https://www.halvorsen.blog

ASP.NET Core Data Binding

Hans-Petter Halvorsen

Contents

In this Tutorial we will see how we can get user data from the **client-side** to the **server-side** in ASP.NET Core

Introduction

Hans-Petter Halvorsen

Table of Contents

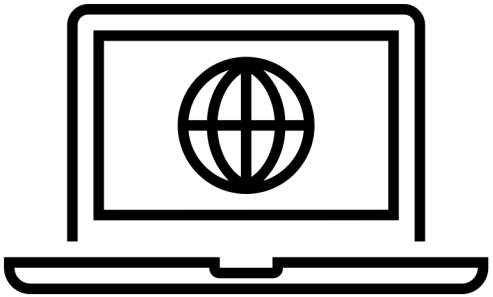
Introduction

- In this Tutorial we will see how we can get user data from the client-side to the server-side in ASP.NET Core
- Web Applications have a more complicated architecture since we have both server-side code and client-side code (which runs inside a web browser)
- This means this must be done in a different way compared to traditional Windows Forms Desktop Applications
- We start with the traditional way used in many Web Programming Frameworks
  - Request.Form[]
  - Request.Query[]
- Then we will use an alternative Data Binding approach which is part of the ASP.NET Core Framework

ASP.NET Core Architecture

Client-side

Web Browser



HTML

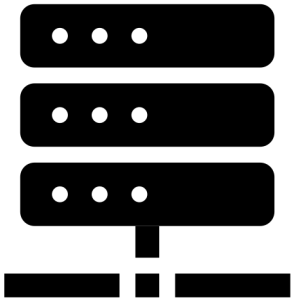
CSS

JavaScript

Data sent between
Server and Client

Server-side

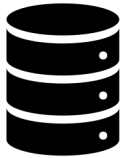
ASP.NET Core



Razor

C#

Database



HTML Forms

- The HTML <form> element is used to create a form on a web page.
- HTML Forms on a web page allows a user to enter data that is sent to a server for processing
- HTML Forms can contain Textboxes, Buttons, etc.
- The Form can have different attributes like **method** and **action**
- The **method** attribute determines the HTTP verb to use when the form is submitted, typically **get** or **post**

HTML Form Example

```
<form method="post">
    <input name="FullName" type="text" />
    <input id="OKButton" type="submit" value="OK" />
</form>
```

For the Form method we have the **get** or **post** attribute

An **HTML Form** is used to collect user input. The user input is most often sent to a server for processing.

ASP.NET Core Event Handlers

In ASP.NET you typically use the following Event Handlers:

- OnGet()
- OnPost()

So, in this Tutorial we will use those to react on Postbacks to the Server and handle data created on the Client

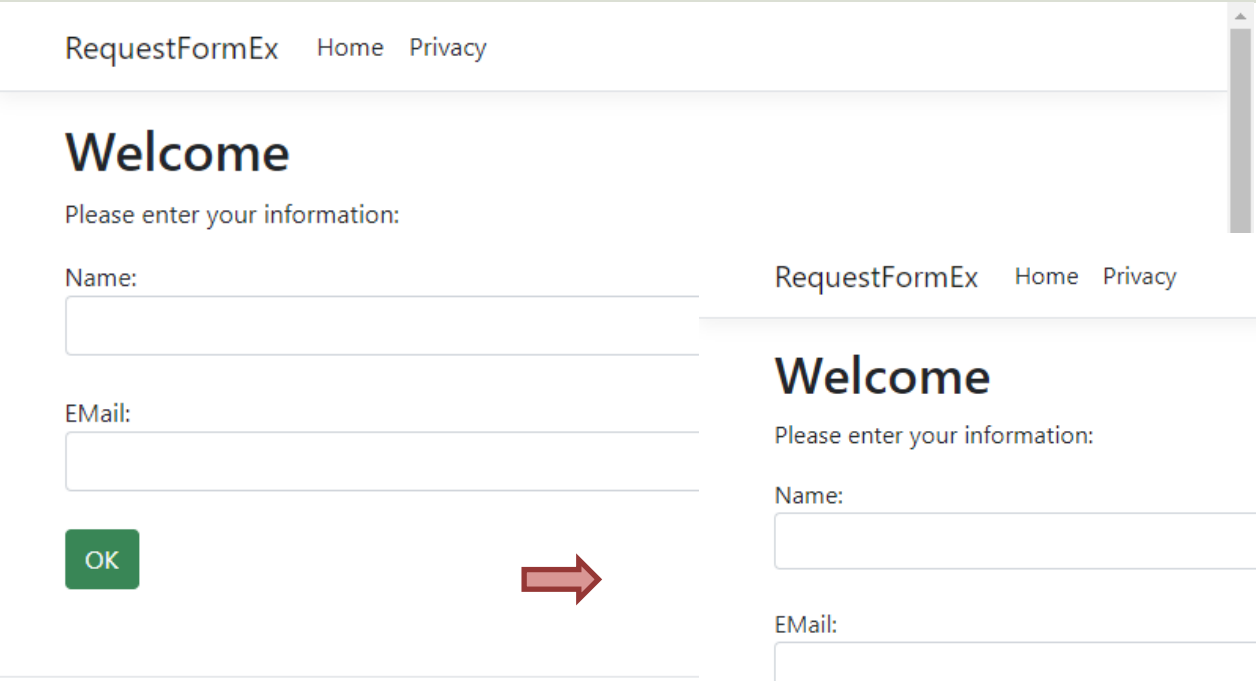
Request.Form[]

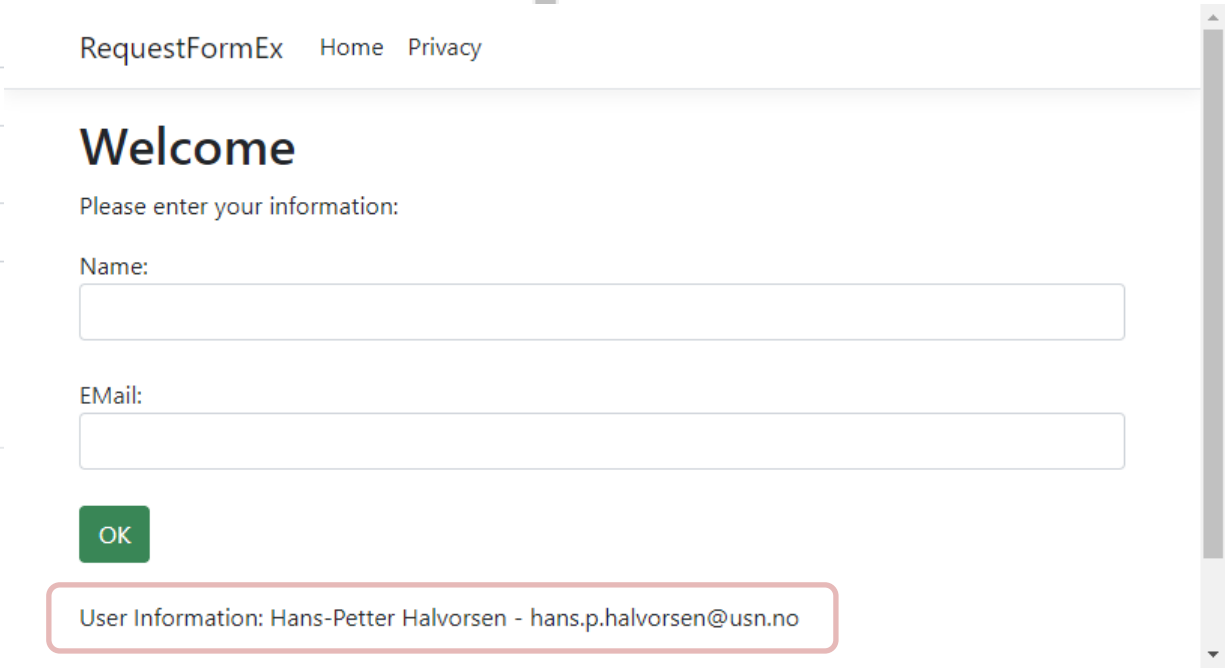
Example #1 – post and OnPost()

Hans-Petter Halvorsen

Table of Contents

Application





Index.cshtml

An **HTML Form** is used to collect user input. The user input is most often sent to a server for processing.

```
@page
@model IndexModel
@{
   ViewData["Title"] = "Home page";
}

<div>

   <h1>Welcome</h1>

   <p>Please enter your information:</p>

   <form name="NameForm" id="NameForm" method="post">

      <label for="FullName">Name:</label>
      <input name="FullName" type="text" class="form-control input-lg" autofocus required />
      <br />

      <label for="EMail">EMail:</label>
      <input name="EMail" type="email" class="form-control input-lg" required />
      <br />

      <input id="OKButton" type="submit" value="OK" class="btn btn-success" />

   </form>
   <br />

   <p>@ViewData["UserData"] </p>

</div>
```

Index.cshtml.cs

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace RequestFormEx.Pages
{
   public class IndexModel : PageModel
   {

      public void OnPost()
      {
         string fullName;
         string eMail;

         fullName = Request.Form["FullName"];
         eMail = Request.Form["EMail"];

         ViewData["UserData"] = "User Information: " + fullName + " - " + eMail;
      }
   }
}
```

The **OnPost()** method is automatically called when you push the Button

Improved Code in Index.cshtml.cs

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace RequestFormEx.Pages
{
   public class IndexModel : PageModel
   {

      public void OnPost()
      {
         MyMethod();
      }

      void MyMethod()
      {
         string fullName;
         string eMail;

         fullName = Request.Form["FullName"];
         eMail = Request.Form["EMail"];

         ViewData["UserData"] = "User Information: " + fullName + " - " + eMail;
      }
   }
}
```

Request.Query[]

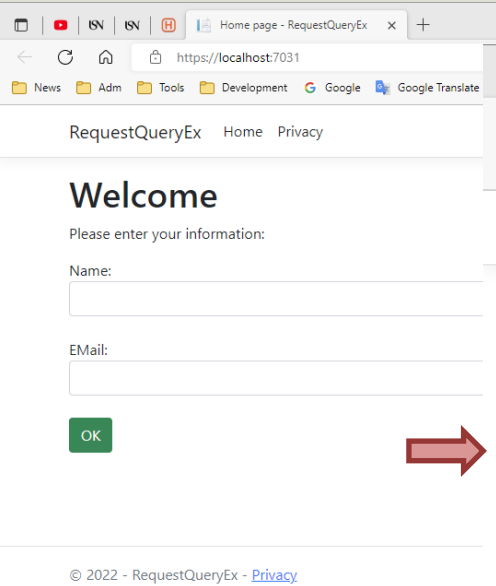
Example #2 – get and OnGet()

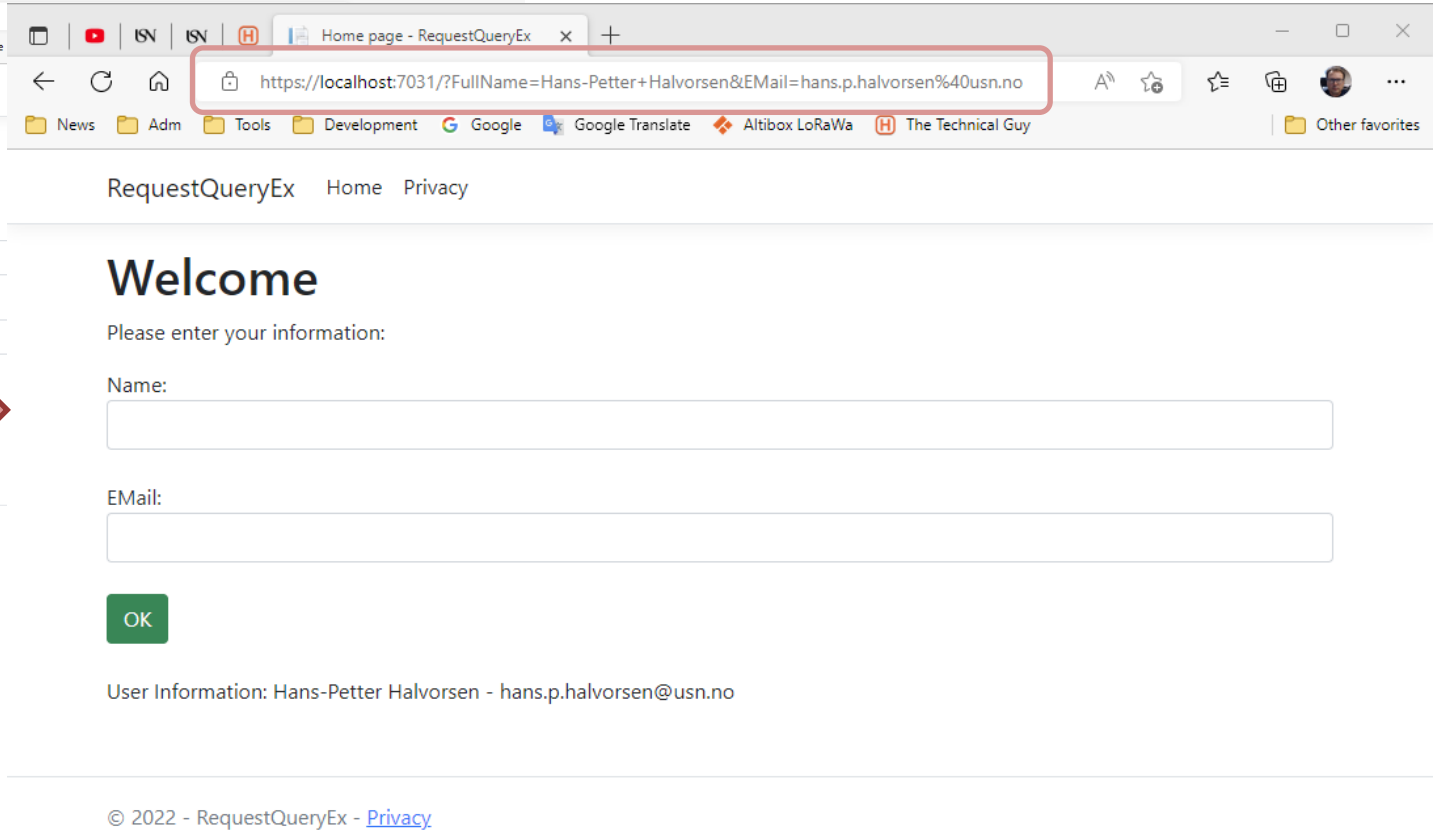
Hans-Petter Halvorsen

Table of Contents

Application

GUI is the same as in previous example





Explainations

- The HTML <form> element is used to create a Form on a web page.
- The form can have different attributes like **method** and **action**
- The method attribute determines the HTTP verb to use when the form is submitted, typically **get** or **post**
- If the **get** verb is used, the form values are appended to the receiving page's URL as query string values
- If the **action** attribute is omitted, the form will be submitted to the current URL i.e., the page that the form is in.

Index.cshtml

```
@page
@model IndexModel
@{
   ViewData["Title"] = "Home page";
}

<div>

   <h1>Welcome</h1>

   <p>Please enter your information:</p>

   <form name="NameForm" id="NameForm" method="get">

      <label for="FullName">Name:</label>
      <input name="FullName" type="text" class="form-control input-lg" autofocus required />
      <br />

      <label for="EMail">EMail:</label>
      <input name="EMail" type="email" class="form-control input-lg" required />
      <br />

      <input id="OKButton" type="submit" value="OK" class="btn btn-success" />

   </form>
   <br />

   <p>@ViewData["UserData"] </p>

</div>
```

An **HTML Form** is used to collect user input. The user input is most often sent to a server for processing.

If the **get** verb is used, the form values are appended to the receiving page's URL as query string values

Index.cshtml.cs

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace RequestQueryEx.Pages
{
   public class IndexModel : PageModel
   {

      public void OnGet()
      {
         string fullName;
         string eMail;

         fullName = Request.Query["FullName"];
         eMail = Request.Query["EMail"];

         if (!string.IsNullOrEmpty(fullName))
            ViewData["UserData"] = "User Information: " + fullName + " - " + eMail;
      }
   }
}
```

Since we have used **method="get"**, we need to use the **OnGet()** method

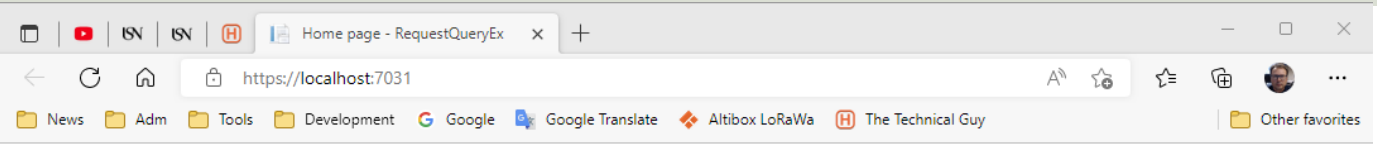
Request.Query[]

Example #2b

Hans-Petter Halvorsen

Table of Contents

Application



RequestQueryEx Home Privacy

Welcome

Please enter your information:

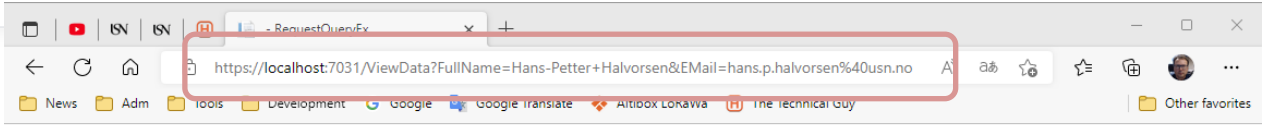
Name:

Hans-Petter Halvorsen

EMail:

hans.p.halvorsen@usn.no

OK



RequestQueryEx Home Privacy

View Data

Information entered in the previous webpage:

User Information: Hans-Petter Halvorsen - hans.p.halvorsen@usn.no

In this Example we send the Form data to another Web Page

© 2022 - RequestQueryEx - Privacy

Explainations

- The HTML <form> element is used to create a form on a web page.
- The form can have different attributes like **method** and **action**
- The method attribute determines the HTTP verb to use when the form is submitted, typically **get** or **post**
- If the **get** verb is used, the form values are appended to the receiving page's URL as query string values
- If the **action** is set to another webpage, the form will be submitted to that URL

ViewData.cshtml

```
@page
@model RequestQueryEx.Pages.ViewDataModel
@{
}

<div>

    <h1>View Data</h1>

    <p>Information entered in the previous webpage:</p>

    <p> @ViewData["UserData"] </p>

</div>
```

ViewData.cshtml.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace RequestQueryEx.Pages
{
   public class ViewDataModel : PageModel
   {
      public void OnGet()
      {
         string fullName;
         string eMail;

         fullName = Request.Query["FullName"];
         eMail = Request.Query["EMail"];

         if (!string.IsNullOrEmpty(fullName))
            ViewData["UserData"] = "User Information: " + fullName + " - " + eMail;
      }

   }
}
```

Index.cshtml

Here we use the action to send the Form Data to another Web Page (ViewData.cshtml)

```
@page
@model IndexModel
@{
   ViewData["Title"] = "Home page";
}

<div>

   <h1>Welcome</h1>

   <p>Please enter your information:</p>

   <form name="NameForm" id="NameForm" method="get" action="ViewData">

      <label for="FullName">Name:</label>
      <input name="FullName" type="text" class="form-control input-lg" autofocus required />
      <br />

      <label for="EMail">EMail:</label>
      <input name="EMail" type="email" class="form-control input-lg" required />
      <br />

      <input id="OKButton" type="submit" value="OK" class="btn btn-success" />

   </form>
   <br />

   <p>@ViewData["UserData"] </p>

</div>
```

Index.cshtml.cs

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace RequestQueryEx.Pages
{
    public class IndexModel : PageModel
    {


    }
}
```

Here we do nothing

https://www.halvorsen.blog

ASP.NET Core Data Binding

Example #3

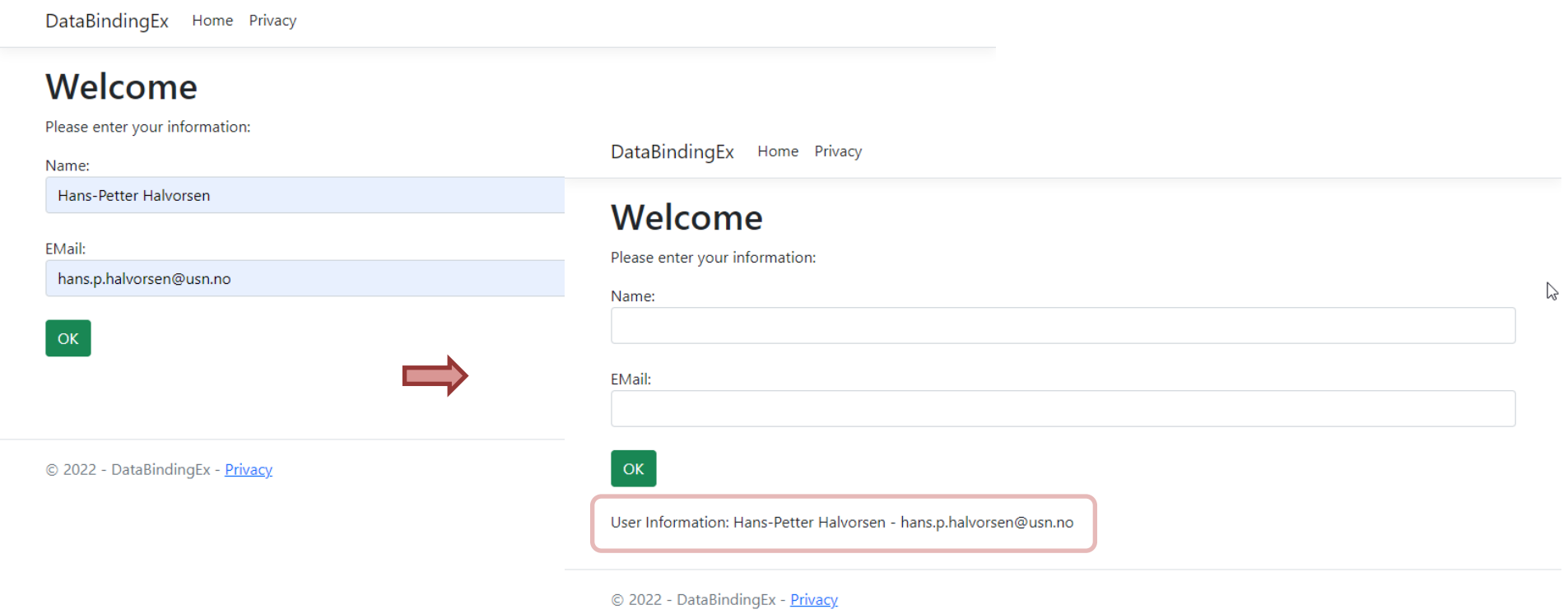
Hans-Petter Halvorsen

Table of Contents

ASP.NET Core Data Binding

- In this Tutorial we will see how we can get user data from the client-side to the server-side in ASP.NET Core
- We started with the traditional way used in many Web Programming Frameworks
  - Request.Form[]
  - Request.Query[]
- Now we will use an alternative approach which is included in the ASP.NET Core Framework

Application



Index.cshtml

```
@page
@model IndexModel
@{
   ViewData["Title"] = "Home page";
}

<div>

   <h1>Welcome</h1>

   <p>Please enter your information:</p>

   <form name="NameForm" id="NameForm" method="post">

      <label for="FullName">Name:</label>
      <input name="FullName" type="text" class="form-control input-lg" autofocus required />
      <br />

      <label for="EMail">EMail:</label>
      <input name="EMail" type="email" class="form-control input-lg" required />
      <br />

      <input id="OKButton" type="submit" value="OK" class="btn btn-success" />

   </form>
   <br />

   <p>@ViewData["UserData"] </p>

</div>
```

Index.cshtml.cs

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace DataBindingEx.Pages
{
   public class IndexModel : PageModel
   {
      public void OnPost(string fullName, string eMail)
      {
         ViewData["UserData"] = "User Information: " + fullName + " - " + eMail;
      }
   }
}
```

By using the built-in Data Binding in ASP.NET Core the Form Data will automatically be assigned to these parameters

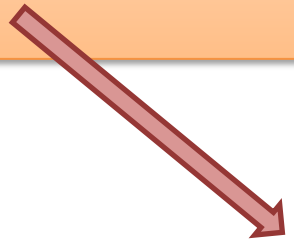
Explainations

```
<form name="NameForm" id="NameForm" method="post">

    <input name="FullName" type="text" />
    <input name="EMail" type="email" />

    <input id="OKButton" type="submit" value="OK"/>

</form>
```



Data Binding: ASP.NET Core automatically assign the Form Data to these parameters

```
public void OnPost(string fullName, string eMail)
{
   ..Use variables fullName and eMail..
}
```

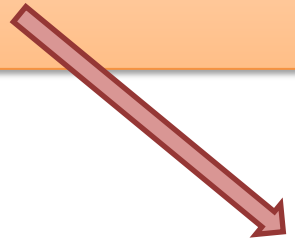
This will also work for get/OnGet()

```
<form name="NameForm" id="NameForm" method="get">

    <input name="FullName" type="text" />
    <input name="EMail" type="email" />

    <input id="OKButton" type="submit" value="OK"/>

</form>
```



Data Binding: ASP.NET Core automatically assign the Form Data to these parameters

```
public void OnGet(string fullName, string eMail)
{
   ..Use variables fullName and eMail..
}
```

https://www.halvorsen.blog

ASP.NET Core Data Binding

Example #3b (BindProperty)

Hans-Petter Halvorsen

Table of Contents

Index.cshtml.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace DataBindingEx.Pages
{
   public class IndexModel : PageModel
   {
      [BindProperty]
      public string? FullName { get; set; }
      [BindProperty]
      public string? EMail { get; set; }

      public void OnPost()
      {
         ViewData["UserData"] = "User Information: " + FullName + " - " + EMail;
      }
   }
}
```

You can also use the [**BindProperty**] and create Properties in the PageModel class

https://www.halvorsen.blog

ASP.NET Core Data Binding

Example #3c (BindProperties)

Hans-Petter Halvorsen

Table of Contents

Index.cshtml.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace DataBindingEx.Pages
{
    [BindProperties]
    public class IndexModel : PageModel
    {
        public string? FullName { get; set; }
        public string? EMail { get; set; }

        public void OnPost()
        {
            ViewData["UserData"] = "User Information: " + FullName + " - " + EMail;
        }
    }
}
```

You can add [**BindProperties**] attribute to the PageModel class rather than applying [BindProperty] to individual properties, which results in all the public properties in the PageModel taking part in data binding

ASP.NET Core Data Binding

Example #3d (Create and Use a Class)

Hans-Petter Halvorsen

Table of Contents

Index.cshtml.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace DataBindingEx.Pages
{
    public class IndexModel : PageModel
    {
        [BindProperty]
        public Person? person { get; set; }

        public void OnPost()
        {
            ViewData["UserData"] = "User Information: " + person.FullName + " - " + person.EMail;
        }
    }

    public class Person
    {
        public string? FullName { get; set; }
        public string? EMail { get; set; }
    }
}
```

If there are many fields in the Form, you may want to create and use a **Class**. Then all the Form fields are automatically assigned to the Class Properties

Summary

- In this Tutorial we have seen how we can get user data from the client-side to the server-side in ASP.NET Core
- We started with the traditional way used in many Web Programming Frameworks
  - Request.Form[]
  - Request.Query[]
- Finally, we used an alternative Data Binding approach which is part of the ASP.NET Core Framework

Resources

- Introduction to Razor Pages in ASP.NET Core: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/
- Learn Razor Pages: https://www.learnrazorpages.com/
- Model Binding: https://www.learnrazorpages.com/razor-pages/model-binding

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog