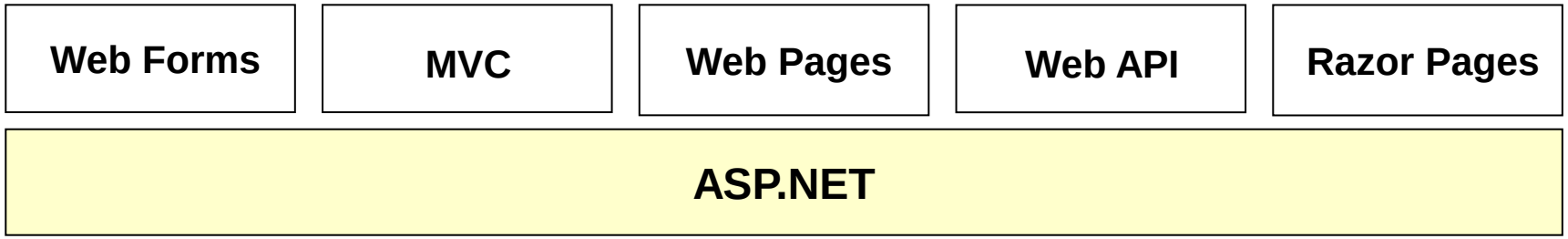
ASP.NET

dr hab. inż. Marek Wojciechowski

ASP.NET

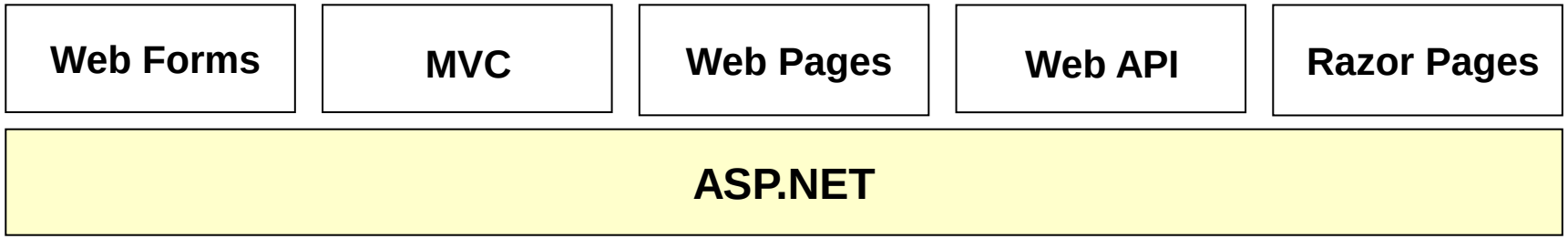
- Technology (also referred to as a web framework) for building modern web applications and services on the .NET platform
  - successor of Active Server Pages (ASP)
  - works on CLR, and thus uses compiled .NET programming languages for application logic
- Supports multiple programming models
- Applications run on application servers, mainly:
  - IIS (Internet Information Services): only for Windows, classic application server from Microsoft, integral part of server versions of Windows
  - Kestrel: multi-platform, lighter than IIS, supports a single application only

ASP.NET programming models (1/2)



- Web Forms (outdated, legacy)
  - Component-based UI ("UI controls")
  - Visual and event-driven programming
  - Classic ASP.NET syntax (<% %>)
- ASP.NET MVC
  - Implementation of the MVC architectural pattern
  - Classic ASP.NET (at first) or (later) Razor syntax for views
- Web Pages
  - Simple old-style web applications as in pure PHP or old ASP
  - Razor syntax

ASP.NET programming models (2/2)



- Web API
  - API for the HTTP protocol
  - REST-style services but not only
  - JSON as the default data exchange format, support for XML
- Razor Pages
  - Page-centric application model
  - Abstraction (simplification) over ASP.NET MVC
  - Concept similar to the Model-View-ViewModel (MVVM) pattern
    - Each page has its own Page Model and bindings to it

ASP.NET Web Forms

- Framework to bulid web application in a way resembling building desktop applications
  - "heart and soul of ASP.NET"
  - "stateful framework over a stateless medium"
- Lost popularity in favor of ASP.NET MVC
- Not available in .NET Core
  - Razor Pages positioned as a successor of Web Forms
- Web Forms pages consist of 2 components:
  - visual page (*.aspx file)
  - "code behind" file (partial class)
- Characteristic terms:
  - postback
  - viewstate
  - server controls

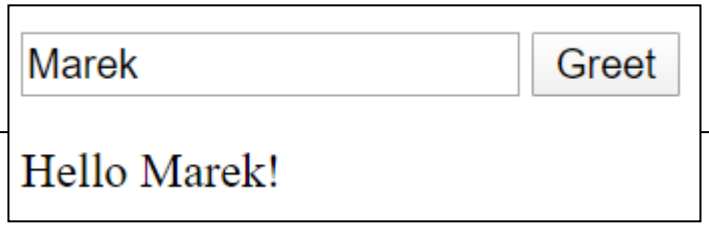
Strengths of Web Forms

- Separation of HTML from application logic (code behind)
- Rich set of server controls
  - Including controls responsible for HTML generation
- Less coding
  - e.g., thanks to functional data access controls
- Event-driven programming familiar to desktop application developers
  - In particular Visual Basic for Windows
- Access to Ajax functionality without coding in JavaScript

ASP.NET Web Forms: Example (1/2)

Hello.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
            CodeFile="Hello.aspx.cs" Inherits="Hello" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server"><title></title></head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:TextBox ID="NameTextBox" runat="server"></asp:TextBox>
      <asp:Button ID="SubmitButton" runat="server" Text="Greet"
        onclick="SubmitButton_Click" /> <br />
      <asp:Label ID="HelloLabel" runat="server" Text=""></asp:Label>
   </div>
   </form>
</body>
</html>
```



ASP.NET Web Forms: Example (2/2)

Hello.aspx.cs

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Hello : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)  {}

    protected void SubmitButton_Click(object sender, EventArgs e)
    {
        HelloLabel.Text = "Hello " + NameTextBox.Text + "!";
    }
}
```

Data access controls in ASP.NET

- Data source controls, e.g., SqlDataSource
  - But also ObjectDataSource, and later LinqDataSource and EntityDataSource, facilitating separation of Data Access Layer (DAL)
- Visual data controls cooperating with data source controls, e.g., GridView

LastName	Salary
Smith	3450
Brown	4500
White	2700
Johnson	3560
Thomas	2700

```
<asp:GridView ID="GridView1" DataSourceID="ds" runat="server"/>
<asp:SqlDataSource ID="ds" runat="server"
  SelectCommand="SELECT [LastName], [Salary]
                 FROM [Employees]"
  ConnectionString="<%$ ConnectionStrings:Emps %>" />
```

Web.config

```
<connectionStrings>
  <add name="Emps" connectionString="Server=(local);Integrated Security=True;
  Initial Catalog=tempdb" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Disadvantages of Web Forms (from today's perspective)

- Abstraction from HTML and HTTP is no longer an advantage
- Costly mechanisms: view state, postbacks, controls, page life cycle
- Makes it easier for programmers to get started quickly, but it is difficult to "properly" implement large systems with it
- Problematic page-centric approach
  - the close relationship between the page file name and the URL
  - difficult and inefficient testing of applications
  - Page Controller "pattern"
  - but... the concept has come back to life in the new Razor Pages framework!

ASP.NET Web Pages

- The simplest programming model of ASP.NET
  - Framework for creating dynamic websites in the style of PHP and classic ASP
  - Uses Razor syntax and engine (ASP.NET Web Pages 2)
  - Open source
  - Created in Visual Studio or the free WebMatrix environment
  - Based on Razor view engine
    - supported programming languages: C# and Visual Basic (.cshtml and .vbhtml page file extensions)

Razor view engine

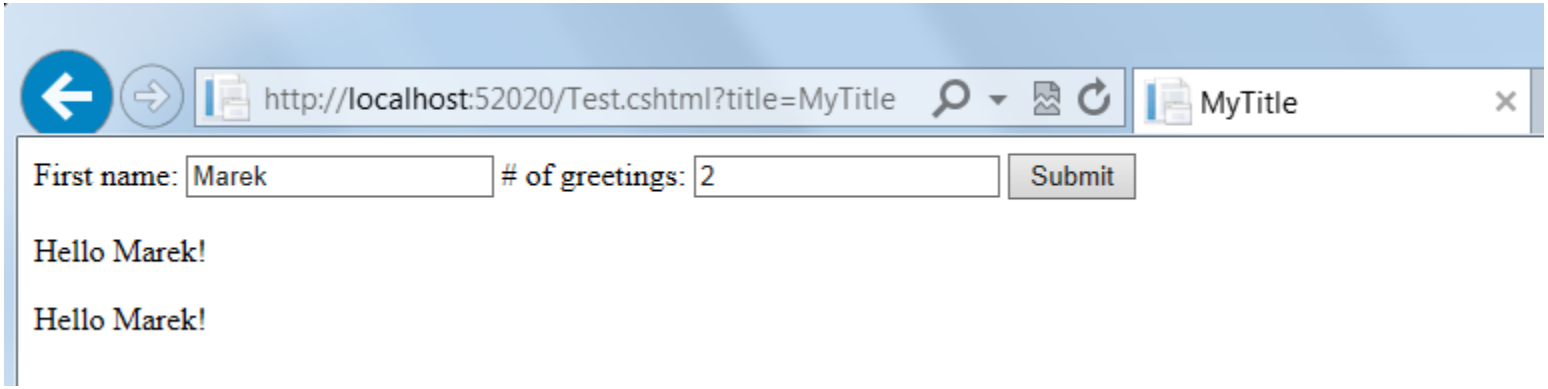
- SP.NET MVC from the beginning supported the concept of view engines - modules implementing different page templates syntax
  - Spark and NHaml as examples of template engines for use in ASP.NET MVC instead of the traditional, known from Web Forms, ASP.NET engine
  - Razor developed for MVC3, used in later MVC versions, but also in ASP.NET Web Pages 2
- Razor's major features
  - transparent code based on C # and VB languages
  - compact and transparent syntax: @ {...}, @expression
  - intelligent parser enables natural HTML interleaving with programming language
  - supports IntelliSense, unit tests, layouts
  - helpers for generating HTML forms

ASP.NET Web Pages: Example (1/2)

Test.cshtml

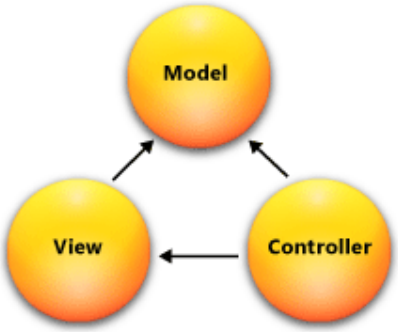
```
@{
   var title = Request.QueryString["title"];
   if (String.IsNullOrEmpty(title))    {  title = "Razor Web Page"; }
   var count = Request["count"].AsInt(1);
}
<html>
   <head><title>@title</title></head>
   <body>
      <form method="post">
         First name: @Html.TextBox("name", @Request["name"])
         # of greetings: @Html.TextBox("count", @count)
         <input type="submit" value="Submit" />
      </form>
      @{
        if(IsPost)  {
         for(int i = 0; i < count;  i++)  {
            <p>Hello @Request.Form["name"]!</p>   } }
      }
   </body></html>
```

ASP.NET Web Pages: Example (2/2)



ASP.NET MVC

- Microsoft's framework for ASP.NET
  - based on the Model-View-Controller (MVC) pattern
  - inspired by the Ruby on Rails framework
  - first production version: March 2009
  - presented as an alternative to Web Forms, not a successor
  - open source (MS-PL license), free, fully supported by Microsoft
  - in MVC3 the Razor view engine was introduced, at first as an alternative to classic syntax, finally replacing it



Advantages of ASP.NET MVC

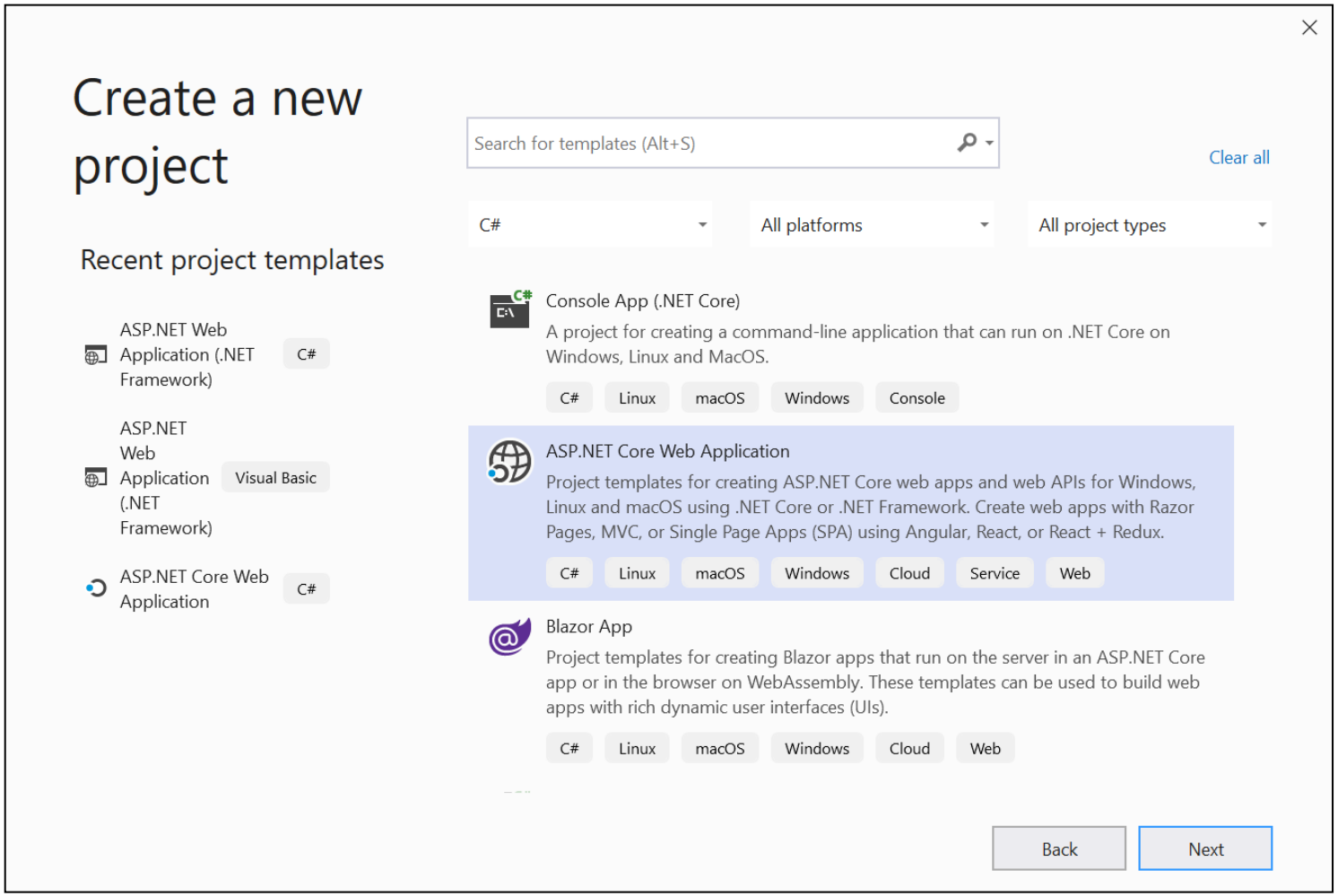
- Full control over generated HTML
  - „Embrace HTTP and HTML - don't hide it”
  - no view state and server-side form representations
- Possible integration with Ajax, jQuery
- Intuitive URL addresses
  - RESTful and friendly for search engines
- Separation of concerns within an application
- Testability
  - support for Test-Driven Development (TDD)
- Manages navigation between pages
  - Front Controller pattern, routing

Separation of concerns in ASP.NET MVC

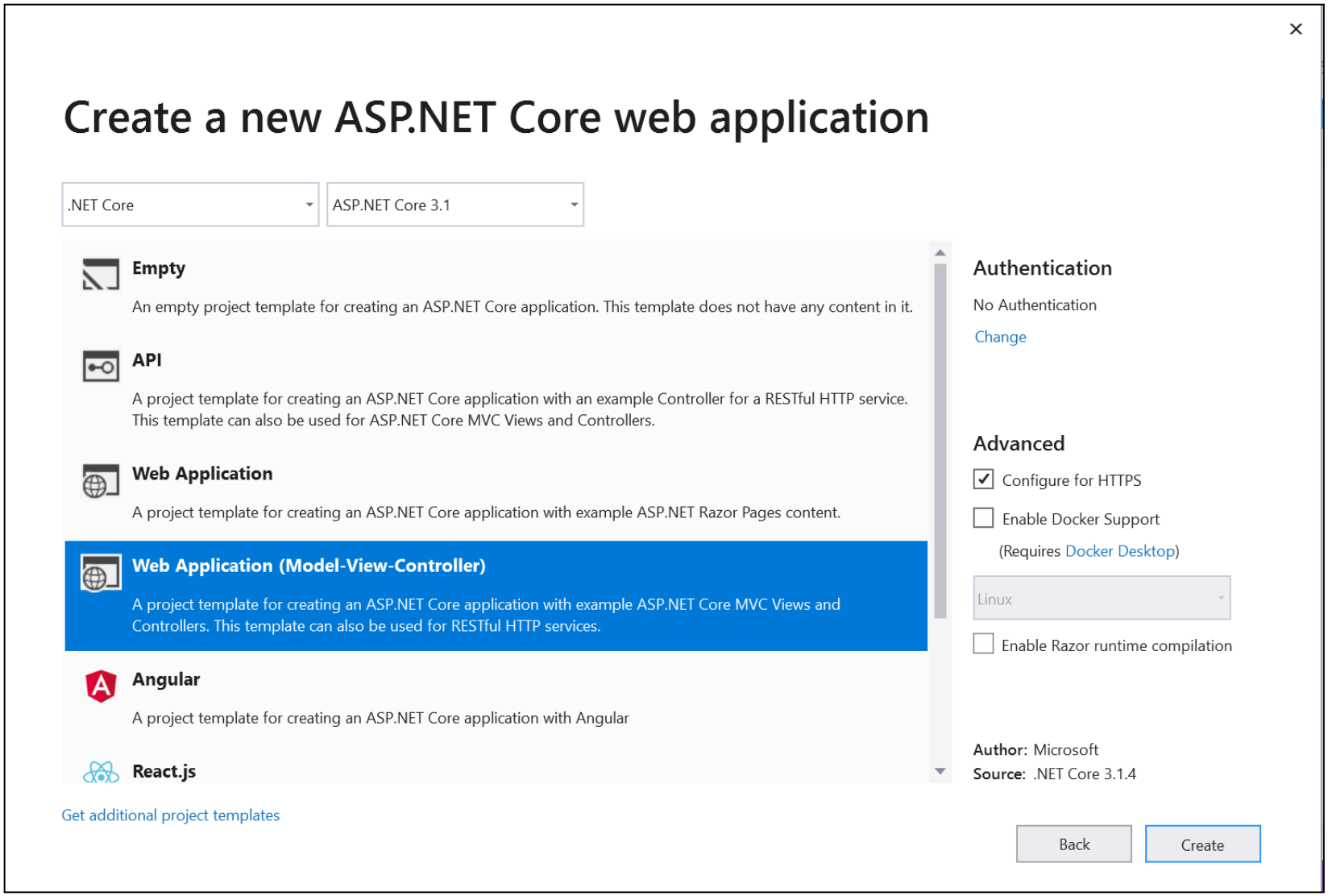
- Models
  - components responsible for maintaining the state
  - the state typically persisted in the database
- Views
  - components responsible for displaying the user interface of the application
  - the view can be generated from a model using a wizard (CRUD scaffolding)
- Controllers
  - components responsible for handling user interaction, manipulating the model, and selecting the view to display

Project wizard (Visual Studio 2019)

- Separate templates for .NET Framework and .NET Core

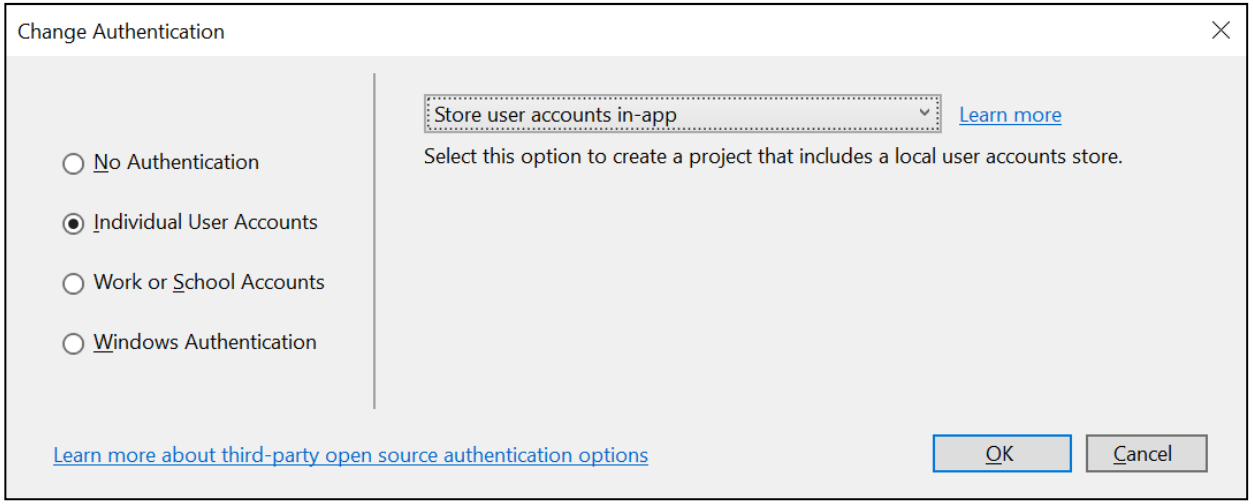


ASP.NET application wizard



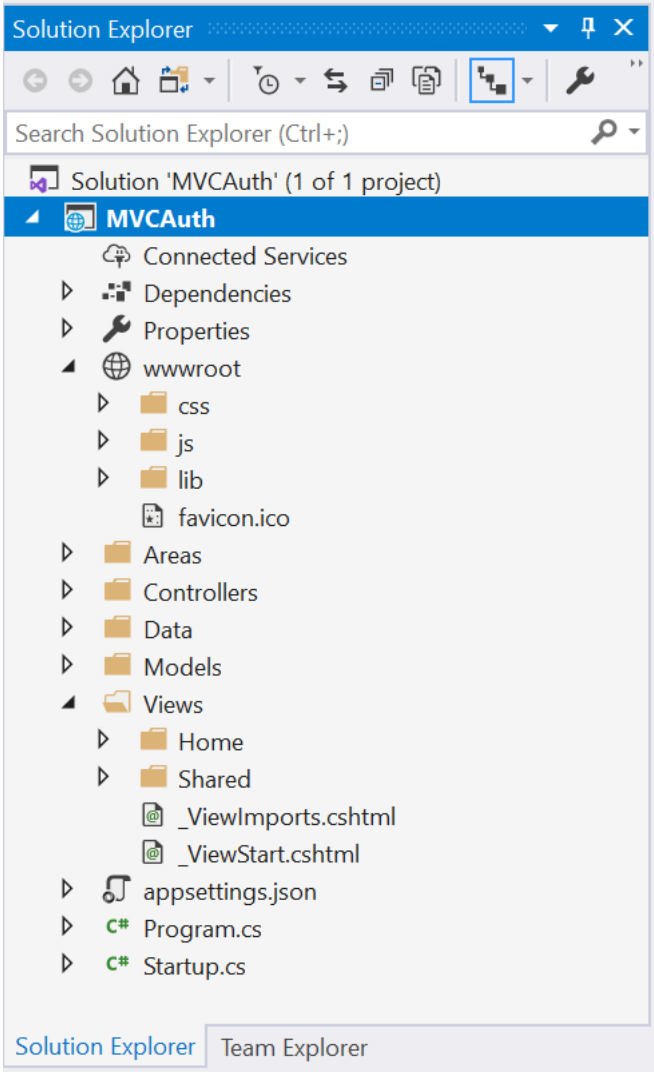
User authentication in ASP.NET

- No Authentication
- Individual User Accounts
  - Registration via apllication
  - Profiles in a local SQL Server database (in-app) or in the cloud
- Work or School Accounts
  - Active Directory / MS Azure AD / Office 365
- Windows Authentication
  - For intranet applications



ASP.NET MVC Core project structure

- Newly created project includes the start application, which can then be adapted and developed
  - controllers and views for the welcome page
  - page template (layout page)
  - authentication pages and code (if selected in the project wizard)
  - Startup.cs – application initialization (e.g., default routing rule)
  - appsettings.json – application configuration (e.g., database connection strings)



ASP.NET MVC starter application

MVCAuth Home Privacy Register Login

Welcome

Learn about building Web apps with ASP.NET Core.

MVCAuth Home Privacy Register Login

Privacy Policy

Use this page to detail your site's privacy policy.

© 2020 - MVCAuth - Privacy

MVCAuth Home Privacy Register Login

Register

Create a new account.

Email

Password

Confirm password

Register

Use another service to register.

There are no external authentication services configured. See this article for details on setting up this ASP.NET application to support logging in via external services.

MVCAuth Home Privacy Register Login

Log in

Use a local account to log in.

Email

marek@microsoft.com

Password

••••

Remember me?

Log in

Forgot your password?

Register as a new user

Use another service to log in.

There are no external authentication services configured. See this article for details

MVC Routing

Startup.cs

```
public class Startup
{
    …
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        ...
        app.UseRouting();

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

MVC Controller

HomeController.cs

```
public class HomeController : Controller
{
        …
        public IActionResult Index()
        {
            return View();
        }

        public IActionResult About()
        {
            ViewBag.Message = "Your application description page.";

            return View();
        }
        …
}
```

View

About.cshtml

```
@{
   ViewBag.Title = "About";
}
<h2>@ViewBag.Title.</h2>
<h3>@ViewBag.Message</h3>

<p>Use this area to provide additional information.</p>
```

Page layout

_Layout.cshtml

```
<html>
<head>
    <title>@ViewData["Title"] - MvcNews</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
…
 <ul>
   <li><a asp-area="" asp-controller="Home" asp-action="Index">Home</a></li>
   <li><a asp-area="" asp-controller="Home" asp-action ="Privacy">Privacy </a></li>
 </ul>
…
 <div class="container">
        <main role="main">
            @RenderBody()
        </main>
 </div>
…
</body>
</html>
```

Sharing code between views

- _ViewStart files
  - In the Views folder
  - In subfolders for views associated with controllers
- Executed and the beginning of view rendering

_ViewStart.cshtml

```
@{
   Layout = "_Layout";
}
```

Passing data from the controller to the view

- ViewData, ViewBag (from MVC 3)
  - Data available during a single request and not surviving the redirect operation
  - ViewData["key"], ViewBag.key
  - ViewBag is an abstraction over ViewData (not available in Razor Pages!)
- TempData
  - Data surviving redirection
  - TempData["key"]
  - Internally uses HTTP session
- View models and strongly typed views
  - Model objects passed to views
  - Preferred solution

ASP.NET MVC View Model Patterns

- Domain model as a view model
  - e.g., Entity Framework entity
- Dedicated view model containg a domain model object (or objects) (+ data needed for presentation)
- Dedicated view model containing data from the data model (+ data needed for presentation)
  - requires translation between domain models and view models

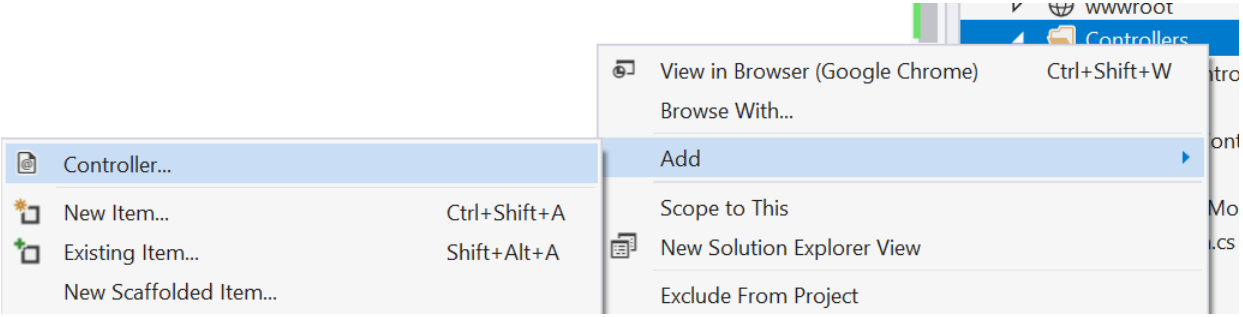
Domain object: Example (EF)

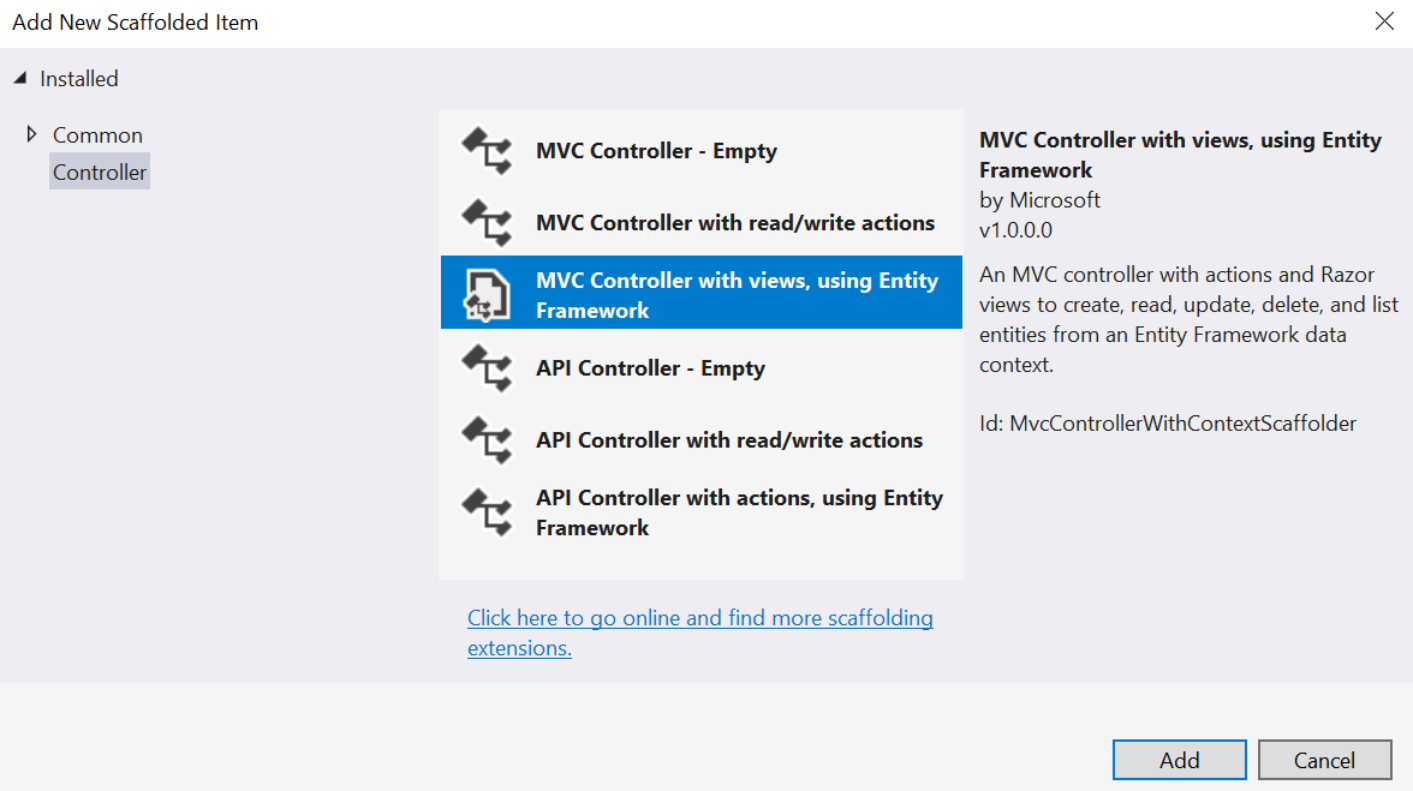
```
namespace MvcNews.Models
{
    public class NewsItem
    {
        [Key]
        public int Id { get; set; }
        public DateTime TimeStamp { get; set; }
        public string Text { get; set; }
    }
}
```

Scaffolding in ASP.NET MVC

- The technique of generating CRUD functionality for a domain object (controller + views)
- Visual Studio wizards for ASP.NET MVC:
  - complete scaffolding from EF entity to a controller with CRUD actions and corresponding views
  - scaffolding from EF entity to a controller with CRUD actions (without views)
  - Views created individually for controller actions with a possibility of selecting a data model class

Complete CRUD scaffolding in ASP.NET MVC (1/2)





Complete CRUD scaffolding in ASP.NET MVC (2/2)

Add MVC Controller with views, using Entity Framework

Model class: NewsItem (MvcNews.Models)

Data context class: NewsDbContext (MvcNews.Data) +

Views:

- [x] Generate views
- [x] Reference script libraries
- [x] Use a layout page: ...

(Leave empty if it is set in a Razor _viewstart file)

Controller name: NewsController

Add Cancel

Selective creation of strongly-typed views

```
20          }
21
22          // GET: News
            3 references
23          public async Task<IActionResult> Index()
24          {
25              return View(await _context.News.ToListAsync()
26          }
```

- Add View...
- Go To View — Ctrl+M, Ctrl+G
- Quick Actions and Refactorings... — Ctrl+.
- Rename... — Ctrl+R, Ctrl+R
- Remove and Sort Usings — Ctrl+R, Ctrl+G
- Peek Definition — Alt+F12

Add MVC View

View name: Index

Template: List

Model class: NewsItem (MvcNews.Models)

Data context class: NewsDbContext (MvcNews.Data)

Options:

- [ ] Create as a partial view
- [x] Reference script libraries
- [x] Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

Add | Cancel

- Create
- Delete
- Details
- Edit
- Empty (without model)
- List

Asynchronous controller actions

- ASP.NET supports the creation of asynchronously called controller action methods (async/await syntax)
- Asynchronous calls are recommended for actions requiring access to external resources
  - If access to resources can be performed asynchronously
- Asynchronous controller actions increase application scalability
  - A server thread waiting for a response from an external service is not blocked by waiting, but returns to the service thread pool and can be assigned to handle another request
  - It should be remembered that asynchronous processing involves a certain overhead
- The MVC scaffolding Wizard in VS 2019 generates asynchronous actions by default when they work with EF
  - Using asynchronous Entity Framework operations

Asychronous MVC Controller: Example

NewsController.cs

```
public class NewsController : Controller
{
        …
        // GET: News
        public async Task<IActionResult> Index()
        {
            return View(await _context.News.ToListAsync());
        }

        // POST: News/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var newsItem = await _context.News.FindAsync(id);
            _context.News.Remove(newsItem);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        …
}
```

Protection against CSRF attacks

- Cross Site Request Forgery (CSRF, XSRF)
  - an attack using the site's trust in the user's browser (request from the logged-in user's browser, "tossed" from another website)
- CSRF protection in ASP.NET MVC based on Anti-Forgery Tokens
  - randomly generated information attached to the form in the hidden field and set as a cookie at the same time
  - after receiving the form data from the browser, the tokens received from the hidden field and from the cookie are compared
- Implicit and explicit token generation
  - automatically for using the POST method with no ACTION (or ACTION="")
  - explicitly using a helper within the form: @Html.AntiForgeryToken()

AntiForgeryToken in ASP.NET MVC

View with a form

```
<form action="..." method="post">
   @Html.AntiForgeryToken()
</form>
```

Controller processing form data

```
 [ValidateAntiForgeryToken]
 public IActionResult Action(…)
{
   …
}
```

Server Error in '/' Application.

A required anti-forgery token was not supplied or was invalid.