Building simple multilingual ASP.NET Core website

Marcin Kawalerowicz / Thursday, October 13, 2016

Introduction

In this tutorial we will create new multilingual website in ASP.NET Core and publish it to IIS. Version 1.0 of ASP.NET Core was released in June 2016, so it's quite new tool. Main feature of it is that we can develop and run our apps cross-platform on Windows, Linux and Mac. Today we're going to concentrate on Windows. ASP.NET Core contains some differences compared to ASP.NET MVC 5, so it's a good idea to start with something simple and our website, which consist of two webpages, both in three languages, it's good offer to start with.

Creating .NET Core environment on Windows

To start with ASP.NET Core, we need to have Visual Studio 2015 with Visual Studio Update 3 installed. Skip this step, if you already installed both of them. If not, you can get Visual Studio Community for free here: Visual Studio Community 2015 Download and Visual Studio Update 3 here: Visual Studio Update 3 Download. During installation of Visual Studio Community 2015, just select the default installation.

Installing .NET Core 1.0 for Visual Studio and .NET Core Windows Server Hosting

Now we need to install .NET Core 1.0 for Visual Studio and .NET Core Windows Server Hosting, so we will be able to build and publish our website. You can get .NET Core 1.0 for Visual Studio here: .NET Core 1.0 for Visual Studio Download and .NET Core Windows Server Hosting here: .NET Core Windows Server Hosting Download.

Creating a website

If we already have installed all these necessaries, we can proceed to create a new website. To do so, open Visual Studio 2015, go to **File/New/Project** and choose **Visual C#/Web/ASP.NET Core Web Application (.NET Core)**. Name it **NETCoreWebsite** (fig. 1).

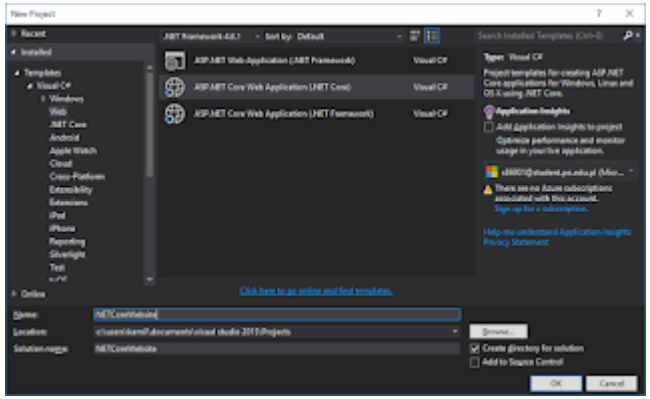


Figure 1 – creating template for ASP .NET Core application

In the next window we need to choose type of template together with type of authentication. In our case, it will be respectively **Web Application** and **No authentication**. **Host in the cloud** option should be unchecked (fig. 2).

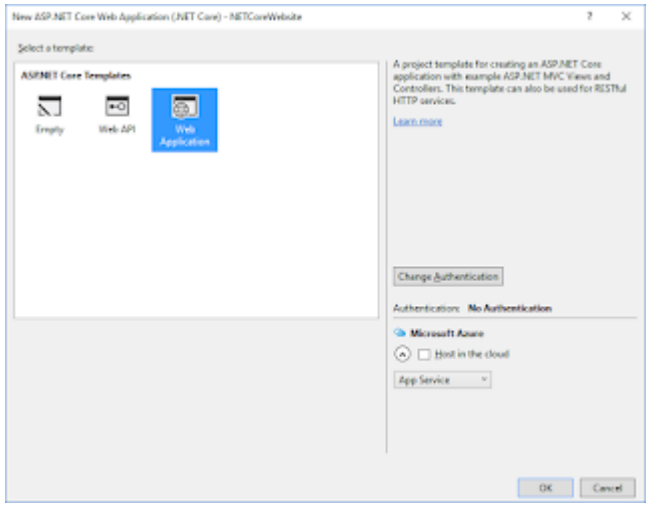


Figure 2 – choosing right template and type of authentication

New ASP.NET Core project has been just created. Moreover, we can display it in our web browser. To do so, click on **IIS Express** button on the navigation bar. After few second default website should appear in our web browser. We can switch between all items on navigation bar.

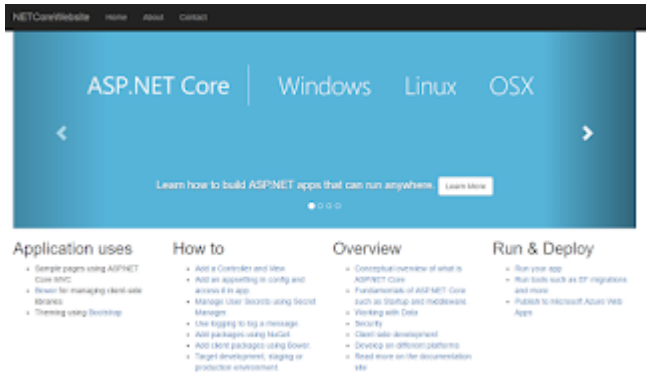


Figure 3 – default website made while creating new ASP.NET Core project

Adding webpages and static files

Now we move on to create our own website. All directories and files which will be mentioned in this tutorial are placed in **src/NETCoreWebsite** in **Solution Explorer**. First of all, we should remove unnecessary files. To do so, go to **Views/Home** and delete all three webpages placed there. After that go to **wwwroot/images** and delete all images that directory contains.
Now it's time to add our webpages to project. Go to **Views/Home**, right click on it and choose **Add/New item** (fig. 4).

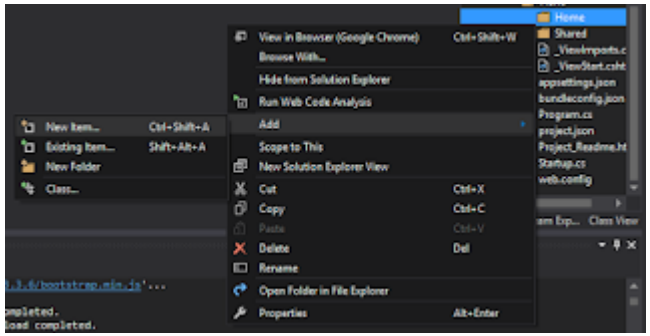


Figure 4 – adding new webpage to project

In new window choose **.NET Core/ASP.NET/MVC View Page**. Name it **FirstWebpage.cshtml** (fig. 5).

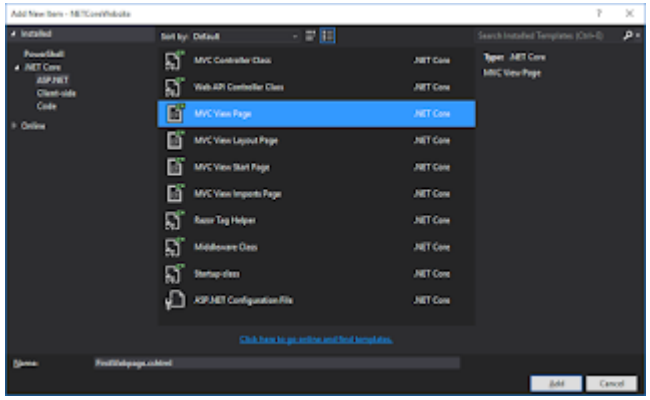


Figure 5 – adding new webpage to project, continuation

Our webpage has just been created. Repeat that step for **SecondWebpage.cshtml**.

Now we're going to fill .cshtml files we've created in last step with HTML code. **IMPORTANT:** those .cshtml files should contain only content of <body> tag **without** declaration of shared elements (like navigation bar or footer), references to CSS files, fonts and <script> tags. <body> tags shouldn't be included as well.

It's time to add static files like images, CSS or JavaScript to our project. Few steps ago we deleted unnecessary images from **wwwroot/images**. Now we're going to add our images right to this directory. Right click on it and choose **Open Folder in File Explorer** (fig. 6). That will open images directory in File Explorer and now all we have to do is simply copy our images here. **NOTE:** Remember to add "~/" at beginning of every image path.

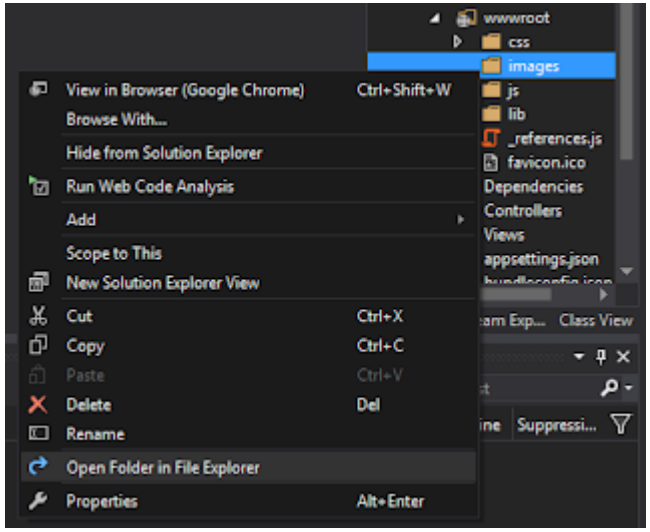


Figure 6 – opening directory in File Explorer to easily add new items to it

In very similar way we can add CSS and JavaScript files. We just have to add them to **wwwroot/css** or **wwwroot/js**.

ASP.NET Core using MVC pattern, which means that Controllers are responsible for display our Views to end users. To display our webpages, we need to edit **HomeController.cs** placed in **Controllers** directory.

In **HomeController.cs** delete methods About() and Contact(). Then copy Index() method and paste it just below original Index() method. After that change "Index" to "FirstWebpage" in first method and to "SecondWebpage" in second method. Those methods only return View which allow to display our webpages in browser. After complete this step, our **HomeController.cs** class should look like this:

```
public class HomeController : Controller
    {
        public IActionResult FirstWebpage()
        {
            return View();
        }
        public IActionResult SecondWebpage()
        {
            return View();
        }
        public IActionResult Error()
        {
            return View();
        }
    }
```

Go to **Startup.cs** class and find method called Configure. In method body we will find code similar to this:

```
app.UseMvc(routes =>
            {
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Home}/{action=Index}/{id?}");
            });
```

Change `{action=Index}` to `{action=FirstWebpage}` so our chosen webpage will be display by default.
In next step we will add references to our CSS and JavaScript files and extra shared files.

Extracting shared files

To extract shared files, we have to edit **_Layout.cshtml** file. A layout page contains the structure and shared content of website. When a web page (content page) is linked to a layout page, it will be displayed according to the layout page (template).

The layout page is just like a normal web page, except from a call to the @RenderBody() method where the content page will be included.

Open **_Layout.cshtml**. As we can see, our <head> tag and references to CSS and JavaScript files are defined right here.

Let's start with CSS. Find environment called **Development**. Note that there are two environments called like this, one in <head> and one in <body>. We want to add reference to CSS file, so of course we're going to change code of <head> **Development** environment.

In **Development** we will find `<link rel="stylesheet" href="~/css/site.css" />`. We want to add reference to our own CSS file, so we need to simply change `site.css` to name of our CSS file.

We need also to add reference to font that would be used on our website. Add `<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Open+Sans:400,600" />` just below CSS files reference.

It's very important to copy contain of **Development** environment to **Staging,Production** environment (it can be found just below), so we will be able to use added references after publishing our project.

Now we will add reference to JavaScript files.

In declaration of <body> find environment called **Development**. There we will find line looks like this:

```
<script src="~/js/site.js" asp-append-version="true"></script>
```

Just like before, all we have to do is to change `site.js` to name of our JavaScript file. In that case, it would be `popups.js`.

Again, copy content of **Development** environment to **Staging, Production** environment.

Last point of editing **_Layout.cshtml** is to define shared elements of our website. In our case, it will be navigation bar and footer, which are the same for both webpages. We just need to replace default navigation bar code with our o navigation bar code and repeat same step for footer.

If it has been done, we can click **IIS Express** button and display our website in browser.

Using resources to localize website

Using recourses is quick and easy way to localize our website. You can read more about it here: Globalization and localization.
Before we add any resources, we need to implement a strategy to select the language for each request. To do so, go to **Startup.cs** and find method called **ConfigureServices**. Replace method body with code like below:

```
      services.AddLocalization(options => options.ResourcesPath =
"Resources");
      services.AddMvc()
        .AddViewLocalization(LanguageViewLocationExpanderFormat.Suffix)
        .AddDataAnnotationsLocalization();
```

We added the localization services to the services container and set resources path to **Resources** (we will create that in a moment). It will also allow us to base localization on the view file suffix.
We want to localize our website in three languages: English, Polish and German. Default language is English. In **Startup.cs** find method called **Configure** and add to it code like below:

```
var supportedCultures = new[]
            {
                new CultureInfo("en-US"),
                new CultureInfo("pl-PL"),
                new CultureInfo("de-DE")
            };
            app.UseRequestLocalization(new RequestLocalizationOptions
            {
                DefaultRequestCulture = new RequestCulture("en-US"),
                SupportedCultures = supportedCultures,
                SupportedUICultures = supportedCultures
            });
```

At the very beginning of **Startup.cs** add following code:

```
using System.Globalization;
using Microsoft.AspNetCore.Mvc.Razor;
using Microsoft.AspNetCore.Localization;
```

Now we will add resources to localize our website. Right click on **src/NETCoreWebsite** in **Solution Explorer** and choose **Add/New Folder** (fig. 7). Name it **Resources.**

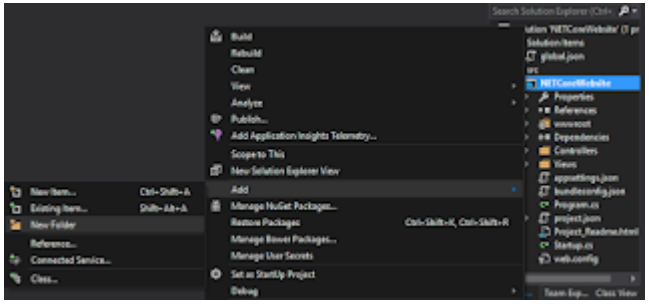
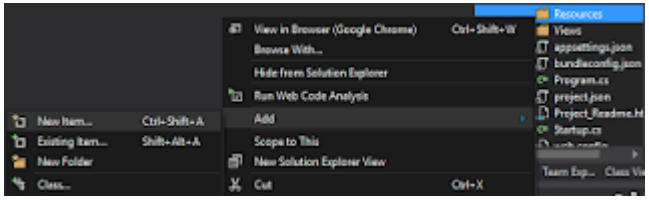


Figure 7 – adding new directory which will contain our resource files

After that, right click on **Resources** directory and choose **Add/New Item** (fig. 8).



In new window choose **.NET Core/Code/Resources File**. Name it **Views.Home.FirstWebpage.en.resx** (fig. 9).

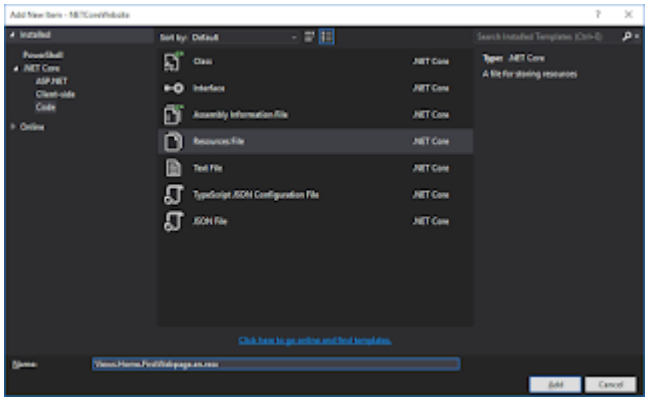


Figure 9 – adding new resources file

Resources file for **FirstWebpage.cshtml** in English language has just been created. Repeat this step for Polish and German language (remember to change **en** to respectively **pl** and **de**). After that we should have 3 resources files in our **Resources** directory.

Now we need to create resources files for **SecondWebpage.cshtml**. Repeat above step three times (for each language). Remember to change **FirstWebpage** to **SecondWebpage** in name of the resources file and to change suffixes.

We need also to create resources files for **_Layout.cshtml**. As you could notice, name of resources file is path to proper .cshtml file plus language suffix. Because **_Layout.cshtml** isn't placed in **Home** directory, but in **Shared** directory, our resources file name for English language will be **Views.Shared._Layout.en.resx**. Repeat this step for Polish and German language.

We can move on to localize our website. Add following code at the very beginning of **FirstWebpage.cshtml, SecondWebpage.cshtml** and **_Layout.cshtml**:

```
@using Microsoft.AspNetCore.Mvc.Localization
@inject IViewLocalizer Localizer
```

To localize any string in our code, we need to replace chosen string in .cshtml file with `@Localizer["String or it's ID"]`. It's good practice to replace short sentences and one word strings with `@Localizer["String"]` and long sentences with `@Localizer["ID"]`. For example, if we want to localize *Contact us*, we should write `@Localizer["Contact us"]`, but if we want to localize *This tutorial will teach you building and publishing your multilanguage website on Windows using ASP .NET Core*, better write `@Localizer["About tutorial"]`.

Let's assume that we used `@Localizer["About tutorial"]` in our code. To translate it to other language, open proper resources file, in **Key** write *About tutorial* and in **Value** translated sentence. That's all.

We can choose proper language in navigation bar of our website. To make it working, we need to code buttons like this in **_Layout.cshtml**:

```
<li><a href="?culture=pl-PL">PL</a></li>
<li><a href="?culture=en-US">EN</a></li>
<li><a href="?culture=de-DE">DE</a></li>
```

2 Log in to like this post!

ASP.NET Core, ASP. NET MVC, web pages, News