Mariusz Wieczorek

[Nazwa firmy] | [Adres firmy]

Kontrolery

asp.net mvc

**Kontrolery**

Użytkownik wpisuje adres w przeglądarce  
Przeglądarka wysyła żądanie do serwera  
Dzięki routingowi wywoływany jest odpowiedni kontroler, który wykonuje jakąś akcję  
i zwraca jej wynik.

Kontroler nie zawsze musi zwracać widok.

|  |
| --- |
| public class HomeController : Controller  {  public **ActionResult** Test()  {  return **View**("About");  } |

ActionResult jest klasą abstrakcyjną, która ma konstruktor i metodę ExecuteResult()

|  |
| --- |
| namespace System.Web.Mvc  {  public abstract class ActionResult  {  protected ActionResult();  public abstract void ExecuteResult(ControllerContext context);  }  } |

Z kolei View() jest metodą, która zwraca **ViewResult**

|  |
| --- |
| protected internal ViewResult View(string viewName); |

ViewResult jest klasą, która dziedziczy po ViewResultBase

|  |
| --- |
| public class ViewResult : ViewResultBase |

A ten z kolei po ActionResult

|  |
| --- |
| public abstract class ViewResultBase : ActionResult |

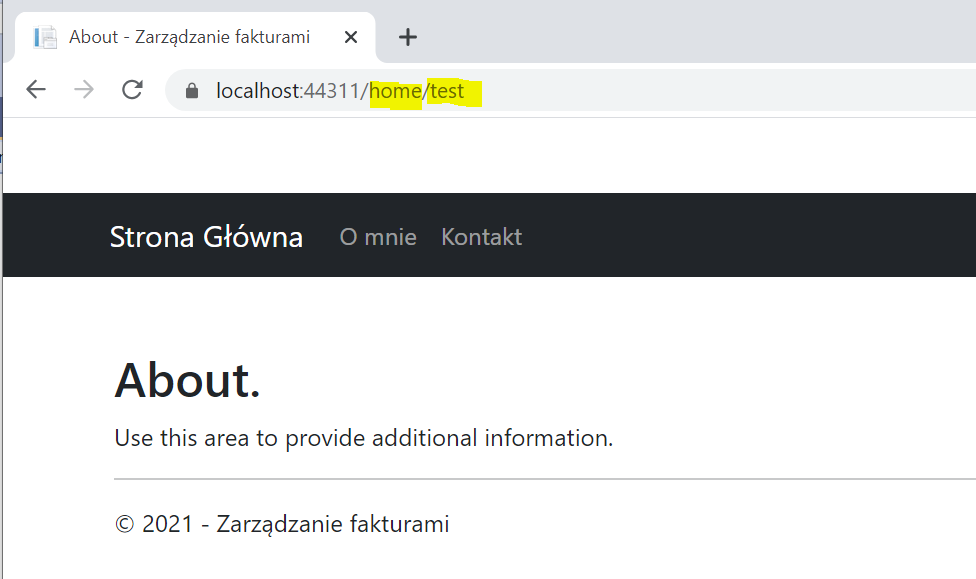
Czyli jeżeli oczekujemy , że zostanie zwrócony ActionResult, to zostanie zwrócony widok, który dziedziczy tą klasę, chociaż nie bezpośrednio.  
Możemy zwrócić obiekty klas: ViewResult, PartialViewResult, RedirectResult, RedirectToActionResult,  
RedirectToRoute, JsonResult, FileResult oraz ContentResult

|  |
| --- |
| public abstract class Controller : ControllerBase, IActionFilter, IAuthenticationFilter, IAuthorizationFilter, IDisposable, IExceptionFilter, IResultFilter, IAsyncController, IController, IAsyncManagerContainer  {  protected Controller();  protected internal PartialViewResult PartialView();  protected internal ViewResult View(string viewName, object model);  protected internal ViewResult View(object model);  protected internal ViewResult View(string viewName);  protected internal virtual ViewResult View(string viewName,string masterName, object model); |

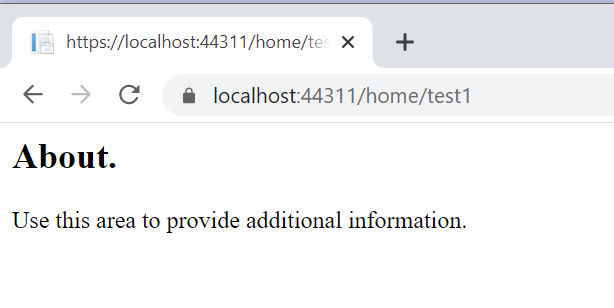
**Zwracanie różnych typów wartości z akcji kontrolera.**

Jeżeli wpiszemy w przeglądarce localhost / Home / Test  
To zostanie odszukany kontroler HomeController a w nim akcja Test

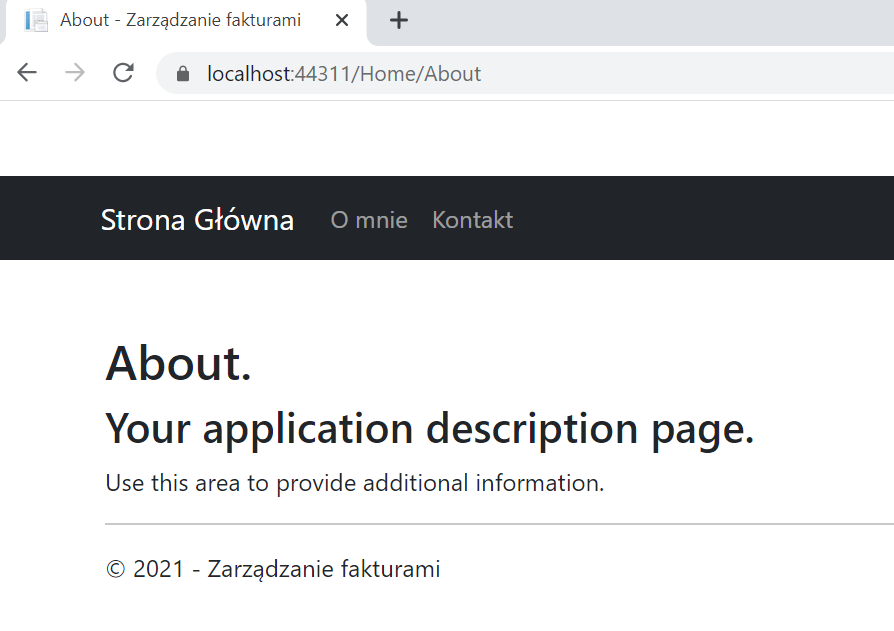
|  |
| --- |
| public class HomeController : Controller  {  public ActionResult Test()  {  return View("About");  } |



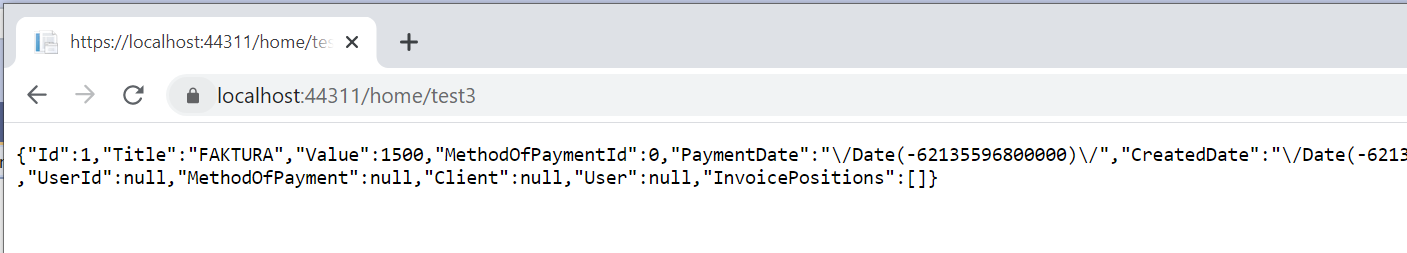
|  |
| --- |
| public ActionResult Test1()  {  return PartialView("About");  } |

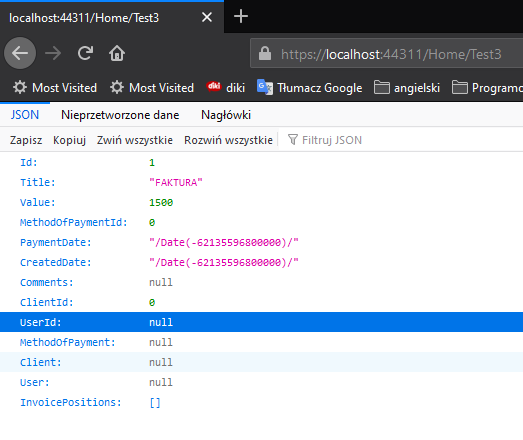


|  |
| --- |
| public ActionResult Test2()  {  return RedirectToAction("About","Home");  } |

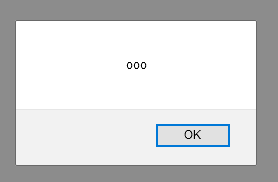


|  |
| --- |
| public ActionResult Test3()  {  var js = new Invoice { Id = 1, Title = "FAKTURA", Value = 1500 };  return Json("About", JsonRequestBehavior.AllowGet);  } |





|  |
| --- |
| public ActionResult Test5()  {  return Content("<script>alert('ooo')</script>");  } |



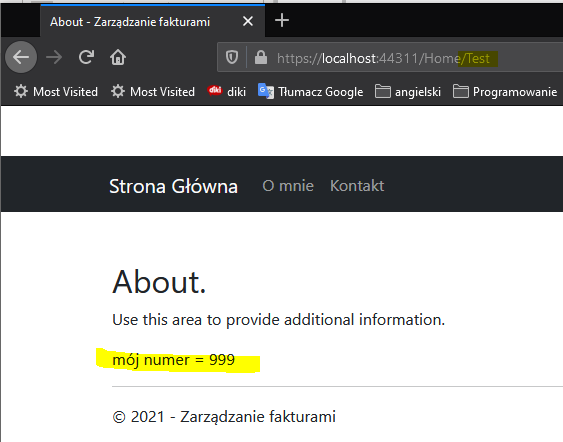
**Przekazywanie wartości z kontrolera do widoku.  
ViewBag**

Możemy to zrobić przez **ViewBag**. Możemy w ten sposób przekazać dowolną zmienną.  
Np. poniżej tworzymy zmienną MyNumber, którą zostanie poprzez obiekt ViewBag  
przekazana do widoku About.

|  |
| --- |
| public ActionResult Test()  {  ViewBag.MyNumber = 999;  return View("About");  } |

W widoku odwołujemy się do tej zmiennej również przez obiekt ViewBag.

|  |
| --- |
| @{  ViewBag.Title = "About";  }  <h2>@ViewBag.Title.</h2>  <h3>@ViewBag.Message</h3>  <p>Use this area to provide additional information.</p>  mój numer = @ViewBag.MyNumber |



ViewBag stosujemy, gdy mamy do przekazania pojedynczą zmienną i nie ma sensu robić specjalnie modelu, czy też modelu widoku.

Należy zwrócić uwagę, aby w kontrolerach nie umieszczać zbyt dużo logiki.  
Nie umieszczamy tu żadnych algorytmów i obliczeń.  
Kontrolery powinny tylko przechwycić parametry, ewentualnie dokonać walidacji i zwrócić widok.

Domyślnie kontroler obsługuje żądania typu GET, ale możemy tworzyć też kontrolery obsługujące inne żądania np. POST.

Budujemy Aplikację cd..

Usuwamy niepotrzebny kod

|  |
| --- |
| 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);  } |

Pobieramy Id użytkownika.  
Odwołujemy się do metody GetInvoices, z repozytorium, które zaraz utworzymy, jako parametr podajemy Id użytkownika aby odfiltrować tylko faktury należące do niego.

|  |
| --- |
| public ActionResult Index()  {  var userId = User.Identity.GetUserId();  var invoices = \_invoicesRepository.GetInvoices(userId);  return View(invoices);  } |

Tworzymy w Modelu nowy folder Repositories

|  |
| --- |
| public class InvoiceRepository  {  public List<Invoice> GetInvoices(string userId)  {  // TODO: Metoda Odczytująca Faktury z Bazy  return new List<Invoice>();  }  } |

Akcja Invoice

|  |
| --- |
| // nagłówki faktur  public ActionResult Invoice(int id = 0)  {  var userId = User.Identity.GetUserId();  var invoice = id == 0 ?  GetNewInvoice(userId) :  \_invoiceRepository.GetInvoice(id, userId);  var vm = PrepareInvoiceVm(invoice, userId);  return View(vm);  } |

Podobnie jak w Index() pobieramy id użytkownika, następnie fakturę jeżeli  
przekazane id jest równe 0 to tworzymy nową fakturę,  
do tego celu tworzymy nową metodę w kontrolerze GetNewInvoice()

|  |
| --- |
| private Invoice GetNewInvoice(string userId)  {  return new Invoice  {  UserId = userId,  CreatedDate = DateTime.Now,  PaymentDate = DateTime.Now.AddDays(7)  };  } |

Jeżeli przekazane id nie jest zerem, to wywołujemy metodę z repozytorium GetInvoice()

|  |
| --- |
| public Invoice GetInvoice(int id, string userId)  {  // TODO: Metoda Odczytująca Pojedynczą Fakturę z Bazy  return new Invoice();  } |

Następnie przygotowujemy model widoku, który przekażemy do widoku. Ponieważ ten kod się nam jeszcze przyda, to wyodrębniamy go do nowej metody PrepareInvoiceVm(), która zwróci nam odpowiedni ViewModel.

|  |
| --- |
| private **EditInvoiceViewModel** PrepareInvoiceVm(Invoice invoice, string userId)  {  return new **EditInvoiceViewModel**  {  Invoice = invoice,  Heading = invoice.Id == 0 ? "Dodawanie Nowej Faktury" : "Edycja",  Clients = \_clientRepository.GetClients(userId),  MethodOfPayments = \_invoiceRepository.GetMethodOfPayment()  };  } |

InvoicePosition() - akcja, która wyświetla nam pozycję faktury, którą dodajemy lub edytujemy

|  |
| --- |
| // akcja, która wyświetla nam pozycję faktury, którą dodajemy lub edytujemy  public ActionResult **InvoicePosition**(int invoiceId = 0, int invoicePositionId = 0)  {  var userId = User.Identity.GetUserId();  var invoicePosition = invoicePositionId == 0 ?  GetNewInvoicePosition(invoiceId, invoicePositionId) :  \_invoiceRepository.GetInvoicePosition(invoicePositionId,userId);  var vm = PrepareInvoicePositionVm(invoicePosition);  return View(vm);  } |

Postępujemy podobnie jak w przypadku Akcji Invoice()  
Tworzymy repozytorium, metodę zwracającą model widoku itd. …

Akcje [HttpPost]

Jeżeli na jakimś widoku np. Invoice.cshtml wypełnimy dane formularza i wciśniemy na przycisk Zapisz

|  |
| --- |
| <div class="row">  <div class="col-lg-6">  @using (Html.BeginForm("Invoice", "Home"))  {  @Html.HiddenFor(x => x.Invoice.Id)  @Html.HiddenFor(x => x.Invoice.ClientId)  @Html.HiddenFor(x => x.Invoice.CreatedDate)  <div class="form-group mb-3">  @Html.LabelFor(x => x.Invoice.Title)  @Html.TextBoxFor(x => x.Invoice.Title,  new { @class = "form-control", autofocus = "autofocus" })  </div> |
|  |
| <button class="btn btn-primary btn-lg mb-5" type="submit">  Zapisz  </button>  }  </div> |

To zostanie wywołana akcja **Invoice** w kontrolerze Home, z tym że to będzie akcja typu **HttpPost**.  
Akcję taką musimy oznaczyć atrybutem **[HttpPost]**.  
Także możemy mieć kilka akcji o tej samej nazwie. Akcja bez żadnego atrybutu określającego typ żądania będzie akcją typu **HttpGet**.

|  |
| --- |
| public ActionResult **Invoice**(int id = 0)  {  var userId = User.Identity.GetUserId();  var invoice = id == 0 ?  GetNewInvoice(userId) :  \_invoiceRepository.GetInvoice(id, userId);  var vm = PrepareInvoiceVm(invoice, userId);  return View(vm);  }    [HttpPost]  public ActionResult **Invoice**()  {  return RedirectToAction("Index");  } |

W Akcji tej dodamy nową fakturę jeżeli będzie to dodawanie, lub dokonamy aktualizacji jeżeli będzie to edycja. Możemy to zrobić na kilka sposobów. Możemy zwrócić fakturę, możemy też po zapisie danych przekierować do innej akcji o nazwie index.

W formularzu

|  |
| --- |
| @Html.HiddenFor(x => x.Invoice.Id)  @Html.HiddenFor(x => x.Invoice.UserId)  @Html.HiddenFor(x => x.Invoice.CreatedDate) |

UserId jest przetrzymywane w polu HiddenFor, co nie jest dobrym rozwiązaniem, ktoś może zmanipulować dane.   
Lepiej w akcji pobrać na nowo zalogowanego użytkownika i operować tylko na tej zmiennej.  
Datę utworzenia tęż wyrzucamy, ponieważ jak ją utworzymy to już nie będziemy edytować jej daty utworzenia. Zostawimy tylko InvoiceId abyśmy wiedzieli, czy jest to edycja , czy dodawanie nowej faktury. Aby aplikacja była bezpieczna ID użytkownika pobieramy w środku w Akcji.

|  |
| --- |
| [HttpPost]  public ActionResult Invoice(Invoice invoice)  {  var userId = User.Identity.GetUserId();  invoice.UserId = userId;  if (invoice.Id == 0)  \_invoiceRepository.Add(invoice);  else  \_invoiceRepository.Update(invoice);  return RedirectToAction("Index");  } |

Podobnie w InvoicePossition

|  |
| --- |
| [HttpPost]  public ActionResult InvoicePosition(InvoicePossition invoicePosition)  {  var userId = User.Identity.GetUserId();  // zweryfikujemy, czy użytkownik próbuje zaktualizować swoją fakturę  // dodajemy lub aktualizujemy pozycję faktury  // wyliczymy wartość pozycji  // wyliczymy i zaktualizujemy wrtość faktury  var product = \_productRepository.GetProduct(invoicePosition.ProductId);  invoicePosition.Value = product.Value \* invoicePosition.Quantity;  if (invoicePosition.Id == 0)  \_invoiceRepository.AddPosition(invoicePosition, userId);  else  \_invoiceRepository.UpdatePosition(invoicePosition, userId);  \_invoiceRepository.UpdateInvoiceValue(invoicePosition.InvoiceId);  return RedirectToAction("Invoice", new { id = invoicePosition.Id });  } |

Do dodania zostały Akcje HttpPost obsługujące usuwanie faktur i pozycji z tych faktur

|  |
| --- |
| public ActionResult DeleteInvoice(int invoiceId)  {  try  {  var userId = User.Identity.GetUserId();  \_invoiceRepository.Delete(invoiceId, userId);  }  catch (Exception exception)  {  // TODO: logowanie do pliku niepowodzenie usunięcia faktury  return Json(new { Success = false, Message = exception.Message });  }    return Json( new { Success = true});  } |

W tym przypadku zwracamy obiekt Json, ponieważ akcja ta będzie wywoływana przez AJAX