Code Review

1. Code Format - Tried Visual Studio **Community** Format Document **Ctrl K, + Ctrl D**?
   - Separate file for each class, interface, record, struct, and enum other than partial classes and partial methods.
   - Code documentation summary, param, returns, and description with period in sentence case.
   - One space before and after colon while extending or implementing a type.
   - One blank line between method definitions, property definitions, and blocks of code.
   - Discard whitespace.
   - Curly braces for single statement if and if else block.
   - Self-documenting naming conventions for types, methods, variables (member or local), and follow consistent naming conventions across the application.
   - Note:
     .NET Compiler Platform (Roslyn) Analyzers - Code Analysis
     - Identifiers should not contain underscores

     DotNet Analyzers - StyleCop Analyzers
     - A violation of rule occurs when a field name begins with an underscore.

     DotNet Analyzers - StyleCop Analyzers
     - A violation of rule occurs when a field name contains an underscore.
   - Identifiers for namespaces, types, members, and parameters cannot differ only by case.
   - Specify access modifiers for default explicitly, private for private members and internal for internal types.
   - Document README to tell why the project is useful, what can be done with the project, and how the project can be used.
   - Verify packages.config file for dependencies and resolve conflicts.
   - Document unresolved references and remove those manually before managing packages using Package Manager.
   - Use Templates for Emails.
2. Configuration:
   - Configure URL, Path, Directory File, Date Time Format, Host, Proxy Host, Port, Proxy Port, App Environment, Session Timeout, Authentication Timeout, App Secret
   - Safe storage of App Secret in development
     - Command-line argument
     - Custom provider, installed or created
     - Directory files
     - In-memory object
     - Key Vault
3. Diagnostics - Logging and tracing for instrumenting code to create log files.
4. Consider logging framework, built-in or third-party logging provider and use it in conjunction with Microsoft.Extensions.Logging ILogger interface for Dependency injection (DI).
5. Remove unused using statements.
6. Sort using statements in the following order:
   1. static built-in System namespaces
   2. Built-in System namespaces
   3. Microsoft namespaces
   4. Third-party namespaces

5. Custom namespaces

7. Reduce the visibility of types and type members whether they can be accessed within the assembly or from friend, satellite or other assemblies.
8. DONOT declare read only mutable reference types : An externally visible type that contains an externally visible read-only field that is a mutable reference type may be a security vulnerability
9. A static class can be used as a convenient container for sets of methods that just operate on input parameters and do not have to get or set any internal instance fields.
10. Use this keyword to qualify members hidden by similar names (member name similar to parameter name) in constructor.
11. Use Description Attribute for enums.
12. For Type Conversion from string use TryParse as Convert and Parse require Exception handling, Convert is more useful as it uses Parse internally.
13. Class library for utility classes and constants.
14. Class library Resources File for custom messages.
15. Globalization - CultureInfo: String comparison may lead to unexpected results when comparisons are affected by culture-specific casing rules.
16. Globalization - CultureInfo: Conversion to string with number format for currency
17. Globalization - CultureInfo - IFormatProvider: Standard date and time format strings.
18. Make the right choice between DateTime.Now and DateTime.UtcNow, and DateTimeOffset.UtcNow.
19. String check for string whitespace: string.IsNullOrWhiteSpace instead of string.IsNullOrEmpty
20. string.IsNullOrWhiteSpace takes precedence over !string.IsNullOrWhiteSpace.
21. Redundant string format for string Interpolation $.
22. Verbatim string literal @ instead of regular string literal \\.
23. Universal Naming Convention (UNC) path for server and file.
24. HttpStatusCode enum
25. throw new ArgumentNullException

```
_ = arg ?? throw new ArgumentNullException(nameof(arg));
```

26. throw new ArgumentNullException

```
if (arg1 == null || arg2 == null)
{
   throw new ArgumentException("arg1 or arg2 is null");
}
```

27. Suppress unjustified warnings

```
#pragma warning disable CA1305 // Specify IFormatProvider
            new Claim(ClaimTypes.NameIdentifier, user.UserId.ToString()),
#pragma warning restore CA1305 // Specify IFormatProvider
```

28. Exceptions must be re-thrown from the point where the stack trace should begin, catch more specific exception and re-throw for preserving call stack, put catch blocks targeted to specific allowed exceptions before a general exception catch block

 # Instead of catching general exception viz. in Service

```
catch (Exception e)
{

}
```

 # Catch more specific allowed exceptions before a general exception catch block

```
catch (MailKit.Net.Smtp.SmtpCommandException e)
{
   // Log exception
   throw;
}
catch (MailKit.Net.Smtp.SmtpProtocolException e)
{
   // Log exception
   throw;
}
catch (MailKit.Security.AuthenticationException e)
{
   // Log exception
   throw;
}
catch (MailKit.Security.SslHandshakeException e)
{
   // Log exception
   throw;
}
```

 # Depending on whether to catch generic exception

```
catch (Exception e)
{
   // Log exception
   throw;
}
```

 # Handle the exception re-thrown viz. in Controller

```
try
{
   // Draft message and email
   Service.Email(message);
}
catch (Exception e)
{
   // Log exception
   return Problem("Email Failed.");
}
```

29. DONOT throw System.Exception, System.SystemException, System.NullReferenceException, or System.IndexOutOfRangeException
30. DONOT create exceptions that can be thrown in debug mode but not release mode. To identify run-time errors during the development phase, use Debug Assert instead.
31. Use ActionResult<T> instead of ActionResult, to return a type deriving from ActionResult or return a specific type.
32. IActionResult return type is appropriate when multiple ActionResult return types are possible in an action.
33. Inject

```
@using Microsoft.Extensions.Options
@inject IOptions<AppSettings> appSetting
```

34. Use C# latest features viz. init-only property
35. Be careful not to accidentally change a type of an element of the iterable collection. For example, it is easy to switch from System.Linq.IQueryable to System.Collections.IEnumerable in a foreach statement, which changes the execution of a query.

 # Sample Code with Code Documentation # Tried and tested # Self-documenting code/variable names

```
//-----------------------------------------------------------------------
// <copyright file=" RandomPasswordGenerator.cs" company="Company Name">
//     Copyright (c) Company Name. All rights reserved.
// </copyright>
// <author>Jane Doe</author>
//-----------------------------------------------------------------------
```

Or

```
// Copyright (c) Company Name. All rights reserved.
// Licensed under the IT License. See LICENSE in the project root for license information.
using System;
using System.Collections.Generic;
using System.Linq;

namespace Utility
{
   /// <summary>
   /// Helper utility for generating a random password.
   /// </summary>
   public static class RandomPasswordGenerator
   {
      /// <summary>
      /// Generates a random password.
      /// <returns>A random password.</returns>
      public static string GenerateRandomPassword()
      {
         const int RequiredLength = 8;
```

```
        const int RequiredUniqueCharacters = 4;
        const bool RequireUppercase = true;
        const bool RequireLowercase = true;
        const bool RequireDigit = true;
        const bool RequireNonAlphanumeric = true;

        string upperCase = string.Join("", Enumerable.Range('A', 'Z' - 'A' + 1).Select(x =>
((char)x).ToString()));
        string lowerCase = string.Join("", Enumerable.Range('a', 'z' - 'a' + 1).Select(x =>
((char)x).ToString()));
        string digits = string.Join("", Enumerable.Range(0, 10).Select(x => x.ToString()));
        string nonAlphanumeric = string.Join("", Enumerable.Range(32, 127).Where(x => x >= 32 && x <=
47 || x >= 58 && x <= 64 || x >= 91 && x <= 96 || x >= 123 && x <= 126).Select(x =>
((char)x).ToString()));

        var ascii = new string[]
        {
           upperCase,
           lowerCase,
           digits,
           nonAlphanumeric
        };

        var randomNumber = new Random(Environment.TickCount);
        var characters = new List<char>();

        if (RequireUppercase)
        {
           characters.Insert(randomNumber.Next(0, characters.Count), ascii[0][randomNumber.Next(0,
ascii[0].Length)]);
        }

        if (RequireLowercase)
        {
           characters.Insert(randomNumber.Next(0, characters.Count), ascii[1][randomNumber.Next(0,
ascii[1].Length)]);
        }

        if (RequireDigit)
        {
           characters.Insert(randomNumber.Next(0, characters.Count), ascii[2][randomNumber.Next(0,
ascii[2].Length)]);
        }

        if (RequireNonAlphanumeric)
        {
           characters.Insert(randomNumber.Next(0, characters.Count), ascii[3][randomNumber.Next(0,
ascii[3].Length)]);
```

```
        }

        for (int index = characters.Count; index < RequiredLength || characters.Distinct().Count() <
RequiredUniqueCharacters; index++)
        {
            string sequence = ascii[randomNumber.Next(0, ascii.Length)];
            characters.Insert(randomNumber.Next(0, characters.Count), sequence[randomNumber.Next(0,
sequence.Length)]);
        }

        return new string(characters.ToArray());
      }
   }
}
```