Cel strona główna z listą faktur i przycisk dodający nową fakturę

|  |
| --- |
| @{  ViewBag.Title = "Faktury";  }  <h1>Faktury</h1>  <button class="btn btn-primary" onclick="@Url.Action("Invoice","Home")">Dodaj </button> |

Dodajemy nagłówek Faktury  
Oraz przycisk Dodaj  
Dodajemy do niego klasy bootstrap’owe oraz obsłużymy zdarzenie onclick()  
W zdarzeniu onclick przechodzimy do C# i wywołujemy Akcję Invoice w kontrolerze Home

W kontrolerze dodajemy Akcję Invoice()

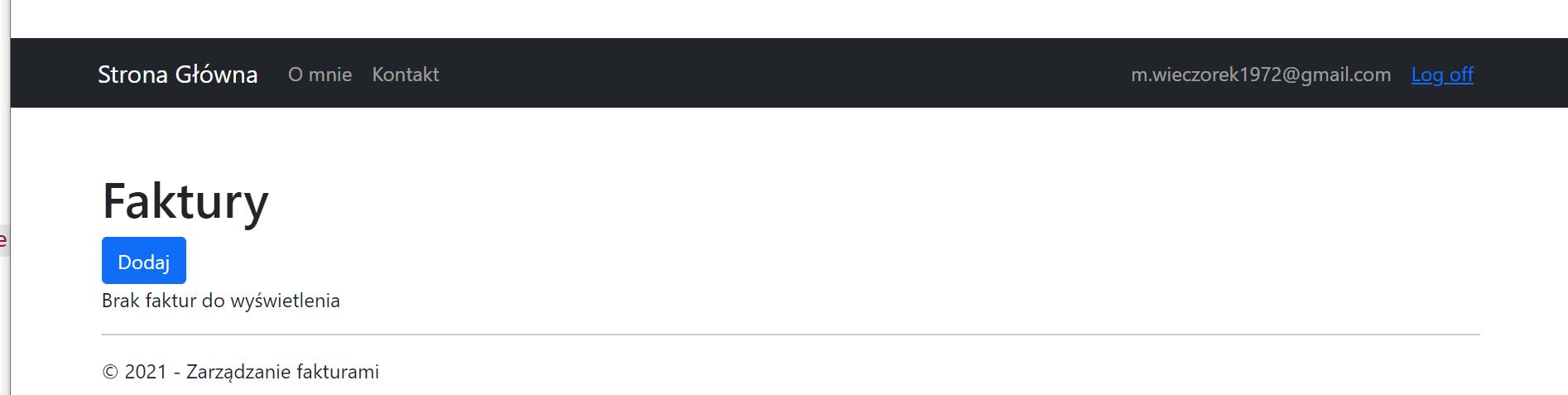
|  |
| --- |
| public class HomeController : Controller  {  public ActionResult Index()  {  return View();  }  public ActionResult Invoice()  {  return View();  } |

Na widoku listy faktur dodajemy tabelę z wszystkimi fakturami  
Musimy na górze widoku zdefiniować, że oczekujemy, że do tego widoku zostanie przekazany model lista faktur.

|  |
| --- |
| @model IEnumerable<InvoiceManager.Models.Domains.Invoice> |

Nie wpisujemy na sztywno, że oczekujemy jakieś konkretnej klasy np. listy List<>, ale oczekujemy dowolnej kolekcji, która implementuje interfejs IEnumerable.

|  |
| --- |
| @if (Model != null && Model.Any())  {  }  else  {  <div>  Brak faktur do wyświetlenia  </div>  } |

Jeżeli model nie jest pusty, oraz zawiera jakąkolwiek pozycję, to będziemy wyświetlać tabelkę,  
w przeciwnym przypadku wyświetlamy tekst Brak faktur do wyświetlenia.   
Wstawiając znacznik **<div>** dajemy Razorowi znać, że przechodzimy na kod html’a.  


|  |
| --- |
| <h1 class="mb-3">Faktury</h1>  <button class="btn btn-primary mb-3" onclick="@Url.Action("Invoice","Home")" >Dodaj</button> |

Do nagłówka h1 oraz przyciska dodajemy klasę mb-3 aby dodać dolny margines.

Tabela z fakturami

|  |
| --- |
| <table>  <thead>  <tr>  <th>tytuł</th>  <th>klient</th>  <th>wartość</th>  <th>data</th>  <th>termin</th>  <th> </th>  </tr>  </thead>  <tbody>  @foreach (var invoice in Model)  {  <tr>  <td>@invoice.Title</td>  <td>@invoice.Client.Name</td>  <td>@invoice.Value zł</td>  <td>@invoice.CreatedDate.ToString("dd-MM-yyyy")</td>  <td>@invoice.PaymentDate.ToString("dd-MM-yyyy")</td>  <td>  <button class="btn btn-danger btn-sm" onclick="deleteInvoice()">Usuń</button>  </td>  </tr>  }  </tbody>  </table> |

Chcemy usunąć pozycję bez przeładowywania strony. Zrealizujemy to za pomocą skrypty javascript, a konkretnie AJAX.

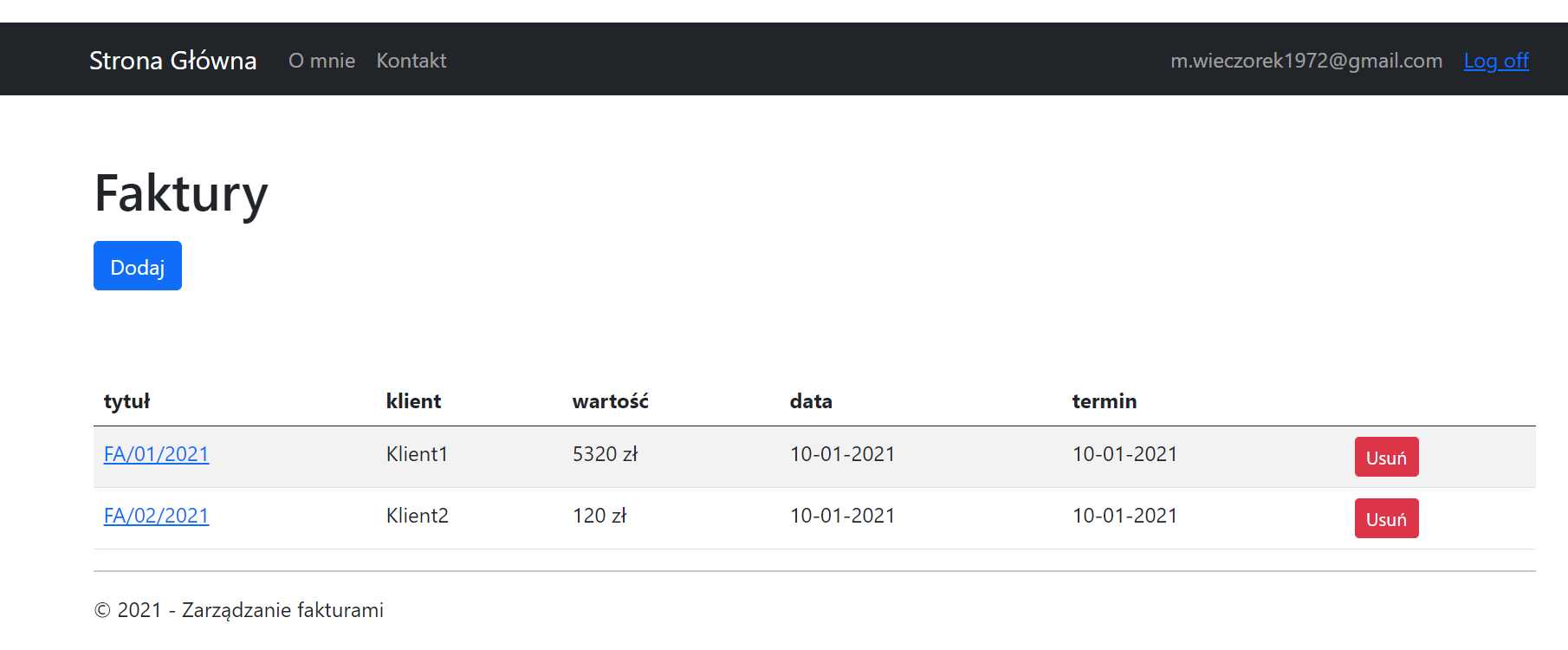
|  |
| --- |
| @section scripts  {  <script type="text/javascript">  function deleteInvoice() {  if (!confirm("Czy na pewno chcesz usunąć fakturę ?"))  return;  }  </script>  } |

Dodajemy sekcję scripts a w niej definicję metody napisanej w javascript deleteInvoice().  
Umieszczamy ją w sekcji script, ponieważ wszystko co tutaj wpiszemy zostanie wywołane  
w layout’cie w linii @RenderSection

|  |
| --- |
| <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/bootstrap")  @RenderSection("scripts", required: false)  </body>  </html> |

Aby przetestować musimy do widoku przekazać jakiś model w kontrolerze.

|  |
| --- |
| public ActionResult Index()  {  var invoices = new List<Invoice>  {  new Invoice  {  Id = 1,  Title = "FA/01/2021",  Value = 5320,  CreatedDate = DateTime.Now,  PaymentDate = DateTime.Now,  Client = new Client { Name = "Klient1" }  },  new Invoice  {  Id = 2,  Title = "FA/02/2021",  Value = 120,  CreatedDate = DateTime.Now,  PaymentDate = DateTime.Now,  Client = new Client { Name = "Klient2" }  }  };    return View(invoices);  } |



W tabeli dodajemy link pod pozycję w kolumnie tytuł, który zaprowadzi nas do edycji pozycji.  
Dodatkowy parametr w tym przypadku id faktury możemy przekazać w ciele funkcji, przekazując obiekt anonimowy.

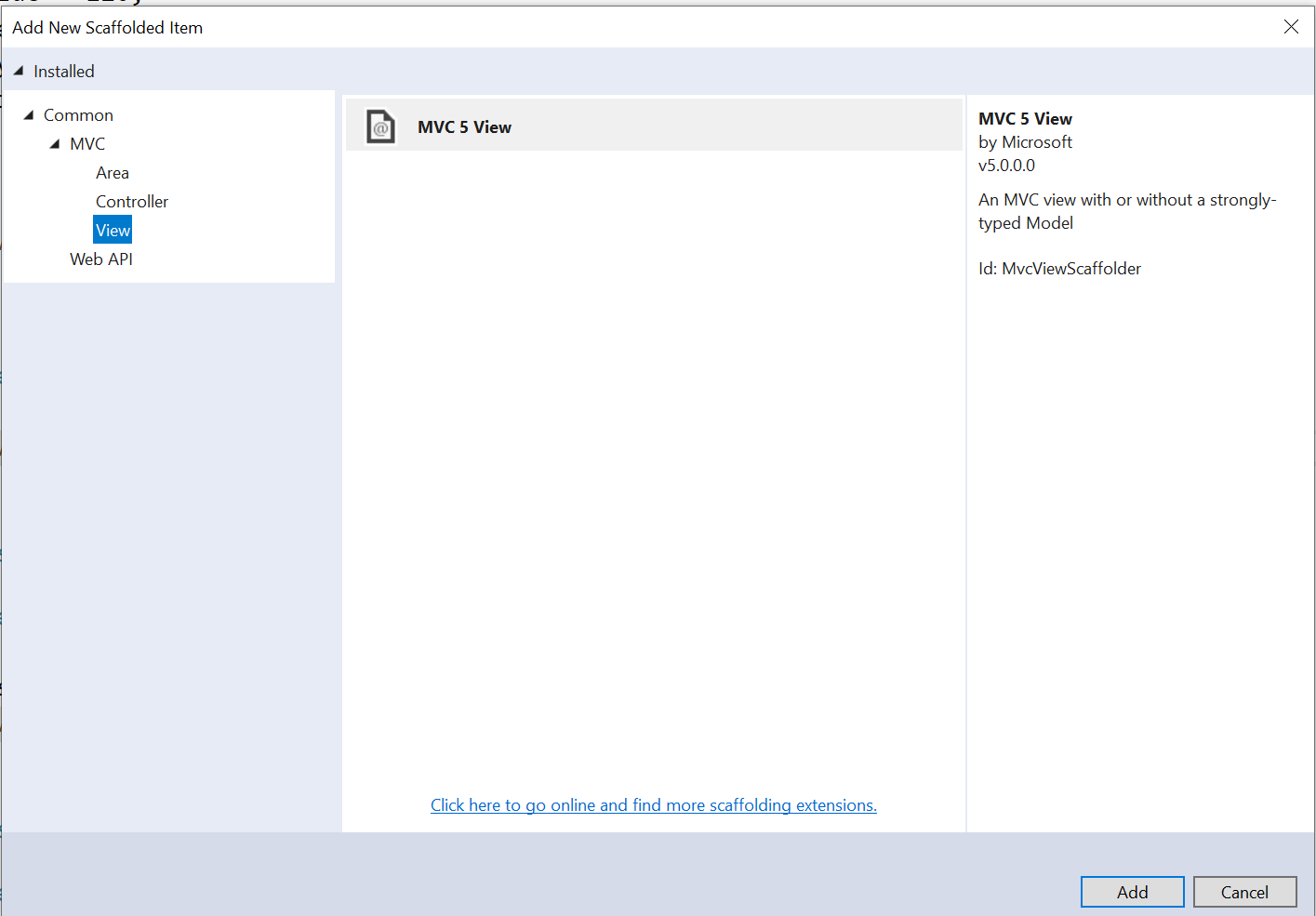
|  |
| --- |
| <td>  <a href="@Url.Action("Invoice","Home", new { id = invoice.Id })">@invoice.Title</a>  </td> |

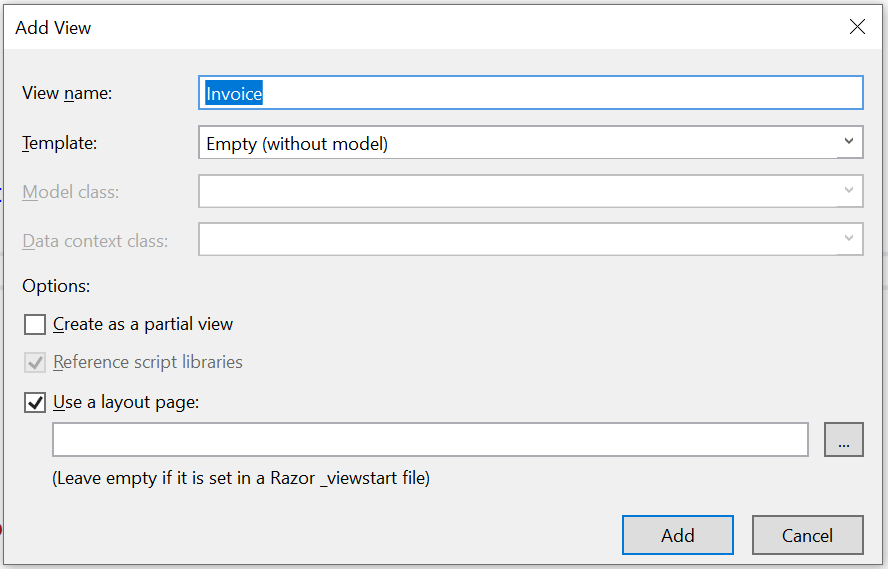
W kontrolerze będziemy oczekiwać id przekazanej faktury, domyślnie 0, jeżeli 0 to będzie dodanie nowej faktury

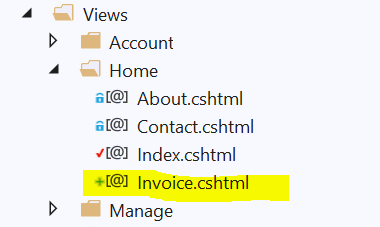
|  |
| --- |
| public ActionResult Invoice(int id = 0)  {  return View();  } |

**Dodawanie widoku.**

Możemy to zrobić również z poziomu kontrolera, z menu podręcznego wybierając Add View





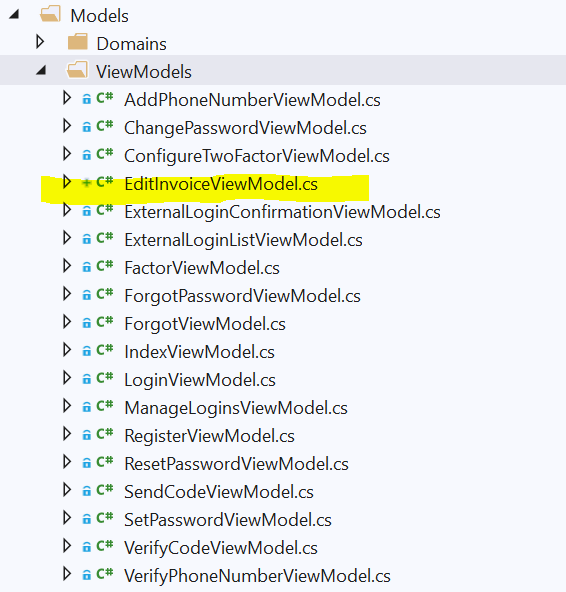


W katalogu Views / Home został utworzony widok Invoice

|  |
| --- |
| @{  ViewBag.Title = "Invoice";  }  <h2>Invoice</h2> |

Możemy dodać, że wymagamy przekazania modelu, faktura do edycji lub nowa faktura do uzupełnienia.  
Do każdej faktury będzie wybierany klient i płatność, będą to dane pobierane z bazy danych,  
zatem również musimy przekazać te listy do tego widoku.  
Zamiast oczekiwać widoku faktury – czyli modelu, będziemy oczekiwać ViewModelu,  
w którym będą zarówno wszystkie informacje o fakturze jak i dostępnych metodach płatności i liście klientów.

Tworzymy więc w katalogu ViewModel klasę EditInvoiceViewModel.cs



|  |
| --- |
| namespace InvoiceManager.Models.ViewModels  {  public class EditInvoiceViewModel  {  public Invoice Invoice { get; set; }  public List<Client> Clients { get; set; }  public List<MethodOfPayment> MethodOfPayments { get; set; }  public string Heading { get; set; }  }  } |

W przypadku edytowanej faktury przykazujemy również do widoku nagłówek edytowanej faktury.  
Chcemy aby po stronie widoku było jak najmniej logiki.

W modelu przekazujemy ViewModel

|  |
| --- |
| @model InvoiceManager.Models.ViewModels.EditInvoiceViewModel  @{  ViewBag.Title = "Invoice";  }  <h2>Invoice</h2> |

|  |
| --- |
| @if (Model.Invoice.Id != 0)  {  <button class="btn btn-primary mb-3"  onclick='window.location.href =   "@Url.Action("InvoicePossition","Home",new { invoiceId = Model.Invoice.Id })"  '>  Dodaj pozycję  </button>  <button class="btn btn-secondary mb-3"  onclick="generatePdf">  onclick=  Generuj PDF  </button>  } |

Tutaj musimy mieć kod, który nas przekieruje na tą akcję. Apostrofy ‘ na początku i końcu aby uniknąć problemów z parowaniem.  
Generowanie Pdf’a – skrypt w javascript. Więca na dole musimy dodać

|  |
| --- |
| @section scripts  {  <script type="text/javascript">  function generatePdf() {  }  </script>  } |

|  |
| --- |
| <div class="row">  <div class="col-lg-6">  </div>    <div class="col-5 offset-1">  </div>  </div> |

Formularz, jeżeli wciśniemy submit, to zostaniemy przekierowani do kontrolera Home i Akcji Invoice,  
ale będzie to Akcja HttpPost.  
Dodajemy formularz i pola zgodnie z dokumentacją bootstrap’a

|  |
| --- |
| @using (Html.BeginForm("Invoice", "Home"))  {  if (Model.Invoice.Id == 0)  {  <p class="alert alert-info mb-4 mt-4">Pola tytuł, klient, sposób płatności i termin są <strong>wymagane</strong> </p>  @Html.HiddenFor(x => x.Invoice.Id)  @Html.HiddenFor(x => x.Invoice.ClientId)  @Html.HiddenFor(x => x.Invoice.CreatedDate)  <div class="form-group">  @Html.LabelFor(x => x.Invoice.Title)  @Html.TextBoxFor(x => x.Invoice.Title,  new { @class = "form-control", autofocus = "autofocus" })  </div>  } |

HiddenFor – dla pól, które nie mają być wyświetlane, ale mają być przekazane.

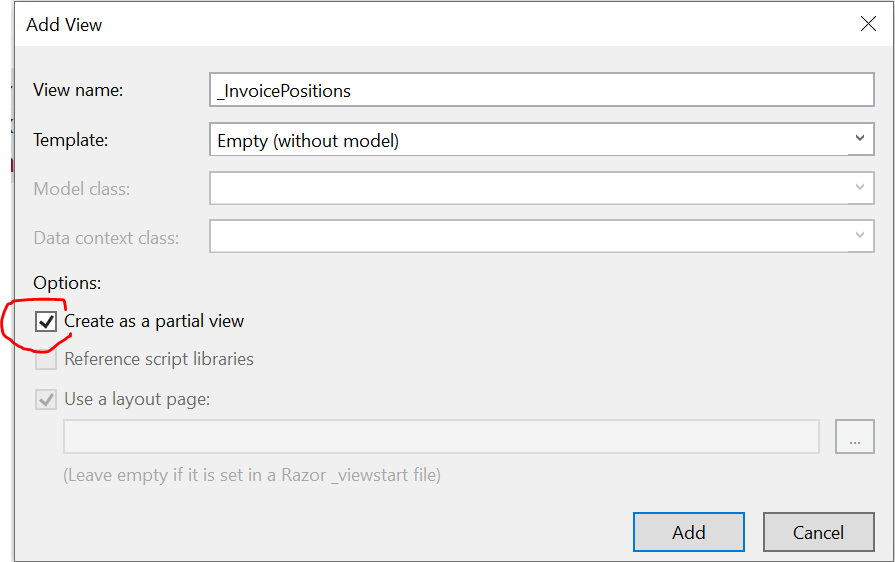
Cała pierwsza część formularza

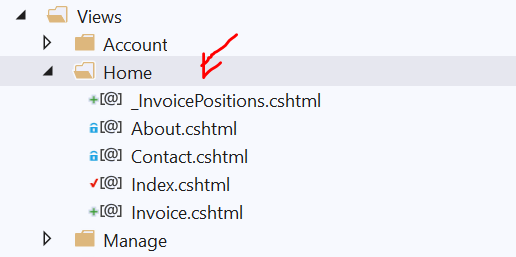
|  |
| --- |
| <div class="col-lg-6">  @using (Html.BeginForm("Invoice", "Home"))  {  if (Model.Invoice.Id == 0)  {  <p class="alert alert-info mb-4 mt-4">Pola tytuł, klient, sposób płatności i termin są <strong>wymagane</strong> </p>  @Html.HiddenFor(x => x.Invoice.Id)  @Html.HiddenFor(x => x.Invoice.ClientId)  @Html.HiddenFor(x => x.Invoice.CreatedDate)  <div class="form-group">  @Html.LabelFor(x => x.Invoice.Title)  @Html.TextBoxFor(x => x.Invoice.Title,  new { @class = "form-control", autofocus = "autofocus" })  </div>  <div class="form-group">  @Html.LabelFor(x => x.Invoice.ClientId)  @Html.DropDownListFor(x => x.Invoice.ClientId,  new SelectList(Model.Clients, "Id", "Name"),  "--- Wybierz klienta ---",  new { @class = "form-control", autofocus = "autofocus" })  </div>  if (Model.Invoice.Id != 0)  {  <div class="form-group">  @Html.LabelFor(x => x.Invoice.Value)  @Html.TextBoxFor(x => x.Invoice.Value,  new { @class = "form-control", @readonly = "readonly" })  </div>  }  <div class="form-group">  @Html.LabelFor(x => x.Invoice.MethodOfPaymentId)  @Html.DropDownListFor(x => x.Invoice.MethodOfPaymentId,  new SelectList(Model.MethodOfPayments, "Id", "Name"),  "--- Wybierz rodzaj płatności ---",  new { @class = "form-control" })  </div>  <div class="form-group">  @Html.LabelFor(x => x.Invoice.PaymentDate)  @Html.TextBoxFor(x => x.Invoice.PaymentDate,  "{0:dd-MM-yyyy}",  new { @class = "form-control" })  </div>  <div class="form-group">  @Html.LabelFor(x => x.Invoice.Comments)  @Html.TextAreaFor(x => x.Invoice.Comments,  new { @class = "form-control"})  </div>  <button class="btn btn-primary btn-lg mb-5" type="submit">  Zapisz  </button>  }  }  </div> |

W drugiej części ekranu wyświetlamy pozycje faktury.  
Realizujemy to za pomocą widoku częściowego

|  |
| --- |
| <div class="col-5 offset-1">  @if (Model.Invoice.Id != 0)  {  @Html.Partial("\_InvoicePositions",Model.Invoice.InvoicePositions)  }  </div> |

Dodajemy widok częściowy. Add View na drzewku widoków

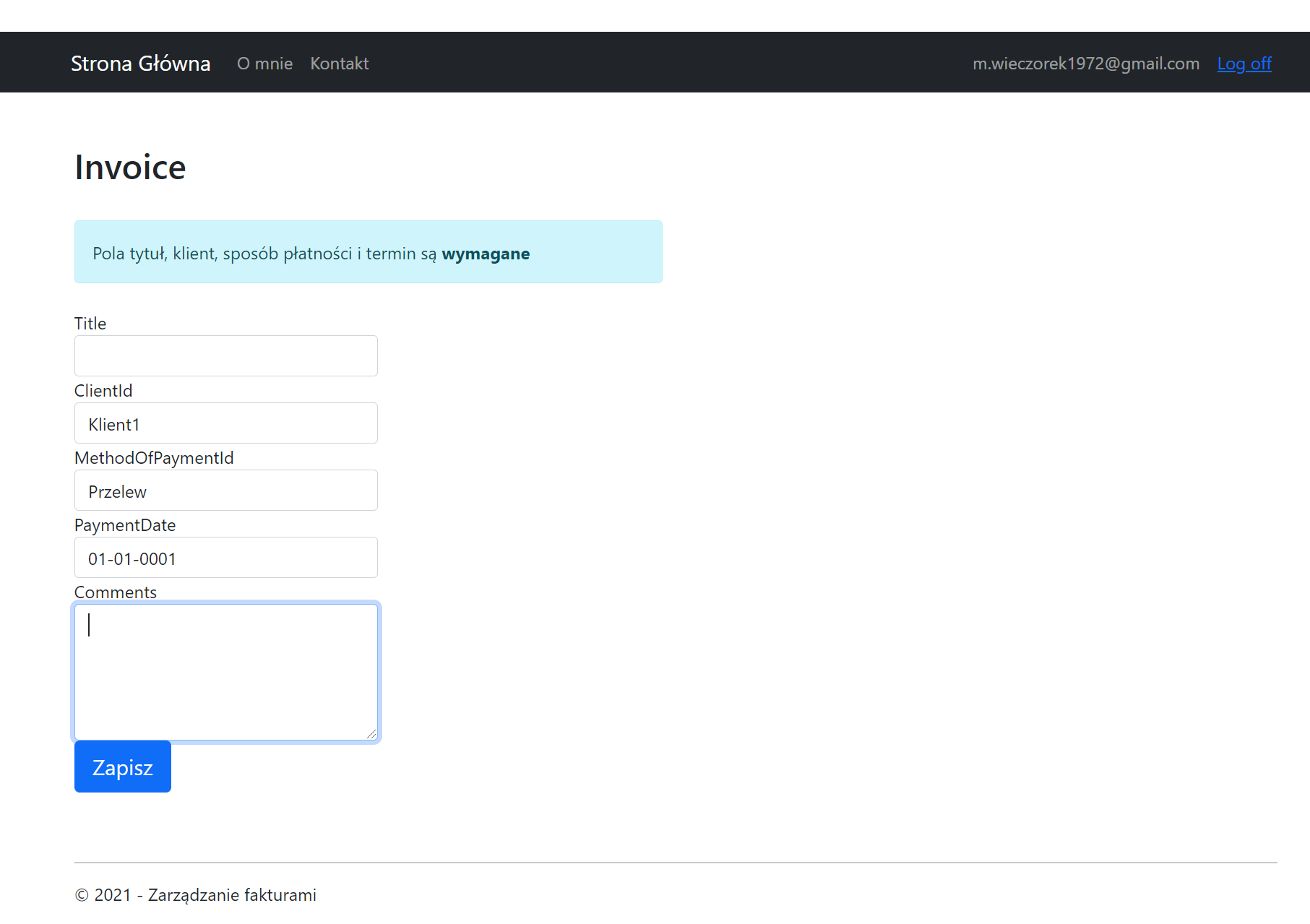




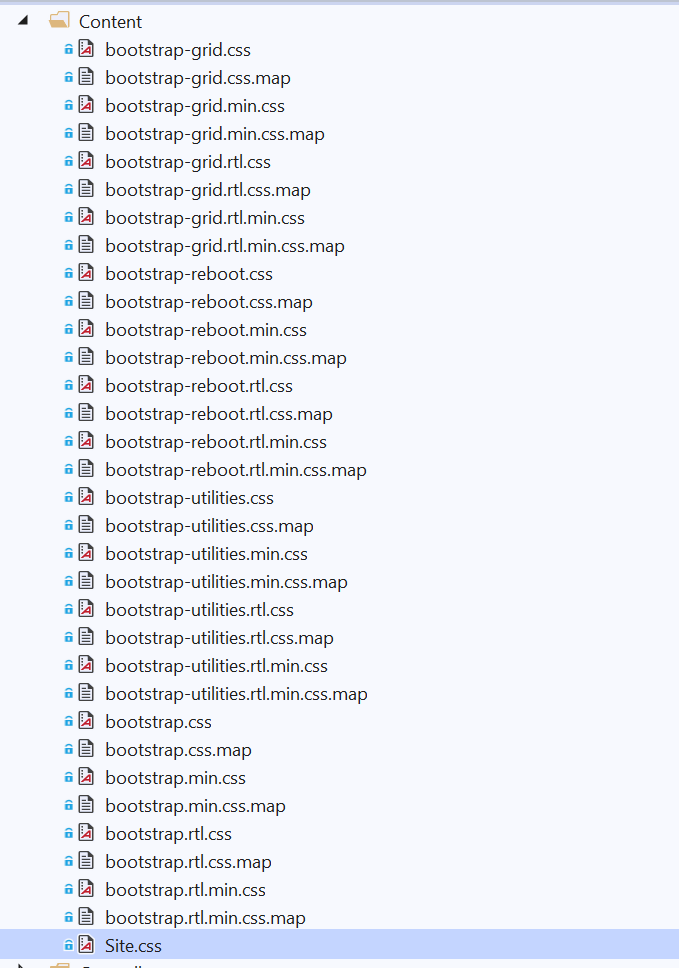
Konwencja jest taka, że widoki częściowe poprzedzamy podkreśleniem. Bez tego też to będzie działało, ale taka jest konwencja, której powinniśmy się stosować.

Przekazanie w kontrolerze ViewModelu

|  |
| --- |
| public ActionResult Invoice(int id = 0)  {  var vm = new EditInvoiceViewModel  {  Clients = new List<Client>  {  new Client {Id = 1, Name = "Klient1"}  },  MethodOfPayments = new List<MethodOfPayment>  {  new MethodOfPayment {Id = 1, Name = "Przelew" }  },  Heading = "Edycja Faktury 1",  Invoice = new Invoice()  };  return View(vm);  } |



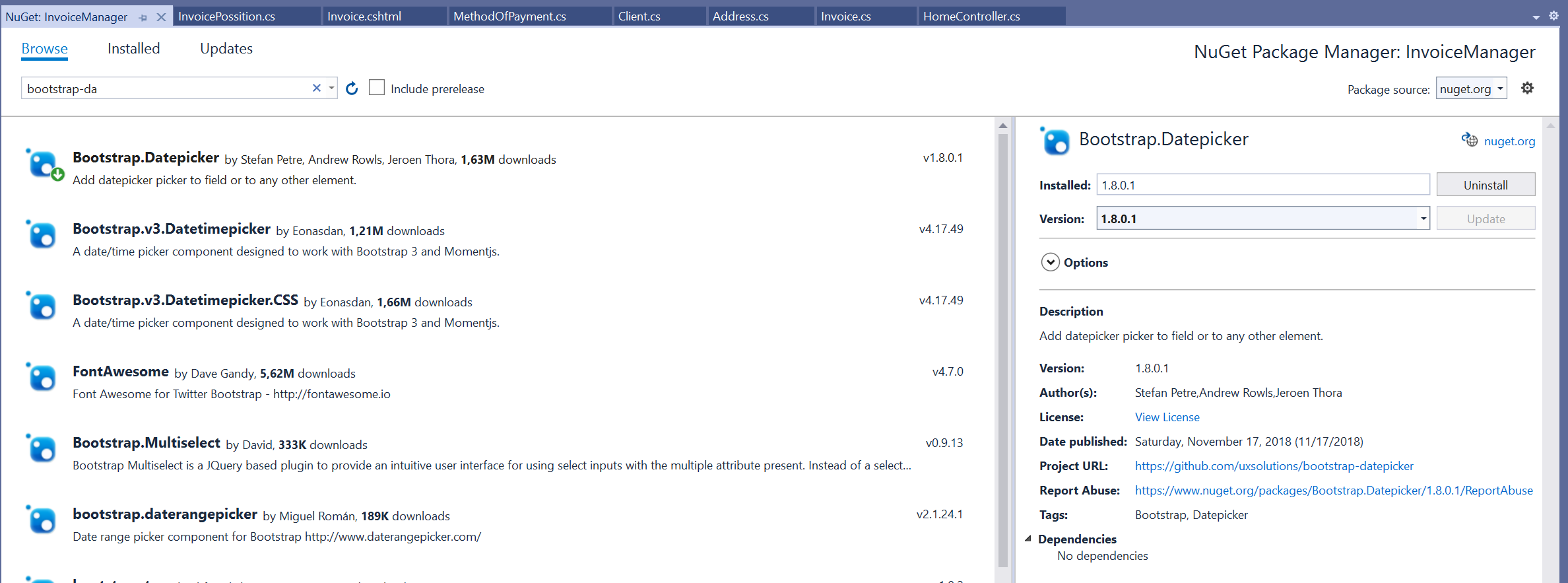
Poprawianie Wyświetlania

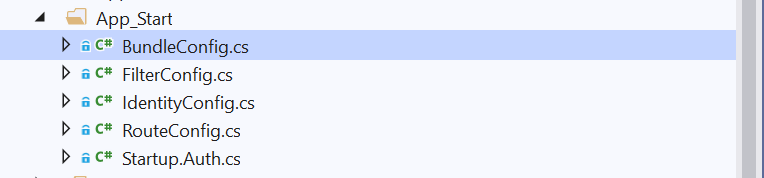


Usuwamy wpis:

|  |
| --- |
| /\* Set width on the form input elements since they're 100% wide by default \*/  input,  select,  textarea {  max-width: 280px;  } |

Do instalujemy bibliotekę nuget Bootstrap.Datepicker





|  |
| --- |
| bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(  "~/Scripts/bootstrap.js")); |

Dodajemy do istniejącego wpisu

|  |
| --- |
| bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(  "~/Scripts/bootstrap.js",  "~/Scripts/bootstrap-datepicker.js"));  bundles.Add(new StyleBundle("~/Content/css").Include(  "~/Content/bootstrap.css",  "~/Content/bootstrap-datepicker.css",  "~/Content/bootstrap-datepicker3.css",  "~/Content/site.css")); |

Dodajemy do widoku Invoice.cshtml wpis javascript

Zmiany przy kontrolce z datą

|  |
| --- |
| <div class="form-group mb-3">  @Html.LabelFor(x => x.Invoice.PaymentDate)  @Html.TextBoxFor(x => x.Invoice.PaymentDate,  "{0:dd-MM-yyyy}",  new { @class = "form-control" })  </div> |

|  |
| --- |
| <div class="form-group mb-3">  @Html.LabelFor(x => x.Invoice.PaymentDate)  @Html.TextBoxFor(x => x.Invoice.PaymentDate,  "{0:dd-MM-yyyy}",  new {  @class = "form-control",  @id = "PaymentDate",  autocomplete = "off"})  </div> |

I w skrypcie

|  |
| --- |
| @section scripts  {  <script type="text/javascript">  function generatePdf() {  }  </script>  } |

|  |
| --- |
| @section scripts  {  <script type="text/javascript">  function generatePdf() {  } |
| $(document).ready(function () {  $('#PaymentDate').datepicker({  format: "dd-mm-yyyy",  language: "pl",  multidate: false,  autoclose: true,  todayHighLight: true,  forceParse: false  })  }); |
| </script>  } |