# Essential ASP.NET Core MVC

Suthep Sangvirotjanaphat

Greatfriends.biz

# Instructor

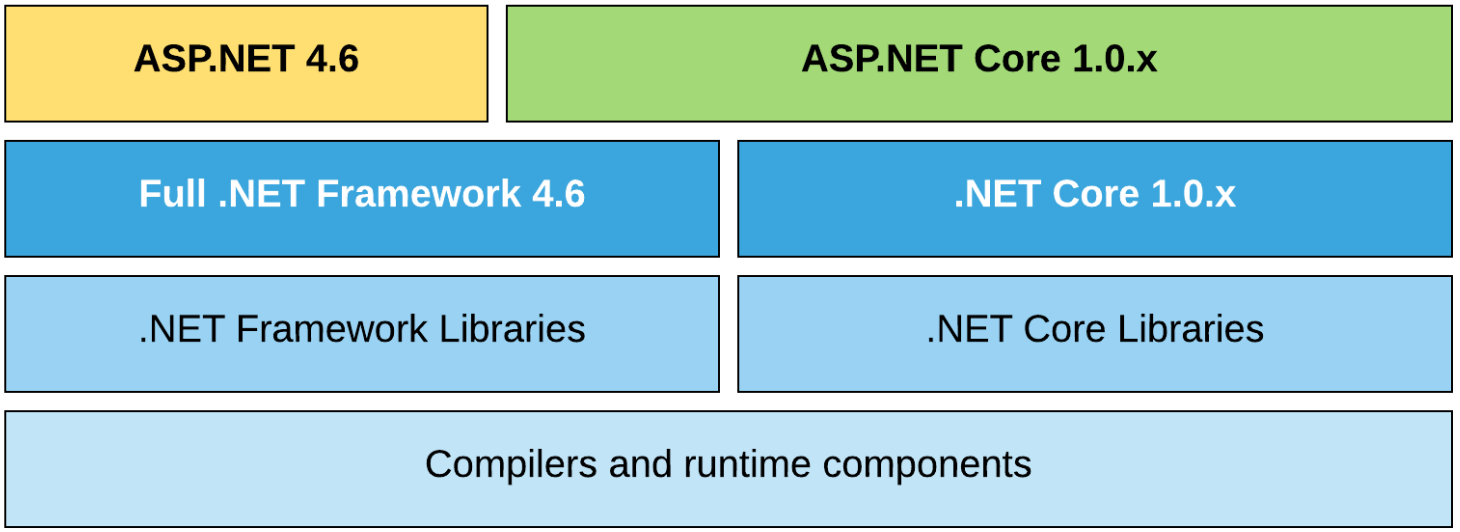
## Suthep Sangvirotjanaphat


- Microsoft MVP 2004-2015 (12 years)
- Opt-out from MVP by myself this year.
- P: 081-915-7816 E: suthep@gfbd.co.th
- LINE: suthep.s
- Web: http://next.greatfriends.biz
- FB: http://fb.com/suthep
- FB: http://fb.com/greatfriends.biz
- FB: http://fb.com/groups/greatfriends.biz

- Feel free to contact me for public and in-house training courses. I'm also provide consulting and software development service.

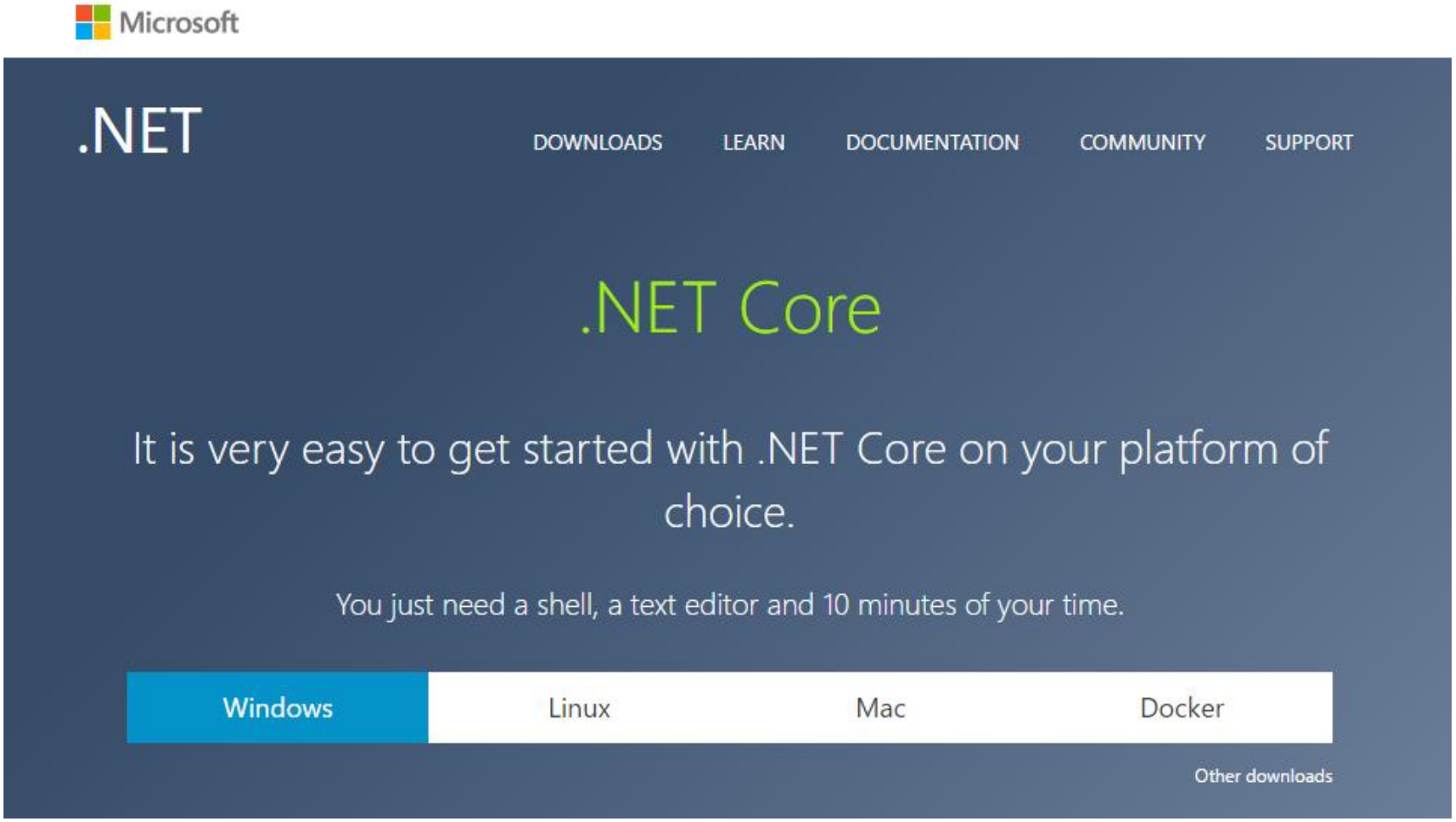
# .NET Core

# ASP.NET Core Stack



# Install .NET Core

## https://www.microsoft.com/net/core



# https://www.microsoft.com/net/download

| | .NET Core SDK Installer | .NET Core Installer | .NET Core SDK binaries only | .NET Core binaries only |
|---|---|---|---|---|
| Windows | x64 / x86 .exe | x64 / x86 .exe | x64 / x86 .zip | x64 / x86 .zip |
| Windows (Server Hosting)[2] | - | x64 / x86 .exe | - | - |
| macOS[3] | x64 .pkg | x64 .pkg | x64 .tar.gz | x64 .tar.gz |
| RHEL[4] | `yum install rh-dotnetcore10` (more...) | | x64 .tar.gz | x64 .tar.gz |
| Ubuntu 14.04[5] | x64 .deb | - | x64 .tar.gz | x64 .tar.gz |
| Ubuntu 16.04[5] | x64 .deb | - | x64 .tar.gz | x64 .tar.gz |
| Debian 8 | - | - | x64 .tar.gz | x64 .tar.gz |
| Fedora 23 | - | - | x64 .tar.gz | x64 .tar.gz |
| CentOS 7.1 | - | - | x64 .tar.gz | x64 .tar.gz |
| openSUSE 13.2 | - | - | x64 .tar.gz | x64 .tar.gz |

# .NET Core Characteristics

- **Flexible deployment:** Can be included in your app or installed side-by-side user- or machine-wide.
- **Cross-platform:** Runs on Windows, macOS and Linux; can be ported to other OSes. The supported Operating Systems (OS), CPUs and application scenarios will grow over time, provided by Microsoft, other companies, and individuals.
- **Command-line tools:** All product scenarios can be exercised at the command-line.
- **Compatible:** .NET Core is compatible with .NET Framework, Xamarin and Mono, via the .NET Standard Library.
- **Open source:** The .NET Core platform is open source, using MIT and Apache 2 licenses. Documentation is licensed under CC-BY. .NET Core is a .NET Foundation project.
- **Supported by Microsoft:** .NET Core is supported by Microsoft, per .NET Core Support

# .NET Core Composition

- A .NET runtime, which provides a type system, assembly loading, a garbage collector, native interop and other basic services.
- A set of framework libraries, which provide primitive data types, app composition types and fundamental utilities.
- A set of SDK tools and language compilers that enable the base developer experience, available in the .NET Core SDK.
- The 'dotnet' app host, which is used to launch .NET Core apps. It selects the runtime and hosts the runtime, provides an assembly loading policy and launches the app. The same host is also used to launch SDK tools in much the same way.

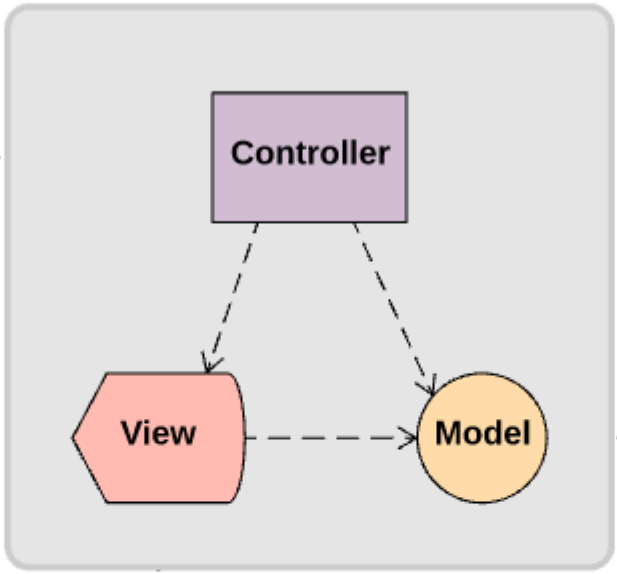
**Controller** is a coordinator,
a manager,
or a music conductor

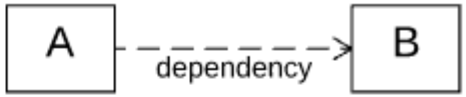
It knows how to
complete the request

Know which models needed to
complete the job and how they
should work together

Controller usually has no states
(most are disposed
after response)

It has **application logics**




*(A depends on B)*

**Model** in MVC is also
*Domain Model*
or *Entity*

we model them from
the real-world objects

It has **data**
and **business logics**

Object should *responsible* for itself

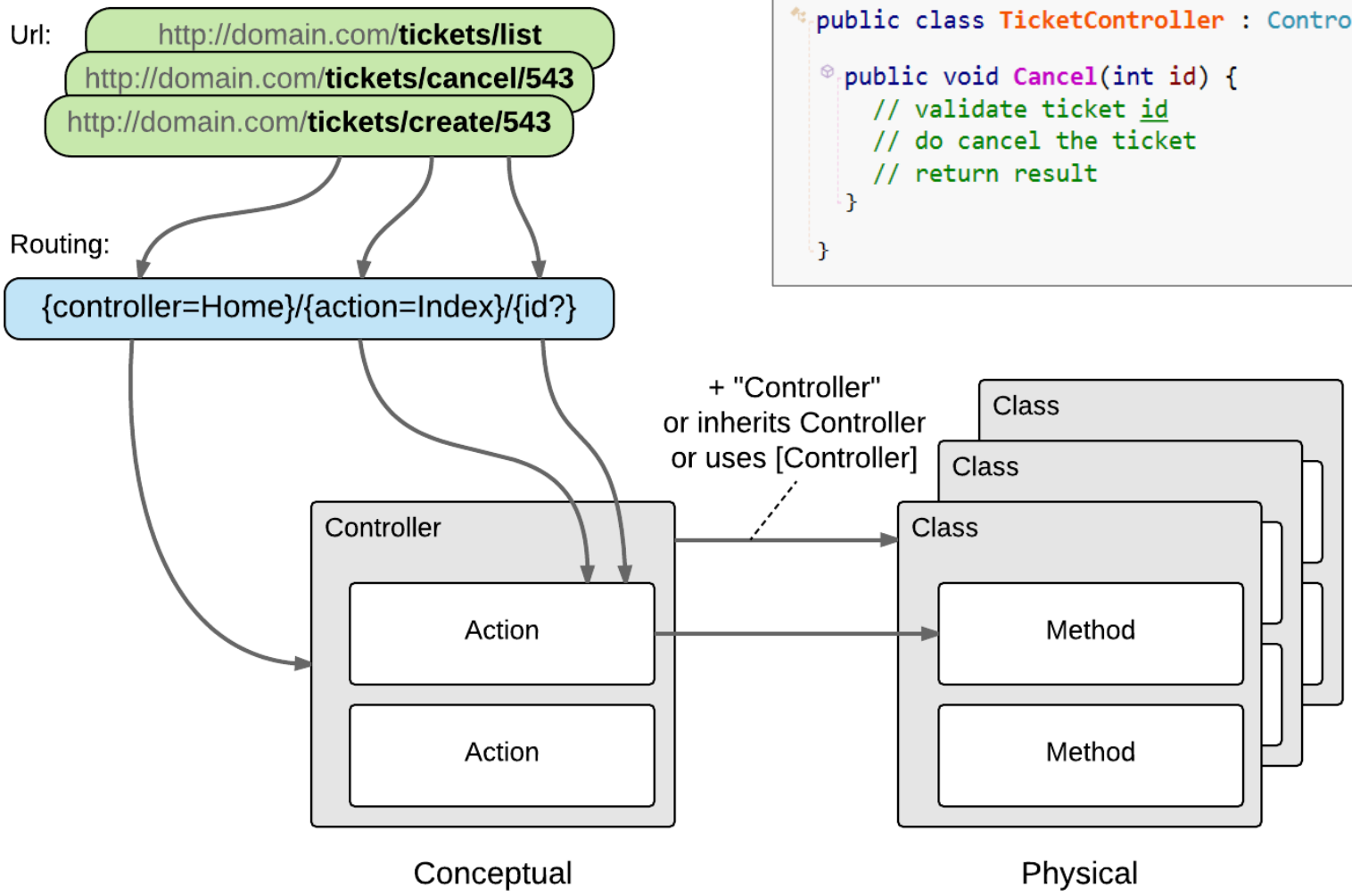
**Independent** from technologies,
application types, UIs,
and database related code

**View** accepts Model from Controller and knows
how to present it.

View contains just only 2 things,
Markup and Code, using brand-new Razor
syntax (.cshtml or .vbhtml files)

Code should only responsible how to
presenting the model, **presentation logics**.

# How Url maps to Code

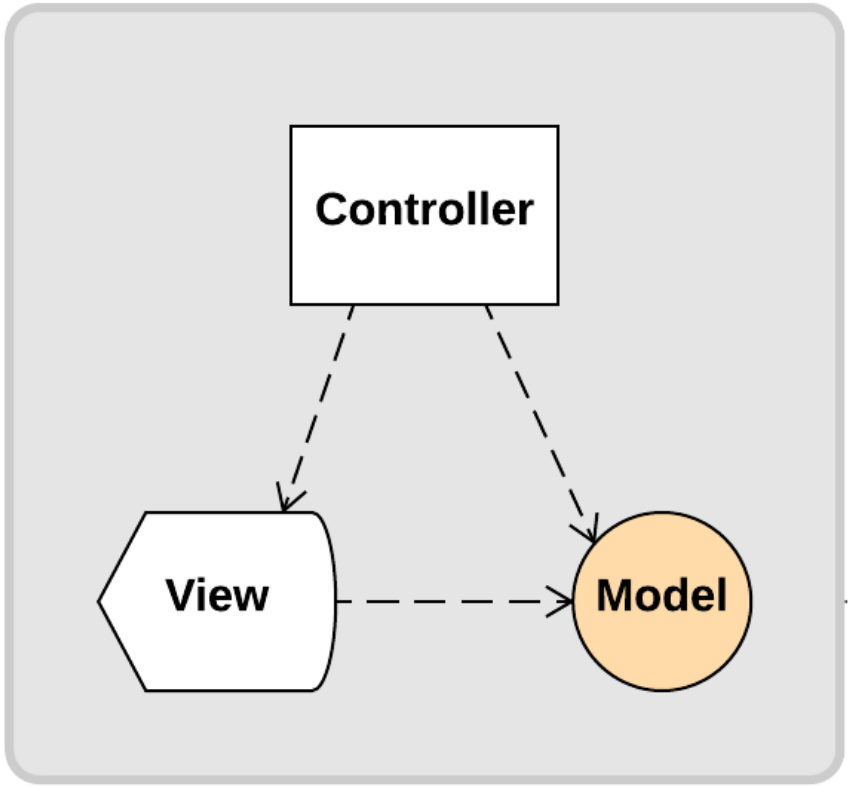



```
public class TicketController : Controller {

    public void Cancel(int id) {
        // validate ticket id
        // do cancel the ticket
        // return result
    }

}
```

# Model




**Model** in MVC is also
*Domain Model*
or *Entity*

we model them from
the real-world objects

It has **data**
and **business logics**

Object should *responsible* for itself

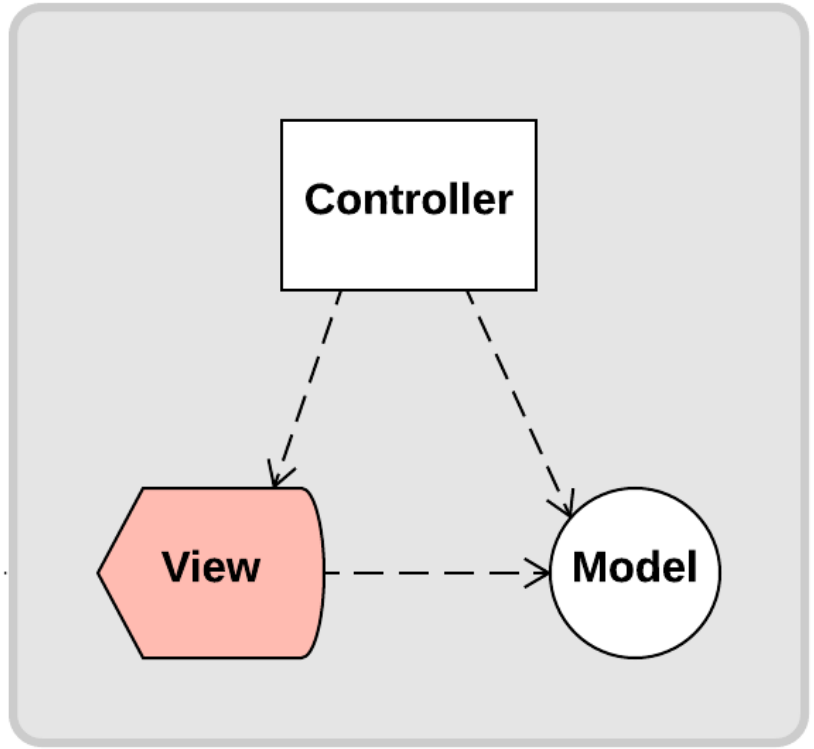
**Independent** from technologies,
application types, UIs,
and database related code

# View

**View** accepts Model from Controller and knows how to present it.

View contains just only 2 things, Markup and Code, using brand-new Razor syntax (.cshtml or .vbhtml files)

Code in the View should only responsible how to presenting the model, that is **presentation logics**.



# Controller

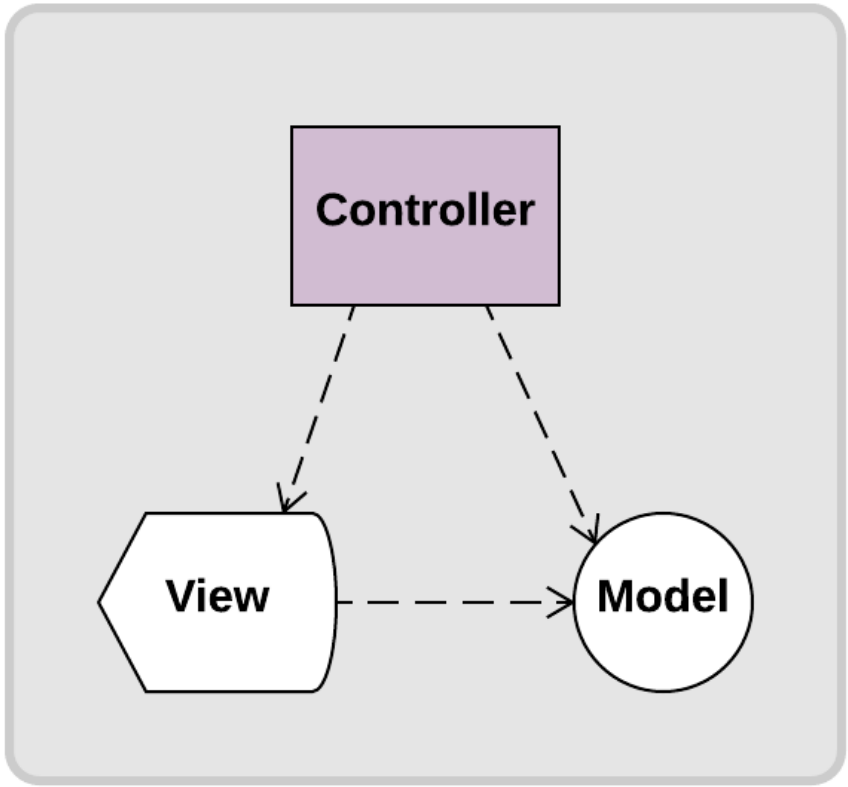
**Controller** is a coordinator,
a manager,
or a music conductor

It knows how to
complete the request

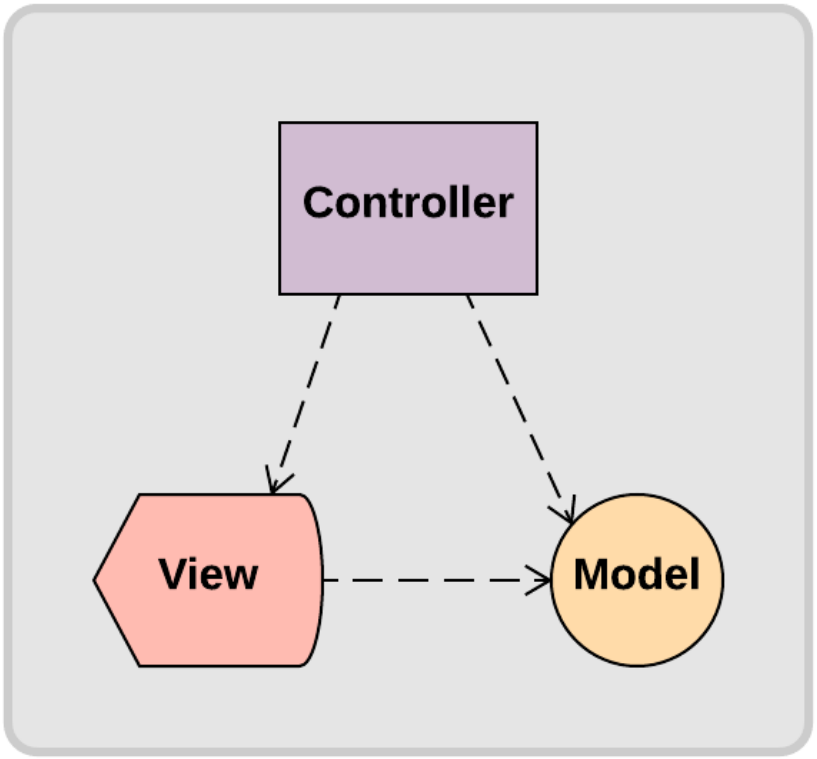
Know which models needed to
complete the job and how they
should work together

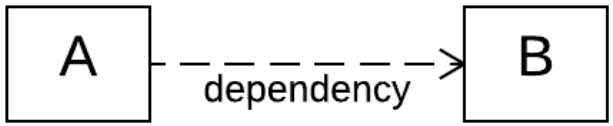
Controller usually has no states
(most are disposed
after response)

It has **application logics**



# MVC Dependencies

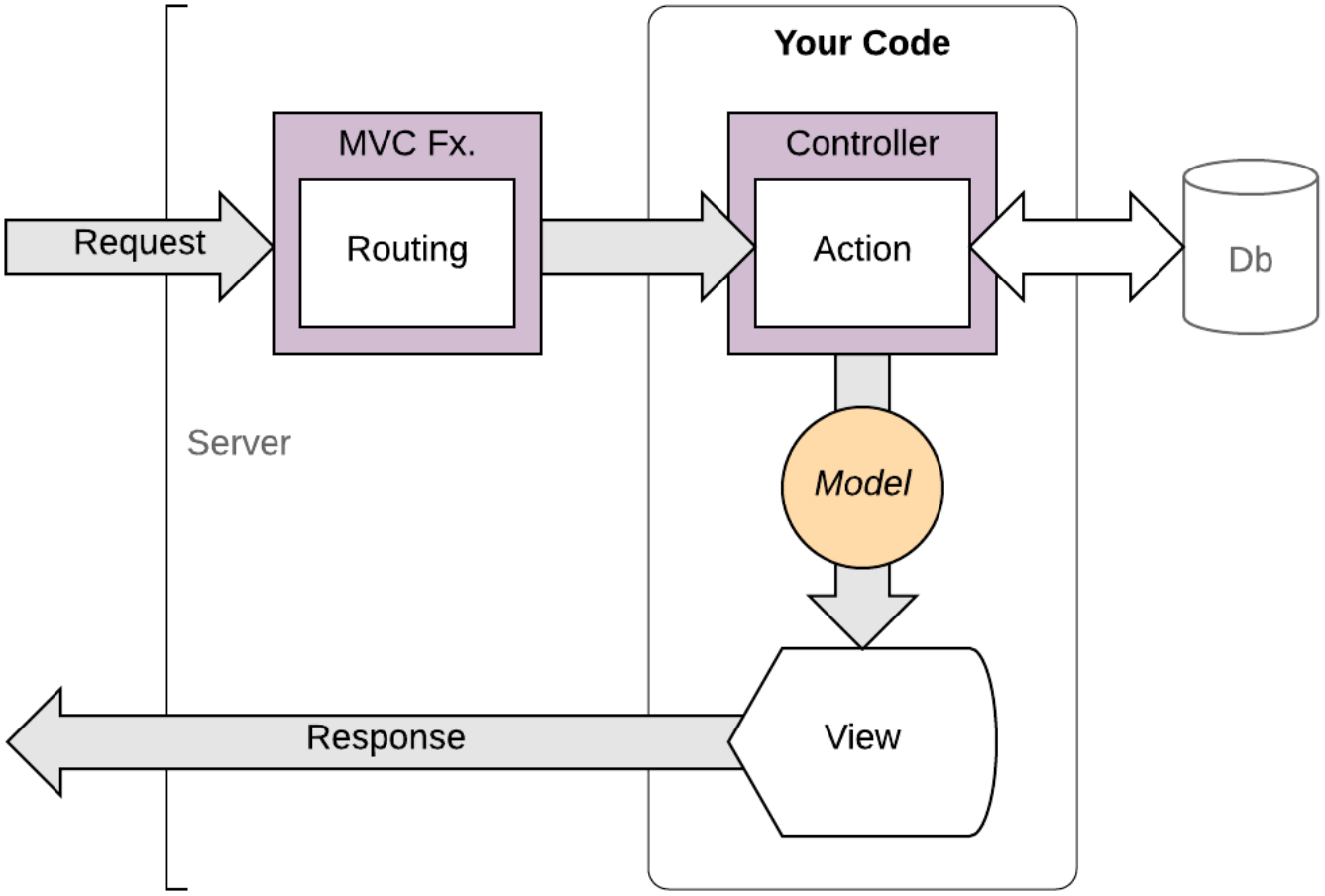



- A depends on B.
- A uses B.
- A refers to B.
- A knows B.
- A cannot works without B.
- A have to change if B changed.

- But not vice versa.

# MVC flows


Your Code
MVC Fx.
Routing
Controller
Action
Request
Db
Server
Model
Response
View

# dotnet CLI

# dotnet CLI

- dotnet new
- dotnet restore
- dotnet build
- dotnet run
- dotnet test
- dotnet pack
- dotnet publish
- dotnet install-script

https://docs.microsoft.com/en-us/dotnet/articles/core/tools/dotnet

# dotnet new

```
> dotnet new -h
.NET Initializer

Usage: dotnet new [options]

Options:
  -h|--help                Show help information
  -l|--lang <LANGUAGE>     Language of project [C#|F#]
  -t|--type <TYPE>         Type of project
```

Type can be console, web, lib and xunittest.

# dotnet restore, build, run, test

```
dotnet restore [--source] [--packages] [--disable-parallel]
[--fallbacksource] [--configfile] [--verbosity] [<root>]
```

```
dotnet build [--output] [--build-base-path] [--framework] [--configuration]
[--runtime] [--version-suffix] [--build-profile] [--no-incremental] [--no-
dependencies] [<project>]
```

```
dotnet run [--framework] [--configuration] [--project] [--help] [--]
```

```
dotnet test [--configuration] [--output] [--build-base-path] [--framework] [-
-runtime] [--no-build] [--parentProcessId] [--port] [<project>]
```

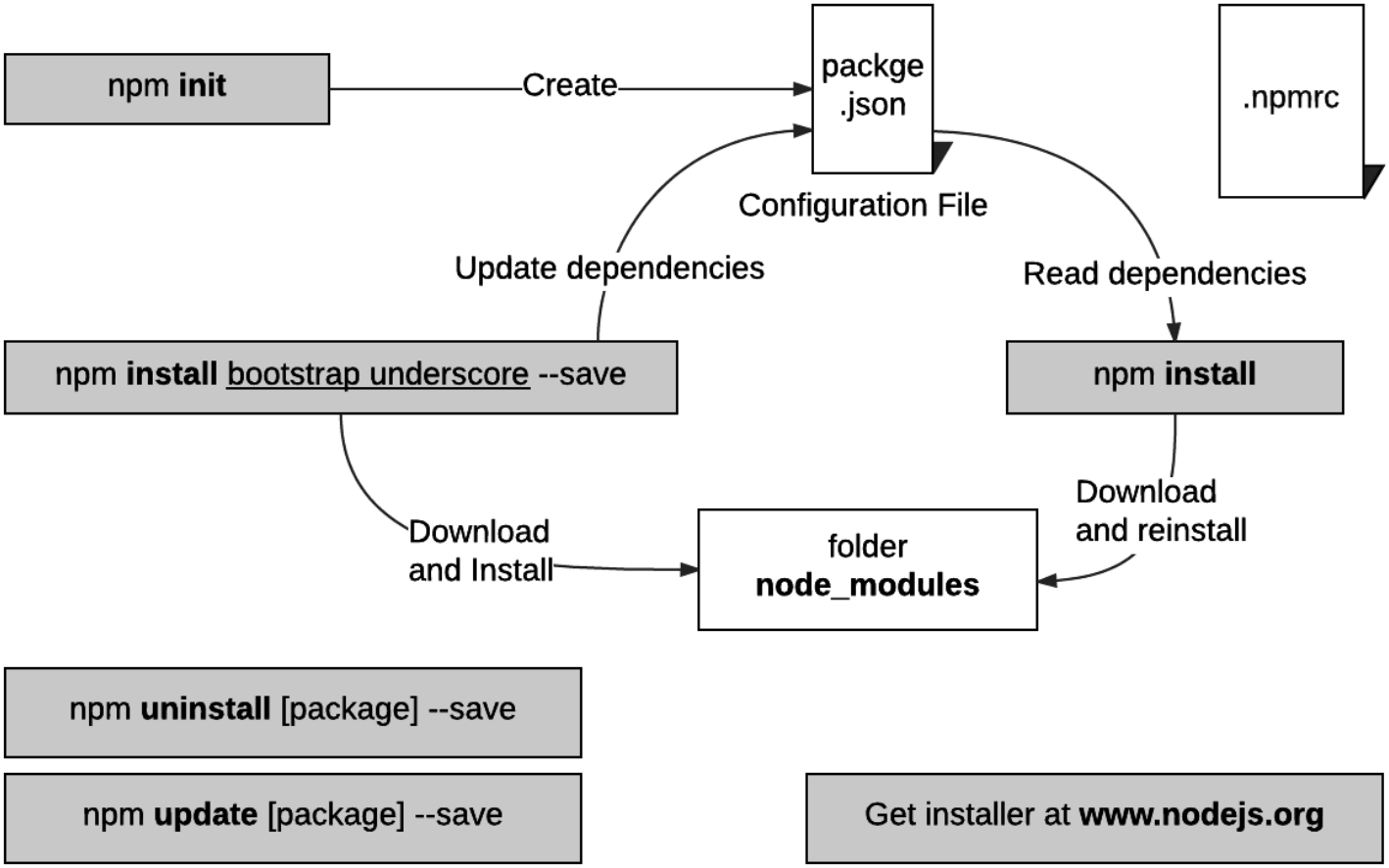
# dotnet pack and publish

```
dotnet pack [--output] [--no-build] [--build-base-path] [--configuration] [--
version-suffix] [<project>]
```

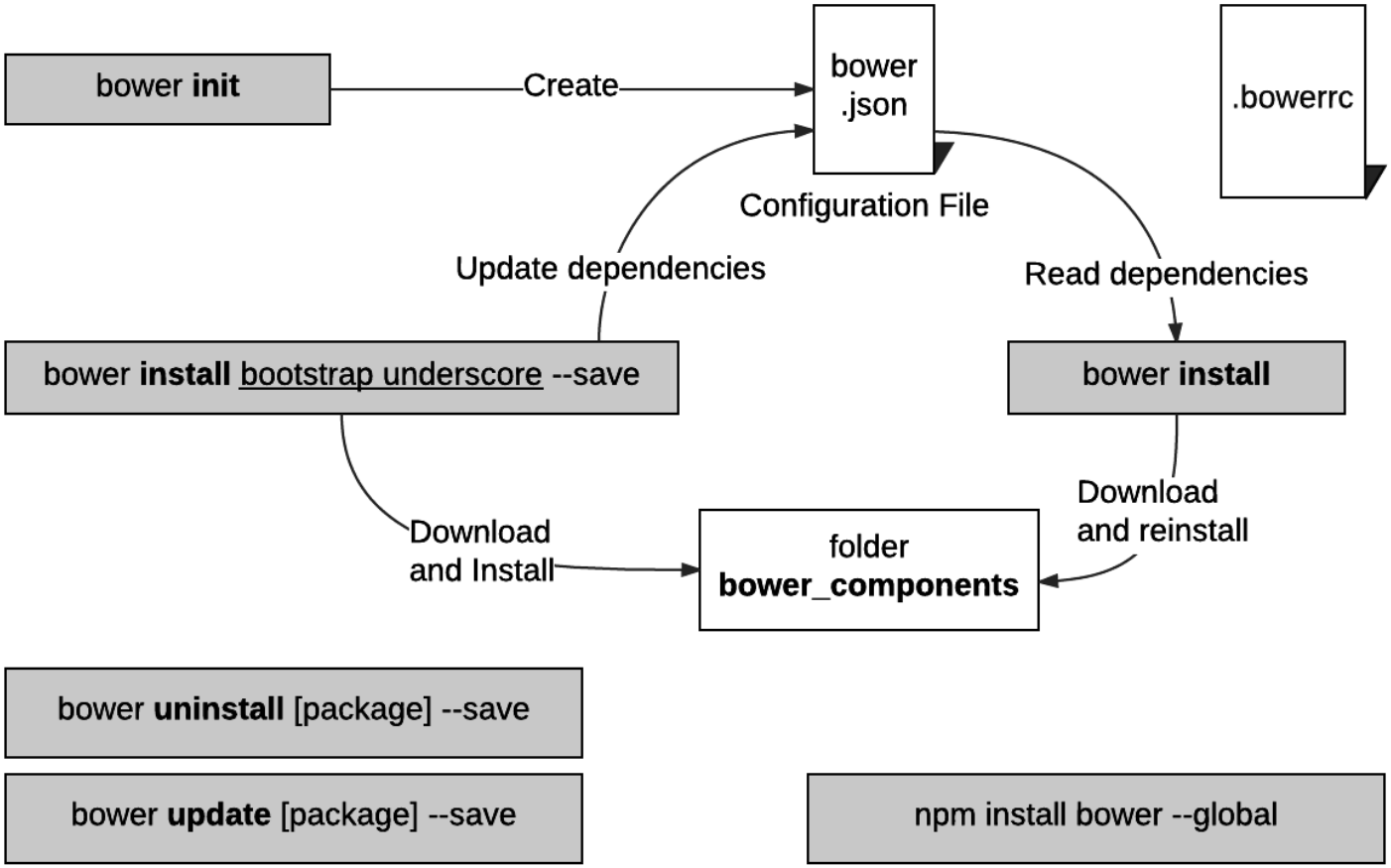
```
dotnet publish [--framework] [--runtime] [--build-base-path] [--output] [--
version-suffix] [--configuration] [<project>]
```

# Client-side Web developement tools

# npm
## Node App package manager



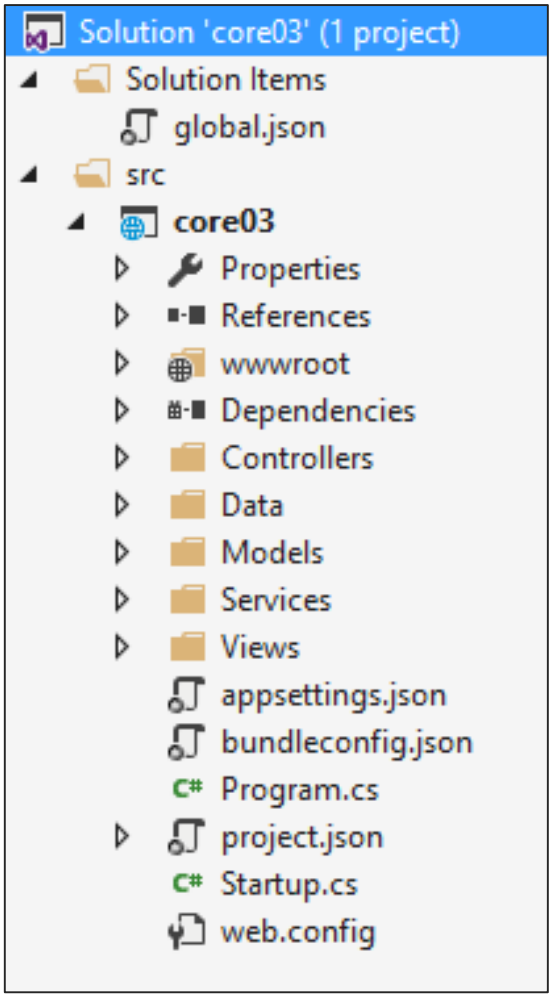
# bower
## web package manager



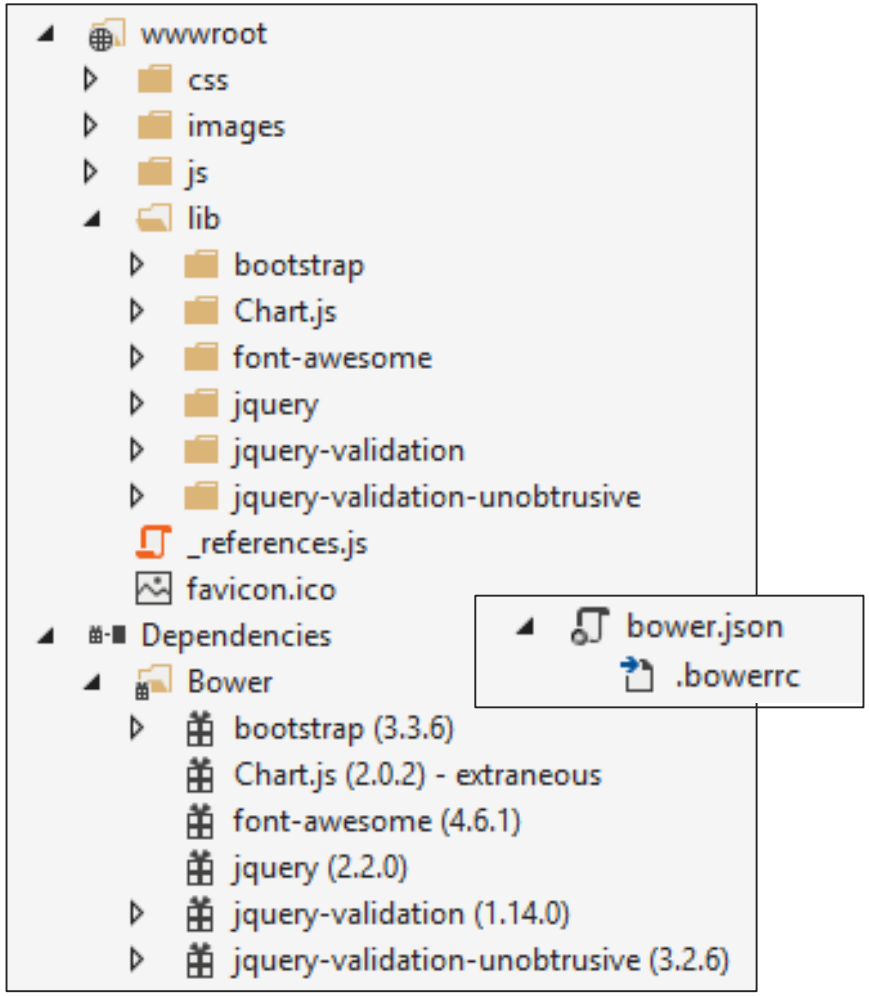
# ASP.NET Core

Creating ASP.NET Core MVC with Membership template

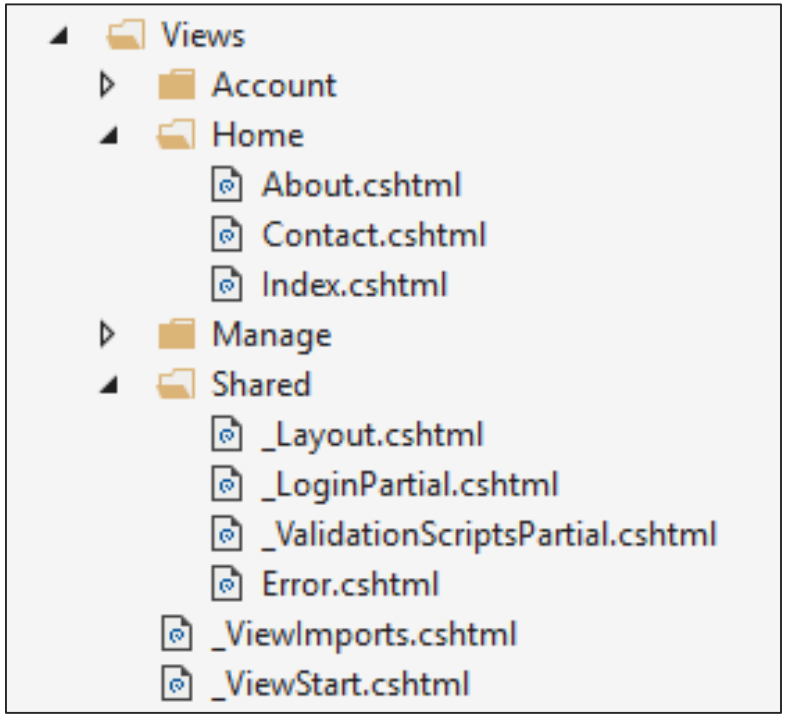
# Explore the project (1)



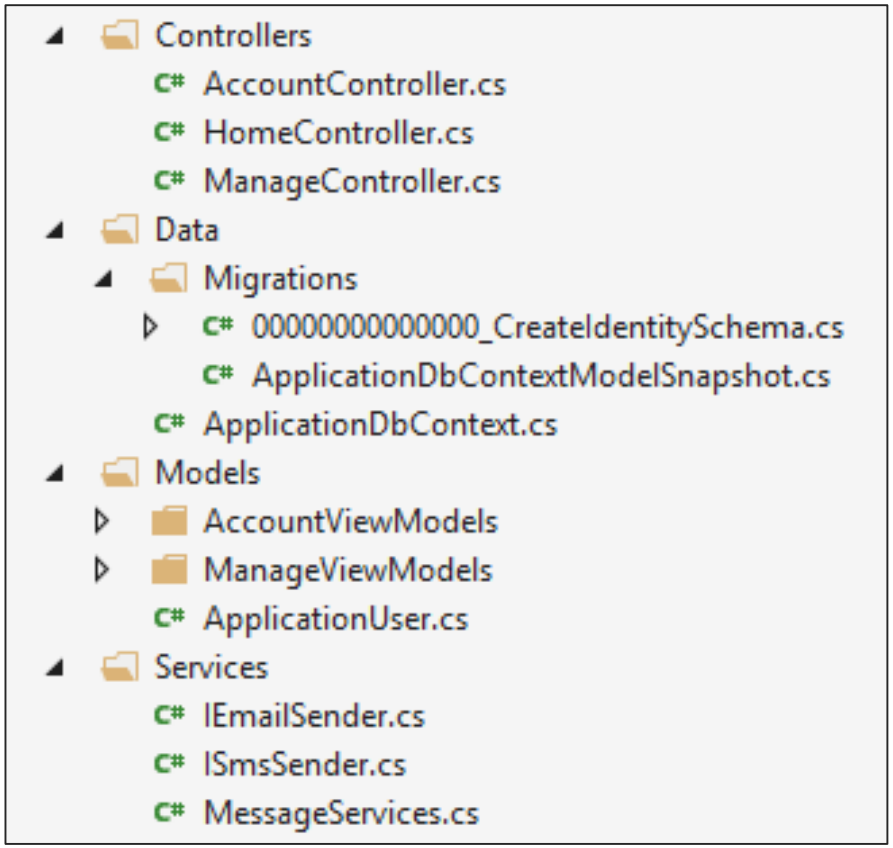
# Explore the project (2)



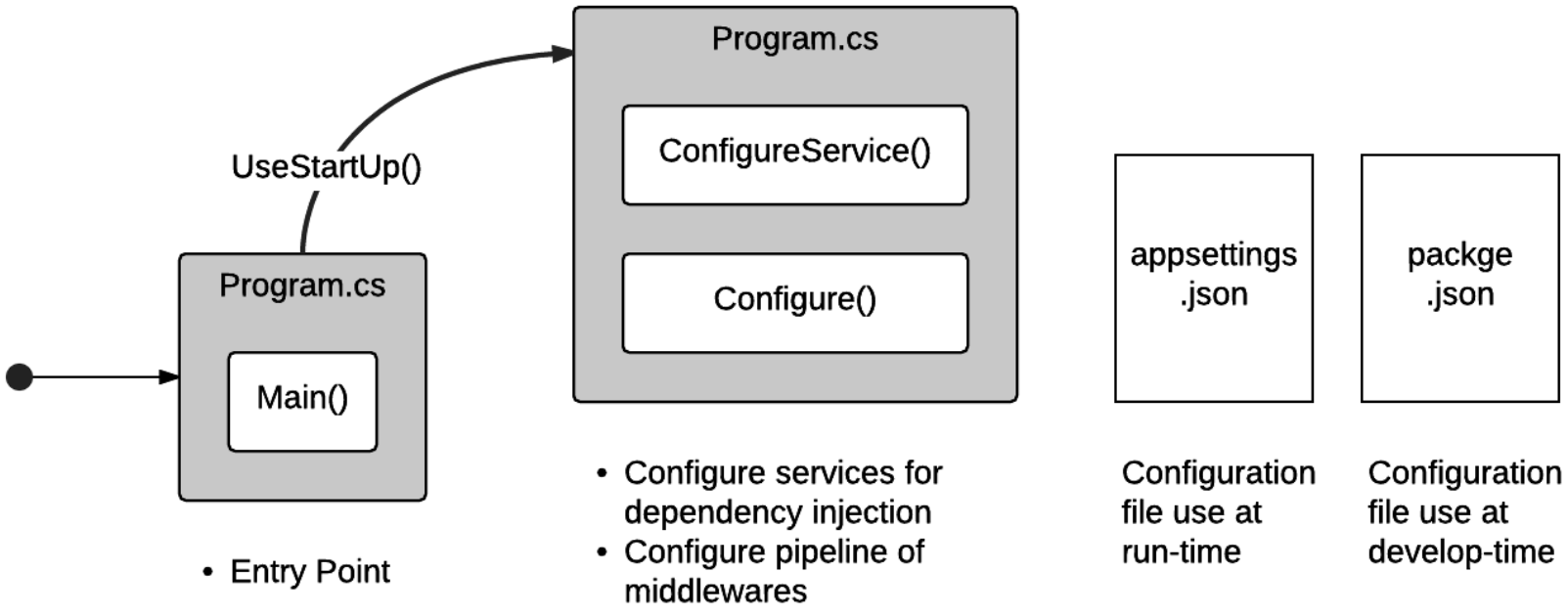
# Explore the project (3)



# Explore the project (4)



# Main items in ASP.NET Core project



# Program.cs

```
using System.IO;
using Microsoft.AspNetCore.Hosting;

namespace core01 {
  public class Program {
    public static void Main(string[] args) {
      var host = new WebHostBuilder()
          .UseKestrel()
          .UseContentRoot(Directory.GetCurrentDirectory())
          .UseIISIntegration()
          .UseStartup<Startup>()
          .Build();

      host.Run();
    } « void Main(string[])
  } « class Program
} « namespace core01
```

# Startup.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;

namespace mvc_core {
  public class Startup {

    public void ConfigureServices(IServiceCollection services) {
    }

    public void Configure(IApplicationBuilder app,
                          IHostingEnvironment env,
                          ILoggerFactory loggerFactory) {
      loggerFactory.AddConsole();

      if (env.IsDevelopment()) {
        app.UseDeveloperExceptionPage();
      }

      app.Run(async (context) => {
        await context.Response.WriteAsync("Hello World!");
      });
    } « void Configure(IApplicationBuilder, IHostingEnvironment, ILoggerFact
  } « class Startup
} « namespace mvc_core
```

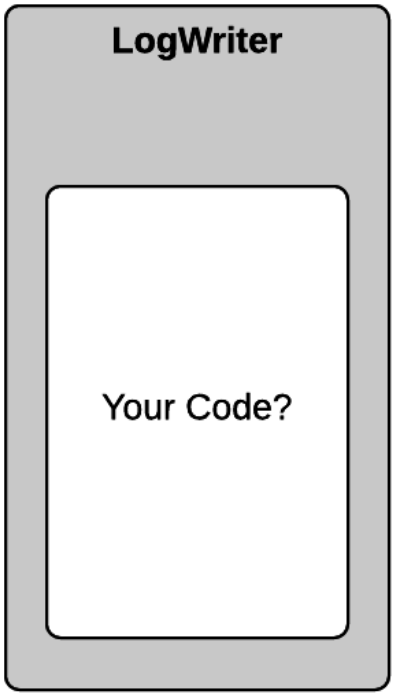
# project.json

```
{
  "dependencies": ...,
  "tools": ...,
  "frameworks": ...,
  "buildOptions": ...,
  "runtimeOptions": ...,
  "publishOptions": ...,
  "scripts": ...
}
```

# appsettings.json

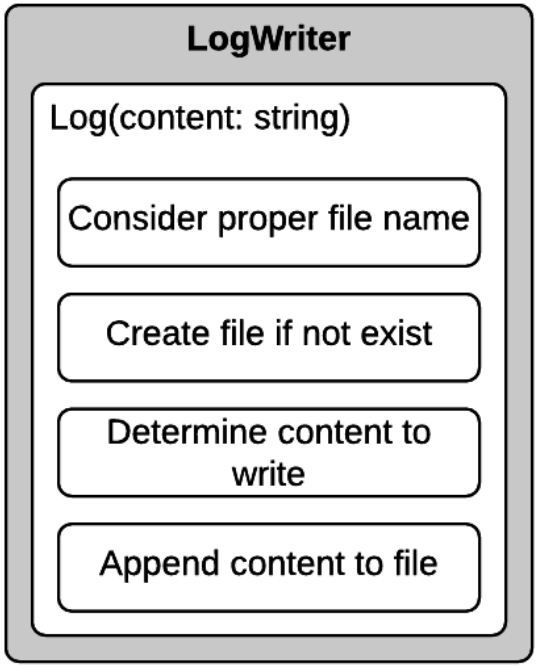
```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mss
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  }
}
```

# Dependency Injection (1)



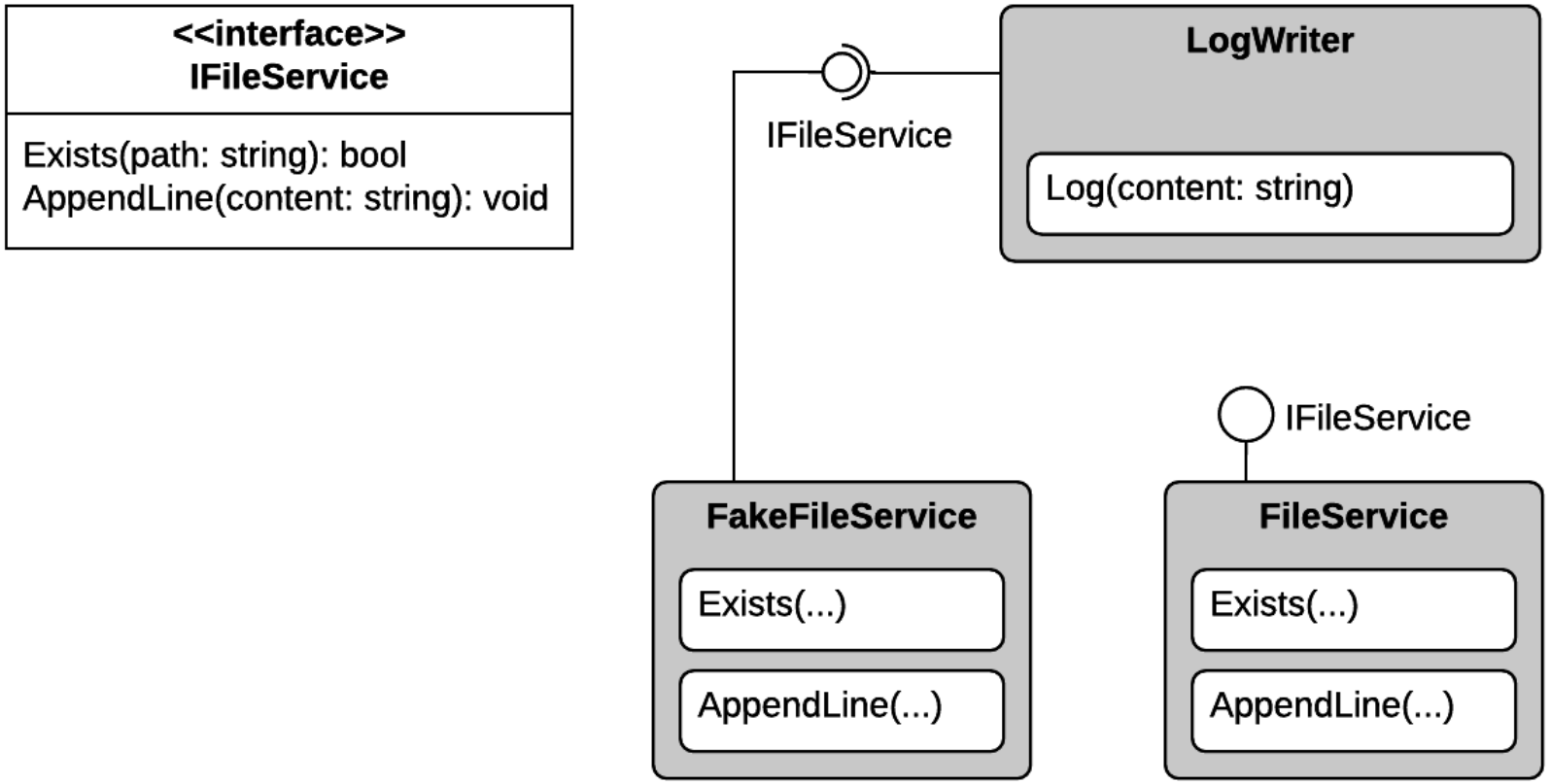

- Write message to daily log file with timestamp.

# Dependency Injection (2)

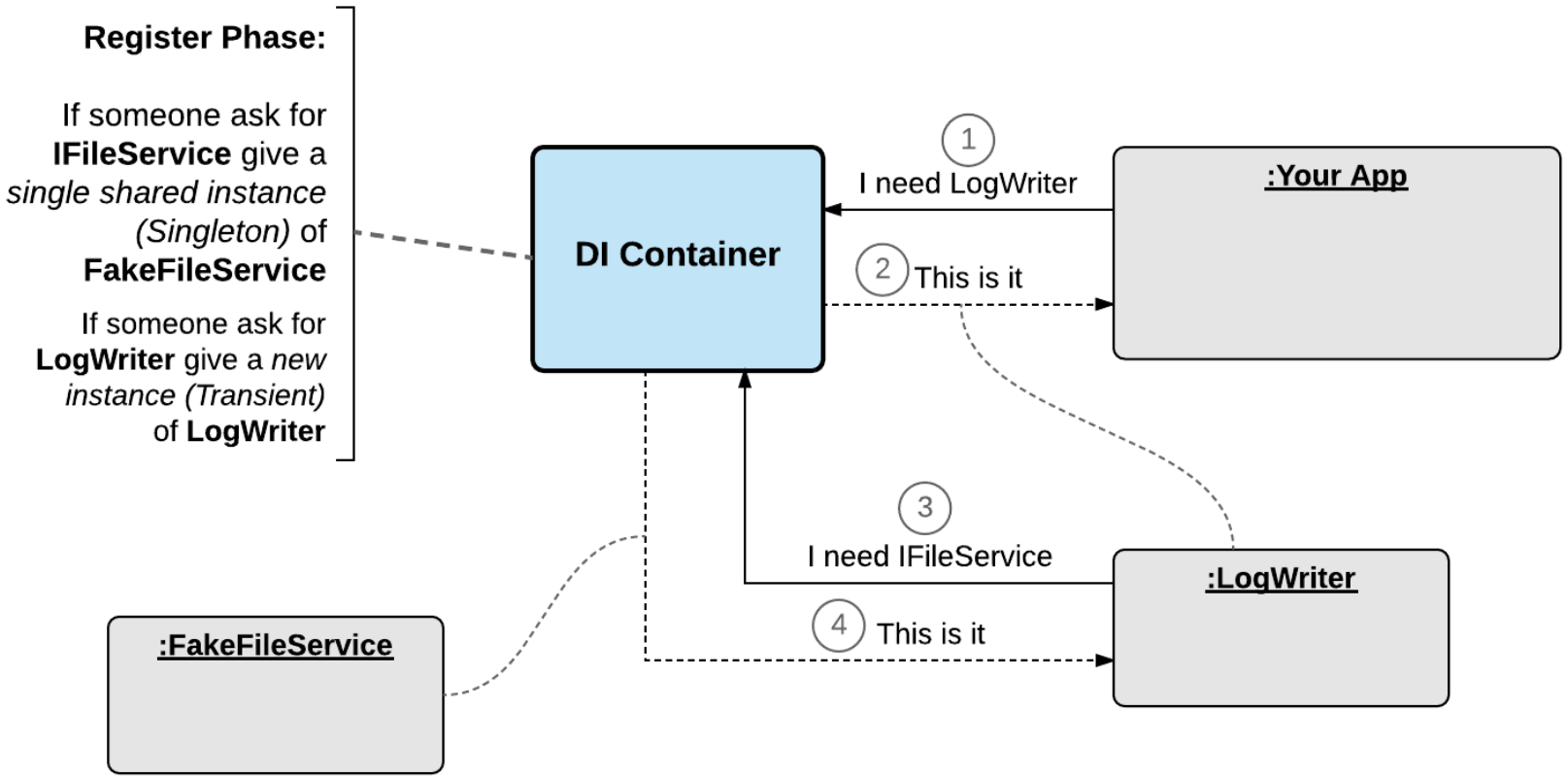



- What are tasks that not the LogWriter's jobs.
- Can LogWriter do its job on its own self?

# Dependency Injection (3)



# Dependency Injection (4)



# ConfigureServices()

- `services.AddScoped<IService, IImplementation>`
- `services.AddSingleton<IService, IImplementation>`
- `services.AddTransient<IService, IImplementation>`

# Configure()

- Pipeline and Middleware

# IServiceCollection

```
public void ConfigureServices(IServiceCollection services) {
```

# IApplicationBuilder

```
public void Configure(IApplicationBuilder app,
                      IHostingEnvironment env,
                      ILoggerFactory loggerFactory) {
```

# IHostingEnvironment

```
public class Startup {

    public Startup(IHostingEnvironment env) {
```

```
    public void Configure(IApplicationBuilder app,
                          IHostingEnvironment env,
                          ILoggerFactory loggerFactory) {
```

- ApplicationName
- ContentRootPath
- EnvironmentName
- WebRootPath

# ILoggerFactory

```
public void Configure(IApplicationBuilder app,
                      IHostingEnvironment env,
                      ILoggerFactory loggerFactory) {
```

- AddConsole
- AddDebug

# IConfigurationRoot

```
public IConfigurationRoot Configuration { get; }
```

- implements IConfiguration
- string this[string key]

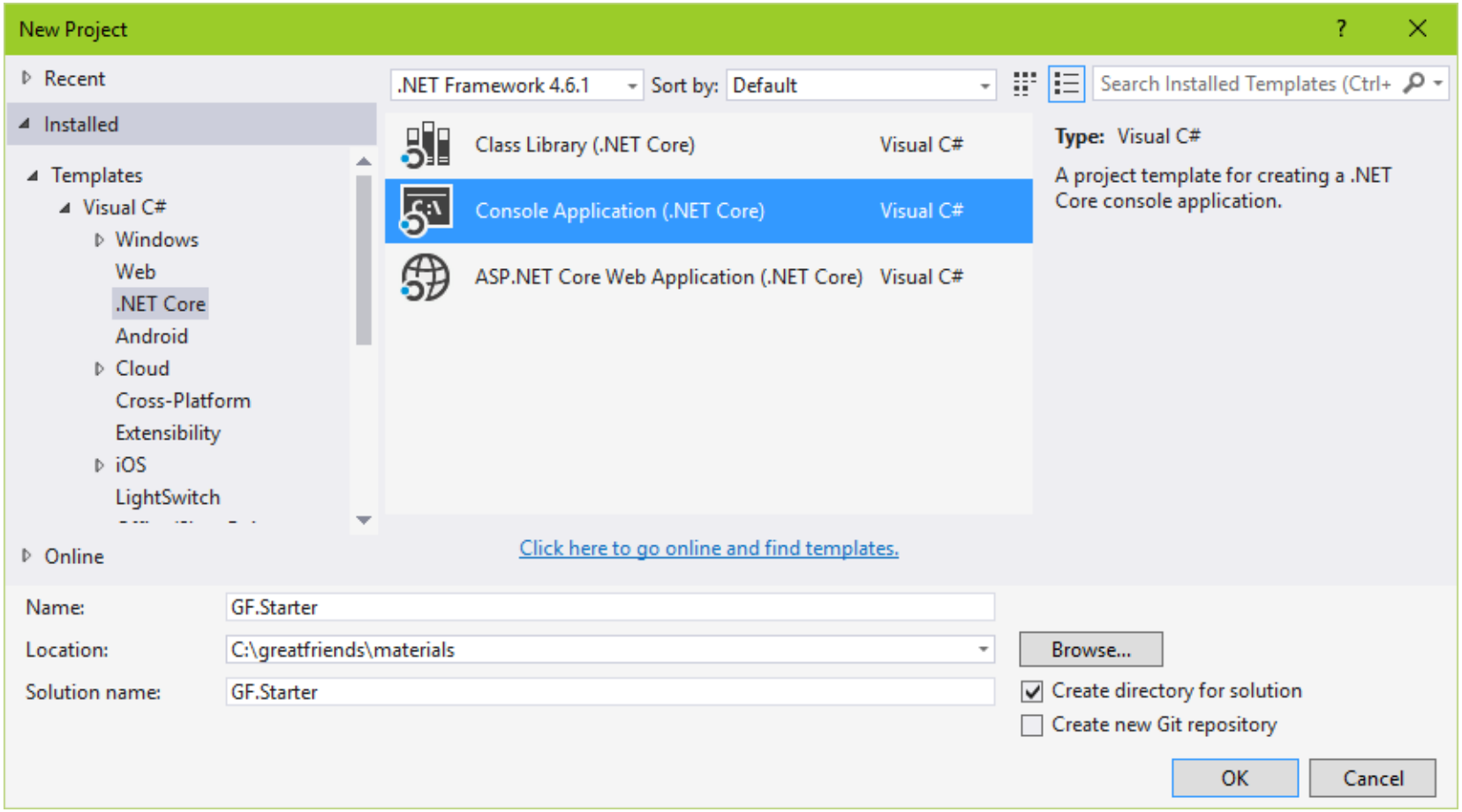
# Learn from scratch

Creating and evolving .NET Core project

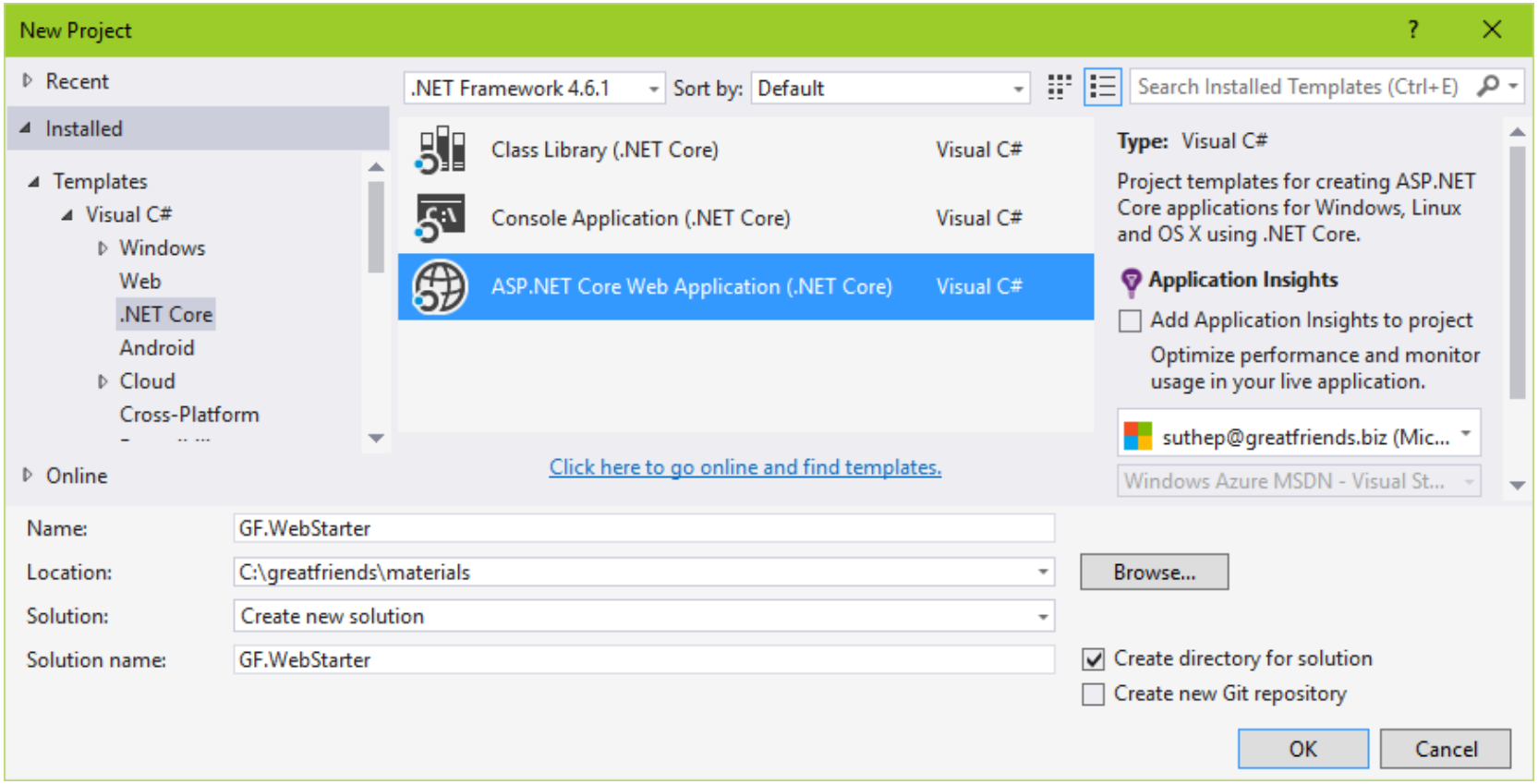
# Basic Console App

- Learn basic usage of dotnet CLI
- Create a console application
- Create a class library
- Create a xunit unit testing project

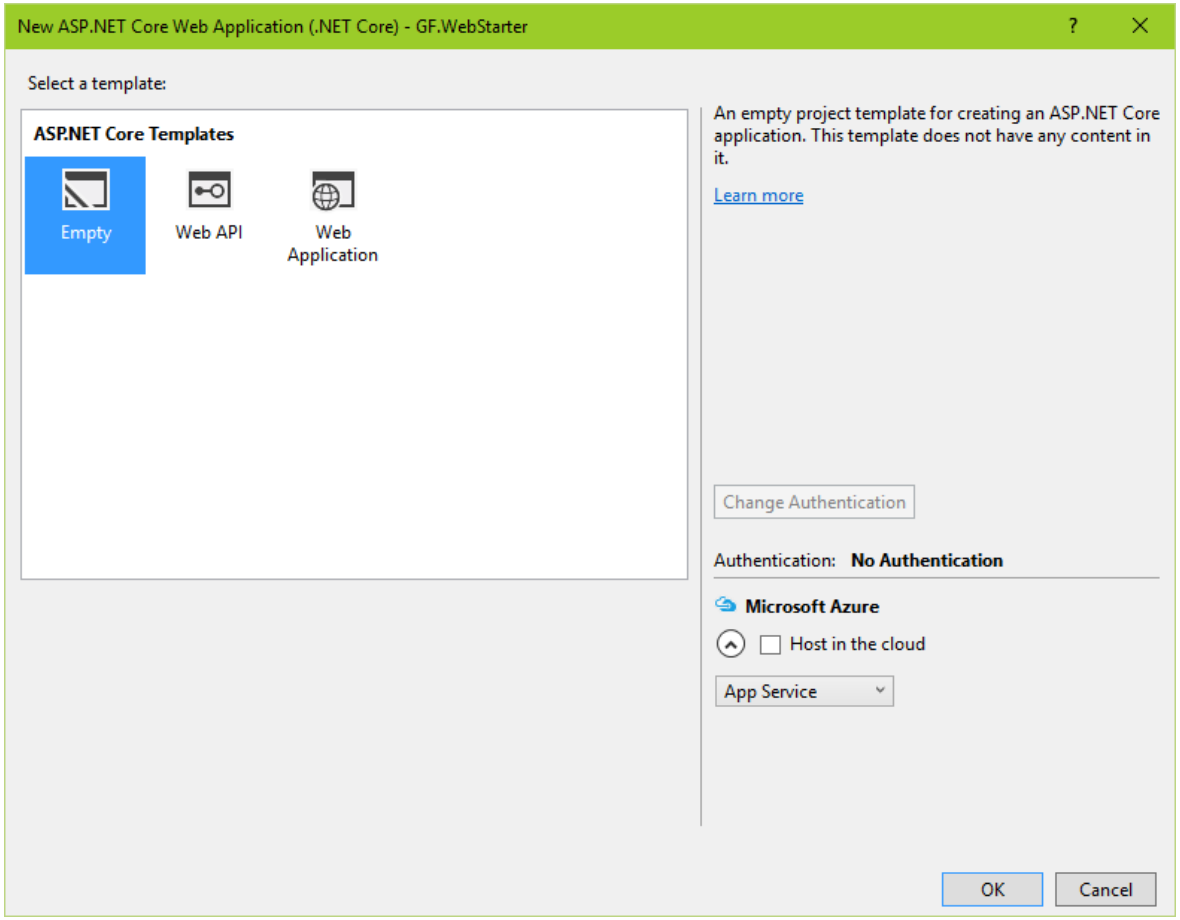
# Creating a new project from Visual Studio



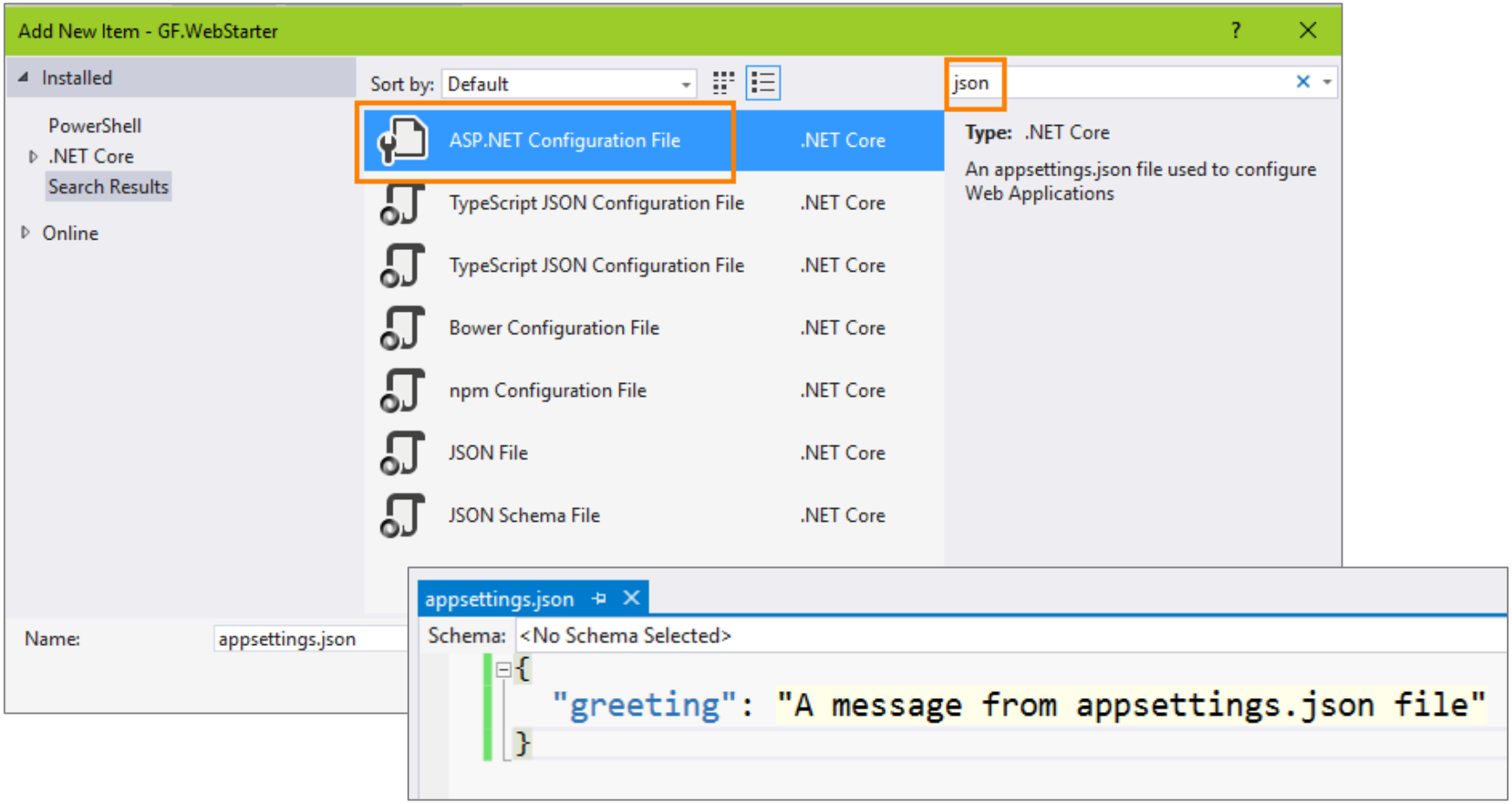
# ASP.NET Core New Project



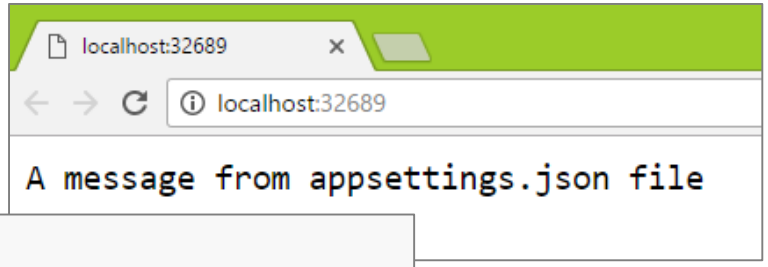
# Template


New ASP.NET Core Web Application (.NET Core) - GF.WebStarter
Select a template:
ASP.NET Core Templates
Empty
Web API
Web Application
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.
Learn more
Change Authentication
Authentication: No Authentication
Microsoft Azure
Host in the cloud
App Service
OK
Cancel

# Add appsettings.json file



# Startup.cs




```
public class Startup {

  public Startup(IHostingEnvironment environment) {
    var builder = new ConfigurationBuilder()
      .SetBasePath(environment.ContentRootPath)
      .AddJsonFile("appsettings.json");

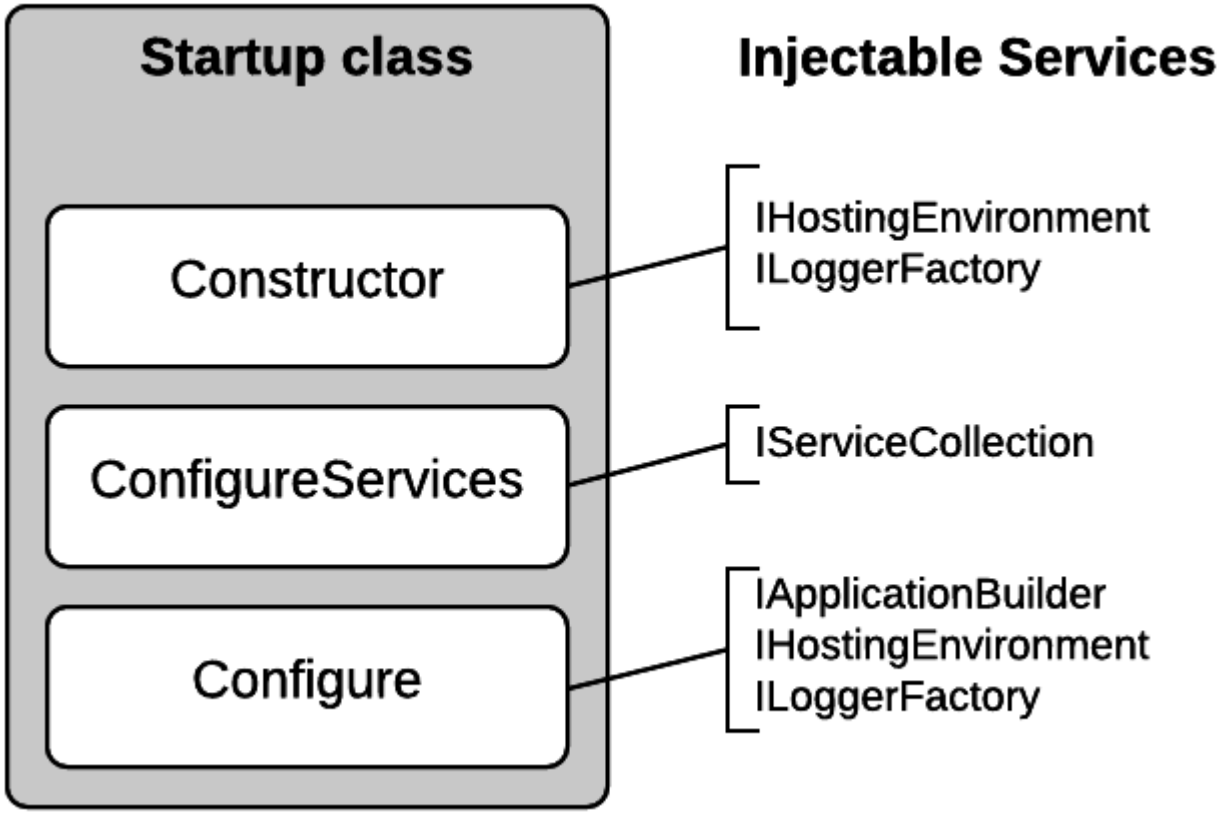
    Configuration = builder.Build();
  }

  public IConfiguration Configuration { get; set; }
```

```
public void Configure(IApplicationBuilder app,
                      IHostingEnvironment env,
                      ILoggerFactory loggerFactory) {
  // ...

  app.Run(async (context) => {
    var greeting = Configuration["greeting"];
    await context.Response.WriteAsync(greeting);
  });
```

# Available Services in Startup class



# Create our service

```
namespace GF.WebStarter {

  public interface IGreeter {
    string GetGreeting();
  }

  public class Greeter : IGreeter {
    public string GetGreeting() {
      return "message from Greeting service.";
    }
  }

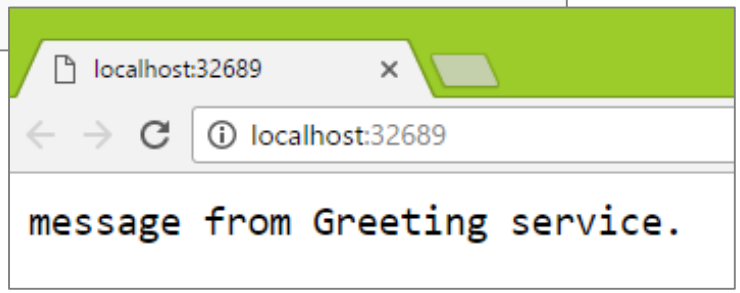
}
```

# Inject and use service

```
public void ConfigureServices(IServiceCollection services) {
  services.AddSingleton<IGreeter, Greeter>();
}

public void Configure(IApplicationBuilder app,
                      IHostingEnvironment env,
                      ILoggerFactory loggerFactory,
                      IGreeter greeter) {
  // ...

  app.Run(async (context) => {
    var greeting = greeter.GetGreeting();
    await context.Response.WriteAsync(greeting);
  });
```



# When service requires a service

```
public class Greeter : IGreeter {

  private IConfiguration _config;

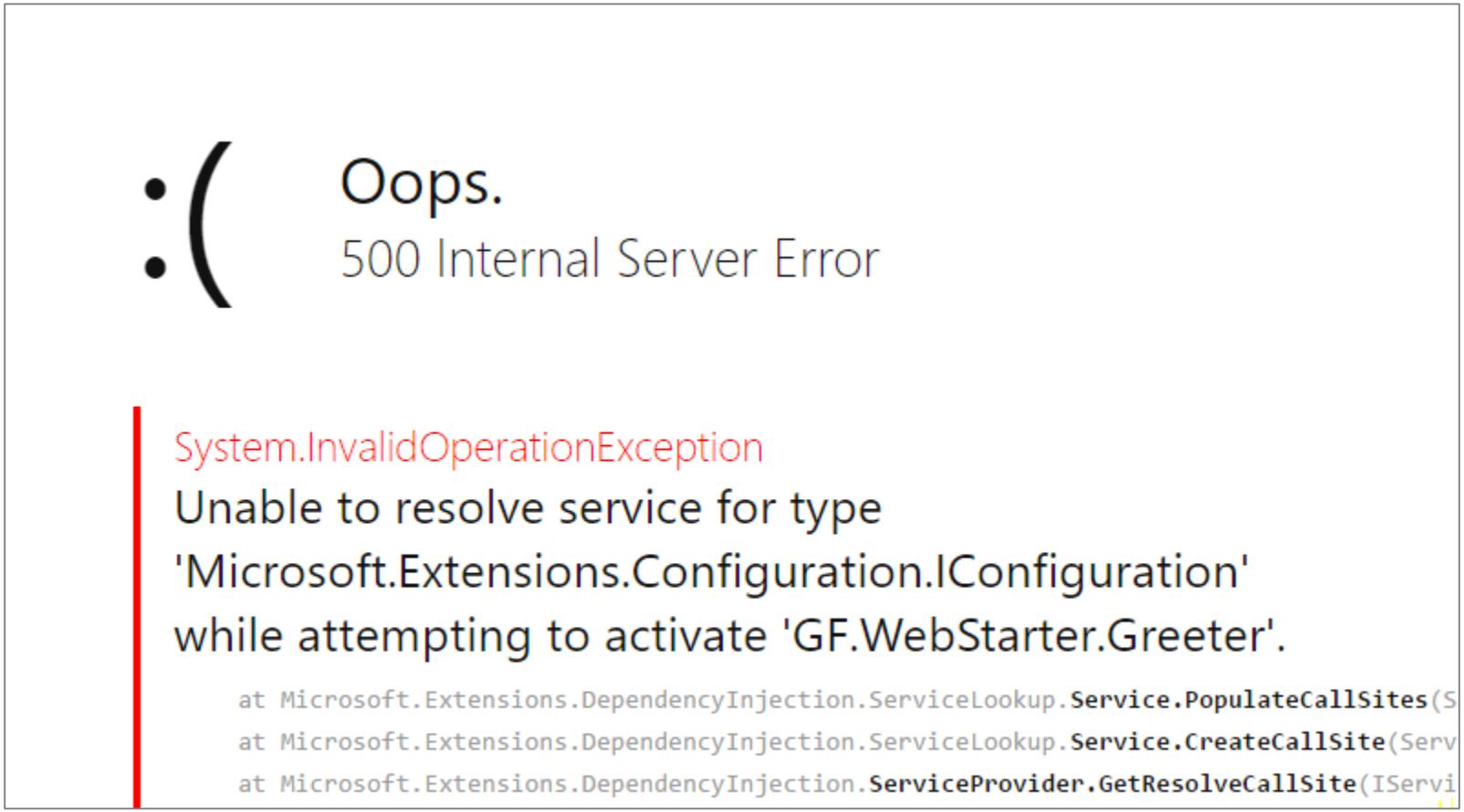
  public Greeter(IConfiguration config) {

    _config = config;

  }

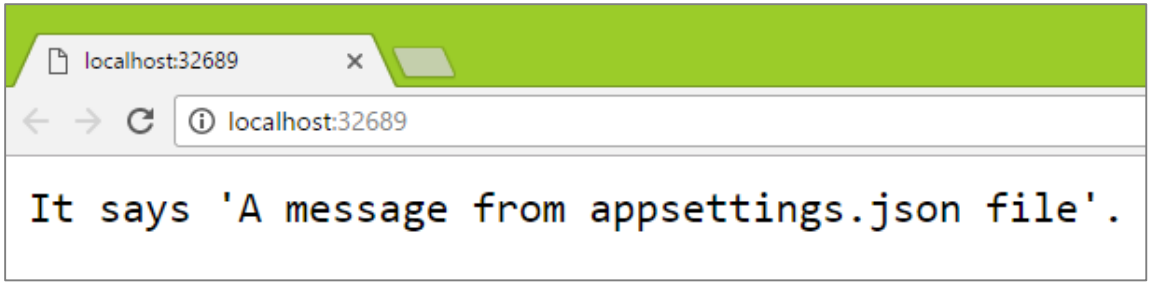
  public string GetGreeting() {
    return $"It says '{_config["greeting"]}'.";
  }
}
```

# Unable to resolve service



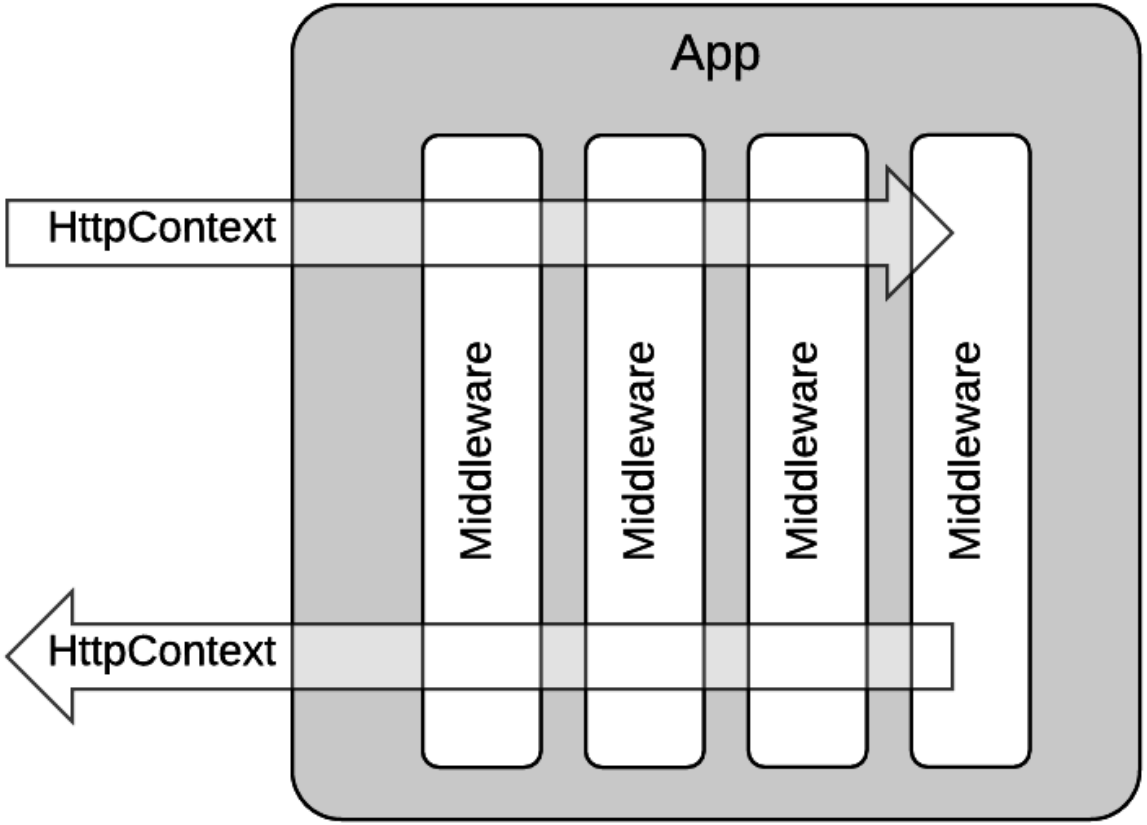
# Add to the DI container

```
public void ConfigureServices(IServiceCollection services) {
  services.AddSingleton(provider => Configuration);
  services.AddSingleton<IGreeter, Greeter>();
}
```



# Middleware and pipeline

# Application Pipeline



# What is Middleware

Middleware are software components that are assembled into an application pipeline to handle requests and responses. Each component chooses whether to pass the request on to the next component in the pipeline, and can perform certain actions before and after the next component is invoked in the pipeline. Request delegates are used to build the request pipeline. The request delegates handle each HTTP request.

Request delegates are configured using Run, Map, and Use extension methods on the IApplicationBuilder type that is passed into the `Configure` method in the `Startup` class. An individual request delegate can be specified in-line as an anonymous method, or it can be defined in a reusable class. These reusable classes are *middleware*, or *middleware components*. Each middleware component in the request pipeline is responsible for invoking the next component in the pipeline, or short-circuiting the chain if appropriate.

# The RequestDelegate

```
//
// Summary:
//     A function that can process an HTTP request.
//
// Parameters:
//   context:
//     The Microsoft.AspNetCore.Http.HttpContext for the request.
//
// Returns:
//     A task that represents the completion of request processing.
public delegate Task RequestDelegate(HttpContext context);
```

# Creating Sample Middleware

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;

public class RequestLoggerMiddleware {
  private readonly RequestDelegate _next;
  private readonly ILogger _logger;

  public RequestLoggerMiddleware(RequestDelegate next,
                                 ILoggerFactory loggerFactory) {
    _next = next;
    _logger = loggerFactory.CreateLogger<RequestLoggerMiddleware>();
  }

  public async Task Invoke(HttpContext context) {
    _logger.LogInformation("Handling request: " + context.Request.Path);
    await _next.Invoke(context);
    _logger.LogInformation("Finished handling request.");
  }
} « class RequestLoggerMiddleware
```

# Add middleware to the pipeline

```
public void Configure(IApplicationBuilder app,
                      IHostingEnvironment env,
                      ILoggerFactory loggerFactory) {

  // ...

  app.UseMiddleware<RequestLoggerMiddleware>();


  app.UseMvc(routes => {
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
  });
} « void Configure(IApplicationBuilder, IHostingEnvironment, ILoggerFact
```

# Create Extension Method

```
using Microsoft.AspNetCore.Builder;

public static class RequestLoggerExtensions {

  public static IApplicationBuilder UseRequestLogger(this IApplicationBuilder builder) {
    return builder.UseMiddleware<RequestLoggerMiddleware>();
  }

}
```

```
public void Configure(IApplicationBuilder app,
                      IHostingEnvironment env,
                      ILoggerFactory loggerFactory) {

  // ...

  app.UseRequestLogger();
```

# Blank Middleware template

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;

public class BlankMiddleware {

  private readonly RequestDelegate _next;

  public BlankMiddleware(RequestDelegate next) {
    _next = next;
  }

  public async Task Invoke(HttpContext context) {
    // do something
    await _next.Invoke(context);
    // do other thing
  }
} « class BlankMiddleware
```

# Routing

# Routing

Routing uses routes (implementations of `IRouter`) to:

- map incoming requests to route handlers
- generate URLs used in responses

```
app.UseMvc(routes => {
  routes.MapRoute(
      name: "default",
      template: "{controller=Home}/{action=Index}/{id?}");
});
```

# DataTokens

```
app.UseMvc(routes => {
  routes.MapRoute(
      name: "default",
      template: "{controller=Home}/{action=Index}/{id?}",
      defaults: new { },
      constraints: new { },
      dataTokens: new { language = "default" });
});
```

```
public IActionResult Index() {

  var lang = RouteData.DataTokens["language"] as string;
```

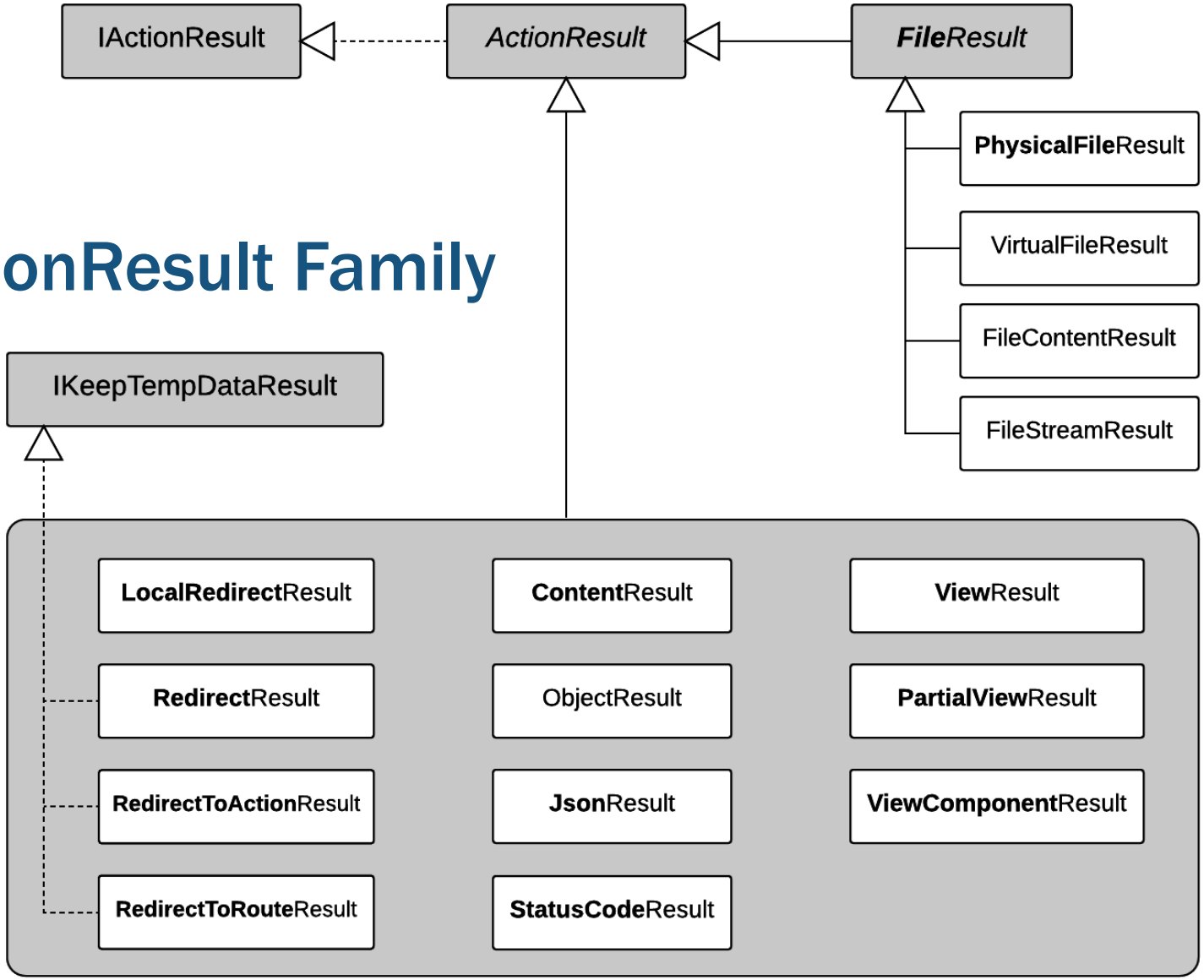
# ActionResult

# IActionResult

Routing uses routes (implementations of `IRouter`) to:

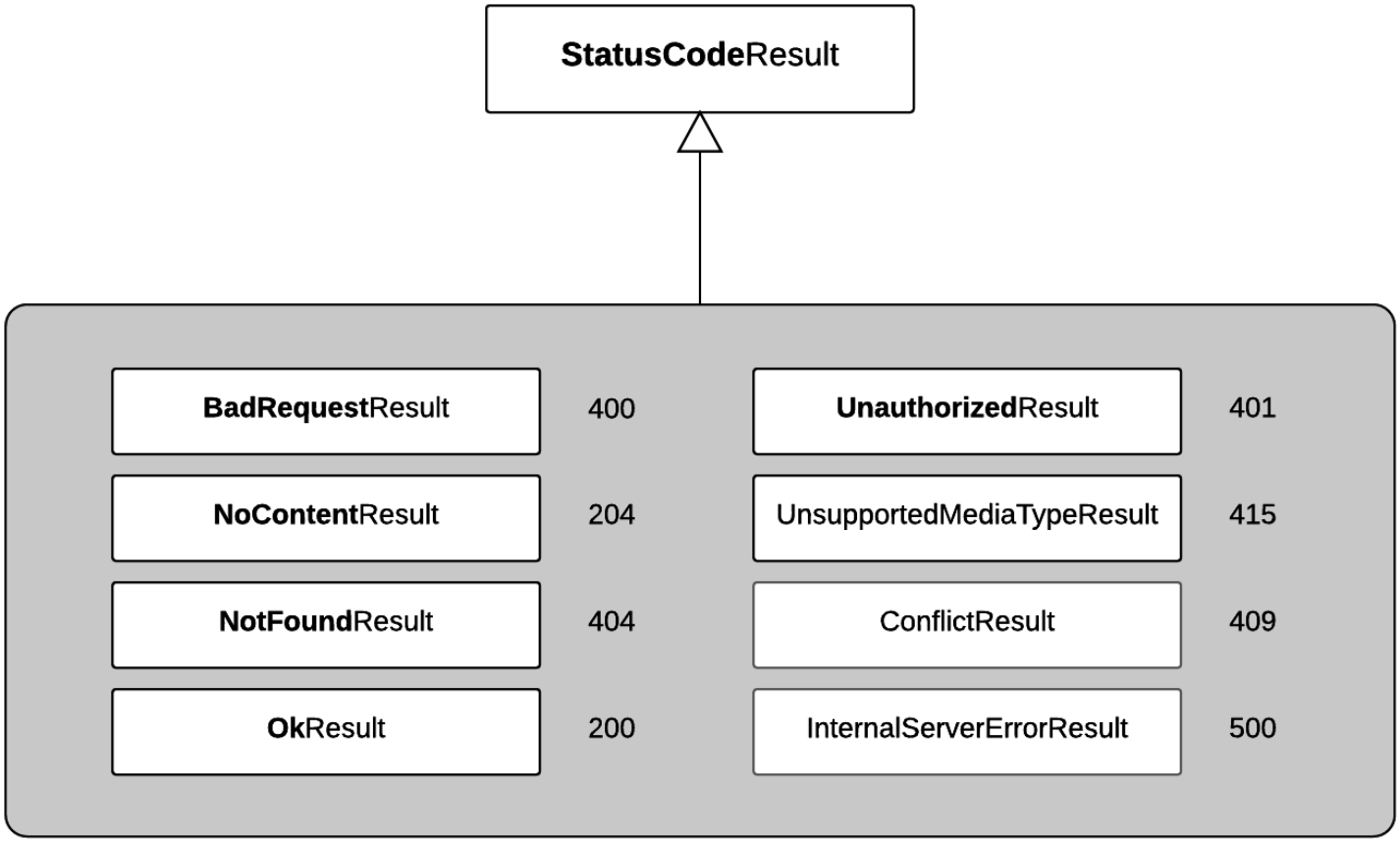
- map incoming requests to route handlers
- generate URLs used in responses

# IActionResult Family




**Bold** text is a helper function name. For example: `return View();`

# StatusCodeResult




**Bold** text is a helper function name. For example: `return OK();`
but you have to use `return new UnsupportedMediaType();`

# HTTP Status Code

- 1xx Information
- 2xx Success
- 3xx Redirection
- 4xx Client Error
- 5xx Server Error

| | |
|---|---|
| 200 | OK |
| 201 | Created (after insertion) |
| 202 | Accepted |
| 204 | No Content (after deletion) |
| 301 | Moved Permanently |
| 302 | Found (Move Temporarily) |
| 303 | See Other |
| 304 | Not Modified (uses cache) |

# HTTP Status Code

- 400 Bad Request
- 401 Unauthorized (really means not authenticated)
- 403 Forbidden (really means not authorized)
- 404 Not Found
- 405 Method Not Allowed
  (ex. Uses GET for POST only)
- 408 Request Timeout
- 410 Gone (Search Engine removal)
- 418 I'm a Teapot

500 Internal Server Error

501 Not Implemented

503 Service Unavailable

# Model Binding (1)
# Sources

1. Form values
2. Route values
3. Query strings

```
public ActionResult Create()
{
    var auction = new Auction() {
        Title = Request["title"],
        CurrentPrice = Decimal.Parse(Request["currentPrice"]),
        StartTime = DateTime.Parse(Request["startTime"]),
        EndTime = DateTime.Parse(Request["endTime"]),
    };
```

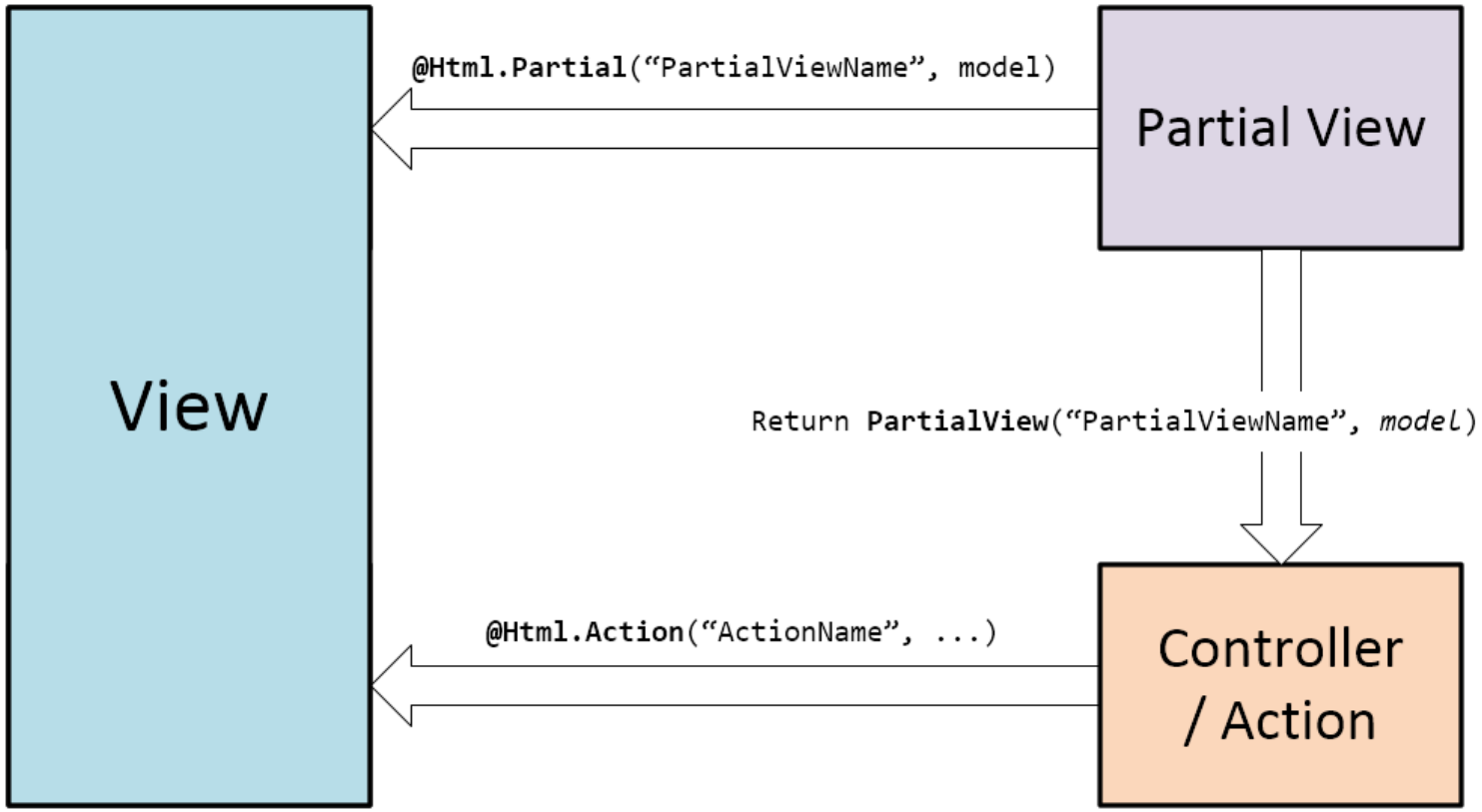
```
public ActionResult Create(
        string title, decimal currentPrice,
        DateTime startTime, DateTime endTime
    )
{
    var auction = new Auction() {
        Title = title,
        CurrentPrice = currentPrice,
        StartTime = startTime,
        EndTime = endTime,
    };
    // ...
}
```

```
public ActionResult Create(Auction auction)
{
    // ...
}
```

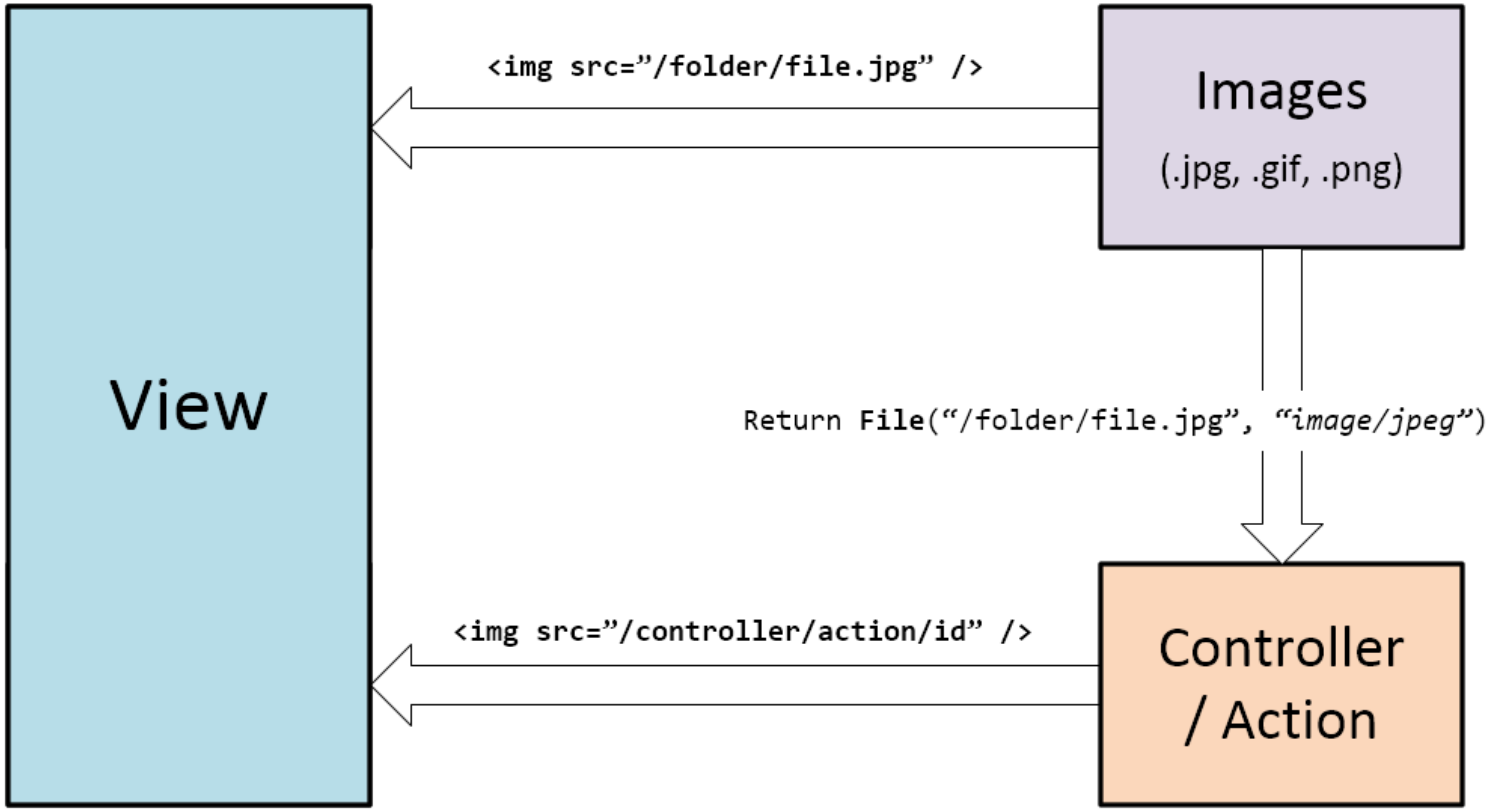
# Model Binding (2)
# Binding Behaviors

- `[BindRequired]`
- `[BindNever]`
- `[FromHeader]`, `[FromQuery]`
  `[FromRoute]`, `[FromForm]`
- `[FromService]`
- `[FromBody]` // can has only one parameter per action

# Partial View



# FileResult class



# Sending data across requests

Session object

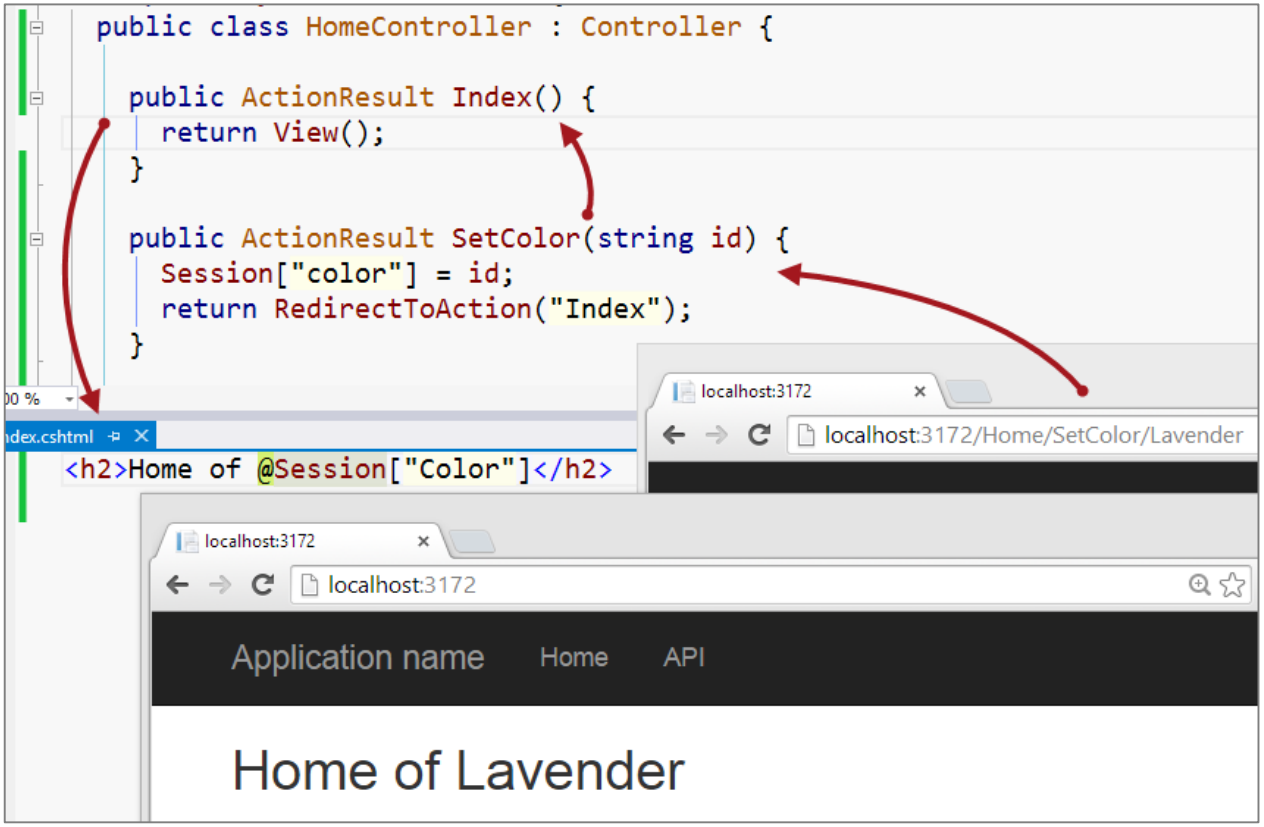
Application object

URL

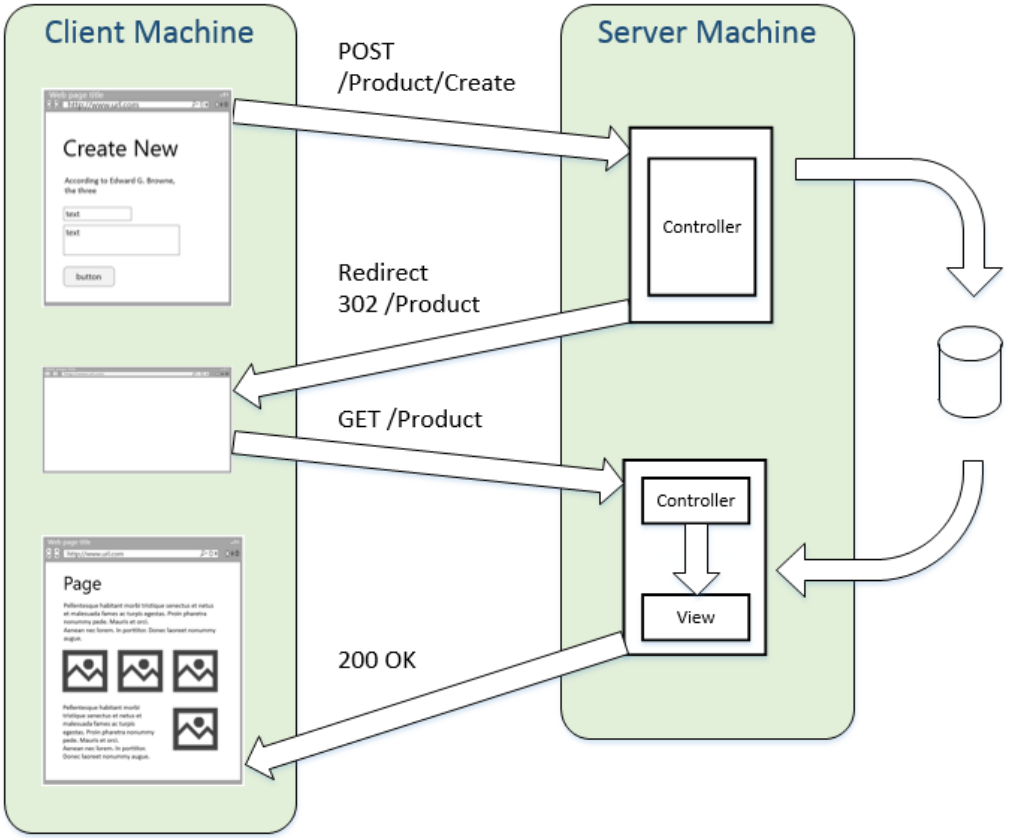
Query String

Form Data

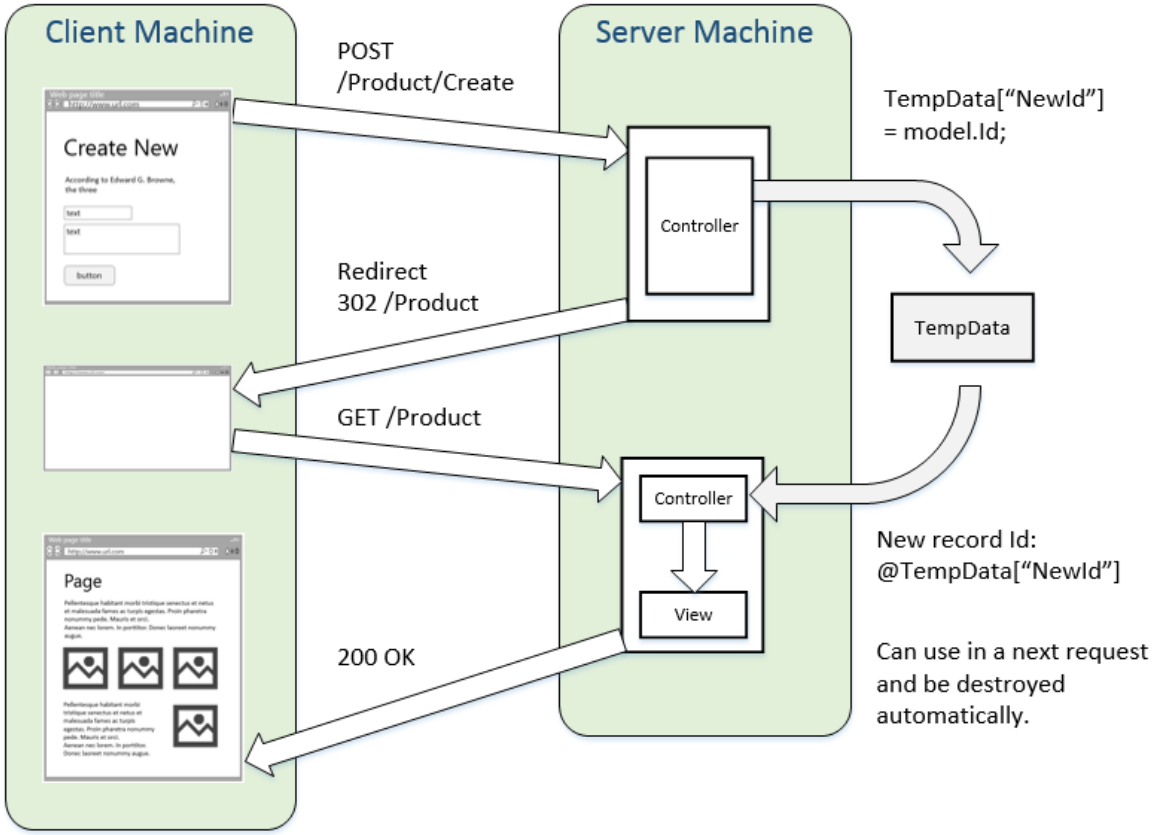
TempData
(see next 2 pages)



# Post-Redirect-Get Pattern



# TempData


Client Machine
Server Machine
POST
/Product/Create
Create New
Controller
TempData["NewId"]
= model.Id;
Redirect
302 /Product
TempData
GET /Product
Controller
View
Page
New record Id:
@TempData["NewId"]
200 OK
Can use in a next request
and be destroyed
automatically.

# Web API

# HTTP Method for RESTful Services

| HTTP Verb | CRUD | Entire Collection (e.g. /customers) | Specific Item (e.g. /customers/{id}) |
|---|---|---|---|
| POST | Create | 201 (Created), 'Location' header with link to /customers/{id} containing new ID. | 404 (Not Found), 409 (Conflict) if resource already exists.. |
| GET | Read | 200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists. | 200 (OK), single customer. 404 (Not Found), if ID not found or invalid. |
| PUT | Update/Replace | 404 (Not Found), unless you want to update/replace every resource in the entire collection. | 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. |
| PATCH | Update/Modify | 404 (Not Found), unless you want to modify the collection itself. | 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. |
| DELETE | Delete | 404 (Not Found), unless you want to delete the whole collection—not often desirable. | 200 (OK). 404 (Not Found), if ID not found or invalid. |

# Default Web API Template

```
[Route("api/[controller]")]
public class ValuesController : Controller {

  // GET: api/values
  [HttpGet]
  public IEnumerable<string> Get() {
    return new string[] { "value1", "value2" };
  }

  // GET api/values/5
  [HttpGet("{id}")]
  public string Get(int id) { return "value"; }

  // POST api/values
  [HttpPost]
  public void Post([FromBody]string value) { }

  // PUT api/values/5
  [HttpPut("{id}")]
  public void Put(int id, [FromBody]string value) { }

  // DELETE api/values/5
  [HttpDelete("{id}")]
  public void Delete(int id) { }
} « class ValuesController
```

# Model Validation

# By Conventions

- Required:
  **Value Type** or **[Required]** on Reference Type
  ex. int, [Required] on string property.
- Optional:
  **Reference Type** or Nullable Value Type
  ex. int?, Product.

```
public class Box {

    public int Id { get; set; }
    public string Color { get; set; }
    public DateTime CreatedDate { get; set; }
    public bool IsUsed { get; set; }
    public byte[] Picture { get; set; }
    public Person LeadDesigner { get; set; }
    public ICollection<Person> Designers { get; set; }

} « class Box
```

# Validation Attributes

- [Required]
- [StringLength(length)]
- [Range(min, max)]
- [DataType(DataType.Date)]
- [Compare]
- [EmailAddress]
- [Phone]
- [RegularExpression]
- [Url]

# Creating New ValidationAttribute

```
public class ClassicMovieAttribute : ValidationAttribute, IClientModelValidator
{
    private int _year;

    public ClassicMovieAttribute(int Year)
    {
        _year = Year;
    }

    protected override ValidationResult IsValid(object value, ValidationContext validationContext)
    {
        Movie movie = (Movie)validationContext.ObjectInstance;

        if (movie.Genre == Genre.Classic && movie.ReleaseDate.Year > _year)
        {
            return new ValidationResult(GetErrorMessage());
        }

        return ValidationResult.Success;
    }
```

# Implementing IValidatableObject interface

```
public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
{
    if (Genre == Genre.Classic && ReleaseDate.Year > _classicYear)
    {
        yield return new ValidationResult(
            "Classic movies must have a release year earlier than " + _classicYear,
            new[] { "ReleaseDate" });
    }
}
```

# Server-Side Validation

- ModelState
  - IsValid
  - IsValidField(key)
  - AddModelError(key, errorMessage)
  - Remove(key)

# Client-side validation using jQuery Validation plugin

```
@section Scripts {
  @await Html.PartialAsync("_ValidationScriptsPartial")
}
```

```
<div class="form-group">
    <label asp-for="ReleaseDate" class="col-md-2 control-label"></label>
    <div class="col-md-10">
        <input asp-for="ReleaseDate" class="form-control" />
        <span asp-validation-for="ReleaseDate" class="text-danger"></span>
    </div>
</div>
```

# Remote Validation

```
public class User
{
    [Remote(action: "VerifyEmail", controller: "Users")]
    public string Email { get; set; }
}
```

```
[AcceptVerbs("Get", "Post")]
public IActionResult VerifyEmail(string email)
{
    if (!_userRepository.VerifyEmail(email))
    {
        return Json(data: $"Email {email} is already in use.");
    }

    return Json(data: true);
}
```

# Razor syntax

# Razor (.cshtml)
# view = code + markup

```
<div>
  markup
</div>

markup
```

```
@{
  // code block
  var condition = true;
}

@if (condition) {
  // code block
}
else {
  // code block
}
```

```
@while (condition) {
  // code block
}
@for (int i = 0; i < 10; i++) {
  // code block
}
@do {
  // code block
} while (condition);
```

# Flow of markup and code

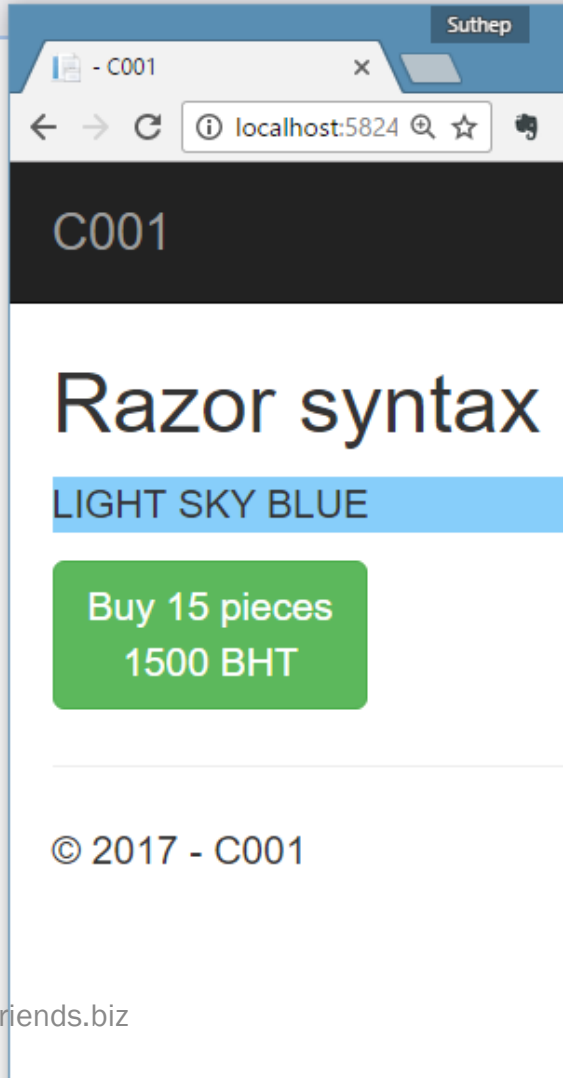
About.cshtml

```
@{
  string s = "Light Sky Blue";
  string color = s.Replace(" ", "").ToLower();
  int qty = 15;
  decimal unitPrice = 100;
}

<h2>Razor syntax</h2>

<p style="background-color: @color">
  @s.ToUpper()
</p>

@if (qty == 0) {
  <button class="btn btn-default" disabled>
    Buy some pieces!
  </button>
} else {
  <button class="btn btn-success">
    Buy @qty pieces<br/>
    @(qty * unitPrice) BHT
  </button>
}
```



# Expression
# . ( ) [ ] ?. await

```
@p.Name
@p.Name.ToString()
@p.Name.ToString()[6 - 2]
@p.Name.Replace("ASPX", "Razor")[i++]
```

```
@1 + 1 ==> @
@p++ ==> @p
@p    .   Name ==> @p
@p.Name.Length - 1 ==> @p.Name.Length
```

```
@(1 + 1)
@(p++)
@(p    .   Name)
@(p.Name.Length - 1)
```

```
@{
    int x = 5;
}

@x

@{
    @x
}

@{
    @: @x
}
```

# ?. ??

About.cshtml

```
@{
    Box b1 = null;
    Box b2 = new Box { Weight = 3.5 };
}

<h2>Razor syntax</h2>

<div>
    @b1?.Weight grams
</div>
<div>
    @(b1?.Weight ?? 0) grams
</div>

@functions {
    class Box {
        public double Weight { get; set; }
    }
}
```

# Email address and @@

```
@{
  var company = "ms";
}

mail to inbox@company.com
<hr/>
mail to inbox@(company).com
<hr/>
mail to inbox@@@(company).com
```

# Text in Code

```
@if(IsPost) {
    // This line has all content between matched <p> tags.
    <p>Hello, the time is @DateTime.Now and this page is a postback!</p>
} else {
    // All content between matched tags, followed by server code.
    <p>Hello <em>stranger</em>, today is: <br /> </p>  @DateTime.Now
}
```

```
@if(IsPost) {
    // Plain text followed by an unmatched HTML tag and server code.
    @: The time is: <br /> @DateTime.Now
    // Server code and then plain text, matched tags, and more text.
    @DateTime.Now @:is the <em>current</em> time.
}
```

# Text in Code (2)

```
@if(IsPost) {
    // Repeat the previous example, but use <text> tags.
    <text>
    The time is: <br /> @DateTime.Now
    @DateTime.Now is the <em>current</em> time.
    </text>
}

@{
    var minTemp = 75;
    <text>It is the month of @DateTime.Now.ToString("MMMM"), and
    it's a <em>great</em> day! <br /><p>You can go swimming if it's at
    least @minTemp degrees. </p></text>
}
```

# Comment

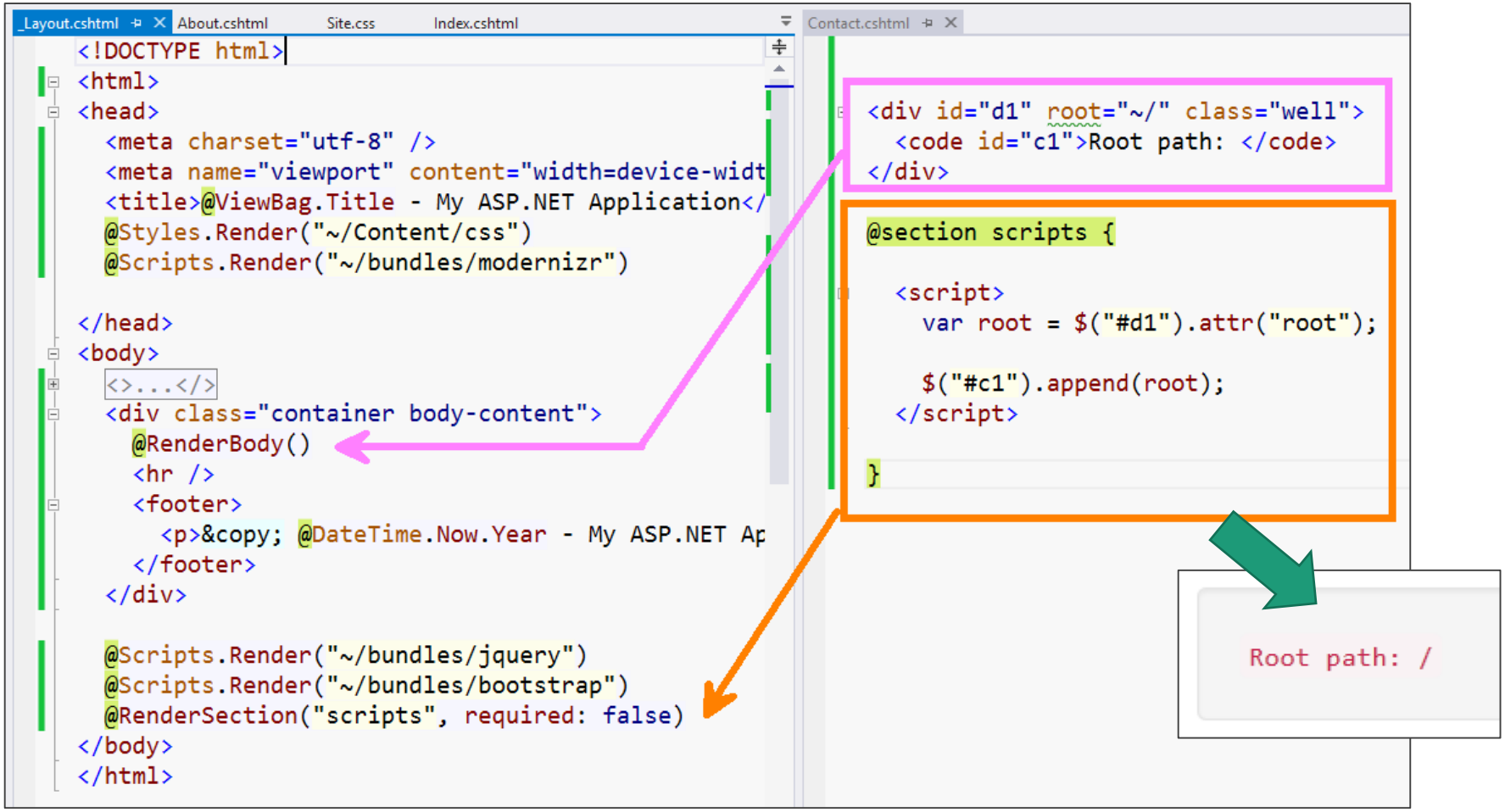
```
@*  A one-line code comment. *@

@*
    This is a multiline code comment.
    It can continue for any number of lines.
*@
```

```
@{
    @* This is a comment. *@
    @* var theVar = 17;  *@
}

@{
    // This is a comment.
    var myVar = 17;
    /* This is a multi-line comment
    that uses C# commenting syntax. */
}
```

# @section



# tilde slash

```
<script src="@Url.Content("~/Scripts/myscript.js")"></script>
```

```
<script src="~/Scripts/myscript.js"></script>
```

# Conditional Attributes

```
@{
  string c1 = null;
  string c2 = "box";
}

<div id="d1" class="@c1">
  div 1
</div>
<div id="d2" class="@c1 @c2">
  div 2
</div>
```

```
46
47 <div id="d1">
48   div 1
49 </div>
50 <div id="d2" class="box">
51   div 2
52 </div>
53
```

# Condition Attributes (bool)

```
@{
  bool b1 = true;
  bool b2 = false;
}

<div class="well">

  <label>
    <input type="checkbox" checked="@b1" />
    Checkbox 1
  </label>

  <label>
    <input type="checkbox" checked="@b2" />
    Checkbox 2
  </label>

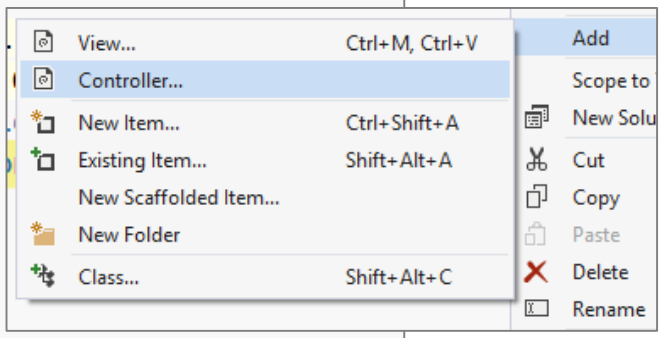
</div>
```

```
46
47 <div class="well">
48
49   <label>
50     <input type="checkbox" checked="checked" />
51     Checkbox 1
52   </label>
53
54   <label>
55     <input type="checkbox" />
56     Checkbox 2
57   </label>
58
59 </div>
60
```

# Code Generation Tools

```
    "Microsoft.EntityFrameworkCore.Design": "1.0.0-preview2-final",
    "Microsoft.VisualStudio.Web.CodeGeneration.Tools": {
      "version": "1.0.0-preview2-final",
      "type": "build"
    },
    "Microsoft.VisualStudio.Web.CodeGenerators.Mvc": {
      "version": "1.0.0-preview2-final",
      "type": "build"
    }
  },

  "tools": {
    "BundlerMinifier.Core": "2.0.238",
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
    "Microsoft.AspNetCore.Razor.Tools": "1.0.0-preview2-final",
    "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.0.0-preview2-final",
    "Microsoft.VisualStudio.Web.CodeGeneration.Tools": {
      "version": "1.0.0-preview2-final",
      "imports": [
        "portable-net45+win8"
      ]
    }
  },
```



# Display and Editor Templates

# Display and Editor Template

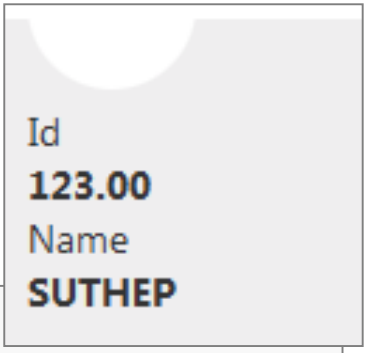
Uses with strongly-typed View only:

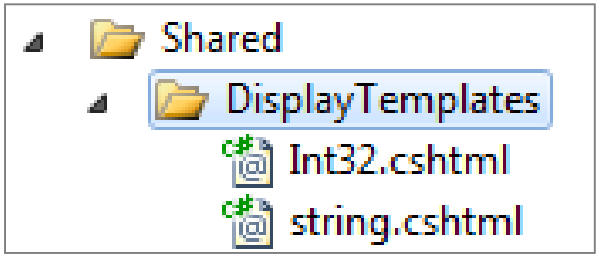
- @Html.Display("Property")
  @Html.Editor("Property")
- @Html.DisplayFor(m => m.Property)
  @Html.EditorFor(m => m.Property)

  @Html.DisplayForModel()
  @Html.EditorForModel()

```
[DisplayFormat(DataFormatString = "{0:N2}",
               ApplyFormatInEditMode = true)]
public double Weight { get; set; }
```

# Display Templates
# DisplayForModel( )




```
public class Person {
  public int Id { get; set; }
  public string Name { get; set; }
}
```

```
public ActionResult Index() {
  var p = new Person();
  p.Id = 123;
  p.Name = "Suthep";
  return View(p);
}
```

100 %

Index.cshtml

Client Objects & Events

```
@using MvcApplication5.Models;
@model Person

@Html.DisplayForModel()
```

Int32.cshtml

Client Objects & Events

```
@model int

<b>@Model.ToString("n2")</b>
```

string.cshtml

Client Objects & Events

```
@model string
<b>@Model.ToUpper()</b>
```

# DisplayFor

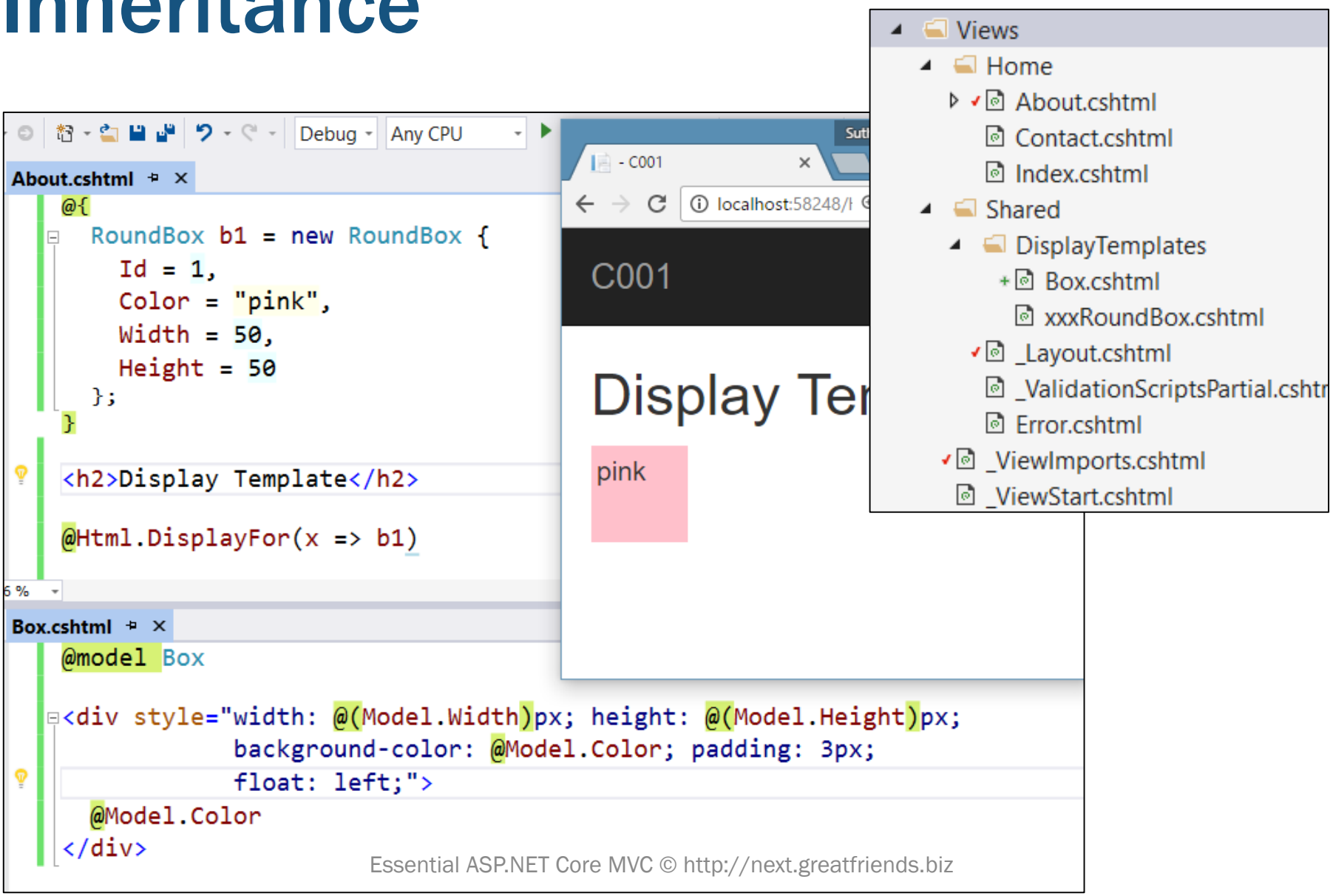
```
@using MvcApplication5.Models;
@model Person

@Html.DisplayFor(m => m.Id)
<hr />
@Html.DisplayFor(m => m.Name)
```

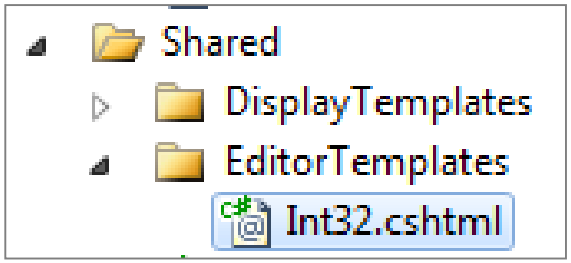
123.00

SUTHEP

# Inheritance



# Editor Templates




Int32.cshtml

Client Objects & Events

```
@model int

<input type="number" value="@Model" />
```

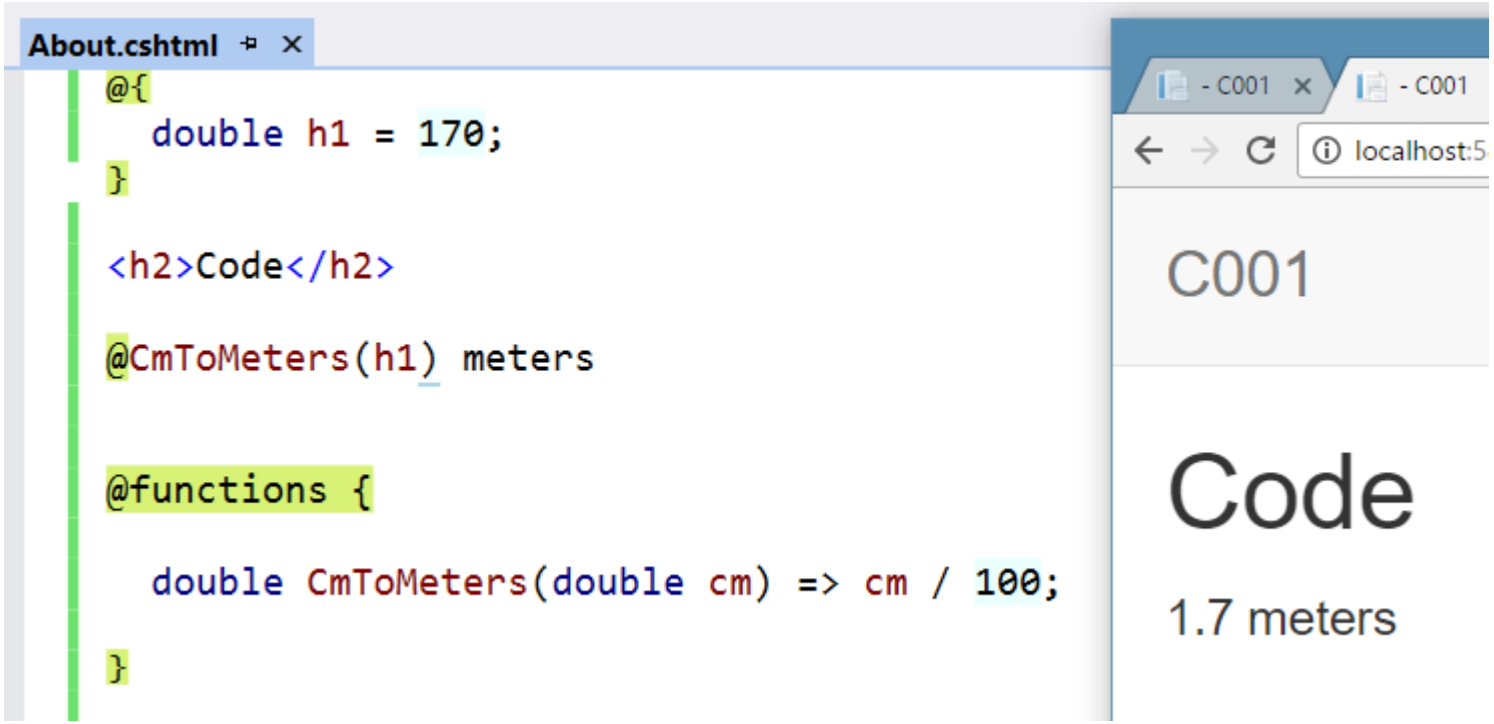
Index.cshtml

Client Objects & Events

```
@using MvcApplication5.Model
@model Person

@Html.EditorFor(m => m.Id)
```

# View Compilation



# Make some error

About.cshtml

```
@{
  double h1 = 170;
}

<h2>Code</h2>

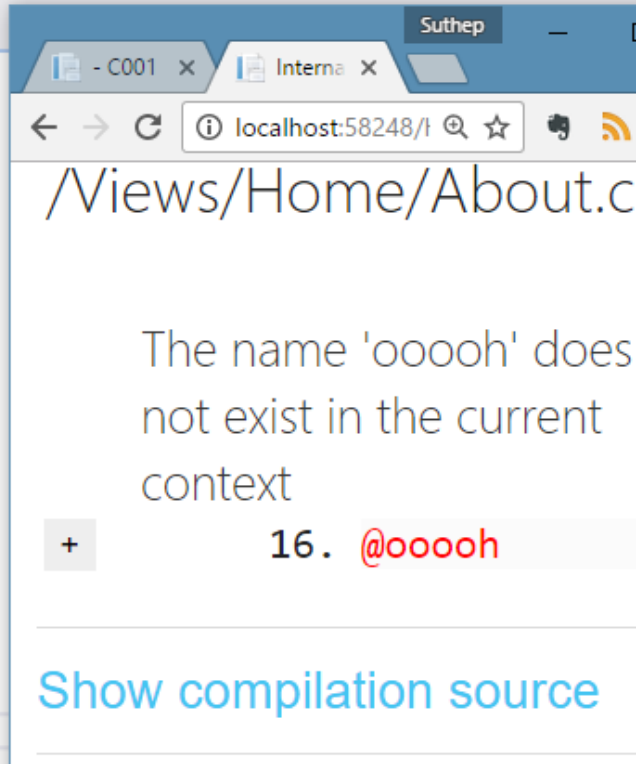
@CmToMeters(h1) meters

@functions {

  double CmToMeters(double cm) => cm / 100;

}

@ooooh
```



# Layout and directives

- _ViewImports.cshtml
- _ViewStart.cshtml
- @using
- @model
- @inject
- @RenderBody()
- @RenderSection("name", required: true/false)

# Html Helpers

- @Html.BeginForm()
- @Html.TextBoxFor()
- ...

# Built-in Tag Helpers

- Anchor
- Cache
- DistributedCache
- Environment
- FormAction
- Form
- Image
- Input
- Label
- Link
- Option
- Script
- Select
- TextArea
- ValidationMessage
- ValidationSummary

# <a>

- asp-action
- asp-controller
- asp-area
- asp-fragment
- asp-host
- asp-protocol
- asp-route

# Creating TagHelper
## <email>support</email>

```
using Microsoft.AspNetCore.Razor.TagHelpers;
using System.Threading.Tasks;

namespace AuthoringTagHelpers.TagHelpers
{
    public class EmailTagHelper : TagHelper
    {
        public override void Process(TagHelperContext context, TagHelperOutput
        {
            output.TagName = "a";    // Replaces <email> with <a> tag
        }
    }
}
```

# SetContent and SetAttributes

## <email mail-to="support"></email>

```
public class EmailTagHelper : TagHelper
{
    private const string EmailDomain = "contoso.com";

    // Can be passed via <email mail-to="..." />.
    // Pascal case gets translated into lower-kebab-case.
    public string MailTo { get; set; }

    public override void Process(TagHelperContext context, TagHelperOutp
    {
        output.TagName = "a";    // Replaces <email> with <a> tag

        var address = MailTo + "@" + EmailDomain;
        output.Attributes.SetAttribute("href", "mailto:" + address);
        output.Content.SetContent(address);
    }
}
```

# [HtmlTargetElement]

```
[HtmlTargetElement("email", TagStructure = TagStructure.WithoutEndTag)]
public class EmailVoidTagHelper : TagHelper
{
    private const string EmailDomain = "contoso.com";
    // Code removed for brevity
```

# RemoveAll, PreContent.SetHtmlContent, PostContent.SetHtmlContent

```
using Microsoft.AspNetCore.Razor.TagHelpers;

namespace AuthoringTagHelpers.TagHelpers
{
    [HtmlTargetElement(Attributes = "bold")]
    public class BoldTagHelper : TagHelper
    {
        public override void Process(TagHelperContext context, TagH
        {
            output.Attributes.RemoveAll("bold");
            output.PreContent.SetHtmlContent("<strong>");
            output.PostContent.SetHtmlContent("</strong>");
        }
    }
}
```

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

# Passing Model to a Tag Helper

```
namespace AuthoringTagHelpers.TagHelpers
{
    public class WebsiteInformationTagHelper : TagHelper
    {
        public WebsiteContext Info { get; set; }

      public override void Process(TagHelperContext context, TagHelperOutput
      {
         output.TagName = "section";
         output.Content.SetHtmlContent(
$@"<ul><li><strong>Version:</strong> {Info.Version}</li>
<li><strong>Copyright Year:</strong> {Info.CopyrightYear}</li>
<li><strong>Approved:</strong> {Info.Approved}</li>
<li><strong>Number of tags to show:</strong> {Info.TagsToShow}</li></ul>");
         output.TagMode = TagMode.StartTagAndEndTag;
      }
   }
}
```

# Passing Model

```
<website-information info="new WebsiteContext {
                                    Version = new Version(1, 3),
                                    CopyrightYear = 1638,
                                    Approved = true,
                                    TagsToShow = 131 }" />
```

# Conditional Tag Helper

```
namespace AuthoringTagHelpers.TagHelpers
{
    [HtmlTargetElement(Attributes = nameof(Condition))]
    public class ConditionTagHelper : TagHelper
    {
        public bool Condition { get; set; }

        public override void Process(TagHelperContext context, TagHelp
        {
            if (!Condition)
            {
                output.SuppressOutput();
            }
        }
    }
}
```

```
<div condition="Model.Approved">
    <p>
        This website has <strong
        Visit www.contoso.com for
    </p>
</div>
```

# Partial Views

```
@Html.Partial("_Copyright")
@await Html.PartialAsync()

@{
    Html.RenderPartial();
    await Html.RenderPartialAsync();
}
```

# Dependency Injection into Views

- **@inject** *Type name*

# View Components

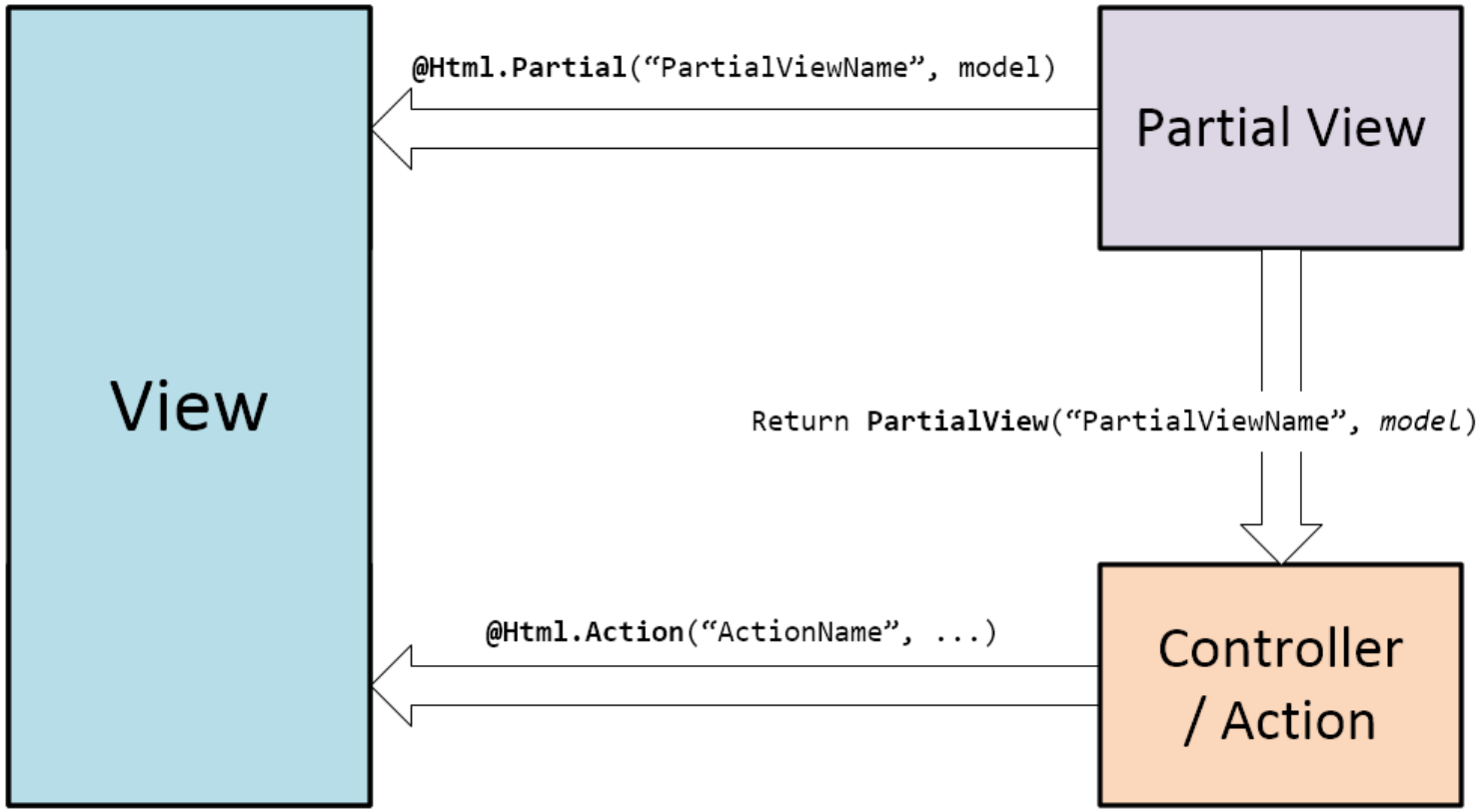
One of the following:

- Inherits from ViewComponent class
- Class name ends with ViewComponent
- Decorate with [ViewComponent] attribute

Views search path:

- Views/<controller_name>/Components/<view_component_name>/<view_name>
- Views/Shared/Components/<view_component_name>/<view_name>

# Partial View



# Entity Framework Core

# Object-Relational Mapping (ORM) Framework

# Entities

```
public class Category {

  public int Id { get; set; }

  [StringLength(200)]
  public string Name { get; set; }

  public virtual ICollection<Product> Products
  { get; set; }

  public Category() {
    Name = "General";
    Products = new HashSet<Product>();
  }
}
```

```
public class Product {
  [Key]
  public int Code { get; set; }

  [Required, StringLength(200)]
  public string Name { get; set; }

  [Range(0, 9999)]
  public decimal Price { get; set; }

  [Required]
  public virtual Category Category { get; set; }

  [ForeignKey("Category")]
  public int CategoryId { get; set; }

  public Product() {
    Name = "Untitled";
    Price = 0m;
  }
}
```

# DbContext

```
public class MyContext : DbContext {

  public DbSet<Category> Categories { get; set; }
  public DbSet<Product> Products { get; set; }

  // optional, just for exsample.
  protected override void OnModelCreating(DbModelBuilder modelBuilder) {
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<Product>().ToTable("Items");
  }
}
```

# Model and DbContext class

```
namespace core01.Models {
    public class Box {

        public int Id { get; set; }
        public double Width { get; set; }
        public int Height { get; set; }
        public string Color { get; set; }

    }
}
```

```
using Microsoft.EntityFrameworkCore;

namespace core01.Models {
    public class BoxDb : DbContext {

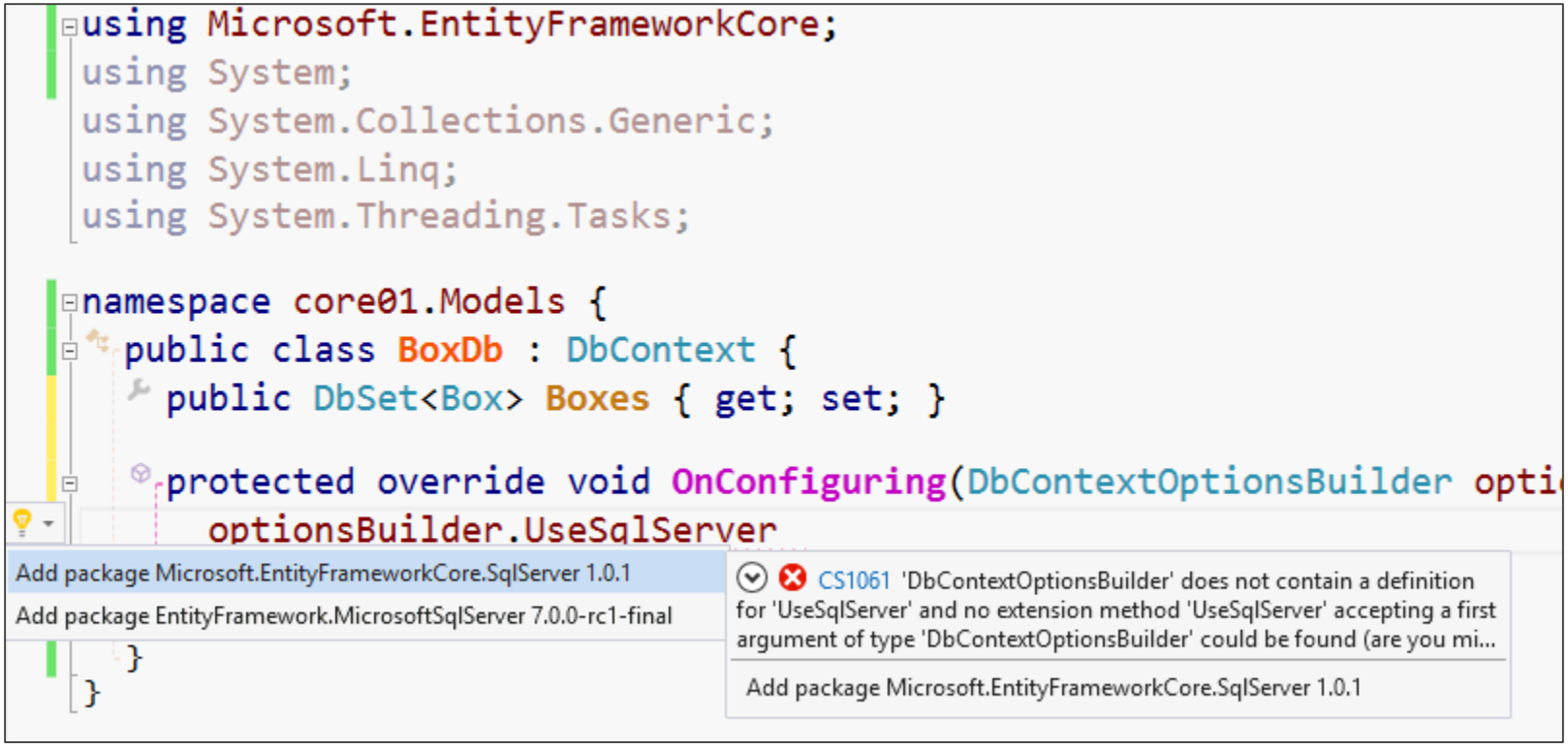
        public DbSet<Box> Boxes { get; set; }

    }
}
```

# DbContext

- In a short moment of time
- In a small space area
- For a single purpose task

# Add SqlServer to EF Core



# DbContext with Configuring

```
using Microsoft.EntityFrameworkCore;

namespace core01.Models {
  public class BoxDb : DbContext {

    public DbSet<Box> Boxes { get; set; }

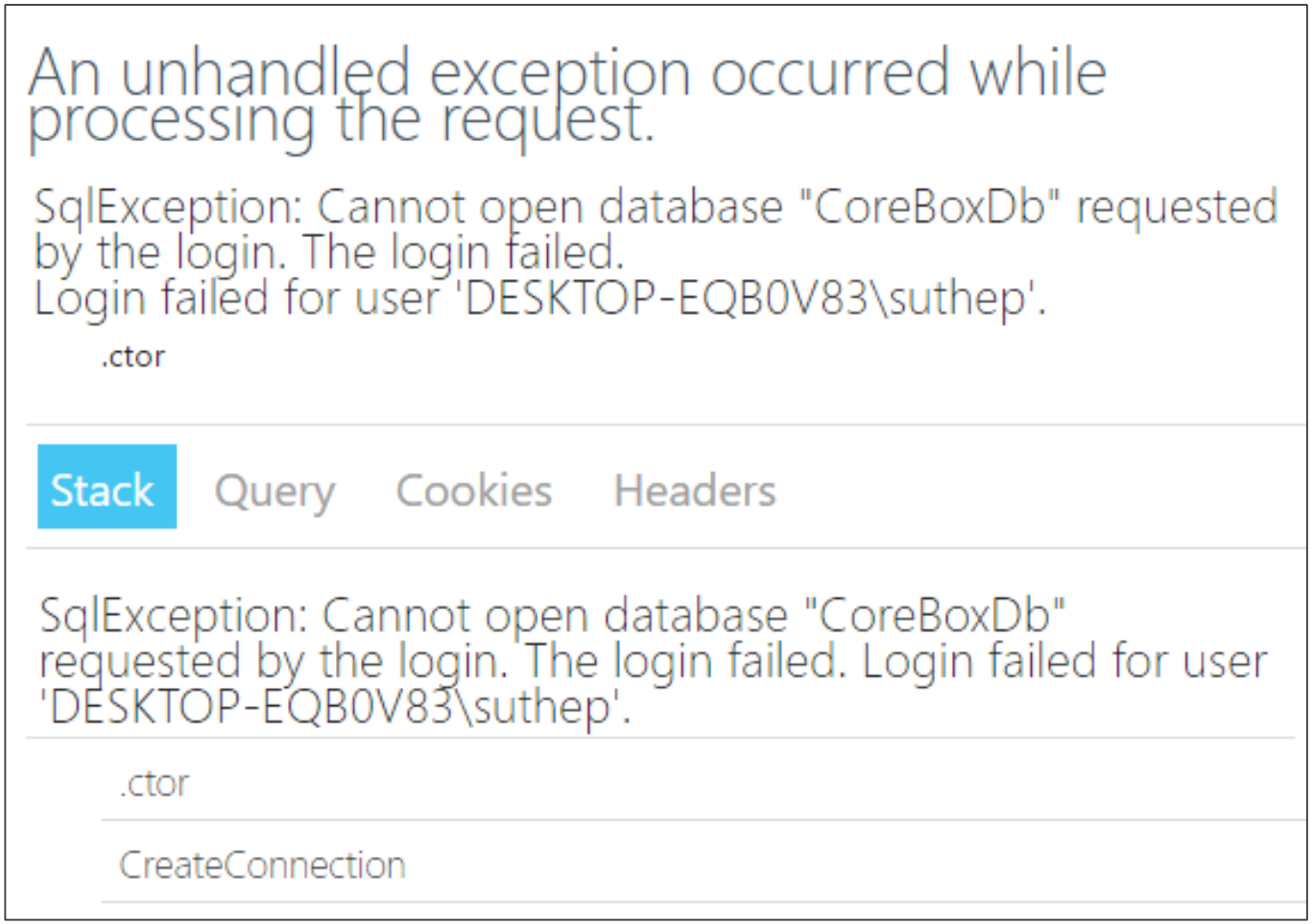
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {
      optionsBuilder.UseSqlServer(@"Data Source=.\\sqlexpress;
          Initial Catalog=CoreBoxDb;Integrated Security=True;");
    }
  }
}
```

# Add EF Required Dependencies

```
    "BundlerMinifier.Core": "2.2.281",
    "Microsoft.EntityFrameworkCore": "1.0.1",
    "Microsoft.EntityFrameworkCore.Design": "1.0.0-preview2-final",
    "Microsoft.EntityFrameworkCore.SqlServer": "1.0.1"
  },

  "tools": {
    "BundlerMinifier.Core": "2.0.238",
    "Microsoft.AspNetCore.Razor.Tools": "1.0.0-preview2-final",
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
    "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.0.0-preview2-final"
  },
```

# Database is not created automatically



# Create database with first migration

> **dotnet ef** migrations add Initial

> **dotnet ef** database update

# EF Core CLI

- dotnet ef **dbcontext** [ list | scaffold ]
- dotnet ef **database** [ drop | update ]
- dotnet ef **migrations** [ add | list | remove | script ]

# Migration with PM Console

Tools > NuGet Package Manager > Package Manager Console

- **Add-Migration** –Name *name* –OutputDir *path*
- Scaffold-DbContext
- Script-Migration
- **Update-Database** –Migration <*0*|*name*>
- **Use-DbContext** *dbContextName*

```
# Invokes the EF Core command
PS> EntityFrameworkCore\Add-Migration

# Invokes the EF 6 command
PS> EntityFramework\Add-Migration
```

# Inject DbContext with DI

```
using Microsoft.AspNetCore.Mvc;
using core01.Models;

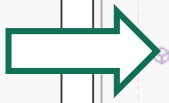
namespace core01.Controllers {

  public class HomeController : Controller {

    public IActionResult Index() {
      using (var db = new BoxDb()) {
        var b = new Box();
        b.Width = 5.5;
        b.Height = 333;
        b.Color = "maroon";

        db.Boxes.Add(b);
        db.SaveChanges();

        ViewBag.count = db.Boxes.Count();
      }
      return View();
    }
```



```
using Microsoft.AspNetCore.Mvc;
using core01.Models;

namespace core01.Controllers {

  public class HomeController : Controller {

    private BoxDb db;

    public HomeController(BoxDb db) {
      this.db = db;
    }
    public IActionResult Index() {

      var b = new Box();
      b.Width = 5.5;
      b.Height = 333;
      b.Color = "maroon";

      db.Boxes.Add(b);
      db.SaveChanges();

      ViewBag.count = db.Boxes.Count();

      return View();
    }
```

# Inject DbContext with DI (2)

```
using Microsoft.EntityFrameworkCore;

namespace core01.Models {
  public class BoxDb : DbContext {

    public DbSet<Box> Boxes { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {
      optionsBuilder.UseSqlServer(@"Data Source=.\\sqlexpress;
          Initial Catalog=CoreBoxDb;Integrated Security=True;");
    }
  }
}
```

```
using Microsoft.EntityFrameworkCore;

namespace core01.Models {
  public class BoxDb : DbContext {

    public BoxDb(DbContextOptions<BoxDb> options): base(options) {
      //
    }

    public DbSet<Box> Boxes { get; set; }

  }
}
```

# Inject DbContext with DI (3)

```
public void ConfigureServices(IServiceCollection services) {

  var cnString = @"Data Source=.\sqlexpress;Initial Catalog=CoreBoxDb;
            Integrated Security=True;";
  services.AddDbContext<BoxDb>(options => options.UseSqlServer(cnString));

  services.AddMvc();
}
```

# Configure Model

# Including and Excluding Types

- **[NotMapped]** to EntityClass.
- modelBuilder.**Ignore**<EntityClass>();

# Including and Excluding Properties

- **[NotMapped]** to Property
- modelBuilder.**Entity**<EntityClass>()
  .**Ignore**(e => e.Property);

# Keys

- **Id** or **<ClassName>Id**
- **[Key]**
- modelBuilder.**Entity**<EntityClass>()
  **.HasKey**(e => e.KeyProperty);
- Composite Key:
  modelBuilder.**Entity**<EntityClass>()
  **.HasKey**(e => new {
  e.KeyProperty1, e.KeyProperty2
  });

# Generated Properties

Apply to property

- `[DatabaseGenerated(DatabaseGeneratedOption.None)]`
- `[DatabaseGenerated(...Identity)]`
- `[DatabaseGenerated(...Computed)]`

- 
```
modelBuilder.Entity<EntityClass>()
            .Property(e => e.Property)
            .ValueGeneratedNever();
          // .ValueGeneratedOnAdd()
          // .ValueGeneratedOnAddOrUpdate()
```

# Required and Optional Properties

- Required:
  **Value Type** or **[Required]** on Reference Type
  ex. int, [Required] on string property.

- Optional:
  **Reference Type** or Nullable Value Type
  ex. int?, Product.

- ```
  modelBuilder.Entity<EntityClass>()
              .Property(e => e.Property)
              .IsRequired();
  ```
  ```

# Maximum Length

Apply to string and byte array

- `[MaxLength(50)]`
- `.HasMaxLength(50) // apply to property`

# Concurrency Tokens (1)

- EF uses an **optimistic concurrency pattern**, meaning it will assume the value has not changed and try to save the data, but throw if it finds the value has been changed.

```
public class Person
{
    public int PersonId { get; set; }
    [ConcurrencyCheck]
    public string LastName { get; set; }
    public string FirstName { get; set; }
}
```

```
UPDATE [Person] SET [FirstName] = @p1
WHERE [PersonId] = @p0 AND [LastName] = @p2;
```

# Concurrency Tokens (2)

- **[ConcurrencyCheck]**
- `modelBuilder.Entity<EntityClass>()`
  `.Property(e => e.Property)`
  `.IsConcurrencyToken();`

- **[Timestamp]**
- `modelBuilder.Entity<EntityClass>()`
  `.Property(e => e.Timestamp)`
  `.ValueGeneratedOnAddOrUpdate()`
  `.IsConcurrencyToken();`

```
[Timestamp]
public byte[] Timestamp { get; set; }
```

# Shadow Properties

- properties that do not exist in entity class but exist only in database table.

- ```
  // Set value
  context.Entry(myBlog)
         .Property("LastUpdated")
         .CurrentValue = DateTime.Now;
  ```

- ```
  // Use in LINQ query
  ```

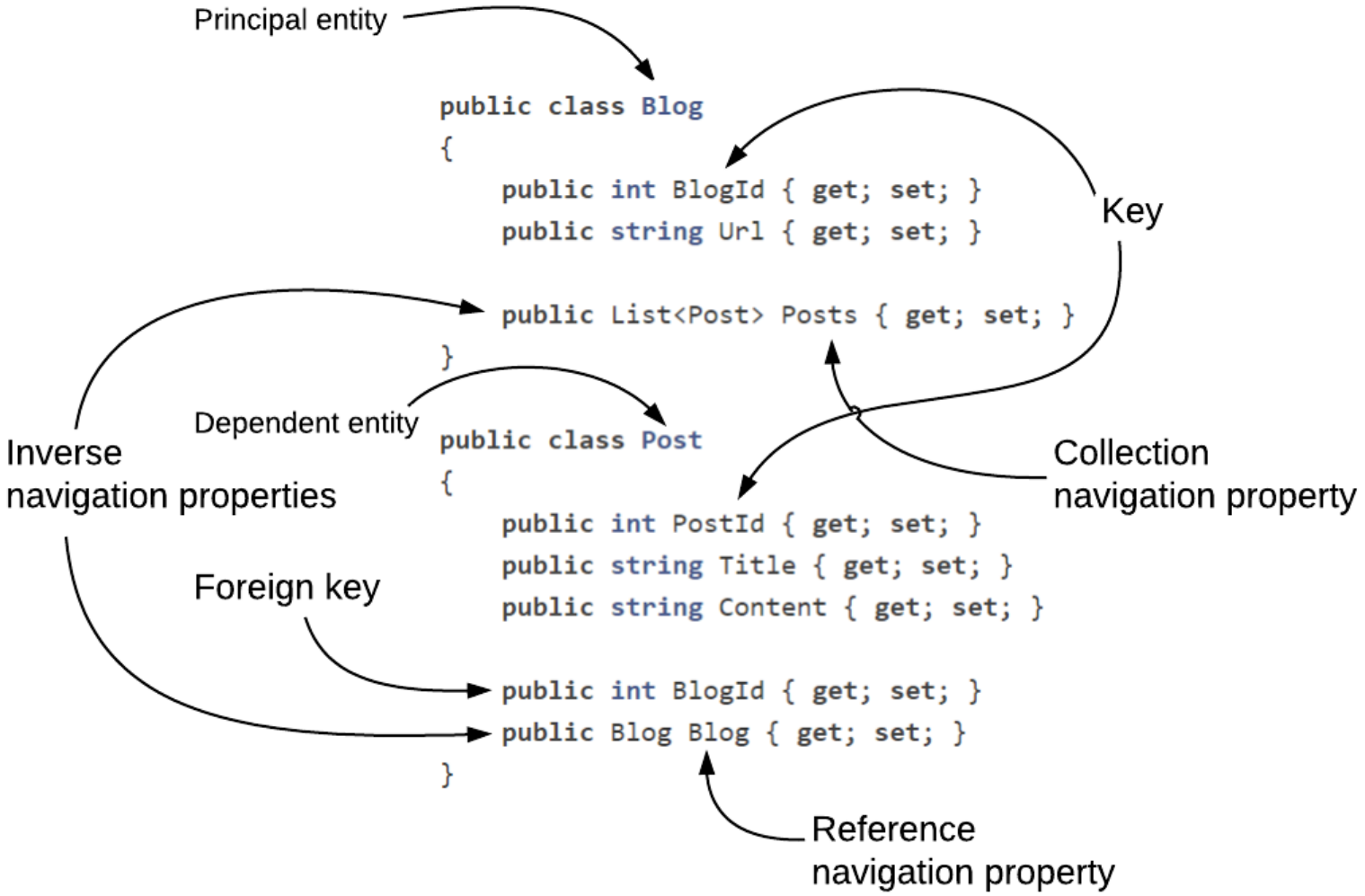
```
var blogs = context.Blogs
    .OrderBy(b => EF.Property<DateTime>(b, "LastUpdated"));
```

# Shadow Properties

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property<DateTime>("LastUpdated");
    }
}
```

# Relationship (1)



# Relationship (2)

- Single Navigation Property (uni-directional association)

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
}
```

# Relationship (3)

**Cascade Delete**

- By convention, cascade delete will be set to **Cascade** for required relationships and **SetNull** for optional relationships. Cascade means dependent entities are also deleted. SetNull means that foreign key properties in dependent entities are set to null.

# Relationship (4)

- ForeignKey attribute

```
public int BlogForeignKey { get; set; }

[ForeignKey("BlogForeignKey")]
public Blog Blog { get; set; }
```

# Relationship (5)

InverseProperty

```
public class Post
{
    public int PostId { get; set; }
    public string Title { get; set;
    public string Content { get; se

    public int AuthorUserId { get;
    public User Author { get; set;

    public int ContributorUserId { get; set; }
    public User Contributor { get; set; }
}
```

```
public class User
{
    public string UserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    [InverseProperty("Author")]
    public List<Post> AuthoredPosts { get; set; }

    [InverseProperty("Contributor")]
    public List<Post> ContributedToPosts { get; set; }
}
```

# Indexes

- By convention, an index is created in each property that are used as a foreign key.
- No data annotation!
- By Fluent API

```
modelBuilder.Entity<Blog>()
    .HasIndex(b => b.Url)
    .IsUnique();
```

- Multiple properties

```
modelBuilder.Entity<Person>()
    .HasIndex(p => new { p.FirstName, p.LastName });
```

# Alternative Keys (1)

- **An alternate key** serves as an alternate unique identifier for each entity instance in addition to the primary key. Alternate keys can be used as the target of a relationship.

```
class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Car>()
            .HasAlternateKey(c => new { c.State, c.LicensePlate });
    }
}

class Car
{
    public int CarId { get; set; }
    public string State { get; set; }
    public string LicensePlate { get; set; }
    public string Make { get; set; }
```

# Alternative Keys (2)

- By convention, an **alternate key** is introduced for you when you identify a property, that is not the primary key, as the target of a relationship.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .HasForeignKey(p => p.BlogUrl)
            .HasPrincipalKey(b => b.Url);
```

# Inheritance (1)

- EF will only setup inheritance if two or more inherited types are explicitly included in the model.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<RssBlog> RssBlogs { get; set; }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}

public class RssBlog : Blog
{
    public string RssUrl { get; set; }
}
```

# Inheritance (2)

- Not expose RssBlog as DbSet<T>

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(Mode
    {
        modelBuilder.Entity<RssBlog>();
    }
}
```

# Backing Fields (1)

- When a **backing field is configured**, EF will write directly to that field when materializing entity instances from the database (rather than using the property setter). This is useful when there is no property setter, or the setter contains logic that should not be executed when setting initial property values for existing entities being loaded from the database.

- <propertyName> differing only by case
- _<propertyName>
- m_<propertyName>

```
public class Blog
{
    private string _url;

    public int BlogId { get; set; }

    public string Url
    {
        get { return _url; }
        set { _url = value; }
    }
}
```

# Backing Fields (2)

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .HasAnnotation("BackingField", "_blogUrl");
    }
}

public class Blog
{
    private string _blogUrl;

    public int BlogId { get; set; }
    public string Url => _blogUrl;
}
```

# Relational Database Modeling (1)
# Table and Column

- [Table("blogs")]
- [Table("blogs", Schema = "blogging")]
- [Column("blog_id")]
- [Column(TypeName = "varchar(200)")]

```
modelBuilder.Entity<Blog>()
    .Property(b => b.Url)
    .HasColumnType("varchar(200)");
```

```
modelBuilder.Entity<Blog>()
    .Property(b => b.Url)
    .ForSqlServerHasColumnType("varchar(200)");
```

# Relational Database Modeling (2)
# Default Schema

```
protected override void OnModelCreating(ModelBuil
{
    modelBuilder.HasDefaultSchema("blogging");
}
```

# Relational Database Modeling (3)
# Computed Column

```
class MyContext : DbContext
{
    public DbSet<Person> People { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Person>()
            .Property(p => p.DisplayName)
            .HasComputedColumnSql("[LastName] + ', ' + [FirstName]");
    }
}

public class Person
{
    public int PersonId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string DisplayName { get; set; }
}
```

# Relational Database Modeling (4)
# Sequence

- A sequence generates a sequential numeric values in the database. Sequences are not associated with a specific table.

```
class MyContext : DbContext
{
    public DbSet<Order> Orders { get; set; }

    protected override void OnModelCreating(ModelBuilde
    {
        modelBuilder.HasSequence<int>("OrderNumbers");
    }
}
```

# Relational Database Modeling (5)
# Sequence

```
public class Order
{
    public int OrderId { get; set; }
    public int OrderNo { get; set; }
    public string Url { get; set; }
}
```

```
class MyContext : DbContext
{
    public DbSet<Order> Orders { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.HasSequence<int>("OrderNumbers", schema: "shared")
            .StartsAt(1000)
            .IncrementsBy(5);

        modelBuilder.Entity<Order>()
            .Property(o => o.OrderNo)
            .HasDefaultValueSql("NEXT VALUE FOR shared.OrderNumbers");
    }
}
```

# Relational Database Modeling (6)
# Default Value

```
modelBuilder.Entity<Blog>()
    .Property(b => b.Rating)
    .HasDefaultValue(3);
```

```
modelBuilder.Entity<Blog>()
    .Property(b => b.Created)
    .HasDefaultValueSql("getdate()");
```

# Relational Database Modeling (7)

# Inheritance

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<RssBlog> RssBlogs { get; set; }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}

public class RssBlog : Blog
{
    public string RssUrl { get; set; }
}
```

Results

| | BlogId | Discriminator | Url | RssUrl |
|---|---|---|---|---|
| 1 | 1 | Blog | http://blogs.msdn.com/dotnet | NULL |
| 2 | 2 | RssBlog | http://blogs.msdn.com/adonet | http://blogs.msdn.com/b/adonet/atom.aspx |

# Relational Database Modeling (8)
# Inheritance

```
protected override void OnModelCreating(ModelBuilde
{
    modelBuilder.Entity<Blog>()
        .HasDiscriminator<string>("blog_type")
        .HasValue<Blog>("blog_base")
        .HasValue<RssBlog>("blog_rss");
}
```

# EF Core Misc.

# Eager Loading

```
var blogs = context.Blogs
    .Include(blog => blog.Posts)
    .ToList();
```

```
var blogs = context.Blogs
    .Include(blog => blog.Posts)
    .Include(blog => blog.Owner)
    .ToList();
```

```
var blogs = context.Blogs
    .Include(blog => blog.Posts)
        .ThenInclude(post => post.Author)
        .ThenInclude(author => author.Photo)
    .ToList();
```

# Explicit Loading

- Not perfect in EF Core now. Just load it to the context and navigation properties will be wired up automatically.

```
var blog = context.Blogs
    .Single(b => b.BlogId == 1);

context.Posts
    .Where(p => p.BlogId == blog.BlogId)
    .Load();
```

# EF 6.x feature that not implemented or limited in EF Core 1

## Creating a Model

- Complex type
- Visualizing a model
- Simple type convention – such as string to xml
- Spatial data types
- Many-to-many relationships without join entity
- Alternate inheritance mapping patterns – just Table per hierarchy (TPH) is supported. (not table per type (TPT) and table per concrete type (TPC))

# EF 6.x feature that not implemented or limited in EF Core 1 (2)

## Querying Data

- Improved translation
- GroupBy translation

- Lazy loading
- Explicit loading

- Raw SQL queries for non-model types

# EF 6.x features that not implemented or limited in EF Core 1 (3)

## Saving Data

- Simple command interception
- Missing EntityEntry APIs – Ex. Reload, GetModifiedProperties, GetDatabaseValues
- Stored procedure mapping
- Connection resiliency

# EF 6.x features that not implemented or limited in EF Core 1 (4)

## Database Schema Management

- Visual Studio wizard for reverse engineer
- Update model from database
- Seed data

# ASPNET Core Identity

# Introduction to Identity

https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity

- Create a new ASP.NET Core Web Application with Individual User Accounts
- Package: Microsoft.AspNetCore.Identity.EntityFrameworkCore
- **ConfigureServices**()
- **Configure**()
  - Configuring IdentiyOption (Password, Lockout, Cookie, User)
- Exploring **AccountController**
- Sign Up, Sign In, Sign Out
- Viewing Database

# User Secrets (1)

- Package CLI Tool:
  **Microsoft.Extensions.SecretManager.Tools**

- dotnet user-secret set <name> <value>

```
$ dotnet user-secrets set smtp smtp.gmail.com
Successfully saved smtp = smtp.gmail.com to the secret store.
```

# User Secrets (2)

- Read user secrets into Configuration

```
public Startup(IHostingEnvironment env)
{
  var builder = new ConfigurationBuilder()
      .SetBasePath(env.ContentRootPath)
      .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
      .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true);

  if (env.IsDevelopment())
  {
    builder.AddUserSecrets<Startup>();
  }

  builder.AddEnvironmentVariables();
  Configuration = builder.Build();
}
```

- Get Value:

```
var smtp = Configuration["smtp"];
```

# SSL

- Enable SSL in IIS Express
- Trusted IIS Express
- Enforcing SSL

  [RequireHttps]

```
services.Configure<MvcOptions>(options =>
{
  options.Filters.Add(new RequireHttpsAttribute());
});
```

# Facebook Authentication

- https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/facebook-logins

```
dotnet add package Microsoft.AspNetCore.Authentication.Facebook
```

```
app.UseFacebookAuthentication(new FacebookOptions()
{
  AppId = Configuration["Authentication:Facebook:AppId"],
  AppSecret = Configuration["Authentication:Facebook:AppSecret"]
});
```

# Google Authentication

- https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/google-logins

```
dotnet add package Microsoft.AspNetCore.Authentication.Google
```

```
app.UseGoogleAuthentication(new GoogleOptions()
{
  ClientId = Configuration["Authentication:Google:ClientId"],
  ClientSecret = Configuration["Authentication:Google:ClientSecret"]
});
```

# Microsoft Authentication

- https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/microsoft-logins

```
dotnet add package
Microsoft.AspNetCore.Authentication.MicrosoftAccount
```

```
app.UseGoogleAuthentication(new GoogleOptions()
{
  ClientId = Configuration["Authentication:Google:ClientId"],
  ClientSecret = Configuration["Authentication:Google:ClientSecret"]
});
```

# Using MailKit

- https://code.msdn.microsoft.com/Send-Email-Using-ASPNET-1c62bdfd/sourcecode?fileId=171277&pathId=1948400523

```
using (var client = new SmtpClient())
{
 client.Connect(_appSettings.SmtpServerAddress, _appSettings.SmtpServerPort, SecureSocketO
 client.AuthenticationMechanisms.Remove("XOAUTH2"); // Must be removed for Gmail SMTP
 client.Authenticate(_appSettings.SmtpServerUser, _appSettings.SmtpServerPass);
 client.Send(Email);
 client.Disconnect(true);
}
```

# Using SendGrid (1)

```
namespace Web.Services {
  public class AuthMessageSenderOptions {
    public string SendGridUser { get; set; }
    public string SendGridKey { get; set; }
  }
}
```

# Using SendGrid (2)

```
public class AuthMessageSender : IEmailSender, ISmsSender {

  public AuthMessageSender(IOptions<AuthMessageSenderOptions> optionsAccessor) {
    Options = optionsAccessor.Value;
  }

  public AuthMessageSenderOptions Options { get; }


  public async Task SendEmailAsync(string email, string subject, string message) {
    var apiKey = Options.SendGridKey;
    var client = new SendGridClient(apiKey);
    var from = new EmailAddress("me@company.com", "Me");
    var to = new EmailAddress(email);
    var plainTextContent = message;
    var htmlContent = message;
    var msg = MailHelper.CreateSingleEmail(from, to, subject, plainTextContent, htmlContent);
    var response = await client.SendEmailAsync(msg);
  }

  public Task SendSmsAsync(string number, string message) {
    return Task.FromResult(0);
  }
}
```

# Account Confirmation and Password Recovery

- https://docs.microsoft.com/en-us/aspnet/core/security/authentication/accconfirm

# Customize ApplicationUser

- Modify ApplicationUser
- And do entity framework migrations

# Get ApplicationUser

- UserManager<ApplicationUser>

- Inject in Controller
- Inject in View

# Update ApplicationUser

- Modify custom properties and update back to database using UserManager.

# Role Basics

- [Authorize]
  - Can be applied in Action and Controller level
- [AllowAnonymous]
  - Take precedence over AuthorizeAttribute

# Role-based Authorization

- [Authorize(Roles = “Editor”)]

- *OR (use comma separated)*

  [Authorize(Roles = “Writer, Editor”)]

- *AND (use multiple attribute declaration)*

  [Authorize(Roles = “Writer”)]
  [Authorize(Roles = “Editor”)]

# Policy-based Authorization

```
services.AddAuthorization(options =>
{
  options.AddPolicy("RequireAdministratorRole",
    policy => policy.RequireRole("Administrator"));
});
```

```
[Authorize(Policy = "RequireAdministratorRole")]
public IActionResult Shutdown()
{
    return View();
}
```

# Manage Roles

- RoleManager<IdentityRole>

# Claim-based Authorization

```
var list = await mgr.GetClaimsAsync(user);
if (!list.Any(c => c.Type == "EmpId"))
{
  await mgr.AddClaimAsync(user, new Claim("EmpId", "5901"));
}
```

```
services.AddAuthorization(options =>
{

  options.AddPolicy("EmployeeOnly",
    policy => policy.RequireClaim("EmpId"));

});
```

```
[Authorize(Policy = "EmployeeOnly")]
public IActionResult VacationBalance()
{
    return View();
}
```

# View-based Authorization

- @inject IauthorizationService auth

```
@using Microsoft.AspNetCore.Authorization
@inject IAuthorizationService auth

@model ApplicationUser

<h2>Home</h2>

@if (await auth.AuthorizeAsync(User, "EmployeeOnly"))
{
  <p>This paragraph is displayed because you fulfilled EmployeeOnly.</p>
}
```

# Enable Cross-Site Origin Requests (CORS)

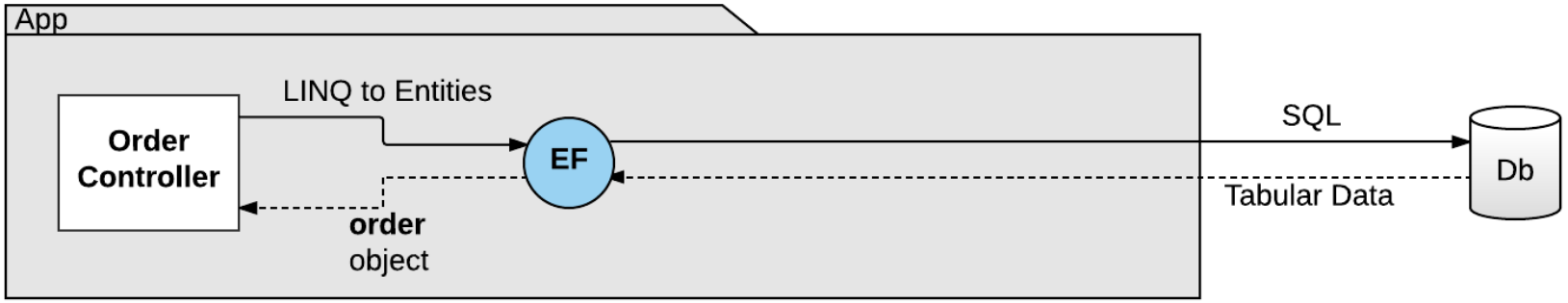
```
Microsoft.AspNetCore.Cors
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors(options =>
    {
        options.AddPolicy("AllowSpecificOrigin",
            builder => builder.WithOrigins("http://example.com"));
    });
}
```

```
// Shows UseCors with named policy.
app.UseCors("AllowSpecificOrigin");
```

# Architecture

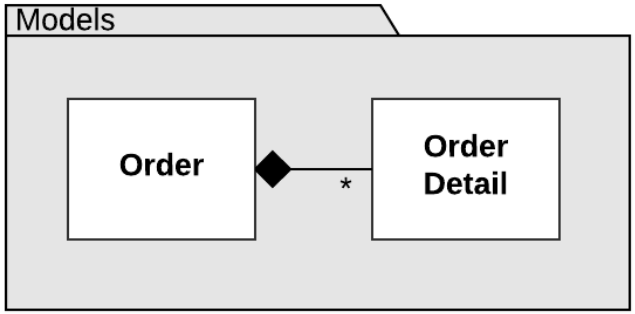
# Basic MVC App




- Application Logics
- Presentation Logics

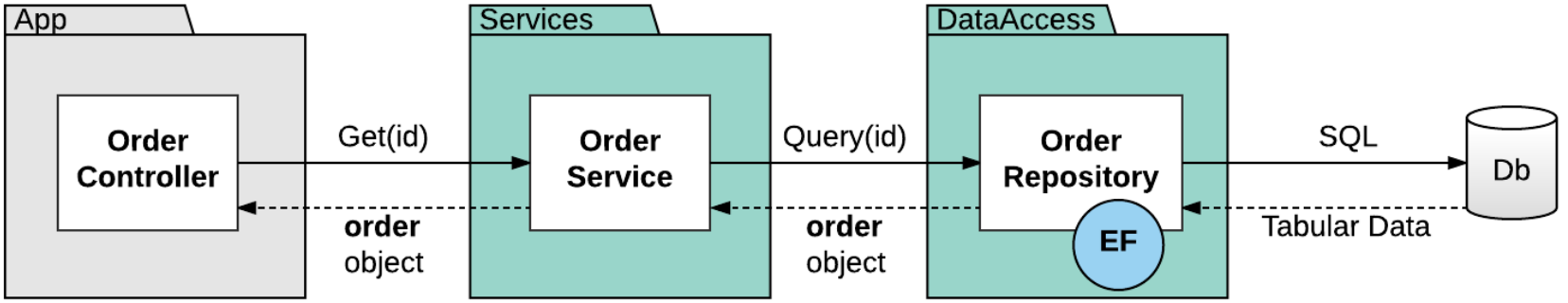
- Business Logics in higher-level and container-level (CRUD).

- Data access logic (independent to vendor-specific data store)
- Best to implement with Entity Framework or any ORM.




- Business Logics in model-level.
- Model is neutral to technology, framework, application types, user-interface, and database.
- Ideal model is a POCO. (plain-old CLR class)

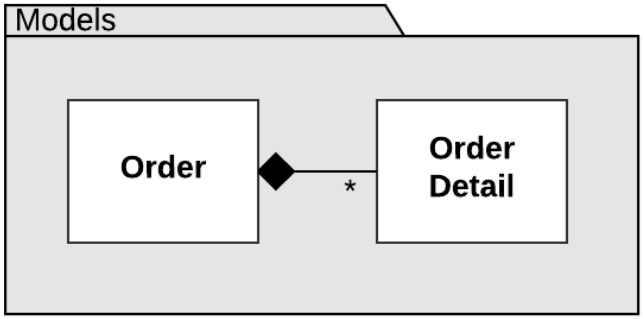
# Suggested MVC App Arch.




- Application Logics
- Presentation Logics

- Business Logics in higher-level and container-level (CRUD).

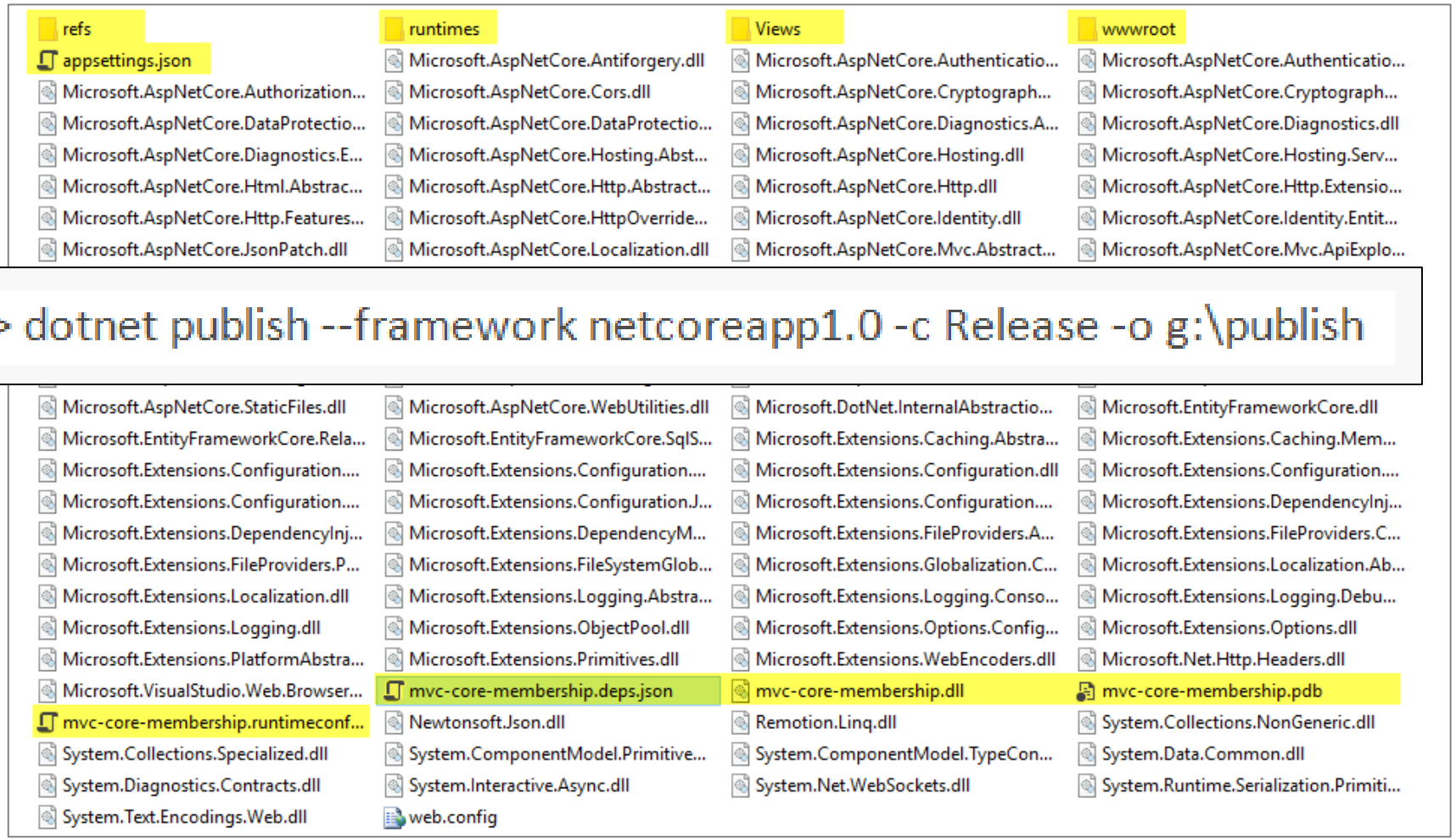
- Data access logic (independent to vendor-specific data store)
- Best to implement with Entity Framework or any ORM.



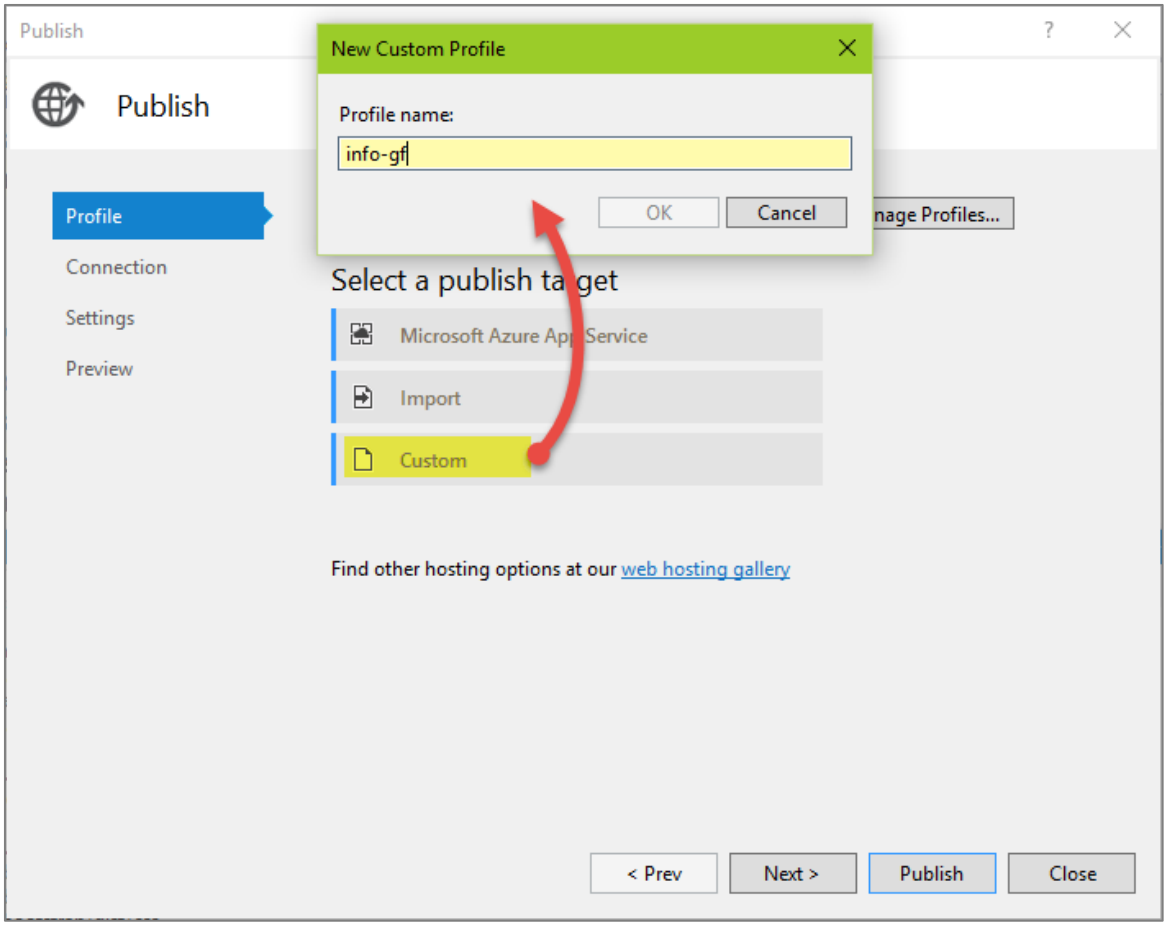

- Business Logics in model-level.
- Model is neutral to technology, framework, application types, user-interface, and database.
- Ideal model is a POCO. (plain-old CLR class)

# Deployment

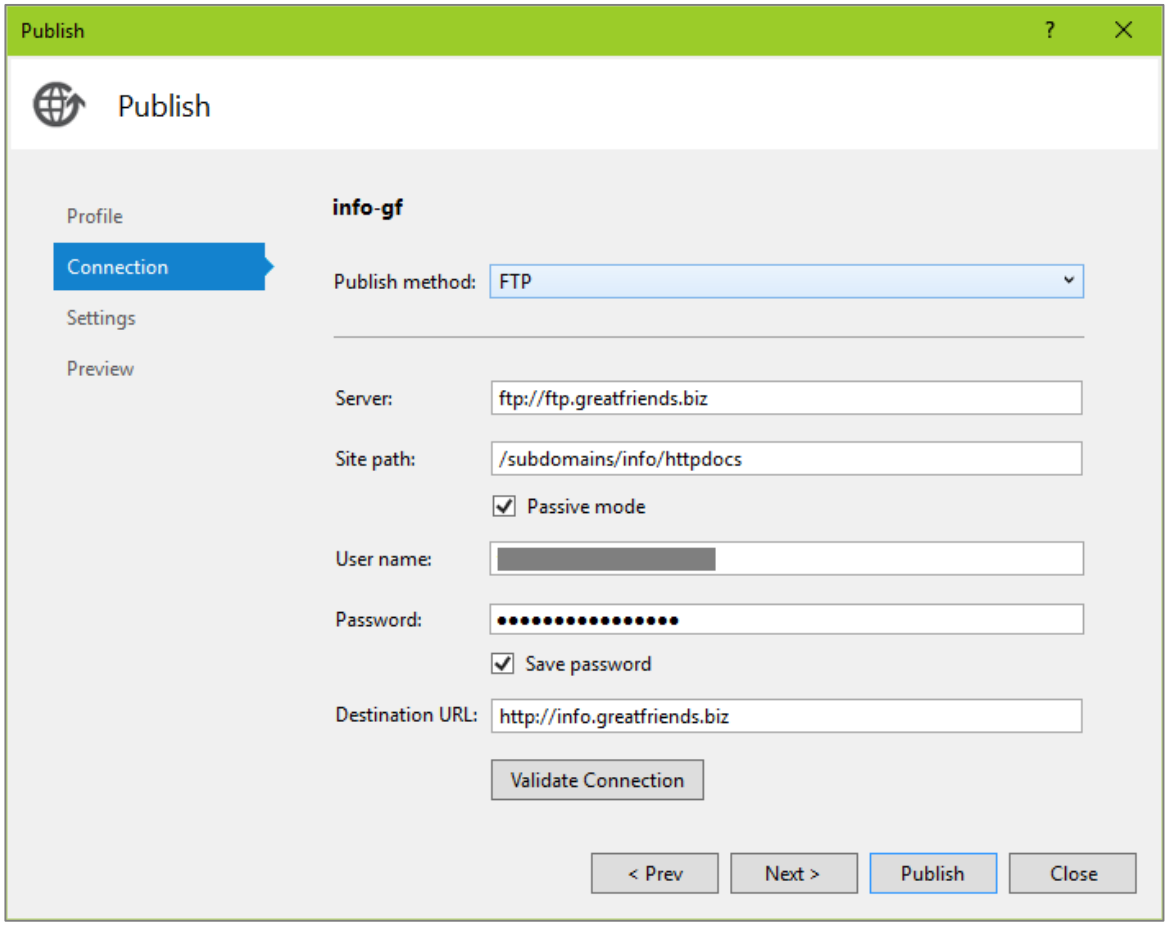
# dotnet publish



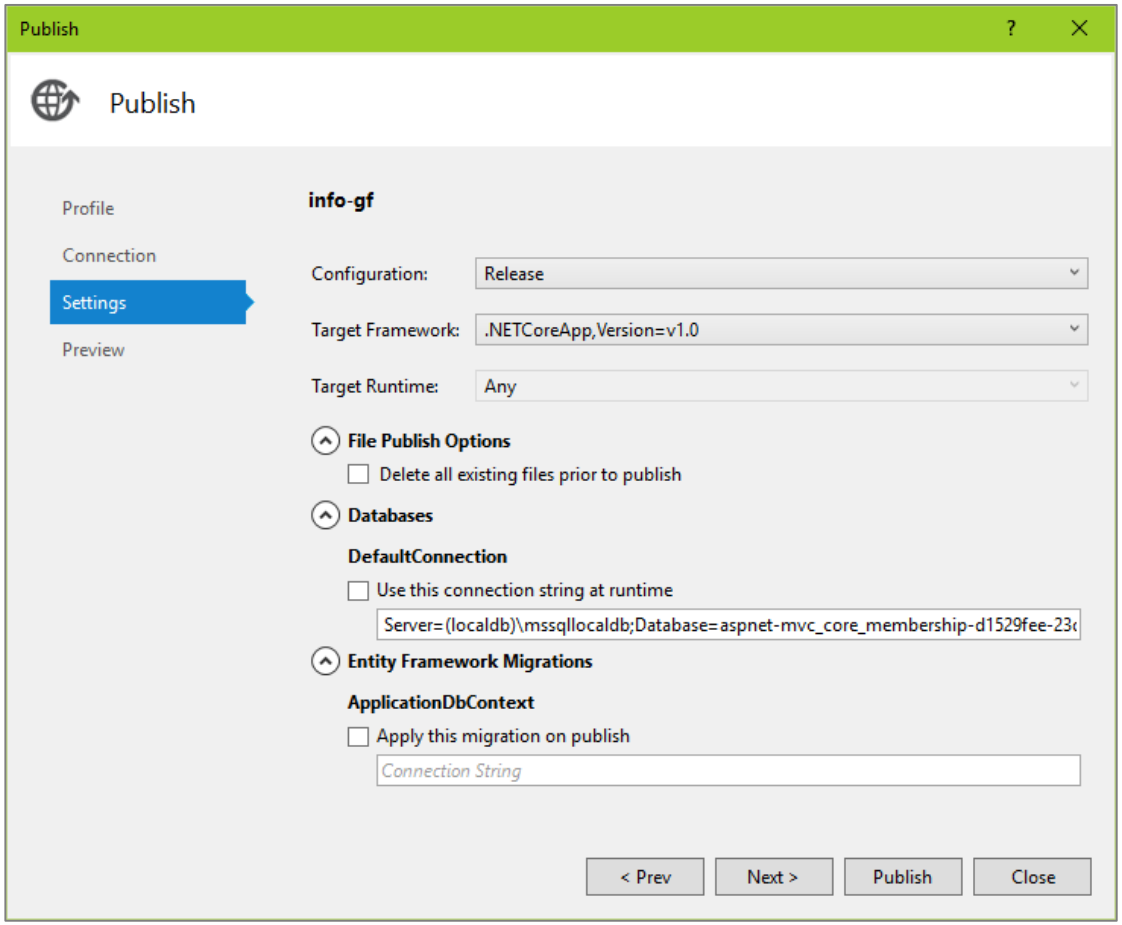
# Build > Publish > Create new Custom Profile



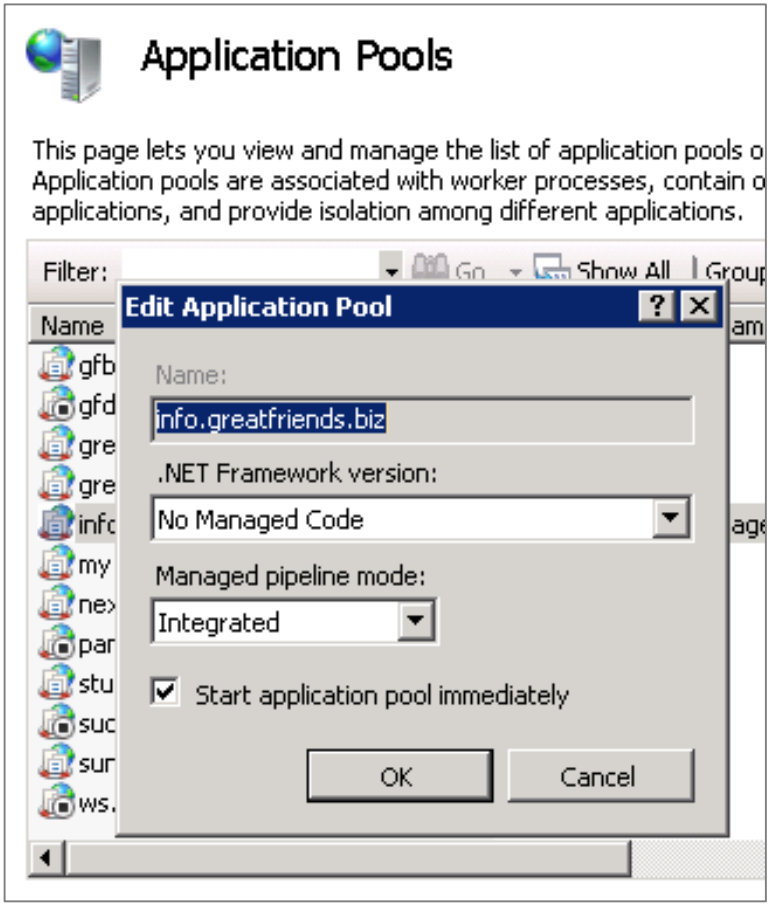
# Connection


Publish
Publish
Profile
Connection
Settings
Preview
info-gf
Publish method:
FTP
Server:
ftp://ftp.greatfriends.biz
Site path:
/subdomains/info/httpdocs
Passive mode
User name:
Password:
Save password
Destination URL:
http://info.greatfriends.biz
Validate Connection
< Prev
Next >
Publish
Close

# Settings


Publish
Publish
Profile
Connection
Settings
Preview
info-gf
Configuration:
Release
Target Framework:
.NETCoreApp,Version=v1.0
Target Runtime:
Any
File Publish Options
Delete all existing files prior to publish
Databases
DefaultConnection
Use this connection string at runtime
Server=(localdb)\mssqllocaldb;Database=aspnet-mvc_core_membership-d1529fee-23
Entity Framework Migrations
ApplicationDbContext
Apply this migration on publish
Connection String
< Prev
Next >
Publish
Close

# Setting Application Pools




app_offline
.htm