**Walidacja danych**

Zp\_9\_20

W kontrolerze walidujemy każdą akcję typu Post, gdy przekazany jest cały obiekt danej klasy.  
Możemy dokonać walidacji dzięki właściwości **ModelState.IsValid**

|  |
| --- |
| public ActionResult Invoice(Invoice invoice)  {  var userId = User.Identity.GetUserId();  invoice.UserId = userId;  if (!ModelState.IsValid)  {  var vm = PrepareInvoiceVm(invoice, userId);  return View("Invoice", vm);  }  if (invoice.Id == 0)  \_invoiceRepository.Add(invoice);  else  \_invoiceRepository.Update(invoice);  return RedirectToAction("Index");  } |

Jeżeli dane są ok, to pomijamy tę sekcję. Gdy dane nie są prawidłowe to przekierowujemy użytkownika z powrotem do tego samego widoku. Ale powinniśmy przekazać do tego widoku te dane, które zostały wcześniej wpisane w formularzu przez użytkownika.  
Jak walidować wiemy już na podstawie samych modeli.  
Możemy wprowadzać własne warunki walidacji w modelu za pomocą DataAnnotation.

Umieszczenie tego kodu powoduje, że w przypadku niewpisania kompletu danych strona się w tle przeładowuje, nie dając przy tym żadnego komunikatu.  
Aby wyświetlić efekt walidacji, w widoku korzystamy z helpera: **HtmlValidationMessageFor** jak w kodzie poniżej.

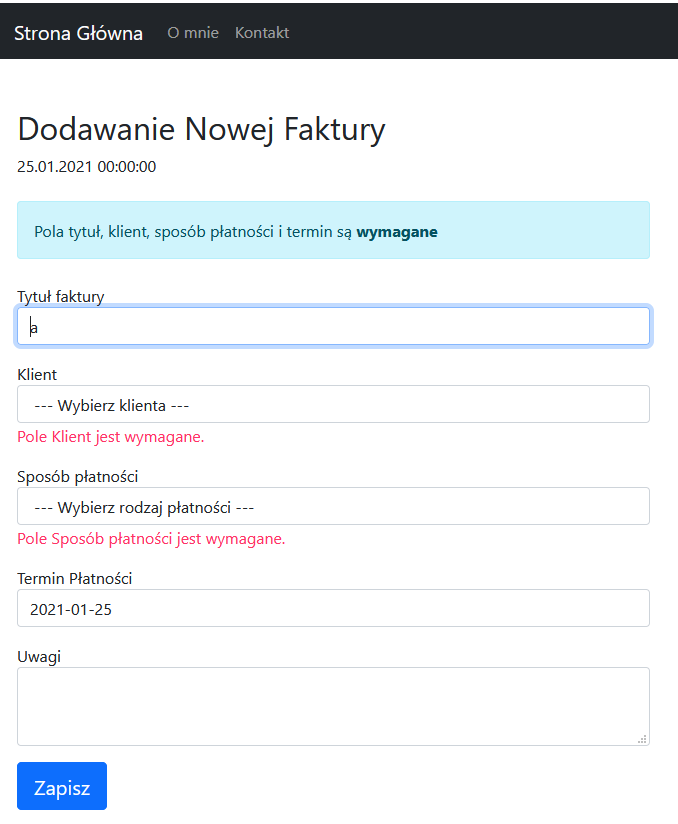
|  |
| --- |
| <div class="form-group mb-3">  @Html.LabelFor(x => x.Invoice.Title)  @Html.TextBoxFor(x => x.Invoice.Title,  new { @class = "form-control", autofocus = "autofocus" })  @Html.ValidationMessageFor(x => x.Invoice.Title)  </div> |



Wskazówka:  
Aby poprawić formatowanie wyświetlania kodu wciskamy kombinację klawiszy:

Ctrl + k + d

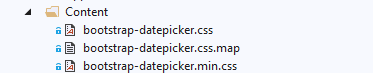
**Efekt walidacji na uruchomionym widoku**



Aby zmienić ten kolor uruchamiamy narzędzia developerskie w przeglądarce F12.  
Przechodzimy na zakładkę inspektor obiektów, klikamy na interesujący nas element na stronie.  
W okienku obok zostaną wyświetlone style powiązane z tym elementem, między innymi wartość oraz nazwa pliku css – w tym przypadku site.css



Ten plik odnajdziemy w katalogu content naszego projektu.



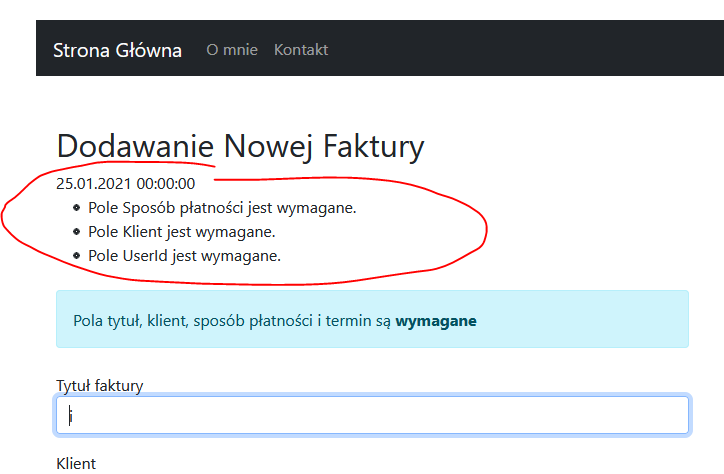
Dokonujemy wpisu

|  |
| --- |
| .field-validation-error {  color: #ff3366;  } |

Aby wyświetlić wszystkie błędy walidacji w jednym miejscu używamy helpera: **Html.ValidationSummary**

|  |
| --- |
| <div class="row">  <div class="col-lg-6">  @using (Html.BeginForm("Invoice", "Home"))  {  @Html.ValidationSummary()  if (Model.Invoice.Id == 0)  {  <p class="alert alert-info mb-4 mt-4">Pola tytuł… i termin są   <strong>wymagane</strong> </p>  }  @Html.HiddenFor(x => x.Invoice.Id) |

Efekt działanie jak poniżej:



Aby spersonalizować treść informacji o błędzie, np. wyświetlić je po polsku  
Musimy dokonać zmian w modelu.

|  |
| --- |
| namespace InvoiceManager.Models.Domains  {  public class Invoice  {  public Invoice()  {  InvoicePositions = new Collection<InvoicePossition>();  }  public int Id { get; set; }  [Required]  [Display(Name = "Tytuł faktury")] |

Możemy dodać parametr do atrybutu [Required]

|  |
| --- |
| public int Id { get; set; }  [Required(ErrorMessage ="Pole Tytuł jest wymagane !!!")]  [Display(Name = "Tytuł faktury")] |

Cała przeprowadzona walidacja jest jednak walidacją przeprowadzoną po stronie serwera.

Dobrą praktyką jest przeprowadzenie walidacji po stronie klienta, przed wysłaniem do serwera.  
Jeżeli jednak klient będzie miał wyłączone skrypty, to taka walidacja nie zadziała.

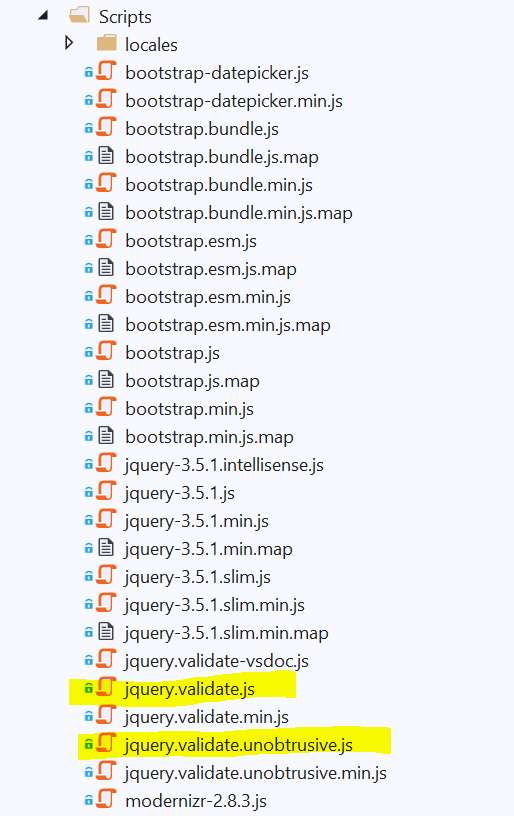
Musimy ustawić dwa przełączniki w pliku Web.config

|  |
| --- |
| <appSettings>  <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" />  </appSettings> |

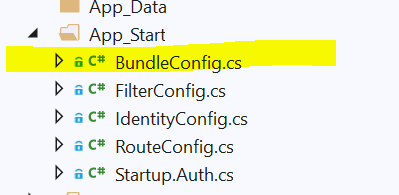
Do strony musimy dodać dwa skrypty: jQueryValidate.js oraz jQueryValidateUnobtrusive.js

**Unobtrusive** – nie rzucający się w oczy.

Na szczęście te skrypty są już dodane do projektu.



Mamy też przygotowanego odpowiedniego bundle: jqueryval



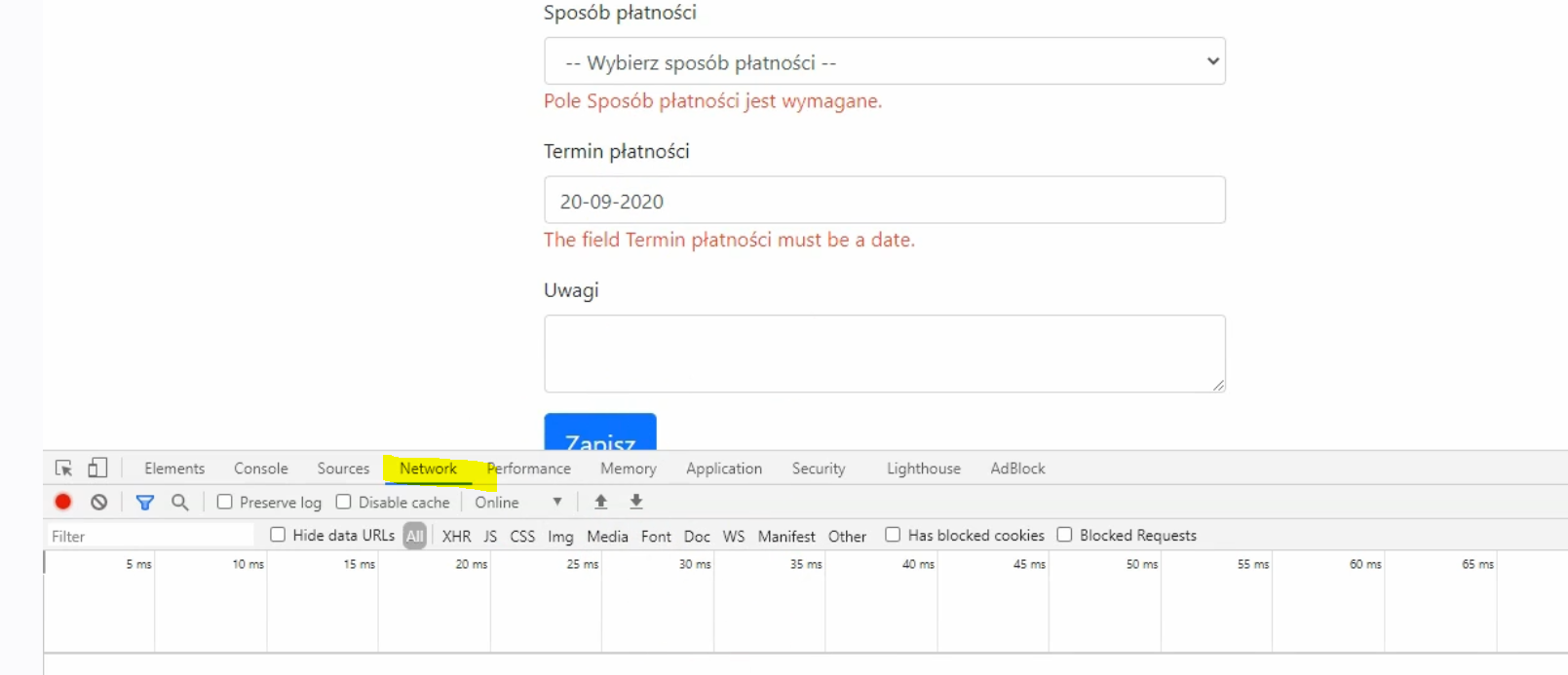
|  |
| --- |
| bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(  "~/Scripts/jquery.validate\*")); |

Ten bundle (paczka, pakiet) będzie zawierał wszystkie skrypty w katalogu Scripts, które zaczynają się od jquery.validate.

Teraz musimy dodać bundla do layoutu, do pliku shared/\_layout.cshtml.  
**Uwaga: ważna kolejność, musi byś za jquery!**

|  |
| --- |
| <div class="container body-content">  @RenderBody()  <hr />  <footer>  <p>&copy; @DateTime.Now.Year - Zarządzanie fakturami</p>  </footer>  </div>  @Scripts.Render("~/bundles/jquery")  @Scripts.Render("~/bundles/jqueryval")  @Scripts.Render("~/bundles/bootstrap")  @RenderSection("scripts", required: false)  </body>  </html> |

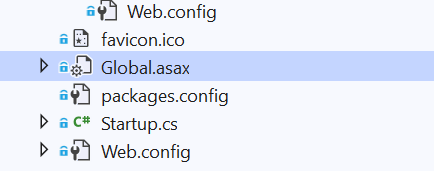
Teraz w narzędziach developerskich widzimy, że żaden pakiet nie został przesłany do serwera,  
wszystko dzieje się po stronie przeglądarki.



**Zmiany dotyczące formatu daty i liczb zmiennoprzecinkowych.**

Zmian musimy dokonać w dwóch miejscach.

Pierwsze miejsce to: Global.asax



Musimy dopisać metodę Application\_BeginRequest()

|  |
| --- |
| namespace InvoiceManager  {  public class MvcApplication : System.Web.HttpApplication  {  protected void Application\_Start()  {  AreaRegistration.RegisterAllAreas();  FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);  RouteConfig.RegisterRoutes(RouteTable.Routes);  BundleConfig.RegisterBundles(BundleTable.Bundles);  } |
| // metoda wywoływana przed każdym żądaniem  protected void Application\_BeginRequest(object sender, EventArgs e)  {  CultureInfo newCulture = (CultureInfo)Thread.CurrentThread.CurrentCulture.Clone();  newCulture.DateTimeFormat.ShortDatePattern = "dd-MM-yyyy";  newCulture.DateTimeFormat.DateSeparator = "-";  newCulture.NumberFormat.NumberDecimalSeparator = ",";  newCulture.NumberFormat.NumberDecimalDigits = 2;  newCulture.NumberFormat.NumberGroupSeparator = "";  Thread.CurrentThread.CurrentCulture = newCulture;  } |
| }  } |

Musimy jeszcze dokonać zmian po stronie widoku. W Invoice.cshtml

|  |
| --- |
|  |