



# BUILDING MULTI CULTURAL WEB APPLICATION

ASP.NET Core 2.1 Razor Pages

## Abstract

Building multi-cultural web application is a challenge for many developers. This article addresses most known localization problems and provides solutions for them.

Eng. Ziya Mollamahmut  
info@ziyad.info

## Contents

Building Multi Culture Web Application Using ASP.NET Core 2.1 Razor Pages .....	0
Introduction .....	2
Create Standard Project .....	3
Route Value Based Localization Infrastructure .....	4
Define Culture Route Model Convention .....	4
Route Value Request Culture Provider .....	7
Create Language Dropdown Navigation .....	11
Localizing Views .....	14
Localizing Data Annotations .....	18
Localizing Property Display Name .....	18
Localizing Validation Messages .....	23
Adding Client Side Validation .....	29
Client Side Validation for Non-English Cultures .....	30

# Building Multi Culture Web Application Using ASP.NET Core 2.1 Razor Pages

## Introduction

In this tutorial, we will learn how to build multi-culture web application using ASP.NET Core 2.1 Razor Pages. We will start from the standard web application template without user login, then we will create localization infrastructure based on route value (e.g. <http://www.example.com/en/articles>). We will use resource files for localizing views, data annotations, model binding error messages and client side validation.

While building the application, we will solve some cultural related issues, like client side validation, decimal numbers validation, Calendar and numbering systems related issues.

At the end, we will have a simple web page with one form to submit DateTime and decimal values in different cultures.

MyTrips

Home

About

Contact

Arabic (العربية) ▾

ال جولات السياحية

يرجى إدخال معلومات الرحلة السياحية

الوجهة السياحية

تركيا - استانبول

تاريخ الذهاب

10-Aug-18 08:08

سعر البطاقة

123.45

إرسال

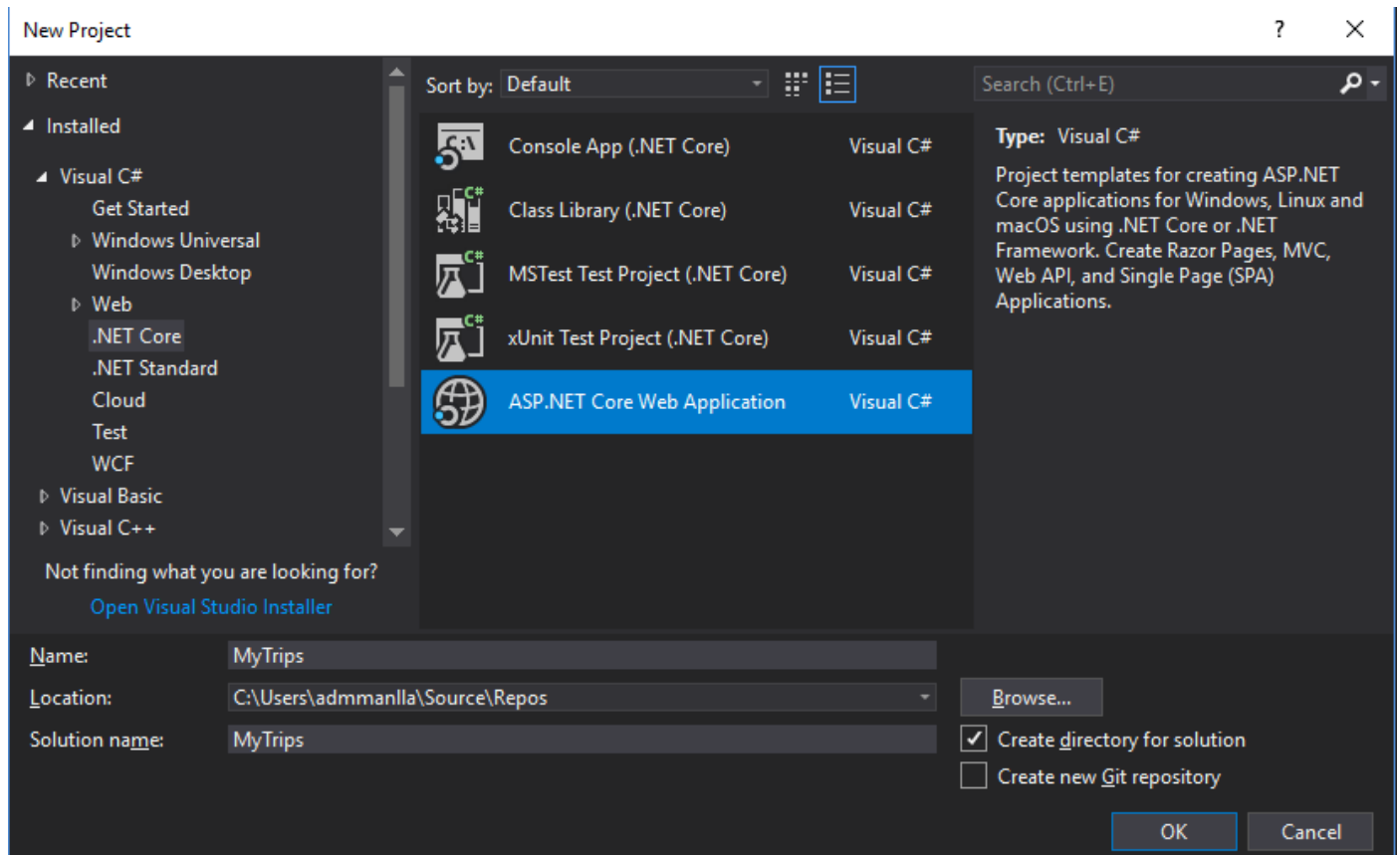
© 2018 - MyTrips

## Prerequisites:

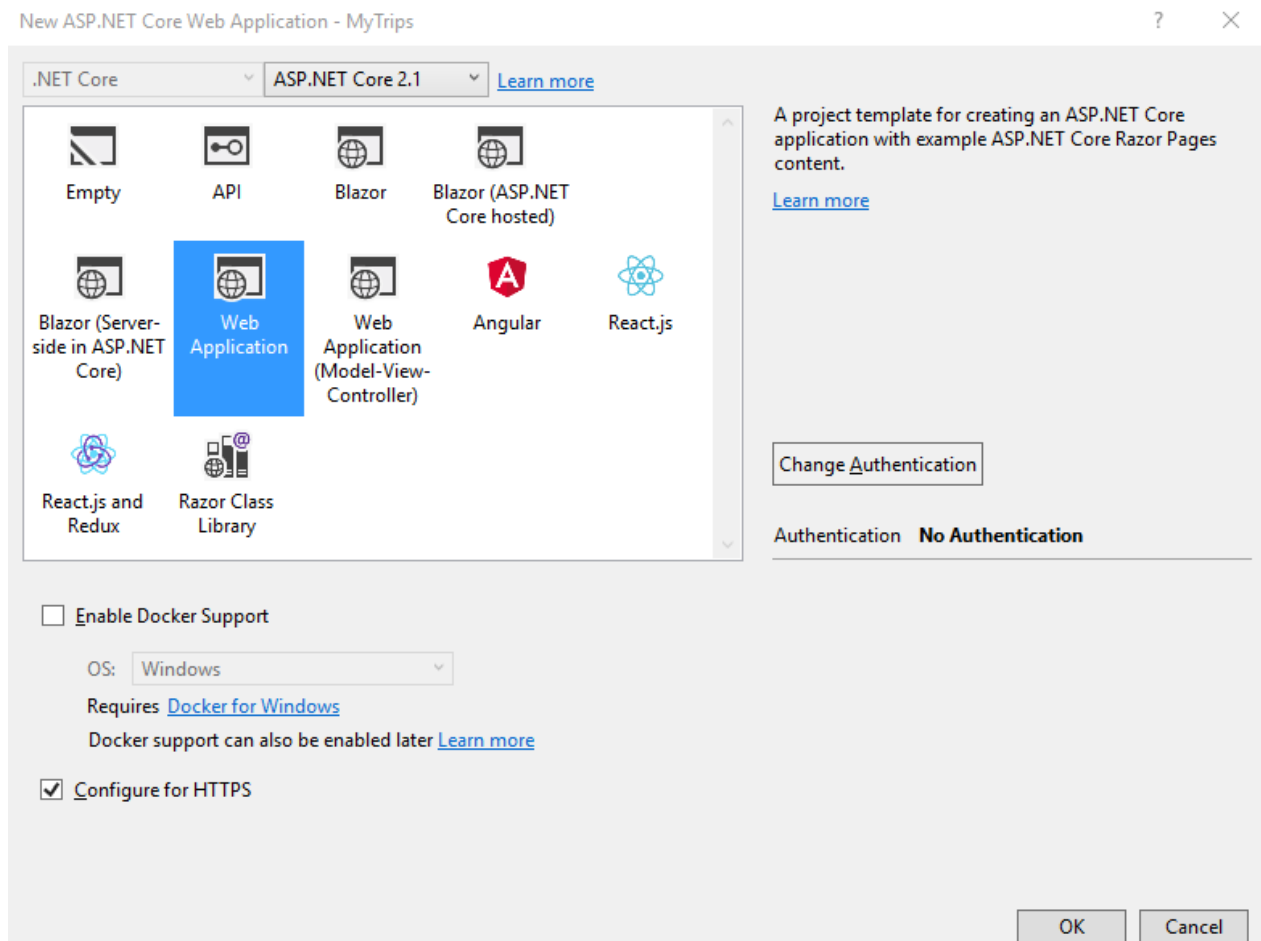
- [Visual Studio 2017](https://visualstudio.microsoft.com/tr/vs/) (<https://visualstudio.microsoft.com/tr/vs/>)
- [ASP.NET Core 2.1 SDK](https://www.microsoft.com/net/download) (<https://www.microsoft.com/net/download>)
- [ResX Manager](https://marketplace.visualstudio.com/items?itemName=TomEnglert.ResXManager) (optional) (<https://marketplace.visualstudio.com/items?itemName=TomEnglert.ResXManager>)

## Create Standard Project

Open Visual Studio, create a new ASP.NET Core Web Application, and name it “MyTrips”



Select Web Application (No Authentication), you can keep “Configure for HTTPS” selected.



Hit "OK" and wait a few seconds till the solution created and all required dependencies are loaded, then you will have basic project files listed in the solution explorer, right click project name and create a new folder named "**Utilities**", this is optional step just to keep our project files more organized.

Then under "**Pages**" folder create a new razor page name it "**Trips**",

Hint: it may take a while to create the first page until Microsoft.VisualStudio.Web.CodeGeneration.Design package installed.

The current solution window will look like below:

[solution explorer basic]

Before we type any code, let us just test the default application; make sure that your build settings are like below:

[build settings]

Build the project (Ctrl + Shift + B) and run it without debugging because we will update our code frequently and come back to the project output to see the results. Select the project name from the solution explorer window and hit (Ctrl + Shift + W), then navigate to trips page: <http://localhost:xxxx/trips>

[first run]

## Route Value Based Localization Infrastructure

In a web application, route value is the best option for localization because people must be able to share page links in a selected culture. There is more options like query string parameter, cookies and accepted page header value, but we will focus on route value.

### Define Culture Route Model Convention

We want our culture parameter to be next to the web application Url and before any other route parameter.

e.g. <https://localhost:xxxx/trips> --> <https://localhost:xxxx/en/trips>

Create a new class inside **Utilities** folder and name it "**CultureTemplateRouteModelConvention.cs**"

[culture-template-route-model-convention-dialog]

This class will implement the `IPageRouteModelConvention` interface as below:

#### CultureTemplateRouteModelConvention.cs

```
using Microsoft.AspNetCore.Mvc.ApplicationModels;

namespace MyTrips.Utilities
{
    public class CultureTemplateRouteModelConvention : IPageRouteModelConvention
    {
        public void Apply(PageRouteModel model)
        {
            var selectorCount = model.Selectors.Count;
            for (var i = 0; i < selectorCount; i++)
            {
                var selector = model.Selectors[i];
                model.Selectors.Add(new SelectorModel
                {
                    AttributeRouteModel = new AttributeRouteModel
                    {
                        Order = -1,
                        Template = AttributeRouteModel.CombineTemplates(
                            "{culture?}",
                            selector.AttributeRouteModel.Template),
                    }
                });
            }
        }
    }
}
```

*Hint: to insert required/missing references put the cursor on the missing reference name and hit (Ctrl + .) you can use this key combination to remove unused references and to implement the interfaces as well.*

We have defined a route parameter named `{culture?}` notice that it has ( ? ) Therefore, it is optional, because URLs are hackable and any user may try to remove it or change the culture from the address bar, in case the selected culture not supported, the system must return default provided culture.

After we defined our culture route value template, we need to configure our application to use this template for all pages. In the project root open "**startup.cs**" file and modify add razor page options as below:

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_1)
    .AddRazorPagesOptions(o => {
        o.Conventions.Add(new CultureTemplateRouteModelConvention());
    });
```

Read more about razor pages route conventions in Microsoft docs [here](https://docs.microsoft.com/en-us/aspnet/core/razor-pages/razor-pages-conventions?view=aspnetcore-2.1) (<https://docs.microsoft.com/en-us/aspnet/core/razor-pages/razor-pages-conventions?view=aspnetcore-2.1>).

Our application configured to use culture route template, next we will build the route culture provider that will use the route parameter to localize the request and provide list of supported cultures.

Lets test our culture route template, run the project and navigate to any of the below links:

- <https://localhost:xxxx/en/trips>
- <https://localhost:xxxx/tr/trips>
- <https://localhost:xxxx/ar/trips>

All links is showing the same “**Trips**” page, we will not see any localization info yet, because we need to configure request localization provider then create localization resources.

## Route Value Request Culture Provider

By default, ASP.NET has three request culture providers:

- QueryStringRequestCultureProvider
- CookieRequestCultureProvider
- AcceptLanguageHeaderRequestCultureProvider

There is no RouteValueRequestCultureProvider, so let's create one.

Inside “**Utilities**” folder, create a new class name it “**RouteValueRequestCultureProvider.cs**”, and implement IRequestCulture provider interface as below:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Localization;
using System.Globalization;
using System.Linq;
using System.Threading.Tasks;

namespace MyTrips.Utilities
{
    public class RouteValueRequestCultureProvider : IRequestCultureProvider
    {
        private readonly CultureInfo[] _cultures;

        public RouteValueRequestCultureProvider(CultureInfo[] cultures)
        {
            _cultures = cultures;
        }

        public Task<ProviderCultureResult> DetermineProviderCultureResult(HttpContext httpContext)
        {
            var defaultCulture = "en";

            var path = httpContext.Request.Path;

            if (string.IsNullOrEmpty(path))
            {
                return Task.FromResult(new ProviderCultureResult(defaultCulture));
            }

            var routeValues = httpContext.Request.Path.Value.Split('/');
            if (routeValues.Count() <= 1)
            {
                return Task.FromResult(new ProviderCultureResult(defaultCulture));
            }

            if (!_cultures.Any(x =>
                x.TwoLetterISOLanguageName.ToLower() == routeValues[1].ToLower() ||
                x.Name.ToLower() == routeValues[1].ToLower()))
            {
                return Task.FromResult(new ProviderCultureResult(defaultCulture));
            }

            return Task.FromResult(new ProviderCultureResult(routeValues[1]));
        }
    }
}
```

We extracted the first path parameter because we managed our {culture?} route value to be the first parameter in the request path. The return result is ProviderCultureResult depending on the route value. If no route value provided default culture “en” will be returned.





Now we need to configure localization options in startup file, so we use our `RouteValueRequestCulture` provider as the default request localization provider.

Under “**Utilities**” folder create a new class name it “**LocalizationExtension.cs**” and create an extension method for `IServiceCollection` as below :

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Localization;
using Microsoft.Extensions.DependencyInjection;
using System.Globalization;

namespace MyTrips.Utilities
{
    public static class LocalizationExtension
    {
        public static void ConfigureRequestLocalization(this IServiceCollection services)
        {
            var cultures = new CultureInfo[]
            {
                new CultureInfo("en"),
                new CultureInfo("tr"),
                new CultureInfo("ar") {
                    // Arabic cultures uses Hijri calendar by default
                    // comment the below line to use Hijri calendar instead of Gregorian
                    DateTimeFormat = { Calendar = new GregorianCalendar() }
                }
            };

            services.Configure<RequestLocalizationOptions>(ops =>
            {
                ops.DefaultRequestCulture = new RequestCulture("en");
                ops.SupportedCultures = cultures;
                ops.SupportedUICultures = cultures;

                // add RouteValueRequestCultureProvider to the beginning of the providers list.
                ops.RequestCultureProviders.Insert(0,
                    new RouteValueRequestCultureProvider(cultures));
            });
        }
    }
}
```

We have defined a list of supported cultures (en, tr and ar) and defined a default culture, you may define your own cultures list as you wish. Then we have inserted our `RouteValueRequestCultureProvider` at the beginning of the request culture providers list, so in case no route value for culture, the next provider will try to find culture value in query string, cookie or accepted header value.

In “**startup.cs**” file call the extension method we have created then configure the app to use request localization middleware, startup file will be like below:

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using MyTrips.Utilities;

namespace MyTrips
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        .
        .
        .
        services.ConfigureRequestLocalization();

        services.AddMvc()
            .SetCompatibilityVersion(CompatibilityVersion.Version_2_1)
            .AddRazorPagesOptions(o => {
                o.Conventions.Add(new CultureTemplateRouteModelConvention());
            });
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        .
        .
        .

        app.UseRequestLocalization();
        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseCookiePolicy();

        app.UseMvc();
    }
}
```

Now lets create a language dropdown list, so we can navigate easily between cultures.

## Create Language Dropdown Navigation

Under Pages/Shared folder create a new razor view and name it “**\_LanguagePartial.cshtml**”, we can simply create a dropdown list and fill culture items manually, but it is better to retrieve supported cultures list from RequestLocalization Options.

```
@using System.Globalization;
@using Microsoft.AspNetCore.Builder;
@using Microsoft.Extensions.Options

@inject IOptions<RequestLocalizationOptions> LocOps

@{
    var requestCulture = CultureInfo.CurrentCulture;
    var supportedCultures = LocOps.Value.SupportedUICultures
        .Select(c => new SelectListItem
        {
            Value = c.TwoLetterISOLanguageName,
            Text = $"{c.DisplayName} ({c.NativeName})"
        }).ToList();
}

<ul class="nav navbar-nav navbar-right">
    <li class="dropdown">
        <a class="dropdown-toggle" data-toggle="dropdown" href="#">
            @($"{requestCulture.DisplayName} ({requestCulture.NativeName})")
            <span class="caret"></span>
        </a>
        <ul class="dropdown-menu">
            @foreach (var culture in supportedCultures)
            {
                if (culture.Value.ToLower() != requestCulture.Name.ToLower())
                {
                    <li>
                        <a
                            asp-route-culture="@culture.Value"
                            asp-route-returnUrl="@((Context.Request.Query["returnUrl"])">
                                @culture.Text
                            </a>
                        </li>
                    }
                }
            }
        </ul>
    </li>
</ul>
```

Then we place our language partial into the top menu bar, insert it right before closing menu list div inside `_Layout.cshtml` page :

```
<nav class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle"
        data-toggle="collapse" data-target=".navbar-collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a asp-page="/Index" class="navbar-brand">MyTrips</a>
    </div>
    <div class="navbar-collapse collapse">
      <ul class="nav navbar-nav">
        <li><a asp-page="/Index">Home</a></li>
        <li><a asp-page="/About">About</a></li>
        <li><a asp-page="/Contact">Contact</a></li>
      </ul>
      <partial name="_LanguagePartial" />
    </div>
  </div>
</nav>
```

Add a small test output to our `"Trips.cshtml"` page:

```
@page
@using System.Globalization;

@model MyTrips.Pages.TripsModel
@{
    ViewData["Title"] = "Trips";
}

<h2>Trips</h2>

<p>
    Current culture :
    @CultureInfo.CurrentCulture.DisplayName (@CultureInfo.CurrentCulture.NativeName)
</p>
<p>Current date : @DateTime.Now</p>
```

Build and run the application, then navigate to trips page (<https://localhost:xxxx/en/trips>), you will see the language list on the right side, change the language and see how different cultures uses different date time formats

MyTrips

Home

About

Contact

Turkish (Türkçe) ▾

## Trips

Curent culture : Turkish (Türkçe)

Current date : 9.08.2018 09:03:49

© 2018 - MyTrips

MyTrips

Home

About

Contact

Arabic (العربية) ▾

## Trips

Curent culture : Arabic (العربية)

Current date : 27/11/39 09:00:33 ص

© 2018 - MyTrips

MyTrips

Home

About

Contact

English (English) ▾

## Trips

Curent culture : English (English)

Current date : 8/9/2018 9:03:26 AM

© 2018 - MyTrips

Now it is the time to localize view texts.

## Localizing Views

In order to localize views we need to add view localization middleware in startup file:

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_1)
    .AddViewLocalization(o => o.ResourcesPath = "Resources")
    .AddRazorPagesOptions(o => {
        o.Conventions.Add(new CultureTemplateRouteModelConvention());
    });
```

The ResourcePath is the folder name where our localization resource files are stored. We have specified it as **"Resources"** so under the project root create a new folder and name it **"Resources"**.

There is different ways to localize views (see [Resource Files for Localization](#)), we will use shared resources because it requires less amount of resource files, but it requires creating culture localizer class to read and return localized texts from shared resource files.

For views localization, we will not use public access modifiers in key names, because we will use normal texts as keys in **"ViewResource.xx.resx"** files, so we need to create a dummy class to group view localization resources and access them via a localizer class. Under **"Resources"** folder create a new class named **"ViewResource.cs"**:

```
namespace MyTrips.Resources
{
    // dummy class for grouping localizarion resources
    public class ViewResource
    {
    }
}
```

Then under **"Utilities"** folder create a new class named **"CultureLocalizer.cs"**:

```
using Microsoft.Extensions.Localization;
using MyTrips.Resources;
using System.Reflection;

namespace MyTrips.Utilities
{
    public class CultureLocalizer
    {
        private readonly IStringLocalizer _localizer;
        public CultureLocalizer(IStringLocalizerFactory factory)
        {
            var type = typeof(ViewResource);
            var assemblyName = new AssemblyName(type.GetTypeInfo().Assembly.FullName);
            _localizer = factory.Create("ViewResource", assemblyName.Name);
        }

        // if we have formatted string we can provide arguments
        // e.g.: @Localizer.Text("Hello {0}", User.Name)
        public LocalizedString Text(string key, params string[] arguments)
        {
            return arguments == null
                ? _localizer[key]
                : _localizer[key, arguments];
        }
    }
}
```

Our **"CultureLocalizer"** class creates a string localizer factory, it has a method which will look for the localized string and return it from **"ViewResource"** files.

Register our localizer class in startup file under ConfigureServices method in “**startup.cs**” file:

```
public void ConfigureServices(IServiceCollection services)
{
    .
    .
    .

    services.AddSingleton<CultureLocalizer>();

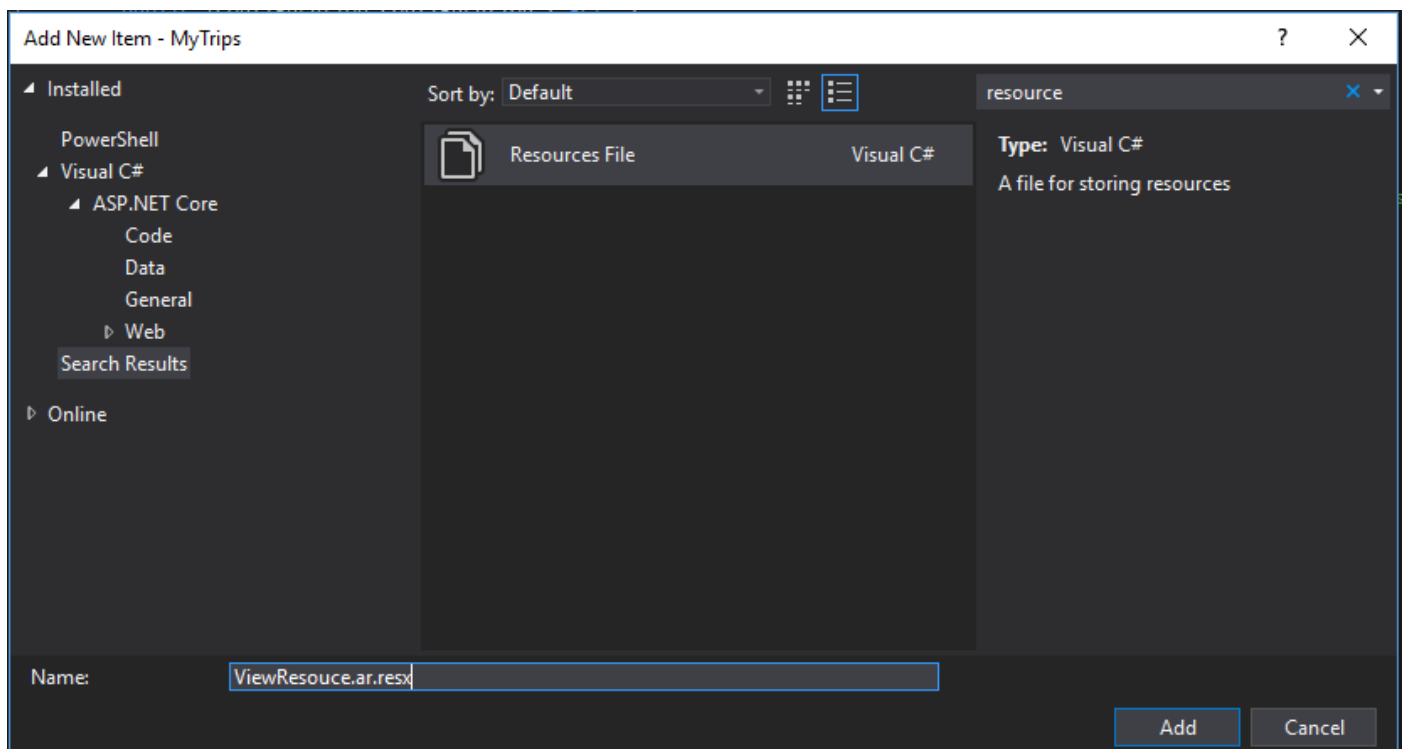
    services.ConfigureRequestLocalization();

    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1)
        .AddViewLocalization(o => o.ResourcesPath = "Resources")
        .AddModelBindingMessagesLocalizer(services)
        .AddRazorPagesOptions(o => {
            o.Conventions.Add(new CultureTemplateRouteModelConvention());
        });
}
```

Now we need to create view resource files; under “**Resources**” folder create a new resource file for each culture :

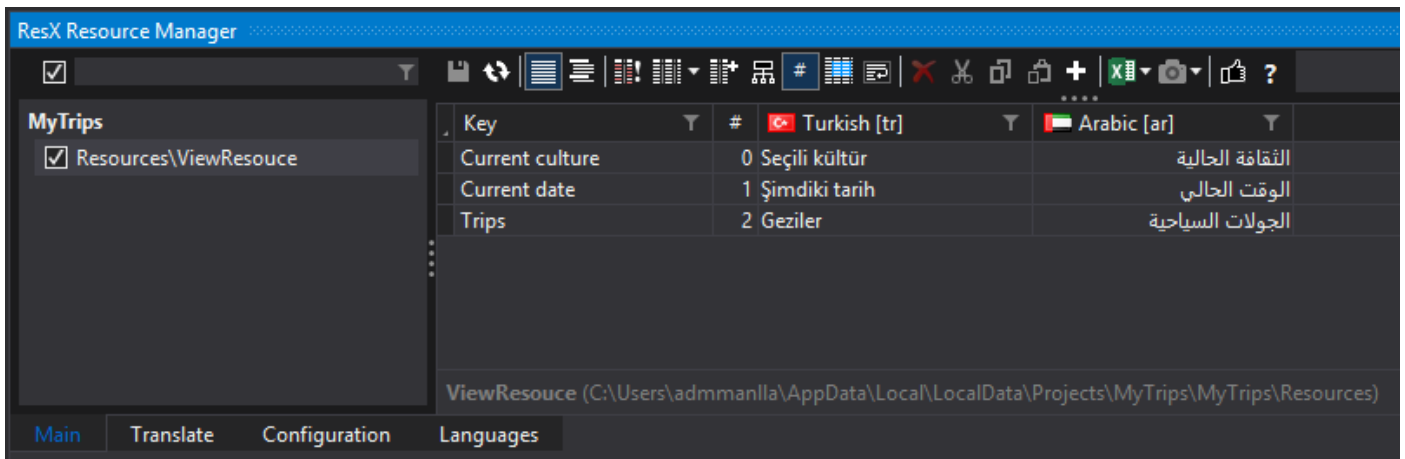
/Resources/ViewResource.tr.resx

/Resources/ViewResource.ar.resx



It is not necessary to create a resource file for the default language (English in our case), because the string localizer will return the key (provided text) in case no resource has found. Fill the files with required local texts:





Notice that I am using ResX manager to manage localization resources, you may use the default editor in Visual Studio but I highly recommend that you download ResX manager and try it.

Next, we need to inject our “**CultureLocalizer**” class to every view; the best place to do so is “\_ViewImports.cshtml” file under “**Pages**” folder:

```
@using MyTrips
@namespace MyTrips.Pages
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

@inject Utilities.CultureLocalizer Localizer
```

Finally, modify “**Trips.cshtml**” page so it uses “**CultureLocalizer**” class to provide local texts according to the selected culture:

```
@page
@using System.Globalization;

@model MyTrips.Pages.TripsModel
@{
    ViewData["Title"] = Localizer.Text("Trips");
}

<h2>@ViewData["Title"]</h2>

<p>
    @Localizer.Text("Current culture") :
    @CultureInfo.CurrentCulture.DisplayName (@CultureInfo.CurrentCulture.NativeName)
</p>

<p>
    @Localizer.Text("Current date") : @DateTime.Now
</p>
```

Build and run the application, go ahead to Trips page and switch between cultures:

## Trips

Current culture : English (English)

Current date : 8/9/2018 10:09:01 AM

---

© 2018 - MyTrips

## Geziler

Seçili kültür : Turkish (Türkçe)

Şimdiki tarih : 9.08.2018 10:06:11

---

© 2018 - MyTrips

## الجولات السياحية

الثقافة الحالية : Arabic (العربية)

الوقت الحالي : 10:09:21 27/11/39 ص

---

© 2018 - MyTrips

## Localizing Data Annotations

### Localizing Property Display Name

Data annotations are attribute validation messages, *like string length and required attribute*, they can be localized similar to views using a shared resource files as described in [Microsoft documentations](#). But this method require using strings in the attribute names and error messages. As a C# developer, I prefer to use strongly typed property names instead of strings in the code side. That is why I will use shared resource files with public access modifiers for localization of data annotations.

Modify “**Trips**” file and create a simple trip model and form:

// Trips.cshtml.cs

```
using System;
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace MyTrips.Pages
{
    public class TripsModel : PageModel
    {
        public class Trip
        {
            [Required]
            public string Destination { get; set; }
            public DateTime TravelDate { get; set; }
            public decimal TicketPrice { get; set; }
        }

        [BindProperty]
        public Trip MyTrip { get; set; }

        public void OnGet()
        {
        }
    }
}
```

// Trips.cshtml

@page

@model MyTrips.Pages.TripsModel

@{  
     ViewData["Title"] = Localizer.Text("Trips");  
}

<h2>@ViewData["Title"]</h2>

<form method="post">

    <p>@Localizer.Text("Please fill below your travel info:")</p>

    <div asp-validation-summary="All" class="alert-danger"></div>

    <div class="form-group">

        <label asp-for="MyTrip.Destination"></label>

        <input asp-for="MyTrip.Destination" class="form-control" />

        <span asp-validation-for="MyTrip.Destination" class="text-danger"></span>

    </div>

    <div class="form-group">

        <label asp-for="MyTrip.TravelDate"></label>

        <input asp-for="MyTrip.TravelDate" class="form-control" />

        <span asp-validation-for="MyTrip.TravelDate" class="text-danger"></span>

    </div>

    <div class="form-group">

        <label asp-for="MyTrip.TicketPrice"></label>

        <input asp-for="MyTrip.TicketPrice" class="form-control" />

        <span asp-validation-for="MyTrip.TicketPrice" class="text-danger"></span>

    </div>

    <button type="submit" class="btn btn-primary">@Localizer.Text("Submit")</button>

</form>

Run the application and switch to any culture:

MyTrips

Home

About

Contact

Turkish (Türkçe) ▾

## Geziler

Please fill below your travel info:

**Destination**

**TravelDate**

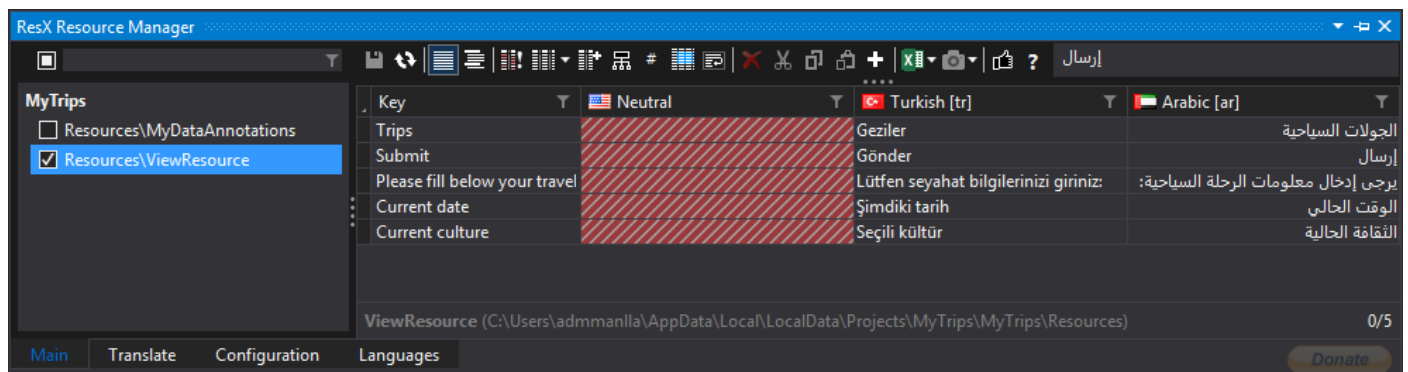
**TicketPrice**

**Submit**

---

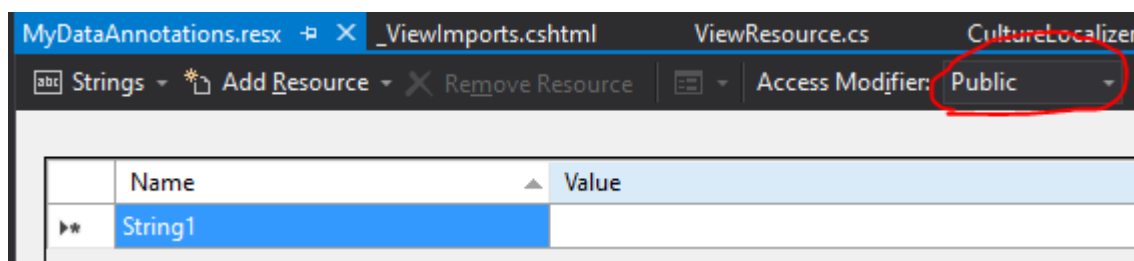
© 2018 - MyTrips

For the form title and submit button, we can easily enter the localization texts into “**ViewResource.xx.resx**” resource files as we already did previously.



Form labels will be localized using a different approach. Inside “**Resources**” folder create a new resource file name it “**MyDataAnnotations.resx**” this file will contain the default culture texts, make sure to change access modifiers to public:

/Resources/MyDataAnnotations.resx

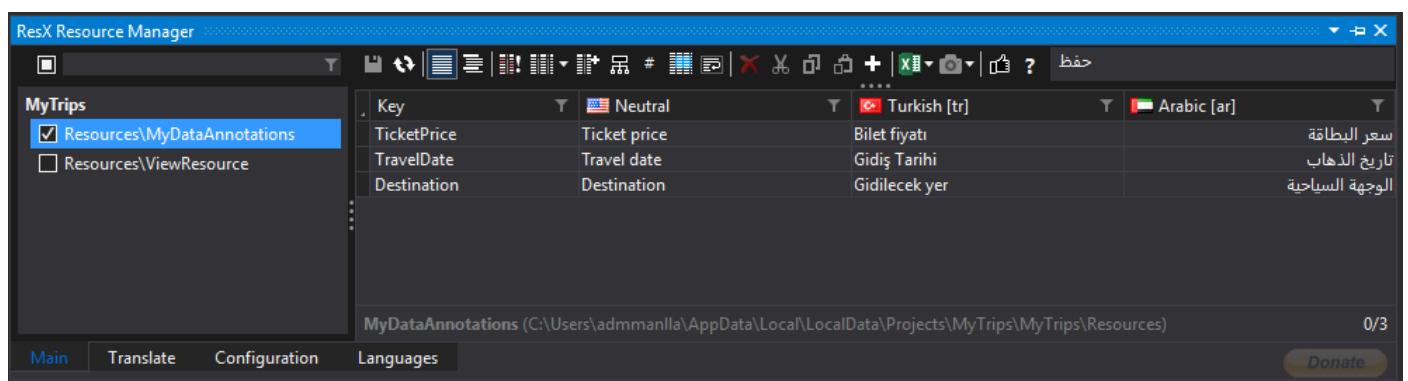


Then create one file for each culture:

/Resources/MyDataAnnotations.tr.resx

/Resources/MyDataAnnotations.ar.resx

Fill the files with localized label texts as below:



Modify “Trip” model by adding [Display(Name=“...”)] attribute as below:

```
public class Trip
{
    [Required]
    [Display(
        Name = nameof(MyDataAnnotations.Destination),
        ResourceType = typeof(MyDataAnnotations))]
    public string Destination { get; set; }

    [Display(
        Name = nameof(MyDataAnnotations.TravelDate),
        ResourceType = typeof(MyDataAnnotations))]
    public DateTime TravelDate { get; set; }

    [Display(
        Name = nameof(MyDataAnnotations.TicketPrice),
        ResourceType = typeof(MyDataAnnotations))]
    public decimal TicketPrice { get; set; }
}
```

Build and run the app, switch between languages and see the results:

MyTrips

Home

About

Contact

Arabic (العربية) ▾

ال جولات السياحية

يرجى إدخال معلومات الرحلة السياحية

الوجهة السياحية

تاريخ الذهاب

سعر التذكرة

إرسال

© 2018 - MyTrips

MyTrips

Home

About

Contact

English (English) ▾

Trips

Please fill below your travel info:

Destination

Travel date

Ticket price

Submit

© 2018 - MyTrips

MyTrips

Home

About

Contact

Turkish (Türkçe) ▾

Geziler

Lütfen seyahat bilgilerinizi giriniz:

Gidilecek yer

Gidiş Tarihi

Bilet fiyatı

Gönder

© 2018 - MyTrips

Submit blank form to see validation error messages:

**MyTrips** [Home](#) [About](#) [Contact](#) Turkish (Türkçe) ▼

## Geziler

Lütfen seyahat bilgilerinizi giriniz:

- The Gidilecek yer field is required.
- The value " is invalid.
- The value " is invalid.

**Gidilecek yer**

The Gidilecek yer field is required.

**Gidiş Tarihi**

The value " is invalid.

**Bilet fiyatı**

The value " is invalid.

**Gönder**

---

© 2018 - MyTrips

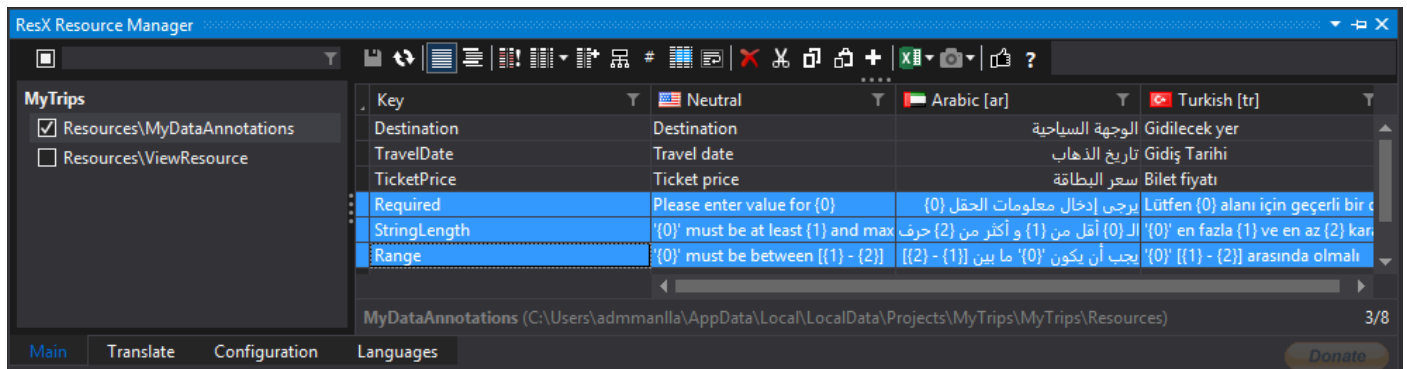
Next topic: Localizing validation messages

## Localizing Validation Messages

There are two types of validation messages, [ValidationAttribute](#) messages like (Required, StringLength, etc.) and [ModelBindingMessage](#) that validates if the entered value matches the field type (int, string or DateTime). Each type of messages requires a different approach for localization.

### Localizing ValidationAttribute Messages

In “Trip” model, we need localized validation messages for [Required], [StringLength] and [Range] attributes. Add localized messages to “MyDataAnnotations.xx.resx” files:



Modify “Trip” model by decorating its properties with appropriate attributes:

```
public class Trip
{
    [Required(
        ErrorMessageResourceName = nameof(MyDataAnnotations.Required),
        ErrorMessageResourceType = typeof(MyDataAnnotations))]
    [Display(
        Name = nameof(MyDataAnnotations.Destination),
        ResourceType = typeof(MyDataAnnotations))]
    public string Destination { get; set; }

    [Required(
        ErrorMessageResourceName = nameof(MyDataAnnotations.Required),
        ErrorMessageResourceType = typeof(MyDataAnnotations))]
    [Display(
        Name = nameof(MyDataAnnotations.TravelDate),
        ResourceType = typeof(MyDataAnnotations))]
    public DateTime? TravelDate { get; set; }

    [Required(
        ErrorMessageResourceName = nameof(MyDataAnnotations.Required),
        ErrorMessageResourceType = typeof(MyDataAnnotations))]
    [Range(10.00, 10000.00,
        ErrorMessageResourceName = nameof(MyDataAnnotations.Range),
        ErrorMessageResourceType = typeof(MyDataAnnotations))]
    [Display(
        Name = nameof(MyDataAnnotations.TicketPrice),
        ResourceType = typeof(MyDataAnnotations))]
    public decimal? TicketPrice { get; set; }
}
```



Notice that we have made the TravelDate and TicketPrice as nullable types by adding ( ? ) in order to trigger [Required] attribute validation (see: [Override Required Validation Attribute Error Message](#)).

Compile and run the project, submit the form with blank fields:

MyTrips

Home

About

Contact

Turkish (Türkçe) ▼

## Geziler

Lütfen seyahat bilgilerinizi giriniz:

- Lütfen Gidilecek yer alanı için geçerli bir değer girin
- 'Gidiş Tarihi' [1.01.2000 00:00:00 - 31.12.2050 00:00:00] arasında olmalı
- 'Bilet fiyatı' [10 - 10000] arasında olmalı

**Gidilecek yer**

Lütfen Gidilecek yer alanı için geçerli bir değer girin

**Gidiş Tarihi**

'Gidiş Tarihi' [1.01.2000 00:00:00 - 31.12.2050 00:00:00] arasında olmalı

**Bilet fiyatı**

'Bilet fiyatı' [10 - 10000] arasında olmalı

Gönder

“Price” field accepts decimal values, what would happen if we enter a text there?

**MyTrips** [Home](#) [About](#) [Contact](#) Turkish (Türkçe) ▼

## Geziler

Lütfen seyahat bilgilerinizi giriniz:

- The value 'AAA' is not valid for Bilet fiyatı.

**Gidilecek yer**

**Gidiş Tarihi**

**Bilet fiyatı**

The value 'AAA' is not valid for Bilet fiyatı.

**Gönder**

---

© 2018 - MyTrips

This is a model binding message, we have entered a text instead of a number. ModelBinding messages cannot be localized like ValidationAttributes. We localize it by calling ModelBindingMessageProvider and setting values of localized messages at startup file.

Inside “Utilities” folder, create a new static class named “MvcOptionsExtensions” and create an extension method AddModelBindingMessagesLocalizer to extend IMvcBuilder:

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Localization;
using MyTrips.Resources;
using System.Reflection;

namespace MyTrips.Utilities
{
    public static class MvcOptionsExtension
    {
        public static IMvcBuilder AddModelBindingMessagesLocalizer
            (this IMvcBuilder mvc, IServiceCollection services)
        {
            return mvc.AddMvcOptions(o =>
            {
                var type = typeof(MyDataAnnotations);
                var assemblyName = new AssemblyName(type.GetTypeInfo().Assembly.FullName);
                var factory = services.BuildServiceProvider().GetService<IStringLocalizerFactory>();
                var localizer = factory.Create("MyDataAnnotations", assemblyName.Name);

                o.ModelBindingMessageProvider
                    .SetAttemptedValueIsInvalidAccessor((x, y) => localizer["InvalidValue"]);

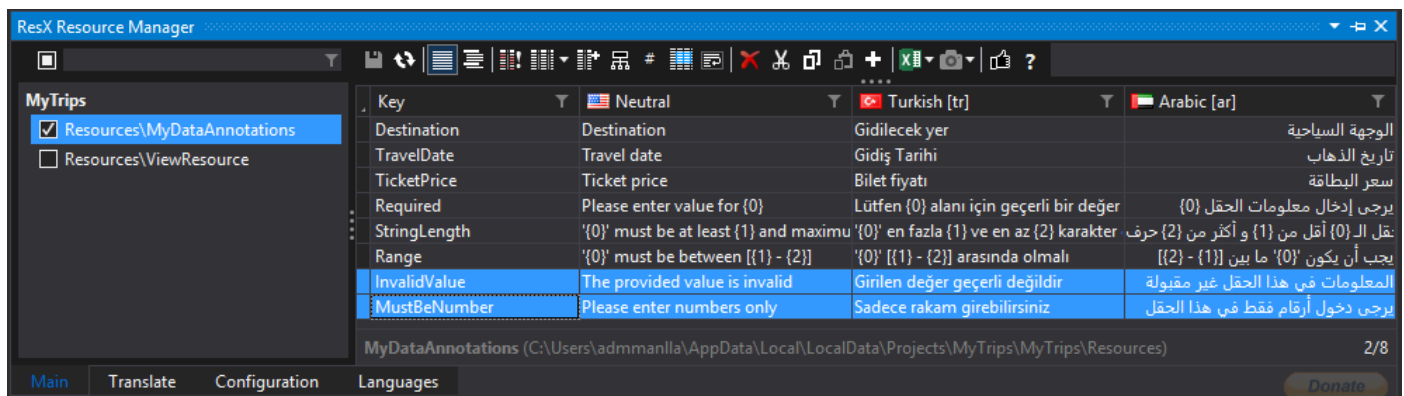
                o.ModelBindingMessageProvider
                    .SetValueMustBeANumberAccessor((x) => localizer["MustBeNumber"]);
            });
        }
    }
}
```

Based on the solution in stackoverflow.com: <https://stackoverflow.com/a/41669552/5519026>

Call the extension method from startup file next to AddMvc():

```
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_1)
    .AddViewLocalization(o=>o.ResourcesPath = "Resources")
    .AddModelBindingMessagesLocalizer(services)
    .AddRazorPagesOptions(o => {
        o.Conventions.Add(new CultureTemplateRouteModelConvention());
    });
```

Add localized “InvalidValue” and “MustBeNumber” to “MyDataAnnotations.xx.resx” files:





Compile and run the project, type some string into the price field and submit the form, you will see localized model binding message:

MyTrips

Home

About

Contact

Turkish (Türkçe) ▾

## Geziler

Lütfen seyahat bilgilerinizi giriniz:

• Girilen değer geçerli değildir

Gidilecek yer

Istanbul

Gidiş Tarihi

09-Nov-18 09:54

Bilet fiyatı

AAA

Girilen değer geçerli değildir

Gönder

---

© 2018 - MyTrips

## Adding Client Side Validation

Trips form application almost done, we still missing client side validation to complete the validation localization process.

Default validation scripts already provided in “**Pages/Shared/\_ValidationScriptsPartial.cshtml**”. Simply include validation partial into “**Trips.cshtml**” file directly after closing the form tag:

```
@page
@model MyTrips.Pages.TripsModel
@{
    ViewData["Title"] = Localizer.Text("Trips");
}

<h2>@ViewData["Title"]</h2>

<form method="post">
    <p>@Localizer.Text("Please fill below your travel info:")</p>
    <div asp-validation-summary="All" class="alert-danger"></div>
    <div class="form-group">
        <label asp-for="MyTrip.Destination"></label>
        <input asp-for="MyTrip.Destination" class="form-control" />
        <span asp-validation-for="MyTrip.Destination" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="MyTrip.TravelDate"></label>
        <input asp-for="MyTrip.TravelDate" class="form-control" />
        <span asp-validation-for="MyTrip.TravelDate" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="MyTrip.TicketPrice"></label>
        <input asp-for="MyTrip.TicketPrice" class="form-control" />
        <span asp-validation-for="MyTrip.TicketPrice" class="text-danger"></span>
    </div>
    <button type="submit" class="btn btn-primary">@Localizer.Text("Submit")</button>
</form>

@section Scripts{
    <partial name="_ValidationScriptsPartial" />
}
```

Validation scripts will show validation messages after switching from input control to another one, without the need to click “**Submit**” button.

Everything seems to okay, but if you type decimal numbers in non-English cultures (e.g. 15,5), you will see validation error messages. That is because some non-English cultures uses comma ( , ) as decimal separator, while it is ( . ) in English cultures. And client side validation rely on “**jquery.validate.js**”, it is developed considering English standards and it uses dot ( . ) as decimal separator! In order to fix this; we have to add more scripts to our “**\_ValidationScriptsPartial**”.

## Client Side Validation for Non-English Cultures

Download and install the following packages:

Globalize <https://github.com/globalizejs/globalize>

Jquery-validation-globalize: <https://github.com/johnnyreilly/jquery-validation-globalize>

cldr-json data: <https://github.com/unicode-cldr/cldr-json#cldr-json>

Modify “**\_ValidationScriptsPartial.cshtml**” file to include newly installed libraries:

```

<environment include="Development">
  <script src="~/lib/jquery-validation/dist/jquery.validate.js"></script>
  <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"></script>
</environment>
<environment exclude="Development">
  <script src="https://ajax.aspnetcdn.com/ajax/jquery.validate/1.17.0/jquery.validate.min.js"
    asp-fallback-src="~/lib/jquery-validation/dist/jquery.validate.min.js"
    asp-fallback-test="window.jQuery && window.jQuery.validator"
    crossorigin="anonymous"
    integrity="sha384-rZfj/ogBloos6wzLGpPkkOr/gpkBNLZ6b6yLy4o+ok+t/SAK1L5mvXlr00XNi1Hp">
  </script>
  <script
src="https://ajax.aspnetcdn.com/ajax/jquery.validation.unobtrusive/3.2.9/jquery.validate.unobtrusive.min.
js"
    asp-fallback-src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"
    asp-fallback-test="window.jQuery && window.jQuery.validator &&
window.jQuery.validator.unobtrusive"
    crossorigin="anonymous"
    integrity="sha384-ifu0TYDwxBHvAk2Z0n8R434FL1Rlv/Av18DXE43N/1rvHyOG4izKst0f2iSLdds">
  </script>
</environment>

<!-- cldr scripts (needed for globalize) -->
<script src="~/lib/cldrjs/dist/cldr.js"></script>
<script src="~/lib/cldrjs/dist/cldr/event.js"></script>
<script src="~/lib/cldrjs/dist/cldr/supplemental.js"></script>

<!-- globalize scripts -->
<script src="~/lib/globalize/dist/globalize.js"></script>
<script src="~/lib/globalize/dist/globalize/number.js"></script>
<script src="~/lib/globalize/dist/globalize/date.js"></script>

<script src="~/lib/jquery-validation-globalize/jquery.validate.globalize.js"></script>

@inject Microsoft.AspNetCore.Hosting.IHostingEnvironment HostingEnvironment
@{
    string GetDefaultLocale()
    {
        const string localePattern = "lib\\cldr-data\\main\\{0}";
        var currentCulture = System.Globalization.CultureInfo.CurrentCulture;
        var cultureToUse = "en"; //Default regionalisation to use

        if (System.IO.Directory.Exists(System.IO.Path.Combine(HostingEnvironment.WebRootPath,
string.Format(localePattern, currentCulture.Name))))
            cultureToUse = currentCulture.Name;
        else if (System.IO.Directory.Exists(System.IO.Path.Combine(HostingEnvironment.WebRootPath,
string.Format(localePattern, currentCulture.TwoLetterISOLanguageName))))
            cultureToUse = currentCulture.TwoLetterISOLanguageName;

        return cultureToUse;
    }
}

<script type="text/javascript">
var culture = "@GetDefaultLocale()";
$.when(
    $.get("/lib/cldr-data/supplemental/likelySubtags.json"),
    $.get("/lib/cldr-data/main/" + culture + "/numbers.json"),
    $.get("/lib/cldr-data/supplemental/numberingSystems.json"),
    $.get("/lib/cldr-data/main/" + culture + "/ca-gregorian.json"),
    $.get("/lib/cldr-data/main/" + culture + "/timeZoneNames.json"),
    $.get("/lib/cldr-data/supplemental/timeData.json"),
    $.get("/lib/cldr-data/supplemental/weekData.json")
).then(function () {
    // Normalize $.get results, we only need the JSON, not the request statuses.
    return [].slice.apply(arguments, [0]).map(function (result) {
        return result[0];
    });
}).then(Globalize.load).then(function () {
    Globalize.locale(culture);
});
</script>

```

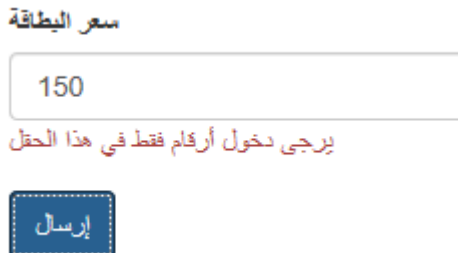


This solution found on <https://github.com/aspnet/Docs/issues/4076#issuecomment-366162798>

Build and run the application, submit decimal numbers in different cultures and see how client side validation works.

### Using Latin Numbering System in non-Latin Cultures

If your application will use numbers in non-Latin cultures (e.g. Arabic, Farsi, Hindu, etc.) you will get model binding error message like "Please use numbers for this field"



This is happening because of the numbering systems defined in cldr-data json files are culture specific. For example, Arabic cultures are using “٠١٢٣٤٥٦٧٨٩” digits as defined in “**numberingSystems.json**” file. However, the numbers we are trying to input in the form are Latin numbers “0123456789”. That is why model-binding error appears.

*The story of numbers and cultural exchange is pretty old; Arabs has founded the numbers as we know it now as Latin numbers “0123456789”, but later on Arab cultures started using Hindu numbering system “٠١٢٣٤٥٦٧٨٩” and the original Arabic numbering system “0123456789” became the standard for Latin cultures. Nevertheless, it worth to mention that currently almost all computer/smartphone users in Arabic cultures are using Latin numbers “0123456789” by default.*

It is enough history info :) let's have a look at numbering systems in cldr-json data files.

Open “~/lib/cldr-data/main/ar/numbers.json” file, you will see that default numbering system is “arab”:

```
"numbers": {
  "defaultNumberingSystem": "arab",
  "defaultNumberingSystem-alt-latn": "latn",
  "otherNumberingSystems": {
    "native": "arab"
  },
}
```

*If you are using country based culture e.g.: “ar-EG” you have to change the file inside that culture folder*

*e.g.: “~/lib/cldr-data/main/ar-EG/numbers.json”*

The referenced numbering systems are defined in “~/lib/cldr-data/supplemental/numberingSystems.json” as below:

```
"arab": {
  "_digits": "٠١٢٣٤٥٦٧٨٩",
  "_type": "numeric"
},
```

So if the application will support only Latin numbering system “0123456789”, just open “~/lib/cldr-data/main/ar/numbers.json” file and modify “defaultNumberingSystem” and “native” values to be “latn” and it will work.

```
"numbers": {  
  "defaultNumberingSystem": "latn",  
  "defaultNumberingSystem-alt-latn": "latn",  
  "otherNumberingSystems": {  
    "native": "latn"  
  },  
}
```