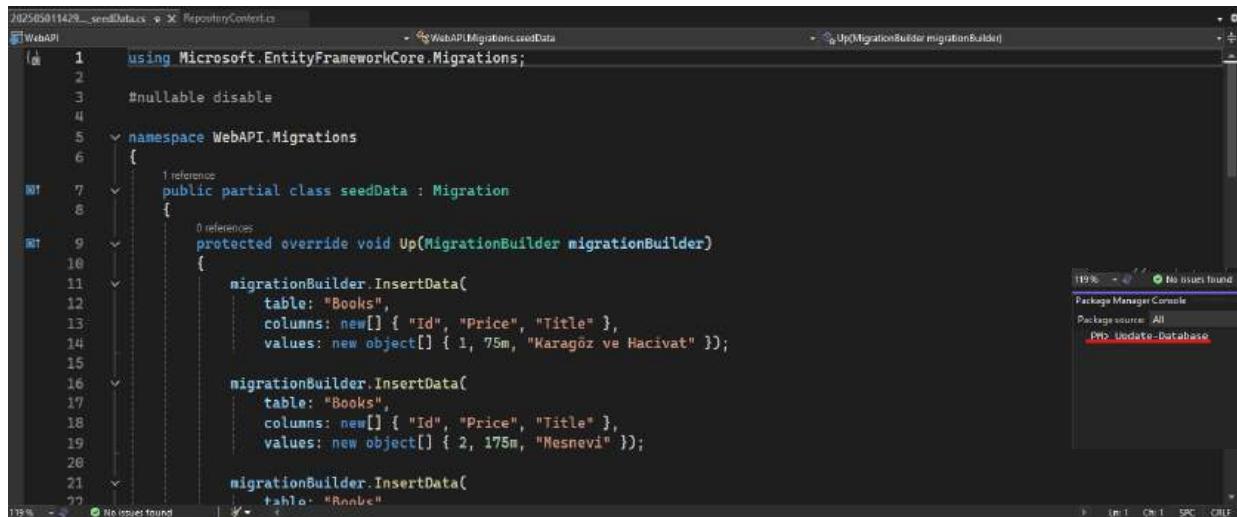
A screenshot of the Package Manager Console in Visual Studio. The console window has a dark theme. At the top, it says "No issues found". Below that, it shows "Package Manager Console" and "Default project: WebAPI". In the main area, the command "PM> Remove-Migration" is typed in, with the entire line underlined in red. The rest of the console window is empty.

# Altıncı Bölüm?

EF Core

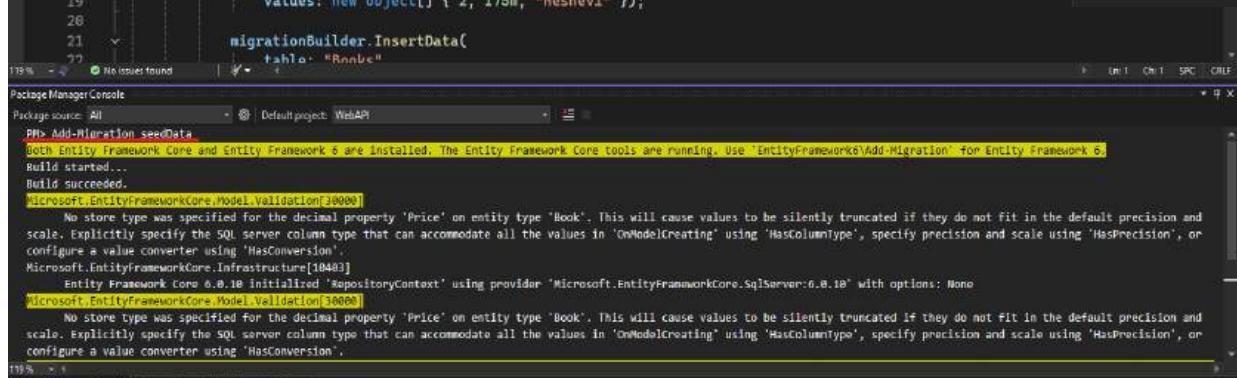
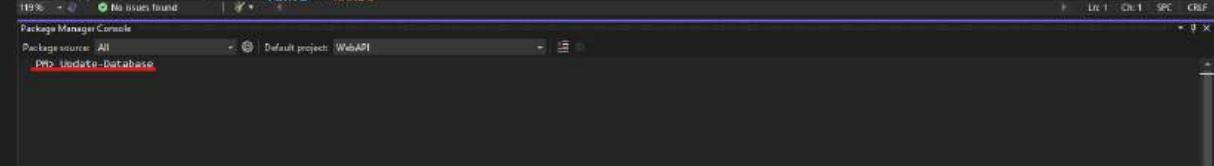
Proje Oluşturma / Tip Konfigürasyonu

Şimdi tekrar **migration** yapalım.



```
1  using Microsoft.EntityFrameworkCore.Migrations;
2
3  #nullable disable
4
5  namespace WebAPI.Migrations
6  {
7      public partial class seedData : Migration
8      {
9          protected override void Up(MigrationBuilder migrationBuilder)
10         {
11             migrationBuilder.InsertData(
12                 table: "Books",
13                 columns: new[] { "Id", "Price", "Title" },
14                 values: new object[] { 1, 75m, "Karagöz ve Hacivat" });
15
16             migrationBuilder.InsertData(
17                 table: "Books",
18                 columns: new[] { "Id", "Price", "Title" },
19                 values: new object[] { 2, 175m, "Mesnevi" });
20
21             migrationBuilder.InsertData(
22                 table: "Books"
23             );
24         }
25     }
26 }
```

Şimdi database'e yansıtırım.



```
PM> Add-Migration seedData
Both Entity Framework Core and Entity Framework 6 are installed. The Entity Framework Core tools are running. Use 'EntityFramework\Add-Migration' for Entity Framework 6.
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Validation[30000]
No store type was specified for the decimal property 'Price' on entity type 'Book'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value converter using 'HasConversion'.
Microsoft.EntityFrameworkCore.Infrastructure[10443]
    Entity Framework Core 6.0.10 initialized 'RepositoryContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer:6.0.10' with options: None
Microsoft.EntityFrameworkCore.Validation[30000]
No store type was specified for the decimal property 'Price' on entity type 'Book'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value converter using 'HasConversion'.

```

Slayt Numarası .../...

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Tip Konfigürasyonu

The screenshot shows the SSMS interface. On the left, the Object Explorer tree view displays the database structure for 'DESKTOP-C8PPKRB (SQL Server 15.0.2000.5 - sa)'. Under the 'bsStoreApp' database, the 'Tables' node is expanded, showing 'Books' and 'Books' again. In the center, the 'SQLQuery1.sql' window contains the SQL query: 'SELECT \* FROM Books'. Below it, the 'Results' tab shows the following data:

	Id	Title	Price
1	1	Karagöz ve Hacivat	75.00
2	2	Mesnevi	175.00
3	3	Devlet	375.00

Şimdi database de kontrol edelim..

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Inversion of Control (IoC)

The screenshot shows the Visual Studio IDE interface. On the left is the code editor with the file 'BooksController.cs' open. The code defines a 'BooksController' class that inherits from 'ControllerBase'. It includes a constructor that takes a 'RepositoryContext' parameter and assigns it to a private field named '\_context'. A TODO comment suggests using dependency injection to resolve the repository context. The 'GetAllBooks()' method uses an 'HttpGet' attribute and returns an 'IActionResult' containing a list of books from the '\_context'. On the right is the 'Solution Explorer' window, which shows a single project named 'bStoreApp' with a single item 'WebAPI'. The 'Controllers' folder contains the 'BooksController.cs' file.

```
[Route("api/{controller}")]
[ApiController]
public class BooksController : ControllerBase
{
    //TODO: İhtiyacınız olan tüm kitapları listeleyin
    //TODO: Aşağıdaki kodun sonuna gelip Ctrl + . tuşlarına basın. Generate constructor seçeneğini seçin. Otomatik olarak
    //oluşturulan constructor'i kullanın.
    private readonly RepositoryContext _context;

    public BooksController(RepositoryContext context)
    {
        //TODO: Burada repository context'i dependency injection ile alıyoruz. Yani repository context'i resolve (çözme)
        //işlemi yapıyoruz. Yani RepositoryContext i new'liyoruz.
        //TODO: Register (Kayıt) işlemini Program.cs dosyasında yaptık.
        _context = context;
    }

    //TODO: GetAllBooks metodu ile tüm kitapları listeleyin
    [HttpGet]
    public IActionResult GetAllBooks()
    {
        var books = _context.Books.ToList();
        return Ok(books);
    }
}
```

Şimdi oluşturduğumuz veri tabanını kullanacağız.

**BooksController** ekliyoruz ama **API** olarak seçmeyi unutma.

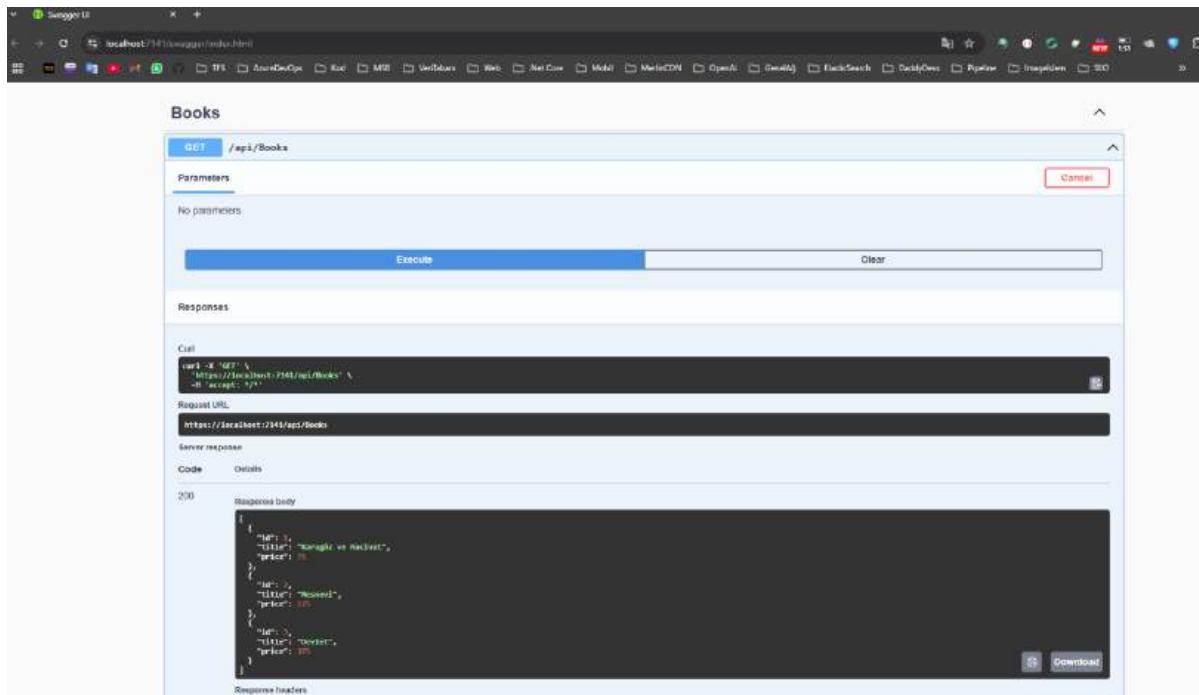
Sonra ilgili kodlamayı yapalım.

**Kodlama TODO lar ile anlattım.**

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Inversion of Control (IoC)



Şimdi uygulamayı çalıştırılarım swagger dan  
dan ilgili metoda istek atalım.

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Inversion of Control (IoC)

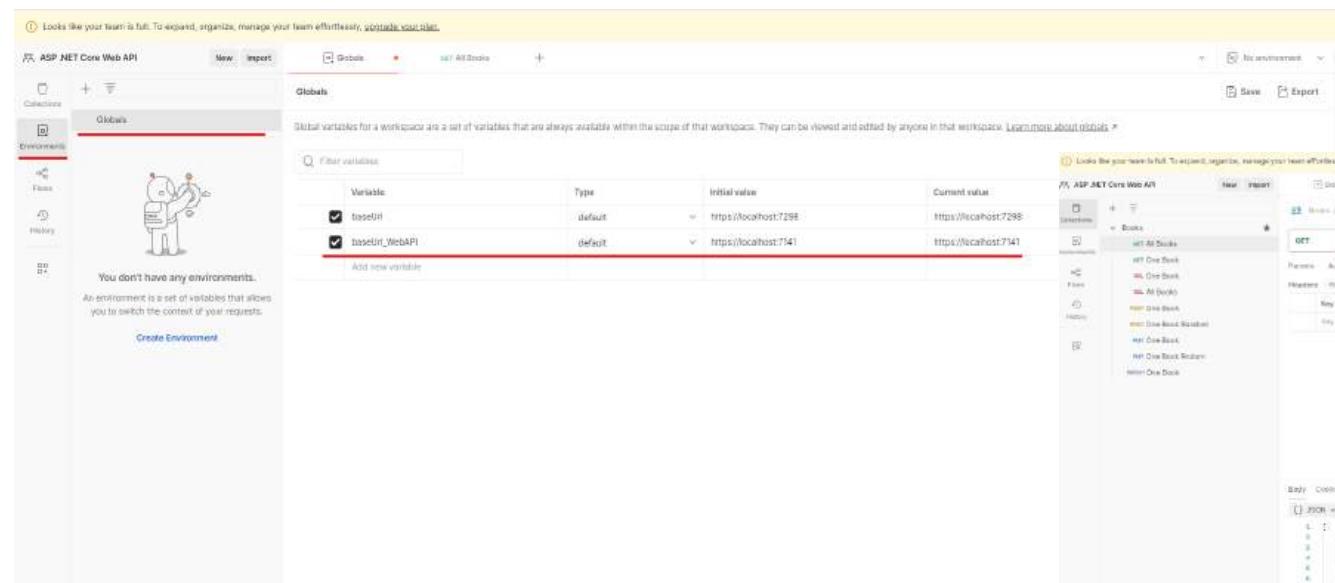
HADİ !!!

ŞİMDİ POSTMAN'ı AÇIN İSTEĞİ ORADAN YAPIN.

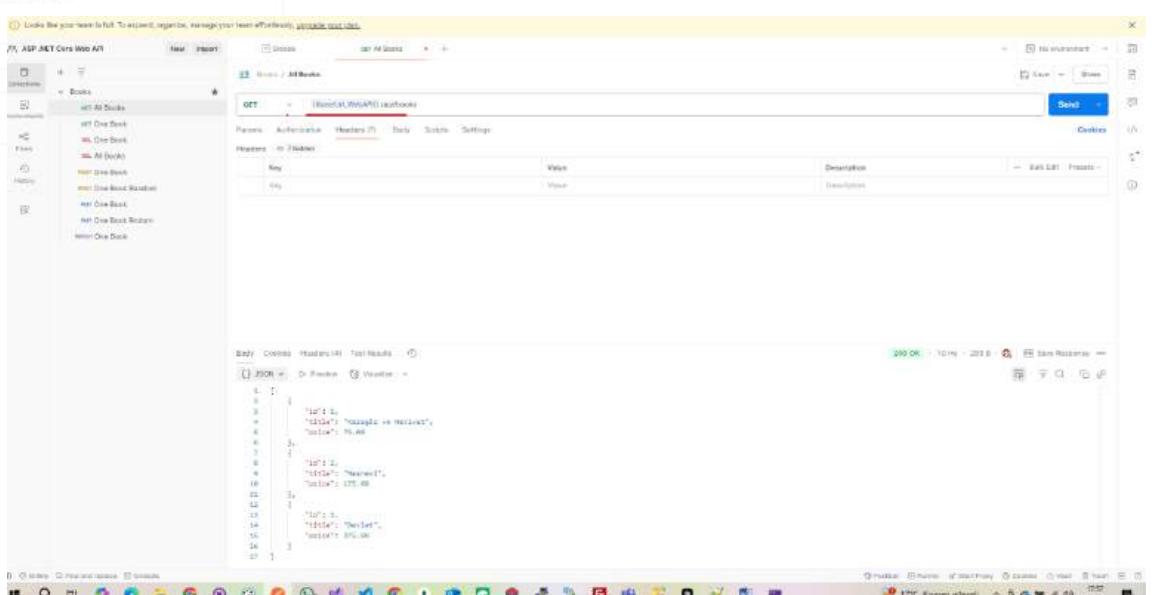
# Altıncı Bölüm?

EF Core

Proje Oluşturma / Inversion of Control (IoC)



The screenshot shows the 'Environment Variables' section of the VSTS interface. It displays two global variables: 'baseurl' (Type: default, Initial value: https://localhost:7298, Current value: https://localhost:7298) and 'baseUrl\_WebAPI' (Type: default, Initial value: https://localhost:7141, Current value: https://localhost:7141). The 'baseurl' variable is highlighted with a red border.



The screenshot shows a POSTMAN collection interface. A GET request is made to 'https://localhost:7298/api/books'. The response body is a JSON array of books:

```
[{"id": 1, "title": "Nineteen Eighty-Four", "author": "George Orwell", "price": 10.99}, {"id": 2, "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "price": 12.99}, {"id": 3, "title": "War and Peace", "author": "Leo Tolstoy", "price": 14.99}, {"id": 4, "title": "Pride and Prejudice", "author": "Jane Austen", "price": 9.99}, {"id": 5, "title": "To Kill a Mockingbird", "author": "Harper Lee", "price": 8.99}]
```

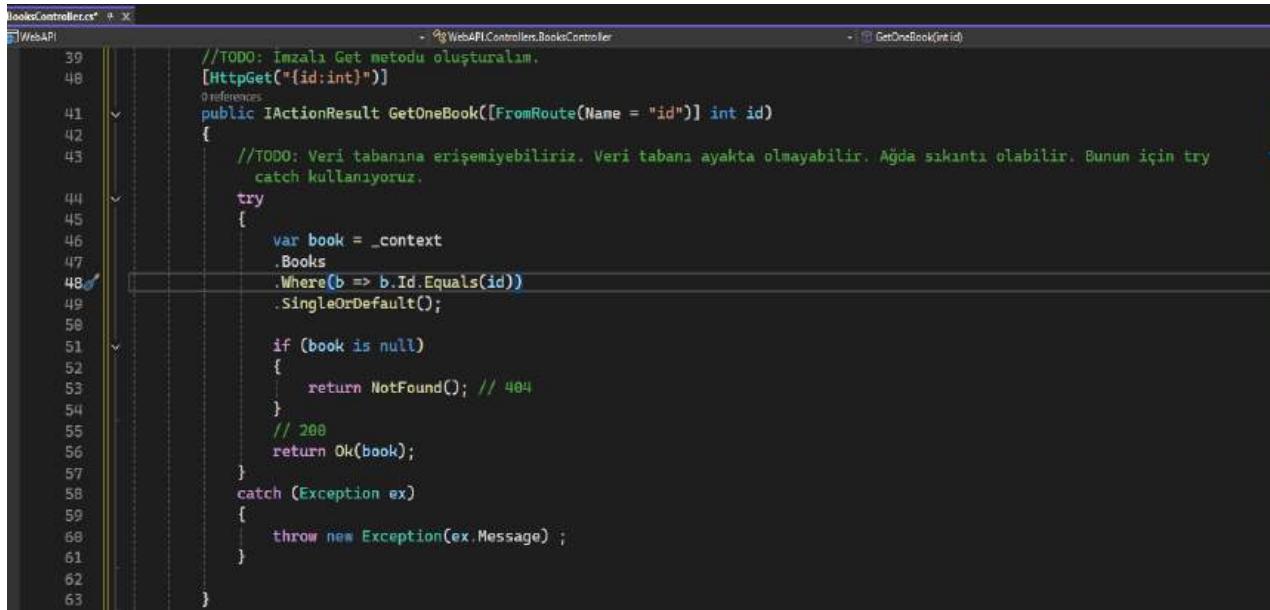
Uygulama çalışmayacak.  
Nedeni, eski projenin çalıştığı port ile bu uygulamanın çalıştığı port farklı

Gizlilik Derecesi [Tasnif Dışı](#)

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Veri Manipülasyonu



```
BooksController.cs
39 //TODO: imzalı Get metodunu oluşturalım.
40 [HttpGet("{id:int}")]
41 public IActionResult GetOneBook([FromRoute(Name = "id")] int id)
42 {
43     //TODO: Veri tabanına erişemeyebiliriz. Veri tabanı ayakta olmayı bilir. Ağda sıkıntı olabilir. Bunun için try
44     //catch kullanıyoruz.
45     try
46     {
47         var book = _context
48             .Books
49             .Where(b => b.Id.Equals(id))
50             .SingleOrDefault();
51
52         if (book is null)
53         {
54             return NotFound(); // 404
55         }
56         // 200
57         return Ok(book);
58     }
59     catch (Exception ex)
60     {
61         throw new Exception(ex.Message);
62     }
63 }
```

Şimdi **HttpGet GetOneBook** apisini yazalım?

Uygulamayı çalıştırılarım yazdığımız api yi test edelim.

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Veri Manipülasyonu

```
//TODO: Post metodu oluşturalım.  
[HttpPost]  
0 references  
public IActionResult CreateOneBook([FromBody] Book book)  
{  
    try  
    {  
        if (book is null)  
        {  
            return BadRequest(); // 400  
        }  
        _context.Books.Add(book);  
        _context.SaveChanges();  
        return StatusCode(201, book); // 201  
    }  
    catch (Exception ex)  
    {  
  
        return BadRequest(ex.Message); // 400  
    }  
}
```

Şimdi **HttpPost CreateOneBook** apisini yazalım?

Uygulamayı çalıştırılarım yazdığımız api yi test edelim.

Ancak bu api yi Postman'de **One Book Random** ile tetikleyelim.

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Veri Manipülasyonu

```
//TODO: Put metodu oluşturalım. (Güncelleme yapmak için)
[HttpPut("{id:int}")]
0 references
public IActionResult UpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] Book book)
{
    try
    {
        //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
        var bookToUpdate = _context
            .Books
            .Where(b => b.Id.Equals(id))
            .SingleOrDefault();
        if (bookToUpdate is null)
            return NotFound(); // 404
        //TODO: Gelen Id ile parametredeki Id aynı mı bakıyoruz.
        if (id != book.Id)
            return BadRequest(); // 400

        bookToUpdate.Title = book.Title;
        bookToUpdate.Price = book.Price;

        _context.SaveChanges();
        return Ok(bookToUpdate); // 200
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

Şimdi **HttpPut UpdateOneBook** apisini yazalım?

Uygulamayı çalıştırılarım yazdığımız api yi hem swagger dan hemde postman dan test edelim.

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Veri Manipülasyonu

```
//TODO: Delete metodunu oluşturalım. (Belirtilen id'yi sil)
[HttpDelete("{id:int}")]
0 references
public IActionResult DeleteOneBook([FromRoute(Name = "id")] int id)
{
    try
    {
        var bookToDelete = _context
            .Books
            .Where(b => b.Id.Equals(id))
            .SingleOrDefault();

        if (bookToDelete is null)
            return NotFound(new { message = $"Silinecek kitap id: {id} bulunamadı." });

        StatusCode = 404;
        message = $"Silinecek kitap id: {id} bulunamadı.";

    } // 404

    _context.Books.Remove(bookToDelete);
    _context.SaveChanges();
    return NoContent(); // 204
}
catch (Exception ex)
{
    throw new Exception(ex.Message);
}
```

Şimdi **HttpDelete DeleteOneBook** apisini yazalım?

Uygulamayı çalıştırıp yazdığımız api yi hem swagger dan hemde postman dan test edelim.

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Veri Manipülasyonu

Şimdi sıra sizde

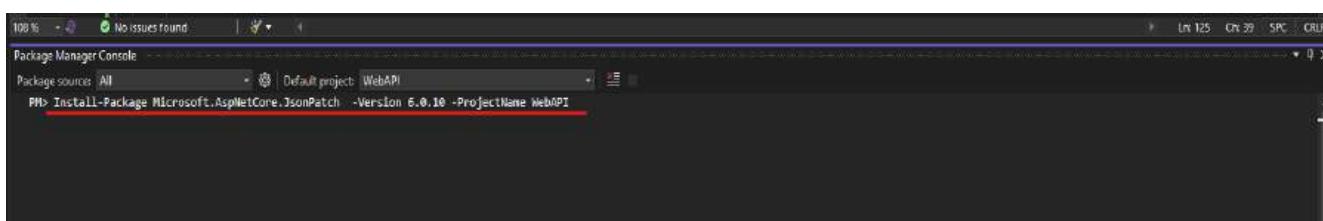
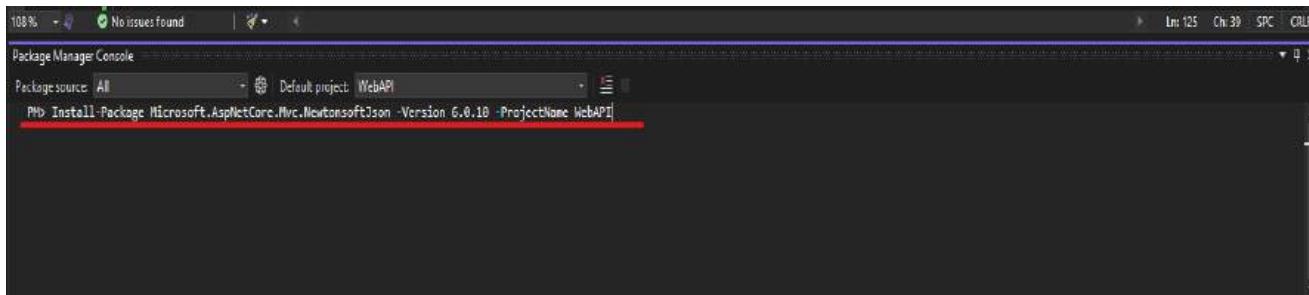
**HttpPatch** **PartiallyUpdateOneBook** apisini yazalım?

Uygulamayı çalıştıralım yazdığımız api yi hem swagger hemde postman dan test edelim.

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Veri Manipülasyonu



A screenshot of the Visual Studio code editor showing the 'Program.cs' file. The code in the 'ConfigureServices' method is as follows:

```
builder.Services.AddControllers()
    .AddNewtonsoftJson();
//TODO: NewtonsoftJson paketini kullanabilmek için gerekli ayarları yapalım.
```

Gizlilik Derecesi [Tasnif Dışı](#)

Uygulama api sini yazmak için,

-Microsoft.AspNetCore.Mvc.NewtonsoftJson 6.0.0  
-Microsoft.AspNetCore.JsonPatch 6.0.0 nuget paketlerine ihtiyaç vardır.

Bu paketlerden sonra Program.cs de ilgili konfigürasyona ihtiyaç vardır.

Slayt Numarası .../..

# Altıncı Bölüm?

EF Core

Proje Oluşturma / Veri Manipülasyonu

The screenshot shows a Visual Studio code editor with three tabs open. The active tab is 'BooksController.cs' under 'WebAPI'. The code implements the 'PartiallyUpdateOneBook' action method. It uses Entity Framework Core to find a book by its ID and apply a patch document to it. If the book is null, it returns a 404 error. Otherwise, it saves the changes and returns a 204 status.

```
149 //TODO: Put metodu oluşturalım.
150 [HttpPatch("{id:int}")]
151 public IActionResult PartiallyUpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] JsonPatchDocument<Book> bookPatch)
152 {
153     try
154     {
155         //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
156         var bookToUpdate = _context
157             .Books
158             .Where(b => b.Id.Equals(id))
159             .SingleOrDefault();
160
161         //TODO: Güncelleme yapmak için gelen book null mı kontrol et.
162         if (bookToUpdate == null)
163             return NotFound(); // 404
164
165         bookPatch.ApplyTo(bookToUpdate);
166         _context.SaveChanges();
167
168         return NoContent(); // 204
169     }
170     catch (Exception ex)
171     {
172
173         throw new Exception(ex.Message);
174     }
175 }
176 }
177 }
```

**HttpPatch** **PartiallyUpdateOneBook** apisini yazalım?

Uygulamayı çalıştırılarım yazdığınıza api yi hem swagger hemde postman dan test edelim.

**Altıncı Bölüm Bitti**  
**Hadi Özetleyelim?**

# Yedinci Bölüm?

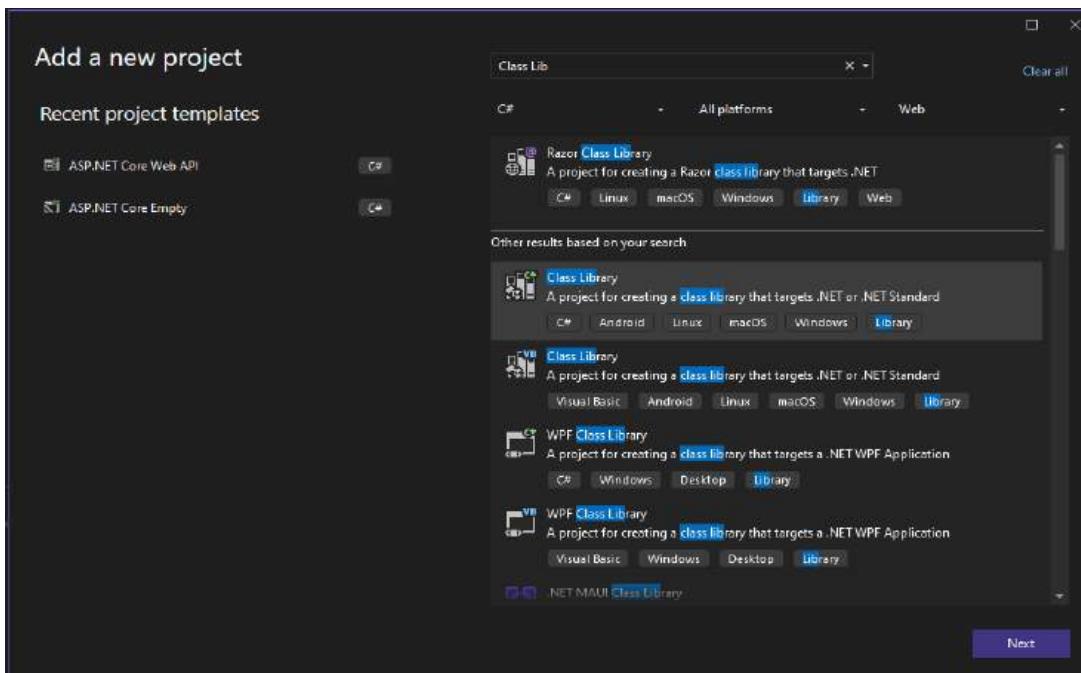
## Katmanlı Mimari

- Entites (Veri Tabanı Tablolarımız olacağı katman olacak)
- Repositories (Veri tabanı tabloların repo ları olacak)
- Repositories Context Factory( Migration yaptığımız bu dizayn path kullanacağız)
- Presentation Layer (Sunum Katmanı)
- Repository Base (Servis Sınıflarımız olacak)
- Repository Manager (Servis Katmanını yönetecek manager sınıfları olacak)
- Book Repository
- Lazy Loading
- Servis Layer (Servis Katmanı)
- Servis Manaeger
- Configure Services (Tüm bu yapıyı kontrol edeceğiz)

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Entites



*Şimdi WebAPI projemizi katmanlı mimariye dönüştürmeye başlayacağız.*

Projemize **new proje** diyerek **Class Library** ekleyeceğiz. ismi **Entities** olacak

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Entites

The screenshot shows the Visual Studio IDE interface. On the left is the Solution Explorer pane, which displays two projects: 'Entites' and 'WebAPI'. Under 'Entites', there is a 'Models' folder containing a 'Book.cs' file. Under 'WebAPI', there is a 'Books' folder containing 'BookController.cs', 'BookConfig.cs', 'BookContext.cs', and 'Program.cs'. In the center is the code editor window, showing the following C# code for the 'Book' class:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Entities.Models
8  {
9      //TODO: internal public yap.
10     public class Book
11     {
12         //TODO: WebAPI projemizdeki var olan Book class'ın propertileri buraya taşıyalım.
13         //TODO: Sonra WebAPI projemizdeki var olan Models klasörünü silelim.
14         public int Id { get; set; }
15         public String Title { get; set; }
16         public decimal Price { get; set; }
17     }
18 }
19
20 
```

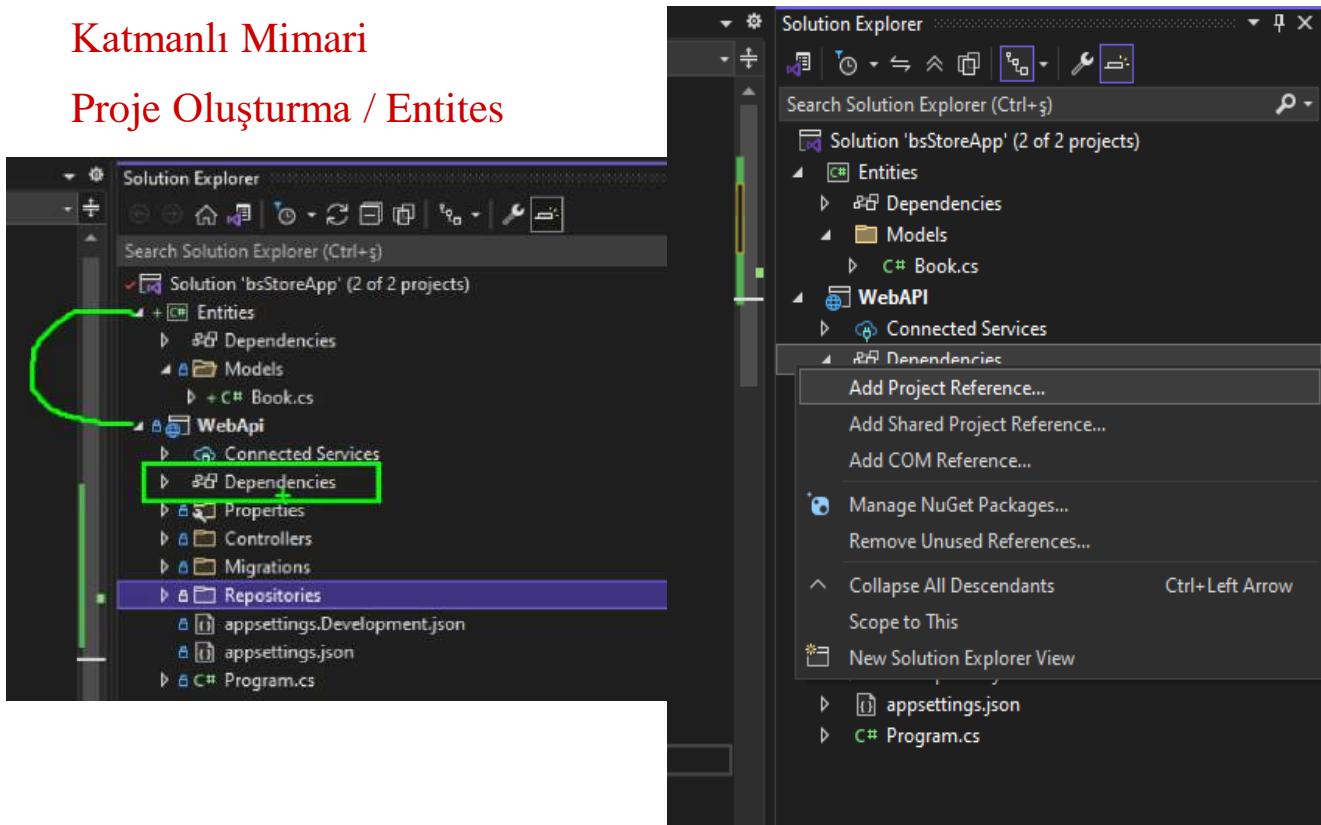
The code editor has syntax highlighting and some TODO comments. At the bottom of the screen, there is an Output window.

Class1 sil  
Models klasörünü oluştur.  
Sonra Book classını oluştur

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Entites



Şimdi bu iki proje arasında bağlantı yok.

Gerekli bağlantıyı Dependencies den sağlayalım

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Entites

```
Book.cs    BooksController.cs
1  using Microsoft.AspNetCore.Http;
2  using Microsoft.AspNetCore.JsonPatch;
3  using Microsoft.AspNetCore.Mvc;
4  using WebAPI.Models; CS0234: The type or namespace name 'Models' does not exist in the namespace 'WebAPI'. Are you missing an assembly reference?
5  using WebAPI.Repositories;
6
7  namespace WebAPI.Controllers
8  {
9      [Route("api/[controller]")]
10     [ApiController]
11     public class BooksController : ControllerBase
12     {
13         //TODO: İhtiyaçınız olan tüm kitapları listeleyin
14         //TODO: Aşağıdaki kodun sonuna gelip Ctrl + . tuşlarına basın. Generate constructor'i kullanın.
15         private readonly IRepository<Book> _repository;
16
17         public BooksController(IRepository<Book> repository)
18         {
19             _repository = repository;
20         }
21     }
22 }
```

```
Book.cs    BooksController.cs
1  //TODO: Yeni eklenen proje referansı
2  using Entities.Models;
3  using Microsoft.AspNetCore.Http;
4  using Microsoft.AspNetCore.JsonPatch;
5  using Microsoft.AspNetCore.Mvc;
6  using WebAPI.Repositories;
7
8  namespace WebAPI.Controllers
9  {
10     [Route("api/[controller]")]
11     [ApiController]
12     public class BooksController : ControllerBase
13     {
14         //TODO: İhtiyacınız olan tüm kitapları listeleyin
15         //TODO: Aşağıdaki kodun sonuna gelip Ctrl + . tuşlarına basın. Generate
16         //constructor'i kullanın.
17         private readonly RepositoryContext _context;
18
19         public BooksController(RepositoryContext context)
20         {
21             _context = context;
22         }
23     }
24 }
```

Uygulamayı derleyelim. Hataları görelim.

Bu hatalar neden?

Şimdi bu hataları giderelim.

Böyleye mevcut projedeki bir klasörü tamamen ayrı bir proje haline getirdik.

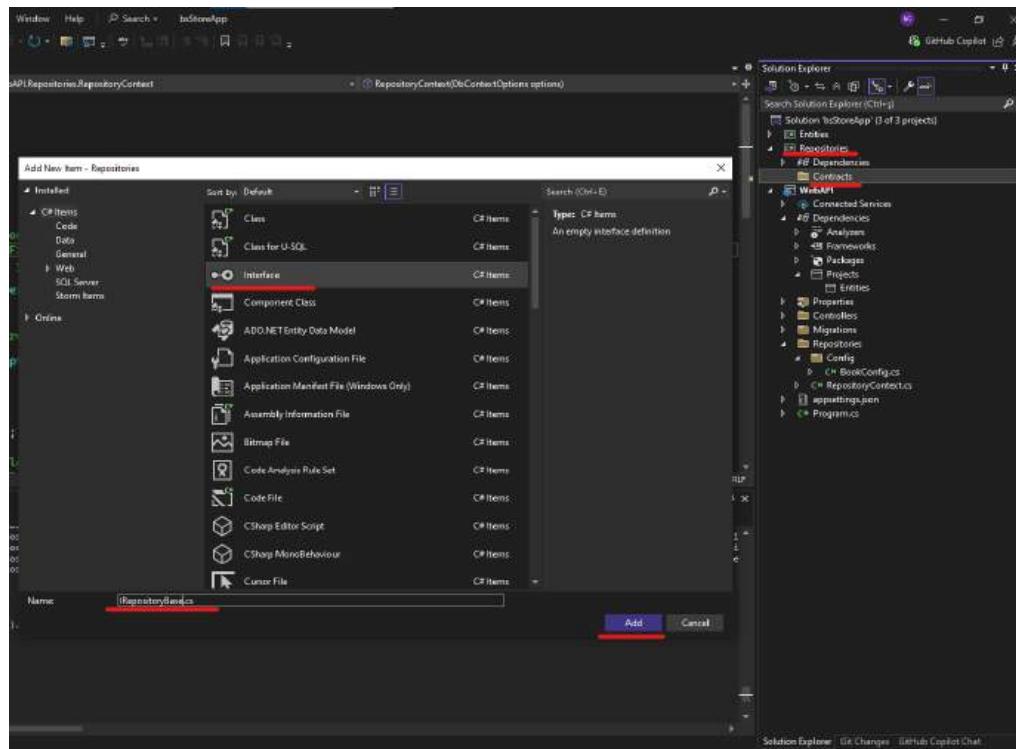
Uygulamayı çalıştırılarım **GetAllBooks** API'sinden sonuç geliyor mu görelim.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Repositories

IRepositoryBase



Projemize **new proje** diyerek Class Library ekleyeceğiz.

İsmi **Repositories** olacak.

Class1 sil

**Contracts** klasörünü oluştur.

Sonra **IRepositoryBase interfaces'ini** oluştur.

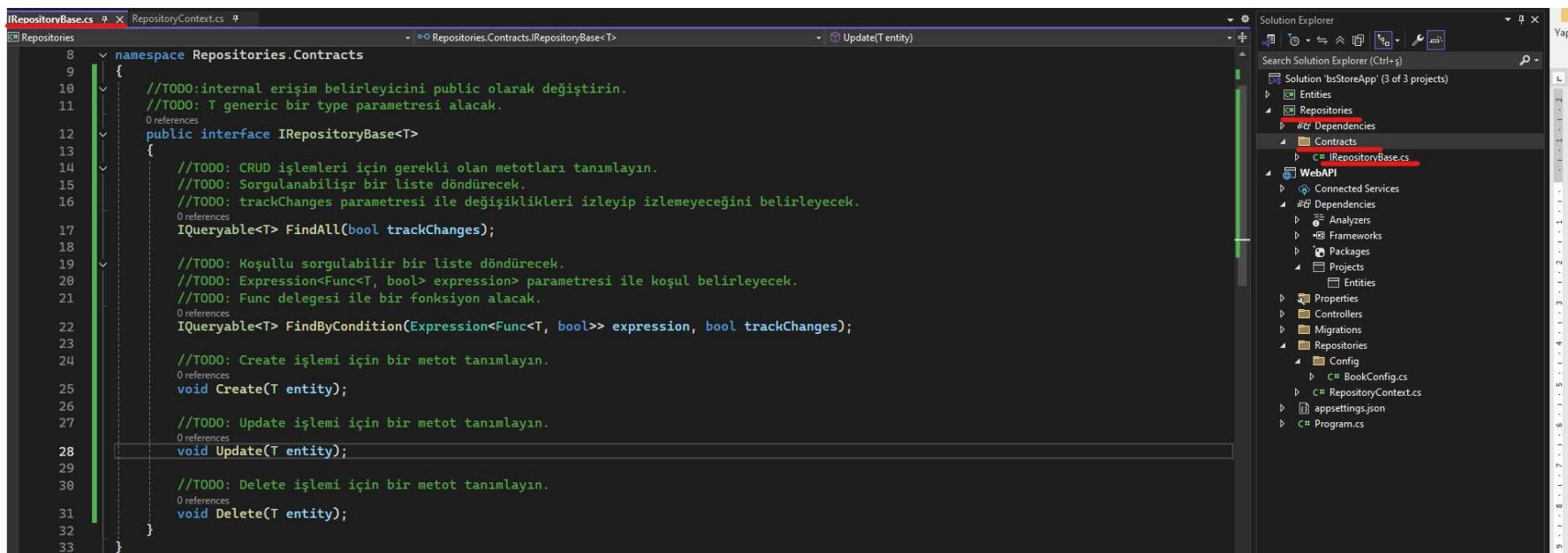
# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Repositories

IRepositoryBase

IRepositoryBase interfaces'ini  
resimdeki kod ile doldur.



The screenshot shows the Visual Studio IDE. On the left, the code editor displays the IRepositoryBase.cs file with C# code. The code defines a public interface IRepositoryBase<T> with methods like FindAll, FindByCondition, Create, Update, and Delete, each accompanied by TODO comments. On the right, the Solution Explorer pane shows the project structure for 'bsStoreApp' with three projects: Entities, Contracts, and WebAPI. The Contracts project contains the IRepositoryBase.cs file. The Solution Explorer also lists various files like BookConfig.cs, RepositoryContext.cs, appsettings.json, and Program.cs.

```
namespace Repositories.Contracts
{
    //TODO: internal erişim belirleyicini public olarak değiştirin.
    //TODO: T generic bir type parametresi alacak.
    public interface IRepositoryBase<T>
    {
        //TODO: CRUD işlemleri için gerekli olan metodları tanımlayın.
        //TODO: Sorgulanabilir bir liste döndürecek.
        //TODO: trackChanges parametresi ile değişiklikleri izleyip izlemeyeceğini belirleyecek.
        IQueryble<T> FindAll(bool trackChanges);

        //TODO: Koşullu sorgulayabilen bir liste döndürecek.
        //TODO: Expression<Func<T, bool> expression> parametresi ile koşul belirleyecek.
        //TODO: Func delegesi ile bir fonksiyon alacak.
        IQueryble<T> FindByCondition(Expression<Func<T, bool>> expression, bool trackChanges);

        //TODO: Create işlemi için bir metot tanımlayın.
        void Create(T entity);

        //TODO: Update işlemi için bir metot tanımlayın.
        void Update(T entity);

        //TODO: Delete işlemi için bir metot tanımlayın.
        void Delete(T entity);
    }
}
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Repositories

Repository Context

The screenshot shows the Visual Studio IDE. On the left, the code editor displays `RepositoryContext.cs` with C# code. The code includes several TODO comments related to Entity Framework Core configuration and context options. On the right, the Solution Explorer shows a solution named 'EntityApp' containing three projects: 'Entities', 'EFCore', and 'WebAPI'. The 'EFCore' project is expanded, showing its structure including 'Contracts', 'Config' (which contains `BookConfig.cs`), and 'RepositoryContexts'. The 'WebAPI' project also shows its structure with 'Entities', 'Properties', 'Controllers', and 'Migrations'.

```
11  namespace Repositories.EFCore
12  {
13      //TODO: internal erişim belirleyicini public olarak değiştirelim.
14      //TODO: WebAPI projenizdeki var olan Repositories/RepositoryContext class'ını buraya taşıyalım.
15      //TODO: Sonra WebAPI projenizdeki var olan Repositories/RepositoryContext class'ını silelim.
16
17      public class RepositoryContext : DbContext
18      {
19          //TODO: DbContextOptions nesnesi verdildiğimizde base(options) ile DbContext göndermiş oluyoruz.
20          //TODO: Install-Package Microsoft.EntityFrameworkCore -Version 6.0.10 -ProjectName Repositories !!! (Package Manager Console dan indir)
21          public RepositoryContext(DbContextOptions options) : base(options)
22          {
23          }
24
25          //TODO: Dependencies Add Project Reference/Entities projesini ekle.
26          public DbSet<Book> Books { get; set; }
27
28          //TODO: Tip konfigürasyonunu için ilgili kodu ekleyin.
29          //TODO: Artık Model oluştururken BookConfig sınıfını kullanacağız.
30          //TODO: Efcore/Config klasörünü oluştur.
31          //TODO: Altına BookConfig.cs dosyasını oluşturun.
32
33          protected override void OnModelCreating(ModelBuilder modelBuilder)
34          {
35              modelBuilder.ApplyConfiguration(new BookConfig());
36          }
37      }
}
```

EFCore klasörünü oluştur.

Sonra **RepositoryContext** classını oluşturacağız.

EFCore/Config klasörünü oluşturacağız.  
Altna **BookConfig** classını oluşturacağız.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Repositories

Repository Context

EFCore klasörünü oluştur.

Sonra **RepositoryContext** classını oluşturacağız.

**EFCore/Config** klasörünü oluşturacağız.  
Altna **BookConfig** classını oluşturacağız.

```
1  using Entities.Models;
2  using Microsoft.EntityFrameworkCore.Metadata.Builders;
3  using Microsoft.EntityFrameworkCore;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace Repositories.EfCore.Config
11 {
12     //TODO:IEntityTypeConfiguration dan kalıtım alıyoruz. Generic yapıda Book sınıfını alıyoruz.
13     public class BookConfig : IEntityTypeConfiguration<Book>
14     {
15         public void Configure(EntityTypeBuilder<Book> builder)
16         {
17             //TODO: Migrasyon işlemlerinde tabloda da veri yoksa çıraklık veriler ekleyeceğiz.
18             //TODO: WebAPI projemizdeki var olan Repositories/Config/BookConfig class'ını buraya taşıyalım. !!! !!! !!!
19             builder.HasData(
20                 new Book { Id = 1, Title = "Karagöz ve Hacivat", Price = 75 },
21                 new Book { Id = 2, Title = "Mesnəvi", Price = 175 },
22                 new Book { Id = 3, Title = "Devlet", Price = 375 }
23             );
24         }
25     }
}
```

Sonra Projeyi build et. Hatayı gider.  
Burada migration klasörü nüde sil !!!

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Repositories

The screenshot shows the Visual Studio IDE with the RepositoryBase.cs file open in the code editor. The file contains abstract repository logic for Entity Framework Core. The Solution Explorer on the right shows a multi-project solution named 'EfCoreApp' containing 'EfCore' and 'WebAPI' projects, along with their respective configurations and repository files.

```
RepositoryBase.cs
18 namespace Repositories.EfCore
19 {
20     //TODO: internal erişim belirleyicini public olarak değiştirelim.
21     //TODO: Bu class IRepositoryBase<T> sınıfını implement edecek.
22     //TODO: T generic bir type i, fadesi referans tipi bir ifadeidir(where T : class).
23     //TODO: Veri işlemleri için repository contextini ihtiyaçımız var.
24     //TODO: base bir class olduğu için classı abstract yapıyoruz. Abstract olmasa denemek RepositoryBase'in nesleneneyeceği anlamına gelir. Diğer repolar
25     //bunu kullanılabilecekler. İhtiyaçları olan diğer metotlarla tausulayabileceğiz.
26     //TODO: RepositoryBase classı abstract olduğu için diğer sınıfların _context'i kullanabilmesi için protected erişim belirleyicisi ile tanımlıyoruz.
27     //Bu sayede sadece bu classı kullanıbilecekler.
28     protected readonly RepositoryContext _context;
29     //TODO: readonly RepositoryContext _context;
30     public RepositoryBase(RepositoryContext context)
31     {
32         _context = context;
33     }
34     //TODO: Create<T entity> => _context.Set<T>().Add(entity); //TODO: _context'i kullanarak entity'i ekliyoruz. Set<T>() metod ile T tipinde bir
35     //DbSet alıyoruz. Add metod ile entity'i ekliyoruz. Burada SaveChanges metodu çağrılmadığı için değişiklikler kaydediliyor. Bu metod sadece eklemeye
36     //işlemi yapıyor. SaveChanges metodу eksik değil. Onuda Manager üzerinden kullanacağız.
37     public void Create(T entity) => _context.Set<T>().Add(entity);
38     //TODO: Delete<T entity> => _context.Set<T>().Remove(entity); //TODO: _context'i kullanarak entity'i siliyoruz. Set<T>() metod ile T tipinde bir
39     //DbSet alıyoruz.
40     public IQueryble<T> FindAll(bool trackChanges) =>
41         !trackChanges ? _context.Set<T>().AsNoTracking() :
42             _context.Set<T>(); //TODO: trackChanges parametresi ile değişiklikleri izleyip izlemeyecğini belirliyoruz. Eğer trackChanges true ise
43             AsNoTracking() metodlu ile değişiklikleri izlemiyoruz. Eğer false ise değişiklikleri izliyoruz. Set<T>() metod ile T tipinde bir DbSet
44             alıyoruz.
45     public IQueryble<T> FindByCondition(Expression<Func<T, bool> expression, bool trackChanges) =>
46         !trackChanges ? _context.Set<T>().Where(expression).AsNoTracking() :
47             _context.Set<T>().Where(expression); //TODO: Mısrulu sorulabilir bir liste döndürüyoruz. Expression<Func<T, bool>> expression parametresi ile
48             neşul belirliyoruz. Set<T>() metodlu ile T tipinde bir DbSet alıyoruz. Where metodlu ile neşul gäre filtreliyoruz. AsNoTracking() metodlu ile
49             değişiklikleri izliyoruz.
50     public void Update(T entity) => _context.Set<T>().Update(entity);
51     //TODO: Tüm CRUD işlemleri için base classı oluşturmuş olduk.
52 }
```

Gizlilik Derecesi [Tasnif Dışı](#)

IRepositoryBase implement edecek

\*\*\* !!! ????.

Slayt Numarası .../..

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Repositories  
Repository Base

Book Repository,

Contracts altına IBookRepository  
ekliyoruz

The screenshot shows a Visual Studio interface. On the left, the Solution Explorer displays a solution named 'bStoreApp' containing three projects: Entities, Repositories, and WebAPI. In the middle, a code editor window shows the file 'IBookRepository.cs' with the following content:

```
1  using Entities.Models;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace Repositories.Contracts
9  {
10     //TODO:internal erişim belirleyicini public olarak değiştirin.
11     //TODO:Burada IRepositoryBase<Book> referans alacağız.
12     public interface IBookRepository : IRepositoryBase<Book>
13     {
14         //TODO: IRepositoryBase deki CRUD işlemlerinden başka bir işlem yapmayacağımızdan dolayı sade bırakıyoruz.
15     }
16 }
17
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Repositories

Repository Base

Şimdi EFCore altına  
Book Repository classı eklenecek.

The screenshot shows the Visual Studio IDE. On the left, the code editor displays `BookRepository.cs` with the following content:

```
1  using Entities.Models;
2  using Entities.Contracts;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace Repositories.EfCore
10 {
11     //TODO:internal erişim belirleyicini public olarak değiştirin.
12     //TODO:IRepositoryBase<T> için oluşturulan RepositoryBase<T> classını kalıtım alındı.
13     //TODO:RepositoryBase<T> dışında yapılacak işlemler için IBookRepository interface'ini implement ettik.
14     //TODO:CH'de bir class birden fazla interface'i implement edebilir. Ama bir class sadece bir class'ı kalıtım alabilir. Bu
15     //      yuzden RepositoryBase<T> classını kalıtım alındı.
16     public class BookRepository : RepositoryBase<Book>, IBookRepository //Ctrl + .
17     {
18         //TODO:Constructor'da RepositoryContext'i istiyor. Bu RepositoryContext'i base gönderiyor. Bu base de RepositoryBase
19         //      oluyor. RepositoryBase gittiğimizde _context'i kullanıyor.
20         public BookRepository(RepositoryContext context) : base(context)
21     }
22 }
```

On the right, the Solution Explorer shows the project structure for `toStoreApp`, which contains three projects: `Entities`, `Repositories`, and `EFCore`. The `EFCore` project includes files like `BookConfig.cs`, `BookRepository.cs`, `RepositoryBase.cs`, and `RepositoryContext.cs`.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Repositories

Repository Base

Şimdi IBookRepository interfacesine yeni metotlar ekleyerek kolayca implement edildiğini görelim.

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays `BookRepository.cs` with the following content:

```
1  using Entities.Models;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace Repositories.Contracts
9  {
10     //TODO: internal erişim belirleyicini public olarak değiştirelim.
11     //TODO: Burada IRepositoryBase<Book> referans alacağız.
12     public interface IBookRepository : IRepositoryBase<Book>
13     {
14         //TODO: IRepositoryBase deki CRUD işlemlerinin之外 başka bir işlem yapmayacağımızdan dolayı sade bırakıyoruz.
15         //!!!
16         //TODO: Kitap oluşturmanın mantığı değişebilir düşününcesiyle burada bir metotlar tanımlıyoruz.
17         //TODO: Ana interface'den kopuk.
18         IQueryable<Book> GetAllBooks(bool trackChanges);
19         IQueryable<Book> GetOneBookById(int id, bool trackChanges);
20         void CreateBook(Book book);
21         void UpdateBook(Book book);
22         void DeleteBook(Book book);
23
24
25
26
27     }
28 }
```

On the right, the Solution Explorer shows the project structure for 'bsStoreApp' (3 of 3 projects):

- Entities
- Repositories
  - BD Dependencies
  - Contracts
    - BookRepository.cs
    - IRepositoryBase.cs
  - BCore
    - Config
      - BookConfigs.cs
      - BookRepository.cs
      - RepositoryBase.cs
      - RepositoryContext.cs
  - WebAPI

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Repositories  
Repository Manager

The screenshot shows the Visual Studio IDE. On the left, the code editor displays the file `Repositories.Contracts.RepositoryManager.cs` with the following content:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Repositories.Contracts
8  {
9      //TODO: internal erişim belirleyicini public olarak değiştirin.
10     public interface IRepositoryManger
11     {
12         //TODO: Tüm repolara Manager üzerinden erişim sağlayacağız.
13         IBookRepository Book { get; }
14         void Save();
15     }
16 }
17
```

On the right, the Solution Explorer shows a solution named "BookstoreApp" containing three projects: Entity, API Dependencies, and WebAPI. The Entity project contains a folder named Contracts with files: BaseRepository.cs, IRepositoryContracts.cs, and IRepositoryManager.cs. The API Dependencies project contains a Config folder with BookConfig.cs and a WebAPI project.

Contracts altına **IRepositoryManger** ekliyoruz

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Repositories  
Repository Manager

Şimdi bunu BookRepository impletant edelim.

The screenshot shows the Visual Studio IDE with the BookRepository.cs file open in the code editor. The code implements the IBookRepository interface using Entity Framework Core. The Solution Explorer on the right shows the project structure with entities, repositories, contracts, and EFCore configurations.

```
namespace Repositories.EfCore
{
    public class BookRepository : RepositoryBase<Book>, IBookRepository
    {
        public BookRepository(RepositoryContext context) : base(context)
        {
        }

        public void CreateBook(Book book) => Create(book);

        public void DeleteBook(Book book) => Delete(book);

        public IQueryable<Book> GetAllBooks(bool trackChanges) =>
            FindAll(trackChanges);

        public IQueryable<Book> GetOneBookById(int id, bool trackChanges) =>
            FindByCondition(x => x.Id.Equals(id), trackChanges);

        public void UpdateBook(Book book) => Update(book);
    }
}
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Repositories  
Repository Manager

Şimdi IRepositoryManger implant edecek yapıyı kuracağız..

The screenshot shows the Visual Studio IDE. On the left, the code editor displays `RepositoryManager.cs` with the following content:

```
namespace Repositories.EfCore
{
    //TODO: internal erişim belirleyicini public olarak değiştirin.
    1 reference
    public class RepositoryManager : IRepositoryManger
    {
        private readonly RepositoryContext _context;

        0 references
        public RepositoryManager(RepositoryContext context)
        {
            _context = context;
        }

        //TODO: Her bir repo yu IoC kaydını kaymak istemiyoruz
        //TODO: Sadece RepositoryManager ve IRepositoryManger IoC ye ekleyeceğiz.
        //TODO: Bunun için diğer repoları burada now leyeceliz. (istisnai bir durum yapacağız.)

        //TODO: !!! Bir class altında başka bir class new lenmez. New lenirse sıkı bir bağılılık olur.
        //TODO: İstesemiz IBookRepository'i BookRepository ve diğer Repository'leri ek tek IoC kaydır edebiliriz.
        //TODO: Tüm Repository IoC yüksarsak Autofac gibi bir uygulama ile yönetebiliriz. Ancak biz genelde microsoftta yüklemek
        //      için böyle bir yol izliyoruz.

        1 reference
        public IBookRepository Book => new BookRepository(_context);

        //TODO: Book ifadesi artık BookRepository karşılık gelecek.

        1 reference
        public void Save()
        {
            _context.SaveChanges();
        }
    }
}
```

On the right, the Solution Explorer shows the project structure:

- Solution: InTercerApp (0 of 3 projects)
  - Entities
  - Repositories
    - EF Dependencies
    - Contact
    - C# BookRepository.cs
    - C# IRepositoryBases.cs
    - C# RepositoryManager.cs
  - EF Core
    - Config
      - C# BookConfig.cs
      - C# BookRepository.cs
      - C# IRepositoryBases.cs
      - C# RepositoryContext.cs
      - C# RepositoryManager.cs
  - WebAPI

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari  
Lazy Loading

**Eager Loading :** Bir nesneye ihtiyacımız olduğunda o nesneyi ve o nesneye ait her şeyi bir defa yüklemek istediğimizde

**Lazy Loading :** Ne zaman ihtiyaç olduğu nesne üretilir. Gereksiz Kaynak kullanımından kaçınmış oluruz.

The screenshot shows the Visual Studio IDE. On the left, the code editor displays `RepositoryManager.cs` with C# code for a repository manager. The code includes a constructor that initializes a `_bookRepository` using `new Lazy<IBookRepository>(() => new BookRepository(_context))`. The code is annotated with several TODO comments. On the right, the Solution Explorer shows a solution named `IsThereApp` containing three projects: `Entities`, `Repositories`, and `EFCore`. The `EFCore` project contains a `Config` folder with configuration files like `BookConfig.cs` and `BookRepository.cs`.

```
namespace Repositories.EfCore
{
    //TODO: internal erişim belirleyicini public olarak değiştirin.
    public class RepositoryManager : IRepositoryManager
    {
        private readonly RepositoryContext _context;
        //TODO: Lazy Loading için ekledik.
        private readonly Lazy<IBookRepository> _bookRepository;

        public RepositoryManager(RepositoryContext context)
        {
            _context = context;
            _bookRepository = new Lazy<IBookRepository>(() => new BookRepository(_context));
        }

        // TODO: Her bir repo yu IoC kaydını kaymak istemiyoruz ...
        public IBookRepository Book => _bookRepository.Value; //Nesne ancak ve ancak kullanıldığında oluşturulacak. Lazy Loading yapmış olduk.

        //TODO: Book ifadesi artık BookRepository karşılık gelecek.
    }
}
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Servis Extensions

The screenshot shows the Visual Studio IDE interface. On the left is the code editor with the file `ServicesExtensions.cs` open. The code defines a static class `ServicesExtensions` with a single method `ConfigureDbContext`. This method is annotated with several TODO comments describing its purpose: adding a DbContext, using RepositoryContext, using Microsoft.EntityFrameworkCore.SqlServer NuGet package, setting up appsettings.json, and adding IOC configuration. Below this, there's a comment about copying the code from Program.cs. The code then adds a `RepositoryContext` to the options and uses SQL Server for the connection string. The Solution Explorer on the right shows a solution named `b5StoreApp` containing three projects: `Entities`, `Repositories`, and `WebAPI`. The `WebAPI` project is selected, showing files like `BookConfig.cs`, `BookRepository.cs`, `RepositoryBase.cs`, `RepositoryManager.cs`, `Program.cs`, and `appsettings.json`. The `Extensions` folder under `WebAPI` contains the `ServicesExtensions.cs` file.

```
1  using Microsoft.EntityFrameworkCore;
2  using Repositories.EfCore;
3
4  namespace WebAPI.Extensions
5  {
6      public static class ServicesExtensions
7      {
8          //TODO: IServiceCollection services genişletmek için extension method
9          public static void ConfigureDbContext(this IServiceCollection services, IConfiguration configuration)
10         {
11             //TODO: Veri Tabrı işlemi için AddDbContext eklenir. Buna da ilgili context sınıfı verilir.
12             //TODO: RepositoryContext sınıfını kullanarak DbContext'i ekleyin.
13             //TODO: Microsoft.EntityFrameworkCore.SqlServer NuGet paketini yükleyin.
14             //TODO: SqlServer için gerekli olan bağlantı dizesini appsettings.json a ekledik.
15             //TODO: IOC ye DBContext tanımlını yapmış oluyoruz. (Bir DBContexte ihtiyacımız olduğunda bunun somut haline
16             //ulaşmamız için)
17
18             //TODO: Program.cs dosyasında ctrl +x ile aldık.
19
20             services.AddDbContext<RepositoryContext>(options =>
21                 options.UseSqlServer(configuration.GetConnectionString("sqlConnection")));
22         }
23     }
}
```

# Yedinci Bölüm?

## Katmanlı Mimari

# Proje Oluşturma / Katmanlı Mimari Servis Extensions

```
ServiceExtensions.cs  Program.cs  x
WebAPI
8     // Add services to the container.
9
10    builder.Services.AddControllers()
11        .AddNewtonsoftJson();
12    //TODO: NewtonsoftJson paketini kullanabilmek için gerekli ayarları yapalım.
13
14    // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
15    builder.Services.AddEndpointsApiExplorer();
16    builder.Services.AddSwaggerGen();
17
18
19    //TODO: Program.cs kendi yazdığımız classı ekliyoruz.
20    //TODO: ServicesExtensions db bağlığı taşıdığı için,
21    builder.Services.ConfigureSqlContext(builder.Configuration);
22
23
24
25
26    var app = builder.Build();
27
28    // Configure the HTTP request pipeline.
29    if (app.Environment.IsDevelopment())
30    {
31        app.UseSwagger();
32        app.UseSwaggerUI();
33    }
34
35    app.UseHttpsRedirection();
36    app.UseStaticFiles();
37
38    app.UseRouting();
39
40    app.UseAuthorization();
41
42    app.MapControllers();
43
44    app.Run();
45}
```

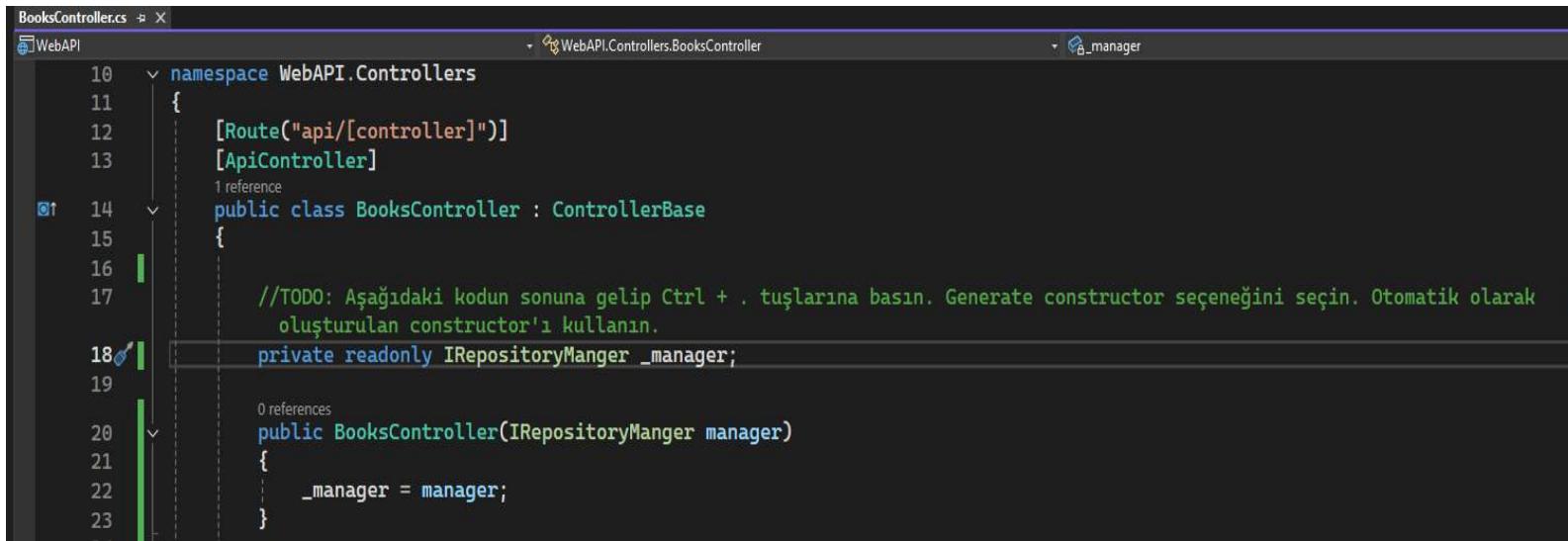
## Güncel Program.cs

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Configure Repository Manager



The screenshot shows the code editor with the file `BooksController.cs` open. The code defines a `BooksController` class that inherits from `ControllerBase`. It includes annotations for `[Route("api/[controller]")]` and `[ApiController]`. A TODO comment at the top of the body suggests generating a constructor for the `_manager` field. The code editor highlights the `_manager` field with a red underline, indicating a potential error or warning.

```
BooksController.cs
10  namespace WebAPI.Controllers
11  {
12      [Route("api/[controller]")]
13      [ApiController]
14      public class BooksController : ControllerBase
15      {
16
17          //TODO: Aşağıdaki kodun sonuna gelip Ctrl + . tuşlarına basın. Generate constructor seçeneğini seçin. Otomatik olarak
18          //olusturulan constructor'i kullanın.
19          private readonly IRepositoryManger _manager;
20
21          public BooksController(IRepositoryManger manager)
22          {
23              _manager = manager;
24          }
25      }
26  }
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Configure Repository Manager

Yeni Hali

Eski Hali

```
//TODO: GetAllBooks metodu ile tüm kitapları listeleyin
[HttpGet]
0 references
public IActionResult GetAllBooks()
{
    try
    {
        var books = _context.Books.ToList(); ✖ CSB103: The name '_context' does not exist in the current context
        return Ok(books);
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

```
//TODO: GetAllBooks metodu ile tüm kitapları listeleyin
[HttpGet]
0 references
public IActionResult GetAllBooks()
{
    try
    {
        var books = _manager.Book.GetAllBooks(false);
        return Ok(books);
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

# **Yedinci Bölüm?**

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Configure Repository Manager

**HADİ DİĞER METOTLARIDA SİZ DÖNÜŞTÜRÜN !!!!**

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Configure Repository Manager

Yeni Hali

```
//TODO: GetAllBooks metodu ile tüm kitapları listeleyin
[HttpGet]
0 references
public IActionResult GetAllBooks()
{
    try
    {
        var books = _manager.Book.GetAllBooks(false);
        return Ok(books);
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

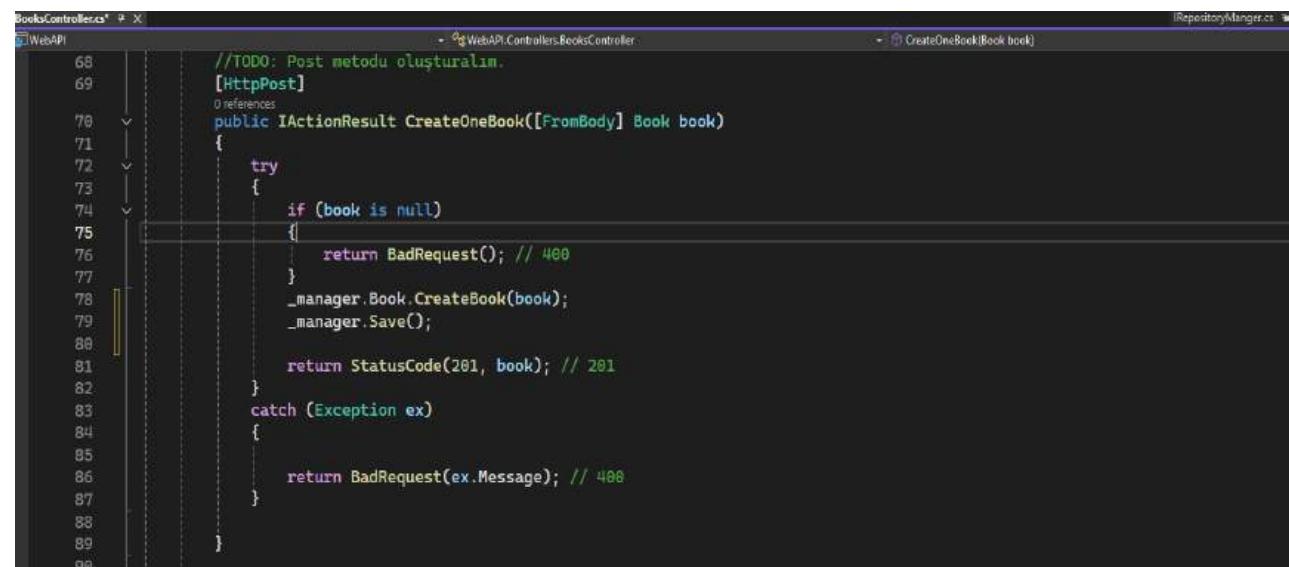
# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Configure Repository Manager

Yeni Hali



```
BooksController.cs # X
[WebAPI]
//TODO: Post metodu oluşturalım.
[HttpPost]
public IActionResult CreateOneBook([FromBody] Book book)
{
    try
    {
        if (book is null)
        {
            return BadRequest(); // 400
        }
        _manager.Book.CreateBook(book);
        _manager.Save();

        return StatusCode(201, book); // 201
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message); // 400
    }
}
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Configure Repository Manager

Yeni Hali

```
//TODO: Put metodu oluşturalım. (Güncelleme yapmak için)
[HttpPut("{id:int}")]
0 references
public IActionResult UpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] Book book)
{
    try
    {
        //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
        var bookToUpdate = _manager
            .Book
            .GetOneBookById(id, false);

        if (bookToUpdate is null)
            return NotFound(); // 404
        //TODO: Gelen Id ile parametredeki Id aynı mı bakıyoruz.
        if (id != book.Id)
            return BadRequest(); // 400

        bookToUpdate.Title = book.Title;
        bookToUpdate.Price = book.Price;

        _manager.Save();
        return Ok(bookToUpdate); // 200
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Configure Repository Manager

Yeni Hali

The screenshot shows a code editor with two tabs open. The left tab is 'BooksController.cs' and the right tab is 'RepositoryManager.cs'. Both tabs are part of a project named 'WebAPI'. The code in 'BooksController.cs' is for a 'DeleteOneBook' action, which uses a repository manager to delete a book by its ID. The code in 'RepositoryManager.cs' defines a 'DeleteBook' method that takes a book object and performs the deletion. The code is annotated with TODO comments and exception handling.

```
BooksController.cs
121 //TODO: Delete metodu oluşturalım. (Belirtileni sil)
122 [HttpDelete("{id:int}")]
123 public IActionResult DeleteOneBook([FromRoute(Name = "id")] int id)
124 {
125     try
126     {
127         var bookToDelete = _manager
128             .Book
129             .GetOneBookById(id, false);
130
131         if (bookToDelete == null)
132             return NotFound(new
133             {
134                 StatusCode = 404,
135                 message = $"Silinecek kitabı id: {id} bulunamadı."
136             });
137
138         _manager.Book.DeleteBook(bookToDelete);
139         _manager.Save();
140         return NoContent(); // 204
141     }
142     catch (Exception ex)
143     {
144
145         throw new Exception(ex.Message);
146     }
147 }
148 }
```

```
RepositoryManager.cs
1 DeleteOneBook(int id)
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Configure Repository Manager

Yeni Hali

```
//TODO: Put metodu oluşturalım.
[HttpPatch("{id:int}")]
public IActionResult PartiallyUpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] JsonPatchDocument<Book>
    bookPatch)
{
    try
    {
        //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
        var bookToUpdate = _manager
            .Book
            .GetOneBookById(id, true);

        //TODO: Güncelleme yapmak için gelen book null mı kontrol et.
        if (bookToUpdate is null)
            return NotFound(); // 404

        bookPatch.ApplyTo(bookToUpdate);
        _manager.Book.Update(bookToUpdate);
        //_manager.SaveChanges();

        return NoContent(); // 204
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Configure Repository Manager

Çalıştır !!!

GetAllBooks API'sini çalıştırıldım aşağıdaki hatayı aldık ...

System.InvalidOperationException: Unable to resolve service  
for type 'Repositories.Contracts.IRepositoryManger' while  
attempting to activate 'WebAPI.Controllers.BooksController'

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Configure Repository Manager

The screenshot shows the Visual Studio IDE with three open windows:

- Solution Explorer:** Shows the project structure for "Solution TestApp" (3 of 3 projects). It includes Entity, Repositories, Dependencies, Contracts, EFCore, WebAPI, and Extensions.
- ServiceExtensions.cs:** Contains code for configuring services. It includes methods for `ConfigureDbContext` and `ConfigureRepositoryManager`.
- Program.cs:** Contains code for building the application and configuring Swagger/OpenAPI.

```
public static class ServiceExtensions
{
    //TODO: IServiceCollection services genişletmek için extension method
    public static void ConfigureDbContext(this IServiceCollection services, IConfiguration configuration)
    {
        //TODO: Veri Tabrı işlemi için AddDbContext eklenir. Bu da ilgili context sınıfı verilir.
        //TODO: RepositoryContext sınıfını kullanarak DbContext'i ekleyin.
        //TODO: Microsoft.EntityFrameworkCore.SqlServer NuGet paketini yükleyin.
        //TODO:SqlServer için gereklili olan bağlantı dizesini appsettings.json'a ekledik.
        //TODO: IOC ye DbContext tanımını yapmış oluyoruz. (Bir DbContexte ihtiyacımız olduğunda bunun somut haline ulaşmanız için)

        //TODO: Program.cs dosyasında ctrl + x ile alındı.

        services.AddDbContext<RepositoryContext>(options =>
            options.UseSqlServer(configuration.GetConnectionString("sqlConnection")));
    }

    //TODO: Hata alındı onun için yeni servis ekleddik.
    public static void ConfigureRepositoryManager(this IServiceCollection services) =>
        services.AddScoped<IRepositoryManager, RepositoryManager>();
    }
}

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

//TODO: Program.cs kendi yardımımız sınıfı ekliyoruz.
//TODO: ServicesExtensions db väյlantısı taşıdığı için,
builder.Services.ConfigureDbContext(builder.Configuration);

//TODO: Parametre kullanmadık. Çünkü ConfigureRepositoryManager metodu sadece this aldı.
builder.Services.ConfigureRepositoryManager();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

Çözüm !!!

Veriler geldi.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / IBookService

The screenshot shows the Visual Studio IDE interface. On the left, the Solution Explorer displays a solution named 'bsüderapp' containing four projects: Entities, Repositories, Services, and WebAPI. The Services project is expanded, showing its Contracts folder and the IBookService.cs file. On the right, the code editor window shows the implementation of the IBookService interface. The code includes methods for getting all books, getting one book by ID, creating a book, updating a book, and deleting a book. A TODO comment indicates that internal access should be changed to public.

```
using System;
using System.Threading.Tasks;

namespace Services.Contracts
{
    //TODO:internal erişim belirleyicini public olarak değiştirin.
    public interface IBookService
    {
        IEnumerable<Book> GetAllBooks(bool trackChanges);
        Book GetOneBookById(int id, bool trackChanges);
        Book CreateBook(Book book);
        void UpdateOneBook(int id, Book book, bool trackChange);
        void DeleteOneBook(int id, bool trackChanges);
    }
}
```

Services isimli Class Library oluşturduk.

Sonra Contracts klasörü oluşturduk.

İçine IBookService interface’ı oluşturduk.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / BookManager

The screenshot shows the Visual Studio IDE interface. On the left is the 'BookManager.cs' code editor window, which contains C# code for a service class. The code includes imports for System, System.Collections.Generic, System.Linq, System.Text, and System.Threading.Tasks. It defines a namespace 'Services' and a class 'BookManager' that implements 'IBookService'. The constructor takes an 'IRepositoryManger' parameter and assigns it to a field named '\_manger'. The 'CreateBook' method checks if the input book is null and throws an 'ArgumentNullException'. If not, it calls the 'CreateBook' method on '\_manger' and then saves the changes. The right side of the interface is the 'Solution Explorer' window, which displays the project structure for 'bsStoreApp' containing four projects: Entities, Repositories, EFCore, and Services. The 'Services' project is expanded to show its internal structure, including 'Contracts' (with 'IBookService.cs'), 'Config' (with 'BookConfig.cs'), 'Repositories' (with 'BookRepository.cs'), 'EFCore' (with 'BookContext.cs'), and 'WebAPI' (with 'BookManager.cs').

```
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace Services
11 {
12     //TODO:internal erişim belirleyicini public olarak değiştirin.
13     public class BookManager : IBookService
14     {
15         //TODO: Managerıhtiyaç vardır.
16         private readonly IRepositoryManger _manger;
17
18         public BookManager(IRepositoryManger repository)
19         {
20             _manger = repository;
21         }
22
23         public Book CreateBook(Book book)
24         {
25             if (book == null)
26                 throw new ArgumentNullException(nameof(book));
27
28             _manger.Book.CreateBook(book);
29             _manger.Save();
30             return book;
31         }
32     }
33 }
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / BookManager

```
public void DeleteOneBook(int id, bool trackChanges)
{
    //check entity
    var entity = _manger.Book.GetOneBookById(id, trackChanges);
    if (entity is null)
        throw new Exception($"Silinmek istenen id:{id} bulunamadı.");

    _manger.Book.DeleteBook(entity);
    _manger.Save();
}

1 reference
public IEnumerable<Book> GetAllBooks(bool trackChanges)
{
    return _manger.Book.GetAllBooks(trackChanges);
}

1 reference
public Book GetOneBookById(int id, bool trackChanges)
{
    return _manger.Book.GetOneBookById(id, trackChanges);
}
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / BookManager

```
1 reference
public void UpdateOneBook(int id, Book book, bool trackChanges)
{
    //check entity
    var entity = _manger.Book.GetOneBookById(id, trackChanges);
    if (entity is null)
        throw new Exception($"Silinmek istenenid:{id} bulunamadi.");

    //check book null
    if (book is null)
        throw new ArgumentNullException(nameof(book));

    entity.Title = book.Title;
    entity.Price = book.Price;

    _manger.Book.UpdateBook(entity);
    _manger.Save();
}
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / IServiceManager

The screenshot shows a Visual Studio interface. On the left, the code editor displays the `IServiceManager.cs` file with the following content:using System.Text;
using System.Threading.Tasks;

namespace Services.Contracts
{
 public interface IServiceManager
 {
 //TODO: internal erişim belirleyicini public olarak değiştirin.
 //TODO: Projemizde kullanacağımız servisleri burada tanımlayacağız. Servisleri tek bir manager ile yöneteceğiz.
 //TODO: Tüm servisleri burada tanımlayacağız.
 IBookService BookService { get; }
 }
}On the right, the Solution Explorer shows a solution named `IsStoreApp` containing four projects: Entities, Repositories, Services, and WebAPI.

**Contracts** klasörün içine **IServiceManager** interface’ı oluşturduk.

Burada yöneteceğimiz servisleri tek tek yazacağız.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Service Manager

The screenshot shows the Visual Studio IDE. On the left, the code editor displays the `ServiceManager.cs` file with the following content:

```
using System.Threading.Tasks;
namespace Services
{
    //TODO: internal erişim belirleyicini public olarak değiştirin.
    public class ServiceManager : IServiceProvider
    {
        //TODO: IRepositoryManger kullanarak repositoryManger üzerinden tüm Repository ve save metoduna erişim sağlayacağız.
        private readonly Lazy<IBookService> _bookService;
        public ServiceManager(IRepositoryManger repositoryManger)
        {
            _bookService = new Lazy<IBookService>(() => new BookManager(repositoryManger));
        }
        //TODO: İhtiyaç olduğunda new lenenecek şekilde Lazy<T> kullanıyoruz.
        public IBookService BookService => _bookService.Value;
    }
}
```

On the right, the Solution Explorer shows the project structure for "InStoreApp" (4 of 4 projects):

- Solution Explorer (InStoreApp)
  - Entities
  - Repositories
  - Services
    - Ad Dependencies
    - Contracts
      - BookService.cs
      - IBookManager.cs
      - BookManager.cs
      - ServiceManager.cs
    - WebAPI
    - Connected Services
    - Properties
    - Controllers
      - BooksController.cs
    - Extensions
      - ServiceExtension.cs
    - Program.cs

ServiceManager clasını oluştuek.

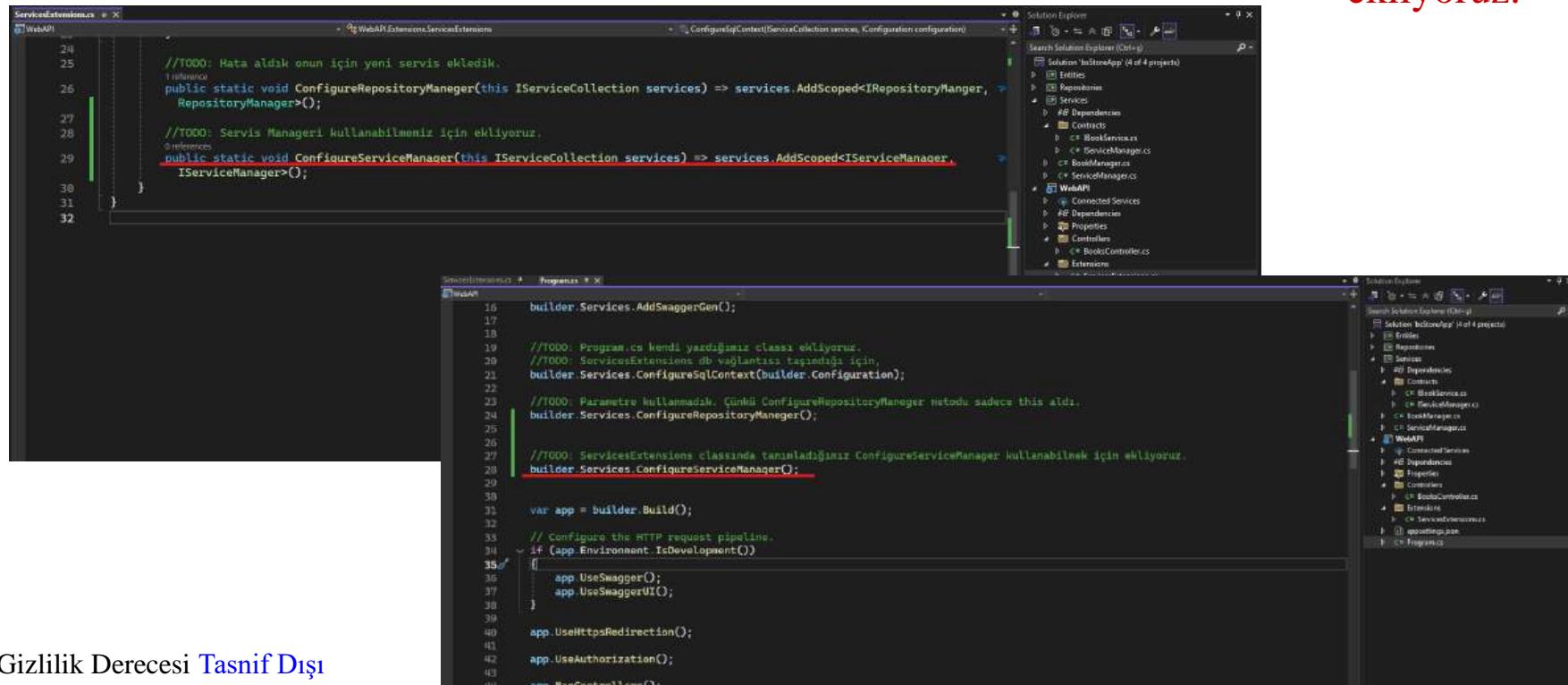
Bu servis manager'i kullanmak  
için ne yapmalıyız ???

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari  
Services / Service Manager IoC Kaydı

ServiceManager kullanmak için,  
ServicesExtensions classına ve  
Program.cs ye ilgili kod bloklarını  
ekliyoruz.



```
ServiceExtensions.cs
24
25
26
27
28
29
30
31
32

    //TODO: Hata alısk onun yeni servis ekledik.
    public static void ConfigureRepositoryManager(this IServiceCollection services) => services.AddScoped<IRepositoryManager, RepositoryManager>();
    //TODO: Servis Managerı Kullanabilmeniz için ekliyoruz.
    public static void ConfigureServicesManager(this IServiceCollection services) => services.AddScoped<IServiceManager, ServiceManager>();

    builder.Services.AddSwaggerGen();
    //TODO: Program.cs kendi yazdığımız classı ekliyoruz.
    //TODO: ServicesExtensions db bağlantısı taşıdığı için,
    builder.Services.ConfigureDbContext(builder.Configuration);
    //TODO: Parametre kullanmadık. Çünkü ConfigureRepositoryManager metodu sadece this aldı.
    builder.Services.ConfigureRepositoryManager();
    //TODO: ServicesExtensions classında tanımladığınız ConfigureServicesManager kullanabilecek için ekliyoruz.
    builder.Services.ConfigureServiceManager();

    var app = builder.Build();
    // Configure the HTTP request pipeline.
    if (app.Environment.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }
    app.UseHttpsRedirection();
    app.UseAuthorization();
    app.MapControllers();
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Service Manager IoC Kaydı

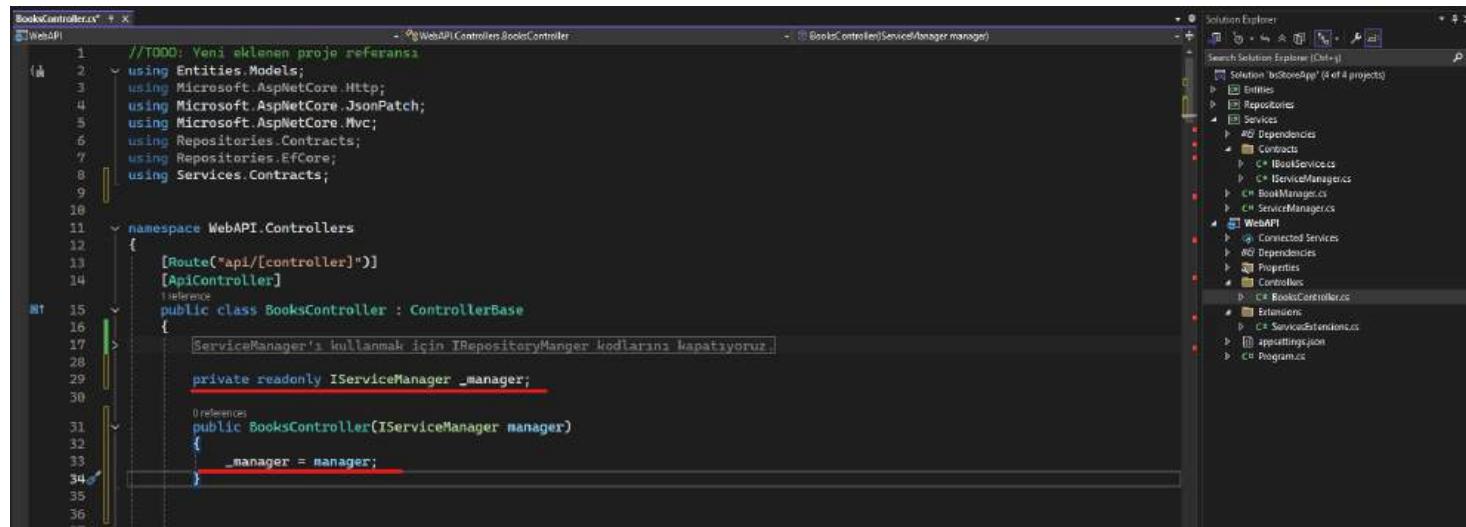
Uygulamamız Repository Manager kullanıyor.  
Services katmanını oluşturduk artık Services Managere çekmemiz gerekiyor.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari  
Services / Service Manager Kullanımı

**ServiceManager** kullanmak için,  
**Controller** readonly değişikliği  
yaptık.



The screenshot shows the Visual Studio IDE. On the left, the code editor displays the `BooksController.cs` file. The code includes a constructor injection for `IServiceManager`. On the right, the Solution Explorer shows the project structure for a solution named `BookStoreApp`, which contains four projects: Entities, Repositories, Services, and WebAPI. The `WebAPI` project is expanded, showing its controllers and services.

```
//TODO: Yeni eklenen proje referansı
using Entities.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.JsonPatch;
using Microsoft.AspNetCore.Mvc;
using Repositories.Contracts;
using Repositories.EfCore;
using Services.Contracts;

namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class BooksController : ControllerBase
    {
        //ServiceManager's kullanmak için IRepositoryManager kodlarını kapatıyoruz.
        private readonly IServiceManager _manager;

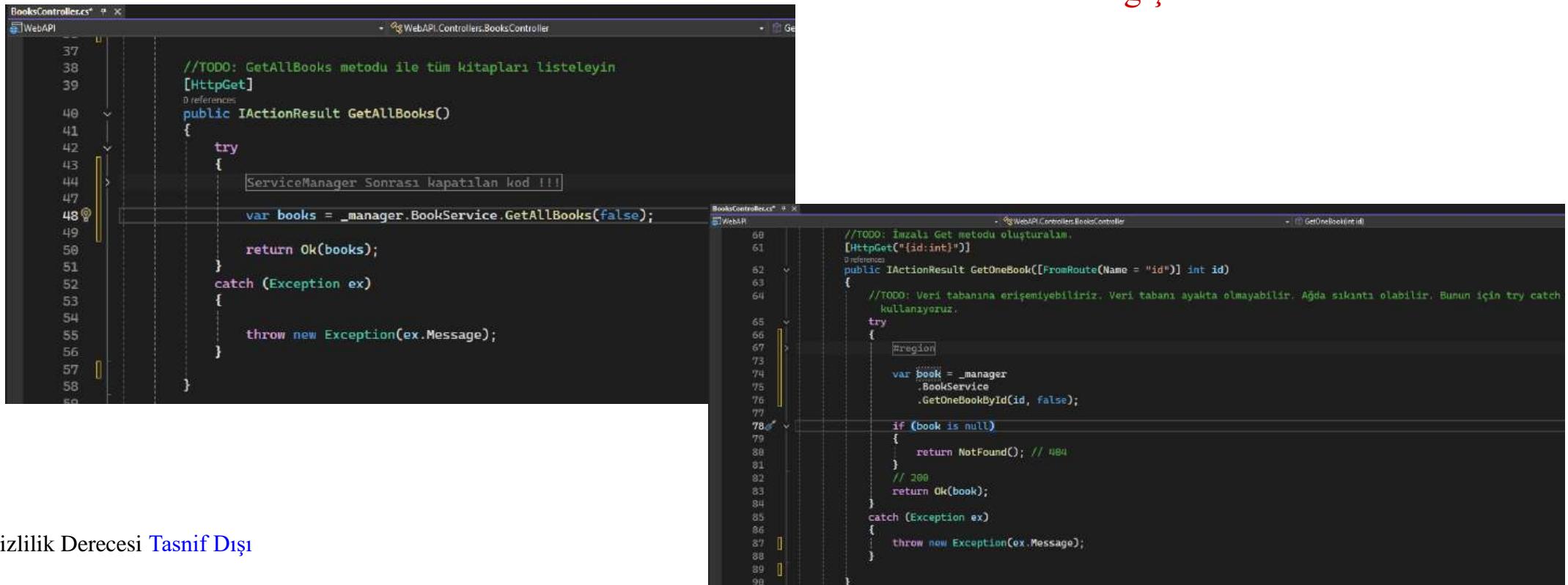
        public BooksController(IServiceManager manager)
        {
            _manager = manager;
        }
    }
}
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari  
Services / Service Manager Kullanımı

**ServiceManager** kullanmak için,  
**Controller** GetAllBooks,  
GetOneBook metodlarında yapılan  
değişiklik.



The image shows two side-by-side code editors in Visual Studio. Both editors are displaying the same file, BooksController.cs, from a Web API project.

**Left Editor (BooksController.cs):**

```
37
38     //TODO: GetAllBooks metodu ile tüm kitapları listeleyin
39     [HttpGet]
40     public IActionResult GetAllBooks()
41     {
42         try
43         {
44             ServiceManager Sonrası kapatılan kod !!!
45
46             var books = _manager.BookService.GetAllBooks(false);
47
48             return Ok(books);
49         }
50         catch (Exception ex)
51         {
52             throw new Exception(ex.Message);
53         }
54     }
55 }
```

**Right Editor (BooksController.cs):**

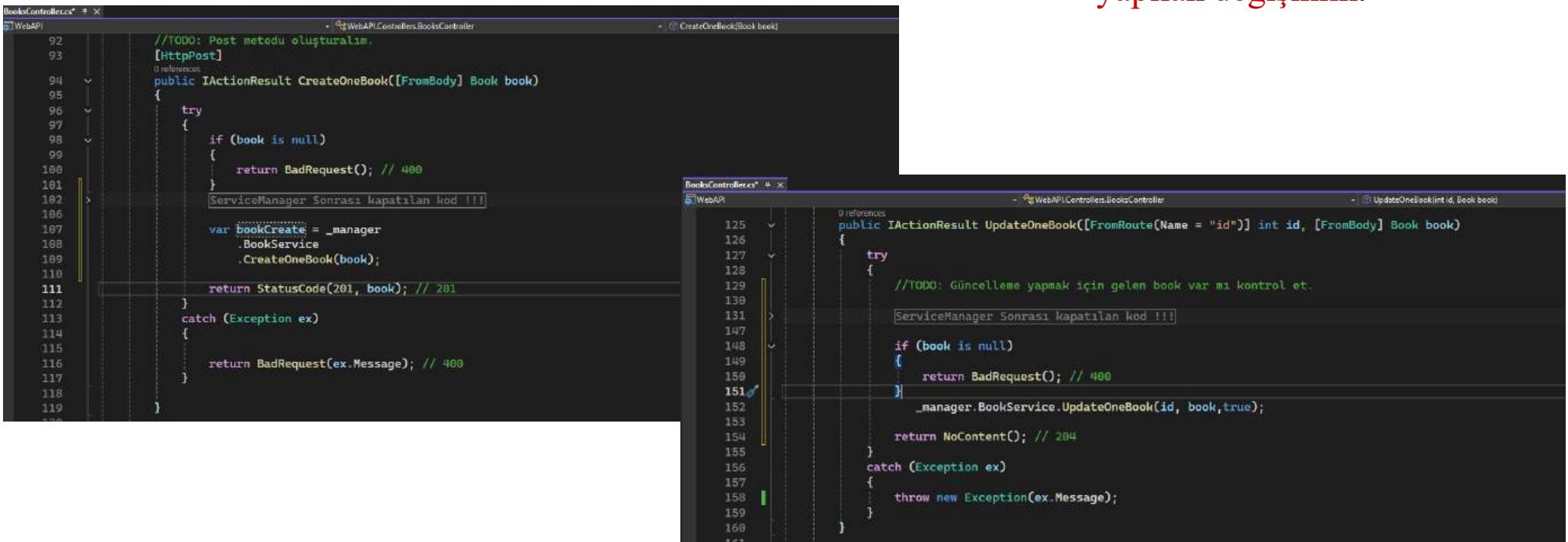
```
60
61
62     //TODO: Imzalı Get metodu oluşturalım.
63     [HttpGet("{id:int}")]
64     public IActionResult GetOneBook([FromRoute(Name = "id")] int id)
65     {
66         //TODO: Veri tabanına erişemeyebiliriz. Veri tabanı ayakta olmayı beklemiyoruz.
67         try
68         {
69             var book = _manager
70                 .BookService
71                 .GetOneBookById(id, false);
72
73             if (book is null)
74             {
75                 return NotFound(); // 404
76             }
77             // 200
78             return Ok(book);
79         }
80         catch (Exception ex)
81         {
82             throw new Exception(ex.Message);
83         }
84     }
85 }
```

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari  
Services / Service Manager Kullanımı

**ServiceManager** kullanmak için,  
**Controller** CreateOneBook ve  
UpdateOneBook metodlarında  
yapılan değişiklik.



```
BooksController.cs # × [WebAPI] - CreateOneBook([FromBody] Book book)
92 //TODO: Post metodu oluşturalım.
93 [HttpPost]
94 public IActionResult CreateOneBook([FromBody] Book book)
95 {
96     try
97     {
98         if (book == null)
99         {
100             return BadRequest(); // 400
101         }
102         ServiceManager Sonrası kapatılan kod !!!
103
104         var bookCreate = _manager
105             .BookService
106             .CreateOneBook(book);
107
108         return StatusCode(201, book); // 201
109     }
110     catch (Exception ex)
111     {
112         return BadRequest(ex.Message); // 400
113     }
114 }
115
116
117
118
119 }
```

```
BooksController.cs # × [WebAPI] - UpdateOneBook(int id, Book book)
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161 }
```

# **Yedinci Bölüm?**

## Katmanlı Mimari

# Proje Oluşturma / Katmanlı Mimari Services / Service Manager Kullanımı

**ServiceManager** kullanmak için,  
**Controller** `CreateOneBook` ve  
`PartiallyUpdateOneBook`  
metotlarında yapılan değişiklik.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Service Manager Kullanımı

ServiceManager kullanmak için,  
Controller dönüşüm bitti.

Çalıştırıralım !!!

Hemen bir hata ile karşılaştık ???

The screenshot shows a Visual Studio code editor with two tabs open: BooksController.cs and Program.cs. The Program.cs tab is active and displays the following C# code:

```
18 //TODO: Program.cs kendi yazdığımız classı ekliyoruz.
19 //TODO: ServicesExtensions db bağlangıtı taşıdığı için,
20 builder.Services.ConfigureDbContext(builder.Configuration);
21
22 //TODO: Parametre kullanmadık. Çünkü ConfigureRepositoryManeger metodu sadece this aldı.
23 builder.Services.ConfigureRepositoryManeger();
24
25
26 //TODO: ServicesExtensions classında tanımladığımız ConfigureServicesManager kullanabilmek için ekliyoruz.
27 builder.Services.ConfigureServiceManager();
28
29
30
31 var app = builder.Build(); ✖
32
33 // Configure the HTTP request
34 if (app.Environment.IsDevelopment)
35 {
36     app.UseSwagger();
37     app.UseSwaggerUI();
38 }
39
40 app.UseHttpsRedirection();
41
42 app.UseAuthorization();
43
44 app.MapControllers();
```

A tooltip window is displayed over the line of code where the error occurred (line 31). The tooltip title is "Exception Unhandled" and the message is "System.ArgumentException: Cannot instantiate implementation type 'Services.Contracts.IServiceManager' for service type 'Services.Contracts.IServiceManager'". Below the message are several options: "Analyze with Copilot", "Show Call Stack", "View Details", "Copy Details", "Start Live Share session", and "Exception Settings".

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari  
Services / Service Manager Kullanımı

Hatanın çözümü !!!

TÜM METOTLARI SWAGGER  
VE POSTMAN ÜZERİNDEN  
TEST EDELİM !!!

The screenshot shows the Visual Studio IDE interface. On the left, the Solution Explorer displays a solution named 'BookStoreApp' containing four projects: Entities, Repositories, Services, and WebAPI. The Services project is expanded, showing its subfolders (Dependencies, Contracts, Services) and files (BookService.cs, ServiceManager.cs). On the right, the main code editor window shows a C# file named 'Program.cs'. The code is as follows:

```
18     //TODO: IOC ye DBContext tanımını yapmış oluyoruz. (Bir DBContexte ihtiyacımız olduğunda bunun somut haline ulaşmanız için)
19
20     //TODO: Program.cs dosyasında ctrl +x ile aldık.
21
22     services.AddDbContext<RepositoryContext>(options =>
23         options.UseSqlServer(configuration.GetConnectionString("sqlConnection")));
24
25
26     //TODO: Hata alındı onun için yeni servis ekledik.
27     public static void ConfigureRepositoryManeger(this IServiceCollection services) => services.AddScoped<IRepositoryManger,
28             RepositoryManager>();
29
30     //TODO: Servis Managerı kullanabilmeniz için ekliyoruz.
31     //TODO: IServiceManager karşılık IServiceManager olmuş.
32     public static void ConfigureServicesManager(this IServiceCollection services) => services.AddScoped<IServiceManager,
33             ServiceManager>();
34 }
```

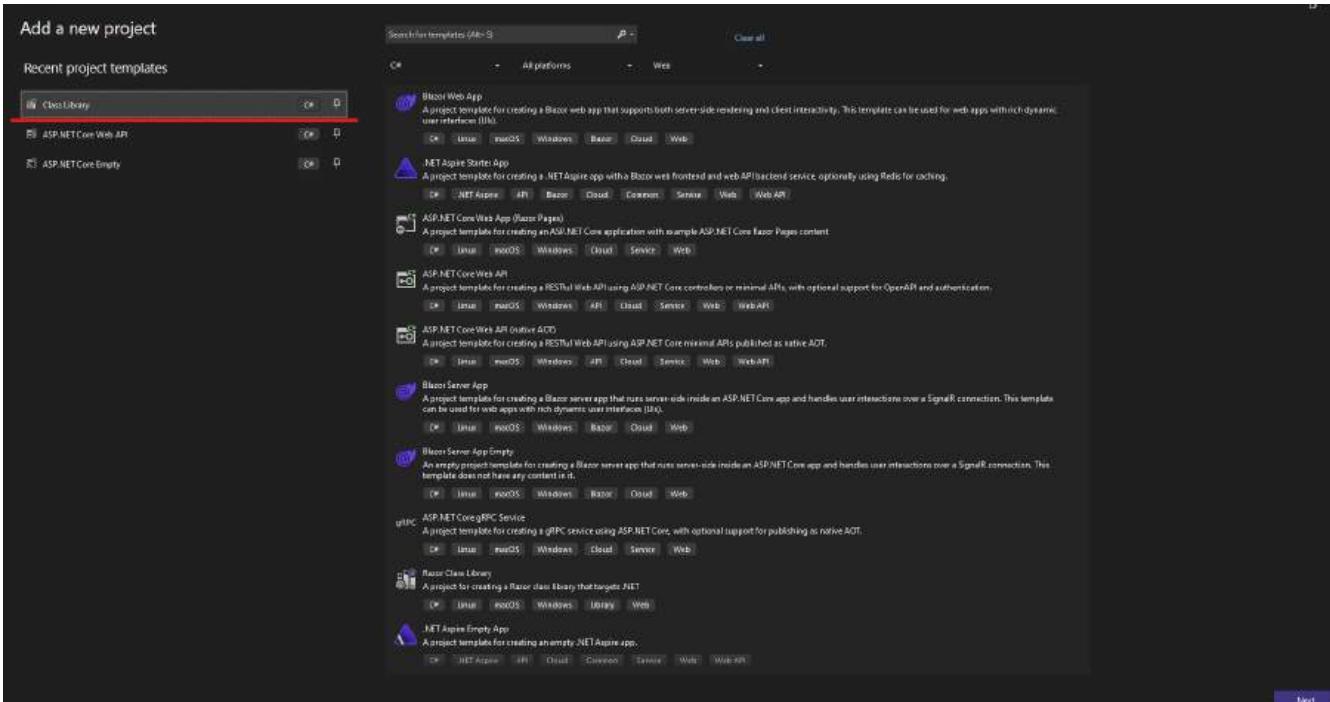
Projemizi servis katmanına taşmış olduk.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Sunum Katmanı



Gizlilik Derecesi [Tasnif Dışı](#)

Projemizde yeni bir **Class Library** ekliyoruz.

## İsmi Presentation

Slayt Numarası .../...

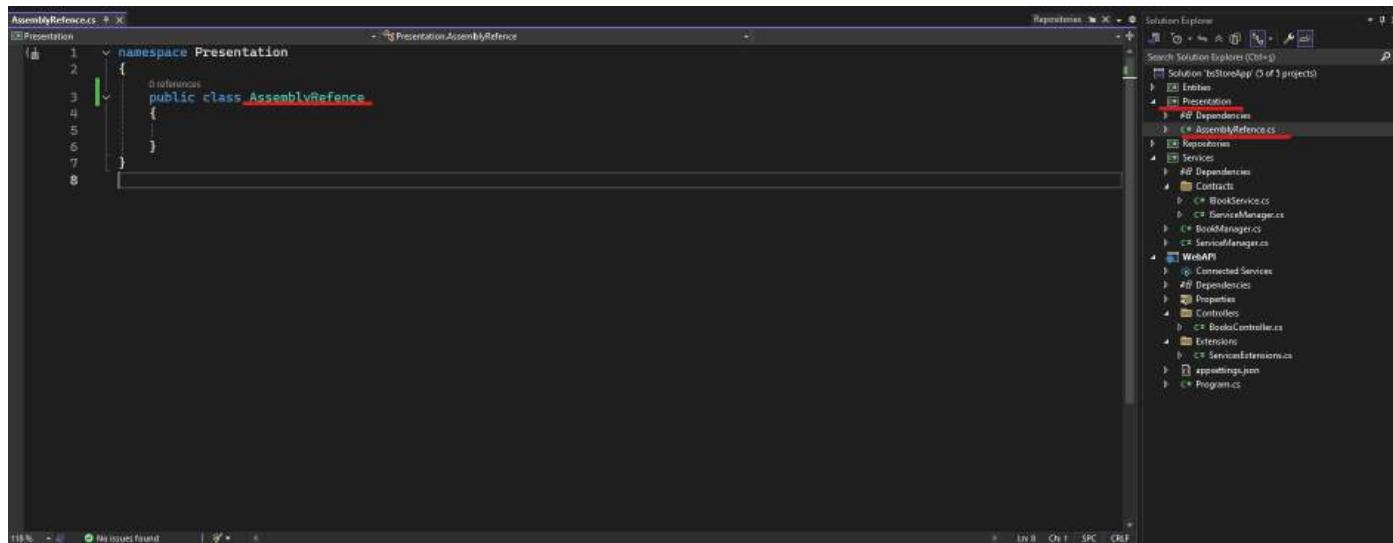
# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Sunum Katmanı

Presentation **class1**'i  
**AssemblyReference**  
dönüştürüyoruz.



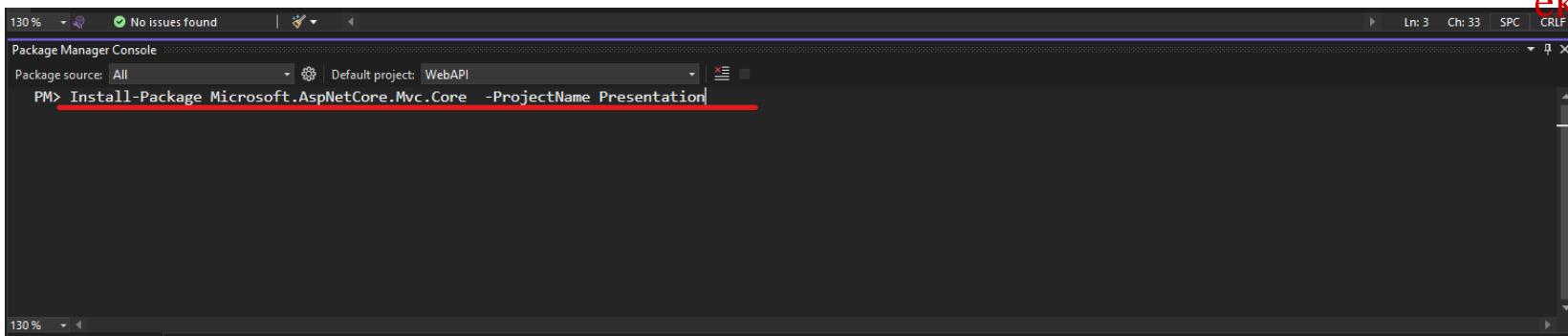
# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Sunum Katmanı

Presentation projesine  
Install-Package  
Microsoft.AspNetCore.Mvc.Core  
-ProjectName Presentation  
ekliyoruz.



The screenshot shows the Package Manager Console window in Visual Studio. The console window has a dark background and white text. At the top, it says "130 %", "No issues found", and "Package source: All". Below that, it says "Default project: WebAPI". The main area of the window contains the command "PM> Install-Package Microsoft.AspNetCore.Mvc.Core -ProjectName Presentation". The entire command line is highlighted with a red underline. The rest of the console window is empty and dark.

Presentation projesine  
Yukarıdaki kitaplığı ekleyerek class'a controller  
olma özelliği kazandırabiliriz.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Sunum Katmanı

Presentation projesinde **Controller** klasörü ekle içine **BooksController** classını ekle

The screenshot shows the Visual Studio IDE interface. On the left, the Solution Explorer window displays a solution named 'BookStoreApp' containing five projects: Entities, Dependencies, Models, Presentation, and WebAPI. The 'Presentation' project is expanded, showing its Controllers folder which contains a file named 'BooksController.cs'. On the right, the main code editor window shows the content of 'BooksController.cs'. The code is as follows:

```
BooksController.cs
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Presentation.Controllers
8  {
9      //TODO: internal public yap.
10     [ApiController]
11     public class BooksController
12     {
13     }
14 }
```

# Yedinci Bölüm?

Katmanlı Mimari

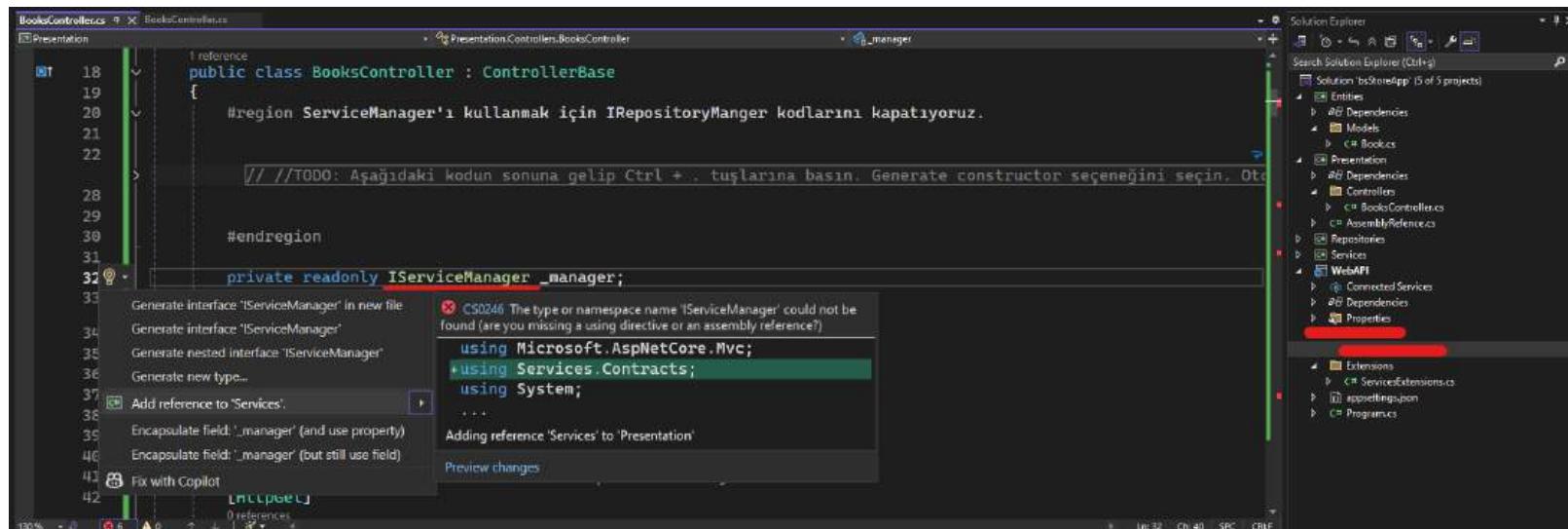
Proje Oluşturma / Katmanlı Mimari

Services / Sunum Katmanı

WebAPI projesindeki  
**BooksController’ı bunun içine**  
taşı.

WebAPI Projesindeki  
**Controller klasörünü sil.**

Daha sonra hatalara gidermeye  
başla



The screenshot shows the Visual Studio IDE interface. On the left is the code editor with the file `BooksController.cs` open. The code defines a `BooksController` class that inherits from `ControllerBase`. It contains a private field `_manager` of type `IServiceManager`. A tooltip for this field indicates a CS0246 error: "The type or namespace name 'IServiceManager' could not be found (are you missing a using directive or an assembly reference?)". Below the code editor, a context menu is open over the `_manager` field, with the option "Add reference to 'Services'" highlighted. On the right is the Solution Explorer window, which displays the project structure for a solution named `bsShopApp`. The `WebAPI` project is selected, showing its contents including `BooksController.cs`, `AssemblyInfo.cs`, `Properties`, and `Program.cs`.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Sunum Katmanı

Hataları gidermeye devam et !!!

The image shows two side-by-side code editors in Microsoft Visual Studio. Both editors are displaying the same file, `BooksController.cs`, which is part of the `Presentation` project.

**Left Editor:** The current tab is `BooksController.cs`. The code is as follows:

```
95 //TODO: Post metodu oluşturalım.  
96 [HttpPost]  
97  
98 public IActionResult CreateOneBook([FromBody] Book book)  
99 {  
100     using Entities.Models;  
101     Entities.Models.Book  
102     Generate type 'Book' > +using Entities.Models;  
103     using Microsoft.AspNetCore.Mvc;  
104     ...  
105     Fix type 'Book'  
106     Change signature...  
107     Fix with Copilot  
108     #endregion  
109  
110     var bookCreate = _manager  
111         .BookService  
112         .CreateOneBook(book);  
113  
114     return StatusCode(201, book); // 201  
115 }  
116 }
```

An error message is displayed in the status bar: "CS0246 The type or namespace name 'Book' could not be found (are you missing a using directive or an assembly reference?)". A tooltip provides the same information. A context menu is open at line 98, with the "using Entities.Models;" option highlighted.

**Right Editor:** The previous tab is `BooksController.cs`, and the current tab is `PatchController.cs`. The code is as follows:

```
167 [HttpDelete("{id:int}")]  
168  
169 public IActionResult DeleteOneBook([FromRoute(Name = "id")] int id)...  
170  
171 //TODO: Put metodu oluşturalım.  
172 [HttpPatch("{id:int}")]  
173  
174 public IActionResult PartiallyUpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] JsonPatchDocument<Book> bookPatch)  
175 {  
176     Generate type 'JsonPatchDocument'  
177     Add null check  
178     Install package 'Microsoft.AspNetCore.JsonPatch' > +using Entities.Models;  
179     Install package 'Azure.Core' > +using Microsoft.AspNetCore.JsonPatch;  
180     Change signature...  
181     Wrap every parameter  
182     Unwrap and indent all parameters  
183     ...  
184     Fix with Copilot  
185     #endregion  
186     return NotFound(); // 404  
187  
188     bookPatch.ApplyTo(entity);  
189     _manager.BookService.UpdateOneBook(id, entity, true);  
190 }
```

An error message is displayed in the status bar: "CS0246 The type or namespace name 'JsonPatchDocument<>' could not be found (are you missing a using directive or an assembly reference?)". A tooltip provides the same information. A context menu is open at line 178, with the "using Microsoft.AspNetCore.JsonPatch;" option highlighted.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Sunum Katmanı

Arkadaşlar Presentation katmanını oluşturduk.

Peki WebAPI projesinde Program.cs ne yapılmalıdır ki Bu katmadaki Controller'ı kullanınsın !!!

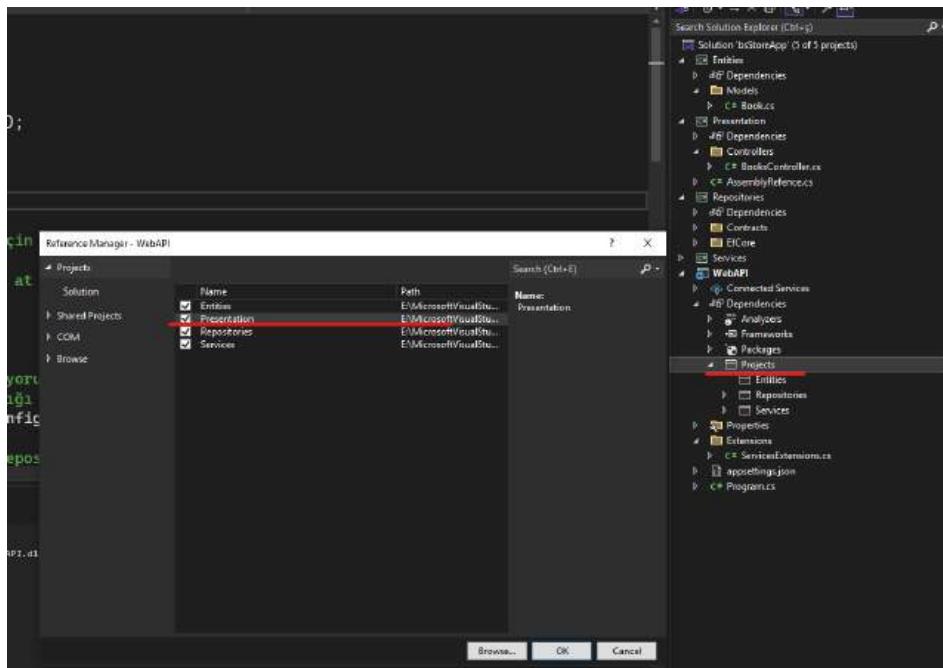
# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Sunum Katmanı

Program.cs de güncelleme!!!



```
1  //using Microsoft.EntityFrameworkCore;
2  //using Repositories.EfCore;
3  //using WebAPI.Extensions;
4
5
6  var builder = WebApplication.CreateBuilder(args);
7
8  // Add services to the container.
9
10 builder.Services.AddControllers()
11     //TODO: Presentation Katmanı için gerekli olan ayarları yapıyoruz.
12     .AddApplicationPart(typeof(Presentation.AssemblyReference).Assembly)
13     .AddNewtonsoftJson();
14
15 //TODO: Newtonsoft.Json paketini kullanabilmek için gerekli ayarları yapalım.
16
17 // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
18 builder.Services.AddEndpointsApiExplorer();
19 builder.Services.AddSwaggerGen();
20
21 //TODO: Program.cs Wendi yazdığımız classı ekliyoruz.
22 //TODO: ServicesExtensions db bağlantısı taşıdığı için,
23 builder.Services.ConfigureSqlContext(builder.Configuration);
```

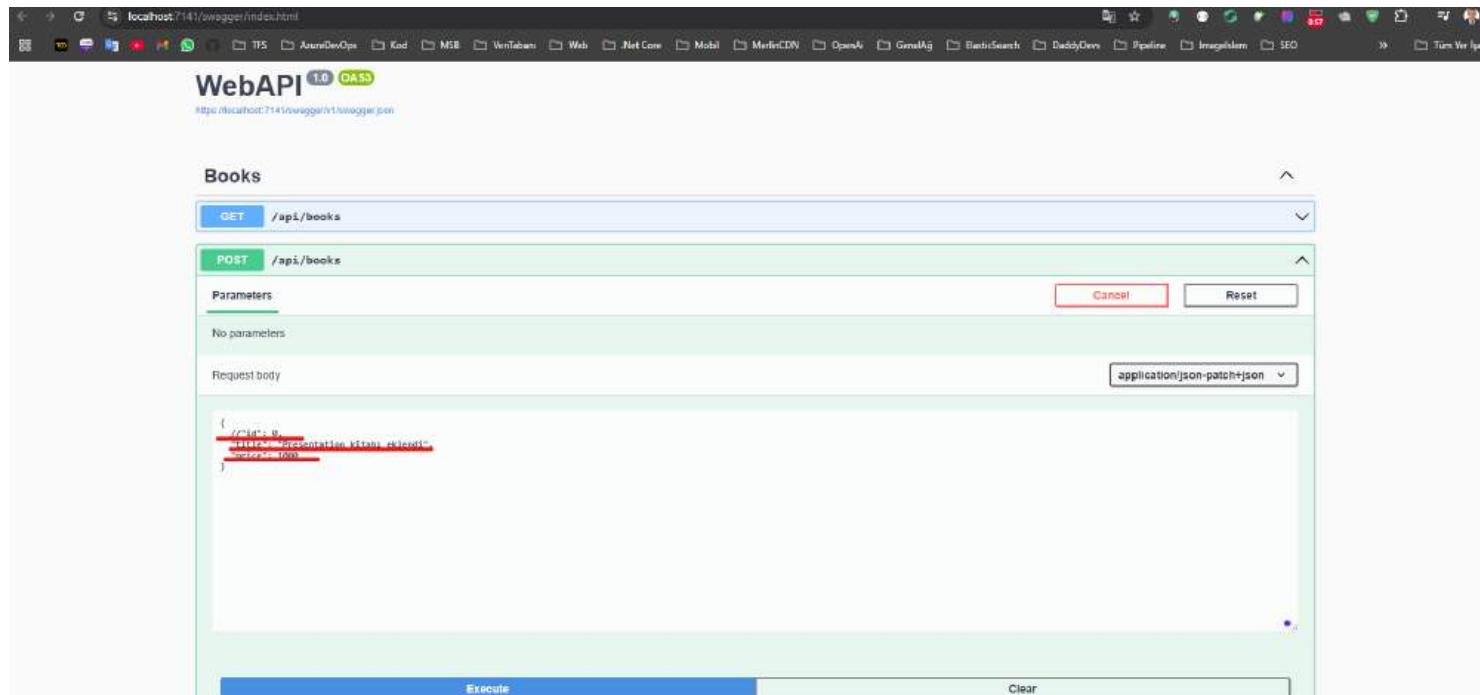
The screenshot shows the 'Program.cs' file in the 'WebAPI' project of the solution. The code is partially commented out with 'TODO' notes. The 'Presentation' project is referenced in the code via 'Presentation.AssemblyReference'. The 'Solution Explorer' on the right shows the overall project structure including 'Entities', 'Presentation', 'Repositories', and 'Services' projects.

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari  
Services / Sunum Katmanı

Uygulamayı çalıştır. Tüm servisleri kontrol et !!!!



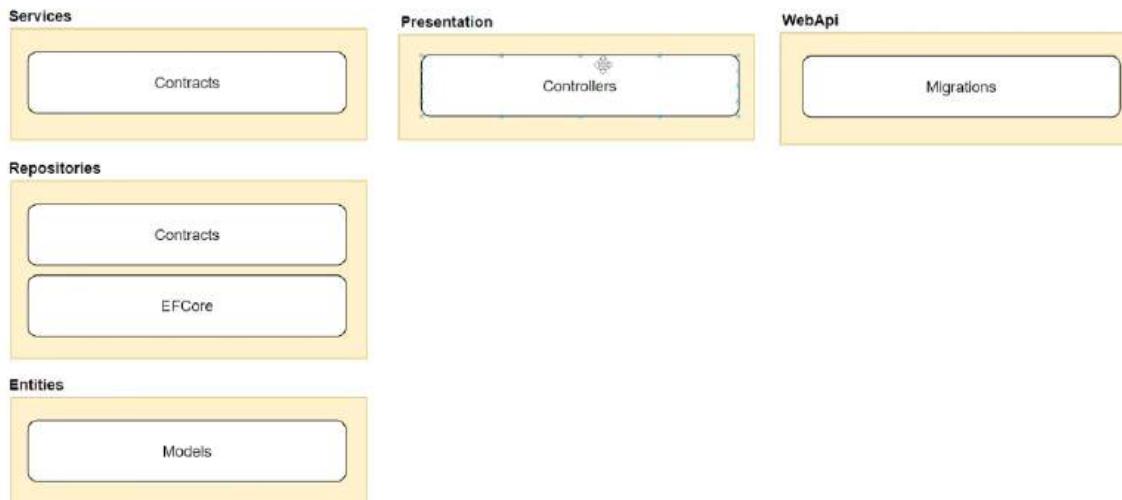
# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Sunum Katmanı

**WebAPI projemizi ne tür bir katmanlı mimariye getirdik. !!!!**



# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Repository ContextFactory

**Bu bölümde;**

- Veri tabanını nasıl sıfırladığımızı göreceğiz.
- Dizayn aşamasında iken veri tabanı scriptleri ve migration için konfigürasyon ifadesi yazıyor olacağız.

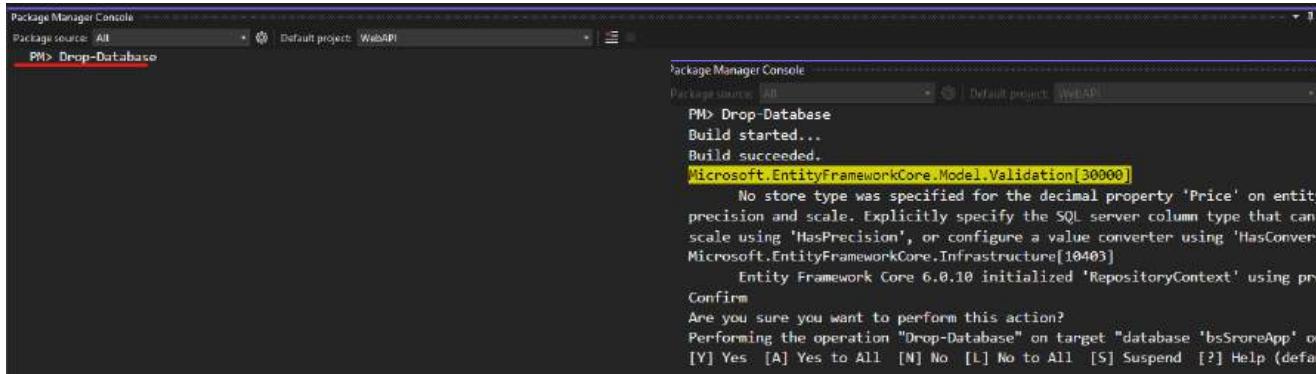
# Yedinci Bölüm?

Katmanlı Mimari

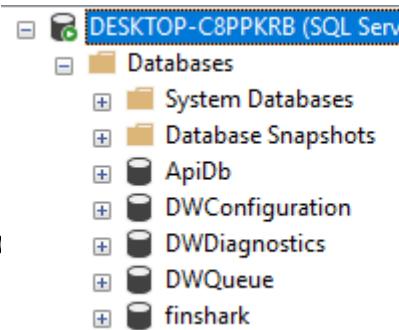
Proje Oluşturma / Katmanlı Mimari

Services / Repository ContextFactory

İlgili veri tabanını sil. !!!



```
PM> Drop-Database
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
    No store type was specified for the decimal property 'Price' on entity type 'Book'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value converter using 'HasConversion'.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
    Entity Framework Core 6.0.10 initialized 'RepositoryContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer:6.0.10' with options: None
Confirm
Are you sure you want to perform this action?
Performing the operation "Drop-Database" on target "database 'bsScoreApp' on server 'DESKTOP-C8PPKRB'".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"):
```



# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari  
Services / Repository ContextFactory

Hadi Yeni katmanlı mimaride migration  
alalım !!!!

Hata aldık, Senin hedef projen WebAPı  
ancak senin katmanın presentation

```
Package Manager Console
Package source: All | Default project: WebAPI
PM> Add-Migration
cmdlet Add-Migration at command pipeline position 1
Supply values for the following parameters:
Name: Presentation
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
No store type was specified for the decimal property 'Price' on entity type 'Book'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value converter using 'HasConversion'.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
Entity Framework Core 6.0.10 initialized 'RepositoryContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer:6.0.10' with options: None
Your target project 'WebAPI' doesn't match your migrations assembly 'Repositories'. Either change your target project or change your migrations assembly.
Change your migrations assembly by using DbContextOptionsBuilder. E.g. options.UseSqlServer(connection, b => b.MigrationsAssembly("WebAPI")). By default, the migrations assembly is the assembly containing the DbContext.
Change your target project to the migrations project by using the Package Manager Console's Default project drop-down list, or by executing "dotnet ef" from the directory containing the migrations project.
PM>
```

# Yedinci Bölüm?

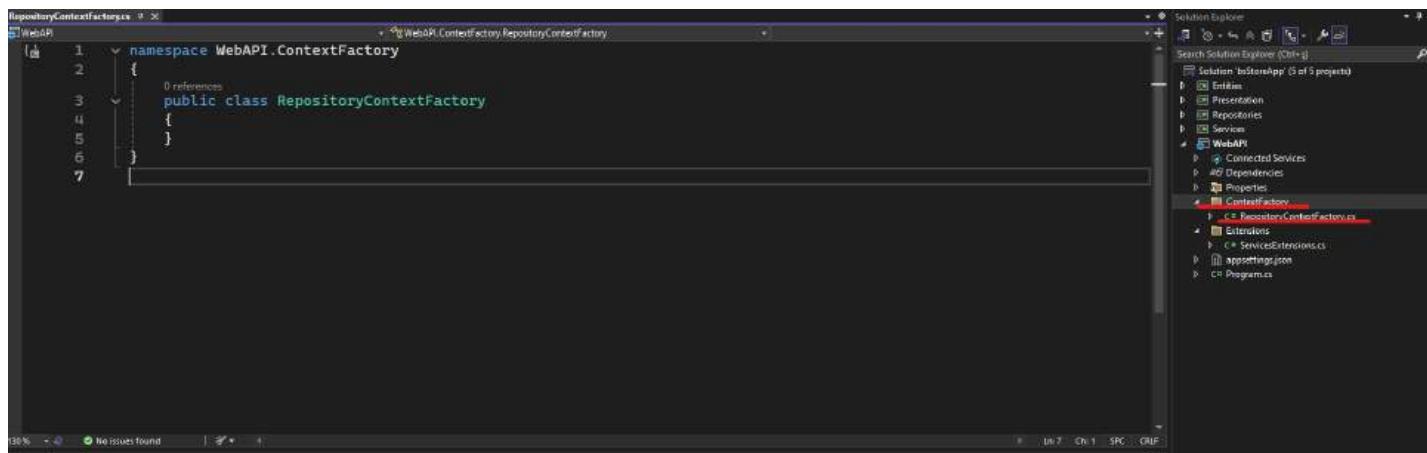
Hadi Hatayı gider !!!

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari  
Services / Repository ContextFactory

WebAPI projesine ContextFactory klasörünü ekle.

İçine RepositoryContextFactory sınıfını ekle



# Yedinci Bölüm?

RepositoryContextFactory sınıfını ekle

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Repository ContextFactory

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays the `RepositoryContextFactory.cs` file with the following content:

```
1  namespace WebAPI.ContextFactory
2  {
3      public class RepositoryContextFactory
4      {
5      }
6  }
```

On the right, the Solution Explorer window shows the project structure for `b1storeApp`, which contains five projects: Entities, Presentation, Repositories, Services, and WebAPI. The `WebAPI` project is expanded, showing its subfolders: Connected Services, Dependencies, Properties, and ContextFactory. Inside the `ContextFactory` folder, the `RepositoryContextFactory.cs` file is listed along with other files like `ServiceExtensions.cs` and `appsettings.json`.

# Yedinci Bölüm?

RepositoryContextFactory sınıfını kodla  
!!!

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Repository ContextFactory

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays the `RepositoryContextFactory.cs` file with C# code. The code implements the `IDesignTimeDbContextFactory<RepositoryContext>` interface, which requires overriding the `CreateDbContext(string[] args)` method. The implementation uses `ConfigurationBuilder` and `DbContextOptionsBuilder<RepositoryContext>` to set up the database context. The Solution Explorer on the right shows a solution named 'IoStoreApp' containing five projects: Entities, Presentation, Repositories, Services, and WebAPI. The WebAPI project is selected, showing its files: `Connected Services`, `Dependencies`, `Properties`, `ContextFactory` (containing `RepositoryContextFactory.cs`), `Extensions` (containing `ServicesExtensions.cs`), `appsettings.json`, and `Program.cs`. The status bar at the bottom indicates slide number 188.

```
1  using Microsoft.EntityFrameworkCore;
2  using Microsoft.EntityFrameworkCore.Design;
3  using Repositories.EfCore;
4
5  namespace WebAPI.ContextFactory
6  {
7      //TODO: RepositoryContextFactory classına IDesignTimeDbContextFactory<RepositoryContext> interface'ini implement et
8      //TODO: IDesignTimeDbContextFactory istediği için CreateDbContext metodunu override et
9      public class RepositoryContextFactory : IDesignTimeDbContextFactory<RepositoryContext>
10     {
11         //TODO: CreateDbContext oluşturuyoruz.
12         //TODO: Neden ???
13         //TODO: Biz DbContext istiyor ise k. 2 şeye ihtiyacımız var 1- Configuration 2- DbContextOptions
14         public RepositoryContext CreateDbContext(string[] args)
15         {
16             //TODO: ConfigurationBuilder nesnesini alıyoruz.
17             var configuration = new ConfigurationBuilder()
18                 .SetBasePath(ApplicationContext.BaseDirectory)
19                 .AddJsonFile("appsettings.json")
20                 .Build();
21
22             //TODO: DbContextOptionsBuilder nesnesini alıyoruz.
23             var builder = new DbContextOptionsBuilder<RepositoryContext>()
24                 .UseSqlServer(configuration.GetConnectionString("sqlConnection"),
25                               prj => prj.MigrationsAssembly("WebApi"));
26             //TODO: sqlConnection ifadesi appsettings.json dosyasındaki connection string'i alıyoruz.
27             //TODO: prj => prj.MigrationsAssembly("WebApi") ifadesi ile migration'ları WebApi projesine yönlendiriyoruz.
28
29             return new RepositoryContext(builder.Options);
30         }
31     }
32 }
```

# Yedinci Bölüm?

Migration yap!!!

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Repository ContextFactory

```
PM> Add-Migration startPoint
Build started...
Build succeeded.
No store type was specified for the decimal property 'Price' on entity type 'Book'. This will cause values to be silently truncated if they do not fit in the default precision and scale.
Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure
a value converter using 'HasConversion'.
To undo this action, use Remove-Migration.
PM>
```

```
2022/05/18 01: StartPoint.cs
using Microsoft.EntityFrameworkCore.Migrations;
#nullable disable
namespace WebAPI.Migrations
{
    public partial class startPoint : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Books",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Title = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Price = table.Column<decimal>(type: "decimal(18,2)", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Books", x => x.Id);
                });
        }
    }
}
```

Solution Explorer (C:\Users\...)

- Solution '02WebAPI' (1 of 1 projects)
  - Entities
  - Presentation
  - Repository
  - Services
  - WebAPI
    - Connected Services
    - Properties
    - Migrations
      - 2022/05/18 01: startPoint.cs
      - 2022/05/18 01: MigrationBuilder.migrationBuilder.cs
    - Extensions
      - ServicesExtensions.cs
    - Program.cs

# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Repository ContextFactory

Database Update yap!!!

The screenshot shows a Windows application window with several panes:

- Package Manager Console:** Shows the command "PM> Update-Database" being run, followed by build logs: "Build started...", "Build succeeded.", and migration details: "No store type was specified for the decimal property 'Price' on entity type 'Book'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value converter using 'HasConversion'. Applying migration '20250505182041\_startPoint'." The command "Done." is shown at the bottom.
- Object Explorer:** Shows the database structure for "DESKTOP-C8PPKRB (SQL Server 15.0.2000.5 - sa)". It includes nodes for Databases, System Databases, Database Snapshots, ApiDb, bsScoreApp (selected), DWConfiguration, DW.Diagnostics, DWQueue, finshark, Security, Server Objects, Replication, PolyBase, Always On High Availability, Management, Integration Services Catalogs, SQL Server Agent (Agent XPs disabled), and XEvent Profiler.
- SQL Query Results:** A query "SELECT \* FROM Books" is run against the "bsScoreApp" database. The results show three rows of book data:

ID	Title	Price
1	Karaköz ve Hırvat	75.00
2	Mesnevi	175.00
3	Davalı	375.00

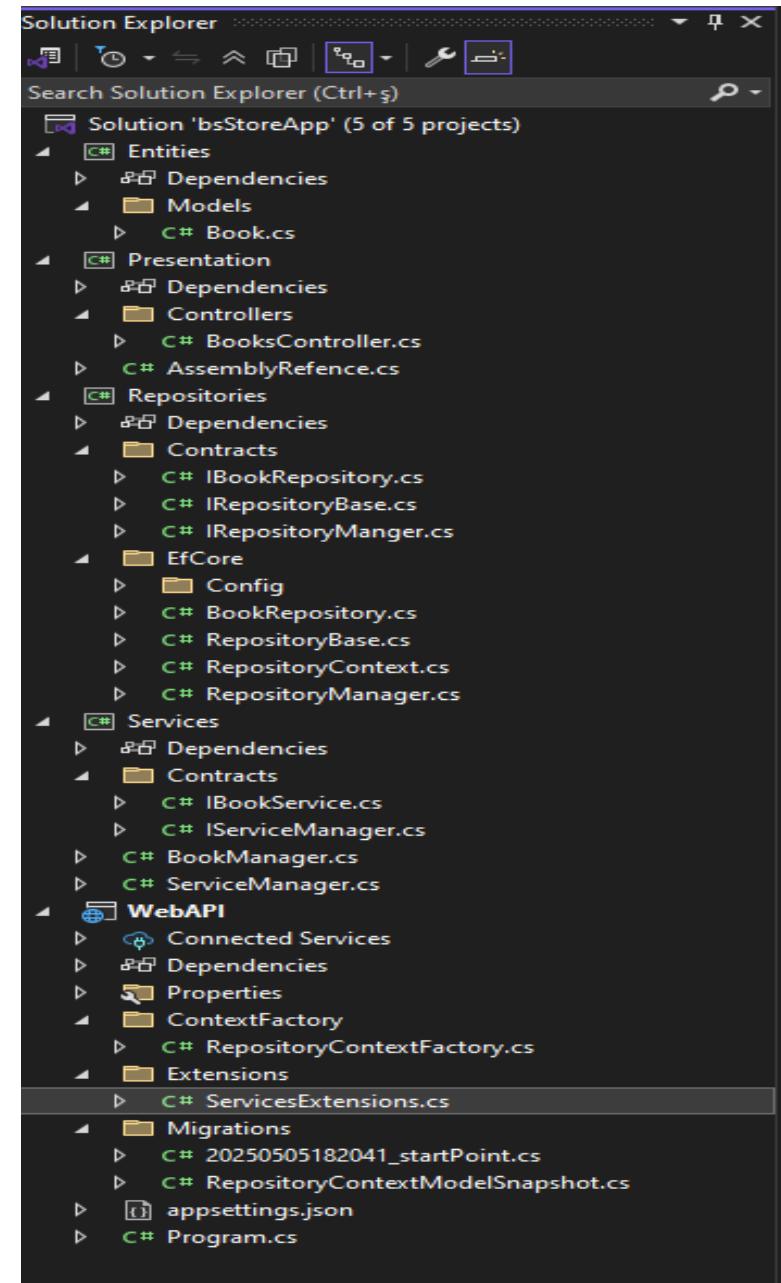
# Yedinci Bölüm?

Katmanlı Mimari

Proje Oluşturma / Katmanlı Mimari

Services / Repository ContextFactory

## Gün sonunda ne oldu !!!



# Yedinci Bölüm Bitti

## Hadi Özetleyelim?

### İçerik

- Entities Layer
- Repositories Layer
- IRepositoryBase
- RepositoryBase
- BookRepository
- RepositoryManager
- Lazy Loading
- Service Extensions
- Services Layer
- BookService
- ServiceManager
- Presentation Layer
- Repository Context Factory

# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane

NLog

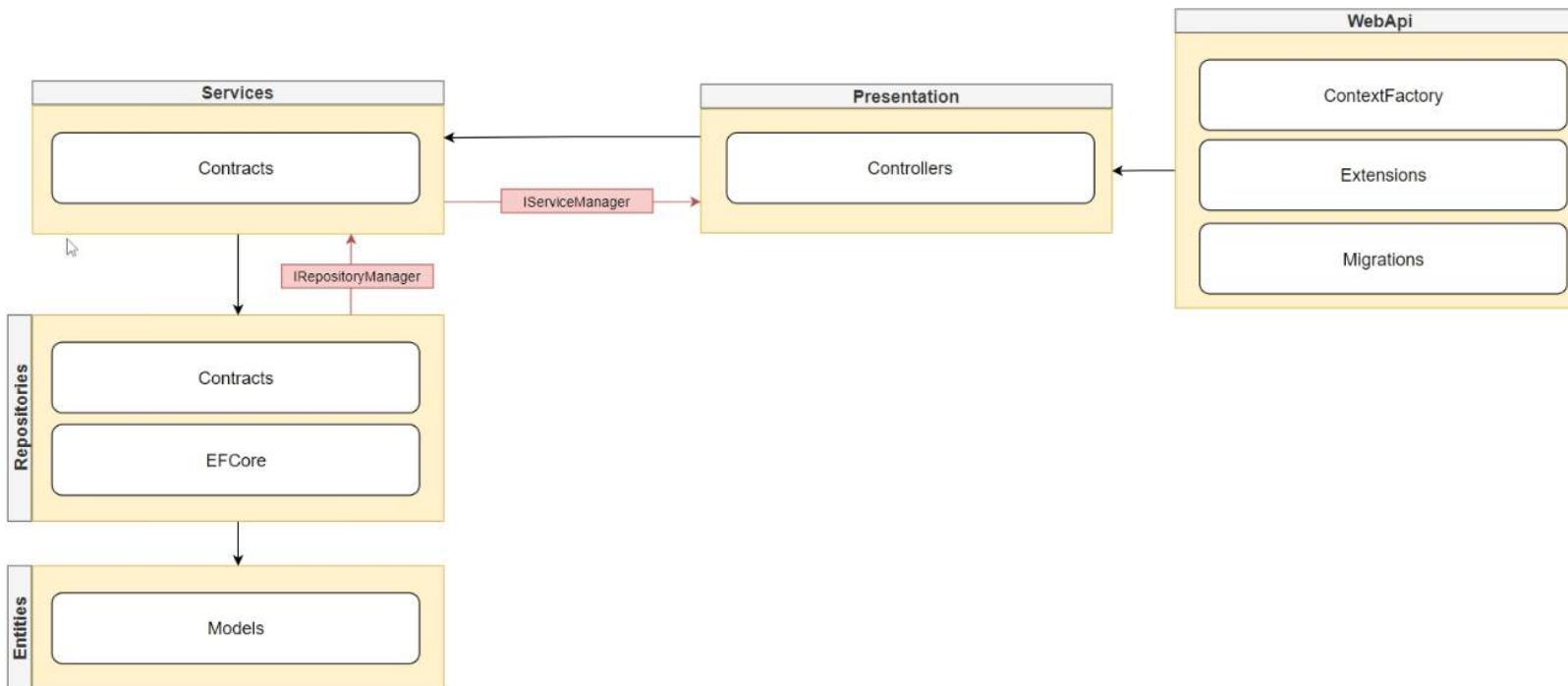
- NLog.Extensions.Logging
- LoggerService
- LoggerManager
- nlog.config
- Servis Kaydı
- BookService tasarımının değiştirilmesi
- ServiceManager tasarıımı değiştirilmesi

# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog)

Uygulamamızın son hali !!!



# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / ILoggerService

Services projesine ILoggerService eklendi !!!

```
IllogService.cs  X
Services
Services.Contracts.ILoggerService
Login(string message)
Solution Explorer
Search Solution Explorer (Ctrl+S)
Solution 'bsStoreApp' (5 of 5 projects)
Entities
Presentation
Repositories
Services
Dependencies
Contracts
C# ILoggerService.cs
C# IServiceManager.cs
BookManager.cs
ServiceManager.cs
WebAPI

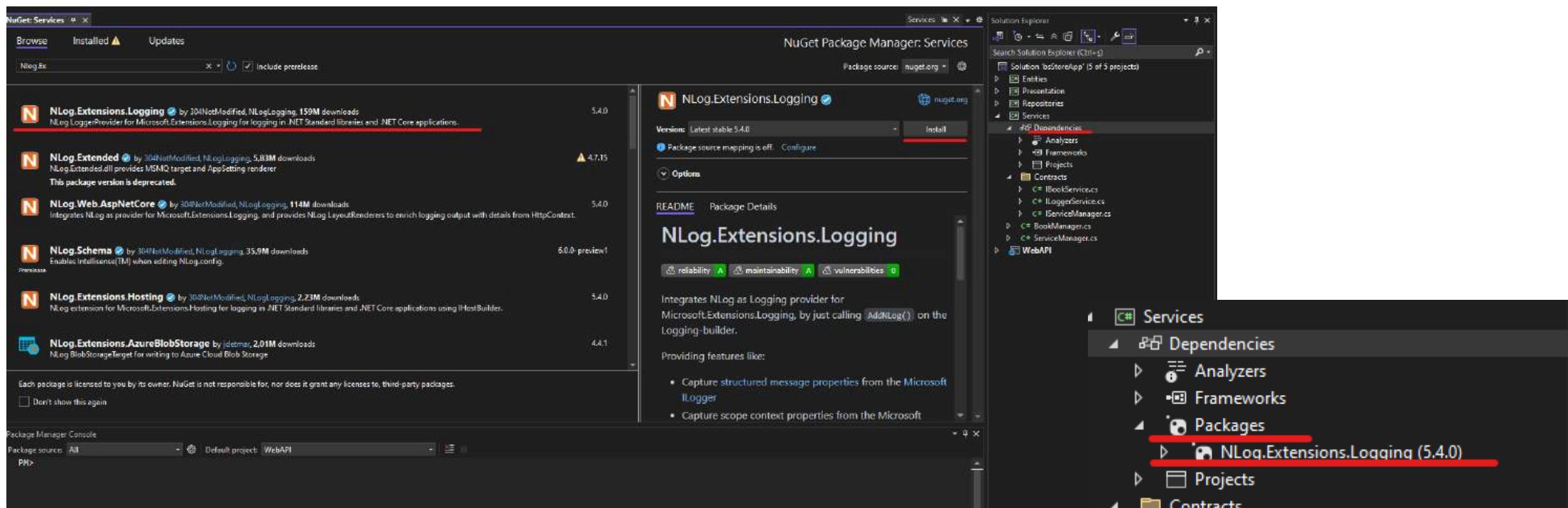
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Services.Contracts
8  {
9      //TODO:internal erişim belirleyicini public olarak değiştirin.
10     public interface ILoggerService
11     {
12         //TODO: Information, Warning, Error ve Debug logları için ayrı ayrı metodlar oluşturduk.
13         void LogInfo(string message);
14         void LogWarning(string message);
15         void LogError(string message);
16         void LogDebug(string message);
17     }
18 }
19
```

# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / NLog.Extensions.Logging

NLog.Extensions.Logging eklendi !!!  
Son versiyon olabilir.



# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / Logger Manager

**LoggerManager** classı eklendi ve  
ILoggerService den implmente edildi..

The screenshot shows the Visual Studio IDE interface. On the left is the code editor window titled "LoggerManager.cs" with the following C# code:

```
using NLog;
using Services.Contracts;

namespace Services
{
    //TODO: internal erişim belirleyicini public olarak değiştirin.
    public class LoggerManager : ILoggerService
    {
        //TODO: ILogger using NLog; üzerinden beslendiğine dikkat edin.
        //TODO: Loger kullanmak için ekle.
        private static ILogger logger = LogManager.GetCurrentClassLogger();

        public void LogDebug(string message) => logger.Debug(message);

        public void LogError(string message) => logger.Error(message);

        public void LogInfo(string message) => logger.Info(message);

        public void LogWarning(string message) => logger.Warn(message);
    }
}
```

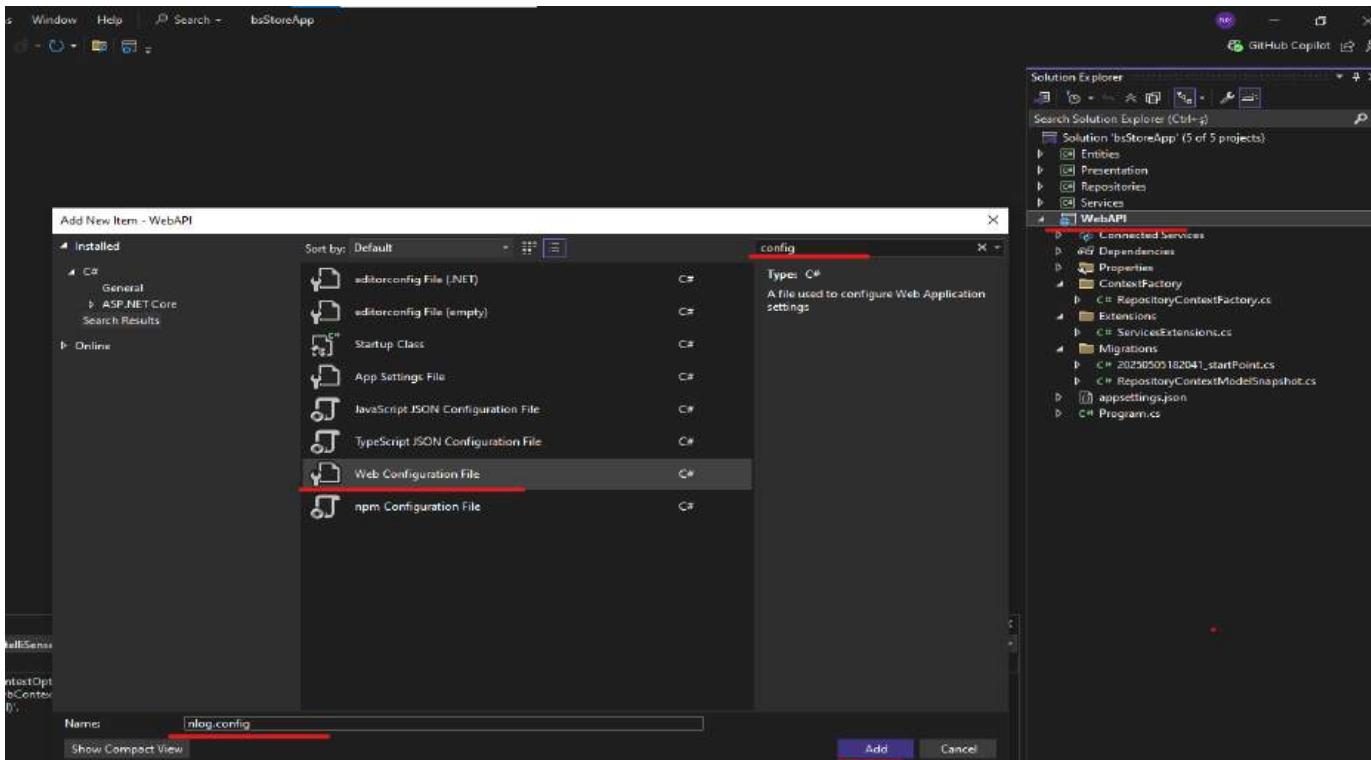
On the right is the "Solution Explorer" window, which displays the project structure for "bsStoreApp" (3 of 5 projects). The "Services" project contains the "LoggerManager.cs" file, which is highlighted.

# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / Yapılandırma

<https://nlog-project.org/config/> sitesi  
Nlog yardımcı dokumani dr.



# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / Yapılandırma

nlog.config dosyası.  
Benden isteyin ...

The screenshot shows the Visual Studio IDE interface. On the left, the Solution Explorer displays a solution named 'bsStoreApp' containing five projects: Entities, Presentation, Repositories, Services, and WebAPI. The WebAPI project is currently selected. On the right, the main editor window shows the XML configuration file 'nlog.config'. The code defines an NLog configuration with an internal log target to a file named 'internal\_log.txt' in the 'internal\_logs' directory. It also defines a target named 'logfile' to a file named 'logfile.txt' in the 'logs' directory, with a layout that includes the date, log level (uppercase), and message. A note in the code indicates that this target can be expanded to include application logs. Below this, there are sections for rules and a logger. The 'nlog.config' file is highlighted in the Solution Explorer.

```
<?xml version="1.0" encoding="utf-8"?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      autoReload="true"
      internalLogLevel="Info"
      internalLogFile=".\\internal_logs\\internallog.txt">
    <!--internal log alır.-->

    <targets>
        <target name="logfile" xsi:type="File"
            fileName=".\\logs\\${shortdate}_logfile.txt"
            layout="${longdate} ${level:uppercase=true} ${message}" />
        <!--Uygulama Logları için-->
        <!--Buraya artıralabiliriz.
        1-Mail'e gönder
        2-Database'e gönder
        3-Logstash'a gönder
        4-ElasticSearch'e gönder
        5-File'a gönder-->
    </targets>

    <rules>
        <logger name="*" minlevel="Debug" writeTo="logfile" />
    </rules>

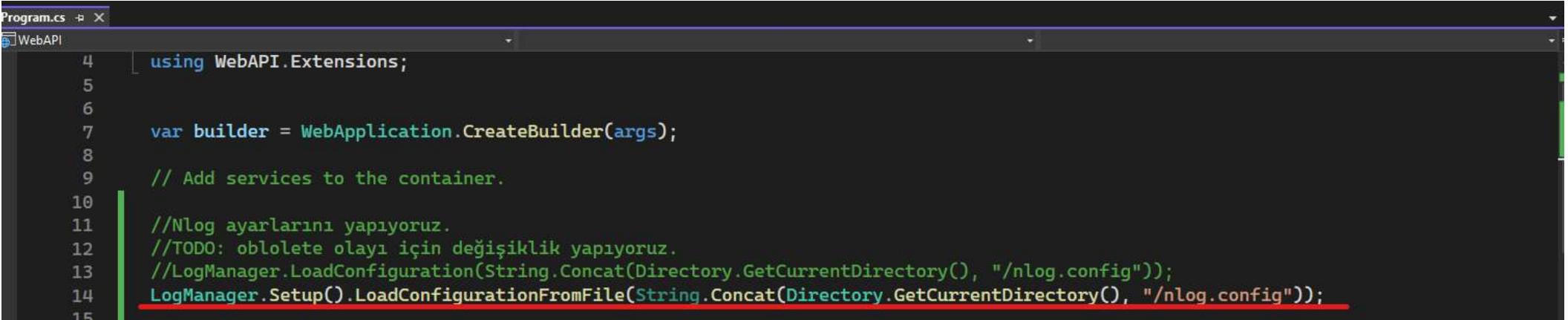
```

# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / Yapılandırma

NLog Program.cs konfigürasyonu



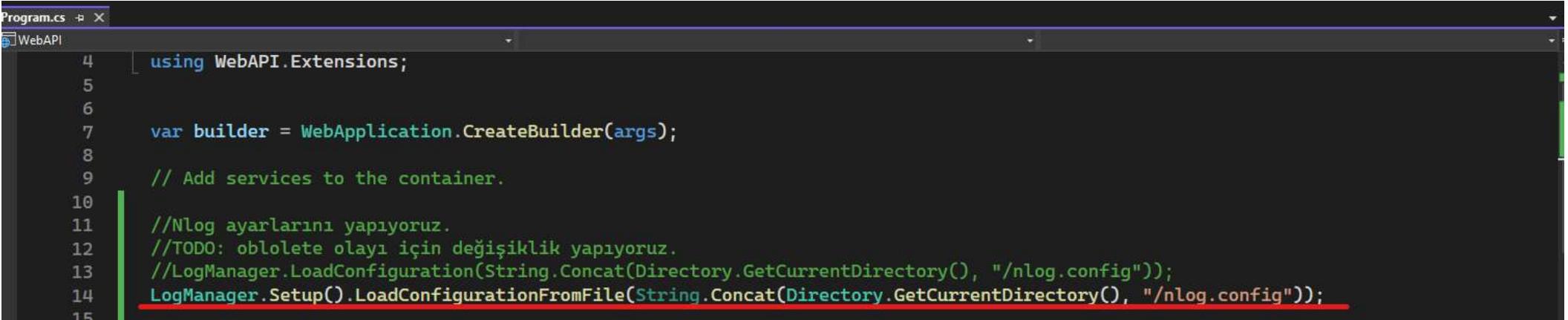
```
Program.cs  ✘ X
WebAPI
4  using WebAPI.Extensions;
5
6
7  var builder = WebApplication.CreateBuilder(args);
8
9  // Add services to the container.
10
11 //Nlog ayarlarını yapıyoruz.
12 //TODO: obsolete olayı için değişiklik yapıyoruz.
13 //LogManager.LoadConfiguration(String.Concat(Directory.GetCurrentDirectory(), "/nlog.config"));
14 LogManager.Setup().LoadConfigurationFromFile(String.Concat(Directory.GetCurrentDirectory(), "/nlog.config"));
15
```

# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / Yapılandırma

NLog Program.cs konfigürasyonu



```
Program.cs  ✘ X
WebAPI
4  using WebAPI.Extensions;
5
6
7  var builder = WebApplication.CreateBuilder(args);
8
9  // Add services to the container.
10
11 //Nlog ayarlarını yapıyoruz.
12 //TODO: obsolete olayı için değişiklik yapıyoruz.
13 //LogManager.LoadConfiguration(String.Concat(Directory.GetCurrentDirectory(), "/nlog.config"));
14 LogManager.Setup().LoadConfigurationFromFile(String.Concat(Directory.GetCurrentDirectory(), "/nlog.config"));
15
```

# Sekizinci Bölüm?

## Nlog

### 3 üncü Part Kütüphane (NLog) / Yapılandırma

## Nlog ServicesExtensions ve Program.cs konfigürasyonu

The screenshot shows two code editors in Visual Studio. The top editor displays the `ServicesExtensions.cs` file, which contains configuration methods for `IServiceManager` and `ILoggerService`. The bottom editor displays the `Program.cs` file, which uses `builder.Services` to configure services like `ConfigureSqlContext`, `ConfigureRepositoryManager`, and `ConfigureLoggerService`. Both files include TODO comments explaining the purpose of each step. To the right, the Solution Explorer shows a multi-project solution named 'bsStoreApp' containing projects for Entities, Presentation, Repositories, Services, and WebAPI.

```
//TODO: IServiceProvider karşılık IServiceProvider olmuş.
public static void ConfigureServicesManager(this IServiceCollection services) => services.AddScoped<IServiceManager,
    ServiceManager>();

//TODO: Loger i kullanabilmemiz için ekliyoruz.
//TODO :AddSingleton yapıyoruz çünkü loger bir defa oluşturulacak ve her yerde kullanılacak.
public static void ConfigureLoggerService(this IServiceCollection services) => services.AddSingleton<ILoggerService,
    LoggerManager>();

25
26
27    //TODO: Program.cs kendi yazdığımız classı ekliyoruz.
28    //TODO: ServicesExtensions db bağlantısı taşıdığı için,
29    builder.Services.ConfigureSqlContext(builder.Configuration);
30    //TODO: Parametre kullanmadık. Çünkü ConfigureRepositoryManager metodu sadece this aldı.
31    builder.Services.ConfigureRepositoryManager();
32    //TODO: ServicesExtensions classında tanımladığımız ConfigureServicesManager kullanabilmek için ekliyoruz.
33    builder.Services.ConfigureServiceManager();
34    //TODO: ServicesExtensions classında tanımladığımız ConfigureLoggerService kullanabilmek için ekliyoruz.
35    builder.Services.ConfigureLoggerService();
```

# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / Nlog Mimaride Kullanımı

BookManager konfigürasyonu

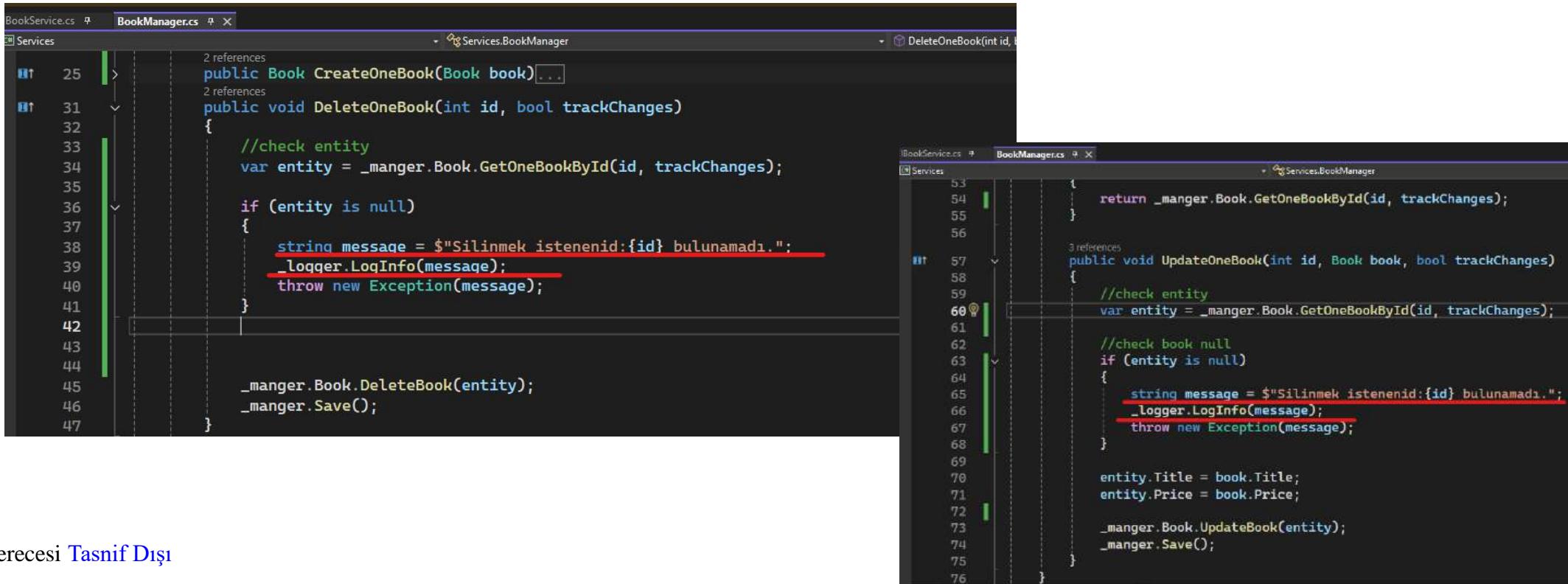
```
1  using System.Text;
2  using System.Threading.Tasks;
3
4  namespace Services
5  {
6      //TODO:internal erişim belirleyicini public olarak değiştirin.
7      public class BookManager : IBookService
8      {
9          //TODO: Manager ihtiyaç vardır.
10         private readonly IRepositoryManger _manger;
11
12         //TODO: ILoggerService ihtiyaç vardır.
13         private readonly ILoggerService _logger; //Ctrl + . yaparak ekleyin.
14
15         ...
16         private readonly ILoggerService _logger; //Ctrl + . yaparak ekleyin.
17         -public BookManager(IRepositoryManger repository)
18         +public BookManager(IRepositoryManger repository, ILoggerService logger)
19         {
20             _manger = repository;
21
22             _logger = logger;
23         }
24     ...
25     }
26 }
```

# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / Nlog Mimaride Kullanımı

BookManager konfigürasyonu



```
BookService.cs  BookManager.cs  Services
 25  public Book CreateOneBook(Book book){...}
 26
 27  31  public void DeleteOneBook(int id, bool trackChanges)
 28  {
 29      //check entity
 30      var entity = _manger.Book.GetOneBookById(id, trackChanges);
 31
 32      if (entity is null)
 33      {
 34          string message = $"Silinmek istenenid:{id} bulunamadi.";
 35          _logger.LogInfo(message);
 36          throw new Exception(message);
 37      }
 38
 39      _manger.Book.DeleteBook(entity);
 40      _manger.Save();
 41  }
 42
 43
 44
 45
 46
 47 }
```

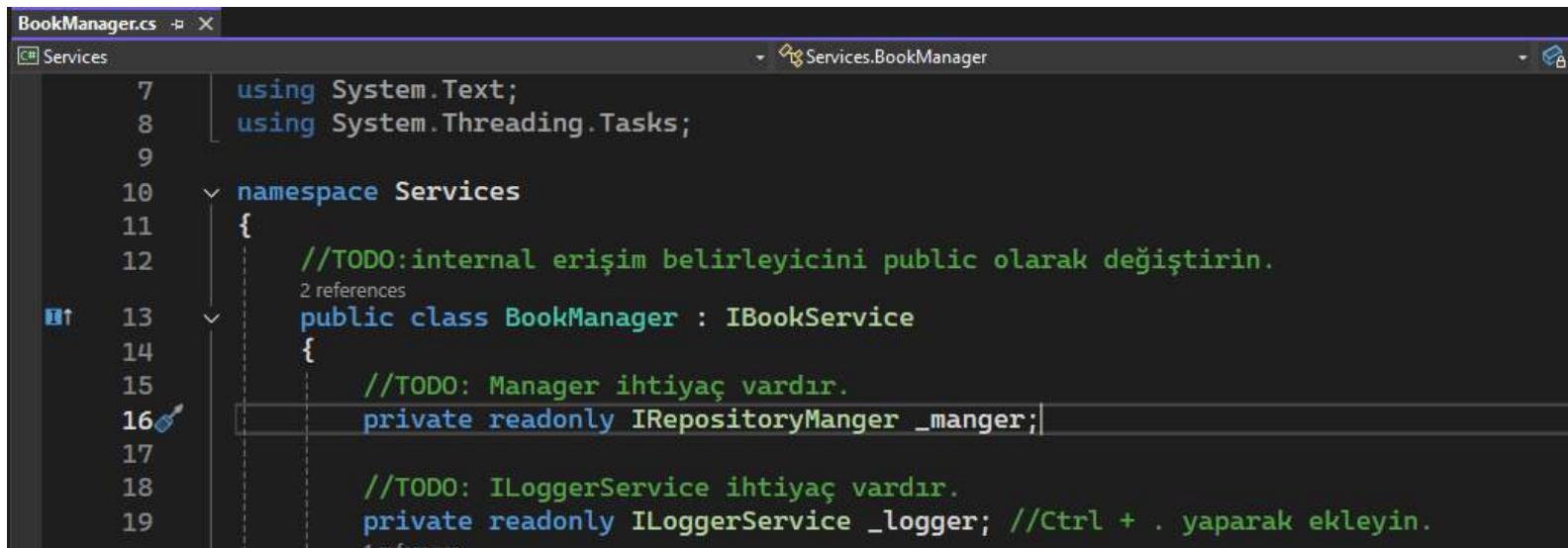
```
BookService.cs  BookManager.cs  Services
 53
 54  57  return _manger.Book.GetOneBookById(id, trackChanges);
 55
 56
 57
 58
 59
 60  60  public void UpdateOneBook(int id, Book book, bool trackChanges)
 61  {
 62      //check entity
 63      var entity = _manger.Book.GetOneBookById(id, trackChanges);
 64
 65      //check book null
 66      if (entity is null)
 67      {
 68          string message = $"Silinmek istenenid:{id} bulunamadi.";
 69          _logger.LogInfo(message);
 70          throw new Exception(message);
 71      }
 72
 73      entity.Title = book.Title;
 74      entity.Price = book.Price;
 75
 76      _manger.Book.UpdateBook(entity);
 77      _manger.Save();
 78  }
```

# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / Nlog Mimaride Kullanımı

BookManager bir sorunumuz var  
görebiliyor musunuz?



```
BookManager.cs
C# Services
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace Services
11 {
12     //TODO:internal erişim belirleyicini public olarak değiştirin.
13     public class BookManager : IBookService
14     {
15         //TODO: Manager ihtiyaç vardır.
16         private readonly IRepositoryManger _manger;
17
18         //TODO: ILoggerService ihtiyaç vardır.
19         private readonly ILoggerService _logger; //Ctrl + . yaparak ekleyin.
```

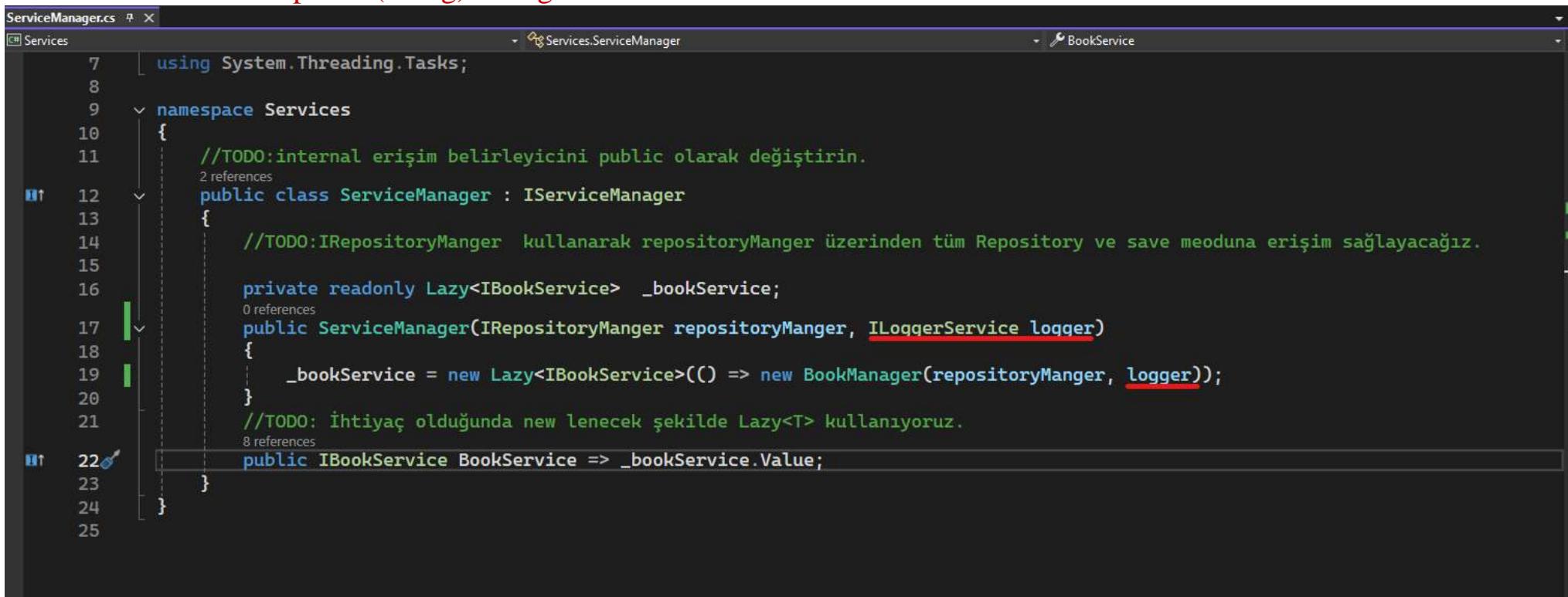
**BookManager da iki tane implemente  
oldu. Bu durum ServiceManager  
etkiler onu güncelleyelim !!!**

# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / Nlog Mimaride Kullanımı

ÇÖZÜM !!!



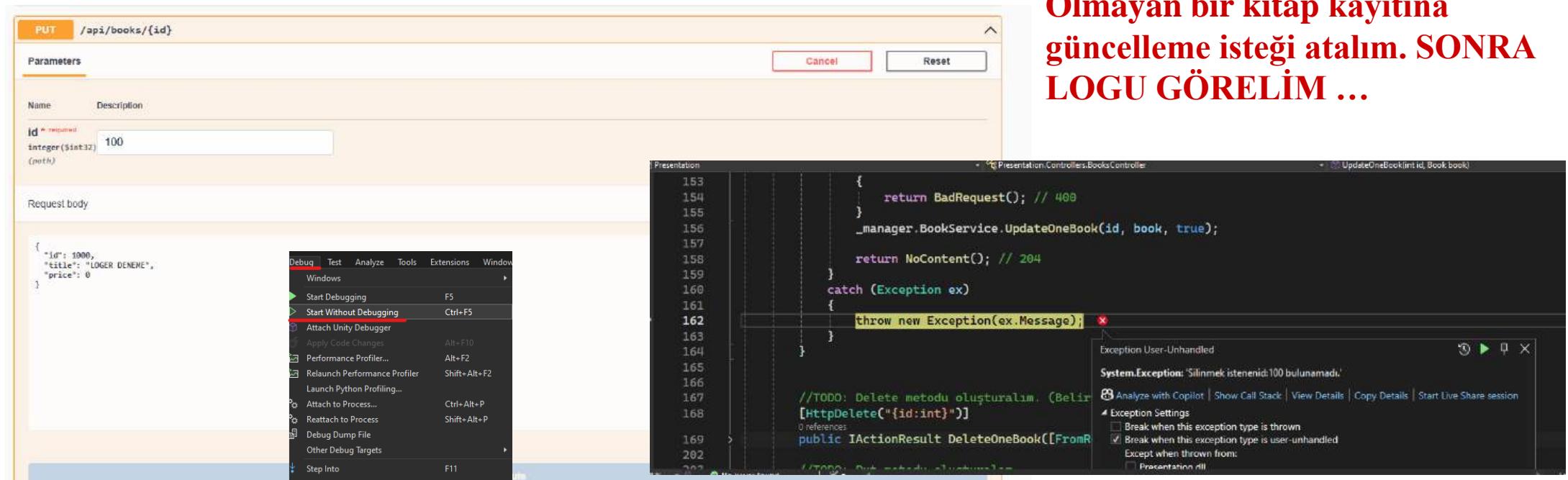
The screenshot shows the code editor in Visual Studio with the file 'ServiceManager.cs' open. The code is written in C# and defines a class 'ServiceManager' that implements 'IServiceManager'. The class uses a dependency injection pattern where it creates a new instance of 'BookService' using a factory method. The 'logger' parameter is underlined in red, indicating it is a missing dependency.

```
ServiceManager.cs
using System.Threading.Tasks;
namespace Services
{
    //TODO:internal erişim belirleyicini public olarak değiştirin.
    public class ServiceManager : IServiceManager
    {
        //TODO:IRepositoryManger kullanarak repositoryManger üzerinden tüm Repository ve save meoduna erişim sağlayacağız.
        private readonly Lazy<IBookService> _bookService;
        public ServiceManager(IRepositoryManger repositoryManger, ILoggerService logger)
        {
            _bookService = new Lazy<IBookService>(() => new BookManager(repositoryManger, logger));
        }
        //TODO: ihtiyaç olduğunda new lenenek şekilde Lazy<T> kullanıyoruz.
        public IBookService BookService => _bookService.Value;
    }
}
```

# Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / Nlog Mimaride Kullanımı



ŞİMDİ LOGER'ı test edelim !!!

Uygulamayı çalıştırıyalım.

Olmayan bir kitap kayıtına güncelleme isteği atalım. SONRA LOGU GÖRELİM ...

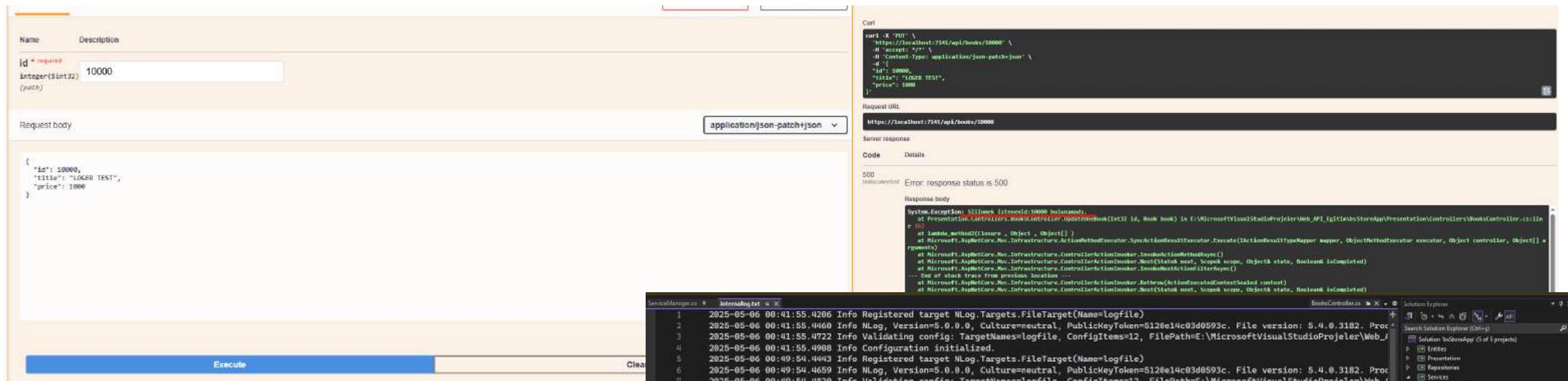
DEBUG Modda çalıştığı için direk durdu? Bunun  
için «Start Without Debugda çalıştır.»

## Sekizinci Bölüm?

Nlog

3 üncü Part Kütüphane (NLog) / Nlog Mimaride Kullanımı

**ŞİMDİ LOGER’ı test edelim !!!  
Uygulamayı çalıştıralım.  
Olmayan bir kitap kayıtına  
güncelleme isteği atalım. SONRA  
LOGU GÖRELİM ...**



## Gizlilik Derecesi Tasnif Dışı

# **Sekizinci Bölüm Bitti**

## **Hadi Özetalylim?**

# Dokuzuncu Bölüm?

## Global Hata Yönetimi

### try-catch vs. Global Exception Middleware

#### Özellik / Yaklaşım



**Kapsam**



**Merkezi Kontrol**



**Kod Tekrarı**



**Güvenlik (Mesaj Gizleme)**



**Loglama**



**Bakım Kolaylığı**

#### **try-catch Blokları**

Sadece belirli bir kod bloğu

Her yerde ayrı ayrı yazmak gereklidir

Fazla tekrar olur

Hata mesajını doğrudan söylebilir

Her blokta log yazmak gereklidir

Dağınık ve zor

#### **Global Exception Middleware**

Tüm uygulamayı kapsar

Tek bir yerde tanımlanır

Tekrarsız ve merkezi

Güvenli mesaj döner: "Internal Server Error"

Merkezi loglama yapılabilir

Kolay güncellenir, merkezi yapı

# Dokuzuncu Bölüm?

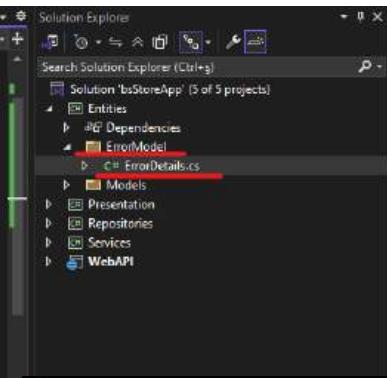
## Global Hata Yönetimi

Entities projesine **ErrorModel** klasörünü ekle.

İçine **ErrorDetails** Classını ekle ve ilgili kodu nu yaz!!!

```
namespace Entities.ErrorModel
{
    //TODO: internal erişim belirleyicini public olarak değiştirin.
    public class ErrorDetails
    {
        public int StatusCode { get; set; }
        public string? Message { get; set; }

        //TODO: Bu classın override edilecek metodun olması. ToString() metodunu override edin.
        public override string ToString()
        {
            //TODO: JsonSerializer ile serialize edin.
            //TODO: this ifadesi sınıfın tamamını ilgilendirdiği için kullanıyoruz.
            return JsonSerializer.Serialize(this);
        }
    }
}
```



```
{
    message: "...",
    statusCode:200
}
```

Serialize işlemi oldu. Bu işlemden de class a gidersek DeSerialize işlemi olur  
Örn NVI'den veri xml geliyor bunu ilgili model classına dönüştürüyoruz..!!!

# Dokuzuncu Bölüm?

## Global Hata Yönetimi Use Exception Handler

WebAPI projesinin Exception klasörün altına ExceptionMiddlewareExtensions classını ekle ve ilgili kodu nu yaz!!!

```
using System.Net;
using System.Threading.Tasks;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using WebAPI.Extensions;

public static class ExceptionMiddlewareExtensions
{
    public static void ConfigureExceptionHandler(this WebApplication app, ILoggerService logger)
    {
        app.UseExceptionHandler(appError =>
        {
            appError.Run(async context =>
            {
                context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;
                context.Response.ContentType = "application/json";
                var contextFeature = context.Features.Get<IExceptionHandlerFeature>();

                if (contextFeature != null)
                {
                    logger.LogError($"Bir seyler ters gitti: {contextFeature.Error}");
                    await context.Response.WriteAsync(new ErrorDetails
                    {
                        StatusCode = context.Response.StatusCode,
                        Message = "Internal Server Error."
                    }.ToString());
                }
            });
        });
    }
}
```

ConfigureExceptionHandler metodu, uygulama genelinde geçerli olacak bir hata yakalama mekanizması kurar.

Herhangi bir istisna oluştuğunda:

- HTTP yanıt kodu 500 (Internal Server Error) olarak ayarlanır.
- Hata detayları loglanır (ILoggerService ile).
- Kullanıcıya sade bir JSON yanıt gönderilir: "Internal Server Error".

# Dokuzuncu Bölüm?

Global Hata Yönetimi

Use Exception Handler Yapılandırılması

Program.cs de ilgili değişiklikler  
yapılacak!!!

```
Program.cs 38
39
40
41
42 var app = builder.Build();
43
44 //TODO: app ne zaman elde ediliyor? Services işlemleri tamamlandıktan sonra elde ediliyor.
45 //TODO: app oluşturduk ancak ConfigureExceptionHandler diye metod yazdık. Buda ILoggerService alıyor. Ne yapacağız.
46 //TODO: Bu işlemi GetRequiredService metodu ile yapıyoruz.
47 var logger = app.Services.GetRequiredService<ILoggerService>();
48 app.ConfigureExceptionHandler(logger);
```

```
Program.cs 47    // Configure the HTTP request pipeline.
48    if (app.Environment.IsDevelopment())
49    {
50        app.UseSwagger();
51        app.UseSwaggerUI();
52    }
53
54    if (app.Environment.IsProduction())
55    {
56        app.UseHsts();
57    }
58
```

# Dokuzuncu Bölüm?

Global Hata Yönetimi  
Test

Olmayan kitap için hata döndü biz şimdi  
kendi exceptionu test edelim!!!

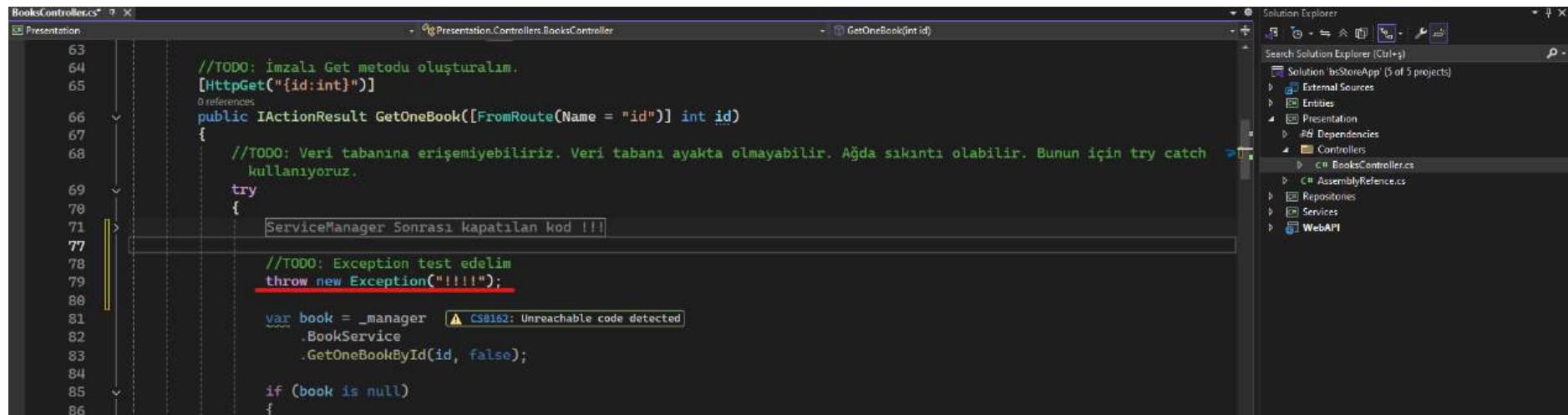
The screenshot shows a Postman collection named 'Books'. A GET request is selected with the URL `(baseUrl_WebAPI) /api/books/69`. The request body is empty, indicated by the message 'This request does not have a body.' In the response tab, a 404 Not Found error is shown with the following JSON payload:

```
{  
  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.4",  
  "title": "Not Found",  
  "status": 404,  
  "traceId": "00-a822aa087738943a773c6edabce96cf-aa3f24230d801fe0-00"  
}
```

# Dokuzuncu Bölüm?

Global Hata Yönetimi  
Test

Uygulamayı çalıştır.  
**ANCAK «Start Without Debugging» ile!!!** Debug modda olduğu için ekranın bir şey göremeyiz onun için öyle çalıştık



The screenshot shows the Visual Studio IDE interface. On the left is the code editor window titled 'BooksController.cs' with the following C# code:

```
63 //TODO: Imzalı Get metodunu oluşturalım.
64 [HttpGet("{id:int}")]
65 0 references
66 public IActionResult GetOneBook([FromRoute(Name = "id")] int id)
67 {
68     //TODO: Veri tabanına erişemeyeceğiz. Veri tabanı ayakta olmayabilir. Ağda sıkıntı olabilir. Bunun için try catch kullanıyoruz.
69     try
70     {
71         ServiceManager Sonrası Kapatılan Kod !!!
72     }
73     //TODO: Exception test edelim
74     throw new Exception("!!!!");
75
76     var book = _manager
77         .BookService
78         .GetOneBookById(id, false);
79
80     if (book is null)
81     {
82
83
84
85
86 }
```

The code editor has several TODO comments and a try-catch block. Line 71 contains the text 'ServiceManager Sonrası Kapatılan Kod !!!'. Line 74 contains the code 'throw new Exception("!!!!");'. Line 76 contains the code 'var book = \_manager .BookService .GetOneBookById(id, false);'. Line 80 contains the code 'if (book is null) {'. A warning CS0162: Unreachable code detected is shown next to the line 76 code.

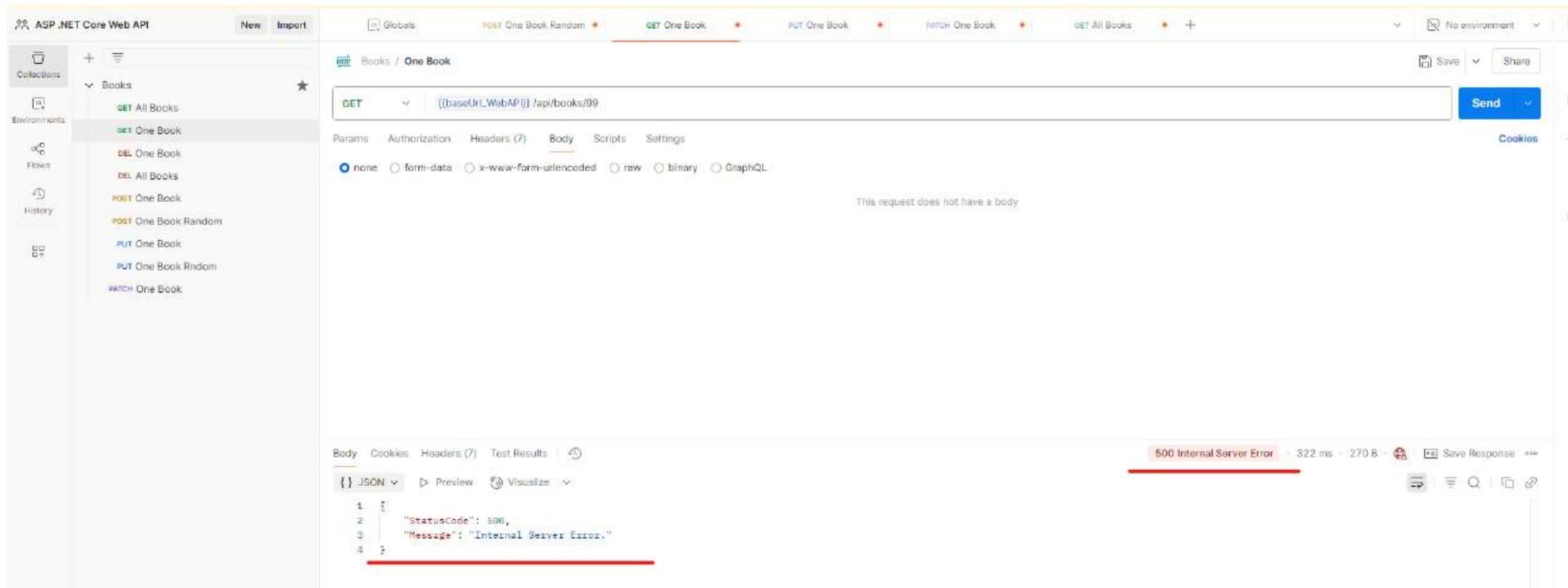
On the right side of the interface is the 'Solution Explorer' window, which lists the project structure:

- Solution 'bsStoreApp' (5 of 5 projects)
  - External Sources
  - Entities
  - Presentation
    - # Dependencies
    - Controllers
      - C# BooksController.cs
      - C# AssemblyReference.cs
    - Repositories
    - Services
    - WebAPI

# Dokuzuncu Bölüm?

Global Hata Yönetimi  
Test

Serialize edilmiş hatayı gördük.  
Bu bize ne kazandıracak,  
Çok daha temiz kod sağlamamızı  
sağlayacak.  
**Try-Catch den kurtulacağız.**



The screenshot shows the Postman interface with the following details:

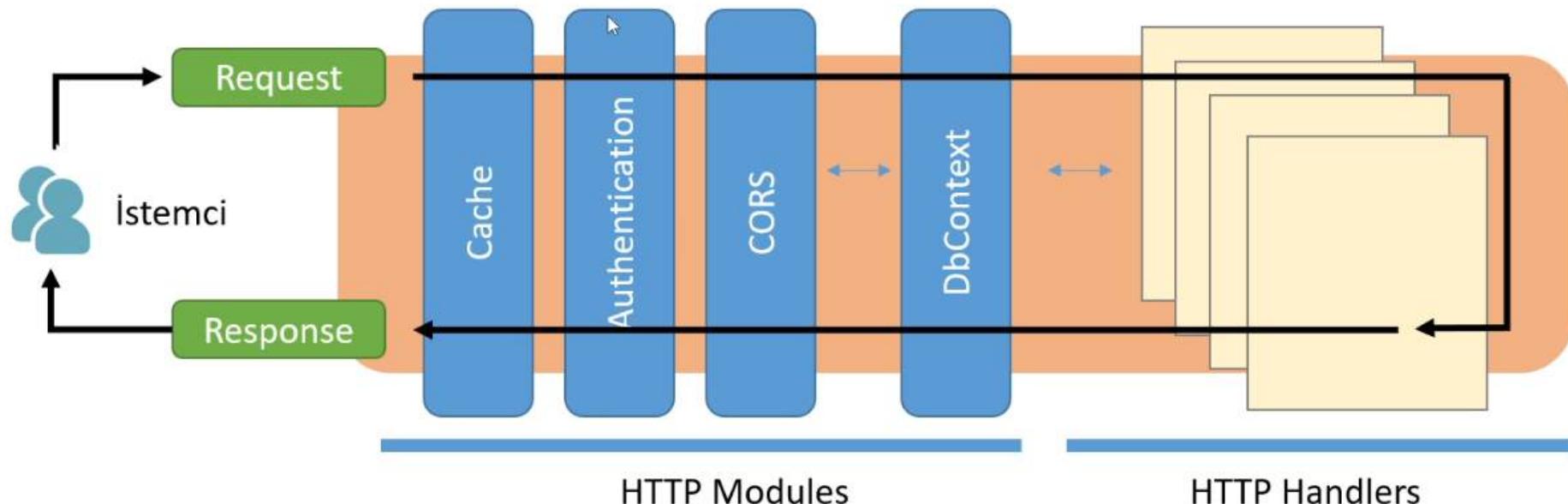
- Collection:** ASP .NET Core Web API
- Environment:** No environment
- Request:** Books / One Book (GET)
- URL:** {{baseUrl}}/WebAPI/api/books/99
- Method:** GET
- Headers:** Authorization, Headers (7), Body, Scripts, Settings
- Body:** None selected
- Response Status:** 500 Internal Server Error
- Response Body:**

```
{ "StatusCode": 500, "Message": "Internal Server Error." }
```

# Dokuzuncu Bölüm?

Global Hata Yönetimi

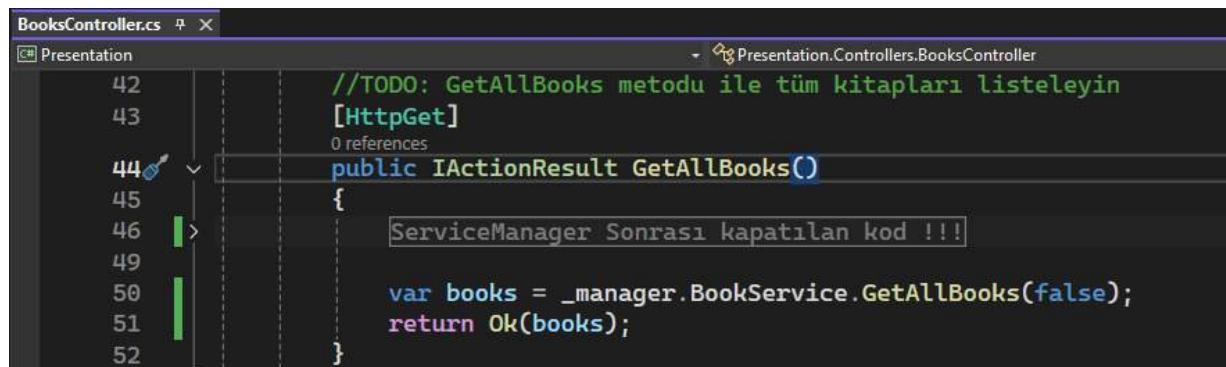
Try-Catch Blokların kaldırılması



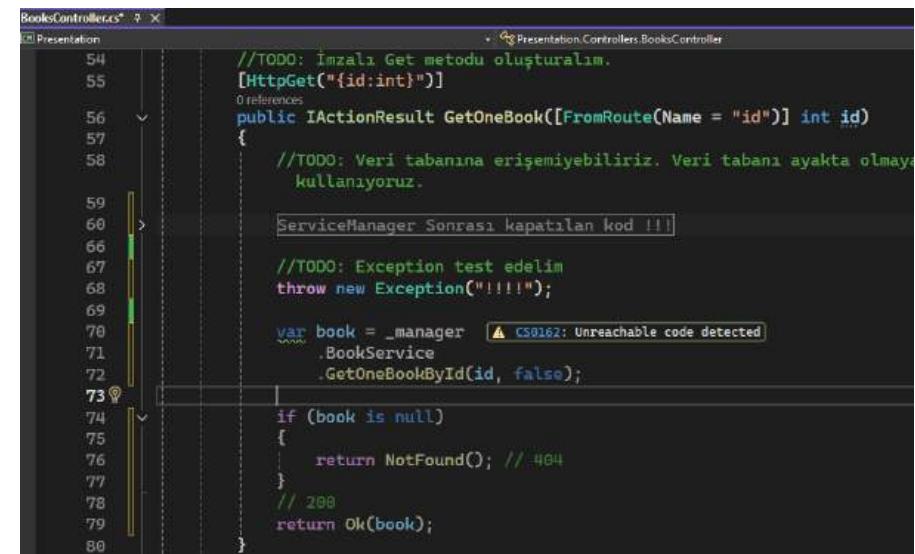
# Dokuzuncu Bölüm?

## Global Hata Yönetimi

### Try-Catch Blokların kaldırılması



```
BooksController.cs
42 //TODO: GetAllBooks metodu ile tüm kitapları listeleyin
43 [HttpGet]
44 public IActionResult GetAllBooks()
45 {
46     ServiceManager Sonrası kapatılan kod !!!
47
48     var books = _manager.BookService.GetAllBooks(false);
49     return Ok(books);
50 }
51 }
```



```
BooksController.cs
54 //TODO: Imzalı Get metodu oluşturalım.
55 [HttpGet("id:int")]
56
57 public IActionResult GetOneBook([FromRoute(Name = "id")] int id)
58 {
59     ServiceManager Sonrası kapatılan kod !!!
60
61     //TODO: Veri tabanına erişemeyebiliriz. Veri tabanı ayakta olmayan
62     //kullanıyoruz.
63
64     //TODO: Exception test edelim
65     throw new Exception("!!!!!");
66
67     var book = _manager
68         .BookService
69         .GetOneBookById(id, false);
70
71     if (book is null)
72     {
73         return NotFound(); // 404
74     }
75     // 200
76     return Ok(book);
77 }
78 }
```

# Dokuzuncu Bölüm?

## Global Hata Yönetimi

Try-Catch Blokların kaldırılması

```
BooksController.cs*  X Presentation.Controllers.BooksController
82 //TODO: Post metodu oluşturalım.
83 [HttpPost]
84 public IActionResult CreateOneBook([FromBody] Book book)
85 {
86     if (book is null)
87     {
88         return BadRequest(); // 400
89     }
90     ServiceManager Sonrası kapatılan kod !!!
91
92     var bookCreate = _manager
93         .BookService
94         .CreateOneBook(book);
95
96     return StatusCode(201, book); // 201
97
98 }
99
100 }
```

```
BooksController.cs*  X Presentation.Controllers.BooksController
102 //TODO: Put metodu oluşturalım. (Güncelleme yapmak için)
103 [HttpPut("{id:int}")]
104 public IActionResult UpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] Book book)
105 {
106     //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
107     ServiceManager Sonrası kapatılan kod !!!
108
109     if (book is null)
110     {
111         return BadRequest(); // 400
112     }
113     _manager.BookService.UpdateOneBook(id, book, true);
114
115     return NoContent(); // 204
116
117 }
```

# Dokuzuncu Bölüm?

## Global Hata Yönetimi

### Try-Catch Blokların kaldırılması

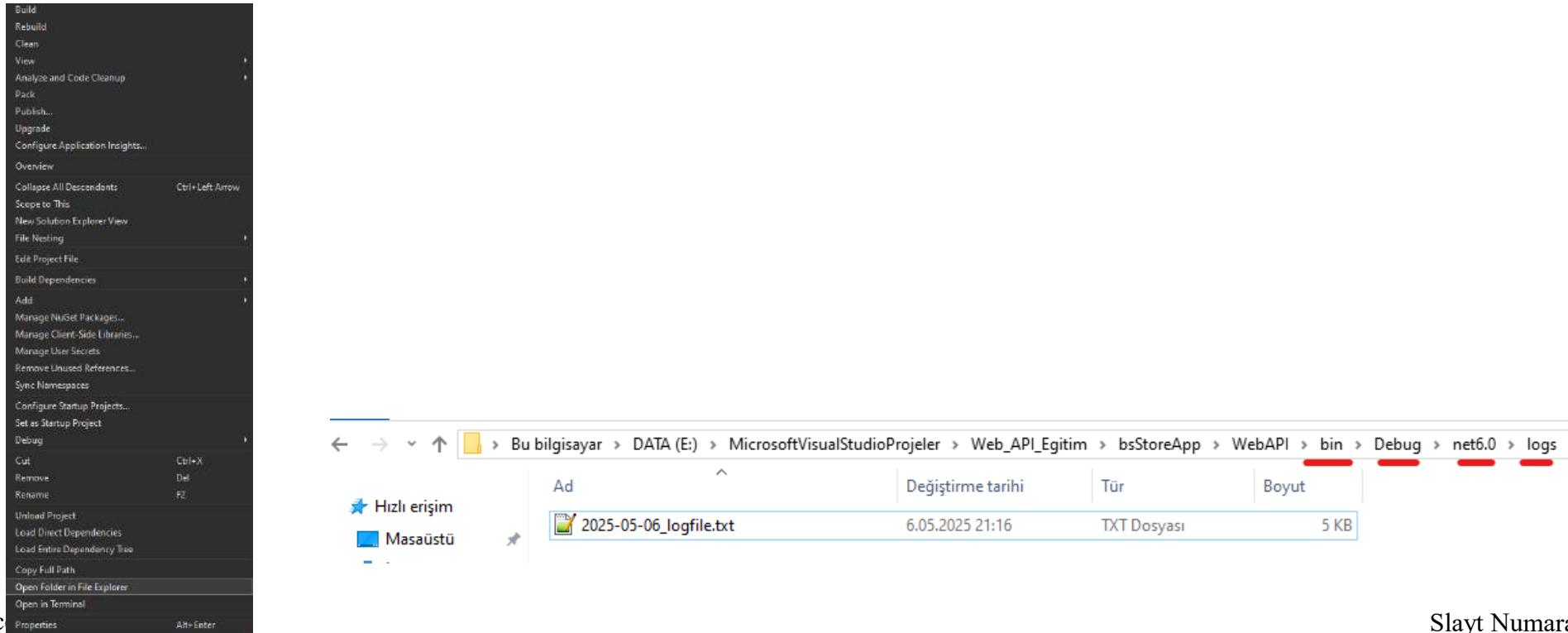
```
BooksController.cs*  ↗ X ↗ Presentation.Controllers.BooksController
161 //TODO: Put metodu oluşturalım.
162 [HttpPatch("{id:int}")]
163 public IActionResult PartiallyUpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] JsonPatchDocument<Book>
164     bookPatch)
165 {
166     //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
167     var entity = _manager
168         .BookService
169         .GetOneBookById(id, true);
170     //TODO: Güncelleme yapmak için gelen book null mı kontrol et.
171     if (entity is null)
172         return NotFound(); // 404
173     bookPatch.ApplyTo(entity);
174     _manager.BookService.UpdateOneBook(id, entity, true);
175
176     return NoContent(); // 204
177 }
178 }
```

```
BooksController.cs*  ↗ X ↗ Presentation.Controllers.BooksController
134 //TODO: Delete metodu oluşturalım. (Belirtilen sil)
135 [HttpDelete("{id:int}")]
136 public IActionResult DeleteOneBook([FromRoute(Name = "id")] int id)
137 {
138     ServiceManager Sonrası kapatılan kod !!!
139
140     _manager.BookService.DeleteOneBook(id, true);
141     return NoContent(); // 204
142 }
```

# Dokuzuncu Bölüm?

Global Hata Yönetimi  
Diğer Hata Görme



# Dokuzuncu Bölüm?

## Global Hata Yönetimi

### Not Found Exception Hataları Yakalama

Entities projesinin Exceptions klasörünü oluştur.

Altına NotFound ve BookNotFound classını ekle sonra ilgili kodu nu yaz!!!

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays the `NotFound.cs` file under the `Entities` project. The code defines an abstract class `NotFound` that inherits from `Exception`. The code includes several TODO comments and a protected constructor. On the right, the Solution Explorer shows the solution structure for 'bsStoreApp' with five projects: Entities, Dependencies, ErrorModel, Exceptions (which contains `BookNotFound.cs` and `NotFound.cs`), Models, Presentation, Repositories, Services, and WebAPI.

```
using System.Text;
using System.Threading.Tasks;

namespace Entities.Exceptions
{
    //TODO:internal erişim belirleyicini public olarak değiştirin.
    //Bu sınıf new leme yapılmaması için abstract sınıf olarak tanımlanmıştır.
    3 references
    public abstract class NotFound : Exception
    {
        //TODO: Bu sınıf new leme yapılmadığından protected constructor tanımlanmıştır.
        //TODO: Sadece devr alınan classlarda kullanacağı için
        1 reference
        protected NotFound(string message) : base(message)
    }
}
```

# Dokuzuncu Bölüm?

## Global Hata Yönetimi

### Not Found Exception Hataları Yakalama

Entities projesinin Exceptions klasörünü oluştur.

Altına NotFound ve BookNotFound classını ekle sonra ilgili kodu nu yaz!!!

The screenshot shows the Visual Studio IDE with the Entity project open. On the left, the code editor displays two files: `NotFound.cs` and `BookNotFound.cs`. The `NotFound.cs` file contains the definition for the `NotFoundException` class, which is an abstract class derived from `Exception`. The `BookNotFound.cs` file contains the definition for the `BookNotFoundException` class, which is a sealed class derived from `NotFoundException`. The Solution Explorer on the right shows the project structure, including the `Exceptions` folder where these files are located. The code for `BookNotFoundException` includes a constructor that takes an integer parameter and a message string.

```
namespace Entities.Exceptions
{
    //TODO:internal erişim belirleyicini public olarak değiştirin.
    //Bu sınıf new leme yapılmaması için abstract sınıf olarak tanımlanmıştır.
    public abstract class NotFound : Exception
    {
        //TODO: Bu class new leme yapılmadığından protected constructor tanımlanmıştır.
        //TODO: Sadece devr alınan classlarda kullanılacağı için
        protected NotFound(string message) : base(message)
        {
        }
    }
}

namespace Entities.Exceptions
{
    //TODO: sealed class kalıtım alınmasını engeller.
    public sealed class BookNotFoundException : NotFoundException
    {
        0 references
        public BookNotFoundException(int id) : base($"{id} ye sahip kitap bulunamadı !!!")
    }
}
```

# Dokuzuncu Bölüm?

Global Hata Yönetimi  
Use Exception Handler Yakalama

NotFound classında hata yaptıktı  
NotFoundException olacaktı.  
Değiştir !!! Aşağıda kısa yol var !!!

The screenshot shows a Visual Studio code editor with two tabs: 'NotFound.cs' and 'ExceptionMidd...Extensions.cs'. The 'NotFound.cs' tab contains the following code:using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Entities.Exceptions
{
 //TODO:internal erişim belirleyicini public olarak değiştirin.
 //Bu sınıf new leme yapılmaması için abstract sınıf olarak tanımlanmıştır.
 public abstract class **NotFoundException** : Exception
}A context menu is open at the end of the 'NotFoundException' class definition, with the option 'Rename 'NotFoundException' to 'NotFoundException'' highlighted. A tooltip provides additional information: '**TODO: Sadece devr alınan classlarda kullanılacağı için**' followed by the original code:

```
-protected NotFoundException(string message) : base(message)
+protected NotFoundException(string message) : base(message)
```

# Dokuzuncu Bölüm?

Global Hata Yönetimi  
Use Exception Handler Yakalama

NotFound classında hata yaptıktı  
NotFoundException olacaktı.  
Değiştir !!! Aşağıda kısa yol var !!!

The screenshot shows a Visual Studio code editor with two tabs: 'NotFound.cs' and 'ExceptionMidd...Extensions.cs'. The 'NotFound.cs' tab contains the following code:using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Entities.Exceptions
{
 //TODO:internal erişim belirleyicini public olarak değiştirin.
 //Bu sınıf new leme yapılmaması için abstract sınıf olarak tanımlanmıştır.
 public abstract class **NotFoundException** : Exception
}A context menu is open at the end of the 'NotFoundException' class definition. The menu items are:- Rename 'NotFound' to 'NotFoundException'
- Generate constructor 'NotFoundException()'
- Generate constructor 'NotFoundException(info, context)'
- Generate constructor 'NotFoundException(message, innerException)'
- Generate all
- Rename file to NotFoundException.cs
- Rename type to NotFound
- Move to namespace...
- Generate Equals(object)...
- Generate Equals and GetHashCode...
- Generate overrides...
- Add 'DebuggerDisplay' attribute
A preview changes dialog is visible, showing the addition of protected constructors for the NotFoundException class.

# Dokuzuncu Bölüm?

Global Hata Yönetimi

Use Exception Handler Yakalama

ExceptionMiddlewareExtensions  
classını güncelle !!!

```
public static class ExceptionMiddlewareExtensions
{
    public static void ConfigureExceptionHandler(this WebApplication app, ILoggerService logger)
    {
        app.UseExceptionHandler(appError =>
        {
            appError.Run(async context =>
            {
                //TODO: Use ExceptionHandler yakaladığımız için artık .(int) HttpStatusCode.InternalServerError; siliyoruz. Artık 500 ve 400 hataları olabilir.
                //context.Response.StatusCode = (int) HttpStatusCode.InternalServerError;

                context.Response.ContentType = "application/json";
                var contextFeature = context.Features.Get<IExceptionHandlerFeature>();

                if (contextFeature is not null)
                {
                    context.Response.StatusCode = contextFeature.Error switch
                    {
                        NotFoundException => StatusCodes.Status404NotFound, //TODO: NotFound ise 404 değil ise 500
                        _ => StatusCodes.Status500InternalServerError
                    };
                }

                logger.LogError($"Bir şeyler ters gitti: {contextFeature.Error}");
                await context.Response.WriteAsync(new ErrorDetails()
                {
                    StatusCode = context.Response.StatusCode,
                    Message = contextFeature.Error.Message//TODO: contextFeature.Error.Message ile hata mesajını alıyoruz.
                }.ToString());
            });
        });
    }
}
```

# Dokuzuncu Bölüm?

Global Hata Yönetimi  
Refactoring

BookNotFound classında hata yaptıktı  
BookNotFoundException olacaktı.  
Değiştir !!! Aşağıda kısa yol var !!!

The screenshot shows two instances of Visual Studio. The top instance displays the `BookNotFoundException.cs` file with the following code:

```
namespace Entities.Exceptions
{
    //TODO: sealed class kalıtım alınmasını engeller.
    public sealed class BookNotFoundException : NotNotFoundException
    {
        public BookNotFoundException(int id) : base($"{id} ye sahip kitap bulunamadı !!!") { }
    }
}
```

A tooltip from the IDE indicates a method return type error: `CS1520: Method must have a return type`. Below the code, a context menu is open with the option `Rename "BookNotFoundException" to "BookNotFoundException"` highlighted.

The bottom instance shows the same code after the rename operation, with the new class name `BookNotFoundException` highlighted in the code editor.

On the right side of the interface, the Solution Explorer shows the project structure:

- Solution: 'isStoreApp' (5 of 5 projects)
  - Entities
    - RG Dependencies
    - EntityModel
    - Exceptions
      - C# BookNotFoundException.cs
      - C# NotFoundException.cs
    - Models
    - Presentation
    - Repositories
    - Services
  - WebAPI

# Dokuzuncu Bölüm?

Global Hata Yönetimi  
Refactoring

GetOneBook metodunu Postman da test et.

```
BooksController.cs
[HttpGet("{id:int}")]
public IActionResult GetOneBook([FromRoute(Name = "id")] int id)
{
    //TODO: Veri tabanına erişemeyiz. Veri tabanı ayakta olmayabilir.
    //ServiceManager Sonrası kapatılan kod !!!
    var book = _manager
        .BookService
        .GetOneBookById(id, false);
    //TODO: Artık NotFound servise taşıyalım !!!
    return Ok(book);
}

BookManager.cs
public Book GetOneBookById(int id, bool trackChanges)
{
    var book = _manger.Book.GetOneBookById(id, trackChanges);
    if (book is null)
        throw new BookNotFoundException(id);
    return book;
}
```

# Dokuzuncu Bölüm?

Global Hata Yönetimi  
Refactoring

Test SONUCU !!!

The screenshot shows a Postman collection named "Books / One Book". The left sidebar lists various API endpoints. The main interface shows a GET request to `((baseUrl_WebAPI)) /api/books/99`. The "Body" tab is selected, showing the response body as JSON:

```
{  
  "StatusCode": 404,  
  "Message": "99 ye sahip kitap bulunamadı !!!"  
}
```

The status bar at the bottom right indicates a 404 Not Found error, a response time of 1.44 s, and a response size of 273 B.

# Dokuzuncu Bölüm?

Global Hata Yönetimi  
Refactoring

**DeleteOneBook** metodunda ilgili  
güncelleme yap !!!

The screenshot shows two open code files in Visual Studio:

- BooksController.cs**:  
A C# file containing HTTP methods for managing books. It includes TODO comments for creating Put and Delete methods. The DeleteOneBook method calls `_manager.BookService.DeleteOneBook()`. A context menu is open over this line, showing options like "Quick Actions and Refactorings...", "Rename...", "Remove and Sort Usings...", "Peek Definition", "Go To Definition", "Go To Base", and "Go To Implementation".
- BookManager.cs**:  
A C# file containing the implementation of the BookService interface. The `DeleteOneBook` method takes an `int id` and a `bool trackChanges`. It checks if an entity exists, throws a `BookNotFoundException` if it does not, and then deletes the entity and saves changes.

```
BooksController.cs
98 //TODO: Put metodu oluşturalım. (Güncelleme yapmak için)
99 [HttpPut("{id:int}")]
100 public IActionResult UpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] Book book)...
101
102 //TODO: Delete metodu oluşturalım. (Belirtilen sil)
103 [HttpDelete("{id:int}")]
104 public IActionResult DeleteOneBook([FromRoute(Name = "id")] int id)
105 {
106     ServiceManager Sonrası Kapatılan kod !!!
107
108     _manager.BookService.DeleteOneBook(id);
109     return NoContent(); // 204
110 }
111
112 //TODO: Put metodu oluşturalım.
113 [HttpPatch("{id:int}")]
114 public IActionResult PartiallyUpdate...
115 {
116     //TODO: Güncelleme yapmak için g...
117 }
```

```
BookManager.cs
32 public void DeleteOneBook(int id, bool trackChanges)
33 {
34     //check entity
35     var entity = _manger.Book.GetOneBookById(id, trackChanges);
36
37     if (entity is null)
38         throw new BookNotFoundException(id);
39
40     _manger.Book.DeleteBook(entity);
41     _manger.Save();
42 }
43 }
```

# Dokuzuncu Bölüm?

Global Hata Yönetimi  
Refactoring

UpdateOneBook metodunda ilgili  
güncelleme yap !!!

```
BooksController.cs
98 //TODO: Put metodu oluşturalım. (Güncelleme yapmak için)
99 [HttpPut("{id:int}")]
100 public IActionResult UpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] Book book, [FromServices] IBookService _manger)
101 {
102     if (_manger == null)
103     {
104         return BadRequest("Book manager is null");
105     }
106     var entity = _manger.BookService.UpdateOneBook(id, book);
107     return Ok(entity);
108 }
109
110 //TODO: Delete metodu oluşturalım.
111 [HttpDelete("{id:int}")]
112 public IActionResult DeleteOneBook([FromRoute(Name = "id")] int id)
113 {
114     var entity = _manger.BookService.GetOneBookById(id);
115     if (entity == null)
116     {
117         return NotFound();
118     }
119     _manger.BookService.DeleteOneBook(id);
120     return Ok();
121 }
```

```
BookManager.cs
49 public Book GetOneBookById(int id, bool trackChanges)
50 {
51     var entity = _manger.Book.GetOneBookById(id, trackChanges);
52     if (entity == null)
53     {
54         throw new BookNotFoundException(id);
55     }
56     entity.Title = book.Title;
57     entity.Price = book.Price;
58     _manger.Book.UpdateBook(entity);
59     _manger.Save();
60 }
```

# Dokuzuncu Bölüm Bitti

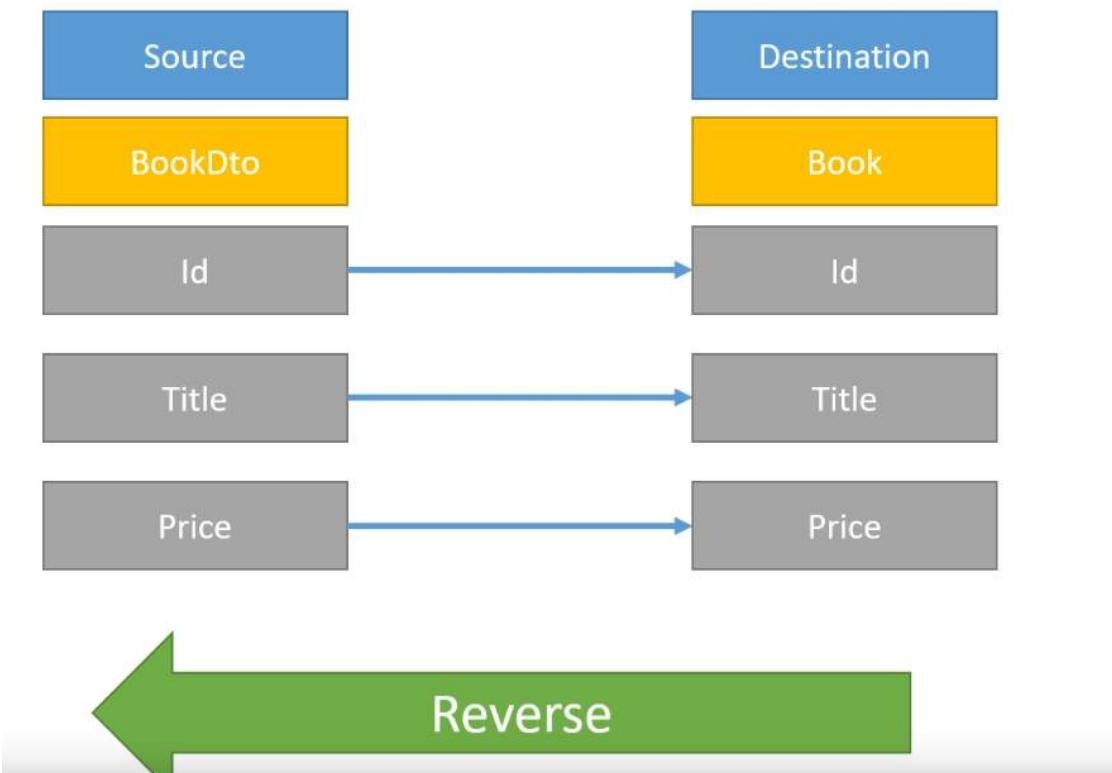
## Hadi Özetleyelim?

UseException  
Handler

- Error Model
- Error Details (Serialize)
- Configure Exception Handler
- UseExceptionHandler
- try-catch-finally
- NotFoundException (abstract)
- BookNotFoundException (sealed)
- Improve Configure Exception Handler

## Onuncu Bölüm?

Automapper



Elimizde DTO var bunu Destination ile otomatik yapılmasını **Automapper kütüphanesi** ile yapacağız. Bu işlem birbirinin tersi olarak da olabilir !!!

- Auto Mapper
- Record type
- Paket Kurulumu
- Mapping Profile
- Servis Kaydı
- Manager Düzenlenmesi
- Sunum Katmanının Düzenlenmesi

# Onuncu Bölüm?

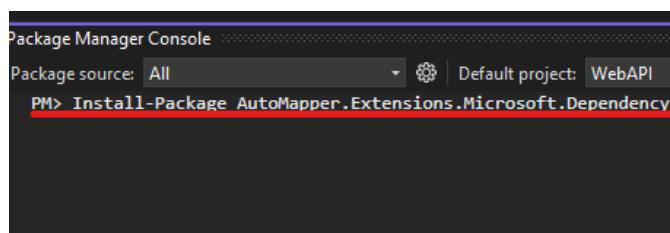
Automapper

Kurulum / Konfigürasyon

Install-Package

AutoMapper.Extensions.Microsoft.DependencyInjection -ProjectName Services  
kurulumunu yap!!!

Sonra eklenmiş mi? kontrol et

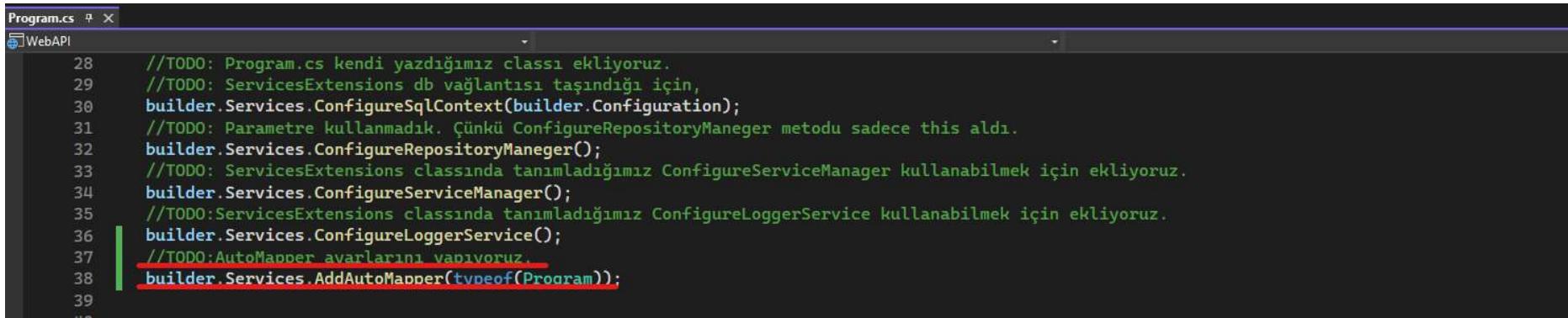


```
Services* < X
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="12.0.1" />
    <PackageReference Include="NLog.Extensions.Logging" Version="5.4.0" />
  </ItemGroup>
  <ItemGroup>
    <ProjectReference Include="..\Entities\Entities.csproj" />
    <ProjectReference Include="..\Repositories\Repositories.csproj" />
  </ItemGroup>
</Project>
```

# Onuncu Bölüm?

Automapper

Kurulum / Konfigürasyon

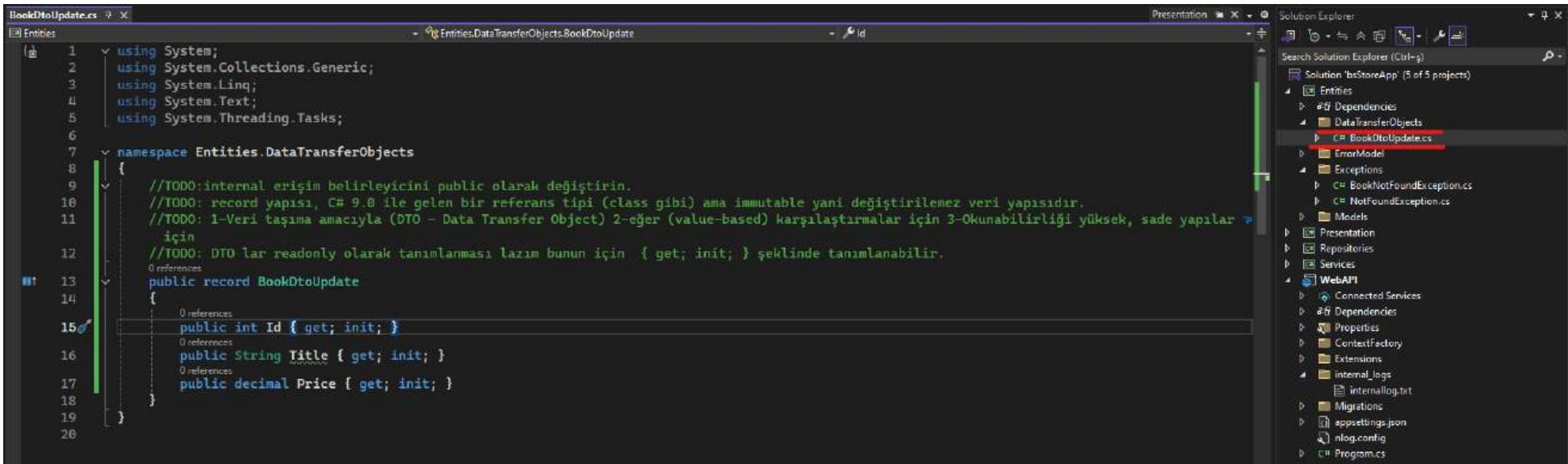


```
Program.cs  ✘
WebAPI
28     //TODO: Program.cs kendi yazdığımız classı ekliyoruz.
29     //TODO: ServicesExtensions db vağlantısı taşındığı için,
30     builder.Services.ConfigureSqlServer(builder.Configuration);
31     //TODO: Parametre kullanmadık. Çünkü ConfigureRepositoryManeger metodu sadece this aldı.
32     builder.Services.ConfigureRepositoryManeger();
33     //TODO: ServicesExtensions classında tanımladığımız ConfigureServiceManager kullanabilmek için ekliyoruz.
34     builder.Services.ConfigureServiceManager();
35     //TODO: ServicesExtensions classında tanımladığımız ConfigureLoggerService kullanabilmek için ekliyoruz.
36     builder.Services.ConfigureLoggerService();
37     //TODO: AutoMapper avarlarını tanıvoruz.
38     builder.Services.AddAutoMapper(typeof(Program));
39 
```

# Onuncu Bölüm?

Automapper  
Data Transfer Object (DTO)

Entities projesinin  
**DataTransferObjects** klasörünü oluştur.  
Altına **BookDtoUpdate** classını ekle  
sonra ilgili kodu nu yaz!!!



The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays the `BookDtoUpdate.cs` file within the `Entities` project. The code defines a record type `BookDtoUpdate` with three properties: `Id`, `Title`, and `Price`, each annotated with `[init;]`. The code editor has syntax highlighting and line numbers from 1 to 28. On the right, the Solution Explorer pane shows the solution structure for `b3StoreApp`, which includes five projects: `Entities`, `Presentation`, `Repositories`, `Services`, and `WebAPI`. The `Entities` project is expanded, showing its subfolders and files, including the `BookDtoUpdate.cs` file.

```
BookDtoUpdate.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Entities.DataTransferObjects
{
    //TODO:internal erişim belirleyicini public olarak değiştirin.
    //TODO: record yapısı, C# 9.0 ile gelen bir referans tipi (class gibi) ama immutable yani değiştirilemez veri yapısıdır.
    //TODO: 1-Veri taşıma amacıyla (DTO - Data Transfer Object) 2-eğer (value-based) karşılaştırmalar için 3-Okunabilirliği yüksek, sade yapılar
    //için
    //TODO: DTO lar readonly olarak tanımlanması lazım bunun için { get; init; } şeklinde tanımlanabilir.
    public record BookDtoUpdate
    {
        public int Id { get; init; }
        public string Title { get; init; }
        public decimal Price { get; init; }
    }
}
```

# Onuncu Bölüm?

Automapper  
Mapping Profile

WebAPI projesinin Utilities klasörünü oluştur. Bunun altına AutoMapper klasörünü oluştur.  
Altına **MappingProfile** classını ekle sonra ilgili kodu nu yaz!!!

The screenshot shows the Visual Studio IDE interface. On the left is the code editor window titled 'MappingProfile.cs' with the following C# code:

```
1  using AutoMapper;
2  using Entities.DataTransferObjects;
3  using Entities.Models;
4
5  namespace WebAPI.Utilities.AutoMapper
6  {
7      //TODO : MappingProfile class Profile dan kalıtım alacak
8      public class MappingProfile : Profile
9      {
10         public MappingProfile()
11         {
12             //TODO : Mapping işlemleri burada yapılacak
13             //TODO:CreateMap<Source, Destination>();
14
15             CreateMap<BookDtoUpdate, Book>();
16         }
17     }
18 }
19
20 }
```

On the right is the 'Solution Explorer' window, which lists the project structure. It shows a 'WebAPI' project containing several subfolders like 'Entities', 'Presentation', 'Repositories', 'Services', 'Utilities', and 'WebAPI'. Inside the 'Utilities' folder, there is a 'AutoMapper' folder containing the 'C# MappingProfile.cs' file, which is highlighted with a red underline.

# Onuncu Bölüm?

## Automapper

### Mapping Profile / AutoMapper Kullanımı

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays `BookManager.cs` with several TODO comments and constructor parameters. In the center, the code editor shows the `UpdateOneBook` method. A context menu is open over the `_manger` variable, with the "Go To Definition" option highlighted. On the right, the Solution Explorer shows a solution named "B2StoreApp" containing five projects: Entities, Presentation, Repositories, Services, and Dependencies.

```
//TODO:internal erişim belirleyicini public olarak değiştirin.
public class BookManager : IBookService
{
    //TODO: Manager ihtiyaç vardır.
    private readonly IRepositoryManger _manger;

    //TODO: ILoggerService ihtiyaç vardır.
    private readonly ILoggerService _logger; //Ctrl + . yaparak ekleyin.

    private readonly IMapper _mapper; //TODO: Mapper ihtiyaç vardır.

    public BookManager(IRepositoryManger repository, ILoggerService logger, IMapper mapper)
    {
        _manger = repository;
        _logger = logger;
        _mapper = mapper;
    }

    public void UpdateOneBook()
    {
        //check entity
        var entity = _manger.GetBookById(id);

        //check book null
        if (entity is null)
            throw new BookNotFoundException("Book not found");

        //TODO: Burada Map
        //TODO: Burada 10
        //entity.Title = b
        //entity.Price = b

        _manger.Book.Update(entity);
        _manger.Save();
    }
}
```

Gizlilik Derecesi [Tasnif Dışı](#)

# Onuncu Bölüm?

## Automapper

### Mapping Profile / AutoMapper Kullanımı

```
BookManager.cs  IBookService.cs  Services.Contracts.IBookService
Services
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace Services.Contracts
11 {
12     //TODO:internal erişim belirleyicini public olarak değiştirin.
13     public interface IBookService
14     {
15         IEnumerable<Book> GetAllBooks(bool trackChanges);
16         Book GetOneBookById(int id, bool trackChanges);
17         Book CreateOneBook(Book book);
18         void UpdateOneBook(int id, BookDtoUpdate bookDto, bool trackChanges);
19         void DeleteOneBook(int id, bool trackChanges);
20     }
21 }
22 
```

```
BookManager.cs  IBookService.cs  Services.BookManager
Services
63 public void UpdateOneBook(int id, BookDtoUpdate bookDto, bool trackChanges)
64 {
65     //check entity
66     var entity = _manger.Book.GetOneBookById(id, trackChanges);
67
68     //check book null
69     if (entity is null)
70         throw new BookNotFoundException(id);
71
72     //TODO: Burada Mapping işlemi var
73     //TODO: Burada 10 tane alan olsa tek tek esleyeceğiz. Bunun için Mapping işlemi yapacağız.
74     //entity.Title = book.Title;
75     //entity.Price = book.Price;
76
77     entity = _mapper.Map<Book>(bookDto); //TODO: Bize <Book> lazımlı nihai olarak bookDto kaynaklık edecek.
78
79     _manger.Book.UpdateBook(entity);
80     _manger.Save();
81 }
82
83
84 }
```

# Onuncu Bölüm?

Şu an uygulamamızda bir sorun var mı?  
Var ?  
Ne Peki?  
Book Manager in imzası değişti.  
IMapper Geldi  
Düzeltem !!!

## Automapper

### Mapping Profile / AutoMapper Kullanımı

```
1 using AutoMapper;
2 using Entities.DataTransferObjects;
3 using Entities.Exceptions;
4 using Entities.Models;
5 using Repositories.Contracts;
6 using Services.Contracts;
7 using System;
8 using System.Collections.Generic;
9 using System.Linq;
10 using System.Text;
11 using System.Threading.Tasks;
12
13 namespace Services
14 {
15     //TODO:internal erişim belirleyicini public olarak değiştirin.
16     public class BookManager : IBookService
17     {
18         //TODO: Manager için bir logger ekleyin.
19         private readonly ILogger<BookManager> _logger;
20
21         //TODO: ILogService
22         private readonly ILogService _logService;
23
24         private readonly IMapper _mapper;
25         public BookManager(IBookRepository bookService, ILogger<BookManager> logger, IMapper mapper)
```

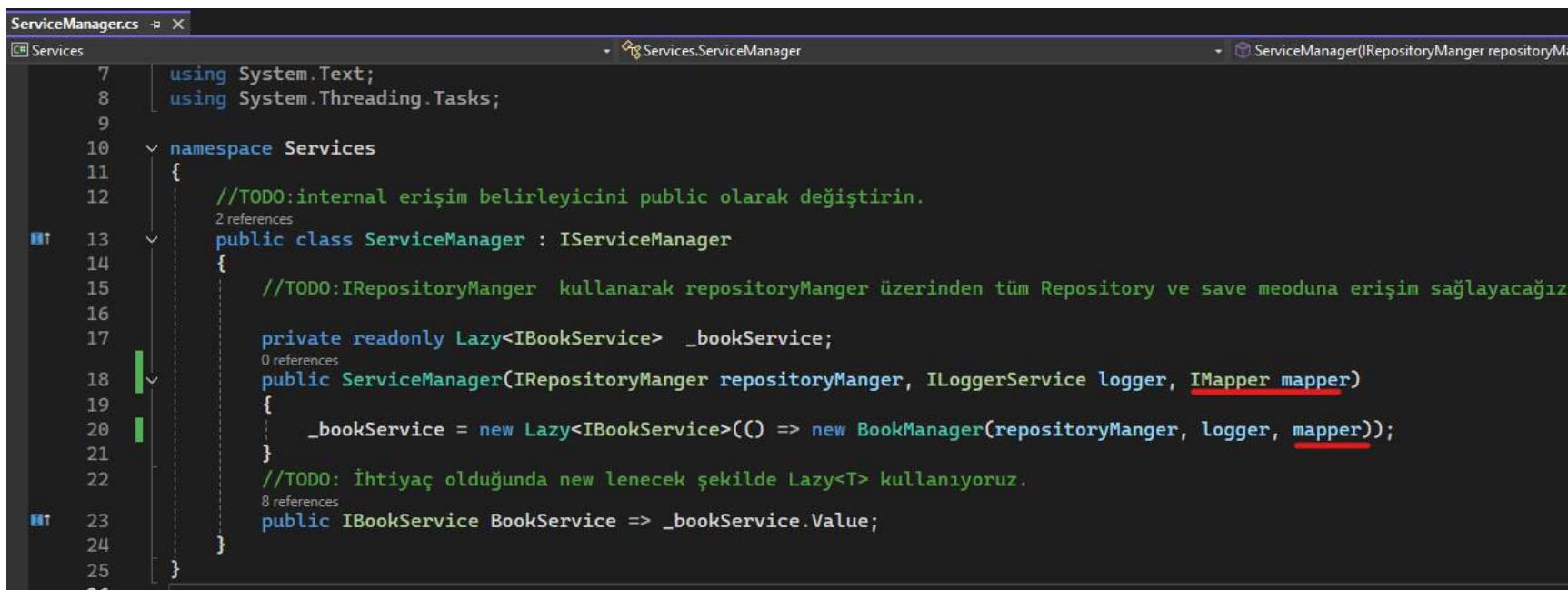
# Onuncu Bölüm?

Automapper

Mapping Profile / AutoMapper Kullanımı

Düzeltilmiş kısım !!!

ÇÖZÜLDÜ MÜ HAYIR. BİULD ET 2 Tane Hatayı gör.



```
ServiceManager.cs
using System.Text;
using System.Threading.Tasks;

namespace Services
{
    //TODO:internal erişim belirleyicini public olarak değiştirin.
    public class ServiceManager : IServiceProvider
    {
        //TODO:IRepositoryManger kullanarak repositoryManger üzerinden tüm Repository ve save meoduna erişim sağlayacağız.

        private readonly Lazy<IBookService> _bookService;
        public ServiceManager(IRepositoryManger repositoryManger, ILoggerService logger, IMapper mapper)
        {
            _bookService = new Lazy<IBookService>(() => new BookManager(repositoryManger, logger, mapper));
        }
        //TODO: İhtiyaç olduğunda new lenenek şekilde Lazy<T> kullanıyoruz.

        public IBookService BookService => _bookService.Value;
    }
}
```

# Onuncu Bölüm?

# Automapper

## Mapping Profile / AutoMapper Kullanımı

## Düzeltilmiş kısım !!!

The screenshot shows two code editors in Visual Studio. The top editor is for `BooksController.cs` and the bottom editor is for `BookDtoUpdate.cs`. Both files are part of the `Presentation` project under the `Presentation.Controllers` namespace.

**BooksController.cs:**

```
98  
99  
100 //TODO: Put metodu oluşturalım. (Güncelleme yapmak için)  
101 [HttpPut("{id:int}")]  
102 public IActionResult UpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] BookDtoUpdate bookDto)  
103 {  
104     //TODO: Güncelleme yapmak için gelen book var mı kontrol et.  
105     //ServiceManager Sonrası kapatılan kod !!!  
106  
107     if (bookDto == null)  
108     {  
109         return BadRequest(); // 400 Bad Request  
110     }  
111     _manager.BookService.UpdateOneBook(id, bookDto);  
112  
113     return Ok(); // 200 OK  
114 }
```

**BookDtoUpdate.cs:**

```
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160 //TODO: Put metodu oluşturalım.  
161 [HttpPatch("{id:int}")]  
162 public IActionResult PartiallyUpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] JsonPatchDocument<Book> bookPatch)  
163 {  
164     //TODO: Güncelleme yapmak için gelen book var mı kontrol et.  
165     var entity = _manager  
166         .BookService  
167         .GetOneBookById(id, true);  
168  
169     bookPatch.ApplyTo(entity);  
170     _manager.BookService.UpdateOneBook(id,  
171         new BookDtoUpdate { Id = entity.Id, Title = entity.Title, Price = entity.Price },  
172         true);  
173  
174     return NoContent(); // 204
```

# Onuncu Bölüm Bitti

## Hadi Özetleyelim?

Data Transfer  
Objects

- Auto Mapper
- Record type
- Paket Kurulumu
- Mapping Profile
- Servis Kaydı
- ServiceManager Düzenlenmesi
- Sunum Katmanının Düzenlenmesi

## Content Negotiation

İçerik Pazarlığı

# On birinci Bölüm?

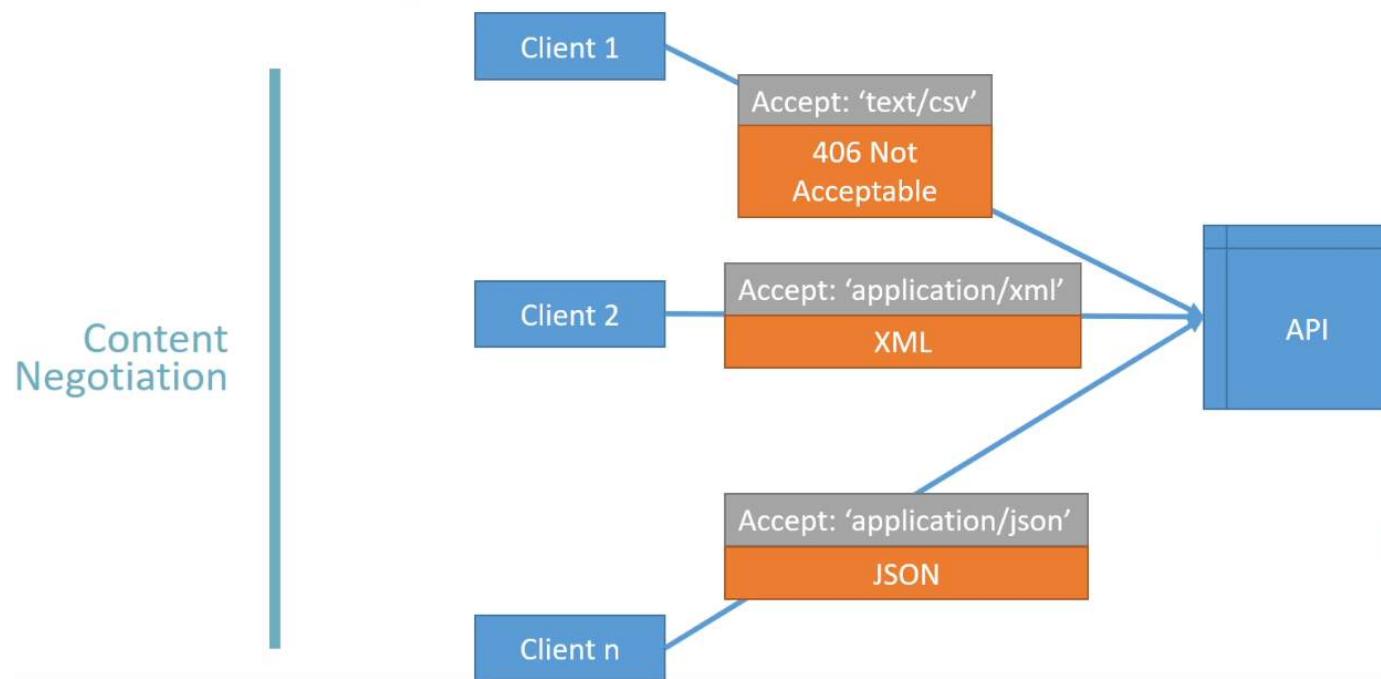
- Accept Header
- Respect Browser Acceptor Header
- AddXmlDataContractSerializerFormatters
- Serializable
- Restrict Media Type
- ReturnHttpNotAcceptable
- Custom Formatters
- IMvcBuilder Extension Method

# On birinci Bölüm?

İçerik Pazarlığı

Siz hangisi ile pazarlık yapacaksınız.  
Hangisine cevap vereceksiniz ???

Hadi Şimdi bizde API mize böyle bir özellik kazandıralım.



# On birinci Bölüm?

## İçerik Pazarlığı Accept Header

The screenshot shows the Postman application interface. A GET request is being made to `{baseUrl}_WebAPI}/api/books`. The Headers tab is selected, showing several Accept header entries. Some of these entries are highlighted with red boxes. The first highlighted entry is `*/*`, followed by `text/xml`, `application/json`, and `application/xml`. Below the headers, the response body is displayed as JSON, listing three books with their IDs, titles, and prices.

```
[{"id": 1, "title": "Kazagöz ve Hacivat", "price": 78.00}, {"id": 2, "title": "Mesnevi", "price": 178.00}, {"id": 3, "title": "Devlet", "price": 378.00}]
```

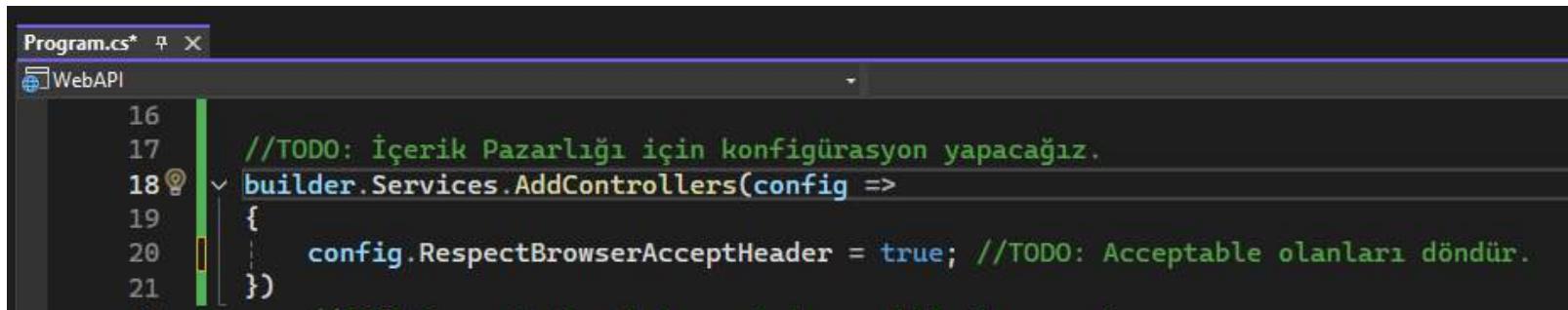
**Accept** olarak ne istersem isteyeyim yanıt vermiyor sonuç geliyor.

*Bu noktada «Respect Browser Acceptor Header» kullanacağız !!!*

# On birinci Bölüm?

İçerik Pazarlığı  
Accept Header

Program.cs de Acceptable ile  
ilgili ayarlama yaptık.



```
Program.cs* ⓘ X
WebAPI
16
17 //TODO: İçerik Pazarlığı için konfigürasyon yapacağız.
18 ⓘ builder.Services.AddControllers(config =>
19 {
20     config.RespectBrowserAcceptHeader = true; //TODO: Acceptable olanları döndür.
21 })
```

«False ise içerik pazarlığına  
kapalı Default değeri böyle !!!  
»

```
builder.Services.AddControllers(config =>
{
    config.RespectBrowserAcceptHeader = true; //TODO: Acceptable olanları döndür.
})
```

//TODO:Presentation ka bool Microsoft.AspNetCore.Mvc.MvcOptions.RespectBrowserAcceptHeader { get; set; } Gets or sets the flag which causes content negotiation to ignore Accept header when it contains the media type /\*. false by default.

UYGULAMAYI TEKRAR BAŞLAT.  
POSTMAN DAN BAK

# On birinci Bölüm?

İçerik Pazarlığı  
Accept Header

The screenshot shows the Postman application interface. A GET request is made to `(baseUrl_WebAPI) /api/books`. In the Headers section, the 'Accept' header is explicitly set to 'text/csv'. The response body displays a JSON array containing three book entries:

```
[{"id": 1, "title": "Kazagöz ve Hacivat", "price": 75.66}, {"id": 2, "title": "Mesnevi", "price": 170.00}, {"id": 3, "title": "Devlet", "price": 375.00}]
```

Gizlilik Derecesi [Tasnif Dışı](#)

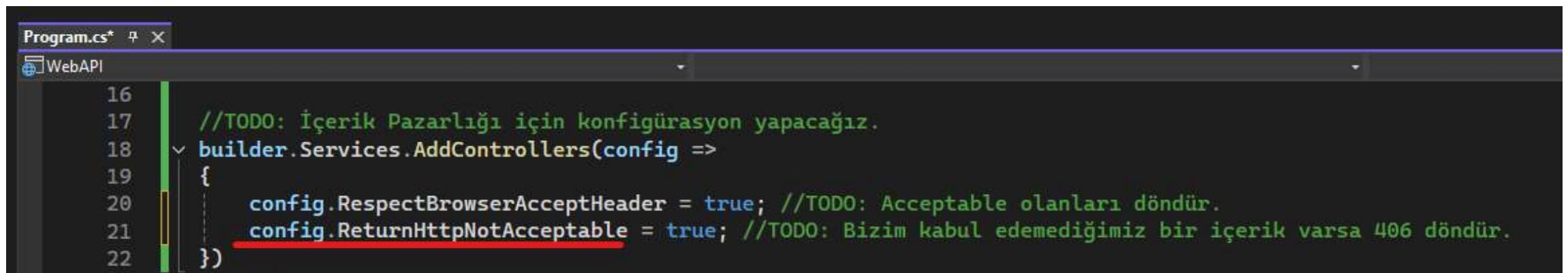
İçerik pazarlığı yaptık. Ancak bu pazarlık konusunda ama kabul edip etmeye konusunda bir şey yamadık.

Slayt Numarası .../..

# On birinci Bölüm?

İçerik Pazarlığı  
Accept Header

İçerik pazarlığı için ilgili konfigürasyonu yaptık !!!



A screenshot of the Visual Studio IDE showing the `Program.cs` file for a `WebAPI` project. The code is as follows:

```
16
17     //TODO: İçerik Pazarlığı için konfigürasyon yapacağız.
18     builder.Services.AddControllers(config =>
19     {
20         config.RespectBrowserAcceptHeader = true; //TODO: Acceptable olanları döndür.
21         config.ReturnHttpNotAcceptable = true; //TODO: Bizim kabul edemediğimiz bir içerik varsa 406 döndür.
22     })

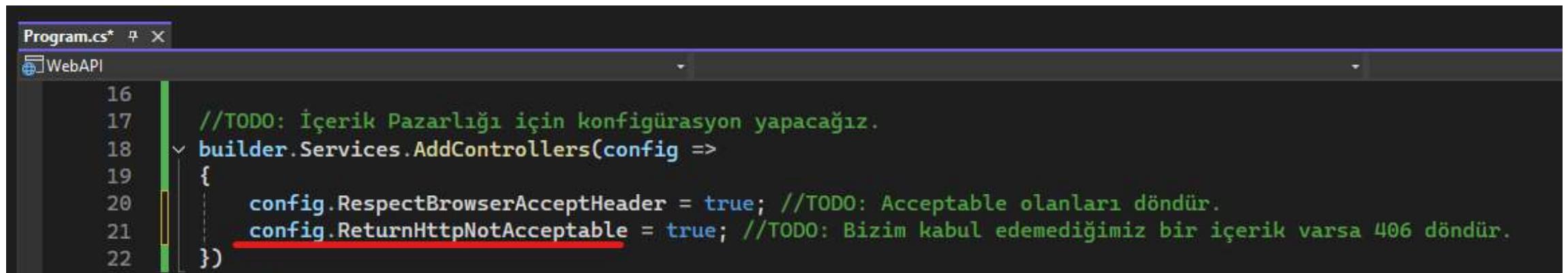
```

The code is annotated with green and red text. The first two lines are green, and the last two lines are red. The red annotations are part of a `config` block.

# On birinci Bölüm?

İçerik Pazarlığı  
Accept Header

İçerik pazarlığı için ilgili konfigürasyonu yaptık !!!



A screenshot of the Visual Studio IDE showing the `Program.cs` file for a `WebAPI` project. The code is as follows:

```
16
17     //TODO: İçerik Pazarlığını için konfigürasyon yapacağız.
18     builder.Services.AddControllers(config =>
19     {
20         config.RespectBrowserAcceptHeader = true; //TODO: Acceptable olanları döndür.
21         config.ReturnHttpNotAcceptable = true; //TODO: Bizim kabul edemediğimiz bir içerik varsa 406 döndür.
22     })

```

The line `config.ReturnHttpNotAcceptable = true;` is highlighted with a red underline.

UYGULAMAYI TEKRAR BAŞLAT.  
POSTMAN DAN BAK

# On birinci Bölüm?

İçerik Pazarlığı  
Accept Header

Baktık !!!

The screenshot shows a Postman collection named 'Books' with several requests. The main request is a GET to `((baseUrl_WebAPI))/api/books`. In the Headers tab, there are multiple 'Accept' fields. One 'Accept' field has a red underline and its value is set to `text/csv`. Other 'Accept' fields have values `text/xml`, `application/json`, and `application/xml`. The 'Accept-Encoding' header is also present with the value `gzip, deflate, br`. The 'Connection' header is set to `keep-alive`. The 'User-Agent' header is set to `PostmanRuntime/7.43.4`. The 'Host' header is calculated when the request is sent. The response status is 406 Not Acceptable.

# On birinci Bölüm?

İçerik Pazarlığı  
Accept Header

Baktık !!!

The screenshot shows the Postman application interface. On the left, there's a sidebar with a list of API endpoints for 'books'. The main area shows a 'GET' request to '[[baseUrl\_WebAPI]]/api/books'. The 'Headers' tab is selected, displaying the following configuration:

Key	Description
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.43.4
Accept	*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
Accept	text/csv
<b>Accept</b>	<b>text/xml</b>
Accept	application/json
Accept	application/xml

The row for 'Accept' with the value 'text/xml' is highlighted with a red underline. At the bottom, the status bar shows '406 Not Acceptable'.

# On birinci Bölüm?

İçerik Pazarlığı  
Accept Header

The screenshot shows the Postman interface for an API endpoint at `{baseUrl}_WebAPI/api/books`. The 'Headers' tab is selected, showing the following configuration:

Key	Description
Accept	application/json

The 'Body' tab displays a JSON response:

```
[{"id": 1, "title": "Kerşaf ve Macivat", "price": 76.00}, {"id": 2, "title": "Mesnevî", "price": 176.00}, {"id": 3, "title": "Devlet", "price": 976.00}]
```

Baktık !!! Güzel Veri geldi.  
Bunu destekliyoruz. !!!

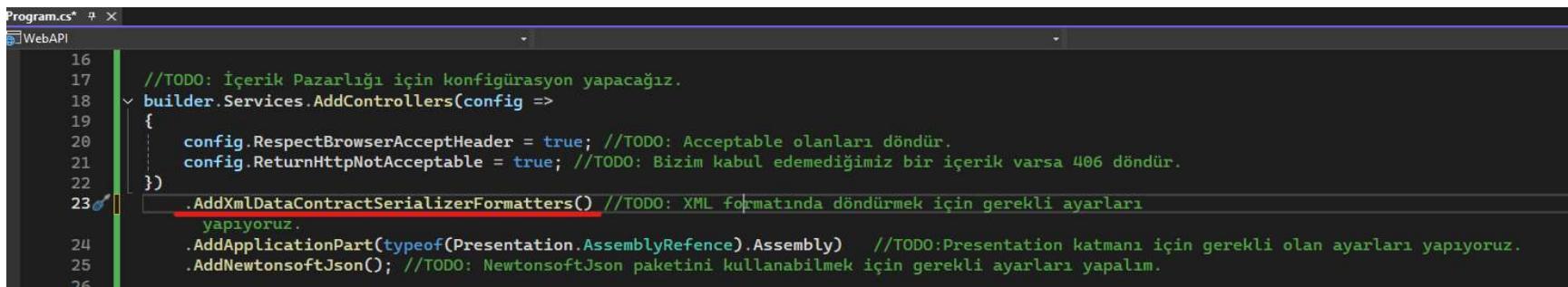
# On birinci Bölüm?

İçerik Pazarlığı

Add XML Data Contract Serializer Formatters da çıktı vermesini sağlayacağız

**Program.cs de XML için ilgili konfigürasyonu ekledik.**

*«Şimdi Uygulamayı çalıştır ve xml formatında veri正在写中文...»*



```
Program.cs* + X
WebAPI
16
17     //TODO: İçerik Pazarlığı için konfigürasyon yapacağız.
18     builder.Services.AddControllers(config =>
19     {
20         config.RespectBrowserAcceptHeader = true; //TODO: Acceptable olanları döndür.
21         config.ReturnHttpNotAcceptable = true; //TODO: Bizim kabul edemediğimiz bir içerik varsa 406 döndür.
22     })
23     .AddXmlDataContractSerializerFormatters() //TODO: XML formatında döndürmek için gerekli ayarları
24         yapıyoruz.
25     .AddApplicationPart(typeof(Presentation.AssemblyRefence).Assembly) //TODO: Presentation katmanı için gerekli olan ayarları yapıyoruz.
26     .AddNewtonsoftJson(); //TODO: NewtonsoftJson paketini kullanabilmek için gerekli ayarları yapalım.
```

# On birinci Bölüm?

İçerik Pazarlığı

Add XML Data Contract Serializert Formatters da çıktı vermesini sağlayacağız

«XML Veri Geldi»

The screenshot shows a Postman collection named 'Books'. A GET request is made to `{baseURL}_WebAPI/api/books`. The 'Headers' tab is selected, showing the following configuration:

- Postman-Token: (checkbox checked)
- Host: (checkbox checked)
- User-Agent: (checkbox checked)
- Accept: (checkbox checked, highlighted with a red border)
- Accept-Encoding: (checkbox checked)
- Connection: (checkbox checked)

The 'Body' tab shows the XML response:

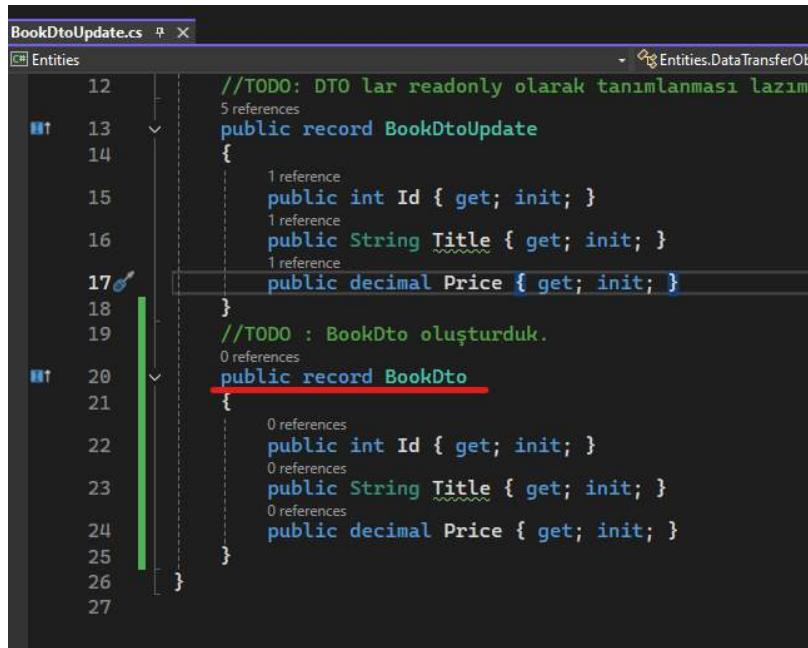
```
<ArrayOfBook xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/Entities.Models">
    <Book>
        <Id>1</Id>
        <Price>70.00</Price>
        <Title>Kısaçız ve Hacivat</Title>
    </Book>
    <Book>
        <Id>2</Id>
        <Price>170.00</Price>
        <Title>Eşenvisi</Title>
    </Book>
    <Book>
        <Id>3</Id>
        <Price>375.00</Price>
        <Title>Devlet</Title>
    </Book>
</ArrayOfBook>
```

# On birinci Bölüm?

İçerik Pazarlığı

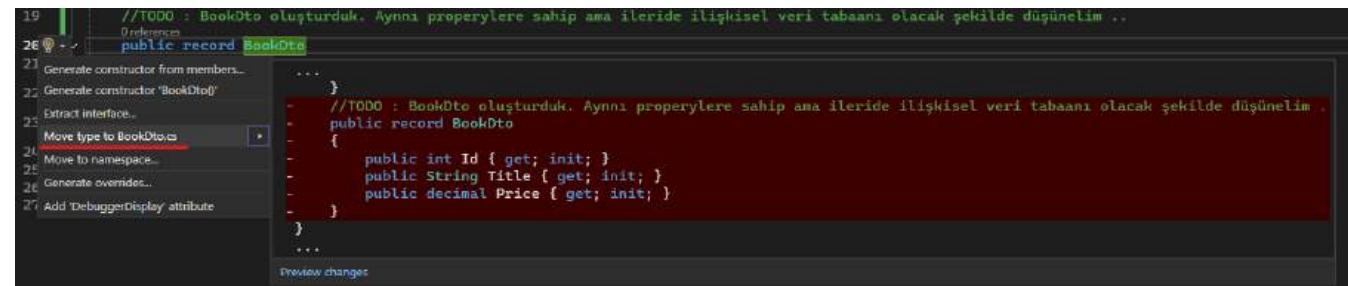
Serializable

«*BookDto oluştur*»



BookDtoUpdate.cs

```
12 //TODO: DTO lar readonly olarak tanımlanması lazım
13 public record BookDtoUpdate
14 {
15     1 reference
16     public int Id { get; init; }
17     1 reference
18     public String Title { get; init; }
19     1 reference
20     public decimal Price { get; init; }
21
22 //TODO : BookDto oluşturduk.
23 public record BookDto
24 {
25     0 references
26     public int Id { get; init; }
27 }
```

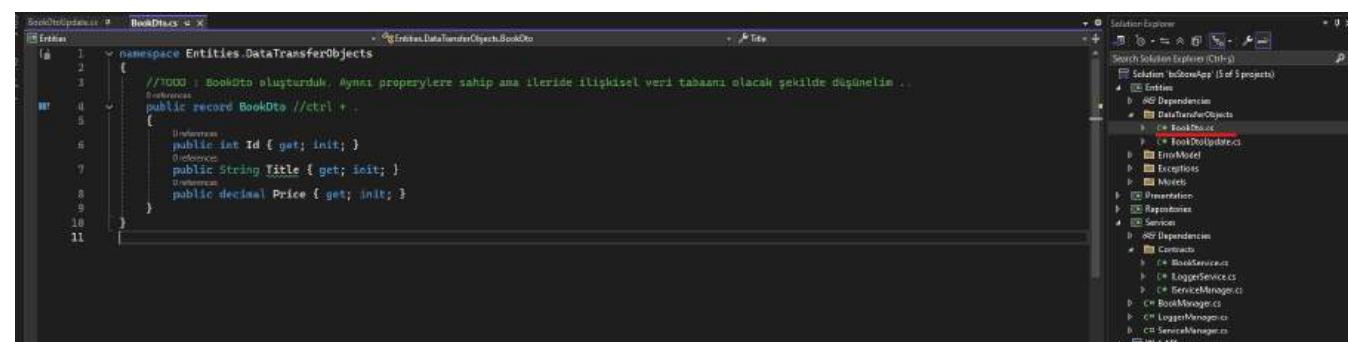


19 //TODO : BookDto oluşturduk. Aynı propertylere sahip ama ileride ilişkisel veri tabanı olacak şekilde düşünelim ..
20 public record BookDto
21
22 Generate constructor from members...
23 Generate constructor 'BookDto'
24 Extract interface...
25 Move type to BookDto.cs
26 Move to namespace...
27 Generate overrides...
28 Add 'DebuggerDisplay' attribute

Preview changes

«*Ctrl + .*»

«*İlgili class oluştur...*»



BookDtoUpdate.cs

```
1 //namespace Entities.DataTransferObjects
2 public record BookDto
3 {
4     //TODO : BookDto oluşturduk. Aynı propertylere sahip ama ileride ilişkisel veri tabanı olacak şekilde düşünelim ..
5     public int Id { get; init; }
6     public String Title { get; init; }
7     public decimal Price { get; init; }
8 }
```

Solution Explorer

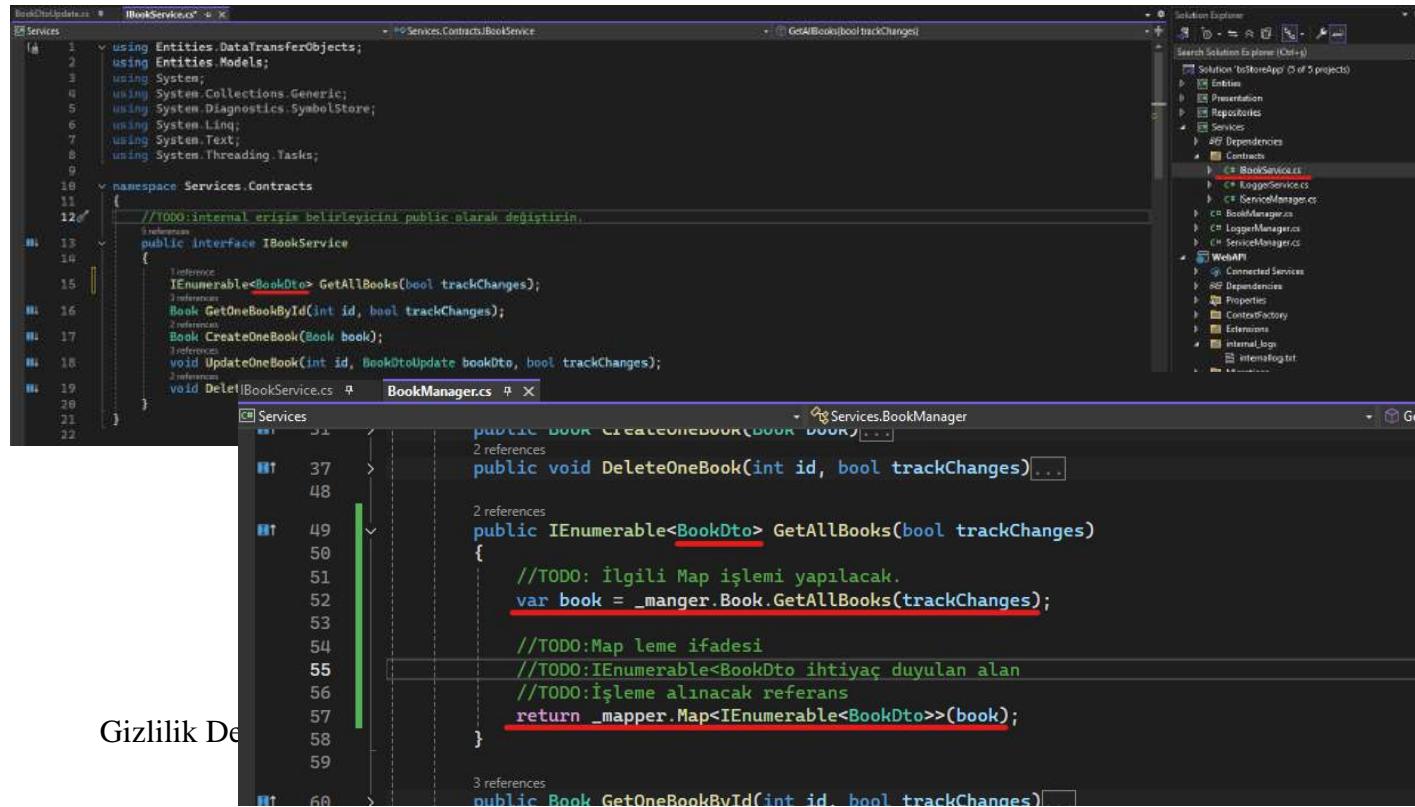
- Search Solution Explorer (Ctrl+Shift+F)
- Search Solution Explorer (Ctrl+Shift+F)
- Entities
- Dependencies
- DataTransferObjects
- BookDtoUpdate.cs
- BookDto.cs
- BookManager.cs
- ErrorModel.cs
- Exceptions.cs
- Model.cs
- Presentation
- Repositories
- Services
- ISF Dependencies
- Contracts
- BookService.cs
- LoggerService.cs
- ServiceManager.cs
- BookManager.cs
- LoggerManager.cs
- SerialManager.cs
- WebAPI

# On birinci Bölüm?

İçerik Pazarlığı

Serializable

«*IBookService* Book Dto dönüştür»



```
IBookService.cs
1  using Entities.DataTransferObjects;
2  using Entities.Models;
3  using System;
4  using System.Collections.Generic;
5  using System.Diagnostics.SymbolStore;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace Services.Contracts
11 {
12     //TODO: internal erişim belirleyicini public olarak değiştirelim.
13
14     public interface IBookService
15     {
16         IEnumerable<BookDto> GetAllBooks(bool trackChanges);
17
18         Book GetOneBookById(int id, bool trackChanges);
19
20         Book CreateOneBook(Book book);
21
22         void UpdateOneBook(int id, BookDtoUpdate bookDto, bool trackChanges);
23
24         void DeleteOneBook(int id, bool trackChanges);
25     }
26 }
```

```
BookManager.cs
1  using Entities.DataTransferObjects;
2  using Entities.Models;
3  using System;
4  using System.Collections.Generic;
5  using System.Diagnostics.SymbolStore;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace Services
11 {
12     public class BookManager : IBookService
13     {
14         private readonly Book _manger;
15
16         public BookManager()
17         {
18             _manger = new Book();
19         }
20
21         public void CreateOneBook(Book book)
22         {
23             _manger.Add(book);
24         }
25
26         public void DeleteOneBook(int id, bool trackChanges)
27         {
28             var book = _manger.GetOneBookById(id);
29
30             if (book != null)
31             {
32                 _manger.Delete(book);
33             }
34         }
35
36         public IEnumerable<BookDto> GetAllBooks(bool trackChanges)
37         {
38             //TODO: İlgili Map işlemi yapılacak.
39             var book = _manger.Book.GetAllBooks(trackChanges);
40
41             //TODO: Map leme ifadesi
42             //TODO: IEnumerable<BookDto> ihtiyaç duyulan alan
43             //TODO: İşleme alınacak referans
44             return _mapper.Map<IEnumerable<BookDto>>(book);
45         }
46
47         public Book GetOneBookById(int id, bool trackChanges)
48         {
49             var book = _manger.GetOneBookById(id);
50
51             if (book != null)
52             {
53                 book.TrackChanges(trackChanges);
54             }
55
56             return book;
57         }
58
59     }
60 }
```

Gizlilik De

«*Book Manager* da İlgili Map işlemi yap»  
ÖNEMLİ: book dan BookDto ya  
geçiş var mı ?  
Yok?  
Hadi onu yapalım !!!

# On birinci Bölüm?

İçerik Pazarlığı

Serializable

Hadi onu yapalım !!!!

Yaptık !!!!

Şimdi Postman da (*api/books*) Accept application/xml olacak şekilde istek atın !!! Hata Gelecek

The screenshot shows the Visual Studio IDE interface. On the left is the code editor with the MappingProfile.cs file open. The code defines a class MappingProfile that inherits from AutoMapper.Profile. It contains two CreateMap<Book, Book>() methods, one for BookDtoUpdate and one for Book. The Solution Explorer on the right shows a solution named 'bsStoreApp' containing five projects: Entities, Presentation, Repositories, Services, and WebAPI. The WebAPI project is expanded, showing its internal structure including Connected Services, Dependencies, Properties, ContextFactory, Extensions, internalJobs, Migrations, Utilities, and AutoMapper. The MappingProfile.cs file is selected in the Utilities/AutoMapper folder.

```
MappingProfile.cs
1  using AutoMapper;
2  using Entities.DataTransferObjects;
3  using Entities.Models;
4
5  namespace WebAPI.Utilities.AutoMapper
6  {
7      //TODO : MappingProfile class Profile dan kalıtım alacak
8      //reference
9      public class MappingProfile : Profile
10     {
11         //References
12         public MappingProfile()
13         {
14             //TODO : Mapping işlemleri burada yapılacak
15             //TODO : CreateMap<Source, Destination>();
16
17             CreateMap<BookDtoUpdate, Book>();
18             CreateMap<Book, BookDto>();
19         }
20     }
21 }
22
```

# On birinci Bölüm?

*Hata ve yapılacak işlem*

İçerik Pazarlığı  
Serializable

The screenshot shows a developer's workspace with several windows open:

- Solution Explorer:** Shows the project structure for "testapp" with various layers like Entities, DataTransferObjects, and WebAPI.
- Code Editor:** Displays a C# file named "BookDto.cs" containing the following code:

```
namespace Entities.DataTransferObjects
{
    //TODO : BookDto oluşturduk. Aynı propertylere sahip ama ileride ilişkisel veri tabanını olacak şekilde düşünelim ..
    [Serializable]
    public record BookDto //ctrl + .
    {
        [References]
        public int Id { get; init; }
        [References]
        public String Title { get; init; }
        [References]
        public decimal Price { get; init; }
    }
}
```

- Postman:** Shows a GET request to "http://localhost:5001/api/books". The Headers tab is selected, showing "Content-Type: application/json". The Body tab contains a JSON response:

```
{"Status": "Error", "Message": "Type 'Entities.DataTransferObjects.BookDto' cannot be serialized. Consider marking it with the DataContractAttribute attribute, and marking all of its members you want serialized with the DataMemberAttribute attribute. Alternatively, you can ensure that the type is public and has a parameterless constructor - all public members of the type will then be serialized, and no attributes will be required."}
```

«KODLAMAYIN BENİ DİNLEYİN !!!»

```
CsvOutputFormatter.cs
1  using Entities.DataTransferObjects;
2  using Entities.Models;
3  using Microsoft.AspNetCore.Mvc.Formatters;
4  using Microsoft.Net.Http.Headers;
5  using System.Text;
6
7  namespace WebAPI.Utilities.Formatters
8  {
9      public class CsvOutputFormatter : TextOutputFormatter
10     {
11         public CsvOutputFormatter()
12         {
13             //TODO: MediaTypeHeaderValue ifadeyi Microsoft.Net.Http.Headers
14             //TODO: Csv ifadesini oluşturmul oluk
15             SupportedMediaTypes.Add(MediaTypeHeaderValue.Parse("text/csv"));
16             SupportedEncodings.Add(Encoding.UTF8);
17             SupportedEncodings.Add(Encoding.Unicode);
18         }
19
20         protected override bool CanWriteType(Type? type)
21         {
22             if (typeof(BookDto).IsAssignableFrom(type) ||
23                 typeof(IEnumerable<BookDto>).IsAssignableFrom(type))
24             {
25                 return base.CanWriteType(type);
26             }
27             return false;
28         }
29
30         private static void FormatCsv(StringBuilder buffer, BookDto book)
31         {
32             buffer.AppendLine($"{book.Id},{book.Title},{book.Price}");
33         }
34     }
35 }
36
37 public override async Task WriteResponseBodyAsync(OutputFormatterWriteContext context, Encoding selectedEncoding)
38 {
39     var response = context.HttpContext.Response;
40     var buffer = new StringBuilder();
41     if (context.Object is IEnumerable<BookDto> bookDto)
42     {
43         foreach (var book in (IEnumerable<BookDto>)context.Object)
44         {
45             FormatCsv(buffer, book);
46         }
47     }
48     else if (context.Object is IEnumerable<BookDto> books)
49     {
50         FormatCsv(buffer, (BookDto)context.Object);
51     }
52     await response.WriteAsync(buffer.ToString(), selectedEncoding);
53 }
```

# On birinci Bölüm?

İçerik Pazarlığı

Csv Output Formatter

«KODLAMAYIN BENİ DİNLEYİN  
!!!»

The screenshot shows a Visual Studio interface with several windows:

- Solution Explorer:** Shows a solution named 'bsStoreApp' containing five projects: Entities, Presentation, Repositories, Services, and WebAPI. The WebAPI project is expanded, showing its structure including 'Extensions' which contains 'IMvcBuilderExtensions.cs'.
- IMvcBuilderExtensions.cs:** A code editor window showing the implementation of the `IMvcBuilderExtensions` interface. It includes a static method `AddCustomCsvFormatter` that adds a `CsvOutputFormatter` to the `MvcOptions`.
- Program.cs:** Another code editor window showing the configuration of the `builder` object. It includes a call to `.AddCustomCsvFormatter()` and other configuration methods like `.AddNewtonsoftJson()` and `.AddEndpointsApiExplorer()`.

# On birinci Bölüm?

İçerik Pazarlığı

Csv Output Formatter

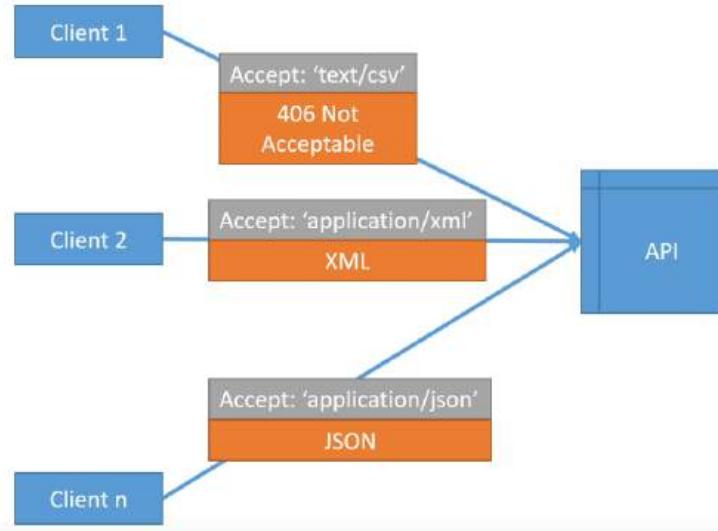
«**KODLAMAYIN BENI DINLEYİN  
!!!»**

The screenshot shows a Postman collection named 'Books' with various API endpoints. The main view displays a GET request to `(baseUrl)_WebAPI/api/books`. In the Headers tab, the `Accept` header is set to `text/csv`. The response is a 200 OK status with a CSV payload containing five book entries:

Sıra	Kitap Adı	Fiyat
1	Kazığır ve Hacivat	75,00
2	Mesnevi	175,00
3	Devlet	375,00
4	1991,deneme	1999,00
5		

# On birinci Bölüm Bitti

## Hadi Özetleyelim?



### Content Negotiation

- Accept Header
- Respect Browser Acceptor Header
- AddXmlDataContractSerializerFormatters
- Serializable
- Restrict Media Type
- ReturnHttpNotAcceptable
- Custom Formatters
- IMvcBuilder Extension Method

## Validation

### Doğrulama (Validation)

# On ikinci Bölüm?

- [ApiController]
- Configure<ApiBehaviorOptions>
  - SuppressModelStateInvalidFilter
  - SuppressInferBindingSourcesForParameters
  - SuppressConsumesConstraintForFormFileParameters
  - SuppressMapClientErrors
- ModelState
- 422 Unprocessable Entity
- Rerun Validation
- Built-in Attributes
- Custom Attributes
- IValidationObject

# On ikinci Bölüm?

Doğrulama (Validation)  
Api Behaviour Options

ControllerBase

Validation

Method	Notes
BadRequest	Returns 400 status code.
NotFound	Returns 404 status code.
PhysicalFile	Returns a file.
TryUpdateModelAsync	Invokes <a href="#">model binding</a> .
TryValidateModel	Invokes <a href="#">model validation</a> .

- Microsoft.AspNetCore.Mvc namespace bazı Attribute ifadeleri sağlar.

```
C#  
Copy  
[HttpPost]  
[ProducesResponseType(StatusCodes.Status201Created)]  
[ProducesResponseType(StatusCodes.Status400BadRequest)]  
public ActionResult<Pet> Create(Pet pet)  
{  
    pet.Id = _petsInMemoryStore.Any() ?  
        _petsInMemoryStore.Max(p => p.Id) + 1 : 1;  
    _petsInMemoryStore.Add(pet);  
  
    return CreatedAtAction(nameof(GetById), new { id = pet.Id }, pet);  
}
```

The screenshot shows the Visual Studio IDE interface. The code editor on the left displays the `BookDtoForManipulation.cs` file, which contains C# code for a record type named `BookDtoUpdate`. The Solution Explorer on the right shows the project structure for `InStoreApp`, which includes several sub-projects like `Entities`, `Data Transfer Objects`, `Models`, `Presentation`, `Repositories`, and `Services`.

```
BookDtoForManipulation.cs
BookDtoUpdate.cs
BookDtoUpdate.cs

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Entities.DataTransferObjects
{
    //TODO: internal erişim belirleyicini public olarak değiştirin.
    //TODO: record yapısı, C# 9.0 ile gelen bir referans tipi (class gibi) ama immutable yani değiştirilemez veri yapısıdır.
    //TODO: 1-Veri taşıma anacıyla (DTO - Data Transfer Object) 2-ğer (value-based) Wargılaştırmalar için 3-Okunabilirliği yükselt, sade yapalar için
    //TODO: DTO lar readonly olarak tanımlanması: lazım bunun için { get; init; } şeklinde tanımlanabilir.
    public record BookDtoUpdate : BookDtoForManipulation //TODO: Artık Validation için yazdığımız BookDtoForManipulation clasından kalıtım alacak
    {
        [Required]
        public int Id { get; init; }
    }
}
```

# On ikinci Bölüm?

Doğrulama (Validation)  
Record Types

«*BookDtoForInsertion classını oluştur  
ilgili kalımı al!!!»*

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays the file `BookDtoForInsertion.cs` with the following content:

```
1  namespace Entities.DataTransferObjects
2  {
3      public record BookDtoForInsertion : BookDtoForManipulaton
4  }
```

On the right, the Solution Explorer window shows the project structure for 'bsStoreApp' (5 of 5 projects). The `DataTransferObjects` folder contains the following files:

- `BookDto.cs`
- `BookDtoForInsertion.cs` (highlighted in red)
- `BookDtoForManipulaton.cs`
- `BookDtoUpdate.cs`

# On ikinci Bölüm?

## Doğrulama (Validation)

### Servis Katmanın Düzenlenmesi

«*IBookService Dto ya göre güncelle  
Sonra BookManeger da ilgili metodları  
güncelle !!!»*

The screenshot shows the Visual Studio IDE interface. On the left is the code editor for `IBookService.cs`, which contains the following C# code:

```
4  using System.Collections.Generic;
5  using System.Diagnostics.SymbolStore;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace Services.Contracts
11 {
12     //TODO:internal erişim belirleyicini public olarak değiştirin.
13     public interface IBookService
14     {
15         IEnumerable<BookDto> GetAllBooks(bool trackChanges);
16         BookDto GetOneBookById(int id, bool trackChanges);
17         BookDto CreateOneBook(BookDtoForInsertion book);
18         void UpdateOneBook(int id, BookDtoUpdate bookDto, bool trackChanges);
19         void DeleteOneBook(int id, bool trackChanges);
20     }
21 }
```

The method `CreateOneBook` is highlighted with a red underline. On the right side, the Solution Explorer shows the project structure with the `IBookService.cs` file selected.

# On ikinci Bölüm?

Doğrulama (Validation)

Servis Katmanın Düzenlenmesi

«*GetOneBookById metodun  
güçellenmiş hali !!!»*

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays `BookManager.cs` with the following code:

```
55     //TODO: IEnumerable<BookDto> ihtiyaç duyulan alan
56     //TODO: işleme alınacak referans
57     return _mapper.Map<IEnumerable<BookDto>>(book);
58
59
60     3 references
61     public BookDto GetOneBookById(int id, bool trackChanges)
62     {
63         var book = _manger.Book.GetOneBookById(id, trackChanges);
64         if (book is null)
65             throw new BookNotFoundException(id);
66
67         return _mapper.Map<BookDto>(book);
68     }
```

The `GetOneBookById` method is highlighted with red underlines, indicating it needs attention. On the right, the Solution Explorer window shows the project structure for `bsStoreApp`, which includes five projects: `External Sources`, `Entities`, `Presentation`, `Repositories`, and `Services`. The `Services` project contains `IBookService.cs`, `ILoggerService.cs`, `IServiceManager.cs`, `BookManager.cs`, `LoggerManager.cs`, and `ServiceManager.cs`.

# On ikinci Bölüm?

Doğrulama (Validation)

Servis Katmanın Düzenlenmesi

«*CreateOneBook metodun  
güçellenmiş hali !!!»*

The screenshot shows a dual-code editor interface with two tabs: 'MappingProfile.cs' and 'BookManager.cs'.

**MappingProfile.cs:**

```
1  using AutoMapper;
2  using Entities.DataTransferObjects;
3  using Entities.Models;
4
5  namespace WebAPI.Utilities.AutoMapper
6  {
7      //TODO : MappingProfile class Profile dan kalıtım alacak
8      public class MappingProfile : Profile
9      {
10         public MappingProfile()
11         {
12             //TODO : Mapping işlemleri burada yapılacak
13             //TODO:CreateMap<Source, Destination>();
14
15             CreateMap<BookDtoUpdate, Book>();
16
17             CreateMap<Book, BookDto>();
18
19             CreateMap<BookDtoForInsertion, Book>();
20         }
21     }
22 }
```

**BookManager.cs:**

```
1  public BookDto CreateOneBook(BookDtoForInsertion bookDto)
2  {
3      var entity = _mapper.Map<Book>(bookDto);
4
5      _manger.Book.CreateBook(entity);
6      _manger.Save();
7
8      return _mapper.Map<BookDto>(entity);
9 }
```

# On ikinci Bölüm?

## Doğrulama (Validation)

### Sunum Katmanın Düzenlenmesi

```
//TODO: Post metodu oluşturulacak.
[HttpPost]
public IActionResult CreateOneBook([FromBody] BookDtoForInsertion bookDto)
{
    if (bookDto == null)
    {
        return BadRequest(); // 400
    }
    ServiceManager Sonrası kapatılan kod !!!
    var bookCreate = _manager
        .BookService
        .CreateOneBook(bookDto);
    return StatusCode(201, bookCreate); // 201
}
```

«Build Yapalım  
BookController daki hataları giderelim  
!!!»

Sonra record typleri postman da test edelim

```
//TODO: Put metodu oluşturulacak.
[HttpPatch("{id:int}")]
public IActionResult PartiallyUpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] JsonPatchDocument<BookDto> bookPatch)
{
    //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
    var bookDto = _manager
        .BookService
        .GetOneBookById(id, true);

    bookPatch.ApplyTo(bookDto);

    _manager.BookService.UpdateOneBook(id,
        new BookDtoUpdate { Id = bookDto.Id, Title = bookDto.Title, Price = bookDto.Price },
        true);

    return NoContent(); // 204
}
```

# On ikinci Bölüm?

## Doğrulama (Validation)

### Sunum Katmanın Düzenlenmesi

«Title max 2 karekter yazmışız hatayı aldık !!!»

Bu alam 1000 yapalım ve uygulamayı tekrar çalıştırıp test edelim

Burada gelen hata 400 hatası ama biz istemci hatası alıyoruz bunun program.cs de güncelleme yapıyoruz

The screenshot shows a Postman interface with the following details:

- Request Method:** POST
- URL:** {{baseUrl\_WebAPI}}/api/books
- Body:** Raw (JSON)
- JSON Body:**

```
1: {
2:   // "id": "{$randomInt}",
3:   "title": "{$randomWord2}",
4:   "price": "{$randomPrice}"
5: }
```
- Response Body:** JSON (Raw)

```
[{"errors": [{"title": ["Title en fazla 2 karakter olmaz."]}], "type": "https://tools.ietf.org/html/rfc7233#section-6.5.1", "title": "One or more validation errors occurred.", "status": 400, "traceId": "00-tfb0ceda80fa89d1be600f3d96f83ac-57e8c17ae77f904b-00"}]
```

# On ikinci Bölüm?

## Doğrulama (Validation)

### Sunum Katmanın Düzenlenmesi

The screenshot shows a .NET Core Web API project in Visual Studio. The `Program.cs` file contains configuration code, including the following validation-related code:

```
builder.Services.Configure<ApiBehaviorOptions>(options =>
{
    options.SuppressModelStateInvalidFilter = true;
});
```

The Solution Explorer shows the project structure with `Contracts`, `IBookService.cs`, and `ILoggerService.cs` files.

Postman is open, showing a successful `POST` request to `(baseUrl/WebAPI)/api/books`. The request body is:

```
{
  "title": "B",
  "price": 1991}
```

The response status is `201 Created`.

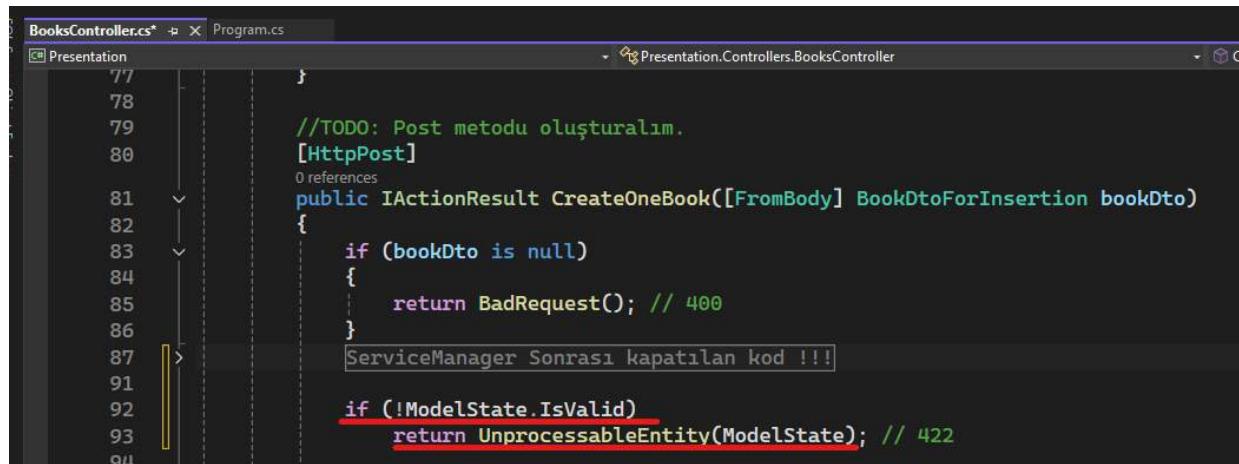
*«Postman uygulama çalıştırı hata vermesi gerekiyordu NEDEN VERMEDİ!!!»*

# On ikinci Bölüm?

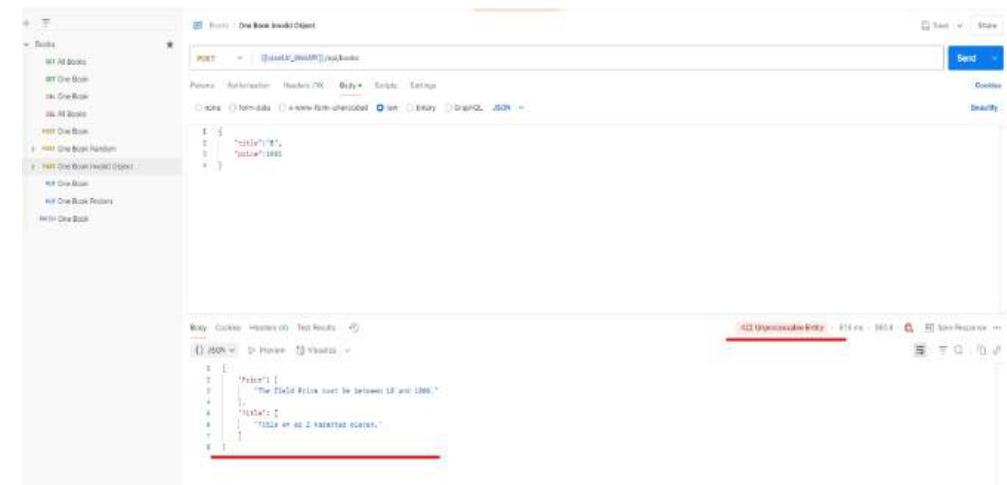
## Doğrulama (Validation)

### Sunum Katmanın Düzenlenmesi

«CreateOneBook metodunda ilgili değişikliği yap!!!»  
**UYGULAMAYI TEKRAR ÇALIŞTIR !!!**



```
BooksController.cs*  X Program.cs
Presentation  ↗ Presentation.Controllers.BooksController  ↗ CreateOneBook
77
78
79    //TODO: Post metodu oluşturalım.
80    [HttpPost]
81    public IActionResult CreateOneBook([FromBody] BookDtoForInsertion bookDto)
82    {
83        if (bookDto is null)
84        {
85            return BadRequest(); // 400
86        }
87        ServiceManager Sonrası kapatılan kod !!!
88
89        if (!ModelState.IsValid)
90            return UnprocessableEntity(ModelState); // 422
91
92
93
94 }
```



# On ikinci Bölüm?

## Doğrulama (Validation)

### Sunum Katmanın Düzenlenmesi

«PUT metodunu test et!!!»

HATA ALDIK !!!

Hatayı düzeltik çalıştır

The screenshot shows a POSTMAN interface. The URL is `(baseurl)_WebAPI/api/books/2`. The Body tab contains the following JSON:

```
1 {  
2   "id":2,  
3   "title":{{#randomWord}},  
4   "price":{{#randomPrice}}  
5 }
```

The response status is `500 Internal Server Error`. The response body is:

```
{  
  "statusCode": 500,  
  "message": "The instance of entity type 'Book' cannot be tracked because another instance with the same key value for ['Id'] is already being tracked. When attaching existing entities, ensure that only one entity instance with a given key value is attached. Consider using 'DbContextOptionsBuilder.EnableSensitiveDataLogging' to see the conflicting key values."  
}
```

```
[HttpPut("{id:int}")]
public IActionResult UpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] BookDtoToUpdate bookDto)
{
    //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
    ServiceManager Sonrası kapatılan kod !!!
    if (bookDto == null)
    {
        return BadRequest(); // 400
    }
    _manager.BookService.UpdateOneBook(id, bookDto, false);
    return NoContent(); // 204
}
```

# On ikinci Bölüm?

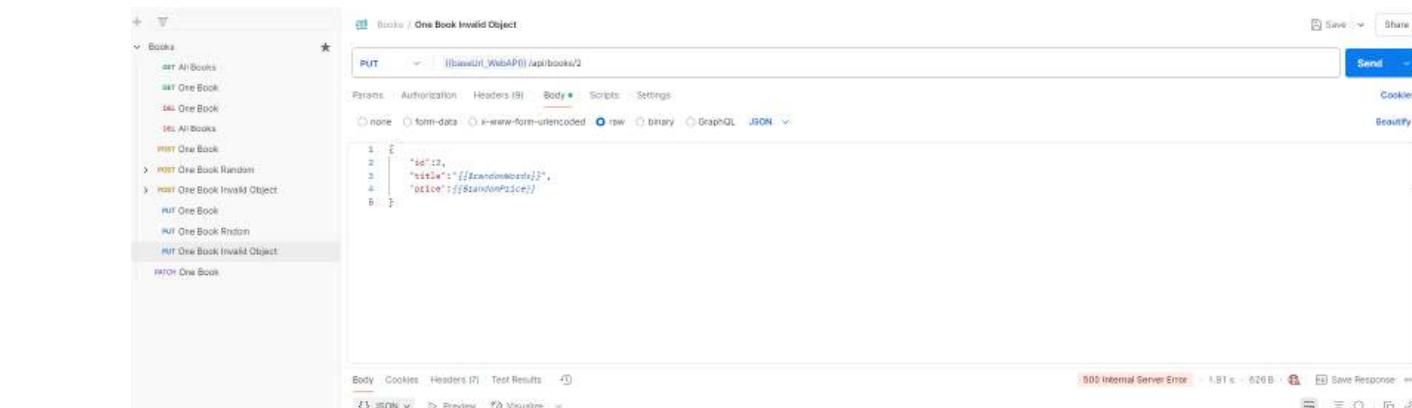
## Doğrulama (Validation)

### Sunum Katmanın Düzenlenmesi

«PUT metodunu test et!!!»

HATA ALDIK !!!

Hatayı düzelttik, Ayrıca Model State ekledik  
çalıştır



```
BooksController.cs*  ↗ Presentation.Controllers.BooksController  ↗ UpdateOneBook(int id, BookDtoUpdate bookDto)
103 [HttpPut("{id:int}")]
104 0 references
105 public IActionResult UpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] BookDtoUpdate bookDto)
106 {
107     //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
108     ServiceManager Sonrası kapatılan kod !!!
109
110     if (bookDto == null)
111     {
112         return BadRequest(); // 400
113     }
114     if (!ModelState.IsValid)
115     {
116         return UnprocessableEntity(ModelState); // 422
117     }
118     _manager.BookService.UpdateOneBook(id, bookDto, false);
119
120     return NoContent(); // 204
121 }
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136 }
```

```
BooksController.cs*  ↗ Presentation.Controllers.BooksController  ↗ UpdateOneBook(int id, BookDtoUpdate bookDto)
103 [HttpPut("{id:int}")]
104 0 references
105 public IActionResult UpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] BookDtoUpdate bookDto)
106 {
107     //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
108     ServiceManager Sonrası kapatılan kod !!!
109
110     if (bookDto == null)
111     {
112         return BadRequest(); // 400
113     }
114     _manager.BookService.UpdateOneBook(id, bookDto, false);
115
116     return NoContent(); // 204
117 }
```

# On ikinci Bölüm?

## Doğrulama (Validation)

### Sunum Katmanın Düzenlenmesi

«PUT metodunu test et!!!»

The screenshot shows two Postman environments and their requests:

- Environment 1 (Left):** Contains a PUT request to `/books/{id}` with the following body:

```
[{"id": "1", "title": "The Great Gatsby", "pageCount": 123}, {"id": "2", "title": "War and Peace", "pageCount": 1234}, {"id": "3", "title": "Pride and Prejudice", "pageCount": 12345}, {"id": "4", "title": "Moby-Dick", "pageCount": 123456}]
```

The response status is 422 Unprocessable Entity, and the JSON error message is:

```
{"errors": [{"id": "3", "message": "The pageCount must be between 33 and 1000."}], "titles": [{"id": "3", "message": "Title is too long. Max length is 100."}]}  
A red horizontal bar highlights the error message.
```
- Environment 2 (Right):** Contains a PUT request to `/books/{id}` with the following body:

```
[{"id": "1", "title": "The Great Gatsby", "pageCount": 123}, {"id": "2", "title": "War and Peace", "pageCount": 1234}, {"id": "3", "title": "Pride and Prejudice", "pageCount": 12345}, {"id": "4", "title": "Moby-Dick", "pageCount": 123456}]
```

The response status is 204 No Content.

# On ikinci Bölüm?

Doğrulama (Validation)

Sunum Katmanın Düzenlenmesi

«*PartiallyUpdateOneBook* metodunu düzenley !!!»

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays the `BooksController.cs` file. The `PartiallyUpdateOneBook` method is highlighted, showing the following code:

```
public IActionResult PartiallyUpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] JsonPatchDocument<BookDto> bookPatch)
{
    //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
    var bookDto = _manager
        .BookService
        .GetOneBookById(id, true);

    bookPatch.ApplyTo(bookDto, ModelState);

    _manager.BookService.UpdateOneBook(id,
        new BookDtoUpdate { Id = bookDto.Id, Title = bookDto.Title, Price = bookDto.Price },
        true);

    return NoContent(); // 204
}
```

The `bookPatch.ApplyTo(bookDto, ModelState);` line is underlined with a red squiggle, indicating a potential issue. On the right, the Solution Explorer window shows the project structure for `bsStoreApp`, which includes five projects: Entities, Presentation, Dependencies, Packages, Projects, Controllers, AssemblyReference.cs, Repositories, Services, and WebAPI.

The screenshot shows the Visual Studio code editor with the MappingProfile.cs file open. The code defines a MappingProfile class that inherits from Profile. It contains several CreateMap methods for mapping BookDtoUpdate to Book, Book to BookDto, and BookDtoForInsertion to Book. A specific CreateMap line for BookDtoUpdate to Book is highlighted with a red rectangle.

```
4
5     namespace WebAPI.Utilities.AutoMapper
6     {
7         //TODO : MappingProfile class Profile dan kalıtım alacak
8         public class MappingProfile : Profile
9         {
10            public MappingProfile()
11            {
12                //TODO : Mapping işlemleri burada yapılacak
13                //TODO:CreateMap<Source, Destination>();
14                //TODO: ReverseMap işlemi Book dan BookDtoUpdate e, BookDtoUpdate dem Book a dönüşüm yapacak
15                CreateMap<BookDtoUpdate, Book>().ReverseMap();
16
17                CreateMap<Book, BookDto>();
18
19                CreateMap<BookDtoForInsertion, Book>();
20
21            }
22        }
23    }
```

The screenshot shows the Visual Studio code editor with the BookManager.cs file open. It contains a GetOneBookForPatch method that retrieves a book by ID, checks if it's null, and then maps it to a BookDtoUpdate object using the IMapper interface. A line of code that performs the mapping is highlighted with a red rectangle.

```
87
88     _manger.UpdateBook(book);
89     _manger.Save();
90
91     public (BookDtoUpdate bookDtoUpdate, Book book) GetOneBookForPatch(int id, bool trackChanges)
92     {
93         var book = _manger.Book.GetOneBookById(id, trackChanges);
94         if (book is null)
95             throw new BookNotFoundException(id);
96
97         var bookDtoForUpdate = _mapper.Map<BookDtoUpdate>(book); //TODO: BookDtoUpdate ihtiyacı var.
98
99         //TODO: Tuple yapısı ile döndürüyoruz.
100        return (bookDtoForUpdate, book);
101    }
```

```
0 references
Presentation.Controllers.BooksController
PartiallyUpdateOneBook(int id, JsonPatchDocument<BookDtoUp
izenle
public IActionResult PartiallyUpdateOneBook([FromRoute(Name = "id")] int id, [FromBody] JsonPatchDocument<
    BookDtoUpdate> bookPatch)
{
    if (bookPatch is null)
        return BadRequest(); // 400

    var result = _manager
        .BookService
        .GetOneBookForPatch(id, false);

    bookPatch.ApplyTo(result.bookDtoUpdate, ModelState);
    TryValidateModel(result.bookDtoUpdate);

    if (!ModelState.IsValid)
        return UnprocessableEntity(ModelState); // 422

    _manager.BookService.SaveChangesForPatch(result.bookDtoUpdate, result.book);

    return NoContent(); // 204
}
```

# On ikinci Bölüm?

## Doğrulama (Validation) Sunum Katmanın Düzenlenmesi

«ÇALIŞTI»

The screenshot shows two Postman requests for validating a book object.

**Request 1: PATCH /api/books/2**

- Body:** Raw JSON with a validation error message: "title": [{"path": "title", "msg": "Title en az 2 karakter olacak."}]
- Response:** JSON response with the validation message: "title": [{"path": "title", "msg": "Title en az 2 karakter olacak."}]

**Request 2: PATCH /api/books/2**

- Body:** Raw JSON with a validation error message: "title": [{"path": "title", "msg": "Title en az 2 karakter olacak."}]
- Response:** JSON response with the validation message: "title": [{"path": "title", "msg": "Title en az 2 karakter olacak."}]

# **On ikinci Bölüm Bitti**

## **Hadi Özetalylim?**

# On üçüncü Bölüm?

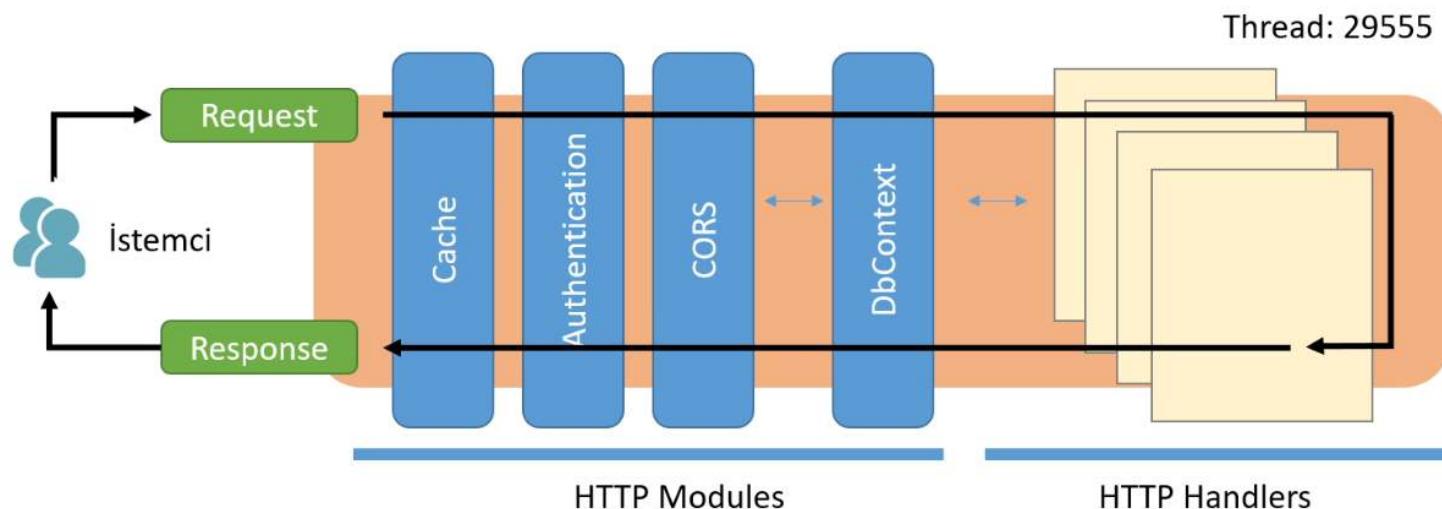
Asenkron Kod

async

- Asenkron programlama
- async, await ve Task Keywords
- Return Types
- Async Code Uygulaması
- Controller Düzenlemesi

# On üçüncü Bölüm?

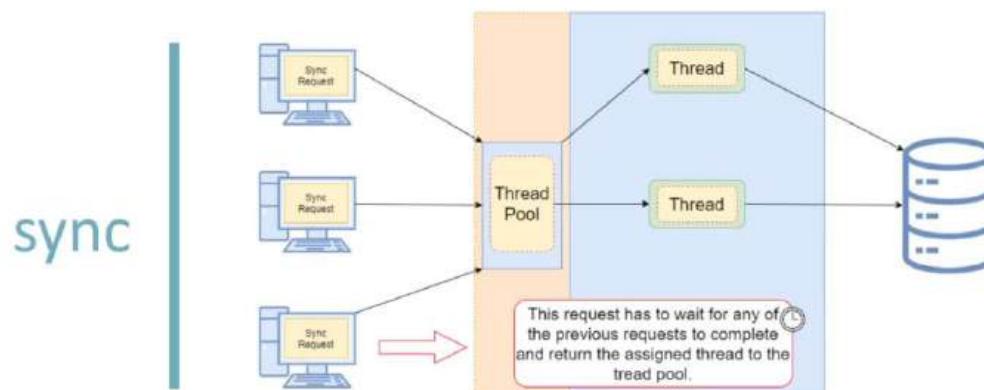
Asenkron Kod  
Sözcük Çalışması



# On üçüncü Bölüm?

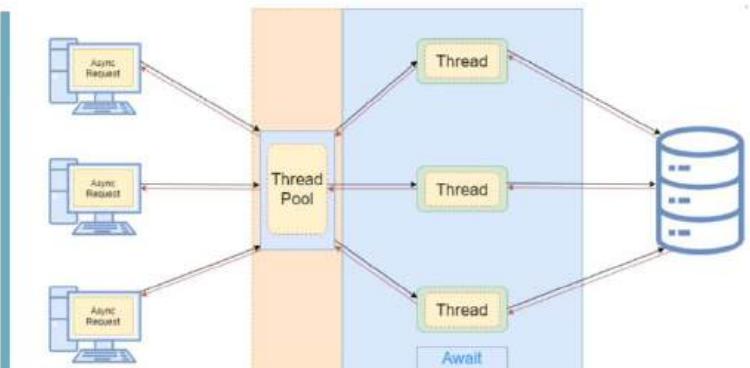
Asenkron Kod  
Sözcük Çalışması

«sekron yapıda thread havuzu dolduğunda  
yeni gelen istek beklemek zorunda kalır!!!»



sync

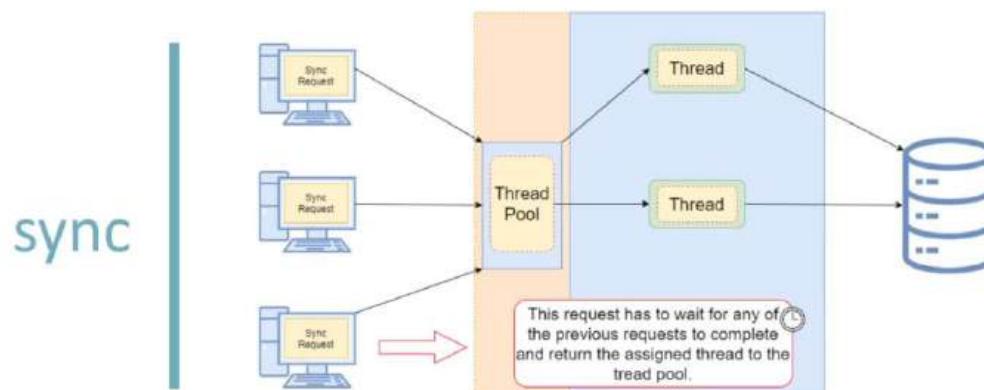
«asekron yapıda ise bekleme yoktur. Bir  
thread bir işlemi bitirmeden diğer işi  
alabilir!!!»



# On üçüncü Bölüm?

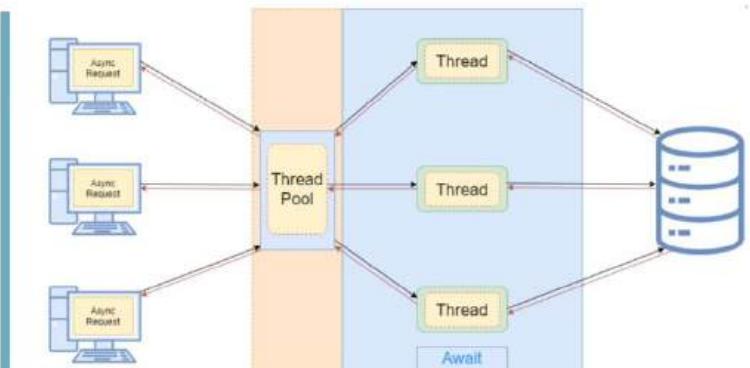
Asenkron Kod  
Sözcük Çalışması

«sekron yapıda thread havuzu dolduğunda  
yeni gelen istek beklemek zorunda kalır!!!»



sync

«asekron yapıda ise bekleme yoktur. Bir  
thread bir işlemi bitirmeden diğer işi  
alabilir!!!»



# On üçüncü Bölüm?

Asenkron Kod  
Sözcük Çalışması

«asenkron çalışmayı TASK ile  
sağlayacağız..!!!»

Task

- .Net dünyasında asenkron programlama için thread pool kullanılıyor.
- Thread'ler thread pool çekilir.
- Çalıştırılan thread yapılarının yönetimi sağlanır.
- Bu noktada Task kullanılır.

Task

- Task anahtar kelimesinin asenkron metod tanımlarken kullanılması gerektiğini ancak sonucu temsil etmediğini anlamak önemlidir.
- Task içerisinde çok **Status**, **IsCompleted**, **IsCanceled**, **IsFaulted** gibi çok sayıda property barındırır.
- Task ile biz operasyonun tamamlanıp tamamlanmadığını izleyebiliriz.

# On üçüncü Bölüm?

Asenkron Kod  
Sözcük Çalışması

await

- Sonuçların asenkron operasyondan çıkartılmasını sağlar.
- Operasyonun başarısını doğrular.
- Zaman uyumsuz yöntemde kodun geri kalanını yürütmek için devamı sağlar.
- `await` anahtar kelimesinin kullanılması zorunlu değildir.
- `await` anahtar kelimesi yalnızca bir kez kullanılmak zorunda da değildir.

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin **Repositories** katmanında değişiklik yapılacak.

The screenshot shows a Visual Studio environment with two code editors and a Solution Explorer window.

**Top Editor:** Displays the `IBookRepository.cs` interface. It contains methods for creating, deleting, getting one book by ID, and getting all books. The `GetAllBooksAsync` method is highlighted with a red underline, indicating a potential issue.

```
namespace Repositories.Contracts
{
    public interface IBookRepository : IRepositoryBase<Book>
    {
        Task<IList<Book>> GetAllBooksAsync(bool trackChanges);
        Book GetOneBookById(int id, bool trackChanges);
        void CreateBook(Book book);
        void UpdateBook(Book book);
        void DeleteBook(Book book);
    }
}
```

**Bottom Editor:** Displays the `BookRepository.cs` implementation. It overrides the `CreateBook`, `DeleteBook`, and `GetOneBookById` methods from the interface. The `GetAllBooksAsync` method is implemented using `ToListAsync` and includes a warning about a possible null reference return.

```
public void CreateBook(Book book) => Create(book);

public void DeleteBook(Book book) => Delete(book);

public async Task<IList<Book>> GetAllBooksAsync(bool trackChanges) =>
    await FindAll(trackChanges)
        .OrderBy(b => b.Id)
        .ToListAsync();

public Book GetOneBookById(int id, bool trackChanges) =>
    FindByCondition(x => x.Id.Equals(id), trackChanges)
        .SingleOrDefault();
```

**Solution Explorer:** Shows the solution structure with projects like Entities, Presentation, Dependencies, Contracts, EfCore, and Services.

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin **Repositories** katmanında değişiklik yapılacak.

The screenshot shows two code editors in Visual Studio. The left editor displays the `BookRepository.cs` file, which contains the following code:namespace Repositories.Contracts  
{  
 public interface IBookRepository : IRepositoryBase<Book>  
 {  
 Task<IList<Book>> GetAllBooksAsync(bool trackChanges);  
 Task<Book> GetOneBookByIdAsync(int id, bool trackChanges);  
 void CreateBook(Book book);  
 void UpdateBook(Book book);  
 void DeleteBook(Book book);  
 }  
}The right editor displays the `RepositoryBase.cs` file, which contains the following code:public async Task<IList<Book>> GetAllBooksAsync(bool trackChanges) ...  
public async Task<Book> GetOneBookByIdAsync(int id, bool trackChanges) =>  
 await FindByCondition(x => x.Id.Equals(id), trackChanges)  
 .SingleOrDefaultAsync();  
 ...  
 public void UpdateBook(Book book) => Update(book);  
}Both files are part of the `Repositories` project. The `BookRepository.cs` file has several TODO comments and a warning about a possible null reference return. The `RepositoryBase.cs` file has a warning about a possible null reference return. Solution Explorer on the right shows the project structure with multiple projects like `Entities`, `Dependencies`, `Contracts`, and `EFCore`.

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin **Repositories** katmanında değişiklik yapılacak.

```
 IRepositoryManager.cs
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6
 7  namespace Repositories.Contracts
 8  {
 9      //TODO: internal erişim belirleyicini public olarak değiştirin.
10      public interface IRepositoryManger
11      {
12          //TODO: Tüm repolara Manager Üzerinden erişim sağlayacağız.
13          IBookRepository Book { get; }
14          Task SaveAsync();
15      }
16  }
17

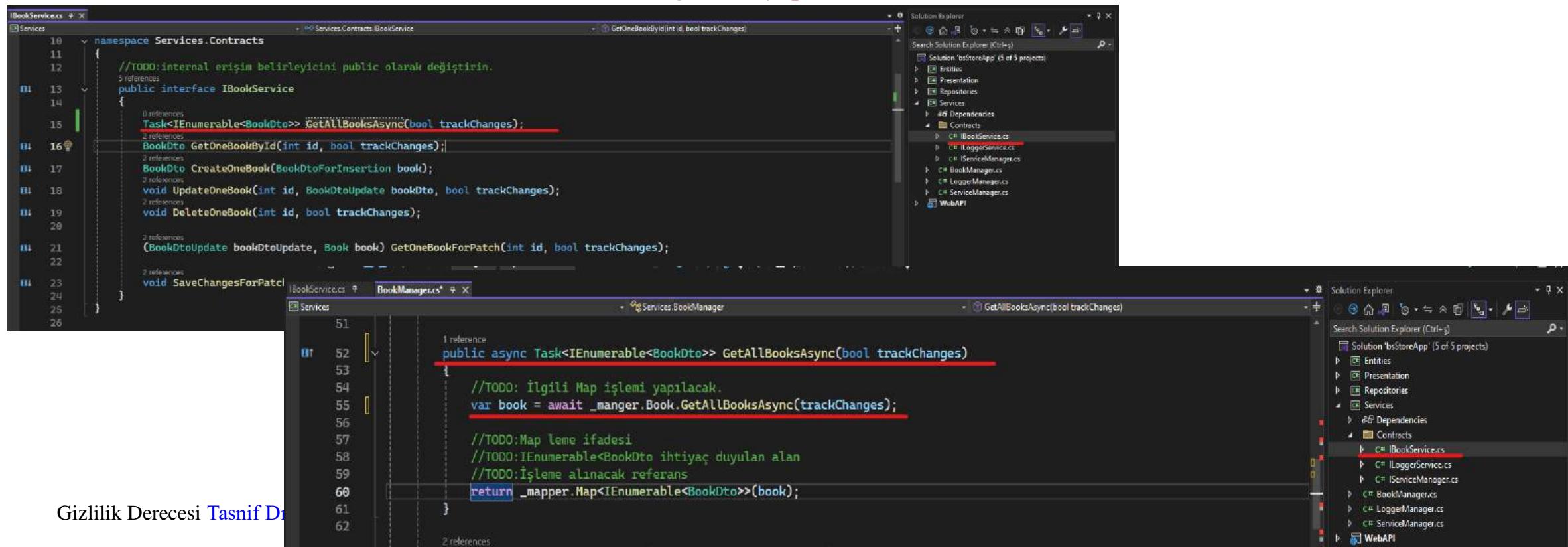
RepositoryManager.cs
 23
 24  // TODO: Her bir repo yu IoC kaydını kaymak istemiyoruz ...
 25  // References
 26  public IBookRepository Book => _bookRepository.Value; // Nesne ancak ve ancak kullanıldığından oluşturulacak. Lazy Loading yapmış olduk.
 27
 28  // TODO: Book ifadesi artık BookRepository karşılık gelecek.
 29
 30  public async Task SaveAsync()
 31  {
 32      await _context.SaveChangesAsync();
 33  }
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44

Solution Explorer
 1. Entities
    - Dependencies
    - DataTransferObjects
    - ErrorModel
    - Exceptions
      - BookNotFoundException.cs
      - NotFoundException.cs
    - Models
 2. Presentation
 3. Repositories
    - Contracts
      - BookRepository.cs
      - IRepositoryBase.cs
      - IRepositoryManager.cs
    - EfCore
    - Config
 4. Services
 5. WebAPI
```

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin Services katmanında değişiklik yapılacak.



The screenshot shows two code editors in Visual Studio. The top editor displays the `IBookService.cs` interface, which includes a method `Task<IEnumerable<BookDto>> GetAllBooksAsync(bool trackChanges);`. The bottom editor displays the `BookManager.cs` class, which implements this method. Both methods are highlighted with red boxes, indicating they are the focus of change. The Solution Explorer on the right shows the project structure, including the `Services` and `WebAPI` projects.

```
namespace Services.Contracts
{
    public interface IBookService
    {
        Task<IEnumerable<BookDto>> GetAllBooksAsync(bool trackChanges);
        BookDto GetOneBookById(int id, bool trackChanges);
        BookDto CreateOneBook(BookDtoForInsertion book);
        void UpdateOneBook(int id, BookDtoUpdate bookDto, bool trackChanges);
        void DeleteOneBook(int id, bool trackChanges);
        BookDtoUpdate bookDtoUpdate, Book book) GetOneBookForPatch(int id, bool trackChanges);
        void SaveChangesForPatch();
    }
}

public class BookManager : IBookService
{
    private readonly IMapper _mapper;
    private readonly BookRepository _book;

    public BookManager(IMapper mapper, BookRepository book)
    {
        _mapper = mapper;
        _book = book;
    }

    public Task<IEnumerable<BookDto>> GetAllBooksAsync(bool trackChanges)
    {
        //TODO: İlgili Map işlemi yapılacak.
        var book = await _book.GetAllBooksAsync(trackChanges);

        //TODO: Map leme ifadesi
        //TODO: IEnumerable<BookDto> ihtiyaç duyulan alan
        //TODO: İşleme alınacak referans
        return _mapper.Map<IEnumerable<BookDto>>(book);
    }

    public BookDto GetOneBookById(int id, bool trackChanges)
    {
        var book = await _book.GetOneBookById(id, trackChanges);
        return _mapper.Map<BookDto>(book);
    }

    public void CreateOneBook(BookDto book)
    {
        var bookEntity = _mapper.Map<Book>(book);
        _book.CreateOneBook(bookEntity);
    }

    public void UpdateOneBook(int id, BookDtoUpdate book)
    {
        var bookEntity = _mapper.Map<Book>(book);
        bookEntity.Id = id;
        _book.UpdateOneBook(bookEntity);
    }

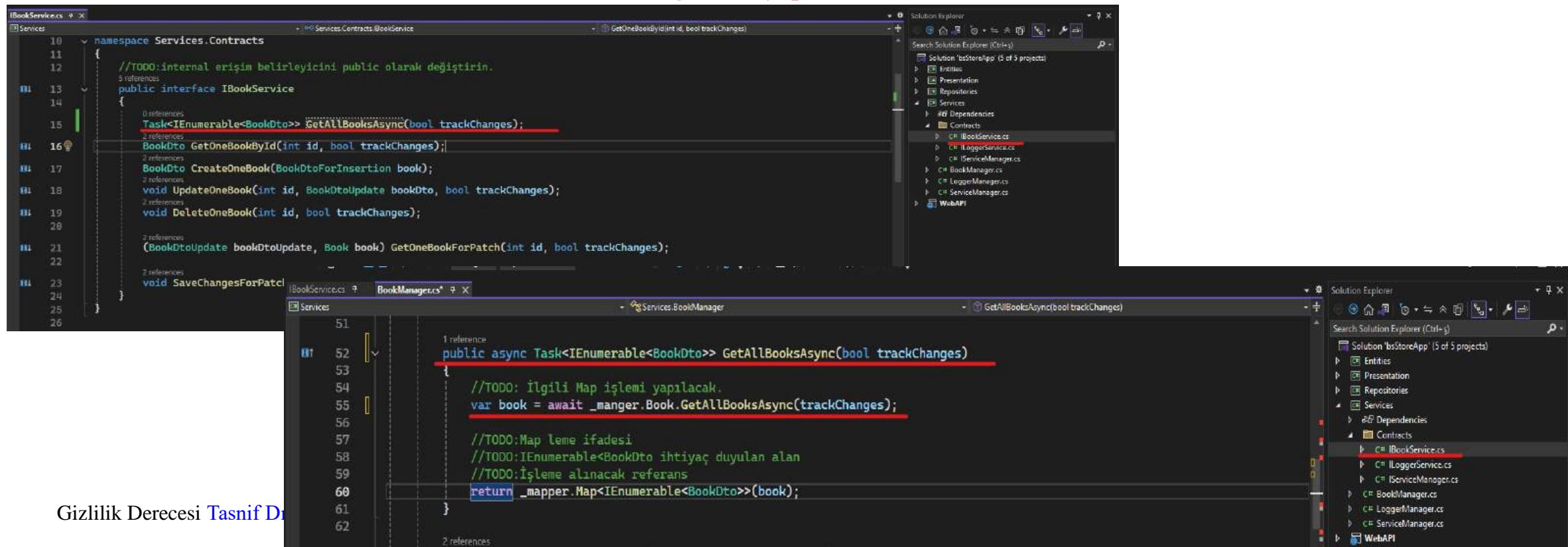
    public void DeleteOneBook(int id)
    {
        _book.DeleteOneBook(id);
    }

    public void SaveChangesForPatch()
    {
        _book.SaveChanges();
    }
}
```

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin Services katmanında değişiklik yapılacak.



The screenshot shows two code editors in Visual Studio. The top editor displays the `IBookService.cs` interface, which includes a method `Task<IEnumerable<BookDto>> GetAllBooksAsync(bool trackChanges);`. The bottom editor displays the `BookManager.cs` class, which implements this method. Both methods are highlighted with red boxes, indicating they are the focus of change. The Solution Explorer on the right shows the project structure, including the `Services` and `WebAPI` projects.

```
namespace Services.Contracts
{
    public interface IBookService
    {
        Task<IEnumerable<BookDto>> GetAllBooksAsync(bool trackChanges);
        BookDto GetOneBookById(int id, bool trackChanges);
        BookDto CreateOneBook(BookDtoForInsertion book);
        void UpdateOneBook(int id, BookDtoUpdate bookDto, bool trackChanges);
        void DeleteOneBook(int id, bool trackChanges);
        BookDtoUpdate bookDtoUpdate, Book book) GetOneBookForPatch(int id, bool trackChanges);
        void SaveChangesForPatch();
    }
}

public async Task<IEnumerable<BookDto>> GetAllBooksAsync(bool trackChanges)
{
    //TODO: İlgili Map işlemi yapılacak.
    var book = await _manger.Book.GetAllBooksAsync(trackChanges);

    //TODO: Map leme ifadesi
    //TODO: IEnumerable<BookDto> ihtiyaç duyulan alan
    //TODO: İşleme alınacak referans
    return _mapper.Map<IEnumerable<BookDto>>(book);
}
```

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin Services katmanında değişiklik yapılacak.

The screenshot shows two code editors in Visual Studio. The top editor displays the `IBookService.cs` interface:

```
10  namespace Services.Contracts
11  {
12      //TODO:internal erişim belirleyicini public olarak değiştirin.
13      public interface IBookService
14      {
15          Task<IList<BookDto>> GetAllBooksAsync(bool trackChanges);
16          Task<BookDto> GetOneBookByIdAsync(int id, bool trackChanges);
17          BookDto CreateOneBook(BookDtoForInsertion book);
18          void UpdateOneBook(int id, BookDtoUpdate bookDto, bool trackChanges);
19          void DeleteOneBook(int id, bool trackChanges);
20          BookDtoUpdate BookService.cs    BookManager.cs
21          Book book) GetOneBookForPatch(int id, bool trackChanges);
22
23          void SaveChangesForPa
24
25      }
26 }
```

The bottom editor displays the `BookManager.cs` implementation:

```
62  public async Task<BookDto> GetOneBookByIdAsync(int id, bool trackChanges)
63  {
64      var book = await _manger.Book.GetOneBookByIdAsync(id, trackChanges);
65      if (book is null)
66          throw new BookNotFoundException(id);
67
68      return _mapper.Map<BookDto>(book);
69  }
70
71  public void UpdateOneBook(int id, BookDtoUpdate bookDto, bool trackChanges)
72  {
73      //check entity
74      var entity = _manger.Book.GetOneBookByIdAsync(id, trackChanges);
75  }
76
77  public void DeleteOneBook(int id, bool trackChanges)
78  {
79      var entity = _manger.Book.GetOneBookByIdAsync(id, trackChanges);
80  }
81
82  private readonly IMapper _mapper;
83  private readonly Book _manger;
```

The Solution Explorer on the right shows the project structure for 'bStoreApp' with multiple projects like Entities, Presentation, Repositories, and Services.

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin Services katmanında değişiklik yapılacak.

The screenshot shows two code editors side-by-side. The left editor displays the `IBookService.cs` interface, which includes methods for getting all books, getting one book by ID, creating a new book, updating a book, deleting a book, and performing a patch operation. The right editor displays the `BookManager.cs` implementation, which uses a mapper to map DTOs to entities, creates the book, saves it, and then maps it back to a DTO. Both files are part of a solution named 'bsStoreApp' with five projects. The `IBookService.cs` file is highlighted in the Solution Explorer on the right.

```
namespace Services.Contracts
{
    //TODO:internal erişim belirleyicini public olarak değiştirin.
    public interface IBookService
    {
        Task<IEnumerable<BookDto>> GetAllBooksAsync(bool trackChanges);
        Task<BookDto> GetOneBookByIdAsync(int id, bool trackChanges);
        Task<BookDto> CreateOneBookAsync(BookDtoForInsertion book);
        void UpdateOneBook(int id, BookDtoUpdate bookDto, bool trackChanges);
        void DeleteOneBook(int id, bool trackChanges);
        (BookDtoUpdate bookDtoUpdate, Book book) GetOneBookForPatch(int id, bool trackChanges);
        void SaveChangesForPatch(BookDtoUpdate bookDtoUpdate, Book book);
    }
}
```

```
public async Task<BookDto> CreateOneBookAsync(BookDtoForInsertion bookDto)
{
    var entity = _mapper.Map<Book>(bookDto);

    _manger.Book.CreateBook(entity);
    await _manger.SaveAsync();

    return _mapper.Map<BookDto>(entity);
}
```

Gizlilik Derecesi [Tasnif Dışı](#)

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin Services katmanında değişiklik yapılacak.

The screenshot shows two code editors in Visual Studio. The left editor displays `BookManager.cs` with several methods annotated with `[TODO]`. The right editor displays `BookServices.cs` with a `public async Task UpdateOneBookAsync(int id, BookDtoUpdate bookDto, bool trackChanges)` method. Both files are part of a solution named `bsStoreApp` which includes projects for Entities, Presentations, Repositories, Services, and WebAPI. The `Services` project contains `IBookService.cs`, `ILoggerService.cs`, `IServiceManager.cs`, `BookManager.cs`, `LoggerManager.cs`, and `ServiceManager.cs`.

```
BookManager.cs
namespace Services.Contracts
{
    public interface IBookService
    {
        Task<IEnumerable<BookDto>> GetAllBooksAsync(bool trackChanges);
        Task<BookDto> GetOneBookByIdAsync(int id, bool trackChanges);
        Task<BookDto> CreateOneBookAsync(BookDtoForInsertion book);
        Task UpdateOneBookAsync(int id, BookDtoUpdate bookDto, bool trackChanges);
        void DeleteOneBook(int id, bool trackChanges);
    }
}

BookServices.cs
public class BookManager : IBookService
{
    private readonly IMapper _mapper;
    private readonly IRepository<Book> _manger;

    public BookManager(IMapper mapper, IRepository<Book> manger)
    {
        _mapper = mapper;
        _manger = manger;
    }

    public async Task<Book> GetOneBookByIdAsync(int id, bool trackChanges)
    {
        return _manger.GetOneBookByIdAsync(id, trackChanges);
    }

    public async Task UpdateOneBookAsync(int id, BookDtoUpdate bookDto, bool trackChanges)
    {
        //check entity
        var entity = await _manger.Book.GetOneBookByIdAsync(id, trackChanges);

        //check book null
        if (entity is null)
            throw new BookNotFoundException(id);

        //TODO: Burada Mapping işlemi var
        //TODO: Burada 10 tane alan olsa tek tek eşleyeceğiz. Bunun için Mapping işlemi yapacağız.
        //entity.Title = book.Title;
        //entity.Price = book.Price;

        entity = _mapper.Map<Book>(bookDto); //TODO: Bize <Book> lazım nihai olarak bookDto kaynaklık edecek.

        _manger.Book.UpdateBook(entity);
        await _manger.SaveAsync();
    }
}
```

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin Services katmanında değişiklik yapılacak.

The screenshot shows a Visual Studio interface with two code editors and a Solution Explorer window.

**IBookService.cs:**

```
10  namespace Services.Contracts
11  {
12      //TODO: internal erişim belirleyicini public olarak değiştirin.
13      public interface IBookService
14      {
15          Task<IEnumerable<BookDto>> GetAllBooksAsync(bool trackChanges);
16          Task<BookDto> GetOneBookByIdAsync(int id, bool trackChanges);
17          Task<BookDto> CreateOneBookAsync(BookDtoForInsertion book);
18          Task UpdateOneBookAsync(int id, BookDtoUpdate bookDto, bool trackChanges);
19          Task DeleteOneBookAsync(int id, bool trackChanges);
20      }
21  }
```

**BookManager.cs:**

```
23  void Save()
24  {
25      _logger = logger;
26      _mapper = mapper;
27  }
28
29
30  public async Task<BookDto> CreateOneBookAsync(BookDtoForInsertion bookDto)...
31
32  public async Task DeleteOneBookAsync(int id, bool trackChanges)
33  {
34      //check entity
35      var entity = await _manger.Book.GetOneBookByIdAsync(id, trackChanges);
36
37      if (entity is null)
38          throw new BookNotFoundException(id);
39
40      _manger.Book.DeleteBook(entity);
41      await _manger.SaveAsync();
42  }
43
44
45
46
47
48
49
50
51 }
```

**Solution Explorer:**

- Solution 'bsStoreApp' (5 of 5 projects)
  - Entities
  - Presentation
  - Repositories
  - Services
    - Dependencies
    - Contracts
      - C# IBookService.cs
      - C# ILoggerService.cs
      - C# ServiceManager.cs
    - C# BookManager.cs
    - C# LoggerManager.cs
    - C# ServiceManager.cs
  - WebAPI

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin Services katmanında değişiklik yapılacak.

The screenshot shows two code editors in Visual Studio. The top editor displays `BookService.cs` with several asynchronous methods defined:

```
using System.Threading.Tasks;
namespace Services.Contracts
{
    //TODO:internal erişim belirleyicini public olarak değiştirin.
    public interface IBookService
    {
        Task<IEnumerable<BookDto>> GetAllBooksAsync(bool trackChanges);
        Task<BookDto> GetOneBookByIdAsync(int id, bool trackChanges);
        Task<BookDto> CreateOneBookAsync(BookDtoForInsertion book);
        Task UpdateOneBookAsync(int id, BookDtoUpdate bookDto, bool trackChanges);
        Task DeleteOneBookAsync(int id, bool trackChanges);

        Task<(BookDtoUpdate bookDtoUpdate, Book book)> GetOneBookForPatchAsync(int id, bool trackChanges);
    }
}
```

The bottom editor displays `BookManager.cs` with a method implementation:

```
public async Task<(BookDtoUpdate bookDtoUpdate, Book book)> GetOneBookForPatchAsync(int id, bool trackChanges)
{
    var book = await _manger.Book.GetOneBookByIdAsync(id, trackChanges);
    if (book is null)
        throw new BookNotFoundException(id);

    var bookDtoForUpdate = _mapper.Map<BookDtoUpdate>(book); //TODO: BookDtoUpdate ihtiyacı var.

    //TODO: Tuple yapısı ile döndürüyorum.
    return (bookDtoForUpdate, book);
}
```

Both code files are part of the `Services` project, which is shown in the Solution Explorer on the right. The `Contracts` folder contains the `IBookService` interface. The `Services` folder contains the `BookService` class and the `BookManager` class.

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin Services katmanında değişiklik yapılacak.

The screenshot shows two code editors in Visual Studio. The top editor displays the `IBookService.cs` interface:

```
8  using System.Threading.Tasks;
9
10 namespace Services.Contracts
11 {
12     //TODO:internal erişim belirleyicini public olarak değiştirin.
13     public interface IBookService
14     {
15         Task<IEnumerable<BookDto>> GetAllBooksAsync(bool trackChanges);
16         Task<BookDto> GetOneBookByIdAsync(int id, bool trackChanges);
17         Task<BookDto> CreateOneBookAsync(BookDtoForInsertion book);
18         Task UpdateOneBookAsync(int id, BookDtoUpdate bookDto, bool trackChanges);
19         Task DeleteOneBookAsync(int id, bool trackChanges);
20
21         Task<BookDtoUpdate> GetOneBookForPatchAsync(int id, bool trackChanges);
22
23         Task SaveChangesForPatchAsync(BookDtoUpdate book);
24     }
25 }
```

The bottom editor displays the `BookManager.cs` implementation:

```
99
100    return (bookToUpdate, book);
101
102    public async Task SaveChangesForPatchAsync(BookDtoUpdate bookToUpdate, Book book)
103    {
104        _mapper.Map(bookToUpdate, book); //TODO: BookDtoUpdate ile Book'u eşleştiriyoruz.
105        await _manger.SaveAsync(); //TODO: Değişiklikleri kaydediyoruz.
106    }
107
108
109 }
```

Both files have several TODO comments and some code is underlined with red, indicating it needs to be updated. The Solution Explorer on the right shows the project structure with `IBookService.cs` and `BookManager.cs` highlighted.

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin Presentation katmanında değişiklik yapılacak.

```
BooksController.cs
public async Task<IActionResult> GetAllBooksAsync()
{
    ServiceManager Sonrası kapatılan kod !!!
    var books = await _manager.BookService.GetAllBooksAsync(false);
    return Ok(books);
}

//TODO: Imzalı Get metodu oluşturalım.
[HttpGet("{id:int}")]
public async Task<IActionResult> GetOneBookAsync([FromRoute(Name = "id")] int id)
{
    //TODO: Veri tabanına erişemeyebiliriz. Veri tabanı ayakta olmamıştır. Ağda silinti olabilir. Bunun için try catch kullanıyoruz.

    ServiceManager Sonrası kapatılan kod !!!
    var book = await _manager
        .BookService
        .GetOneBookByIdAsync(id, false);
}

//TODO: Post metodu oluşturalım.
[HttpPost]
public async Task<IActionResult> CreateOneBookAsync([FromBody] BookDtoForInsertion bookDto)
{
    if (bookDto == null)
    {
        return BadRequest(); // 400
    }
    ServiceManager Sonrası kapatılan kod !!!
    if (!ModelState.IsValid)
        return UnprocessableEntity(ModelState); // 422
    var bookCreate = await _manager
        .BookService
        .CreateOneBookAsync(bookDto);
    return StatusCode(201, bookDto); // 201
}
```

Slayt Numarası .../...

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin **Presentation** katmanında değişiklik yapılacak.

The screenshot shows the BooksController.cs file in the Visual Studio editor. The code implements an asynchronous update method:

```
public async Task<IActionResult> UpdateOneBookAsync([FromRoute(Name = "id")] int id, [FromBody] BookDtoUpdate bookDto)
{
    //TODO: Güncelleme yapmak için gelen book var mı kontrol et.
    ServiceManager Sonrası kapatılan kod !!!
    if (bookDto == null)
    {
        return BadRequest(); // 400
    }
    if (!ModelState.IsValid)
        return UnprocessableEntity(ModelState); // 422
    await _manager.BookService.UpdateOneBookAsync(id, bookDto, false);
    return NoContent(); // 204
}
```

The Solution Explorer on the right shows the project structure for 'bsStoreApp' with five projects: Entities, Presentation, Dependencies, Services, and WebAPI. The BooksController.cs file is selected under the Presentation project's Controllers folder.

The screenshot shows the BooksController.cs file in the Visual Studio editor. The code implements an asynchronous delete method:

```
//TODO: Delete metodü oluşturulacak. (Belirtilen sil)
[HttpDelete("{id:int}")]
public async Task<IActionResult> DeleteOneBookAsync([FromRoute(Name = "id")] int id)
{
    ServiceManager Sonrası kapatılan kod !!!
    await _manager.BookService.DeleteOneBookAsync(id, true);
    return NoContent(); // 204
}
```

The Solution Explorer on the right shows the project structure for 'bsStoreApp' with five projects: Entities, Presentation, Dependencies, Services, and WebAPI. The BooksController.cs file is selected under the Presentation project's Controllers folder.

# On üçüncü Bölüm?

## Asenkron Kod

Asenkron Kod İçin **Presentation** katmanında değişiklik yapılacak.

The screenshot shows the Visual Studio IDE with the BooksController.cs file open in the main editor. The code implements an asynchronous controller action for updating a book. Several lines of code are highlighted with red underlines, indicating they are part of the changes being made. The Solution Explorer on the right shows the project structure, including the Presentation project which contains the BooksController.cs file.

```
BooksController.cs
165 [HttpPatch("{id:int}")]
166 public async Task<IActionResult> PartiallyUpdateOneBookAsync([FromRoute(Name = "id")] int id, [FromBody]
167     JsonPatchDocument<BookDtoUpdate> bookPatch)
168 {
169     if (bookPatch == null)
170         return BadRequest(); // 400
171
172     var result = await _manager
173         .BookService
174         .GetOneBookForPatchAsync(id, false);
175
176     bookPatch.ApplyTo(result.bookDtoUpdate, ModelState);
177
178     TryValidateModel(result.bookDtoUpdate);
179
180     if (!ModelState.IsValid)
181         return UnprocessableEntity(ModelState); // 422
182
183     await _manager.BookService.SaveChangesForPatchAsync(result.bookDtoUpdate, result.book);
184
185     return NoContent(); // 204
186 }
187 }
```

# On üçüncü Bölüm?

«Diğer API'leri siz test edin.!!!»

## Asenkron Kod API Test

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Books' selected, displaying various API endpoints such as 'GET All Books', 'GET One Book', etc. The main area shows a 'Books / All Books' collection with a single 'GET' request highlighted. The request URL is `{baseUrl}/api/books`. The 'Body' tab is selected, showing the message 'This request does not have a body'. Below the request, the response is displayed in a JSON table format. The response status is 200 OK, with a duration of 3.67 s and a size of 923.8. The JSON data contains 28 items, each representing a book with properties like id, title, and price.

1		"id": 1,
2		"title": "Karakötür ve Hanımat",
3		"price": 75.00
4		2,
5		3,
6		"id": 2,
7		"title": "650",
8		"price": 100.00
9		4,
10		5,
11		"id": 3,
12		"title": "Devlet",
13		"price": 370.00
14		6,
15		7,
16		"id": 1001,
17		"title": "Dense",
18		"price": 1000.00
19		8,
20		9,
21		10,
22		11,
23		12,
24		13,
25		14,
26		15,
27		16,
28		17,

# **On üçüncü Bölüm Bitti**

## **Hadi Özetleyelim?**



TÜRKİYE CUMHURİYETİ  
**MİLLÎ SAVUNMA BAKANLIĞI**

**TEŞEKKÜRLER**