Mariusz Wieczorek

mariusz.wieczorek@kabat.pl

Streszczenie

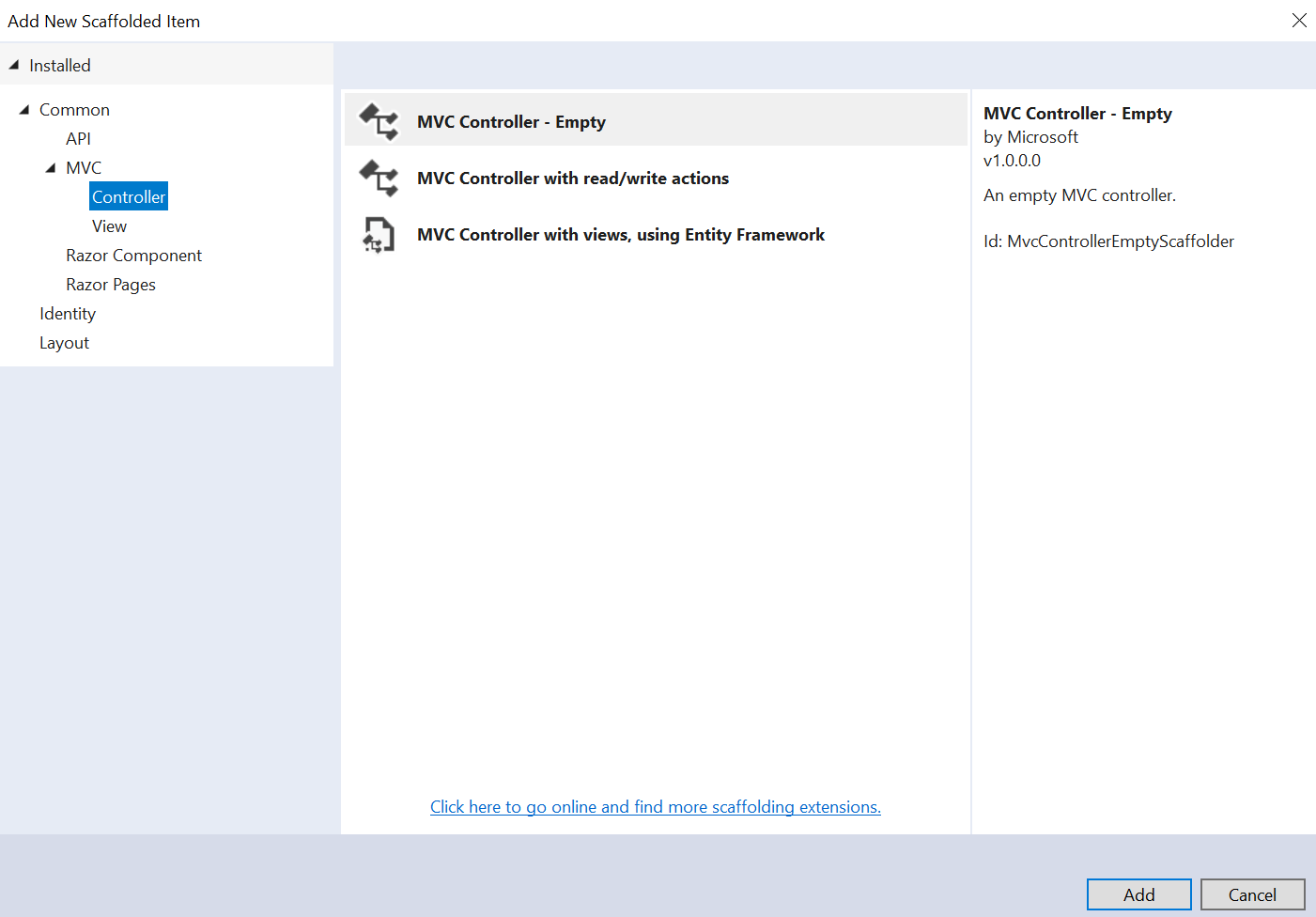
Tworzymy kontroler  
Wykorzystanie metod rozszerzających do pobrania identyfikaora użytkownika

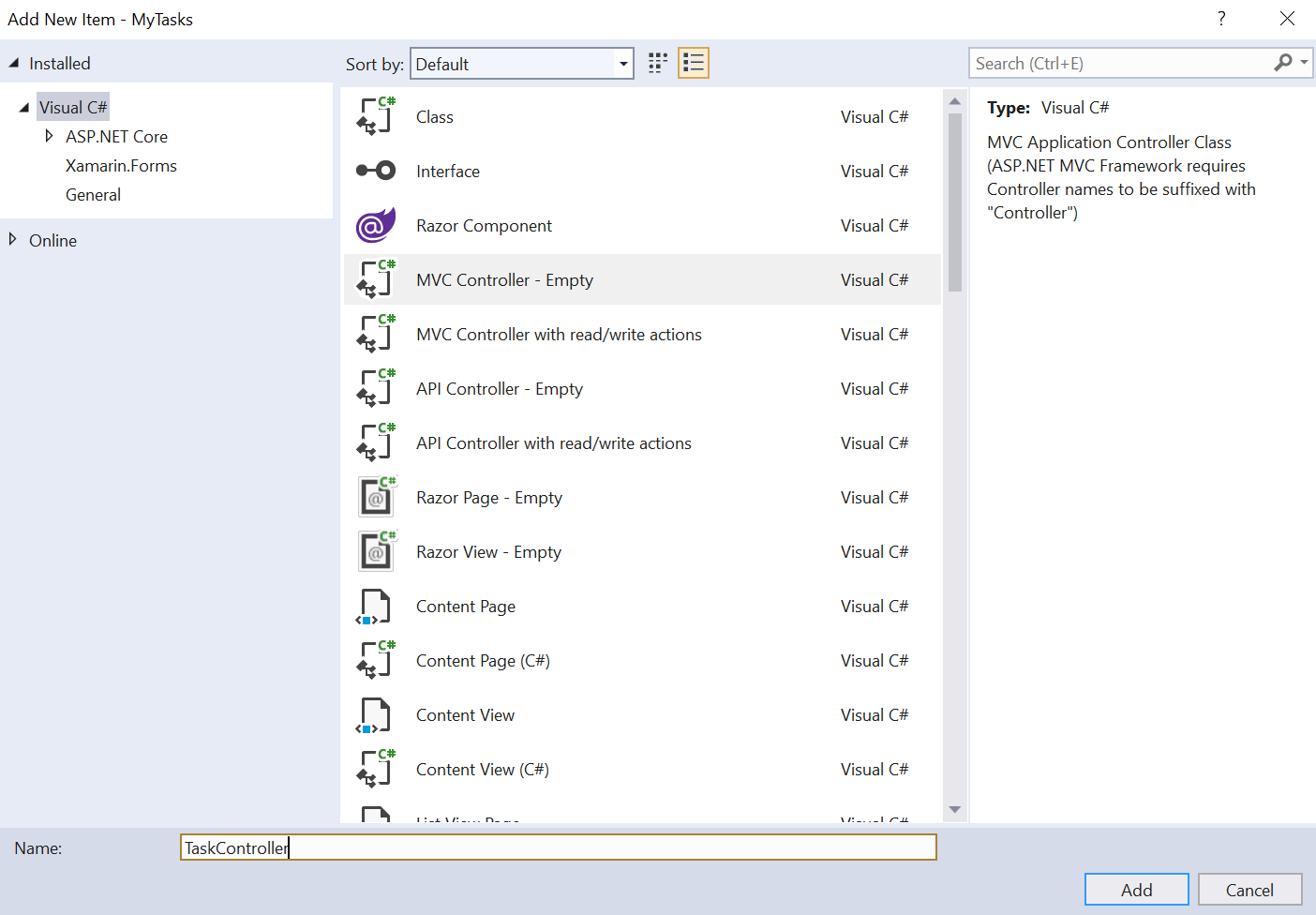
asp.net Core

Tworzenie Kontrolerów

**Tworzymy Kontroler**

Dodajemy nowy kontroler





Na początek oznaczamy go atrybutem Authorize

|  |
| --- |
| namespace MyTasks.Controllers  {  [Authorize]  public class TaskController : Controller  {  public IActionResult Index()  {  return View();  }  }  } |

Wprowadzamy zmiany w routingu w pliku Startup.cs

|  |
| --- |
| app.UseEndpoints(endpoints =>  {  endpoints.MapControllerRoute(  name: "default",  pattern: "{controller=Home}/{action=Index}/{id?}");  endpoints.MapRazorPages();  }); |

|  |
| --- |
| app.UseEndpoints(endpoints =>  {  endpoints.MapControllerRoute(  name: "default",  pattern: "{controller=Task}/{action=Tasks}/{id?}");  endpoints.MapRazorPages();  }); |

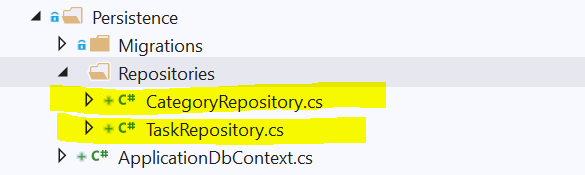
Potem zmieniamy w \_Layout.cshtml

|  |
| --- |
| <div class="container">  <a class="navbar-brand" **asp-area**="" **asp-controller**="Task" **asp-action**="Tasks">Moje zadania</a>  <button class="navbar-toggler" type="button" data-toggle="collapse"  data-target=".navbar-collapse" aria-controls="navbarSupportedContent"  aria-expanded="false" aria-label="Toggle navigation">  <span class="navbar-toggler-icon"></span>  </button>  <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">  <ul class="navbar-nav flex-grow-1">  <li class="nav-item">  <a class="nav-link text-dark" **asp-area**=""  **asp-controller**="Task" **asp-action**="Task">Dodaj Nowe</a>  </li>  <li class="nav-item">  <a class="nav-link text-dark" **asp-area**=""   **asp-controller**="Home" **asp-action**="Privacy">Polityka prywatności</a>  </li>  </ul>  <**partial** **name**="\_LoginPartial" />  </div> |

Wracamy do kontrolera

|  |
| --- |
| public class TaskController : Controller  {  private TaskRepository \_taskRepository = new TaskRepository();  private CategoryRepository \_categoryRepository = new CategoryRepository();  public IActionResult Tasks()  {  var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);  var vm = new TasksViewModel()  {  FilterTasks = new FilterTasks(),  Tasks = \_taskRepository.Get(userId),  Categories = \_categoryRepository.GetCategories()  };  return View(vm);  }  } |

Pobieramy zalogowanego użytkownika  
Pobieramy zadania tego użytkownika z repozytorium.  
W tym celu tworzymy repozytoria.



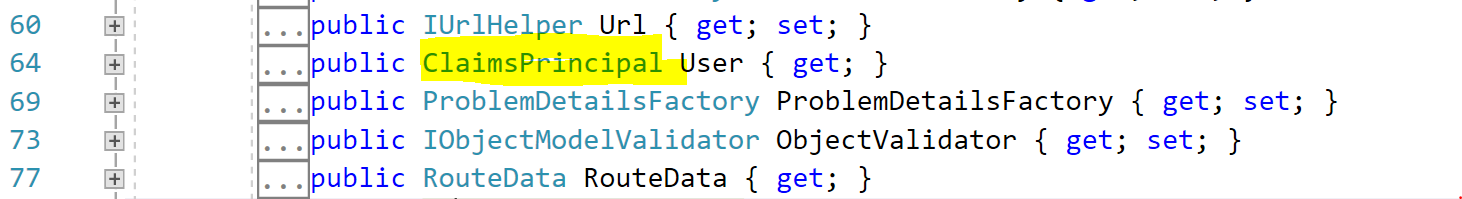
|  |
| --- |
| namespace MyTasks.Persistence.Repositories  {  public class CategoryRepository  {  public IEnumerable<Category> GetCategories()  {  throw new NotImplementedException();  }  }  } |

|  |
| --- |
| namespace MyTasks.Persistence.Repositories  {  public class TaskRepository  {  public IEnumerable<Task> Get(string userId)  {  throw new NotImplementedException();  }  }  } |

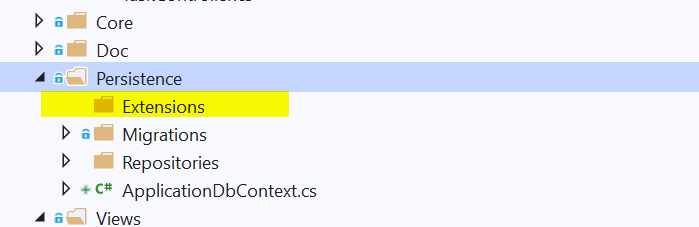
Niezbyt fajnie wygląda też pobieranie użytkownika,

|  |
| --- |
| var userId = User.FindFirstValue(ClaimTypes.NameIdentifier); |

dlatego też napiszemy metodę rozszerzającą dla klasy użytkownika **ClaimsPrincipal**.   
Aby zobaczyć jaka to klasa na słowie User w powyższej linii kodu wciskamy F12.



Tworzymy nowy folder Extensions w folderze Persistence



A w nim klasę **ClaimsPrincipalExtensions**  
Dla przypomnienia musi to być klasa statyczna, metoda również musi być statyczna.  
W argumentach wpisujemy słowo kluczowe this i nazwę typu, który rozszerzamy, czyli   
ClaimsPrincipal a następnie nazwę zmiennej, zwyczajowo model.

|  |
| --- |
| namespace MyTasks.Persistence.Extensions  {  public static class ClaimsPrincipalExtensions  {  public static string GetUserId(this ClaimsPrincipal model)  {  return model.FindFirstValue(ClaimTypes.NameIdentifier);  }  }  } |

Dzięki utworzeniu metody rozszerzającej możemy w kontrolerze zamiast

|  |
| --- |
| var userId = User.FindFirstValue(ClaimTypes.NameIdentifier); |

Użyć dużo bardziej czytelnego zapisu

|  |
| --- |
| using MyTasks.Persistence.Extensions;  var userId = User.GetUserId(); |

Teraz piszemy nową akcję, która będzie wywoływana w widoku **Tasks** po kliknięciu przycisku **submit** szukaj. Funkcja ta wywoła **Ajax’a**, który wywołuje akcję **Tasks** typu **Post** w kontrolerze **Task**. Z Ajax’a zostanie zwrócony html,

|  |
| --- |
| $('#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;  }); |

który zostanie wyświetlony w miejscu wyświetlenia widoku częściowego.

|  |
| --- |
| @\* tabela z zadaniami \*@  <div>  @await Html.PartialAsync("\_TasksTablePartial.cshtml",Model.Tasks)  </div> |

Kod kontrolera wygląda następująco

|  |
| --- |
| // akcja wywoływana w widoku Tasks po kliknięciu na submit  [HttpPost]  public IActionResult Tasks(TasksViewModel viewModel)  {  var userId = User.GetUserId();  var tasks = \_taskRepository.Get(userId,  viewModel.FilterTasks.IsExecuted,  viewModel.FilterTasks.CategoryId,  viewModel.FilterTasks.Title);    return PartialView("\_TasksTable",tasks);  } |

przerabiamy też w repozytorium metodę Get, aby uwzględniała przekazane filtry

|  |
| --- |
| public class TaskRepository  {  public IEnumerable<Task> Get(string userId,  bool isExecuted = false,  int categoryId = 0,  string title = null)  {  throw new NotImplementedException();  }  } |

13:22

|  |
| --- |
| public IActionResult Tasks()  {  var userId = User.GetUserId();  var vm = new TasksViewModel()  {  FilterTasks = new FilterTasks(),  Tasks = \_taskRepository.Get(userId),  Categories = \_categoryRepository.GetCategories()  };  return View(vm);  }    // akcja wywoływana w widoku Tasks po kliknięciu na submit służącym do filtrowania zadań  [HttpPost]  public IActionResult Tasks(TasksViewModel viewModel)  {  var userId = User.GetUserId();  var tasks = \_taskRepository.Get(userId,  viewModel.FilterTasks.IsExecuted,  viewModel.FilterTasks.CategoryId,  viewModel.FilterTasks.Title);    return PartialView("\_TasksTable",tasks);  } |

|  |
| --- |
| #region Task: edycja/dodawanie/usuwanie zadania -----------------------  // Wejście na ekran dodawania/edycji zadania z przycisku dodaj nowe zadanie  // lub po kliknięciu na link na liście zadań służący do edycji zadania    [HttpGet]  public IActionResult Task(int id = 0)  {  var userId = User.GetUserId();  var task = id == 0 ?  new Task { Id = 0, UserId = userId, Term = DateTime.Now } :  \_taskRepository.Get(id, userId);  var vm = new TaskViewModel()  {  Task = task,  Categories = \_categoryRepository.GetCategories(),  Heading = id == 0 ?  "nowe zadanie" :  "edycja zadania"  };  return View(vm);  } |
| // Wciśnięcie przycisku typu submit na ekranie edycji/dodania zadania  // Po którym nastąpi przekazanie zawartości pól na formularzu  // i wywołanie metod w repozytorium dodania lub aktualizacji zadania  [HttpPost]  [ValidateAntiForgeryToken]  public IActionResult Task(Task task)  {  var userId = User.GetUserId();  task.UserId = userId;  if (!ModelState.IsValid)  {  var vm = new TaskViewModel()  {  Task = task,  Categories = \_categoryRepository.GetCategories(),  Heading = task.Id == 0 ?  "nowe zadanie" :  "edycja zadania"  };    return View("Task", vm);  }  if (task.Id == 0)  \_taskRepository.Add(task);  else  \_taskRepository.Update(task);  return RedirectToAction("Tasks", "Task");  } |

|  |
| --- |
| // po kliknięciu przycisku deleteTask  // zostanie wywołany ajax  // który wywoła akcję Delete typu Post w kontrolerze Task  // i to jest ta akcja  public IActionResult Delete(int id)  {  try  {  var userId = User.GetUserId();  \_taskRepository.Delete(id, userId);  }  catch (Exception ex)  {  // logowanie do pliku  return Json(new { success = false, message = ex.Message });  }  return Json(new { success = true });  } |
| // po kliknięciu przycisku finishTask  // zostanie wywołany ajax  // który wywoła akcję Delete typu Post w kontrolerze Task  // i to jest ta akcja  public IActionResult Finish(int id)  {  try  {  var userId = User.GetUserId();  \_taskRepository.Finish(id, userId);  }  catch (Exception ex)  {  // logowanie do pliku  return Json(new { success = false, message = ex.Message });  }  return Json(new { success = true });  }  #endregion |

W kolejnej lekcji uzupełnienie metod w repozytorium