Mariusz Wieczorek

mariusz.wieczorek@kabat.pl

Streszczenie

Tworzenie Widoków Tasks, Task  
przygotwanie modeli widoków  
nowość: formularz bez helpera beginform, obsługiwany za pomocą ajax’a  
bez przeładowywania strony

asp.net Core

Tworzenie Widoków

**Tworzymy Widoki**

**Analiza \_Layout.cshtml**

W widokach mamy również stosowaną składnie Razor, ale jest tutaj jeden inny zapis.  
Mamy coś takiego, co się nazywa **Anchor Tag Helper**

|  |
| --- |
| <li class="nav-item">  <a class="nav-link text-dark" **asp-area**="" **asp-controller**="Home" **asp-action**="Index">Home</a>  </li> |

Aby zostać przekierowanym do akcji Indeks w kontrolerze Home w linku wpisujemy   
**asp-controller**="Home" **asp-action**="Index". Dodatkowo możemy podać area.

Na górze w sekcji **head** mamy odwołania do css’ów.  
Odwołujemy się do **ViewData**, podajemy w nawiasach kwadratowych jako string klucza.

|  |
| --- |
| <head>  <meta charset="utf-8" />  <meta name="viewport" content="width=device-width, initial-scale=1.0" />  <title>@ViewData["Title"] - Moje Zadania</title>  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />  <link rel="stylesheet" href="~/css/site.css" />  </head> |

Inny zapis zastosowany jest również w przypadku załączania widoków częściowych, ale równie dobrze możemy stosować zapis za pomocą helpera tak jak robiliśmy to w aplikacji ASP.NET

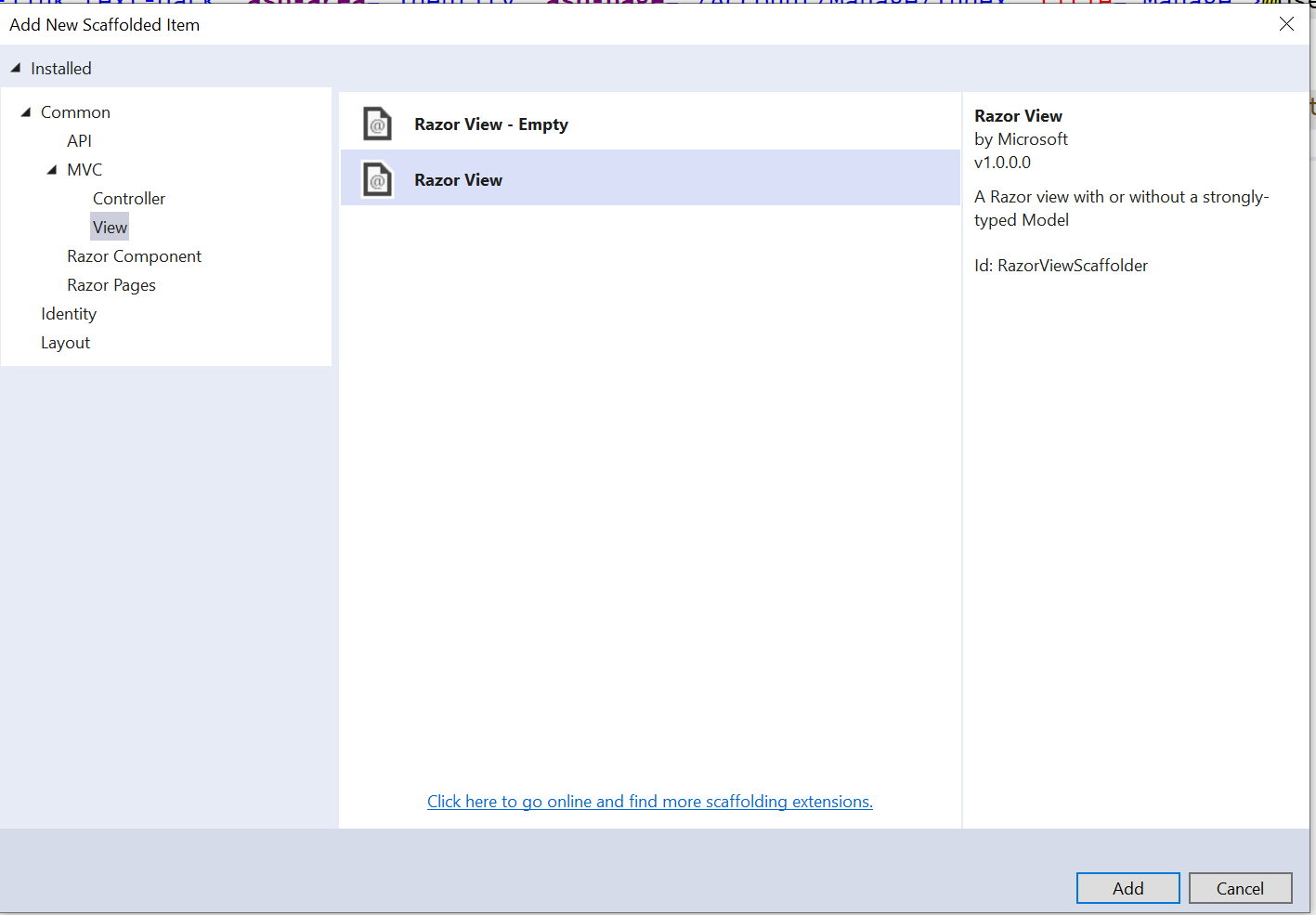
|  |
| --- |
| <**partial** **name**="\_LoginPartial" /> |

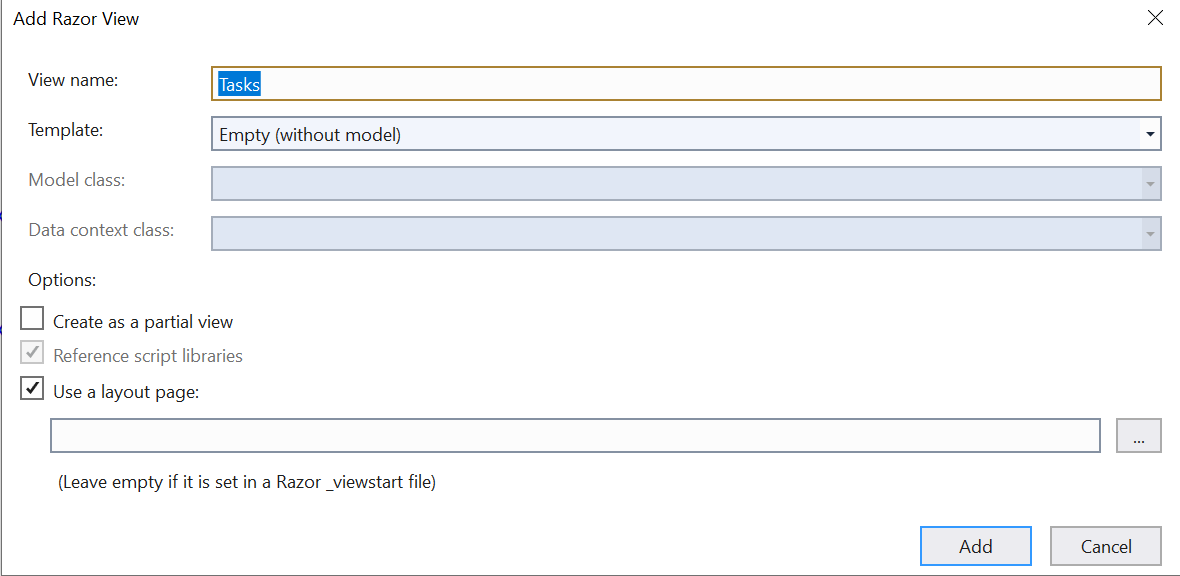
W tym widoku cząstkowym również stosujemy anchor tag helper’y.

|  |
| --- |
| <li class="nav-item">  <a class="nav-link text-dark" **asp-area**="Identity" **asp-page**="/Account/Manage/Index"   title="Manage">@User.Identity.Name</a>  </li> |

**Tworzymy nowy widok**

W folderze Views, tworzymy nowy folder Task  
A w nim nowy widok Tasks



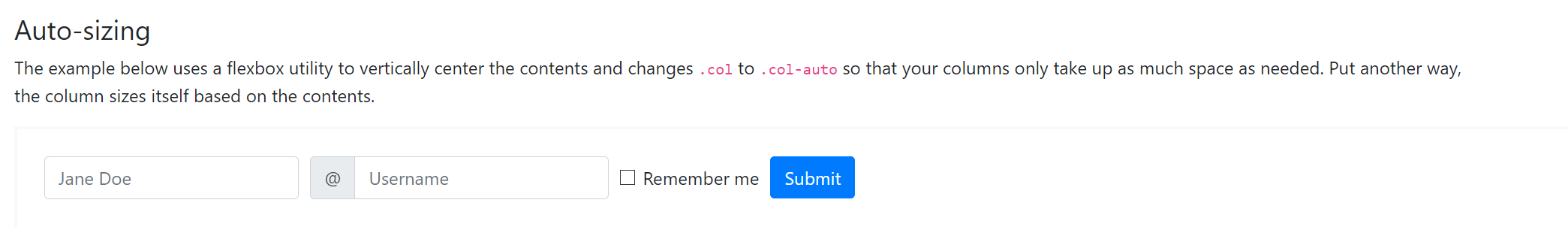


Został wygenerowany pusty widok tak jak poniżej.

|  |
| --- |
| @{  ViewData["Title"] = "Tasks";  }  <h1>Moje zadania</h1>  @\* formularz filtrowania \*@  @\* tabela z zadaniami \*@ |

Na górze wstawimy formatkę do filtrowania, wykorzystamy gotowy kod ze strony bootstrap’a.  
Przeglądamy dostępne formatki i wybieramy najbardziej pasujące.

<https://getbootstrap.com/docs/4.6/components/forms/>



Kopiujemy kod i wklejamy

**ViewModel**

Będziemy oczekiwać, że do tego widoku zostanie przekazana lista zadań, lista kategorii, oraz klasa zawierająca pola, które będą w filtrze.

Oczekujemy, aby lista zadań i lista kategorii były dowolną kolekcją, która implementuje   
interfejs **IEnumerable**. Jest to dobra praktyka.

|  |
| --- |
| namespace MyTasks.Core.ViewModels  {  public class TasksViewModel  {  public IEnumerable<Task> Tasks { get; set; }  public IEnumerable<Category> Categories { get; set; }  public FilterTasks FilterTasks { get; set; }  }  } |

Tworzymy klasę zawierającą filtry do filtrowania listy zadań.

|  |
| --- |
| using MyTasks.Core.Models;  using MyTasks.Core.Models.Domains;  using System;  using System.Collections.Generic;  using System.Linq;  //using System.Threading.Tasks;  namespace MyTasks.Core.ViewModels  {  public class TasksViewModel  {  public IEnumerable<Task> Tasks { get; set; }  public IEnumerable<Category> Categories { get; set; }  public FilterTasks FilterTasks { get; set; }  }  } |

Do widoku dopisujemy na górze model

|  |
| --- |
| @model MyTasks.Core.ViewModels.TasksViewModel  @{  ViewData["Title"] = "Tasks";  }  <h1>Moje zadania</h1>  @\* formularz filtrowania \*@ |

Formularz z filtrami zostawiamy tak jak jest bez helpera **beginform**

|  |
| --- |
| <form>  <div class="form-row align-items-center">  <div class="col-auto">  <label class="sr-only" for="inlineFormInput">Name</label> |

, ponieważ ten formularz będziemy submit’ować trochę w inny sposób.  
Chcemy to zrobić tak, aby po kliknięci przycisku submit nie przeładowywała się strona, tylko odświeżała tabelka z zadaniami.

|  |
| --- |
| @Html.TextBoxFor(x=>x.FilterTasks.Title, new { @class="form-control mb-2", placeholder="Tytuł"}) |

**Placeholder** to wyszarzony napis wewnątrz textboxa, który znika, gdy zaczniesz cokolwiek wpisywać.  
Zaoszczędza miejsca, bo nie musimy tworzyć dodatkowych labeli.

|  |
| --- |
| <div class="col-auto">  @Html.DropDownListFor(x=>x.FilterTasks.CategoryId,  new SelectList(Model.Categories,"Id","Name"),  " -- Wybierz kategorię --",  new { @class = "form-control mb-2", placeholder = "Tytuł" }  )  </div> |

Wstawiamy dropboxa chcemy aby źródłem danych byłą lista kategorii, aby kluczem było id oraz aby wyświetlana byłą nazwa.

|  |
| --- |
| **Przypomnienie !**  W obu powyższych helperach, słowo **class** musimy poprzedzić znakiem **@**, bo gdybyśmy tego nie zrobili było by one traktowane jako słowo kluczowe c# |

Cały formularz filtrowania wygląda jak poniżej

|  |
| --- |
| @\* formularz filtrowania \*@  <form id="filterForm">  <div class="form-row align-items-center">  <div class="col-auto">  @Html.TextBoxFor(x=>x.FilterTasks.Title,  new { @class="form-control mb-2", placeholder="Tytuł"})  </div>  <div class="col-auto">  @Html.DropDownListFor(x=>x.FilterTasks.CategoryId,  new SelectList(Model.Categories,"Id","Name"),  " -- Wybierz kategorię --",  new { @class = "form-control mb-2"}  )  </div>  <div class="col-auto">  <div class="form-check mb-2">  @Html.CheckBoxFor(x=>x.FilterTasks.IsExecuted,  new { @class = "form-check-label" }  )  @Html.LabelFor(x=>x.FilterTasks.IsExecuted)  </div>  </div>  <div class="col-auto">  <button type="submit" class="btn btn-primary mb-2">Szukaj</button>  </div>  </div>  </form>  @\* tabela z zadaniami \*@  <div>  @await Html.PartialAsync("\_TasksTablePartial.cshtml",Model.Tasks)  </div>  @section Scripts  {  <script type="text/javascript">  </script>  } |

Teraz do zrobienia pozostała tabelka z listą zadań. Chcemy aby odświeżyła się ona po wciśnięciu przycisku submit, najwygodniej będzie, gdy zrobimy tą tabelkę jako widok częściowy.  
Widok wywołujemy asynchronicznie korzystając z helpera **Html.PartialAsync**, poprzedzamy go słowem await. Na koniec wstawiamy sekcję gotową do wstawienia skryptów.

Widok częściowy z tabelką

|  |
| --- |
| @model IEnumerable<MyTasks.Core.Models.Domains.Task>  <div id="tasksTable">  <table class="table mt-3 table-bordered">  <thead class="thead-dark">  <tr>  <th scope="col">Tytuł</th>  <th scope="col">Opis</th>  <th scope="col">Kategoria</th>  <th scope="col">Termin</th>  <th scope="col"></th>  </tr>  </thead>  <tbody>  @if (Model != null && Model.Any())  {  foreach (var task in Model)  {  <tr>  <td>  <a href="@Url.Action("Task","Task",  new { id = task.Id })"  class="font-weight-bold">@task.Title</a>  </td>  <td>@task.Description</td>  <td>@task.Category.Name</td>  <td>@task.Term?.ToString("dd-MM-yyyy")</td>  <td>  @if (!task.IsExecuted)  {  <button class="btn btn-success btn-sm mr-2"  onclick="finishTask('@task.Id',this)">  Zrealizuj  </button>  }  <button class="btn btn-danger btn-sm"  onclick="deleteTask('@task.Id',this)">  Usuń  </button>  </td>  </tr>  }    }  else  {  <tr>  <td colspan="5" class="text-center">  Brak zaplanowanych zadań  </td>  </tr>  }  </tbody>  </table>  </div> |

Skrypty

Obsługa przycisku submit

|  |
| --- |
| // funkcja zostanie wywołana gdy zostanie załadowany już cały widok  $(document).ready(function () {  // najpierw zaznaczamy formularz o id = filterForm  // chcemy aby po kliknięci przycisku zostałą wywołana taka funkcja  // a chcemy, żeby za pomocą ajax'a została wywołana odpowiednia funkcja z kontrolera  // przesyłamy zserializowane dane z tego formularza  // jeżeli zakończone sukcesem, to zostanie zwrócona tabelka z widoku częściowego  $('#filterForm').submit(function () {  $.ajax({  type: "POST",  url: "@Url.Action("Tasks","Tasks")",  data: $(this).serialize(),  success: function (data) {  $('#taskTable').html(data);  },  error: function (data) {  alert(data.message)  },  dataType: "html"  });  // return false konieczne aby nie wykonał się jeszcze raz submit  return false;  });  }); |

Usuwanie zadania

|  |
| --- |
| // usuwanie zadania  function deleteTask(id, btn) {  if (!confirm("Czy na pewno chcesz usunąć zadanie ?")) {  return;  }  $.ajax({  type: "POST",  url: "@Url.Action("Delete","Tasks")",  data: {  id: id  },  success: function (data) {  if (data.success) {  var row = btn.parentNode.parentNode;  row.parentNode.removeChild(row);  }  else {  alert(data.message);  }  },  error: function (data) {  alert(data.message);  },  dataType: "json"  });  } |

Oznaczanie zadania jako zakończone

|  |
| --- |
| // oznaczanie zadania jako zrealizowane  function finishTask(id, btn) {  if (!confirm("Czy na pewno chcesz oznaczyć zadanie jako zrealizowane ?")) {  return;  }  $.ajax({  type: "POST",  url: "@Url.Action("Delete","Tasks")",  data: {  id: id  },  success: function (data) {  if (data.success) {  var row = btn.parentNode.parentNode;  row.parentNode.removeChild(row);  }  else {  alert(data.message);  }  },  error: function (data) {  alert(data.message);  },  dataType: "json"  });  } |

Przygotowanie modelu widoku do widoku dodawania/edycji zadania.

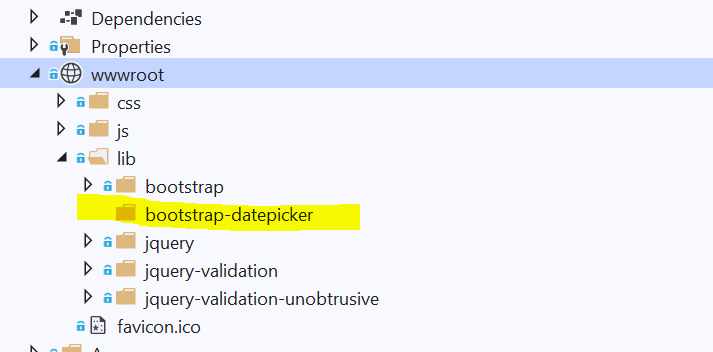
|  |
| --- |
| using MyTasks.Core.Models.Domains;  using System;  using System.Collections.Generic;  using System.Linq;  namespace MyTasks.Core.ViewModels  {  public class TaskViewModel  {  public string Heading { get; set; }  public Task Task { get; set; }  public IEnumerable<Category> Categories { get; set; }  }  } |

Widok do dodawania / edycji zadania

|  |
| --- |
| @model MyTasks.Core.ViewModels.TaskViewModel  @{  ViewData["Title"] = Model.Heading;  }  <h1>@Model.Heading</h1>  @Html.BeginForm("Task","Task")  {  <p class="alert alert-info mt-4 mb-4">  Pola Tytuł, opis i kategoria są <strong>wymagane</strong>  </p>  @Html.ValidationSummary()  @Html.AntiForgeryToken()  @Html.HiddenFor(x=>x.Task.Id)  <div class="form-group">  @Html.LabelFor(x=>x.Task.Title)  @Html.TextBoxFor(x => x.Task.Title, new { @class="form-control", autofocus="autofocus" })  @Html.ValidationMessageFor(x => x.Task.Title)  </div>  <div class="form-group">  @Html.LabelFor(x => x.Task.Description)  @Html.TextBoxFor(x => x.Task.Description, new { @class = "form-control" })  @Html.ValidationMessageFor(x => x.Task.Description)  </div>  <div class="form-group">  @Html.LabelFor(x => x.Task.Title)  @Html.DropDownListFor(x => x.Task.CategoryId,  new SelectList(Model.Categories, "Id", "Name"),  "-- wybierz kategorię --",  new { @class = "form-control" });  @Html.ValidationMessageFor(x => x.Task.Title)  </div>  <div class="form-group">  @Html.LabelFor(x => x.Task.Term)  @Html.TextBoxFor(x => x.Task.Term,"{0:dd-MM-yyyy}"   ,new { @class = "form-control", @id="Term", autocomplete="off" })  @Html.ValidationMessageFor(x => x.Task.Term)  </div>  @if (Model.Task.Id > 0)  {  <div class="form-check">  @Html.LabelFor(x => x.Task.IsExecuted)  @Html.CheckBoxFor(x => x.Task.IsExecuted, new { @class = "form-check-input" })  </div>  }  <button type="submit" class="btn btn-primary btn-lg mt-2">Zapisz</button>  } |

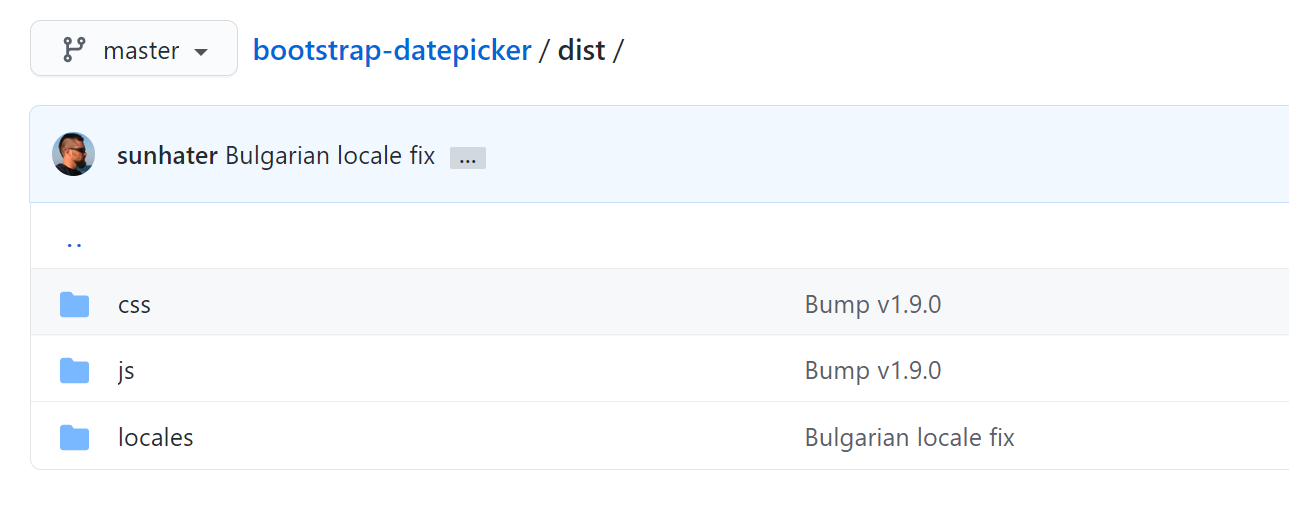
Doinstalowanie klasy bootstrapowej datapicker

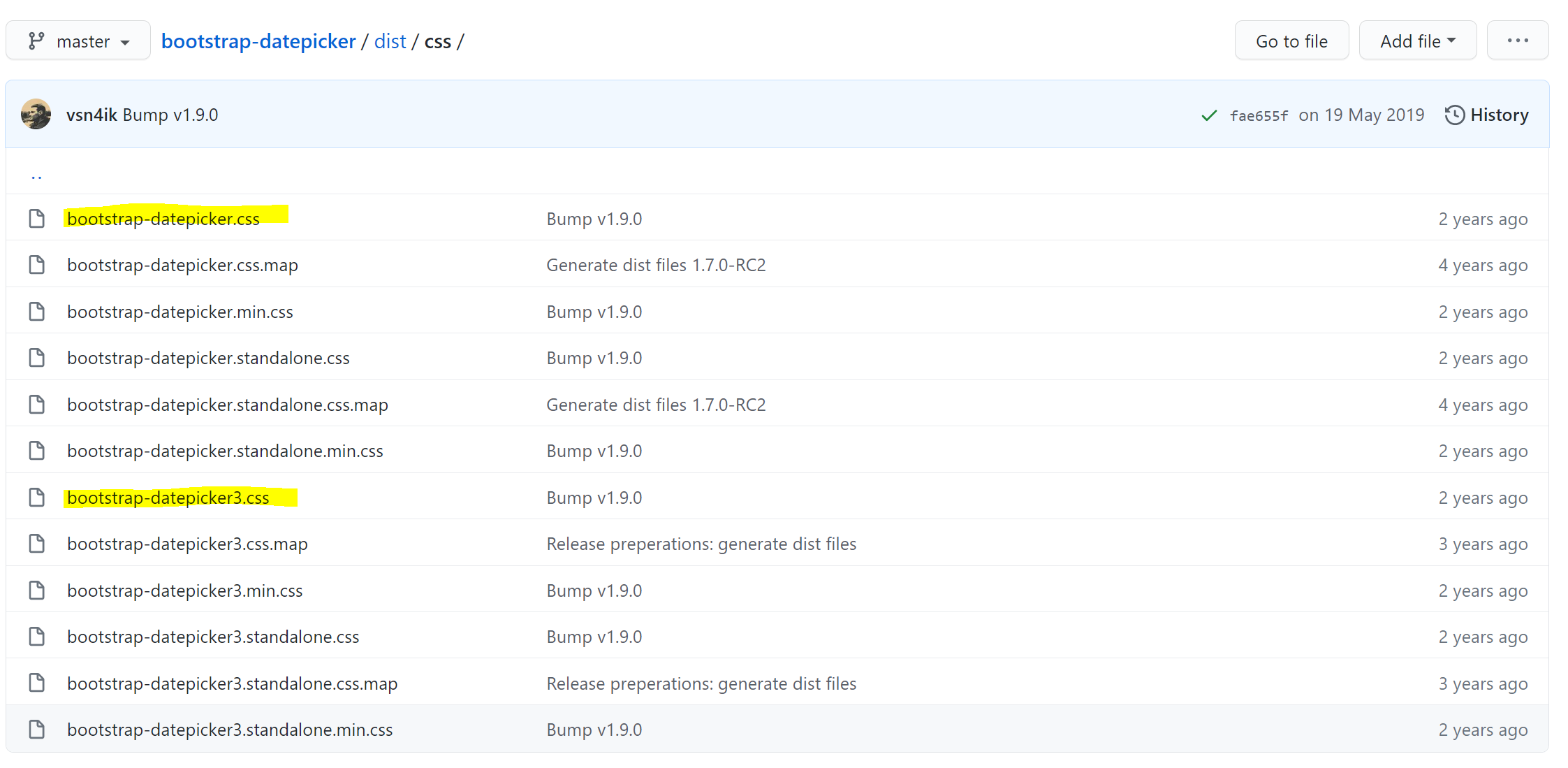
Tworzymy nowy folder w folderze wwwroot.

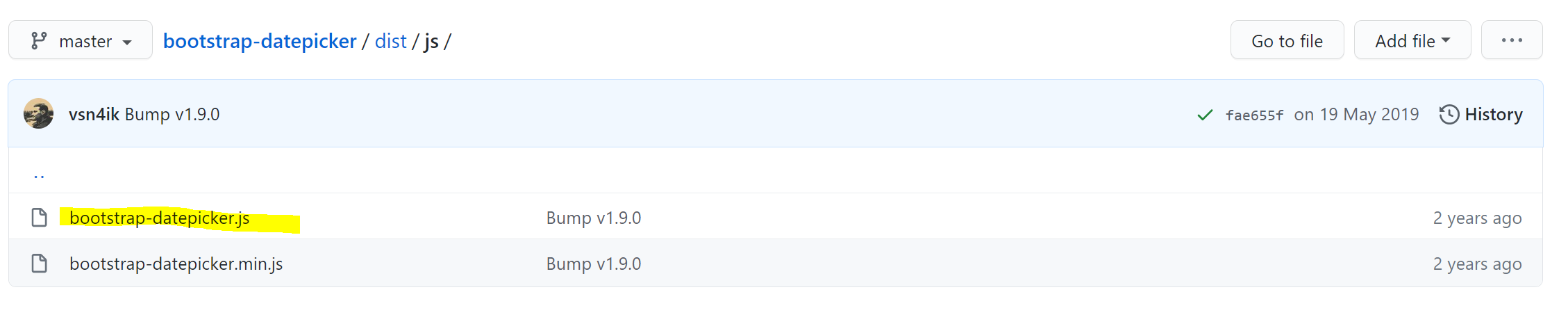


Szukamy w google bootstrap-datepicker

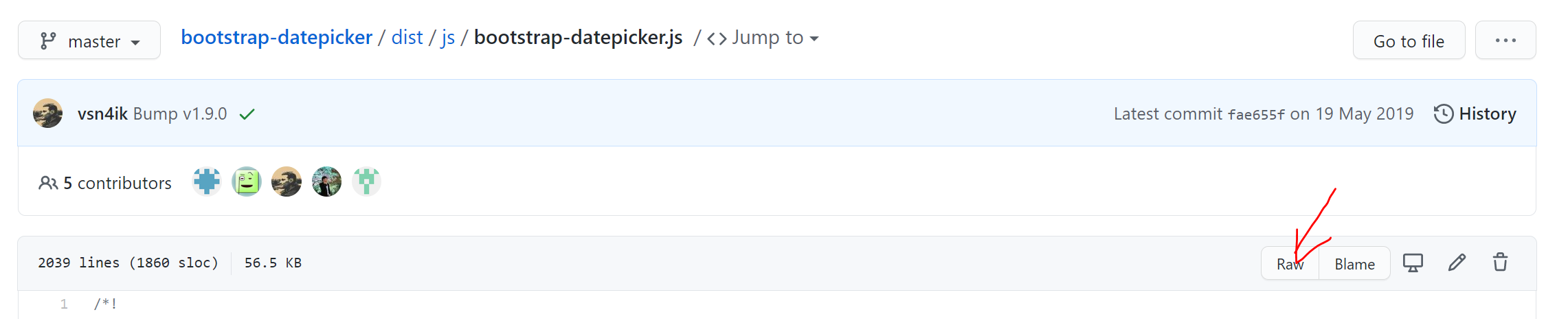
<https://github.com/uxsolutions/bootstrap-datepicker>



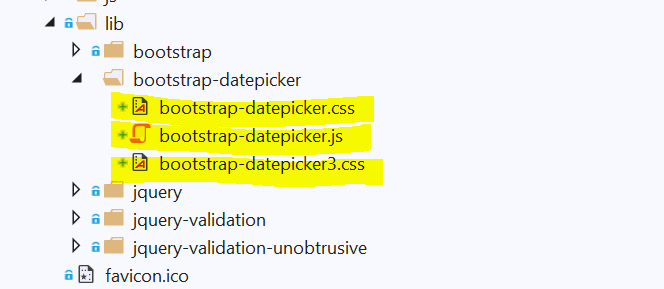




Klikamy na Raw, później na save as



Następnie wgrywamy te pliki



Dodajemy linki do plików css do szablonu \_layout.css  
Możemy techniką drag@drop

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="utf-8" />  <meta name="viewport" content="width=device-width, initial-scale=1.0" />  <title>@ViewData["Title"] - Moje Zadania</title>  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />  <link href="~/lib/bootstrap-datepicker/bootstrap-datepicker.css" rel="stylesheet" />  <link href="~/lib/bootstrap-datepicker/bootstrap-datepicker3.css" rel="stylesheet" />  <link rel="stylesheet" href="~/css/site.css" />  </head> |

Na dole link do skryptu css

|  |
| --- |
| <script src="~/lib/jquery/dist/jquery.min.js"></script>  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>  <script src="~/lib/bootstrap-datepicker/bootstrap-datepicker.js"></script>  <script **src**="~/js/site.js" **asp-append-version**="true"></script>  @await RenderSectionAsync("Scripts", required: false) |

Wracamy jeszcze do widoku Task.cshtml

Kopiujemy skrypt ten sam, co został użyty w lekcji ASP.NET w zeszłym tygodniu

|  |
| --- |
| <script type="text/javascript">  $(document).ready(function () {  // w formatowaniu ma być precinek, a nie kropka  $.validator.methods.number = function (value, element) {  return this.optional(element) || /^-?(?:\d+|\{1,3}(?:[\s\.,]\d{3})+)(?:[\.,]\d+)?$/.test(value);  }  // format daty  $.validator.methods.date = function (value, element) {  var isChrome = /Chrome/.test(navigator.userAgent) && /Google Inc/.test(navigator.vendor);  var isFirefox = /Firefox/.test(navigator.userAgent);  if (isChrome || isFirefox) {  var d = new Date();  return this.optional(element) || !/Invalid|NaN/.test(new Date(d.toLocaleDateString(value)));  } else {  return this.optional(element) || !/Invalid|NaN/.test(new Date(value));  }  }  $('#Term').datepicker({  format: "dd-mm-yyyy",  language: "pl",  multidate: false,  autoclose: true,  todayHighlight: true,  forceParse: false  });  });  </script>  } |

Mamy gotowe widoki, kolejny krok to przygotowanie odpowiednich akcji w kontrolerze.