Mariusz Wieczorek

[Adres e-mail]

Streszczenie

[Przyciągnij uwagę czytelnika interesującym streszczeniem. Zazwyczaj jest to krótkie podsumowanie dokumentu.   
Aby dodać zawartość, wystarczy kliknąć tutaj i zacząć wpisywanie.]

ASP.NET MVC

Razor

**Razor**

Widoki w ASP.NET MVC są pisane w Razorze i mają rozszerzenie cshtml.  
**Razor** jest to silnik generujący html, łączy on ze sobą html i c#.  
Piszemy w html, a jeżeli chcemy się przełączyć na c# używamy znacznika @.

|  |
| --- |
| <head>  <meta charset="utf-8" />  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>@ViewBag.Title - My ASP.NET Application</title>  @Styles.Render("~/Content/css")  @Scripts.Render("~/bundles/modernizr")  </head> |

Za pomocą **ViewBag** przekazujemy zmienne z kontrolera lub z widoku w c#  
Następnie załączamy css’y i skrypty skonfigurowane w **App\_Start/BundleConfig.cs**

Używamy helper’ów generujących linki:  
przypisujemy do atrybutu href adres wygenerowany na podstawie nazwy akcji i kontrolera.

|  |
| --- |
| <a class="navbar-brand" href="@Url.Action("Index","Home")">Strona Główna</a> |

Wyświetlamy link razem z całym znacznikiem <a></a>

|  |
| --- |
| @Html.ActionLink("O mnie", "About", "Home", null, htmlAttributes: new { @class = "nav-link" }) |

Wyświetlamy widok cząstkowy

|  |
| --- |
| @Html.Partial("\_LoginPartial") |

Przełączamy się na c# za pomocą znacznika @

|  |
| --- |
| @using Microsoft.AspNet.Identity  @if (Request.IsAuthenticated) |

**BeginForm** tworzy formularz z danymi, po którym zostanie wykonana metoda typu **Post** w kontrolerze **Account** i Akcja **LogOff**

|  |
| --- |
| using (Html.BeginForm("LogOff", "Account", FormMethod.Post, new { id = "logoutForm", @class = "navbar-right" })) |

RenderBody() – generuje widok w obrębie Layoutu

|  |
| --- |
| @RenderBody() |

Dodajemy kolejne skrypty skonfigurowane w **App\_Start/BundleConfig.cs**  
I na końcu dodatkowe skrypty, które nie są wymagane.

|  |
| --- |
| @Scripts.Render("~/bundles/jquery")  @Scripts.Render("~/bundles/bootstrap")  @RenderSection("scripts", required: false) |

Register.cshtml wywołujemy w kontrolerze Account w widoku może być przekazany jakiś model.

|  |
| --- |
| // GET: /Account/Register  [AllowAnonymous]  public ActionResult Register()  {  return View();  } |

Skrypt może przyjmować model, jeżeli nie zostanie przekazany, to zostanie utworzony nowy pusty model.

|  |
| --- |
| @model InvoiceManager.Models.RegisterViewModel |

Kod c#, użycie zmiennej tekst

|  |
| --- |
| @{  ViewBag.Title = "Register";  var tekst = "11111";  }  <h4>Create a new account. @tekst</h4> |

Wyświetlamy błędy walidacyjne

|  |
| --- |
| @Html.ValidationSummary("", new { @class = "text-danger" }) |

Label + pole tekstowe

|  |
| --- |
| <div class="form-group">  @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })  </div>  </div> |

Wygeneruje kod html taki jak poniżej

|  |
| --- |
| <div class="form-group">  <label class="col-md-2 control-label" for="Email">Email</label>  <div class="col-md-10">  <input class="form-control" data-val="true"  data-val-email="The Email field is not a valid e-mail address."  data-val-required="The Email field is required."  id="Email"  name="Email"  type="text"  value="" />  </div>  </div> |

Przycisk

|  |
| --- |
| <div class="form-group">  <div class="col-md-offset-2 col-md-10">  <input type="submit" class="btn btn-default" value="Register" />  </div>  </div> |

Odwołanie się do modelu w kodzie cshtml

|  |
| --- |
| @using InvoiceManager.Models  @model LoginViewModel  @{  ViewBag.Title = "Log in";  }  <h2>@ViewBag.Title.</h2>  <div class="row">  <div>  @Model.Email  </div> |

Dodajemy dodatkowe dane do użytkownika w formularzu register.

|  |
| --- |
| @model InvoiceManager.Models.RegisterViewModel |

|  |
| --- |
| namespace InvoiceManager.Models  {  public class RegisterViewModel  {  [Required]  [EmailAddress]  [Display(Name = "Email")]  public string Email { get; set; }  [Required]  [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.",  MinimumLength = 6)]  [DataType(DataType.Password)]  [Display(Name = "Password")]  public string Password { get; set; }  [DataType(DataType.Password)]  [Display(Name = "Confirm password")]  [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]  public string ConfirmPassword { get; set; } |
| public string Name { get; set; }  public Address Address { get; set; } |
| }  } |

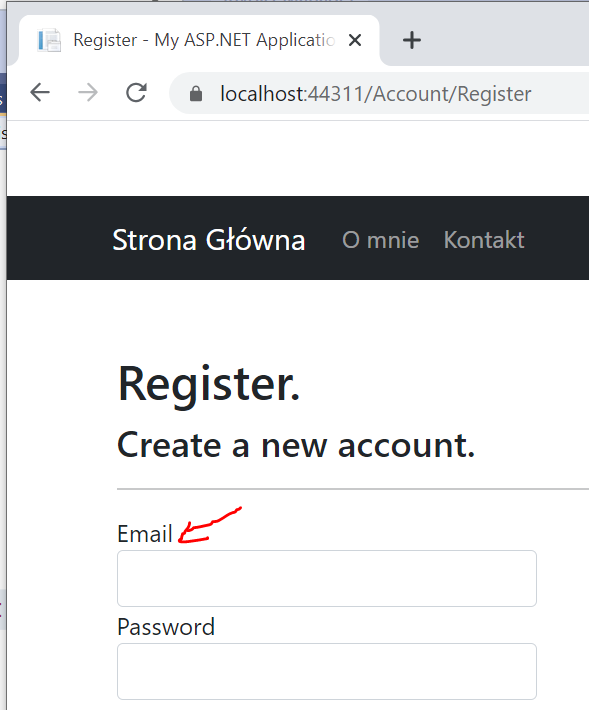
W widoku dodajemy

|  |
| --- |
| <div class="form-group">  @Html.LabelFor(m => m.Name, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.TextBoxFor(m => m.Name, new { @class = "form-control" })  </div>  </div>  <div class="form-group">  @Html.LabelFor(m => m.Address.City, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.TextBoxFor(m => m.Address.City, new { @class = "form-control" })  </div>  </div>  <div class="form-group">  @Html.LabelFor(m => m.Address.PostalCode, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.TextBoxFor(m => m.Address.PostalCode, new { @class = "form-control" })  </div>  </div>  <div class="form-group">  @Html.LabelFor(m => m.Address.Street, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.TextBoxFor(m => m.Address.Street, new { @class = "form-control" })  </div>  </div>  <div class="form-group">  @Html.LabelFor(m => m.Address.Number, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.TextBoxFor(m => m.Address.Number, new { @class = "form-control" })  </div>  </div> |

Zmiany w kontrolerze

Właściwość Display odpowiada za treść wyświetlaną w labelu

|  |
| --- |
| public class RegisterViewModel  {  [Required]  [EmailAddress]  [Display(Name = "Email")]  public string Email { get; set; } |



|  |
| --- |
| [DataType(DataType.Password)]  [Display(Name = "Potwierdź Hasło")]  [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]  public string ConfirmPassword { get; set; }  [Display(Name = "Nazwa Użytkownika")]  public string Name { get; set; }  public Address Address { get; set; } |

Dodajemy opis do pola Nazwa  
Natomiast opis do pól adresowych uzupełniamy w klasie Address

|  |
| --- |
| public class Address  {  public Address()  {  Clients = new Collection<Client>();  ApplicationUsers = new Collection<ApplicationUser>();  }  public int Id { get; set; }  [Required]  [Display(Name = "Ulica")]  public string Street { get; set; }  [Required]  [Display(Name = "Numer Domu")]  public string Number { get; set; }  [Required]  [Display(Name = "Miasto")]  public string City { get; set; }  [Required]  [Display(Name = "Kod Pocztowy")]  public string PostalCode { get; set; } |