Overview

Scan Result

Application Name	Project Name	Application Type	Application Components
DemoCpApp	DemoProject	Web Application	10
Application Platform	**Scanned Date**		
.Net	03/03/2022		

.Net Core Assessment

Application Migration Recommendation

.NET Core is the modern and modular version of .NET Framework. Microsoft guidance states that .NET Core is the future of .NET. Although Microsoft plans to support .NET Framework, any new features will be added only to .NET Core (and eventually .NET 5). Any new development should occur on .NET Core. One of the most important aspects of .NET Core is that it's multi-platform. You can containerize a .NET Core application into a Linux container. Linux containers are more lightweight, and they run on more platforms more efficiently.

Recommendations

Datapoint

Http Runtime Section

Module

Application & Platform Design

Reason for change

Syntax and Implementation changed

Code block	Line no.	File path
<httpRuntime targetFramework="4.5" maxRequestLength="10485760" executionTimeout="3600" />	78	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Web.config

Recommendation

In a classic ASP.NET application everything is hosted inside of an IIS Worker Process (w3wp.exe) which is the IIS Application Pool. The AppPool hosts your ASP.NET application and your application is instantiated by the built-in ASP.NET hosting features in IIS. The native runtime manager instantiates the .NET Runtime on your application's behalf and brings up the HttpRuntime object which is then used to fire requests through the ASP.NET application pipeline as requests come in from the native http.sys driver. Requests come in from http.sys and are dispatched to the appropriate site that is mapped to the Application Pool and the HttpRuntime instance hosted there.

ASP.NET Core with IIS

Things are quite different with ASP.NET Core which doesn't run in-process to the IIS worker process, but rather runs as a separate, out of process Console application that runs its own Web server using the Kestrel component. Kestrel is a .NET Web Server implementation that has been heavily optimized for throughput performance. It's fast and functional in getting network requests into your application, but it's 'just' a raw Web server. It does not include Web management services as a full featured server like IIS does.

If you run on Windows you will likely want to run Kestrel behind IIS to gain infrastructure features like port 80/443 forwarding via Host Headers, process lifetime management and certificate management to name a few.

ASP.NET Core applications are standalone Console applications invoked through the dotnet runtime command. They are not loaded into an IIS worker process, but rather loaded through a native IIS module called AspNetCoreModule that executes the external Console application. Once you've installed the hosting bundle (or you install the .NET Core SDK on your Dev machine) the AspNetCoreModule is available in the IIS native module list:

The AspNetCoreModule is a native IIS module that hooks into the IIS pipeline very early in the request cycle and immediately redirects all traffic to the backend ASP.NET Core application. All requests - even those mapped to top level Handlers like ASPX bypass the IIS pipeline and are forwarded to the ASP.NET Core process. This means you can't easily mix ASP.NET Core and other frameworks in the same Site/Virtual directory, which feels a bit like a step back given that you could easily mix frameworks before in IIS. While the IIS Site/Virtual still needs an IIS Application Pool to run in, the Application Pool should be set to use No Managed Code. Since the App Pool acts merely as a proxy to forward requests, there's no need to have it instantiate a .NET runtime.

The AspNetCoreModule's job is to ensure that your application gets loaded when the first request comes in and that the process stays loaded if for some reason the application crashes. You essentially get the same behavior as classic ASP.NET applications that are managed by WAS (Windows Activation Service).

Estimated Efforts

16 Hours **Size :** Medium

Impact

Mandatory

Help URL

No Record Found

Datapoint

AppSettingsSection

Module

Application & Platform Design

Reason for change

Syntax and Implementation changed

Code block	Line no.	File path
<appSettings> <add key="Email" value="gifting@s1mgift.com" /> <add key="Password" value="********" /> <add key="MailServer" value="smtp.office365.com" /> <add key="MailPort" value="587" /> <add key="IsSSLEnabled" value="true" /> <add key="EncryptionKey" value="D8215811-32B0-4276-9766-0984D6F53AD7" /> <add key="webpages:Version" value="3.0.0.0" /> <add key="webpages:Enabled" value="false" /> <add key="ClientValidationEnabled" value="true" /> <add key="UnobtrusiveJavaScriptEnabled" value="true" /> <add key="ApiPath" value="https://s1mapi.azurewebsites.net/" /> <add key="powerbi:AccessKey" value="HgnbWB2Q98MNWV+EKFNtDWNo7GiQq/b1Fx4wUN60mjk7y3Yq8rd+lwCdyTl78joPYJ1fBwRixaQnzBi4btte5A==" /> <add key="powerbi:ApiUrl" value="https://api.powerbi.com" /> <add key="powerbi:WorkspaceCollection" value="s1mws" /> <add key="powerbi:WorkspaceId" value="a4e0e2ed-5d4b-45fb-9173-a4a77dc4cba9" /> <add key="powerbi:reportId" value="59315de6-bc92-425b-985f-de37de3e6131" /> <add key="BlobStorage" value="https://s1mblob.blob.core.windows.net/s1m/" /> <add key="StorageConnectionString"	69	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Web.config

```
value="DefaultEndpointsProtocol=https;Accoun
tName=s1mblob;AccountKey=********" /> <add
key="appURL"
value="https://s1mclientapp.azurewebsites.net/
?campaignid=" /> <add
key="aspnet:MaxJsonDeserializerMembers"
value="21407483647" /> </appSettings>
```

Recommendation

With ASP.NET, the configuration file used an XML file. Whereas, with ASP.NET Core, it uses the JSON format. JSON is a lot more compact. No more having to set lengthy opening and closing tags. And that's not forget schemas! In-turn, that means that the file is easily to read and is a lot more lightweight. Here is an example of how it looks:

```
      {   "Logging": {     "LogLevel": {       "Default": "Information",       "Microsoft": "Warning
",       "Microsoft.Hosting.Lifetime": "Information"     }   },   "AllowedHosts": "*" }
```

Setting up Custom Configuration

We are going to set up a RoundTheCodeSync custom configuration. To do this, we added some settings within the appsettings.json file, grouped by a RoundTheCodeSync key.

```
  {   "Logging": {     "LogLevel": {       "Default": "Information",       "Microsoft": "Warning",
     "Microsoft.Hosting.Lifetime": "Information"     }   },   "AllowedHosts": "*",   "RoundTheCodeSy
nc": {     "Title": "Our Sync Tool",     "Interval": "0:30:00",     "ConcurrentThreads": 10   } }
```

From there, we can set up an interface and class in our ASP.NET Core MVC application to read our RoundTheCodeSync custom configuration.

```
  public interface IRoundTheCodeSync {     string Title { get; set; }     TimeSpan Interval { get; s
et; }     int ConcurrentThreads { get; set; } }
      public class RoundTheCodeSync : IRoundTheCodeSync {     public string Title { get; set; }
public TimeSpan Interval { get; set; }     public int ConcurrentThreads { get; set; } }
```

Reading our Custom Configuration

There are a number of ways we can read our custom configuration into our ASP.NET Core application.

#1: IConfiguration Instance

The first way is to pass the IConfiguration instance into our AppSettingsController constructor.

```
  // AppSettingsController.cs [Route("appsettings")] public class AppSettingsController : Controller
{     protected readonly IConfiguration _configuration;     public AppSettingsController(IConfigurat
ion configuration)     {         _configuration = configuration;     } }
```

With the IConfiguration instance, we can read the Title property.

To do this, we have created a FirstWay action and used the IConfiguration instance to return the title.

```
      // AppSettingsController.cs [Route("appsettings")] public class AppSettingsController : Contro
ller {     ...     [Route("first-way")]     public IActionResult FirstWay()     {         return Con
tent(_configuration.GetSection("RoundTheCodeSync").GetChildren().FirstOrDefault(config => config.Key
== "Title").Value, "text/plain");     } }
```

However this can be simplified by simply returning the GetValue method in our IConfiguration instance:

```
      // AppSettingsController.cs [Route("appsettings")] public class AppSettingsController : Contro
ller {     ...     [Route("first-way")]     public IActionResult FirstWay()     {         return Con
tent(_configuration.GetValue("RoundTheCodeSync:Title"), "text/plain");     } }
```

#2: Options Pattern

The second way is to add the custom configuration using the options pattern.

From there, we can pass the IOptions interface through dependency injection, using our custom class as the generic type.

The first thing we need to do is to add the service.

We do that by going into our Startup class and going to the ConfigureServices method.

```
  // Startup.cs public class Startup {     ...     // This method gets called by the runtime. Use th
is method to add services to the container.     public void ConfigureServices(IServiceCollection ser
vices)     {         ...         services.AddOptions().Bind(Configuration.GetSection("RoundTheCodeSy
nc"));         } }
```

Then we can inject it into our AppSettingsController constructor using our IOptions instance.

```
      // AppSettingsController.cs [Route("appsettings")] public class AppSettingsController : Contro
ller {     ...     protected readonly IOptions _roundTheCodeSyncOptions;     public AppSettingsContr
oller(..., IOptions roundTheCodeSyncOptions)     {          ...          _roundTheCodeSyncOptions = ro
undTheCodeSyncOptions;     }     ...     [Route("second-way")]     public IActionResult SecondWay()
     {          return Content(_roundTheCodeSyncOptions.Value.Title, "text/plain");     } }
```

#3 Add The Configuration as a Singleton

Another way is to bind the configuration file to our RoundTheCodeSync class, then add it to dependency injection as a singleton using the IRoundTheCodeSync interace.

In our Startup class, we need to add the service into DI:

```
            // Startup.cs public class Startup {     public Startup(IConfiguration configuration)     {
           Configuration = configuration;     }     public IConfiguration Configuration { get; }     //
This method gets called by the runtime. Use this method to add services to the container.     public
void ConfigureServices(IServiceCollection services)     {          ...                    services.Add
Singleton((serviceProvider) =>          {               return Configuration.GetSection("RoundTheCodeSy
nc").Get();          });               }     ... }
```

From there, we can inject our IRoundTheCode sync interface into our AppSettingsController constructor, and call the properties directly.

```
               // AppSettingsController.cs [Route("appsettings")] public class AppSettingsController :
Controller {     ...     protected readonly IRoundTheCodeSync _roundTheCodeSync;     public AppSetti
ngsController(..., IRoundTheCodeSync roundTheCodeSync)     {          ...          _roundTheCodeSync =
roundTheCodeSync;     }     ...     [Route("third-way")]     public IActionResult ThirdWay()     {
       return Content(_roundTheCodeSync.Title, "text/plain");     } }
```

Estimated Efforts

16 Hours **Size :** Medium

Impact

Mandatory

Help URL

No Record Found

Datapoint

Compilation TagSection

Module

Application & Platform Design

Reason for change

Syntax and Implementation changed

Code block	Line no.	File path
<compilation debug="true" targetFramework="4.5"> <assemblies> <add assembly="System.Runtime, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" /> </assemblies> </compilation>	77	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Web.config

Recommendation

set up ASP.NET Core Environment for Deploy/Publish

Setting up different host environments is an important and essential aspect of application development.

Setting up Environment Locally

These local settings are often necessary for debugging purposes or temporary in nature.

Such environment settings can be set using the command line or Visual studio runtime environment or using launch settings.

- Command-line
- Visual studio runtime environment or using
- Launch settings

Command-prompt

```
set ASPNETCORE_ENVIRONMENT=Staging dotnet run your-application
```

PowerShell command

```
$Env:ASPNETCORE_ENVIRONMENT = "Staging" dotnet run your-app
```

If using Mac OS please use the below commands to set up environment variable using Bash shell,

```
export ASPNETCORE_ENVIRONMENT=Staging
```

Set Environment for Local Debugging purpose

This can be done using Visual Studio or VScode editor easily,

- In VSCode Use .vscode/launch.json for setting the environment for debugging purposes.
- In Visual Studio use launchSettings.json or use Project->Properties->Debug->Enviornment Variable to set the environment for debugging purposes.

Setting up Environment Globally

The above-discussed techniques are used for setting the environment temporarily for executing the current process with the given environment configuration. These settings will not remain effective when the new session of the command prompt is used.

In such cases to set the value globally use either of the following approaches if using Windows,

Using Control Panel

Open the Control Panel > System > Advanced system settings

Select Environment Variables

Using Command Line

Open a command prompt or PowerShell command prompt in administrative mode.

Use the below command,

setx ASPNETCORE_ENVIRONMENT Production /M

The**/M** switch lets you set the environment variable at the system level.

PowerShell command

```
    [Environment]::SetEnvironmentVariable("ASPNETCORE_ENVIRONMENT", "Staging", "Machine")
```

In the above option "Machine" set the environment variable at the system level. If used "User" option, the environment variable will be set for the user account only.

Estimated Efforts

16 Hours **Size :** Medium

Impact

Mandatory

Help URL

No Record Found

Datapoint

CustomErrorsTagSection

Module

Security

Reason for change

Syntax and Implementations changed

Code block	Line no.	File path
<customErrors mode="Off" />	71	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Web.config

Recommendation

Custom Error

.net Core has completely re-invented how custom errors work. Partly because with no web.config, any XML configuration is out the window, and partly because the new "middleware pipeline" mostly does away with the plethora of MVC filters you had to use in the past.

Developer Exception Page

The developer exception page is more or less the error page you used to see in full .net framework if you had custom errors off. That is, you could see the stack trace of the error and other important info to help you debug the issue.

By default, new ASP.net core templates come with this turned on when creating a new project. You can check this by looking at the Configure method of your startup.cs file. It should look pretty close to the following.

```
      public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerF
actory) {     if (env.IsDevelopment())     {         app.UseDeveloperExceptionPage();     }    app.U
seMvcWithDefaultRoute(); }
```

Note that checking if the environment is development is so important! Just like the CustomErrors tag in the full framework, you only want to leak out your stacktrace and other sensitive info if you are debugging this locally or you specifically need to see what's going on for testing purposes. Under no circumstances should you just turn on the "UseDeveloperExceptionPage" middleware without first making sure you are working locally (Or some other specific condition).

Another important thing to note is that as always, the ordering of your middleware matters. Ensure that you are adding the Developer Exception Page middleware before you are going into MVC. Without it, MVC (Or any other middleware in your pipeline), can short circuit the process without it ever reaching the Developer Exception page code.

Exception Handler Page

ASP.net core comes with a catch all middleware that handles all exceptions and will redirect the user to a particular error page. This is pretty similar to the default redirect in the CustomErrors attribute in web.config or the HandleError attribute in full framework MVC. An important note is that this is an "exception" handler. Other status code errors (404 for example) do not get caught and redirected using this middleware.

The pattern is usually to show the developer error page when in the development environment, otherwise to redirect to the error page. So it might look a bit like this :

```
 public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory) {     if (env.IsDevelopment())     {
        app.UseDeveloperExceptionPage();     }     else     {         app.UseExceptionHandler("/home/error");     }
  app.UseMvcWithDefaultRoute(); }
```

You would then need to create an action to handle the error. One very important thing to note is that the ExceptionHandler will be called with the same HTTP Verb as the original request. e.g. If the exception happened on a Post request, then your handler at /home/error should be able to accept Posts.

What this means in practice is that you should not decorate your action with any particular HTTP verb, just allow it to accept them all.

Statuscode Pages

ASP.net core comes with an inbuilt middleware that allows you to capture other types of HTTP status codes (Other than say 500), and be able to show them certain content based on the status code.

There are actually two different ways to get this going. The first is :

```
 app.UseStatusCodePagesWithRedirects("/error/{0}");
```

Using "StatusCodePagesWithRedirects" you redirect the user to the status page. The issue with this is that the client is returned a 302 and not the original status code (For example a 404). Another issue is that if the exception is somewhere in your pipeline you are essentially restarting the pipeline again (And it could throw the same issue up).

The second option is :

```
app.UseStatusCodePagesWithReExecute("/error/{0}");
```

Using ReExecute, your original response code is returned but the content of the response is from your specified handler. This means that 404's will be treated as such by spiders and browsers, but it still allows you to display some custom content (Such as a nice 404 page for a user).

Custom Middleware

Remember that you can always create custom middleware to handle any exception/status code in your pipeline. Here is an example of a very simple middleware to handle a 404 status code.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory
) {     app.Use(async (context, next) =>     {         await next.Invoke();         //After going dow
n the pipeline check if we 404'd.         if (context.Response.StatusCode == StatusCodes.Status404No
tFound)         {             await context.Response.WriteAsync("Woops! We 404'd");         }     })
;    app.UseMvcWithDefaultRoute(); }
```

Estimated Efforts

16 Hours **Size :** Medium

Impact

Mandatory

Help URL

No Record Found

Datapoint

SessionStateSection

Module

Application & Platform Design

Reason for change

Session implementation is different from previous version of Asp.Net.

Code block	Line no.	File path
loginInfo = Session["LoginInfo"] as LoginResponse;	43	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignSubCategoriesController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	85	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignSubCategoriesController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	46	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignTypeMasterController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	94	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignTypeMasterController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	22	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\Sponsors\Controllers\SponsorsGiftController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	38	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\Sponsors\Controllers\SponsorsGiftController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	73	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\Sponsors\Controllers\SponsorsGiftController.cs
loginInfo = Session["LoginInfo"] as	36	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Are

LoginResponse;	36	EventManagement\Areas\S1M\Controllers\CampaignCategoryController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	82	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignCategoryController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	52	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignFormatController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	97	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignFormatController.cs
LoginResponse loginInfo = Session["LoginInfo"] as LoginResponse;	48	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\AccountMasterController.cs
LoginResponse loginInfo = Session["LoginInfo"] as LoginResponse;	93	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\AccountMasterController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	40	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\OpportunitiesController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	165	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\OpportunitiesController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	184	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\OpportunitiesController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	77	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\ManageSponsorController.cs
		D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M

loginInfo = Session["LoginInfo"] as LoginResponse;	264	endo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\ManageSponsorController.cs
if (Session["LoginInfo"] != null)	20	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\Sponsors\Controllers\SponsorController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	22	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\Sponsors\Controllers\SponsorController.cs
if (Session["LoginInfo"] != null)	55	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\Sponsors\Controllers\SponsorController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	57	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\Sponsors\Controllers\SponsorController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	46	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	73	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	234	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	284	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	297	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	794	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
		D:\Project\S1mAdminP

loginInfo = Session["LoginInfo"] as LoginResponse;	894	anel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	963	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	1090	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	1238	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	1354	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	1453	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
LoginResponse loginInfo = Session["LoginInfo"] as LoginResponse;	1813	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
loginInfo = Session["LoginInfo"] as LoginResponse;	1830	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\CampaignController.cs
LoginResponse log = (LoginResponse)Session["LoginInfo"];	118	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\S1MUserController.cs
LoginResponse log = (LoginResponse)Session["LoginInfo"];	135	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Areas\S1M\Controllers\S1MUserController.cs
Session["LoginInfo"] = log;	39	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Controllers\HomeController.cs

Recommendation

There are two Session state in asp.net core, In-Proc (In-memory) and Out-Proc (Distributed session). But in asp.net core you have to do some additional work to use session object in application, you have to add session in middleware pipeline Like earlier Asp.Net framework, session object is not available by default, you need to install and register in http pipeline. To use session in Asp.net Core Application you need to install **Microsoft.AspNetCore.Session package** from NuGet Package Manager.

```
install Microsoft.AspNetCore.Session
```

Step 1

Open startup.cs file and inside ConfigureServices method register the AddSession() method, and add UseSession method to IApplicationBuilder instance in Configure method.

```
   public void ConfigureServices(IServiceCollection services) { services.AddSession(); }
```

Alternatively, you also can set the optional parameter session time out value at the time of registering session object in middleware pipeline.

```
services.AddSession(options => { options.IdleTimeout = TimeSpan.FromMinutes(1); });
```

IdleTimeout" is actually indicates session timeout, it says after that specified time duration, the session will get expired, if there is not activity from that user session. We also can set session cookie name, by default the cookie name is .AspNetCore.Session.

```
services.AddSession(options => { options.Cookie.Name = ".WebTrainingRoom.Session"; options.IdleTimeo
ut = TimeSpan.FromMinutes(1); options.Cookie.IsEssential = true;  });
```

Step 2
Now in Configure method call UseSession() method

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env) { app.UseSession(); }
```

Step 3
Now install Microsoft.AspNetCore.Session utility from Nugget

Set and get Session values Setting and retrieving values in Asp.net Core Session object In ASP.NET Core Session object there are 3 methods to set the session value, which are Set, SetInt32 and SetString.

```
HttpContext.Session.SetString(""sessionKeyString"", ""WebTrainingRoom""); HttpContext.Session.SetInt
32(""sessionKeyInt"", 10);
```

And three methods for retrieving values from session , which are Get, GetInt32 and GetString

```
string _sessionStringvalue = HttpContext.Session.GetString(""sessionKeyString""); int? _sessionIntVa
lue = HttpContext.Session.GetInt32(""sessionKeyInt"");
```

So far we have seen how to use session object to store simple information, but there is no straight way to store complex object in asp.net core session. Now we learn how to store complex object in session

Asp net core session complex objects example Let's see how to store complex object in Asp.net core session! So we have already understood there is no straight way of storing complex data in session like we used to do in earlier asp.net version But, there is way to store complex object in session, we can convert complex object into Json String, then store them in session object using the same SetString function, here is an working example of how we can store complex data in asp.net core session. We have to create an additional class to add any object current Session, look at the example bwlow.

```
using Microsoft.AspNetCore.Http; using Newtonsoft.Json; public static class SessionHelper {     publ
ic static void SetObjectAsJson(this ISession session, string key, object value) {         session.Se
tString(key, JsonConvert.SerializeObject(value));     }     public static T GetObjectFromJson(this I
Session session, string key) {         var value = session.GetString(key);         return value == n
ull ?     default(T):          JsonConvert.DeserializeObject(value);     } }
```

Now let call the above methods to add and retrieve complex object in session. In following example we add a student object list in session and retrieve from session

Session handling in Controller Once user hit the login button on web page data gets submitted to action result IActionResult index(UserModel model). If authenticated successfully, then user object gets stored in session and UI gets redirected to controlpanel. In control panel we check the session, if user object found we display user information on view, otherwise throw out user to login page or unauthorised page.

```
  public class userController: Controller {     IAuthService _authService;     public userController
(IAuthService authservice) {         _authService = authservice;      } [HttpPost] public IActionResu
lt index(UserModel model) {          User u = _authService.GetUser(model.Username, model.Password);
       if (u != null) {           SessionHelper.SetObjectAsJson(HttpContext.Session, ""
  userObject "", u);           return RedirectToAction(""              controlpanel "");          }
        return View(model);     }    public IActionResult controlpanel() {         User user = Sessi
onHelper.GetObjectFromJson(HttpContext.Session, ""         userObject "");         if (user == null)
{ //throw out; }              // do whatever you want with user object.          ViewBag.CurrentUs
er = user;              return View();          }      }
```

Remove Session Key Now you come across situation when you want to remove some session key explicitly from current session, as we have seen in above example that we can add any number of keys in session object, so here we see how to remove any key from session object.

```
      public IActionResult Logout()     { HttpContext.Session.Remove(""userObject""); return View();
    }
```

We also can remove all keys from session, which is essential for successful sign-out.

```
  public IActionResult Logout()     { HttpContext.Session.Clear(); HttpContext.SignOutAsync(); retur
n View();     }
```

Estimated Efforts

26 Hours **Size :** Large

Impact

Mandatory

Help URL

No Record Found

Datapoint

Connection Strings Section

Module

Application & Platform Desing

Reason for change

Synatax and Implementation changed

Code block	Line no.	File path
<add name="defaultcon***	11	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\obj\Release\TransformWebConfig\transformed\Web.config
<add name="s1mmarketi***	17	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\obj\Release\TransformWebConfig\transformed\Web.config
<add name="defaultcon***	11	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\obj\Release\CSAutoParameterize\original\Web.config
<add name="s1mmarketi********************************		D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.

***	17	EventManagement\obj\Release\CSAutoParameterize\original\Web.config
<add name="defaultcon***	11	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\obj\Release\TransformWebConfig\original\Web.config
<add name="s1mmarketi***	17	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\obj\Release\TransformWebConfig\original\Web.config
<add name="defaultcon***	11	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Web.config
<add name="s1mmarketi***	14	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.EventManagement\Web.config

Recommendation

How to read connection string in .Net Core:

Microsoft has replaced System.Configuration class with IConfiguration interface in .Net Core 2.0 & above.

IConfiguration

The IConfiguration is an interface for .Net Core 2.0.

The IConfiguration interface need to be injected as dependency in the Controller and then later used throughout the Controller.

The IConfiguration interface is used to read Settings and Connection Strings from AppSettings.json file.
Namespaces You will need to import the following namespace.

using Microsoft.Extensions.Configuration;

Adding the AppSettings.json file

In order to add AppSettings.json file, right click on the Project in Solution Explorer. Then click Add, then New Item and then choose App Settings File option and click Add button.

Once the File is created, it will have a DefaultConnection, below that a new Connection String entry is added.

```
{ "ConnectionStrings": {     "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=_CHANGE_ME;Trusted_Connection=True;MultipleActiveResultSets=true",     "MyConn": "Data Source=.\\SQL2017;Initial Catalog=AjaxSamples;Integrated Security=true" } }
```

Reading Connection String from AppSettings.json file using IConfiguration interface

In the below example, the IConfiguration is injected in the Controller and assigned to the private property Configuration. Then inside the Controller, the Connection String is read from the AppSettings.json file using the GetConnectionString function

```
  public class HomeController : Controller {     private IConfiguration Configuration;     public HomeController(IConfiguration _configuration)     {         Configuration = _configuration;     }     public IActionResult Index()     {         string connString = this.Configuration.GetConnectionString("MyConn");         return View();     } }
```

Estimated Efforts

128 Hours **Size :** Large

Impact

Mandatory

Help URL

No Record Found

Datapoint

SessionStateSection

Module

Application & Platform Design

Reason for change

Session implementation is different from previous version of Asp.Net.

Code block	Line no.	File path
LoginResponse sessionValue = HttpContext.Current.Session["LoginInfo"] as LoginResponse;	19	D:\Project\S1mAdminPanel\S1mAdminPanel-kendo_15-09-2016\S1M.BAL\AssetMappingBL.cs

Recommendation

There are two Session state in asp.net core, In-Proc (In-memory) and Out-Proc (Distributed session). But in asp.net core you have to do some additional work to use session object in application, you have to add session in middleware pipeline Like earlier Asp.Net framework, session object is not available by default, you need to install and register in http pipeline. To use session in Asp.net Core Application you need to install **Microsoft.AspNetCore.Session package** from NuGet Package Manager.

```
install Microsoft.AspNetCore.Session
```

Step 1

Open startup.cs file and inside ConfigureServices method register the AddSession() method, and add UseSession method to IApplicationBuilder instance in Configure method.

```
    public void ConfigureServices(IServiceCollection services) { services.AddSession(); }
```

Alternatively, you also can set the optional parameter session time out value at the time of registering session object in middleware pipeline.

```
services.AddSession(options => { options.IdleTimeout = TimeSpan.FromMinutes(1); });
```

IdleTimeout" is actually indicates session timeout, it says after that specified time duration, the session will get expired, if there is not activity from that user session. We also can set session cookie name, by default the cookie name is .AspNetCore.Session.

```
services.AddSession(options => { options.Cookie.Name = ".WebTrainingRoom.Session"; options.IdleTimeo
ut = TimeSpan.FromMinutes(1); options.Cookie.IsEssential = true;  });
```

Step 2
Now in Configure method call UseSession() method

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env) { app.UseSession(); }
```

Step 3
Now install Microsoft.AspNetCore.Session utility from Nugget

Set and get Session values Setting and retrieving values in Asp.net Core Session object In ASP.NET Core Session object there are 3 methods to set the session value, which are Set, SetInt32 and SetString.

```
HttpContext.Session.SetString(""sessionKeyString"", ""WebTrainingRoom""); HttpContext.Session.SetInt
32(""sessionKeyInt"", 10);
```

And three methods for retrieving values from session , which are Get, GetInt32 and GetString

```
string _sessionStringvalue = HttpContext.Session.GetString(""sessionKeyString""); int? _sessionIntVa
lue = HttpContext.Session.GetInt32(""sessionKeyInt"");
```

So far we have seen how to use session object to store simple information, but there is no straight way to store complex object in asp.net core session. Now we learn how to store complex object in session

Asp net core session complex objects example Let’s see how to store complex object in Asp.net core session! So we have already understood there is no straight way of storing complex data in session like we used to do in earlier asp.net version But, there is way to store complex object in session, we can convert complex object into Json String, then store them in session object using the same SetString function, here is an working example of how we can store complex data in asp.net core session. We have to create an additional class to add any object current Session, look at the example bwlow.

```
using Microsoft.AspNetCore.Http; using Newtonsoft.Json; public static class SessionHelper {      publ
ic static void SetObjectAsJson(this ISession session, string key, object value) {          session.Se
tString(key, JsonConvert.SerializeObject(value));      }      public static T GetObjectFromJson(this I
Session session, string key) {          var value = session.GetString(key);          return value == n
ull ?      default(T):          JsonConvert.DeserializeObject(value);      } }
```

Now let call the above methods to add and retrieve complex object in session. In following example we add a student object list in session and retrieve from session

Session handling in Controller Once user hit the login button on web page data gets submitted to action result IActionResult index(UserModel model). If authenticated successfully, then user object gets stored in session and UI gets redirected to controlpanel. In control panel we check the session, if user object found we display user information on view, otherwise throw out user to login page or unauthorised page.

```
  public class userController: Controller {      IAuthService _authService;     public userController
(IAuthService authservice) {          _authService = authservice;      } [HttpPost] public IActionResu
lt index(UserModel model) {          User u = _authService.GetUser(model.Username, model.Password);
      if (u != null) {            SessionHelper.SetObjectAsJson(HttpContext.Session, ""
  userObject "", u);             return RedirectToAction(""            controlpanel "");          }
       return View(model);      }    public IActionResult controlpanel() {          User user = Sessi
onHelper.GetObjectFromJson(HttpContext.Session, ""          userObject "");          if (user == null)
{ //throw out; }             // do whatever you want with user object.          ViewBag.CurrentUs
er = user;           return View();          }      }
```

Remove Session Key Now you come across situation when you want to remove some session key explicitly from current session, as we have seen in above example that we can add any number of keys in session object, so here we see how to remove any key from session object.

```
      public IActionResult Logout()      { HttpContext.Session.Remove(""userObject""); return View();
    }
```

We also can remove all keys from session, which is essential for successful sign-out.

```
  public IActionResult Logout()     { HttpContext.Session.Clear(); HttpContext.SignOutAsync(); retur
n View();     }
```

Estimated Efforts

1 Hours **Size :** Small

Impact

Mandatory

Help URL

No Record Found

Datapoint

Connection Strings Section

Module

Application & Platform Desing

Reason for change

Synatax and Implementation changed

Code block	Line no.	File path
<add name="s1mmarketi******************************* ***	7	D:\Project\S1mAdminPanel\S1mAdminPanel-kende_15.09.2016\S1M

** *******	endo_15-09-2016\SIM. DAL\App.Config

Recommendation

How to read connection string in .Net Core:

Microsoft has replaced System.Configuration class with IConfiguration interface in .Net Core 2.0 & above.

IConfiguration

The IConfiguration is an interface for .Net Core 2.0.

The IConfiguration interface need to be injected as dependency in the Controller and then later used throughout the Controller.

The IConfiguration interface is used to read Settings and Connection Strings from AppSettings.json file.
Namespaces You will need to import the following namespace.

using Microsoft.Extensions.Configuration;

Adding the AppSettings.json file

In order to add AppSettings.json file, right click on the Project in Solution Explorer. Then click Add, then New Item and then choose App Settings File option and click Add button.

Once the File is created, it will have a DefaultConnection, below that a new Connection String entry is added.

```
{ "ConnectionStrings": {     "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=_CHANGE_M
E;Trusted_Connection=True;MultipleActiveResultSets=true",     "MyConn": "Data Source=.\\SQL2017;Init
ial Catalog=AjaxSamples;Integrated Security=true" } }
```

Reading Connection String from AppSettings.json file using IConfiguration interface

In the below example, the IConfiguration is injected in the Controller and assigned to the private property Configuration. Then inside the Controller, the Connection String is read from the AppSettings.json file using the GetConnectionString function

```
  public class HomeController : Controller {     private IConfiguration Configuration;     public Ho
meController(IConfiguration _configuration)     {         Configuration = _configuration;     }
public IActionResult Index()     {         string connString = this.Configuration.GetConnectionStrin
g("MyConn");         return View();     } }
```

Estimated Efforts

16 Hours **Size :** Medium

Impact

Mandatory

Help URL

No Record Found

Datapoint

GAC Dependency

Module

Application & Platform Design

Reason for change

Syntax and Implementation changed

Recommendation

GAC Dependency

.NET Core 2.0 introduces the "Runtime package store":

Starting with .NET Core 2.0, it's possible to package and deploy apps against a known set of packages that exist in the target environment. The benefits are faster deployments, lower disk space usage, and improved startup performance in some cases.

This feature is implemented as a runtime package store, which is a directory on disk where packages are stored (typically at /usr/local/share/dotnet/store on macOS/Linux and C:/Program Files/dotnet/store on Windows). Under this directory, there are subdirectories for architectures and target frameworks. The file layout is similar to the way that NuGet assets are laid out on disk:

```
    \dotnet     \store          \x64               \netcoreapp2.0                    \microsoft.applicatio
ninsights                    \microsoft.aspnetcore                    ...          \x86          \netcor
eapp2.0                    \microsoft.applicationinsights                    \microsoft.aspnetcore
```

```
. . .
```

A target manifest file lists the packages in the runtime package store. Developers can target this manifest when publishing their app. The target manifest is typically provided by the owner of the targeted production environment.

Preparing a runtime environment

The administrator of a runtime environment can optimize apps for faster deployments and lower disk space use by building a runtime package store and the corresponding target manifest.

The first step is to create a package store manifest that lists the packages that compose the runtime package store. This file format is compatible with the project file format (csproj).

```
    <'Project Sdk="Microsoft.NET.Sdk">   <'ItemGroup>     <'PackageReference Include="NUGET_PACKAGE" Version="VERSION" /> '   <'/Project>
```

Example

The following example package store manifest (packages.csproj) is used to add Newtonsoft.Json and Moq to a runtime package store:

```
      <'Project Sdk="Microsoft.NET.Sdk">   <'ItemGroup>     <'PackageReference Include="Newtonsoft.Json" Version="10.0.3" />     <'PackageReference Include="Moq" Version="4.7.63" />   <'/ItemGroup> <'/Project>
```

Provision the runtime package store by executing dotnet store with the package store manifest, runtime, and framework:

```
            dotnet store --manifest --runtime --framework
```

Example

```
               dotnet store --manifest packages.csproj --runtime win10-x64 --framework netcoreapp2.0 --framework-version 2.0.0
```

You can pass multiple target package store manifest paths to a single dotnet store command by repeating the option and path in the command.

By default, the output of the command is a package store under the .dotnet/store subdirectory of the user's profile. You can specify a different location using the –output option. The root directory of the store contains a target manifest artifact.xml file. This file can be made available for download and be used by app authors who want to target this store when publishing.

Example

The following artifact.xml file is produced after running the previous example. Note that Castle.Core is a dependency of Moq, so it's included automatically and appears in the artifacts.xml manifest file.

```
                  <'StoreArtifacts>   <'Package Id="Newtonsoft.Json" Version="10.0.3" />   <'Package Id="Castle.Core" Version="4.1.0" />   <'Package Id="Moq" Version="4.7.63" /> <'/StoreArtifacts>
```

Estimated Efforts

20 Hours **Size :** Large

Impact

Mandatory

Help URL

No Record Found

Datapoint

Secure Socket Layer (SSL)

Module

Security

Reason for change

Synax and Implementation changed

Recommendation

Configure certificate authentication in ASP.NET Core

Microsoft.AspNetCore.Authentication.Certificate contains an implementation similar to Certificate Authentication for ASP.NET Core. Certificate authentication happens at the TLS level, long before it ever gets to ASP.NET Core. More accurately, this is an authentication handler that validates the certificate and then gives you an event where you can resolve that certificate to a ClaimsPrincipal.

Configure your server for certificate authentication, be it IIS, Kestrel or whatever else you are using.

Proxy and load balancer scenarios

Certificate authentication is a stateful scenario primarily used where a proxy or load balancer does not handle traffic between clients and servers. If a proxy or load balancer is used, certificate authentication only works if the proxy or load balancer:

- Handles the authentication.
- Passes the user authentication information to the app (for example, in a request header), which acts on the authentication information.

An alternative to certificate authentication in environments where proxies and load balancers are used is Active Directory Federated Services (ADFS) with OpenID Connect (OIDC).

Get started
Acquire an HTTPS certificate, apply it, and configure your server to require certificates.

In your web app, add a reference to the Microsoft.AspNetCore.Authentication.Certificate package. Then in the Startup.ConfigureServices method, call

services.AddAuthentication(CertificateAuthenticationDefaults.AuthenticationScheme).AddCertificate(...); with your options, providing a delegate for OnCertificateValidated to do any supplementary validation on the client certificate sent with requests. Turn that information into a ClaimsPrincipal and set it on the context.Principal property.

If authentication fails, this handler returns a 403 (Forbidden) response rather a 401 (Unauthorized), as you might expect. The reasoning is that the authentication should happen during the initial TLS connection. By the time it reaches the handler, its too late. There is no way to upgrade the connection from an anonymous connection to one with a certificate.

Also add app.UseAuthentication(); in the Startup.Configure method. Otherwise, the HttpContext.User will not be set to ClaimsPrincipal created from the certificate. For example:

```
public void ConfigureServices(IServiceCollection services) {     services.AddAuthentication(
 CertificateAuthenticationDefaults.AuthenticationScheme)         .AddCertificate()         // Adding
an ICertificateValidationCache results in certificate auth caching the results.         // The defau
lt implementation uses a memory cache.         .AddCertificateCache();    // All other service confi
guration } public void Configure(IApplicationBuilder app, IHostingEnvironment env) {     app.UseAuth
entication();    // All other app configuration }
```

Configure certificate validation
The CertificateAuthenticationOptions handler has some built-in validations that are the minimum validations you should perform on a certificate. Each of these settings is enabled by default.

AllowedCertificateTypes = Chained, SelfSigned, or All (Chained | SelfSigned)
Default value: CertificateTypes.Chained

This check validates that only the appropriate certificate type is allowed. If the app is using self-signed certificates, this option needs to be set to CertificateTypes.All or CertificateTypes.SelfSigned.

ValidateCertificateUse
Default value: true

This check validates that the certificate presented by the client has the Client Authentication extended key use (EKU), or no EKUs at all. As the specifications say, if no EKU is specified, then all EKUs are deemed valid.

ValidateValidityPeriod
Default value: true

This check validates that the certificate is within its validity period. On each request, the handler ensures that a certificate that was valid when it was presented has not expired during its current session.

RevocationFlag
Default value: X509RevocationFlag.ExcludeRoot

A flag that specifies which certificates in the chain are checked for revocation.

Revocation checks are only performed when the certificate is chained to a root certificate.

RevocationMode
Default value: X509RevocationMode.Online

A flag that specifies how revocation checks are performed.

Specifying an online check can result in a long delay while the certificate authority is contacted.

Revocation checks are only performed when the certificate is chained to a root certificate.

Can I configure my app to require a certificate only on certain paths?
This is not possible. Remember the certificate exchange is done at the start of the HTTPS conversation, its done by the server before the first request is received on that connection so its not possible to scope based on any request fields.

Estimated Efforts

4 Hours **Size :** Small

Impact

Mandatory

Help URL

No Record Found