Helpers

1. What Are Helpers?

Definition:

Helpers are **utility classes or methods** designed to perform common tasks that are not specific to your business logic. They provide reusable functionality that can be used across the application.

Why Use Helpers?

1. **Reusability**:
    - Encapsulate logic that can be reused throughout the project.
2. **Separation of Concerns**:
    - Keep controllers, services, and other layers clean by moving repetitive or non-essential code to helpers.
3. **Maintainability**:
    - Centralize utility logic in one place, making it easier to update or debug.

n **.NET Core C#**, a **helper** is a class that provides utility functions or reusable code snippets to make tasks easier. Helpers are not special classes in .NET; they are regular classes designed to perform common operations, like formatting data, logging, or interacting with external APIs.

Key Points:

1. Helpers typically contain **static methods** so you don't need to create an instance of the class to use them.
2. They are not automatically recognized by .NET as "helper classes." Instead, developers use them for convenience and call their methods explicitly.

Simple Example:

Imagine you often need to format dates in a specific way across your project. Instead of repeating the same code, you can create a **DateHelper**.

```
public static class DateHelper
{
    // A method to format a date in "dd/MM/yyyy" format
    public static string FormatDate(DateTime date)
    {
    return date.ToString("dd/MM/yyyy");
    }
}
```

Using the Helper:

You can use this helper method wherever needed:

```
using System;

class Program

{

        static void Main()

        {

        DateTime today = DateTime.Now;

        // Using the DateHelper to format the date

        string formattedDate = DateHelper.FormatDate(today);

        Console.WriteLine("Formatted Date: " + formattedDate);

        }

}
```

How .NET Knows the Class Is a Helper:

.NET doesn't have a built-in mechanism to identify a "helper" class. The term "helper" is just a convention used by developers. You define and organize it based on your needs.

Where to Place Helper Classes:

1. **In a Helpers Folder**: Many projects have a folder named `Helpers` or `Utilities` where all helper classes are stored.
2. **Using Namespaces**: You can place helper classes in a specific namespace, such as `MyProject.Helpers`, to organize them.

```
namespace MyProject.Helpers
{     public static class StringHelper
      {
      public static bool IsNullOrEmpty(string value)
      {           return string.IsNullOrEmpty(value);        }
      }
}
```

Example Usage:

```
using MyProject.Helpers;
class Program
{
	static void Main()
	{
	string input = null;
	bool result = StringHelper.IsNullOrEmpty(input);
	Console.WriteLine("Is null or empty: " + result);
	}
}
```

2. Common Types of Helpers

1. **JWT Token Generator**: For creating JSON Web Tokens for authentication.
2. **Date-Time Helper**: For formatting or converting dates and times.
3. **Email Sender**: For sending emails programmatically.

3. Implementing Helpers

3.1 JWT Token Generator

Step 1: Create a Helper Class

```
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using Microsoft.IdentityModel.Tokens;
public static class JwtTokenHelper
{       public static string GenerateToken(string username, string secretKey, int expiryMinutes)
        {       var tokenHandler = new JwtSecurityTokenHandler();
        var key = Encoding.ASCII.GetBytes(secretKey);
        var tokenDescriptor = new SecurityTokenDescriptor
        {
        Subject = new ClaimsIdentity(new[]
        {
        new Claim(ClaimTypes.Name, username)
        }),
```

```
            Expires = DateTime.UtcNow.AddMinutes(expiryMinutes),
            SigningCredentials = new SigningCredentials(
            new SymmetricSecurityKey(key),
            SecurityAlgorithms.HmacSha256Signature
            )
        };

        var token = tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
        }
}
```

Step 2: Use the Helper Use this helper in a service or controller to generate tokens.

```
[HttpPost("login")]
public IActionResult Login(string username)
{
        var secretKey = "YourSecretKey12345"; // Should come from appsettings
        var token = JwtTokenHelper.GenerateToken(username, secretKey, 30); // Token valid for 30 minutes
        return Ok(new { Token = token });
}
```

3.2 Date-Time Helper

Step 1: Create a Helper Class

```
using System;
public static class DateTimeHelper
{       public static string FormatDate(DateTime date)
        {
        return date.ToString("yyyy-MM-dd HH:mm:ss"); // Format: 2024-11-17 15:30:45       }
        public static DateTime ConvertToUtc(DateTime date, string timeZoneId)
        {
        var timeZone = TimeZoneInfo.FindSystemTimeZoneById(timeZoneId);
        return TimeZoneInfo.ConvertTimeToUtc(date, timeZone);   }
}
```

Step 2: Use the Helper Example usage in a controller or service:

```
[HttpGet("formatted-date")]
public IActionResult GetFormattedDate()
{
        var now = DateTime.Now;
        var formattedDate = DateTimeHelper.FormatDate(now);
        return Ok(new { FormattedDate = formattedDate });
}
[HttpGet("convert-to-utc")]
public IActionResult ConvertToUtc(DateTime date)
{
        var utcDate = DateTimeHelper.ConvertToUtc(date, "Eastern Standard Time");
        return Ok(new { UtcDate = utcDate });
}
```

3.3 Email Sender

Step 1: Install Required Package Install the `MailKit` NuGet package for sending emails:

```
dotnet add package MailKit
```

Step 2: Create an Email Helper Class

```
using MailKit.Net.Smtp;

using MimeKit;

using System.Threading.Tasks;

public static class EmailHelper

{       public static async Task SendEmailAsync(string toEmail, string subject, string body, string fromEmail, string smtpServer, int smtpPort, string smtpUser, string smtpPass)

        {       var message = new MimeMessage();

        message.From.Add(new MailboxAddress("Your App Name", fromEmail));

        message.To.Add(new MailboxAddress("", toEmail));

        message.Subject = subject;

        var bodyBuilder = new BodyBuilder { HtmlBody = body };

        message.Body = bodyBuilder.ToMessageBody();
```

```
        using (var client = new SmtpClient())
        {
            await client.ConnectAsync(smtpServer, smtpPort, true); // Use true for
SSL
            await client.AuthenticateAsync(smtpUser, smtpPass);
            await client.SendAsync(message);
            await client.DisconnectAsync(true);
        }
        }
}
```

Step 3: Use the Helper Call the helper to send an email.

```
[HttpPost("send-email")]
public async Task<IActionResult> SendEmail(string toEmail)
{       var subject = "Welcome to Our App";
        var body = "<h1>Hello!</h1><p>Thank you for joining us.</p>";
        var fromEmail = "yourapp@example.com";
        var smtpServer = "smtp.example.com";
        var smtpPort = 587;
        var smtpUser = "smtp-user";
        var smtpPass = "smtp-password";
        await EmailHelper.SendEmailAsync(toEmail, subject, body, fromEmail, smtpServer, smtpPort, smtpUser, smtpPass);
        return Ok(new { Message = "Email sent successfully" });
}
```

4. Key Takeaways

1. **What Are Helpers?**
    - Helpers are utility classes or methods for reusable tasks like generating tokens, formatting dates, or sending emails.
2. **Why Use Helpers?**
    - Reduce code duplication.
    - Keep your controllers and services focused on business logic.
3. **Examples of Helpers:**
    - **JWT Token Generator:** Simplifies authentication.
    - **Date-Time Helper:** Formats or converts dates.
    - **Email Sender:** Sends emails programmatically.

Final Notes

Helpers should:

- Be **stateless**.
- Be used wherever functionality is repetitive or not business-specific.
- Be well-documented to ensure proper usage.