تابع render

تابع render در jQuery نیست، اما در DataTables که یک افزونه jQuery است، از تابع render استفاده می‌شود. این تابع برای تعیین نحوه نمایش داده‌ها در ستون‌های DataTables استفاده می‌شود. به عنوان مثال، می‌توان از این تابع برای تغییر نحوه نمایش یک مقدار

مشخص در ستون استفاده کرد، مانند تغییر یک مقدار boolean به یک متن خاص یا ایجاد یک دکمه عملیاتی بر اساس مقادیر دریافتی.

در کدی که ارائه داده‌اید، تابع render برای تعیین نحوه نمایش وضعیت و توضیحات در ستون‌های مربوطه استفاده شده است. این تابع به صورت شخصی‌سازی شده و بر اساس شرایط مختلف، مقادیر مختلفی را برای نمایش تولید می‌کند.

{

title: 'وضعیت',

data: 'State',

render: function (data, type, full) {

var result = "";

if (full.Accepted == 1) {

result = "تایید شده" ;

}

else if (full.Accepted == 0 || (full.Accepted == null && full.Stateid == 11))

{

result = "رد شده";

}

//else if (full.Stateid == 12) {

// result = "ویرایش شد و درگردش جدید ";

//}

else {

result = "در حال بررسی";

}

return result;

}

},

این کد یک تابع render را در DataTables نشان می‌دهد. این تابع برای تعیین نحوه نمایش وضعیت در یک ستون استفاده می‌شود. این تابع به عنوان ورودی سه پارامتر data، type و full را دریافت می‌کند.

در این تابع، مقدار وضعیت ورودی data و سایر مقادیر موجود در ردیف (full) بررسی می‌شود و بر اساس شرایط مختلف، یک مقدار نمایشی برای وضعیت تولید می‌شود. به عنوان مثال، اگر Accepted برابر با 1 باشد، مقدار "تایید شده" تولید می‌شود. اگر Accepted برابر

با 0 یا null و Stateid برابر با 11 باشد، مقدار "رد شده" تولید می‌شود. در غیر این صورت، مقدار "در حال بررسی" تولید می‌شود.

تابع render در DataTables یک تابع callback است که بیشتر در جاوااسکریپت استفاده می‌شود. این تابع به شما امکان می‌دهد مقدار یک ستون را به صورت دلخواه تغییر داده و نمایش دهید. این تابع به شما امکان می‌دهد که مقادیر ستون‌ها را بر اساس شرایط خاصی تغییر داده و نمایش دهید.

در DataTables، تابع render به صورت زیر استفاده می‌شود:

$('#example').DataTable( {

"columnDefs": [ {

"targets": 0,

"data": "name",

"render": function ( data, type, full, meta ) {

return type === 'display' && data.length > 10 ?

'<span title="'+data+'">'+data.substr( 0, 10 )+'...</span>' :

data;

}

} ]

} );

در این مثال، تابع render برای ستون اول تعریف شده است. این تابع مقدار ورودی (data) را بررسی می‌کند و اگر طول مقدار بیشتر از 10 باشد، مقداری کوتاه شده و با اضافه کردن علامت سه نقطه به انتها به جای آن نمایش داده می‌شود.

به طور کلی، تابع render به شما امکان می‌دهد تا مقادیر ستون‌ها را به صورت شخصی‌سازی شده تغییر داده و نمایش دهید.

DataTable

در مثال ارائه شده، DataTable یک کتابخانه JavaScript است که برای ایجاد و مدیریت جداول داده در وب‌سایت‌ها استفاده می‌شود. DataTables امکانات بسیاری برای جستجو، مرتب‌سازی، فیلترینگ و نمایش داده‌ها را فراهم می‌کند و به شما امکان می‌دهد تا جداول داده را به راحتی و با امکانات گسترده‌ای نمایش دهید.

با استفاده از DataTables، شما می‌توانید داده‌های خود را از منابع مختلفی مانند آرایه‌ها، اشیاء JSON، سرورهای وب و... بخوانید و آنها را به صورت جدول نمایش دهید. DataTables امکانات متنوعی را برای سفارشی‌سازی نمایش داده‌ها ارائه می‌دهد، از جمله تغییر نحوه نمایش مقادیر ستون‌ها که در مثال قبلی آورده شده است.

به طور خلاصه، DataTables یک کتابخانه قدرتمند JavaScript برای نمایش جداول داده در وب‌سایت‌ها است که امکانات گسترده‌ای برای مدیریت و نمایش داده‌ها فراهم می‌کند.

Targets

در مثال بالا، `targets` یک ویژگی از تنظیمات DataTables است که برای تعیین ستون‌هایی که می‌خواهید تنظیمات خاصی را بر روی آنها اعمال کنید، استفاده می‌شود. این ویژگی به شما امکان می‌دهد تا شماره ستون یا نام ستون مورد نظر را مشخص کنید.

در مثال بالا، `targets: 0` تعیین شده است که به معنی این است که تنظیمات مورد نظر بر روی ستون اول اعمال می‌شود. این شماره ستون ممکن است بر اساس اینکه چگونه DataTables شماره‌گذاری ستون‌ها را انجام می‌دهد (مانند از صفر شروع می‌کند یا نه)، متفاوت باشد.

به طور کلی، با استفاده از `targets` شما می‌توانید ستون‌های مورد نظر خود را بر اساس شماره یا نام آنها مشخص کنید و تنظیمات خاصی را بر روی آنها اعمال کنید.

columnDefs

در مثال بالا، `columnDefs` یک ویژگی از تنظیمات DataTables است که برای تعریف تنظیمات ستون‌ها استفاده می‌شود. این ویژگی به شما امکان می‌دهد تا تنظیمات خاصی را برای هر ستون تعریف کنید.

در مثال بالا، `columnDefs` تعریف شده است که به شما امکان می‌دهد تا تنظیمات خاصی را برای ستون اول (با `targets: 0`) اعمال کنید. در این حالت، ما `render` را برای تغییر نحوه نمایش مقادیر ستون اول تعریف کرده‌ایم.

به طور کلی، `columnDefs` به شما امکان می‌دهد تا تنظیمات خاصی را برای هر ستون تعریف کنید و از آن برای اعمال تغییرات شخصی‌سازی شده بر روی مقادیر ستون‌ها استفاده کنید.

کار کد زیر چیست؟

private readonly ILogger<WeatherForecastController> \_logger;

public WeatherForecastController(ILogger<WeatherForecastController> logger)

{

\_logger = logger;

}

این کد یک کنترلر به نام WeatherForecastController است که از یک وابستگی به ILogger برخوردار است. ILogger یک واسط است که برای ثبت و نگهداری لاگ‌ها در برنامه‌های .NET استفاده می‌شود. در اینجا از ILogger برای ثبت لاگ‌ها و دیباگ کردن کنترلر WeatherForecast استفاده می‌شود. در constructor کلاس WeatherForecastController، یک نمونه از ILogger به عنوان ورودی دریافت می‌کند و آنرا در یک readonly field ذخیره می‌کند تا بتواند از آن در سراسر کلاس استفاده کند.

لاگ گیری یا Logging چیست؟

امروزه لاگ گرفتن از سیستم یکی از اساسی ترین مواردی است که هر اپلیکیشنی به آن نیاز دارد. برای عیب یابی هر گونه مشکل در سیستم استفاده از logger امری ضروری است. Logger برای ایجاد فایل های خطای سفارشی شده یا رویدادهای سیستم استفاده می شود.

با کمک logging نه تنها می‌ توانیم آنچه را که در برنامه اتفاق می‌افتد ردیابی کنیم، بلکه به ردیابی هر نوع exception نیز کمک می‌کند. این exception در واقع زمانی مفید است که متد دقیق و شماره خط را با جزئیات پیام مناسب نشان دهد. بنابراین با کمک log، توسعه دهندگان می توانند به راحتی بفهمند که خطای واقعی چیست.

در معماری سه لایه، DTO (Data Transfer Object) ها در لایه سرویس (Service Layer) باید ساخته شوند. لایه سرویس وظیفه‌ی ارتباط با لایه داده (Data Access Layer) و لایه معرفی (Presentation Layer) را دارد، بنابراین DTO ها برای انتقال داده بین این دو لایه باید در این لایه ساخته شوند. با استفاده از DTO ها، داده‌ها به صورت ساختاری واحد و منظم بین لایه‌های مختلف انتقال می‌یابند و از این روش برای جلوگیری از وابستگی مستقیم بین لایه‌ها و افزایش انعطاف‌پذیری استفاده می‌شود.

modelها در معماری clean در کدام لایه قرار میگیرند؟

در معماری Clean، مدل‌ها (Models) به عنوان نماینده‌های داده‌ای و منطقی کسب و کار، در لایه Domain قرار می‌گیرند. لایه Domain مسئولیت اصلی اجرای منطق کسب و کار و نگهداری مدل‌های داده‌ای را دارد.

مدل‌ها در لایه Domain به عنوان شیوه‌ای برای نمایش و ذخیره داده‌ها و همچنین اجرای قوانین و منطق کسب و کار استفاده می‌شوند. این مدل‌ها می‌توانند شامل کلاس‌ها، ساختارهای داده‌ای، رویدادها و سایر اجزایی باشند که برای نمایش و پردازش داده‌ها مورد استفاده قرار می‌گیرند.

استفاده از لایه Domain برای قرار دادن مدل‌ها کمک می‌کند تا منطق کسب و کار از سایر جزئیات مانند دیتابیس، رابط‌های کاربری و سایر فناوری‌ها جدا شود. این کار باعث می‌شود تا مدل‌ها قابلیت تست و تغییرات راحت‌تری داشته باشند و از پیچیدگی‌های ناشی از وابستگی‌های زیاد به فناوری‌های خاص جلوگیری شود.

محل قرارگیری DTO ها در مماری clean کجاست؟

در معماری Clean، DTO ها (Data Transfer Objects) معمولاً در لایه Presentation یا لایه Interface قرار می‌گیرند. این لایه مسئولیت نمایش اطلاعات به کاربر و دریافت اطلاعات از کاربر را دارد و به عنوان واسط بین لایه‌های مختلف در سیستم عمل می‌کند.

DTO ها به عنوان یک مدل ساده و بدون منطق کسب و کار، برای انتقال داده‌ها بین لایه Presentation و لایه Domain استفاده می‌شوند. زمانی که اطلاعات از لایه Domain به لایه Presentation ارسال می‌شود یا از لایه Presentation به لایه Domain دریافت می‌شود، DTO ها برای نمایش و انتقال داده‌ها استفاده می‌شوند.

به طور کلی، DTO ها باید به عنوان یک واسط ساده و بدون منطق کسب و کار بین لایه Presentation و لایه Domain قرار بگیرند تا از ایجاد وابستگی‌های زیاد و پیچیدگی‌های اضافی در سیستم جلوگیری شود.

محل قرار گرفتن businesslogic در معماری clean

در معماری Clean، Business Logic در لایه Domain قرار می‌گیرد. این لایه مسئولیت اصلی اجرای منطق کسب و کار، پردازش داده‌ها و اعمال قوانین کسب و کار را دارد. لایه Domain به عنوان قلب سیستم، مدل‌ها، ریپازیتوری‌ها، سرویس‌ها و سایر عناصری را که مربوط به منطق کسب و کار هستند، شامل می‌شود.

به عنوان مثال، اگر یک برنامه مدیریت سفارشات داشته باشیم، قوانین مربوط به ایجاد، ویرایش و حذف سفارشات، محاسبه قیمت‌ها، اعتبارسنجی و سایر فعالیت‌های مربوط به سفارشات در لایه Domain قرار دارد.

استفاده از لایه Domain برای قرار دادن Business Logic کمک می‌کند تا منطق کسب و کار از سایر جزئیات مانند دیتابیس، رابط‌های کاربری و سایر فناوری‌ها جدا شود. این کار باعث می‌شود تا Business Logic قابلیت تست و تغییرات راحت‌تری داشته باشد و از پیچیدگی‌های ناشی از وابستگی‌های زیاد به فناوری‌های خاص جلوگیری شود.

در معماری Clean، ما از اصول Dependency Inversion و Dependency Injection برای مدیریت وابستگی‌ها و refrenceدهی استفاده می‌کنیم. این اصول به ما کمک می‌کنند تا لایه‌های مختلف معماری Clean به صورت مستقل از یکدیگر عمل کنند و وابستگی‌های آن‌ها کاهش یابد.

در معماری Clean، وابستگی‌ها به این صورت مدیریت می‌شوند که لایه‌های داخلی به لایه‌های بیرونی وابستگی نداشته باشند. به عبارت دیگر، لایه‌های داخلی نباید به جزئیات پیاده‌سازی لایه‌های بیرونی وابسته باشند. این کار باعث می‌شود که لایه‌های مختلف به صورت مستقل قابل تست و تغییر باشند.

برای ایجاد این اصل، ما از الگوی Dependency Injection استفاده می‌کنیم. در این الگو، وابستگی‌های یک کلاس به صورت مستقیم تزریق می‌شوند و کلاس خود مسئول مدیریت وابستگی‌های خود نیست. این کار باعث می‌شود که وابستگی‌ها به صورت معکوس (Inversion of Control) مدیریت شوند و کلاس‌ها به صورت مستقل از وابستگی‌های خود قابل استفاده باشند.

به عنوان مثال، در معماری Clean، ما می‌توانیم از الگوی Dependency Injection برای تزریق وابستگی‌های لایه‌های مختلف مانند Repository ها، UseCase ها و Presenter ها استفاده کنیم. این کار باعث می‌شود که این لایه‌ها به صورت مستقل از یکدیگر قابل تست و تغییر باشند و وابستگی‌هایشان به صورت معکوس مدیریت شود.

آیا لایه application همان presentation است؟

نه، لایه application و لایه presentation در معماری نرم افزاری متفاوت هستند. لایه application مسئولیت اجرای منطق کسب و کار و کنترل جریان کاربری است، در حالی که لایه presentation مسئولیت نمایش دادن اطلاعات به کاربر و برقراری تعامل با واسط کاربری را دارد. به عبارت دیگر، لایه application به عنوان مدیر کلی سیستم عمل می‌کند در حالی که لایه presentation مسئولیت نمایش و ارتباط با کاربر را دارد.

بهترین محل قرار گرفتن dto در Api در کدام لایه است؟

DTO (Data Transfer Object) معمولاً برای انتقال داده‌ها بین لایه‌های مختلف سیستم استفاده می‌شود. بهترین مکان برای قرار دادن DTO در یک API، به وابستگی‌های واقعی پروژه و معماری مورد استفاده برمی‌گردد. اما در بسیاری از موارد، DTO ها در لایه Presentation یا همان لایه Controller قرار می‌گیرند. در این لایه، DTO برای ارتباط با کلاینت‌ها و انتقال داده‌ها بین سیستم و کاربران استفاده می‌شود.

در بعضی موارد، DTO ها ممکن است در لایه Service نیز استفاده شوند تا داده‌های ورودی و خروجی سرویس‌ها را تعریف کنند. اما عموماً DTO ها باید در لایه‌های Presentation یا Controller قرار گیرند تا به عنوان واسط بین سیستم و کاربران عمل کنند.

چطور در یک پروژه APi هم از dto و هم از BL استفاده کنم؟

در یک پروژه API، معمولاً از DTO (Data Transfer Object) برای انتقال داده‌ها بین لایه‌های مختلف و از BL (Business Logic) برای پردازش و منطق کسب و کار استفاده می‌شود. برای استفاده هر دو در یک پروژه API، می‌توانید به روش زیر عمل کنید:

1. تعریف DTO: ابتدا باید DTO های مربوطه را تعریف کنید. این DTO ها معمولاً شامل ورودی و خروجی‌های مختلفی که بین لایه‌های مختلف API استفاده می‌شوند می‌باشند.

2. تعریف Business Logic: سپس باید منطق کسب و کار خود را در لایه BL پیاده سازی کنید. این شامل هرگونه پردازش، محاسبات و منطق کسب و کار است که برای انجام عملیات مورد نیاز در API لازم است.

3. استفاده از DTO در BL: در لایه BL، می‌توانید از DTO ها برای دریافت ورودی‌ها، ارسال خروجی‌ها و انتقال داده‌ها به و از سرویس‌ها استفاده کنید.

4. استفاده از DTO در Controller: در لایه‌های Controller نیز می‌توانید از DTO ها برای دریافت و ارسال داده‌ها به و از سرویس‌ها استفاده کنید.

با این روش، شما می‌توانید از DTO برای انتقال داده‌ها بین لایه‌ها و از BL برای پردازش داده‌ها و انجام منطق کسب و کار استفاده کنید. این روش به شما کمک می‌کند تا کد خود را به شکل منظم و قابل توسعه‌ای طراحی کنید و از جداسازی منطق کسب و کار و منطق انتقال داده‌ها بهره مند شوید.

در یک پروژه api BL و dto را درکدام لایه ها قرار دهم؟

در یک پروژه API، معمولاً BL (Business Logic) و DTO (Data Transfer Object) در لایه‌های مختلف قرار می‌گیرند. این لایه‌ها عبارتند از:

1. لایه Controller یا Presentation: در این لایه، DTO ها برای انتقال داده‌ها بین سیستم و کاربران استفاده می‌شوند. همچنین در این لایه، DTO ها برای دریافت و ارسال داده‌ها از و به سرویس‌ها استفاده می‌شوند. همچنین BL نیز ممکن است در این لایه قرار بگیرد تا منطق کسب و کار برای درخواست‌های ورودی اعمال شود.

2. لایه Service: در این لایه، BL قرار می‌گیرد. این لایه شامل منطق کسب و کار و پردازش داده‌ها است. BL در اینجا برای انجام عملیات‌های کسب و کار مورد نیاز استفاده می‌شود.

بنابراین، DTO ها بیشتر در لایه‌های Presentation یا Controller قرار می‌گیرند تا به عنوان واسط بین سیستم و کاربران عمل کنند و BL بیشتر در لایه Service قرار می‌گیرد تا منطق کسب و کار را پردازش کند.

به طور کلی، بهترین روش برای قرار دادن BL و DTO در یک پروژه API این است که DTO ها در لایه‌های Presentation یا Controller قرار بگیرند و BL در لایه Service قرار گیرد. این روش به شما کمک می‌کند تا منطق کسب و کار و منطق انتقال داده‌ها را به شکل منظم و قابل توسعه‌ای جدا کنید و کد خود را به شکل سازمان‌یافته‌تری طراحی کنید.

البته! در اینجا یک مثال ساده از یک پروژه API سیستم مدیریت کتابخانه با استفاده از ASP.NET Core و C# است. در این مثال، ما از معماری Clean Architecture استفاده می‌کنیم تا لایه‌های مختلف را جدا کنیم.

ابتدا، یک DTO برای نمایش اطلاعات یک کتاب ایجاد می‌کنیم:

```csharp

// BookDTO.cs

public class BookDTO

{

public int Id { get; set; }

public string Title { get; set; }

public string Author { get; set; }

// سایر ویژگی‌های کتاب

}

```

سپس، یک سرویس برای مدیریت کتب ایجاد می‌کنیم:

```csharp

// IBookService.cs

public interface IBookService

{

Task<BookDTO> GetBookById(int id);

Task<IEnumerable<BookDTO>> GetAllBooks();

Task AddBook(BookDTO book);

Task DeleteBook(int id);

}

```

```csharp

// BookService.cs

public class BookService : IBookService

{

private readonly IBookRepository \_bookRepository;

public BookService(IBookRepository bookRepository)

{

\_bookRepository = bookRepository;

}

public async Task<BookDTO> GetBookById(int id)

{

var book = await \_bookRepository.GetBookById(id);

return MapBookToDTO(book);

}

public async Task<IEnumerable<BookDTO>> GetAllBooks()

{

var books = await \_bookRepository.GetAllBooks();

return books.Select(MapBookToDTO);

}

public async Task AddBook(BookDTO book)

{

var newBook = MapDTOToBook(book);

await \_bookRepository.AddBook(newBook);

}

public async Task DeleteBook(int id)

{

await \_bookRepository.DeleteBook(id);

}

private BookDTO MapBookToDTO(Book book)

{

return new BookDTO

{

Id = book.Id,

Title = book.Title,

Author = book.Author,

// سایر ویژگی‌های کتاب

};

}

private Book MapDTOToBook(BookDTO bookDTO)

{

return new Book

{

Title = bookDTO.Title,

Author = bookDTO.Author,

// سایر ویژگی‌های کتاب

};

}

}

```

در این مثال، ما یک DTO به نام "BookDTO" ایجاد کرده‌ایم که اطلاعات یک کتاب را نمایش می‌دهد. سپس یک سرویس به نام "BookService" ایجاد کرده‌ایم که از ریپازیتوری کتاب استفاده می‌کند تا منطق کسب و کار مربوط به مدیریت کتب را پردازش کند. این سرویس از DTO برای انتقال داده‌ها استفاده می‌کند و اطلاعات کتاب‌ها را بین لایه‌ها منتقل می‌کند.

این مثال نشان دهنده استفاده از DTO و BL در یک پروژه API ASP.NET Core است. این البته یک مثال ساده است و در یک پروژه واقعی، ممکن است نیاز به ساختاربندی و گسترش بیشتری داشته باشید.

اگر پروژه ی من شامل لایه های application core و لایه domain و infrastructure و api باشد مثال بالا را در کدام لایه ها تعریف کنم؟

با توجه به ساختار پروژه‌ای که شما توضیح داده‌اید، می‌توانید مثال بالا را در لایه‌های مختلف به صورت زیر تعریف کنید:

1. لایه Domain:

- مدل داده‌ها (مانند کلاس "Book") و معماری دیتایی (مانند ریپازیتوری) را در این لایه تعریف کنید.

2. لایه Application Core:

- کلاس "BookService" را در این لایه تعریف کنید. این کلاس مسئولیت‌هایی مانند دریافت اطلاعات کتاب، اضافه کردن کتاب جدید، حذف کتاب و تبدیل اطلاعات بین DTO و مدل داده‌ها را دارد. این لایه می‌تواند برای پردازش منطق کسب و کار استفاده شود.

3. لایه Infrastructure:

- ریپازیتوری (مانند "IBookRepository") و هرگونه ارتباط با داده‌ها (مانند پایگاه داده) را در این لایه تعریف کنید. این لایه مسئولیت دسترسی به داده‌ها و ارتباط با منابع خارجی را دارد.

4. لایه API:

- در این لایه، می‌توانید کنترلر‌ها و مدل‌های DTO را تعریف کنید. کنترلر‌ها مسئول دریافت درخواست‌ها از کلاینت‌ها و فراخوانی متدهای مربوط به لایه Application Core و ارسال پاسخ‌ها به کلاینت‌ها هستند. مدل‌های DTO نیز برای انتقال داده‌ها بین سیستم و کلاینت‌ها استفاده می‌شوند.

با این تقسیم بندی، شما قادر خواهید بود تا مسئولیت‌ها و وظایف مختلف را در لایه‌های مختلف پروژه‌تان جدا کنید و به شکلی منطبق با معماری Clean Architecture عمل کنید.

مطمئنا! در اینجا یک مثال ساده از چگونگی تعریف کلاس "BookService" در لایه‌های مختلف پروژه .NET Core را می‌آورم.







این مثال نشان می‌دهد که چگونه می‌توانید کلاس "BookService" و سایر کلاس‌ها را در لایه‌های مختلف پروژه .NET Core تعریف کنید و مسئولیت‌های مختلف را جدا کنید.

در مثال بالا BL کدام است؟

در مثال بالا، لایه `Service` (که به عنوان Business Logic شناخته می‌شود) مسئول انجام منطق کسب و کار و پردازش داده‌ها است. در این مثال، کلاس `BookService` در لایه `Service` قرار دارد و مسئولیت‌های مربوط به کسب و کار مربوط به کتاب‌ها را انجام می‌دهد. به عنوان مثال، این کلاس مسئولیت‌هایی مانند دریافت کتاب با استفاده از شناسه، افزودن کتاب جدید و حذف کتاب را بر عهده دارد.

به عنوان مثال، متد `GetBookById` در کلاس `BookService` مربوط به لایه `Service` است و وظیفه بازیابی کتاب با استفاده از شناسه آن را دارد. این متد می‌تواند منطقی داشته باشد که به عنوان بخشی از Business Logic در این لایه قرار دارد.

بنابراین، کلاس `BookService` و متدهای آن می‌توانند به عنوان Business Logic در این مثال در نظر گرفته شوند.

در مثال بالا چرا از تعریف کلاس "BookService" در لایه Application Core: استفاده کردیم؟

از تعریف کلاس "BookService" در لایه Application Core استفاده کردیم زیرا:

1. اصول معماری Clean Architecture را رعایت می‌کند: لایه Application Core (یا همان Business Logic) باید مسئولیت اصلی انجام منطق کسب و کار را بر عهده داشته باشد. با تعریف کلاس "BookService" در این لایه، ما از اصول Clean Architecture پیروی کرده‌ایم و اطمینان حاصل کرده‌ایم که منطق مربوط به کتب در یک لایه مستقل و قابل تست قرار دارد.

2. افزایش قابلیت استفاده مجدد: با تعریف کلاس "BookService" در لایه Application Core، ما این کلاس را به صورت مستقل از جزئیات فنی دیگر مانند دیتابیس یا فریمورک‌ها پیاده‌سازی کرده‌ایم. این امر باعث افزایش قابلیت استفاده مجدد این کلاس و امکان استفاده آن در بخش‌های دیگر سیستم می‌شود.

3. انطباق با اصول SOLID: با تعریف کلاس "BookService" در لایه Application Core، ما از اصول SOLID پیروی کرده‌ایم. به عنوان مثال، این کلاس مسئولیت‌های مربوط به کتب را بر عهده دارد و از اصل Single Responsibility Principle پیروی می‌کند.

بنابراین، با تعریف کلاس "BookService" در لایه Application Core، ما از اصول Clean Architecture، قابلیت استفاده مجدد و اصول SOLID پیروی کرده‌ایم.

در تعریف ریپازیتوری کتاب در لایه Infrastructure: دقیقا چه چیزی تعریف میشود؟

در لایه Infrastructure، تعریف ریپازیتوری کتاب به عنوان یک واسط بین لایه های بالاتر (مانند( Application Core و جزئیات فنی مانند دیتابیس یا فریمورک‌ها انجام می‌شود. ریپازیتوری کتاب مسئولیت‌هایی مانند ذخیره، بازیابی، حذف و به روزرسانی کتاب‌ها در دیتابیس یا منبع داده مربوطه را بر عهده دارد.

به عنوان مثال، می‌توانیم یک ریپازیتوری برای کتب به شکل زیر تعریف کنیم:



در این مثال، BookRepository یک کلاس است که از واسط IBookRepository ارث بری می‌کند و متدهای مربوط به مدیریت کتاب‌ها را پیاده‌سازی می‌کند. این کلاس از یک شیء DbContext برای ارتباط با دیتابیس استفاده می‌کند و مسئولیت‌های مربوط به ذخیره، بازیابی، حذف و به روزرسانی کتاب‌ها را بر عهده دارد.

با تعریف ریپازیتوری کتاب در لایه Infrastructure، ما از اصول Clean Architecture پیروی می‌کنیم و اطمینان حاصل می‌کنیم که جزئیات فنی مانند دیتابیس و فریمورک‌ها در این لایه قرار دارند و لایه‌های بالاتر از این جزئیات مستقل هستند.

در مثال بالا DTO در کجا تعریف شد که در api مورد استفاده قرار گرفت؟

در مثال بالا، DTO (Data Transfer Object) برای کتاب در لایه Application Core تعریف شده است. DTO ها به عنوان واسطی بین لایه‌های مختلف سیستم استفاده می‌شوند و اطلاعات مربوط به یک موجودیت را برای انتقال بین این لایه‌ها به صورت ساده و مستقل از جزئیات فنی تعریف می‌کنند.

به عنوان مثال، می‌توانیم یک DTO برای کتاب به شکل زیر تعریف کنیم:



سپس، در لایه Application Core، می‌توانیم از این DTO برای انتقال اطلاعات کتاب بین سرویس‌ها و واسط‌های کاربری مانند API استفاده کنیم. به عنوان مثال، می‌توانیم از این DTO برای ارسال اطلاعات کتاب به عنوان پاسخ از یک سرویس در API استفاده کنیم:



در این مثال، ما از DTO برای تبدیل اطلاعات کتاب به یک شیء ساده و مستقل از جزئیات فنی استفاده کرده‌ایم تا این اطلاعات به عنوان پاسخ از API ارسال شود.

بنابراین، DTO در لایه Application Core تعریف شده است و در API برای انتقال اطلاعات بین سرویس‌ها و کلاینت‌ها استفاده شده است.

ممکنه بخش controller مثال بالا را کامل با کد .net بنویسی؟

البته! در زیر یک قطعه کد .NET Core برای بخش کنترلر مربوط به درخواست GET یک کتاب با استفاده از DTO تعریف شده در مثال قبلی آورده شده است.



در این کد، ما یک کنترلر به نام BooksController تعریف کرده‌ایم که از IBookService برای ارتباط با سرویس‌های مربوط به کتاب استفاده می‌کند. در این کنترلر، یک درخواست GET با یک پارامتر id برای دریافت جزئیات یک کتاب تعریف شده است. سپس اطلاعات کتاب از سرویس گرفته شده و به یک شیء BookDto تبدیل شده و به عنوان پاسخ ارسال می‌شود.

لطفاً توجه داشته باشید که نام فضای نام سرویس‌ها و DTO‌ها باید با نام فضای نام واقعی برنامه شما جایگزین شود.

برای رفع خطای "Unable to resolve service for type" که به مشکل در Dependency Injection (DI) مربوط است، می‌توانید مراحل زیر را دنبال کنید:

1. ابتدا، مطمئن شوید که کلاس `StateService` به عنوان یک سرویس تعریف شده است و به کنترلر `StateController` تزریق شده است. برای این کار، باید از یکی از روش‌های تزریق سرویس‌ها در ASP.NET Core مانند `AddScoped`، `AddTransient` یا `AddSingleton` استفاده کنید.

2. اگر کلاس `StateService` به درستی تعریف شده و به کنترلر `StateController` تزریق شده است، مطمئن شوید که namespace و نام کامل کلاس `StateService` درست تعیین شده باشد. اگر namespace یا نام کلاس اشتباه باشد، DI قادر به یافتن و تزریق این سرویس نخواهد بود.

3. همچنین، مطمئن شوید که کلاس `StateService` از رابط `IStateService` پیاده‌سازی شده باشد. این کار به DI کمک می‌کند تا از انتزاع استفاده کند و به جای مستقیم استفاده از کلاس، از رابط استفاده کند.

4. در صورتی که مشکل هنوز حل نشده است، ممکن است بخش‌های دیگری از کد یا تنظیمات DI شما مشکل داشته باشند. برای مثال، ممکن است یک تداخل در تنظیمات DI وجود داشته باشد که باعث ایجاد این خطا شده است.

با انجام این مراحل، مشکل شما باید حل شود و DI برای کلاس `StateService` به صورت صحیح کار خواهد کرد.

علت استفاده از base در

public StateReository(ApiContext apiContext) : base(apiContext)

{

}

استفاده از کلمه کلیدی base در اینجا به منظور فراخوانی کانستراکتور پدر است. در این حالت، این کد به معنای فراخوانی کانستراکتور کلاس پدر است و مقداردهی اولیه این کلاس از طریق فراخوانی کانستراکتور پدر انجام می‌شود.

چرا در قطعه کد زیر از Istaterepository به عنوان ورودی constructor استفاده کردیم؟

public class StateService

{

private readonly IStateRepository \_stateRepository;

public StateService(IStateRepository stateRepository)

{

\_stateRepository = stateRepository;

}

public List<State> GetAllState()

{

return \_stateRepository.GetAll();

}

public State GetStateById(int id)

{

return \_stateRepository.GetById(id);

}

}

در قطعه کد فوق از IStateRepository به عنوان ورودی constructor استفاده شده است به دلیل اینکه از اصول برنامه‌نویسی SOLID و به خصوص اصل Dependency Inversion استفاده شده است. این اصل به ما می‌گوید که باید وابستگی‌های کلاس‌ها به ابسترکشن‌ها و نه به جزئیات بسته‌شونده باشد. با استفاده از این اصل، کلاس StateService به جای وابستگی به یک پیاده‌سازی خاص از StateRepository، به اینترفیس IStateRepository وابسته است. این کار امکان تعویض پیاده‌سازی‌های مختلف از IStateRepository را بدون تغییر در کلاس StateService فراهم می‌کند و انعطاف‌پذیری کد را افزایش می‌دهد. به عبارت دیگر، این اصل به ما می‌گوید که باید از وابستگی به ابسترکشن‌ها (مانند interface) استفاده کنیم و نه از جزئیات (مانند کلاس‌های خاص).

ممکنه بخش های زیر را معنی کنی و علت استفاده از آنها رو بگی؟

"ConnectionStrings": {

"ApiDatabase": "Server = 94.232.171.228;Database=najm;User ID=sa;Password=[]N@h@d1391;TrustserverCertificate=true;MultipleActiveResultSets=True;",

"providerName": "\"System.Data.SqlClient\""

}

این بخش از یک فایل پیکربندی (مثل یک فایل JSON یا فایل تنظیمات) به نظر می‌رسد که اطلاعات مربوط به رشته‌های اتصال به پایگاه داده را در خود دارد. در اینجا، دو رشته اتصال به دو پایگاه داده مختلف ("ApiDatabase" و "providerName") آورده شده است.

"ApiDatabase": "Server = 94.232.171.228;Database=najm;User ID=sa;Password=[]N@h@d1391;TrustserverCertificate=true;MultipleActiveResultSets=True;"

این رشته اتصال شامل اطلاعات مربوط به سرور پایگاه داده، نام پایگاه داده، نام کاربری و رمز عبور است. این اطلاعات برای برقراری اتصال به پایگاه داده مورد استفاده قرار می‌گیرد.

"providerName": "\"System.Data.SqlClient\""

این بخش ممکن است برای تعیین نوع فراهم‌کننده اتصال به پایگاه داده استفاده شود. در اینجا، نوع فراهم‌کننده اتصال به پایگاه داده به عنوان "System.Data.SqlClient" مشخص شده است.

استفاده از این بخش‌ها در یک فایل پیکربندی به ما این امکان را می‌دهد که اطلاعات اتصال به پایگاه داده را به صورت جداگانه و خارج از کد برنامه‌نویسی نگهداری کنیم. این کار امنیت و انعطاف‌پذیری برنامه را افزایش می‌دهد و امکان تغییر اطلاعات اتصال بدون نیاز به تغییر در کد برنامه را فراهم می‌کند.

این یک رشته اتصال به پایگاه داده است که شامل اطلاعات مورد نیاز برای برقراری اتصال به پایگاه داده مورد نظر می‌باشد. این رشته اتصال شامل اطلاعات زیر است:

- Server = 94.232.171.228: آدرس IP یا نام سرور پایگاه داده

- Database = najm: نام پایگاه داده

- User ID = sa: نام کاربری برای ورود به پایگاه داده

- Password = []N@h@d1391: رمز عبور برای ورود به پایگاه داده

- TrustserverCertificate = true: نشان‌دهنده اعتماد به گواهی نامه سرور

- MultipleActiveResultSets = True: فعال‌سازی چندین مجموعه نتایج فعال

این اطلاعات برای برقراری اتصال به پایگاه داده SQL Server استفاده می‌شود.

علت استفاده از [from route] در متد زیر چیست؟

[HttpGet]

[Route("{id}")] //localhost:port/api/village/{id}

public IActionResult GetVillageById([FromRoute] int id)

{

}

استفاده از `[FromRoute]` در متد زیر، برای استخراج مقدار پارامتر از مسیر (Route) در ASP.NET Core استفاده می‌شود. در این حالت، پارامتر `id` از مسیر دریافت می‌شود.

با استفاده از `[FromRoute]`، ASP.NET Core مقدار `id` را از مسیر دریافت می‌کند و به عنوان ورودی برای متد `GetVillageById` استفاده می‌کند. به این ترتیب، وقتی که یک درخواست GET به مسیر مشخص شده (مثلاً `localhost:port/api/village/10`) ارسال می‌شود، مقدار `10` از مسیر استخراج و به عنوان ورودی به متد `GetVillageById` منتقل می‌شود.

به طور کلی، استفاده از `[FromRoute]` به شما این امکان را می‌دهد که پارامتر‌هایی که از مسیر دریافت می‌شوند را به صورت مستقیم از مسیر دریافت کرده و به متد ارسال کنید.

میخواهم با کمک api زمانی که id مربوط به state وارد شود نام townshipهای مرتبط نمایش داده شود.:

برای این کار، شما می‌توانید یک متد دیگر در کنترلر خود ایجاد کنید که با دریافت شناسه ایالت (state id)، تمام townshipهای مربوط به آن را بازگرداند. سپس می‌توانید این متد را از متد مورد نظر خود فراخوانی کنید و نتیجه را به عنوان خروجی ارسال کنید. در زیر یک مثال از چگونگی این کار آورده شده است:



در این مثال، متد GetTownshipsByStateId برای بازگرداندن تمام townshipهای مربوط به یک ایالت ایجاد شده است. سپس در متد GetTownshipsByState، ابتدا ایالت مورد نظر با استفاده از stateId دریافت می‌شود و سپس تمام townshipهای مربوط به آن ایالت به همراه نام ایالت به عنوان خروجی ارسال می‌شود

چرا در کلاس زیر از `[ForeignKey("StateCode")]` استفاده کردیم؟

public class Township

{

[Key]

public int TownshipCode { get; set; }

[StringLength(50)]

public string? TownshipName { get; set; }

public int StateCode { get; set; }

[ForeignKey("StateCode")]

public State State { get; set; }

}

استفاده از `[ForeignKey("StateCode")]` در کلاس `Township` به معنای این است که این پراپرتی `State` به عنوان navigation property به مدل `State` مرتبط است و از طریق کلید خارجی `StateCode` به آن متصل شده است.

در اینجا، `StateCode` به عنوان یک کلید خارجی برای ارتباط با مدل `State` استفاده شده است. با اضافه کردن `[ForeignKey("StateCode")]`، شما به Entity Framework می‌گویید که این پراپرتی `State` با استفاده از `StateCode` به مدل `State` متصل شود.

به این ترتیب، اگر یک `Township` را بخواهید بازیابی کنید، می‌توانید به راحتی از navigation property `State` برای دسترسی به مدل `State` مرتبط استفاده کنید.

اگر `StateCode` یک کلید خارجی به جدول `State` باشد، استفاده از `[ForeignKey("StateCode")]` به Entity Framework کمک می‌کند تا ارتباط بین دو مدل را به درستی تشخیص دهد و از آن استفاده کند.

چرا [HttpGet("getTownshipsByStateId/{stateId}")] از getTownshipsByStateId/{stateId} استفاده میکنیم؟

استفاده از "getTownshipsByStateId/{stateId}" در یک وب سرویس به دلیل استاندارد RESTful بودن است. در استاندارد RESTful، از الگوی URI های معین برای ارتباط با منابع استفاده می شود. در این الگو، از پارامترها برای ارسال اطلاعات به سرور استفاده می شود.

در اینجا، "getTownshipsByStateId" نشان دهنده عملیات دریافت شهرستان ها بر اساس شناسه استان است و "{stateId}" نشان دهنده پارامتری است که مقدار آن برابر با شناسه استان مورد نظر است. با استفاده از این الگو، می توان به راحتی از وب سرویس استفاده کرد و اطلاعات مورد نیاز را با استفاده از شناسه مناسب دریافت کرد.

در زبان برنامه نویسی C# و فریمورک ASP.NET Core، attribute ها برای تنظیم و تعریف رفتارهای مختلف برای کلاس ها و متد ها استفاده می شوند. در مثالی که ارائه دادید، از attribute هایی برای تنظیم مسیر و نوع درخواست HTTP برای یک متد استفاده شده است. در اینجا، توضیحات بیشتری در مورد این attribute ها ارائه می شود:

1. `[HttpGet("getTownshipsByStateId/{stateId}")]`: این attribute برای تعریف یک متد به عنوان یک متد درخواست HTTP GET استفاده می شود. در اینجا، مسیر متد تعیین شده است که به شکل "getTownshipsByStateId/{stateId}" است. این مسیر به عنوان یک الگو برای ارسال درخواست به سرور استفاده می شود و با ارسال شناسه استان به عنوان پارامتر در URL، اطلاعات مربوط به شهرستان های آن استان را دریافت می کنیم.

این attribute ها به توسعه دهنده امکان می دهند تا رفتارهای مختلفی را برای متدها و کنترلرها تعریف کنند و به راحتی واکنشگرا responsive برنامه نویسی کنند. این attribute ها به طور گسترده ای در ASP.NET Core برای تعریف و تنظیم ویژگی های مختلف وب سرویس ها و کنترلرها استفاده می شوند.

فایل swagger.json یک فایل تعریف API است که شامل تمام اطلاعات مربوط به API شما است، از جمله مسیرها، پارامترها، مدل‌ها و توضیحات. این فایل معمولاً توسط کتابخانه Swashbuckle.AspNetCore در ASP.NET Core به صورت خودکار تولید می‌شود.

به طور پیش‌فرض، فایل swagger.json در پوشه /swagger در پروژه ASP.NET Core شما قرار دارد. اگر از تنظیمات پیش‌فرض Swashbuckle.AspNetCore استفاده می‌کنید، فایل swagger.json از طریق URL /swagger/v1/swagger.json در دسترس است.

برای اعمال تغییرات در فایل swagger.json، می‌توانید از ویژگی‌های کتابخانه Swashbuckle.AspNetCore مانند SwaggerGenOptions استفاده کنید. با استفاده از این ویژگی‌ها، می‌توانید تنظیمات مختلفی را برای تولید فایل swagger.json تعیین کنید، از جمله اضافه کردن توضیحات، تعیین نام ورژن API و موارد دیگر.

به عنوان مثال، برای اضافه کردن یک توضیح به یک مسیر در فایل swagger.json، می‌توانید از ویژگی OpenApiInfo استفاده کنید و توضیحات مورد نظر خود را اضافه کنید.

برای اطلاعات بیشتر در مورد تنظیمات و تغییرات در فایل swagger.json با استفاده از Swashbuckle.AspNetCore، می‌توانید به مستندات رسمی Swashbuckle مراجعه کنید: <https://github.com/domaindrivendev/Swashbuckle.AspNetCore>

# **اموزش QueryString ها در asp.net - انتقال مقادیر بین صفحات در Asp.Net**

کوئری استرینگ ها همان استخراج اطلاعات ارسال شدهبه همرا لینک میباشد  
حتما شما لینک هایی به صورت زیر دیده اید  
www.example.com/page.aspx?id=2&name=ali  
این لینک علاوه بر ادرس شمال یکسری اطلاعات که عبارتند از ایدی و نام میباشند  
موضوع بحث این مقاله در این رابطه است که این مقادیر چگونه استخراج و استفاده مشوند.  
  
اگر دقت کرده باشید ادرس همین مقاله ای که میخوانید دارای کوئری استرینگ میباشد که ادرس به این شکل است  
[www.hypersource.ir/post.aspx?id=58](http://www.hypersource.ir/post.aspx?id=58)  
این بیانگر این موضوع است که مقاله ای که دارای ایدی 58 است را در این صفحه به نمایش در اورده ام  
یعنی اگر شما مقدار 58 را تغییر دهید به پست و مقاله ای دیگر هدایت خواهید شد  
مانند لینک های زیر :  
[www.hypersource.ir/post.aspx?id=57](http://www.hypersource.ir/post.aspx?id=57) استفاده از ای جکس در asp.net و php  
[www.hypersource.ir/post.aspx?id=14](http://www.hypersource.ir/post.aspx?id=14) اموزش تصویری جاا اسکریپت به زبان فارسی  
و...  
فرض کنید ما میخواهیم با کلیک بر روی یک لینک نام و سن یک فرد را به صفحه ای دیگر انتقال دهیم  
Name : Parsa  
Age : 22  
به این صورت ما لینکی به این حالت ایجاد میکنیم

<Send User </a> href="www.example.com/users.aspx?Name=parsa&Age=22" target="\_blank"

نکته : اطلاعات ارسالی مختلف در لینک را توسط کارکتر '&' از هم جدا میکنند

بعد از ارسال اگر ما بخواهیم این اطلاعات در صفحه user.aspx دریافت کنیم به صورت زیر عمل میکنیم

|  |  |
| --- | --- |
| 1 | **protected** **void** Page\_Load(...){ |

|  |  |
| --- | --- |
| 2 | **string** Name = Request.QueryString["Name"].ToString(); |

|  |  |
| --- | --- |
| 3 | **string** Age = Request.QueryString["Age"].ToString(); |

|  |  |
| --- | --- |
| 4 |  |
| 5 | **this**.Title = "Welcome To My Site "+ Name ; |

نکته : روش کوئری اترینگ یک روش خوب برای انتقال اطلاعات بین صفحات وب میباشد ولی به هیچ وجه از ان برای انتقال اطلاعات مهم و ضروری استفاده نکنید

برای انتقال اطلاعات مهم بین صفحات از Session که دارای امنیت بهمراتب بالاتر از اینهاست استفاده میشود

# **کار با Session ها در Asp.Net - انتقال مقادیر بین صفحات در Asp.Net**

در پست قبلی با انتقال مقادیر بین صفحات از طریق [QueryString](http://www.hypersource.ir/post.aspx?id=58" \t "_blank" \o "اموزش کار با QueryString ها در Asp.Net)اشنا شدید  
ولی همانطور که گفتیم کوئری استرینگ برای انتقال مقادیر ضروری و مهم دارای امنیت کافی نیستند  
همچنین گفتیم که برای انتقال مقادیر بین صفحات با **امنیت بالا** از [Session](http://www.hypersource.ir/post.aspx?id=59)استفاده می کنند  
  
این نکته را به یاد داشته باشید که نباید زیاد از Session ها استفاده کرد و از ان فقط برای اطلاعات مهم استفاده کنید زیرا این امر موجب کاهش سرعت بارگذاری صفحات میشود  
  
برعکس عنوانی که دارند کار با Session ها بسیار اسان است  
نحوه تعریف یک Session به این صورت است

|  |  |
| --- | --- |
| 1 | Session.Add(SessionName,SessionValue); |

برای مثال فرض کنیم ما میخواهیم یوزر نیم ادمین را در Session ذخیره و در موارد نیاز ان را لود کنیم  
 **نکته مهم : Session ها ابزار قدرتمندی هستند و میتوانند حتی یک دیتاتیبل و یا حتی DataSet را در خود ذخیره نمایند**

|  |  |
| --- | --- |
| 1 | Session.Add("User\_Admin","Admin\_5411"); |

برای تعیین انقضا برای Session ها از عبارت زیر استفاده میشود که یک مقدار صحیح بر اساس دقیقه است

|  |  |
| --- | --- |
| 1 | Session.TimeOut = 60; |

یعنی انقضای این Session 60 دقیقه یا یک ساعت است  
برای استفاده از مقدار ذخیره شده در Session ها به این صورت عمل میکنند

|  |  |
| --- | --- |
| 1 | **string** UserName = Session["User\_Admin"].ToString() |

|  |  |
| --- | --- |
| 2 | //و یا |

|  |  |
| --- | --- |
| 3 | **string** UserName = (**string**)Session["User\_Admin"]; |

استفاده میشود  
در نهایت برای پاک کردن یک Session از عبارت زیر استفاده میشود

|  |  |
| --- | --- |
| 1 | Session.Remove("User\_Admin"); |

معنی خط زیر چیست؟

onvan = onvan ?? "";

این خط یک عملگر null-coalescing است که مقدار onvan را برابر با خودش قرار می‌دهد مگر اینکه onvan تهی یا null باشد. در این صورت، مقدار onvan را برابر با یک رشته خالی قرار می‌دهد. این کار می‌تواند جلوی خطاهای ناشی از استفاده از متغیرهای تهی یا null را بگیرد.

null-coalescing یک عملگر در زبان‌های برنامه‌نویسی است که مقدار یک عبارت را برمی‌گرداند مگر اینکه عبارت تهی یا null باشد. در این صورت، مقدار عبارت دیگری را برمی‌گرداند. [به عنوان مثال، در C# عملگر ?? این کار را انجام می‌دهد](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-coalescing-operator" \t "_blank)[1](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-coalescing-operator)[2](https://stackoverflow.com/questions/770186/null-coalescing-operator-what-does-coalescing-mean)[3](https://en.wikipedia.org/wiki/Null_coalescing_operator)

در این خط کد، از عبارت ?? برای بررسی و تنظیم مقدار متغیر استفاده شده است. این عبارت بررسی می‌کند که مقدار متغیر onvan خالی یا null باشد یا خیر. اگر مقدار آن خالی یا null باشد، مقدار آن به رشته‌ی خالی تنظیم می‌شود. اما اگر مقدار متغیر onvan خالی یا null نباشد، مقدار اولیه آن بدون تغییر باقی می‌ماند. این کار برای اطمینان از اینکه مقدار متغیر onvan از نوع رشته و همچنین تهی نباشد، انجام می‌شود.

عملگر null-coalescing ?? مقدار عملوند سمت چپ خود را برمی‌گرداند اگر null نباشد؛ در غیر این صورت، عملوند سمت راست خود را ارزیابی می‌کند و نتیجه آن را برمی‌گرداند. عملگر ?? عملوند سمت راست خود را ارزیابی نمی‌کند اگر عملوند سمت چپ خود null نباشد. عملگر اختصاص null-coalescing ??= مقدار عملوند سمت راست خود را به عملوند سمت چپ خود اختصاص می‌دهد فقط اگر عملوند سمت چپ خود null باشد. عملگر ??= عملوند سمت راست خود را ارزیابی نمی‌کند اگر عملوند سمت چپ خود null نباشد.

معنی تعریف این کلاس چیه؟

public class drpStuff

{

public string Text { get; set; }

public string Value { get; set; }

}

این کلاس یک ساختار داده‌ای است که دو خصوصیت از نوع رشته دارد: Text و Value. این کلاس می‌تواند برای نمایش اطلاعات در یک کنترل کشویی (dropdown) استفاده شود. Text نشان دهندهٔ متنی است که کاربر می‌بیند و Value نشان دهندهٔ مقداری است که با آن متن مرتبط است. به عنوان مثال، اگر یک کنترل کشویی داشته باشیم که گزینه‌های زیر را نشان می‌دهد:

* ایران
* ترکیه
* چین

می‌توانیم از کلاس drpStuff برای نگهداری اطلاعات مربوط به هر گزینه استفاده کنیم. برای این کار، باید سه شیء از کلاس drpStuff ایجاد کنیم و مقادیر Text و Value آن‌ها را مشخص کنیم. به عنوان مثال:

drpStuff iran = new drpStuff();

iran.Text = "ایران";

iran.Value = "IR";

drpStuff turkey = new drpStuff();

turkey.Text = "ترکیه";

turkey.Value = "TR";

drpStuff china = new drpStuff();

china.Text = "چین";

china.Value = "CN";

سپس می‌توانیم این شیء‌ها را به یک لیست اضافه کنیم و آن را به کنترل کشویی متصل کنیم. [به این ترتیب، هر وقت کاربر یک گزینه را انتخاب کند، می‌توانیم مقدار Value مربوط به آن گزینه را بدست آوریم](https://abadis.ir/entofa/class/" \t "_blank)[1](https://abadis.ir/entofa/class/)[2](https://abadis.ir/)

معنی این خط JObject finalResult;

این خط یک تعریف متغیر است که یک شیء از نوع JObject را نشان می‌دهد. JObject یک کلاس در کتابخانهٔ Json.NET است که برای کار با داده‌های JSON در زبان C# استفاده می‌شود. JSON یک فرمت متنی است که برای انتقال و ذخیره‌سازی داده‌های ساختاریافته استفاده می‌شود. [finalResult نام متغیر است که می‌تواند هر مقداری از نوع JObject را در خود نگه دارد](https://abadis.ir/translator/fa-en/" \t "_blank)[1](https://abadis.ir/translator/fa-en/)[2](https://abadis.ir/)[3](https://www.parsi.wiki/fa/wiki/235648/%D8%AE%D8%B7)

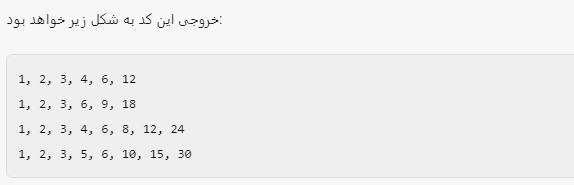
این کد در .net زده شده است. معنی select چیست؟

if (!DrpSearchType.Select(a => a.Value).Contains(searchType)) { }

این کد در .NET نوشته شده است. معنی select این است که هر عنصر از یک دنباله را به یک شکل جدید تبدیل می‌کند. برای مثال، در کد شما، select همه مقادیر DrpSearchType را به یک دنباله از مقادیر تبدیل می‌کند و سپس بررسی می‌کند که آیا searchType در آن دنباله وجود دارد یا نه. برای اطلاعات بیشتر در مورد select می‌توانید به [این صفحه](https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.select?view=net-8.0) یا [این صفحه](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/select-clause) مراجعه کنید. 🙏

select یک متد از کلاس Enumerable است که بر روی دنباله‌هایی از نوع IEnumerable<T> قابل استفاده است. این متد یک تابع به عنوان پارامتر می‌گیرد و برای هر عنصر از دنباله، آن تابع را اجرا می‌کند و نتیجه را در یک دنباله جدید قرار می‌دهد. بنابراین، select یک روش برای تبدیل یا تغییر شکل داده‌های موجود است. برای مثال، اگر بخواهیم از یک دنباله از اعداد صحیح، یک دنباله از رشته‌هایی که نشان‌دهنده مقسوم‌علیه‌های آن اعداد هستند، بسازیم، می‌توانیم از select استفاده کنیم. کد زیر این کار را انجام می‌دهد:





public class BP\_modelForSearch

{

public int PageIndex { get; set; }

public int PageSize { get; set; }

public int Language { get; set; }

public int MaterialID { get; set; }

public string order { get; set; }

public int OrgIdType { get; set; }

public int SearchType { get; set; }

}

این کد یک **کلاس** به نام BP\_modelForSearch را تعریف می‌کند. این کلاس دارای چند **ویژگی (Property)** است که هر کدام نوع داده‌ای خاص خود را دارند:

1. PageIndex: یک عدد صحیح که نمایانگر شماره صفحه‌ای است که در جستجوی مواد مورد نظر استفاده می‌شود.
2. PageSize: یک عدد صحیح که تعداد مواد نمایش داده شده در هر صفحه را مشخص می‌کند.
3. Language: یک عدد صحیح که نمایانگر زبان مورد استفاده برای جستجو است.
4. MaterialID: یک عدد صحیح که شناسه‌ی ماده‌ی مورد نظر را نمایش می‌دهد.
5. order: یک رشته که نمایانگر ترتیب مواد در نتایج جستجو است.
6. OrgIdType: یک عدد صحیح که نمایانگر نوع شناسه‌ی سازمان مورد نظر است.
7. SearchType: یک عدد صحیح که نمایانگر نوع جستجو (مثلاً جستجوی متنی یا جستجوی شناسه) است.

این کلاس به نظر می‌آید برای انجام عملیات جستجو در یک پروژه‌ی برنامه‌نویسی مورد استفاده قرار می‌گیرد.

public async Task<HttpResponseMessage> bp\_fehrestSearch(string OrgIdType, int? Page, int? Pagesize, string order, string searchType, string MaterialID, string Language, string onvan, string padidAvar, string nasher, string tarikhNashr, string mahalNashr, string RecordNumber){}

این کد یک تابع به نام bp\_fehrestSearch را تعریف می‌کند. این تابع با استفاده از پارامترهای ورودی، جستجوی مواد مورد نظر را انجام می‌دهد. به طور خلاصه:

* OrgIdType: نوع شناسه‌ی سازمان مورد نظر.
* Page و Pagesize: شماره صفحه و تعداد مواد نمایش داده شده در هر صفحه.
* order: ترتیب مواد در نتایج جستجو.
* searchType: نوع جستجو (مثلاً جستجوی متنی یا جستجوی شناسه).
* MaterialID: شناسه‌ی ماده‌ی مورد نظر.
* Language: زبان مورد استفاده برای جستجو.
* onvan، padidAvar، nasher، tarikhNashr، mahalNashr، RecordNumber: اطلاعات مرتبط با ماده‌ی مورد نظر.

این تابع به نظر می‌آید برای جستجو در پروژه‌های برنامه‌نویسی مورد استفاده قرار می‌گیرد.

علت استفاده از Ilist در کد زیر :

IList<BP\_FehrestSearchResults> resultData = await

استفاده از `IList` در کد زیر به دلیل این است که `IList` یک رابط ژنریک است که اجازه می‌دهد تا یک لیست از عناصر مشخص را نگهداری کند. در این حالت، `resultData` یک لیست از نتایج جستجوی `BP\_FehrestSearchResults` است که با استفاده از `await` به صورت همزمان و از طریق `async` از یک عملیات ناهمگام دریافت می‌شود. این روش از `IList` به دلیل امکان استفاده از متدهای مفیدی برای مدیریت و دسترسی به عناصر لیست استفاده می‌شود.

استفاده از `IList` به جای `List` به دلیل انعطاف‌پذیری بیشتر است. زیرا `IList` یک رابط است که تمام ویژگی‌های یک لیست را ارائه می‌دهد اما به عنوان یک رابط عمل می‌کند و از این رو می‌توان از طریق آن به راحتی از توابع دیگری که از `IList` ارث‌بری می‌کنند نیز استفاده کرد. از طرفی `List` یک کلاس کانکرت است و به طور مستقیم از `IList` ارث‌بری می‌کند. اما اگر نیاز به انعطاف بیشتر و استفاده از روش‌های عملیاتی بر روی لیست دارید، استفاده از `IList` مناسب‌تر است.

در .net

#region CreatingWhereClause

string WhereClause = ""

چه کاری انجام میده؟

در .NET، `WhereClause` معمولاً به عنوان یک رشته استفاده می‌شود که شامل شرط‌هایی برای فیلتر کردن داده‌ها است. این رشته معمولاً برای ایجاد یک شرط WHERE در یک کوئری SQL یا LINQ استفاده می‌شود.

به عنوان مثال، در یک کوئری SQL، شما می‌توانید یک `WhereClause` بسازید که شامل شرط‌هایی برای فیلتر کردن رکوردها باشد. به صورت ساده، `WhereClause` ممکن است شامل یک عبارت شرطی مانند `WHERE Age > 18 AND Gender = 'Male'` باشد.

در LINQ نیز می‌توانید از `WhereClause` برای اعمال شرایط فیلتر روی یک لیست یا داده‌های دیگر استفاده کنید. به عنوان مثال:

```csharp

var filteredList = originalList.Where(item => item.Age > 18 && item.Gender == "Male").ToList();

```

در این مثال، `WhereClause` به صورت لمبدا (lambda expression) برای فیلتر کردن `originalList` بر اساس شرایط مشخص استفاده شده است.

بنابراین، `WhereClause` معمولاً برای تعیین شرایط فیلتر در کوئری‌ها یا لیست‌ها استفاده می‌شود.

چرا باید از SqlParameter(); در متدزیر استفاده کنیم؟

var result1 = new List<BP\_FehrestSearchResults>();

try

{

var outParam = new SqlParameter();

outParam.ParameterName = "RecordCount";

outParam.SqlDbType = SqlDbType.Int;

outParam.Direction = ParameterDirection.Output;

result1 = await

db.Database.SqlQuery<BP\_FehrestSearchResults>("sp\_FSearch\_BlindPeople1………. …..)

استفاده از `SqlParameter()` در متد بالا به دلیل مزایای زیر است:

1. \*\*پیشگیری از حملات SQL Injection\*\*:

- با استفاده از `SqlParameter()`، مقادیر پارامترها به صورت پیش‌فرض به‌طور امن به کوئری اضافه می‌شوند. این کار از حملات SQL Injection جلوگیری می‌کند.

- در صورت استفاده از روش‌های دیگر مانند ترکیب مستقیم مقادیر در کوئری، امکان نفوذ به دیتابیس وجود دارد.

2. \*\*پارامترایزه کردن کوئری‌ها\*\*:

- با استفاده از `SqlParameter()`، می‌توانید پارامترهای مختلف را به کوئری اضافه کنید.

- این کار به شما امکان می‌دهد کوئری‌ها را پارامترایزه کنید و مقادیر آن‌ها را در زمان اجرا تعیین کنید.

3. \*\*کد قابل‌خوانی‌تر\*\*:

- استفاده از `SqlParameter()` باعث می‌شود کد شما قابل‌خوانی‌تر و قابل‌نگهداری‌تر باشد.

- پارامترها به صورت مناسب در کد قرار می‌گیرند و کد بهتر سازماندهی می‌شود.

به طور خلاصه، استفاده از `SqlParameter()` بهترین روش برای ایمن‌سازی کوئری‌ها و بهبود قابلیت‌های کد نویسی است .

نمونه sp برای جستجوی api

CREATE PROCEDURE BP\_FehrestSearch

@BpModel int,

@Onvan nvarchar(50),

@PadidAvar nvarchar(50),

@Nasher nvarchar(50),

@TarikhNashr nvarchar(50),

@MahalNashr nvarchar(50),

@RecordNumber nvarchar(50)

AS

BEGIN

DECLARE @sqlQuery nvarchar(max);

SET @sqlQuery = 'SELECT \* FROM BP\_FehrestSearchResults WHERE 1 = 1';

IF @BpModel IS NOT NULL

BEGIN

SET @sqlQuery = @sqlQuery + ' AND BpModel = ' + CAST(@BpModel as nvarchar(10));

END

IF @Onvan IS NOT NULL

BEGIN

SET @sqlQuery = @sqlQuery + ' AND Onvan = ''' + @Onvan + '''';

END

IF @PadidAvar IS NOT NULL

BEGIN

SET @sqlQuery = @sqlQuery + ' AND PadidAvar = ''' + @PadidAvar + '''';

END

IF @Nasher IS NOT NULL

BEGIN

SET @sqlQuery = @sqlQuery + ' AND Nasher = ''' + @Nasher + '''';

END

IF @TarikhNashr IS NOT NULL

BEGIN

SET @sqlQuery = @sqlQuery + ' AND TarikhNashr = ''' + @TarikhNashr + '''';

END

IF @MahalNashr IS NOT NULL

BEGIN

SET @sqlQuery = @sqlQuery + ' AND MahalNashr = ''' + @MahalNashr + '''';

END

IF @RecordNumber IS NOT NULL

BEGIN

SET @sqlQuery = @sqlQuery + ' AND RecordNumber = ''' + @RecordNumber + '''';

END

EXEC sp\_executesql @sqlQuery;

END;

در این stored procedure، ابتدا یک متغیر @sqlQuery تعریف شده و سپس با توجه به ورودی‌های مختلف، یک کوئری SQL برای جستجو در جدول BP\_FehrestSearchResults ایجاد می‌شود. سپس این کوئری با استفاده از sp\_executesql اجرا می‌شود.

حالا می‌توانید از این stored procedure در کد C# خود به صورت زیر استفاده کنید:

public List<BP\_FehrestSearchResults> BP\_FehrestSearch(Bp\_ModelForSearch bp\_Model, string onvan, string padidAvar, string nasher, string tarikhNashr, string mahalNashr, string RecordNumber)

{

var result = \_apiCoetext.BP\_FehrestSearchResults

.FromSqlInterpolated($"EXEC BP\_FehrestSearch {bp\_Model}, {onvan}, {padidAvar}, {nasher}, {tarikhNashr}, {mahalNashr}, {RecordNumber}")

.ToList();

return result;

}

جایگزین SqlParameter در .net core 7

در ASP.NET Core 7، کلاس SqlParameter که در Entity Framework Core برای ایجاد پارامترها استفاده می‌شد، از رده خارج شده و به جای آن از RelationalParameter استفاده می‌شود. RelationalParameter یک کلاس انتزاعی است که بیشتر به عنوان DbParameter در Entity Framework Core 7 شناخته می‌شود.

برای استفاده از RelationalParameter در Entity Framework Core 7 برای تزریق پارامترها، می‌توانید از متد FromSqlInterpolated یا FromSqlRaw به همراه استفاده از RelationalParameter استفاده کنید. در ادامه یک نمونه کد برای استفاده از RelationalParameter آورده شده است:

exec sp\_executesql N'sp\_FSearch\_BlindPeople1 @p0, @p1 ,@p2 ,@p3 ,@p4 ,@p5,@p6 OUT

',N'@p0 nvarchar(4000),@p1 int,@p2 int,@p3 int,@p4 int,@p5 nvarchar(4000),@p6 bigint',@p0=N' and title like N''زراعت'' ',@p1=1,@p2=10,@p3=33,@p4=1,@p5=N'title asc',@p6=1

exec sp\_executesql N'sp\_FSearch\_BlindPeople1 @p0, @p1 ,@p2 ,@p3 ,@p4 ,@p5,@p6 OUT

',N'@p0 nvarchar(4000),@p1 int,@p2 int,@p3 int,@p4 int,@p5 nvarchar(4000),@p6 bigint',@p0=N' and title like N''زراعت'' ',@p1=1,@p2=10,@p3=33,@p4=1,@p5=N'title asc',@p6=1

select

DISTINCT

[Mark2].[dbo].[BlockRecord].[RecordNumber] as RecordNumber,

catalog.creator as padidavarande,

catalog.title as onvanAsli,

catalog.publishDate as saleNashr,

catalog.publisher as nasher,

catalog.publishloc as mahalNashr,

catalog.translator as code,RecordCount = count(1) over()

from

[Mark2].[dbo].[BlockRecord] with(nolock)

inner JOIN [Mark2].[dbo].[B101\_a] with(nolock) ON [Mark2].[dbo].[BlockRecord].[RecordNumber]= [Mark2].[dbo].[B101\_a].[RecordNumber]

left join catalog on [Mark2].[dbo].[BlockRecord].RecordNumber = catalog.RecordNumber

where [Mark2].[dbo].[BlockRecord].MaterialID = 1 and catalog.title like N'زراعت'

and [Mark2].[dbo].[B101\_a].a = 33

دستور "SET NOCOUNT ON" در SQL به معنای غیرفعال کردن شمارش تعداد رکوردهای تحت تاثیر دستورات DML (Data Manipulation Language) می‌باشد. وقتی این دستور فعال است، SQL Server تعداد رکوردهای تحت تاثیر دستورات SELECT، INSERT، UPDATE و DELETE را به عنوان پیام برنمی‌گرداند که می‌تواند به بهبود عملکرد برنامه و کاهش ترافیک شبکه کمک کند.

## Stored procedures in EF 7

We can divide methods to execute stored procedures in EF into two groups, methods to query the data and methods to add/update/delete data.

If we want to query the data, we will use FromSql, FromSqlInterpolated, or FromSqlRaw methods that are available only on the DbSet object.

To add, update or remove records, we need to use ExecuteSql, ExecuteSqlInterpolated, or ExecuteSqlRaw methods, which are available for DbContext.Database object.

تفاوت اصلی بین دو گروه متد برای اجرای stored procedures در Entity Framework در این است که گروه اول برای استعلام داده‌ها (Query) و گروه دوم برای اضافه کردن/به‌روزرسانی/حذف داده‌ها (CRUD operations) استفاده می‌شود.

1. روش‌های استعلام داده: این روش‌ها برای اجرای فرآیندهای ذخیره‌شده که تنها اطلاعات را بازیابی می‌کنند و تغییری در پایگاه داده انجام نمی‌دهند، استفاده می‌شوند. به عبارت دیگر، این روش‌ها برای خواندن اطلاعات از پایگاه داده استفاده می‌شوند بدون اینکه اطلاعاتی تغییر کنند.

2. روش‌های اضافه کردن/به‌روزرسانی/حذف داده: این روش‌ها برای اجرای فرآیندهای ذخیره‌شده که منجر به اضافه کردن، به‌روزرسانی یا حذف اطلاعات در پایگاه داده می‌شوند، استفاده می‌شوند. به عبارت دیگر، این روش‌ها برای تغییر داده‌ها در پایگاه داده استفاده می‌شوند.

به طور خلاصه، روش‌های استعلام داده برای خواندن اطلاعات بدون تغییر، و روش‌های اضافه کردن/به‌روزرسانی/حذف داده برای تغییر داده‌ها در پایگاه داده استفاده می‌شوند.

USE [StateProjectDb.ConsoleApp.NewDb]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[FindstatesAlt] Script Date: 3/5/2024 2:04:20 PM \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

-- =============================================

-- Author: Taherzadeh

-- Create date:

-- Description:

-- =============================================

ALTER PROCEDURE [dbo].[FindstatesAlt]

@SearchFor nvarchar(50)

AS

BEGIN

SET NOCOUNT ON;

-- Insert statements for procedure here

SELECT s.OrgName , p.Name

FROM States S

LEFT JOIN Parts P

on s.OrgId = p.OrgId

WHERE S.OrgName LIKE '%' + @SearchFor + '%'

END

USE [StateProjectDb.ConsoleApp.NewDb]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[FindStates] Script Date: 3/5/2024 2:08:02 PM \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[FindStates]

@SearchFor nvarchar(50)

as

begin

set NOCOUNT on;

select \* from States where [orgId] LIKE '%' + @SearchFor + '%'

end

# **JSON Web Token Authentication in RESTful API .NET Core 7.0**

نمای کلی

تولید توکن: توکن‌های JWT توسط سرور ایجاد و امضا می‌شوند. این توکن شامل ادعاها (جفت‌های کلید-مقدار) است که اطلاعاتی درباره کاربر، نقش‌ها، مجوزها و غیره ارائه می‌دهد. این توکن معمولاً با یک کلید مخفی یا یک کلید خصوصی امضا می‌شود اگر از رمزنگاری نامتقارن استفاده شود.

**Token Structure**

ساختار توکن: یک توکن JWT از سه بخش تشکیل شده است:

* header
* payload (claims), and
* signature

هدر

بار (ادعاها)، و

امضا

این بخش‌ها به صورت base64 رمزگذاری شده و با نقطه‌ها ادغام می‌شوند. هدر معمولاً شامل نوع توکن ("typ": "JWT") و الگوریتم امضا استفاده شده ("alg": "HS256" برای HMAC SHA-256) است. بار حاوی ادعاها است که ویژگی‌ها و حقوق دسترسی کاربر را تعریف می‌کند.

**Token Usage**: استفاده از توکن: توکن پس از احراز هویت موفق به مشتری ارسال می‌شود. مشتری توکن را در هدر مجوزهای درخواست‌های بعدی به عنوان یک توکن Bearer ("Authorization: Bearer ") ارسال می‌کند. سرور امضای توکن را تأیید کرده و ادعاهای آن را بررسی می‌کند تا دسترسی کاربر به منابع خاص را تأیید کند.

**Token Validation** اعتبارسنجی توکن: سرور با استفاده از همان کلید مخفی یا کلید عمومی استفاده شده برای امضای توکن، توکن را اعتبارسنجی می‌کند. سرور امضای توکن، انقضاء (ادعای exp) و صادر کننده (ادعای iss) را بین ادعاهای دیگر بررسی می‌کند. اختلاف ساعت ممکن است برای در نظر گرفتن تفاوت‌ها بین زمان‌های ساخت سرور و توکن استفاده شود.

**Middleware and Libraries**میان‌افزار و کتابخانه‌ها: ASP.NET Core میان‌افزار (Microsoft.AspNetCore.Authentication.JwtBearer) را برای مدیریت احراز هویت JWT فراهم می‌کند. شما میان‌افزار را با تنظیمات و اعتبارسنجی‌های لازم پیکربندی می‌کنید.

**Benefits of JWT** مزایای JWT: بی‌حالت: نیازی به ذخیره توکن‌ها در سمت سرور نیست که آن را قابل مقیاس کردن می‌کند. خودنگهدار: تمام اطلاعات لازم در خود توکن قرار دارد. پشتیبانی گسترده: JWT یک فرمت استاندارد استفاده شده در پلتفرم‌های مختلف.

ملاحظات امنیتی: رمزها را به صورت امن ذخیره کنید و در برابر نشت توکن محافظت کنید. از HTTPS برای رمزنگاری ارتباط استفاده کنید. مقدار اطلاعات حساس موجود در توکن را محدود کنید. در زیر یک مثال سطح بالایی از نحوه پیکربندی احراز هویت JWT در ASP.NET Core آورده شده است:

در دامنه‌ی API، من آماده‌ام تا مدل اجازه‌دهی خود را تزریق کنم، لایه‌ی قدرتمندی از کنترل. به‌طور قابل توجه، برای این پروژه، من یک Entity کاربر ساده معرفی کرده‌ام. ارزشمند است که در یک پروژه واقعی، Entity‌های احراز هویت AspNetUser به‌طور طبیعی به چشم می‌خورند.

پس از اجرای یک API RESTful به‌طور پیوسته، مسیر من منجر به معرفی یک پروژه ClassLibrary می‌شود که به ایجاد پروژه لایه‌ی داده می‌پردازد. در این دامنه، کانوس با Entity‌ها و Repositories تزئین می‌شود که به‌طور آرام و با UnitOfWorkFilter کلاس، یک جزء که اهمیت آن را در یک مقاله آتی شفاف خواهم کرد، تکمیل می‌شود. لمس نهایی Context است که بافت این سمفونی محور داده‌ای را با هم پیوند می‌زند. در پروژه JWTAuth API، همانطور که در تصویر زیر به خوبی نشان داده شده است، من به‌طور استراتژیک AuthorizationContext را گنجانده‌ام. این افزودنی به منظور فعال‌سازی ایجاد کاربر از API به پایگاه داده، با استفاده از تکنیک کد اول ارزشمند استفاده شده است.

تولید توکن: توکن‌های JWT توسط سرور ایجاد و امضا می‌شوند. توکن شامل ادعاها (جفت‌های کلید-مقدار) است که اطلاعاتی درباره‌ی کاربر، نقش‌ها، دسترسی‌ها و موارد دیگر فراهم می‌کنند. توکن به طور معمول با یک کلید مخفی یا یک کلید خصوصی در صورت استفاده از رمزنگاری نامتقارن امضا می‌شود.

ساختار توکن: یک توکن JWT از سه بخش تشکیل شده است:

هدر،

بار (ادعاها)، و

امضا

این بخش‌ها با نقطه‌ها به هم پیوسته و base64 کدگذاری می‌شوند. هدر به طور معمول شامل نوع توکن ("typ": "JWT") و الگوریتم امضای استفاده شده ("alg": "HS256" برای HMAC SHA-256) است. بار ادعاها را شامل می‌شود که ویژگی‌های کاربر و حقوق دسترسی را تعریف می‌کنند.

استفاده از توکن: توکن پس از احراز هویت موفق به کاربر ارسال می‌شود. کاربر توکن را در هدر احراز هویت درخواست‌های بعدی به عنوان یک توکن Bearer ("Authorization: Bearer ") ارسال می‌کند. سرور امضای توکن را تأیید کرده و ادعاهای آن را بررسی می‌کند تا دسترسی کاربر به منابع خاص را تأیید کند.

اعتبارسنجی توکن: سرور توکن را با استفاده از همان کلید مخفی یا کلید عمومی استفاده شده برای امضا تأیید می‌کند. سرور امضای توکن، انقضاء (ادعای exp) و صادرکننده (ادعای iss) را بررسی می‌کند به علاوه ادعاهای دیگر. انحراف ساعت ممکن است برای در نظر گرفتن تفاوت‌های بین زمان‌های ایجاد سرور و توکن استفاده شود.

میان افزار و کتابخانه‌ها: ASP.NET Core میان افزار (Microsoft.AspNetCore.Authentication.JwtBearer) را برای رسیدگی به احراز هویت JWT فراهم می‌کند. شما میان افزار را با تنظیمات و اعتبارسنجی‌های لازم پیکربندی می‌کنید.

مزایای JWT: بدون وضعیت: نیازی به ذخیره توکن‌ها در سمت سرور نیست که آن را قابل مقیاس کردن می‌کند. خودمحتوا: تمام اطلاعات لازم در توکن خودشان قرار دارد. پشتیبانی گسترده: JWT یک فرمت استاندارد استفاده شده در پلتفرم‌های مختلف است.

ملاحظات امنیتی: اسرار را به طور امن ذخیره کرده و در برابر نشت توکن محافظت کنید. از HTTPS برای رمزنگاری ارتباط استفاده کنید. مقدار اطلاعات حساس ذخیره شده در توکن را محدود کنید.

ارث بری از Icommand در .net

در .NET، رابط ICommand یک رابط استاندارد است که برای پیاده‌سازی الگوی Command Pattern استفاده می‌شود. این رابط شامل دو متد اصلی به نام‌های CanExecute و Execute می‌باشد که بتواند دستورات مختلف را اجرا کند.

برای ارث‌بری از رابط ICommand در .NET، شما باید کلاسی ایجاد کنید که این رابط را پیاده‌سازی کند. این کلاس باید دو متد CanExecute و Execute را پیاده‌سازی کند.

اینجا یک مثال ساده از یک کلاس که از رابط ICommand ارث‌بری می‌کند در .NET است:

using System;

using System.Windows.Input;

public class MyCommand : ICommand

{

public event EventHandler CanExecuteChanged;

public bool CanExecute(object parameter)

{

// اینجا قوانین بررسی CanExecute را پیاده‌سازی کنید

return true; // مثال: همیشه می‌تواند اجرا شود

}

public void Execute(object parameter)

{

// اینجا عملیات اجرای دستور را پیاده‌سازی کنید

Console.WriteLine("Command executed");

}

}

در این مثال، کلاس MyCommand از رابط ICommand ارث‌بری کرده است و متدهای CanExecute و Execute را پیاده‌سازی کرده است. در متد CanExecute می‌توانید قوانین بررسی اینکه آیا دستور می‌تواند اجرا شود یا خیر را پیاده‌سازی کنید و در متد Execute عملیات اجرای دستور را پیاده‌سازی کنید.

این کد فقط یک مثال ساده است و شما می‌توانید آن را براساس نیاز خود تغییر دهید.

A [record](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record) in C# is a [class](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class) or [struct](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/struct) that provides special syntax and behavior for working with data models. The record modifier instructs the compiler to synthesize members that are useful for types whose primary role is storing data. These members include an overload of [ToString()](https://learn.microsoft.com/en-us/dotnet/api/system.object.tostring" \l "system-object-tostring) and members that support value equality.

یک ریکورد در C# یک کلاس یا ساختار است که ارائه دهنده نحو و رفتار ویژه برای کار با مدل‌های داده است. این اصلاح‌گر ریکورد به کامپایلر دستور می‌دهد که اعضایی را که برای انواعی که نقش اصلی آن‌ها ذخیره داده است، مفید هستند، تولید کند. این اعضا شامل یک نسخه از ToString() و اعضایی که از برابری مقدار پشتیبانی می‌کنند.

موارد زیر را در نظر بگیرید وقتی بخواهید یک رکورد را به جای یک کلاس یا ساختار استفاده کنید:

1. می‌خواهید یک مدل داده تعریف کنید که بر اساس برابری مقدار وابسته باشد.
2. می‌خواهید یک نوع تعریف کنید که اشیاء آن قابل تغییر نباشند.

مثال زیر یک رکورد عمومی تعریف می‌کند که از پارامترهای موقعیتی برای اعلام و نمونه‌سازی یک رکورد استفاده می‌کند. سپس نام نوع و مقادیر ویژگی‌ها را چاپ می‌کند:

public record Person(string FirstName, string LastName);

public static class Program

{

public static void Main()

{

Person person = new("Nancy", "Davolio");

Console.WriteLine(person);

// output: Person { FirstName = Nancy, LastName = Davolio }

}

}

Claim چیست؟

 در authentication، ما مفهومی به نام claim base identity داریم که در حقیقت با استفاده از آن می‌توان امنیت identity را تامین کرد و فرایند احراز هویت امن‌تری برای ما ایجاد کرد.

**Identity چیست؟**

Identity یکی از سیستم‌های احراز هویت است که پس از سیستم‌هایی همچون ASP.NET Membership و Simple Membership برای ASP.NET MVC معرفی شده است. claim base identity نیز به نوعی یکی از attribute های identity است که به ما برای احراز هویت امن‌تر کمک می‌کند.

**Claim چیست؟**

Claim در حقیقت اعطای اعتبار دسترسی به بخشی از سایت یا اپلیکیشن تحت وب ما، پس از دریافت اطلاعاتی از کاربر است. بهتراست claim را با استفاده از یک مثال توضیح دهیم. ممکن است برای شما نیز پیش آمده باشد که وارد سایتی شوید و این سایت برای ثبت نام شما اطلاعاتی را از شما درخواست کند. این اطلاعات می‌تواند شهر محل زندگی، کشور محل زندگی و یا تاریخ تولد شما و اطلاعات دیگری باشد. مورد مهم اما این است که برنامه‌نویس سایت قانونی مبنی بر اینکه فقط افراد چند کشور مشخص، توانایی استفاده از این سایت را دارند گذاشته باشد. پس از این اتفاق داده‌های دریافت شده از شما بررسی می‌شود و برای نمونه اگر کشور شما با کشورهای مجاز که در درون سایت وجود دارند همخوانی نداشت. اجازه دسترسی شما به محتوای سایت گرفته می‌شود. این دقیقاً مثالی از claim و روش کار آن است.

# تفاوت URL و URI

با وجود اینکه مفهوم این دو اصطلاح بصورت قابل تعویض استفاده میشوند، تفاوت‌های بسیار ریزی میانشان وجود دارد. بطور خلاصه **URI** مخففuniform resource **identifier** و **URL** مخفف uniform resource **locator** است. با توجه به معنی لغوی آنها میتوان پی برد که URI در واقع**تعیین کننده هویت یکنواخت منابع آنلاین** بوده و URL فقط **آدرس** محل قرارگیری یک منبع در اینترنت است.

[HttpPost]

public async Task<IActionResult> Login(string userName, string password)

{

var result = "";

result = await \_loginRepository.Login(userName, password);

if (!result.IsNullOrEmpty())

{

var userProfile = await \_loginRepository.GetUserInfo(result);

#region SetCookie

var claims = new List<Claim>()

{

new Claim("Token" , result),

new Claim("UserId" ,userProfile.userId.ToString()),

};

var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);

var principal = new ClaimsPrincipal(identity);

var properties = new AuthenticationProperties

{

};

await HttpContext.SignInAsync(principal, properties);

#endregion

return Ok("Login True");

}

else

{

return BadRequest();

}

}

در کد احراز هویت شما، از **claims** استفاده شده است زیرا **claims** به عنوان بخشی از سیستم احراز هویت مبتنی بر ادعاها (claims-based authentication) عمل می‌کنند. [این روش احراز هویت به جای استفاده صرف از نام کاربری و رمز عبور، از ادعاها یا اطلاعاتی درباره هویت کاربر استفاده می‌کند تا به کاربر اجازه دسترسی به منابع را بدهد](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-8.0)[1](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-8.0).

[ادعاها (claims) به صورت جفت‌های نام-مقدار هستند که نشان‌دهنده ویژگی‌های کاربر می‌باشند، مانند نام، نقش، ایمیل یا هر داده دیگری که برای احراز هویت لازم است](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-8.0)[2](https://dotnettutorials.net/lesson/claims-based-authorization-in-asp-net-core-identity/). این اطلاعات می‌توانند توسط یک طرف مورد اعتماد صادر شوند و به عنوان بخشی از هویت دیجیتال کاربر در نظر گرفته شوند.

استفاده از **claims** در کد شما به این دلیل است که:

* **Declarative** [**اعلامیه‌ای** هستند: ادعاها به صورت اعلامیه‌ای در کد تعبیه می‌شوند و می‌توانند به صفحات Razor، کنترلرها، یا اکشن‌های درون یک کنترلر اعمال شوند](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-8.0)[1](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-8.0).
* **Multiple** [**چندگانه**: یک هویت می‌تواند شامل چندین ادعا با مقادیر متعدد باشد و می‌تواند شامل چندین ادعا از یک نوع باشد](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-8.0)[1](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-8.0).
* **Policy-Based** [**مبتنی بر سیاست**: الزامات ادعاها مبتنی بر سیاست هستند؛ توسعه‌دهنده باید سیاستی را بسازد و ثبت کند که الزامات ادعاها را بیان می‌کند](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-8.0)[1](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-8.0).

به عنوان مثال، در کد شما، claims هایی مانند “Token” و “UserId” به هویت کاربر اضافه می‌شوند که می‌توانند برای احراز هویت و مجوزهای بعدی استفاده شوند. این claims ها به سیستم اجازه می‌دهند تا بر اساس این اطلاعات، دسترسی‌های کاربر را مدیریت کند.

معنی این سه خط:

var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);

var principal = new ClaimsPrincipal(identity);

var properties = new AuthenticationProperties

این سه خط کد در C# به معنی ایجاد و استفاده از اشیاء مرتبط با احراز هویت مبتنی بر ادعاها (claims-based authentication) هستند:

1. var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);

 این خط یک شی **ClaimsIdentity** جدید ایجاد می‌کند که حاوی ادعاهایی (claims) است که شما تعریف کرده‌اید. [این شی به عنوان هویت دیجیتال کاربر عمل می‌کند و می‌تواند برای احراز هویت و مجوزهای بعدی استفاده شود](https://learn.microsoft.com/en-us/dotnet/api/system.security.claims.claimsidentity?view=net-8.0" \t "_blank)[1](https://learn.microsoft.com/en-us/dotnet/api/system.security.claims.claimsidentity?view=net-8.0).

1. var principal = new ClaimsPrincipal(identity);

 این خط یک شی **ClaimsPrincipal** ایجاد می‌کند که می‌تواند چندین **ClaimsIdentity** را در خود جای دهد. [این شی نماینده کاربر در محیط امنیتی است و می‌تواند برای دسترسی به اطلاعات هویت و ادعاهای مرتبط استفاده شود](https://learn.microsoft.com/en-us/dotnet/api/system.security.claims.claimsprincipal?view=net-8.0" \t "_blank)[2](https://learn.microsoft.com/en-us/dotnet/api/system.security.claims.claimsprincipal?view=net-8.0).

1. var properties = new AuthenticationProperties {};

 این خط یک شی **AuthenticationProperties** ایجاد می‌کند که می‌تواند خصوصیات مرتبط با جلسه احراز هویت را تعریف کند، مانند مدت زمان اعتبار توکن، روش‌های احراز هویت، و غیره.

در کل، این سه شی با هم کار می‌کنند تا یک فرآیند احراز هویت مبتنی بر ادعاها را ایجاد کنند که در آن اطلاعات کاربر به صورت ادعاها ذخیره و مدیریت می‌شوند و می‌توانند برای تصمیم‌گیری‌های مرتبط با احراز هویت و مجوز استفاده شوند. این روش انعطاف‌پذیری بیشتری نسبت به مدل‌های سنتی احراز هویت که تنها بر اساس نقش‌ها کار می‌کنند، ارائه می‌دهد.

Guid id = Guid.Parse(User.Claims.FirstOrDefault(c => c.Type == "UserId").Value ?? Guid.NewGuid().ToString());

این کد بخشی از استخراج شناسه کاربر از اطلاعات کلیم‌ها (Claims) است که توسط ورودی کاربر ارسال شده است.

در اینجا:

- `User.Claims.FirstOrDefault(c => c.Type == "UserId").Value

`: این بخش از کد به دنبال اطلاعاتی با نوع (Type) "UserId" در اطلاعات کلیم‌ها می‌گردد و مقدار آن را استخراج می‌کند.

- `?? Guid.NewGuid().ToString()`:

این بخش از کد یک عملگر به نام `??` است که به عنوان یک عملگر پیش‌فرض (null-coalescing operator) شناخته می‌شود. این عملگر بررسی می‌کند که آیا مقدار قبلی (مقدار استخراج شده از کلیم‌ها) خالی (null) است یا نه. اگر مقدار خالی باشد، مقدار پیش‌فرض جدیدی (که در اینجا یک شناسه یکتا جدید است) تولید می‌شود و استفاده می‌شود.

به طور خلاصه، این کد به دنبال استخراج شناسه کاربر از اطلاعات کلیم‌ها می‌گردد و اگر شناسه موجود نباشد، یک شناسه جدید تولید می‌کند تا از خطاهای پتانسیلی که ممکن است ایجاد شوند، جلوگیری کند.

کاربرد .FirstOrDefault()

استفاده از .FirstOrDefault() به این دلیل است که این متد اولین مورد مطابق با شرط موردنظر را برمی‌گرداند و اگر هیچ موردی مطابق با شرط وجود نداشته باشد، مقدار پیش‌فرض آن نوع داده را برمی‌گرداند که در این حالت null است. به این ترتیب، از وقوع خطا در صورت عدم وجود خانه موردنظر جلوگیری می‌شود.

stratdateEn = new Helper.UsefulFunctions().ShamsiToMiladi(int.Parse(startdate.Substring(0,4)), int.Parse(startdate.Substring(5,2)), int.Parse(startdate.Substring(8,2)));

کارش چیه؟

این کد به زبان برنامه‌نویسی C# است و به نظر می‌رسد که یک تابع یا متد به نام ShamsiToMiladi از یک کلاس به نام UsefulFunctions فراخوانی شده است. این تابع به تاریخ شمسی را به تاریخ میلادی تبدیل می‌کند. ورودی‌های این تابع شامل سه عدد است که به ترتیب سال، ماه و روز تاریخ شمسی است.

بله، این سه خط کد از یک رشته به نام startdate استفاده می‌کنند و از تابع Substring برای بازیابی زیررشته‌های مختلف این رشته استفاده می‌کنند. اولین خط از اندیس ۰ تا ۳ (به تعداد ۴ حرف)، دومین خط از اندیس ۵ تا ۶ (به تعداد ۲ حرف) و سومین خط از اندیس ۸ تا ۹ (به تعداد ۲ حرف) این رشته را استخراج می‌کنند.

برای استفاده از AutoMapper در متد GetLibrariesListView، شما باید مراحل زیر را دنبال کنید:

1. **نصب AutoMapper**: اگر هنوز AutoMapper را نصب نکرده‌اید، می‌توانید آن را با استفاده از NuGet Package Manager نصب کنید:
2. dotnet add package AutoMapper.Extensions.Microsoft.DependencyInjection
3. **ایجاد پروفایل نقشه‌برداری**: یک پروفایل برای نقشه‌برداری بین مدل‌های Library و LibOrgViewModelViews ایجاد کنید.
4. public class MappingProfile : Profile
5. {
6. public MappingProfile()
7. {
8. CreateMap<Library, LibOrgViewModelViews>();
9. }
10. }
11. **تنظیم AutoMapper**: در Startup.cs یا Program.cs، AutoMapper را تنظیم کنید:
12. services.AddAutoMapper(typeof(MappingProfile));
13. **استفاده از AutoMapper در متد**: حالا می‌توانید از AutoMapper در متد GetLibrariesListView استفاده کنید. به‌جای ایجاد شی جدید از LibOrgViewModelViews به‌صورت دستی، می‌توانید از متد Map استفاده کنید.

### کد نهایی با استفاده از AutoMapper:

public async Task<List<LibOrgViewModelViews>> GetLibrariesListView()

{

try

{

var libraries = await \_apiContext.LibrariesList\_view.Take(10).ToListAsync();

return \_mapper.Map<List<LibOrgViewModelViews>>(libraries);

}

catch (Exception ex)

{

// می‌توانید لاگ خطا را اضافه کنید

throw;

}

}

### توضیحات:

* **نقشه‌برداری خودکار**: با استفاده از \_mapper.Map<List<LibOrgViewModelViews>>(libraries)، AutoMapper به‌طور خودکار مقادیر را از لیست libraries به لیست LibOrgViewModelViews منتقل می‌کند.
* **کاهش کد تکراری**: این روش به شما کمک می‌کند تا کد خود را تمیزتر و قابل نگهداری‌تر کنید و از تکرار کد جلوگیری کنید.
* **مدیریت خطا**: در صورت بروز خطا، می‌توانید لاگ‌گذاری را اضافه کنید تا اطلاعات بیشتری درباره خطاها داشته باشید.

با این تغییرات، متد شما به‌طور مؤثرتری عمل خواهد کرد و از قابلیت‌های AutoMapper بهره‌مند خواهد شد.

Get smarter answer from GPT-4o

آموزش LINQ در تاپ لرن

|  |  |
| --- | --- |
| int[] numbers = { 1, 3, 5, 9, 11, 13, 15, 19, 25, 35, 45, 51, 63, 68, 70, 90, 100 };  int[] result = (from n in numbers select n).ToArray();  var result2 = (from n in numbers  orderby n descending  select n).ToArray(); | Linq orderby  مرتب سازی آرایه  N قرمز در اینجا alias است. |
| int[] numbers = { 1, 3, 5, 9, 11, 13, 15, 19, 25, 35, 45, 51, 63, 68, 70, 90, 100 };  var result3 = (from n in numbers  where n>30 && n<100  orderby n descending  select n).ToArray(); | Linq where  اعداد بزرگتر از 30 و کوچکتر از 100 را نشان بده. (با where شرط را میگذاریم و سپس با orderby به صورت نزولی مرتب و سپس با select n تمام داده ها را برگردان. به چه شکلی؟ یا ToArray و یا ToList. چون var است مشکلی ندارد. |
| int[] numbers = { 1, 3, 5, 9, 11, 13, 15, 19, 25, 35, 45, 51, 63, 68, 70, 90, 100 };  var result4 = (from n in numbers  where n == 3  select n).First(); | Linq first()  برو در مجموعه numbers و به دنبال 3 بگرد و هرچندتا که پیدا کردی اولیش رو برام بیار. اگرم پیدا نکرد exeption میده. |
| int[] numbers = { 1, 3, 5, 9, 11, 13, 15,3, 19, 25, 35, 45, 51, 63, 68, 70, 90, 100 };  var result5 = (from n in numbers  where n == 3  select n).FirstOrDefault(); | Linq FirstOrDefault()  برو بگرد اگه هرچندتا پیدا کردی اولیش را بیار و اگه نبود default متغیر سمت چپ مساوی یعنی n را قرار بده که 0 است. اگر مقداری پیدا نکرد دیگه error نمیدهد. بود میاره نبود 0 میده. |
| دو دستور بالا دو معادل دیگه هم دارند. Single و singleOrDefault  تفاوت ها: first کل مجموعه رو پیمایش میکنه و همه رو پیدا میکنه، بعد list میکنه و اولی رو برمیگردونه. firstOrDefault هم مثل همون. ولی single یا singleOrDefault کل مجموعه رو پیمایش نمیکنه که دنبال چند تا بگرده. به اولی که برسه برمیگردونه. اگه در این مسیر دوتا پیدا کنه exeption میده. یعنی single حتما باید یک دانا باشه. |  |
| var result6 = (from n in numbers  where n == 3  select n).Single();  var result7 = (from n in numbers  where n == 3  select n).SingleOrDefault(); | Linq  Single() singleOrDefault() |
| int[] numbers = { 1, 3, 5, 9, 11, 13, 15,3, 19, 25, 35, 45, 51, 63, 68, 70, 90, 100 };  int count = (from n in numbers  select n).Count(); | Linq Count()  بدست آوردن تعداد کل اعضا  میتواند وسطش where هم بگیرد. |
| int[] numbers = { 1, 3, 5, 9, 11, 13, 15,3, 19, 25, 35, 45, 51, 63, 68, 70, 90, 100 };  int sum = (from n in numbers  select n).Sum(); | Linq Sum()  جمع حقوق دریافتی ایمان مدائنی یا جمع اعضا. میتواند وسطش where هم بگیرد. |
| int[] numbers = { 1, 3, 5, 9, 11, 13, 15,3, 19, 25, 35, 45, 51, 63, 68, 70, 90, 100 };  int min = (from n in numbers  select n).Min();  int max = (from n in numbers  select n).Max(); | Linq min() max()  بزرگترین عضو و کوچکترین عضو. |
| int[] numbers = { 1, 3, 5, 9, 11, 13, 15,3, 19, 25, 35, 45, 51, 63, 68, 70, 90, 100 };  var result8 = (from n in numbers  orderby n descending  select n).Take(4).ToList(); | Linq  در یک سایت خبری 4 تا از پربازدیدترین خبرهام رو نشون بدم. پس براساس بازدید خبر مرتب بشوند. حالا Take() که داخلش تعداد میزنیم. 4 تای بزرگ را برمیگرداند. حتما انتهاش ToList یا ToArray باشد. |
| var result9 = (from n in numbers  orderby n descending  select n).Skip(4).Take(4).ToArray();  با استفاده از Take() و Skip() میتونیم صفحه بندی پروژه هامون رو بدست بیاریم. آموزش صفحه بندی سفارشی در سایت تاپلرن  https://toplearn.com/courses/web/%D8%B3%D8%A7%D8%AE%D8%AA-%D8%B5%D9%81%D8%AD%D9%87-%D8%A8%D9%86%D8%AF%DB%8C-(Pagging)-%D8%B3%D9%81%D8%A7%D8%B1%D8%B4%DB%8C-%D8%AF%D8%B1-MVC- | Linq  4تای بزرگ اول را نمیخوام. 4 تای بزرگ دوم را میخوام. پس اول با دستور Skip(4) از روی 4 تای اول میپرم و بعد Take() میکنم. |
| List<string> Names = new List<string>();  Names.Add("Iman");  Names.Add("Aala");  Names.Add("Ali");  Names.Add("Sara");  Names.Add("Mina");  Names.Add("Parham");  var result11 = (from n in Names  where n.Contains("i")  select n).ToList();  در خروجی ایمان را نیاورد. چون I بود (حساسیت به حروف بزرگ و کوچک یا case sensitive داره.) . راه حل: | Linq Contains()  در sql از like %% استفاده میکردیم. الان از Contains() |
| List<string> Names = new List<string>();  Names.Add("Iman");  Names.Add("Aala");  Names.Add("Ali");  Names.Add("Sara");  Names.Add("Mina");  Names.Add("Parham");  var result11 = (from n in Names  where n.ToLower().Contains("i")  select n).ToList(); |  |
| var result12 = (from n in Names  where n.ToLower().StartsWith("i")  select n).ToList();  var result13 = (from n in Names  where n.ToLower().EndsWith("i")  select n).ToList(); | Linq  اونایی که با I شروع شده و اونایی که با I تمام شدند. |
| var result14 = (from n in numbers  where n.ToString().Contains("1")  select n).ToList(); | Linq  میتونم در numbers دنبال اعدادی بگردم که 1 دارند؟ نه چون contains() برای string است. اول تبدیل بعد جستجو. |
| بهینه تر و پر سرعت تر از دستورات بالاست. |  |
|  |  |
| int[] numbers = { 1, 3, 5, 9, 11, 13, 15, 3, 19, 25, 35, 45, 51, 63, 68, 70, 90, 100 };  var lambda = numbers.ToArray(); |  |
| var result2 = (from n in numbers  orderby n descending  select n).ToArray();  معادل هم هستند.  var l2 = numbers.OrderByDescending(n => n).ToArray(); |  |
| var result3 = (from n in numbers  where n > 30 && n < 100  orderby n descending  select n).ToArray();  معادل هم هستند.  var l3 = numbers.Where(n => n>30 && n<100).OrderByDescending(n =>n).ToArray(); |  |
| var result4 = (from n in numbers  where n == 3  select n).First();  var l4 = numbers.First(n => n == 3); |  |
| List<Person> people = new List<Person>(); //ساخت لیستی از کلاس  Person p1 = new Person();//ساخت نمونه ای از کلاس  p1.Id = 1; //مقدار دهی به روش اول  p1.Name = "Test";  p1.Web = "wwwtest";  people.Add(p1);    Person p2 = new Person()//مقداردهی به روش دوم Object intioalazer  {  Id = 2,  Name = "Test2",  Web = "wwwTest2"  };  people.Add(p2);  people.Add(new Person() { Id = 3, Name = "test3", Web = "wwwTest3" });//مقداردهی به روش سوم  public class Person //ساخت کلاس  {  public int Id { get; set; }  public string Name { get; set; }  public string Web { get; set; }  } |  |
| var r = people.ToList(); |  |
| var r2 = people.OrderByDescending(I => I.Id).ToList(); |  |
| var r3 = people.Where(p =>p.Name.ToLower().Contains("t")).ToList(); |  |
| var r4 = people.FirstOrDefault(p =>p.Id == 1); |  |
| var r5 = people.Select(p => p.Name).ToList();  میتونستیم قبلش where یا orderby هم بگذاریم.  اینجا فقط یکی استخراج میکند .فقط یکی. و این بده. |  |
| var r6 = people.Select(p => new  {  p.Name,  p.Web  }).ToList(); |  |
| دیدن تمام دستورات linq : Linq 101 |  |