Using Dependency Injection (DI) to register and resolve application services

Creating a service to assign request IDs

1. Create a new application using the ASP.NET Core Web Application (.NET Core) (Empty Template)
2. Or, if using VSCode, call `yo aspnet` and select `Empty Web Application`
3. Create a folder in the application called `Services`
4. Create a new class file in the `Services` folder `RequestId`
5. In the file, create an interface `IRequestIdFactory` with a single method `string MakeRequestId()`
6. In the same file, create a class `RequestIdFactory` that implements `IRequestIdFactory` by using `Interlock.Increment` to make an increasing request ID
7. The file should look something like this:

```
public interface IRequestIdFactory
{
    string MakeRequestId();
}

public class RequestIdFactory : IRequestIdFactory
{
    private int _requestId;

    public string MakeRequestId()
        => Interlocked.Increment(ref _requestId).ToString();
}
```

8. In the same file, create an interface `IRequestId` with a single property `string Id { get; }`
9. In the same file, create a class `RequestId` that implements `IRequestId` by taking an `IRequestIdFactory` in the constructor and calling its `MakeRequestId` method to get a new ID.
10. The whole file should now look something like this:

```
using System.Threading;

public interface IRequestIdFactory
{
    string MakeRequestId();
}

public class RequestIdFactory : IRequestIdFactory
{
    private int _requestId;

    public string MakeRequestId()
        => Interlocked.Increment(ref _requestId).ToString();
}
```

```
public interface IRequestId
{
    string Id { get; }
}

public class RequestId : IRequestId
{
    private readonly string _requestId;

    public RequestId(IRequestIdFactory requestIdFactory)
    {
        _requestId = requestIdFactory.MakeRequestId();
    }

    public string Id => _requestId;
}
```

Register the request ID service in DI

1. In the application's `Startup.cs` file, find the `ConfigureServices(IServiceCollection services)` method.
2. Register the `IRequestIdFactory` service as a singleton: `services.AddSingleton<IRequestIdFactory, RequestIdFactory>();`
3. Register the `IRequestId` service as scoped: `services.AddScoped<IRequestId, RequestId>();`
   1 The `ConfigureServices` method should now look something like this:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IRequestIdFactory, RequestIdFactory>();
    services.AddScoped<IRequestId, RequestId>();
}
```

Create and add a middleware that logs the request ID

1. Create a new folder in the application `Middleware`
2. In the folder, create a class `RequestIdMiddleware`
3. Create a constructor `public RequestIdMiddleware(RequestDelegate next, IRequestId requestId, ILogger<RequestIdMiddleware> logger)` and store the parameters in private fields
4. Add a method `public Task Invoke(HttpContext context)` and in its body log the request ID using the `ILogger` and `IRequestId` injected from the constructor
5. Your middleware class should look something like this:

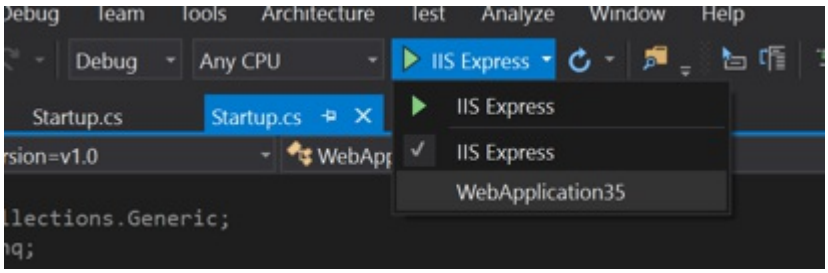
```
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using System.Threading.Tasks;

public class RequestIdMiddleware
```

```
{
    private readonly RequestDelegate _next;
    private readonly ILogger<RequestIdMiddleware> _logger;

    public RequestIdMiddleware(RequestDelegate next, IRequestId requestId,
        ILogger<RequestIdMiddleware> logger)
    {
        _next = next;
        _logger = logger;
    }

    public Task Invoke(HttpContext context, IRequestId requestId)
    {
        _logger.LogInformation($"Request {requestId.Id} executing.");

        return _next(context);
    }
}
```

6. Add the middleware to your pipeline back in `Startup.cs` by calling `app.UseMiddleware<RequestIdMiddleware>();` before the call to `app.Run()`:

```
    app.UseMiddleware<RequestIdMiddleware>();

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
```

7. Change the Debug drop down in the toolbar to the application name as shown below.



8. Run the application. You should see the logging messages from the middleware in the console output.

Extra: Adding a unit test project

Follow the instructions at https://xunit.github.io/docs/getting-started-dotnet-core.html to add an xUnit testing project.