



.NET Core: Developing Cross-Platform Web Apps with ASP.NET Core – Workshop*PLUS*

Wael Kdounh - @waelkdounh

Senior Consultant

v3.0

Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

© 2016 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at

<http://www.microsoft.com/en-us/legal/intellectualproperty/permissions/default.aspx>

Internet Explorer, Microsoft, Microsoft Corporate Logo, MSDN, Visual Studio, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Module 3: Controllers

Module Overview

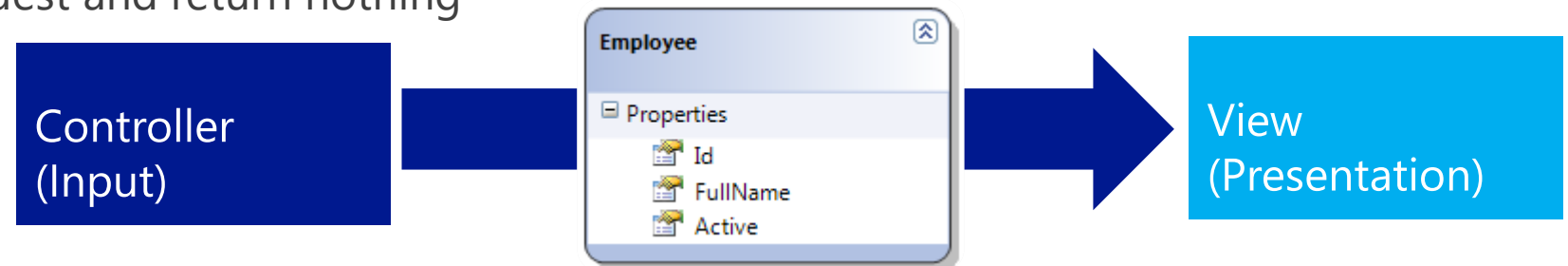
Module 3: Controllers

Section 1: Controller Fundamentals

Lesson: Role of Controllers

Controller

- Controller is responsible for:
 - Responding to requests made against ASP.NET MVC
 - Manipulating the model (if necessary)
 - Selecting a View to send back to user via response
- Each browser request is mapped against a particular controller
- Controller may:
 - Return a view
 - Redirect the user to another controller
 - Perform action on the request and return nothing



Role of Controller

- Controller handles and responds to user input and interaction
- Example
 1. User sends a URL request with query string values
 2. *Controller* is triggered against the request
 3. *Controller* handles query-string values
 4. *Controller* passes the values to the model
 5. Model uses the value to query the database and returns the results
 6. *Controller* selects a View to render the UI
 7. *Controller* returns the View to the requesting browser

Module 3: Controllers

Section 2: Developing Controllers

Lesson: Controller Development

Controller Development

- Controller class inherits from `Microsoft.AspNetCore.Mvc.Controller`
- Controller class name has 'Controller' suffix
 - For example: `HomeController`, `AccountController`, etc.

```
5 references
public class HomeController : Controller
{
    4 references
    public IActionResult Index()
    {
        // Index action implementation...
        return View();
    }

    0 references
    public IActionResult About()
    {
        ViewData["Message"] = "Your application description page.";

        return View();
    }
}
```

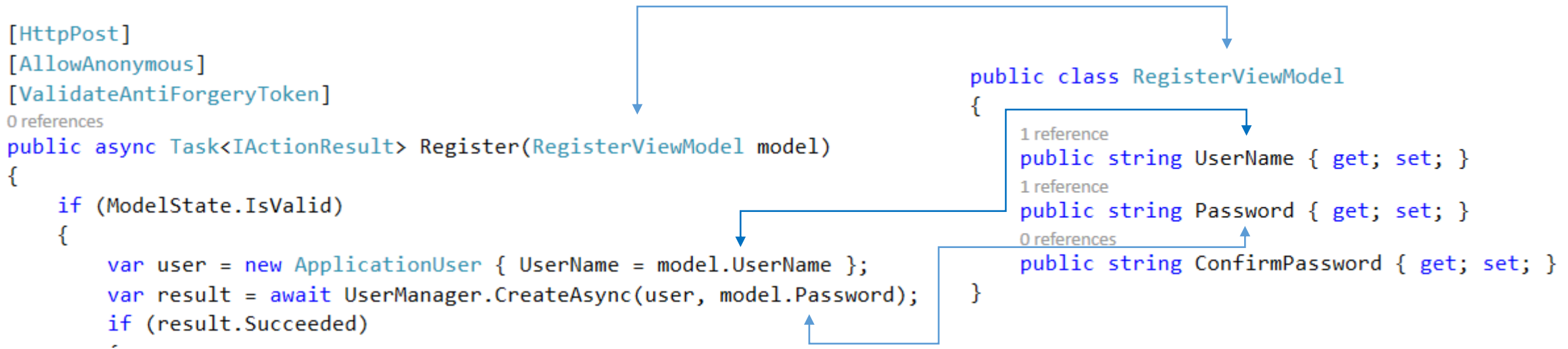

Default Controllers

- HomeController
 - Responsible for the home page at the root of the website
- AccountController
 - Responsible for application security, such as login and registration
- ManageController
 - Responsible for account management tasks, such as password change

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
1 reference
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)
{
    EnsureDatabaseCreated(_applicationDbContext);
    ViewData["ReturnUrl"] = returnUrl;
    if (ModelState.IsValid)
    {
        // This doesn't count login failures towards account lockout
        // To enable password failures to trigger account lockout, set lockoutOnFailure:
        var result = await _signInManager.PasswordSignInAsync(model.Email, model.Password,
```

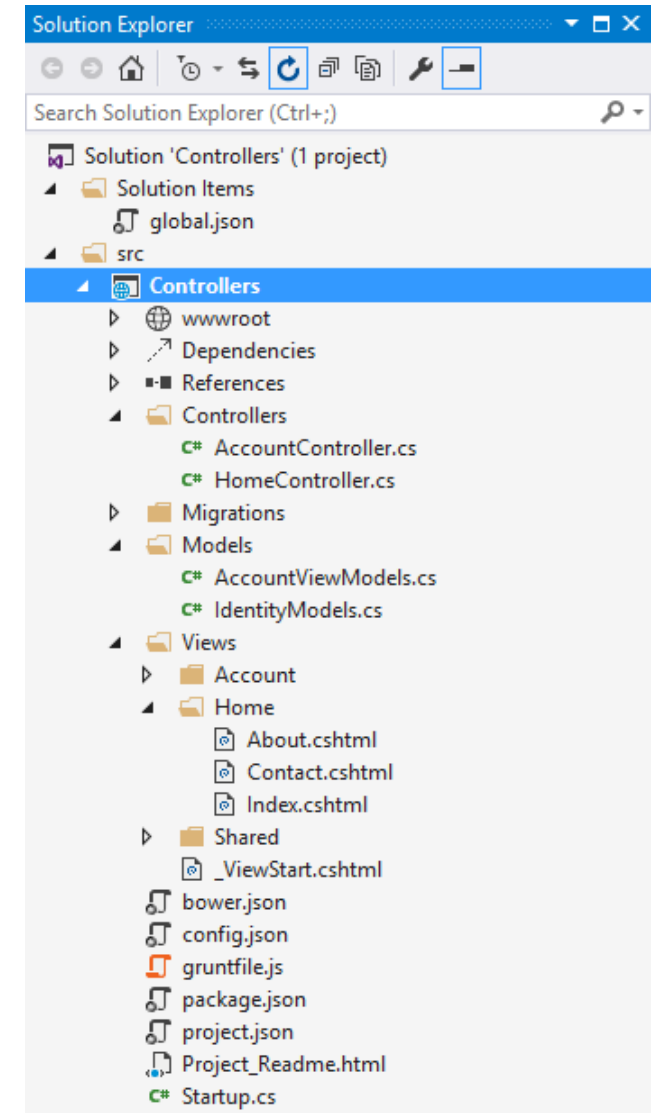
Default Model Binder

- Default Model Binder automatically populates controller action parameters
 - Takes care of mundane property mapping and type conversion
 - Uses the name attribute of input elements
 - Automatically matches parameter names for simple data types
 - Complex objects are mapped by property name; use dotted notation



ASP.NET MVC Project File Organization

- Default conventions can be overridden
 - ASP.NET MVC does not require this structure
- Convention over configuration
 - Default directory structure keeps application concerns clean
 - For example: Allows you to omit location paths when referencing views
 - By default, ASP.NET MVC looks for the View template file in **`\Views\[ControllerName]\`**



Demo: Organization of Controllers, Views, and Models

Module 3: Controllers

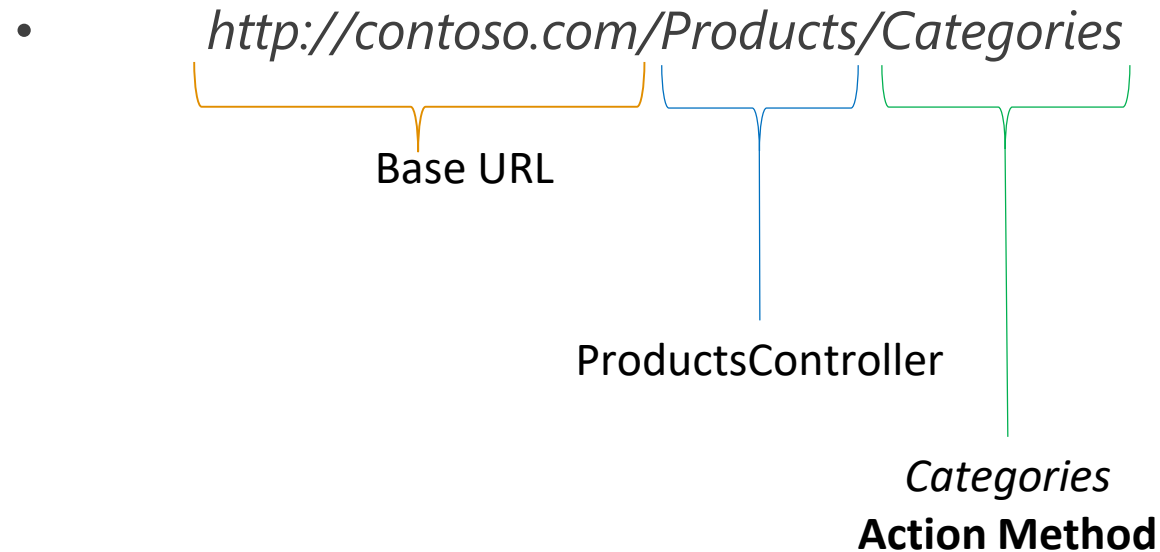
Section 2: Developing Controllers

Lesson: Action Methods

Action Methods

- Action methods have 1:1 mapping with user interaction
 - Form submission, clicking a link, etc.
- User actions lead to action method calls

- Example:



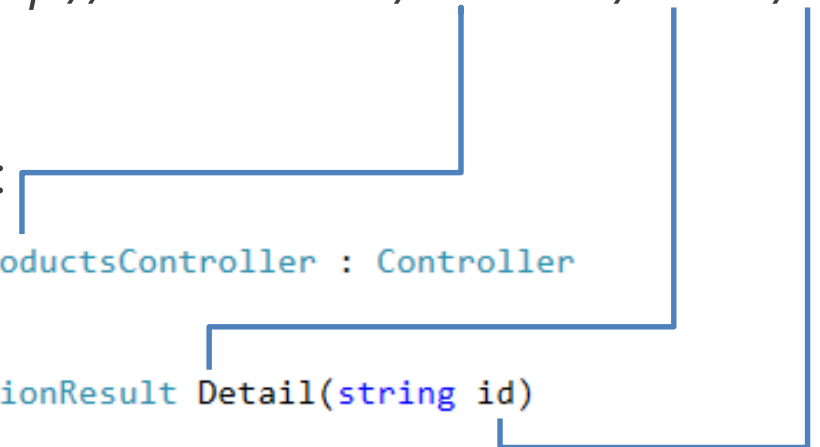
Action Method Parameters

- Query string or form post parameters are passed into Action methods as parameters

- URL: *http://contoso.com/Products/Detail/5*

- Action Method:

```
public class ProductsController : Controller
{
    0 references
    public IActionResult Detail(string id)
    {
        // Retrieve product detail using ID.
        return View();
    }
}
```

A diagram consisting of blue lines that maps parts of the URL 'http://contoso.com/Products/Detail/5' to the code. A line from 'Products' connects to the class name 'ProductsController'. A line from 'Detail' connects to the method name 'Detail'. A line from '5' connects to the parameter 'id' in the method signature.

Action Method Results

- The Action method typically returns a View, but not always

```
0 references
public class StoreController : Controller
{
    // GET: /Store/
    0 references
    public IActionResult Index()
    {
        return View();
    }

    // GET: /Store/Details
    0 references
    public IActionResult Details()
    {
        return View("Details");
    }

    // GET: /Store/Browse?Genre=Disco
    0 references
    public string Browse(string id)
    {
        return "Store.Browse, Genre = " + id;
    }
}
```

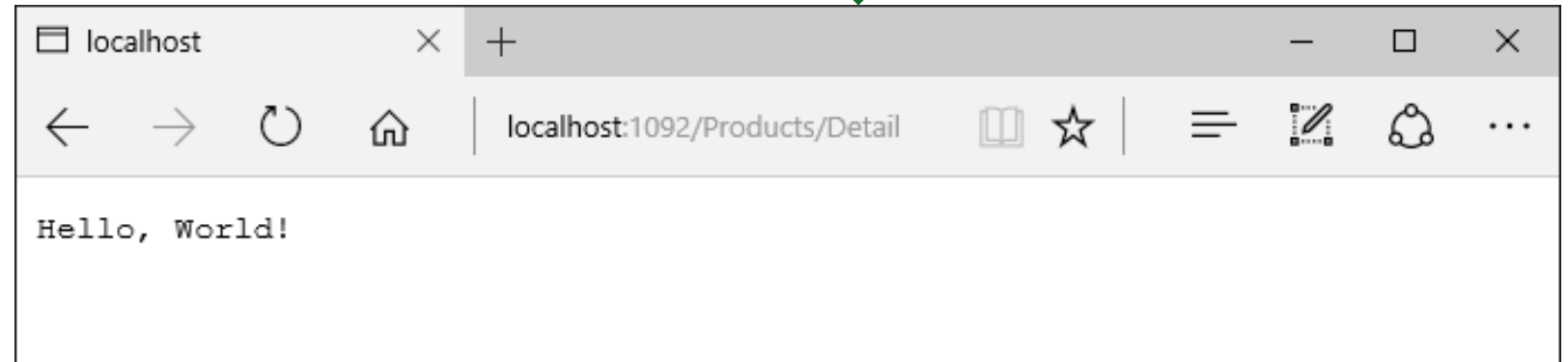
```
public FileResult SomeFile()
{
    string filename = @"C:\Example.pdf";
    string contentType = "application/pdf";
    string downloadName = "ExampleFile.pdf";

    return File(filename, contentType, downloadName);
}
```


Built-in Action Return Types

- All derive from **ActionResult**
 - ViewResult
 - PartialViewResult
 - RedirectToRouteResult
 - RedirectResult
 - ContentResult
 - FileResult
 - JsonResult
 - JavascriptResult
 - HttpUnauthorizedResult
 - HttpNotFoundResult
 - HttpStatusCodeResult
 - EmptyResult

```
public ContentResult Index()  
{  
    return Content("Hello, World!");  
}
```



Demo: Controller Development

Module 3: Controllers

Section 2: Developing Controllers

Lesson: Filters

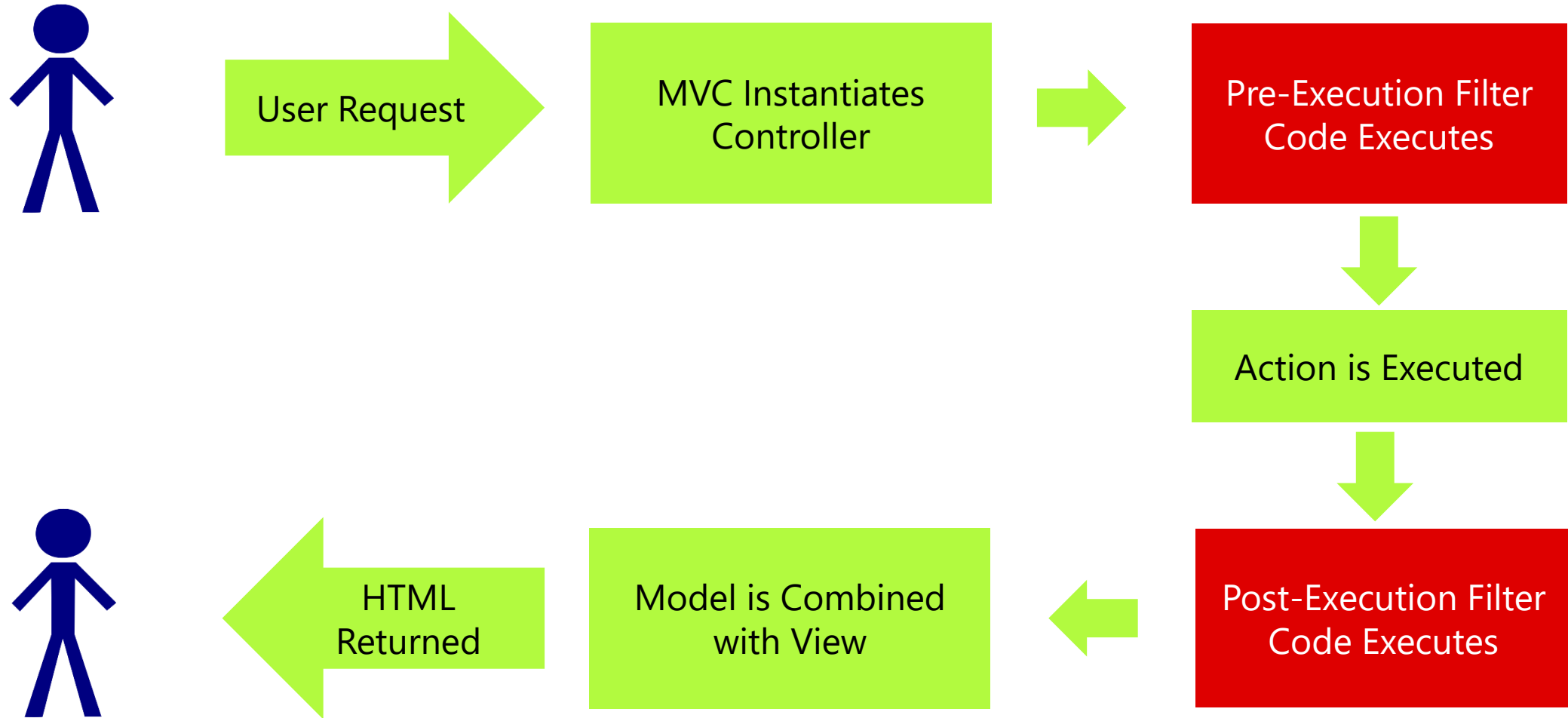
Action Filter and Selector Attributes

- Public controller methods are actions by default
- **Selectors** differentiate Actions
- **Filters** control access

```
//  
// POST: /Account/Register  
[HttpPost]  
[AllowAnonymous]  
[ValidateAntiForgeryToken]  
0 references  
public async Task<IActionResult> Register(RegisterViewModel model)  
{
```

Account Controller - Register Method

Actions with Filters



Selector Attributes

Selector Attribute	Description
[NonAction]	To mark a method as non-action
[HttpPost]	To restrict a method to handling only HTTP POST requests
[HttpPut]	To restrict a method to handling only HTTP PUT requests
[HttpGet]	To restrict a method to handling only HTTP GET requests
[AcceptVerbs]	Specifies which HTTP verbs an action method will respond to
...	...

Filter Attributes

Selector Attribute	Description
[AllowAnonymous]	To allow anonymous users to call the action method
[Authorize]	To restrict access to only those users that are authenticated and authorized
[RequireHttps]	To force an unsecured HTTP request to be resent over HTTPS
[ValidateAntiForgeryToken]	To prevent forgery of a request (Defense against CSRF attack)
[ValidateInput]	To mark action methods whose input must be validated
[HandleError]	To handle an exception that is thrown by an action method
...	...

Demo: Filters

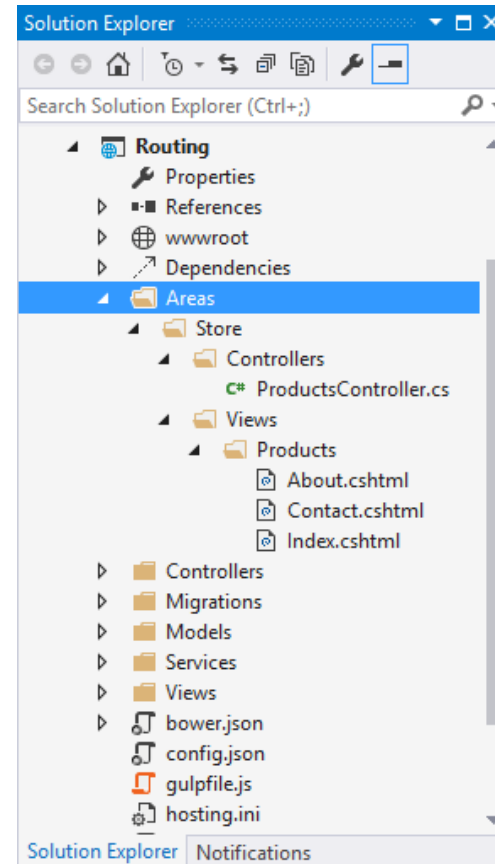
Module 3: Controllers

Section 2: Developing Controllers

Lesson: Areas

Areas

- MVC Areas separate a large MVC application into smaller functional groupings
 - For example: large e-commerce site divided into areas for storefront, product reviews, user accounts, etc.
- Area Guidelines
 - Areas directory must exist as a project child directory
 - Areas contains subdirectory for each area
 - Controllers should be located at:
/Areas/[area]/Controllers/[controller].cs
 - Views should be located at:
/Areas/[area]/Views/[controller]/[action].cshtml



Area Registration and Linking

- Area Registration

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

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

Startup.cs

```
namespace DemoApp.Areas.Store.Controllers
{
    [Area ("Store")]
    0 references
    public class ProductsController : Controller
    {
        0 references
        public IActionResult Index()
        {
            return Content("it works!");
        }
    }
}
```

~/Areas/Store/Controllers/ProductController.cshtml

Demo: Areas

Module 3: Controllers

Section 3: Advanced Controller Design

Lesson: Advanced Controller Design

Asynchronous Controller Action Methods

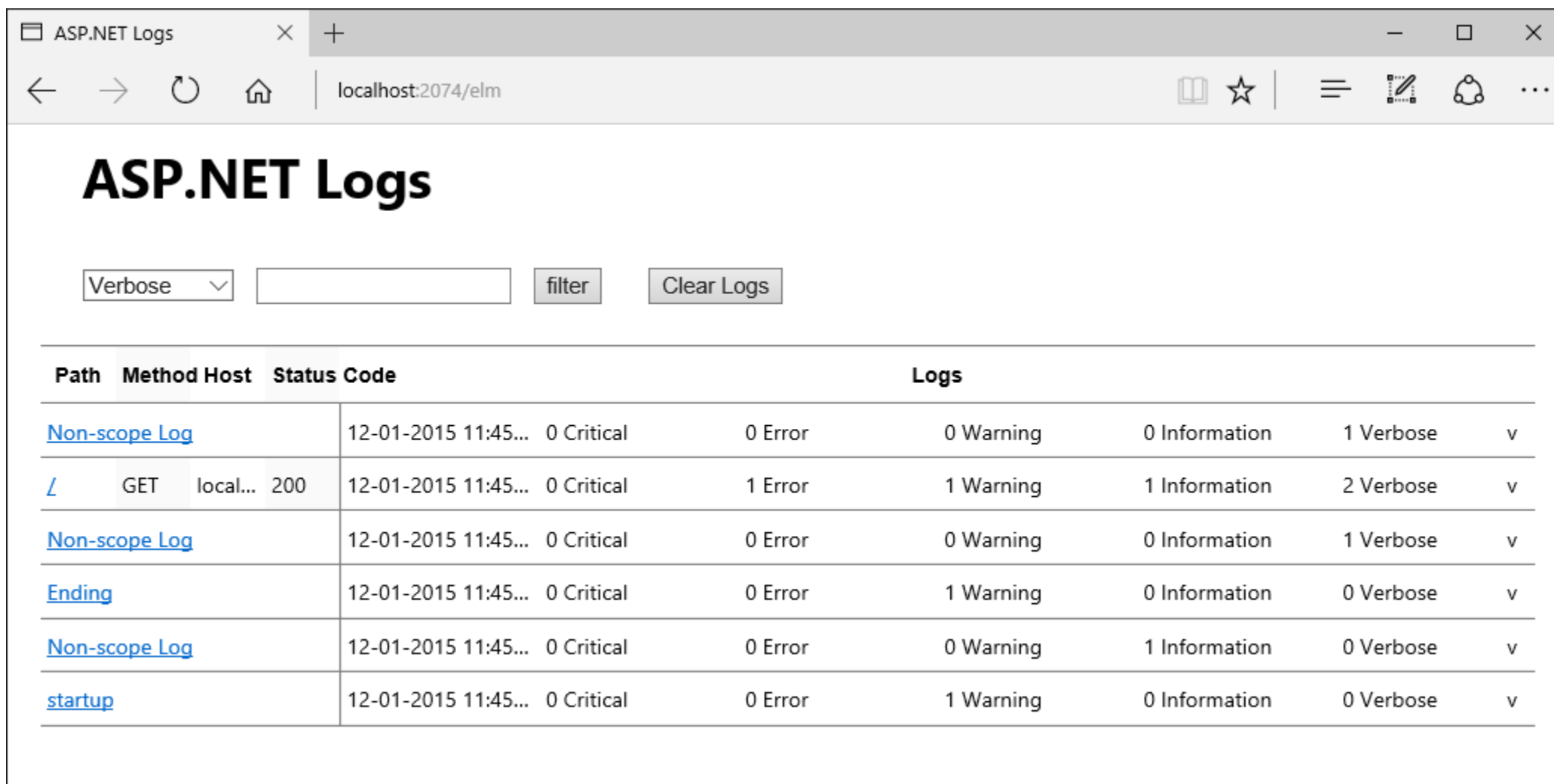
- ASP.NET MVC supports asynchronous action methods

```
public class PortalController : Controller {  
    public async Task<ActionResult> Index(string city)  
    {  
        NewsService newsService = new NewsService();  
        WeatherService weatherService = new WeatherService();  
        SportsService sportsService = new SportsService();  
  
        var newsTask = newsService.GetNewsAsync(city);  
        var weatherTask = weatherService.GetWeatherAsync(city);  
        var sportsTask = sportsService.GetScoresAsync(city);  
  
        await Task.WhenAll(newsTask, weatherTask, sportsTask);  
  
        PortalViewModel model = new PortalViewModel  
        {  
            newsTask.Result,  
            weatherTask.Result,  
            sportsTask.Result  
        };  
        return View(model);  
    }  
}
```

Sync vs. Async Action Methods

- Use **synchronous** methods when:
 - The operations are simple or short-running
 - Simplicity is more important than efficiency
 - The operations are primarily CPU operations instead of operations that involve extensive disk or network overhead
- Use **asynchronous** methods when:
 - You are calling services
 - The operations are network-bound or I/O-bound instead of CPU-bound
 - Parallelism is more important than simplicity of code
 - Enabling cancelation of a long-running request

Error Logging Middleware (ELM)



The screenshot shows a web browser window titled "ASP.NET Logs" with the address bar at "localhost:2074/elm". The page has a search bar with a dropdown set to "Verbose", a "filter" button, and a "Clear Logs" button. Below is a table with columns: Path, Method, Host, Status, Code, and Logs. The Logs column contains a detailed breakdown of log levels and counts.

Path	Method	Host	Status	Code	Logs
Non-scope Log				12-01-2015 11:45...	0 Critical 0 Error 0 Warning 0 Information 1 Verbose v
/	GET	local...	200	12-01-2015 11:45...	0 Critical 1 Error 1 Warning 1 Information 2 Verbose v
Non-scope Log				12-01-2015 11:45...	0 Critical 0 Error 0 Warning 0 Information 1 Verbose v
Ending				12-01-2015 11:45...	0 Critical 0 Error 1 Warning 0 Information 0 Verbose v
Non-scope Log				12-01-2015 11:45...	0 Critical 0 Error 0 Warning 1 Information 0 Verbose v
startup				12-01-2015 11:45...	0 Critical 0 Error 1 Warning 0 Information 0 Verbose v

Microsoft.AspNetCore.Diagnostics.Elm

ELM Logs

ASP.NET Logs									
Path	Method	Host	Status Code	Logs					
Non-scope Log				12-01-2015 11:45...	0 Critical	0 Error	0 Warning	0 Information	1 Verbose v
/	GET	local...	200	Date	Time	Name	Severity	State	Error
				12/01/15	11:45:29	Microsoft.AspNet.Hosti...	Verbose	Beginning Request Id: 8...	
				12/01/15	11:45:29	Microsoft.AspNet.Diagn...	Verbose	Beginning Request...	
				12/01/15	11:45:29	Microsoft.AspNet.Loade...	Verbose	Sending Response:...	
				12/01/15	11:45:29	ELMDemo.Startup	Verbose	Beginning He...	
				12/01/15	11:45:29	ELMDemo.Startup	Information	Hello world!	
				12/01/15	11:45:29	ELMDemo.Startup	Error	This is an Error!	
				12/01/15	11:45:29	ELMDemo.Startup	Warning	Warnings loo...	
				12/01/15	11:45:29	ELMDemo.Startup	Verbose	Completed H...	
				12/01/15	11:45:29	ELMDemo.Startup	Verbose	Beginning ve...	
				12/01/15	11:45:29	ELMDemo.Startup	Verbose	some verbos...	
				12/01/15	11:45:29	ELMDemo.Startup	Verbose	Completed v...	

ELM – Request Details

ASP.NET Logs		
localhost:2074/Elm/4e75bebd-b1b4-40bc-a7b9-a75be35bba2d		
Path	/	
Host	localhost:2074	
Content Type		
Method	GET	
Protocol	HTTP/1.1	
Headers	Variable	Value
	Connection	Keep-Alive
	Accept	text/html, application/xhtml+xml, image/jxr, */*
	Accept-Encoding	gzip, deflate, peerdist
	Accept-Language	en-US,en;q=0.5
	Host	localhost:2074
	User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2486.0 Safari/537.36 Edge/13.10586

Demo: Error Logging Middleware (ELM)

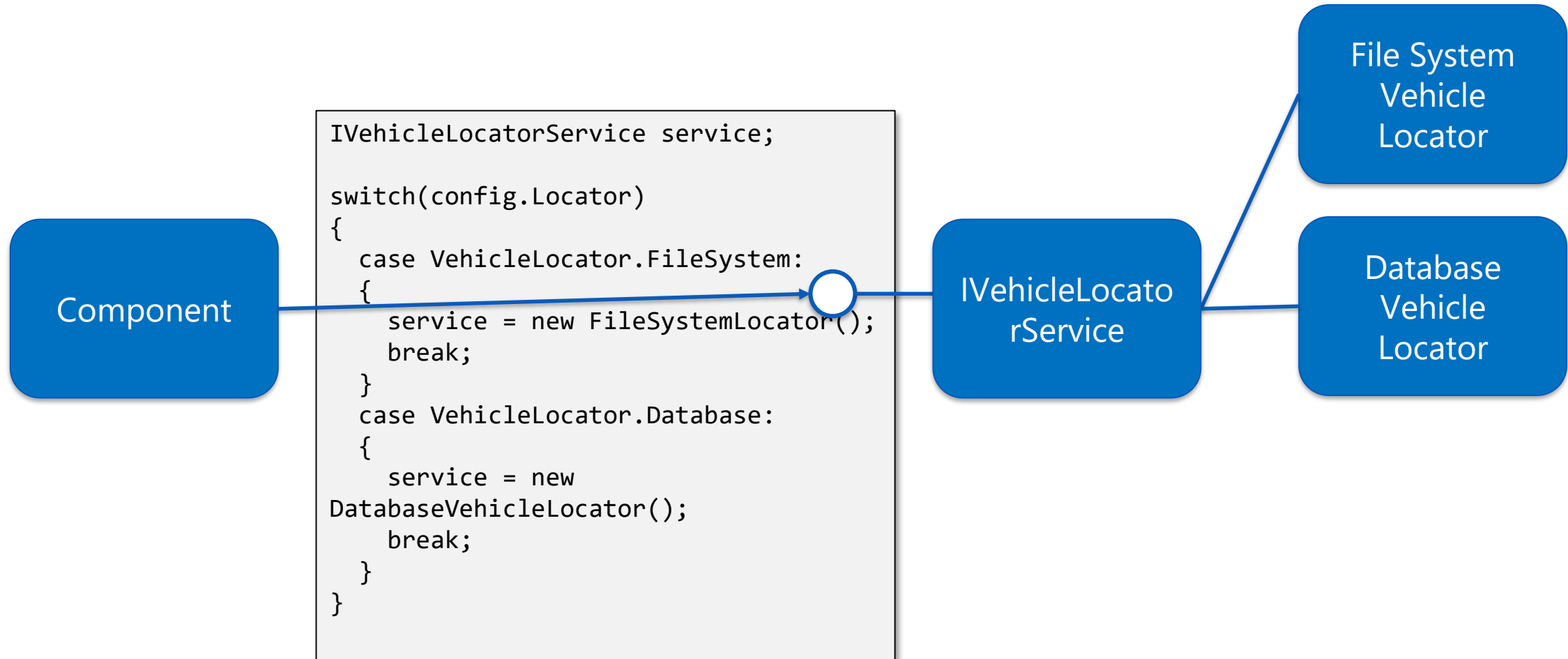
Module 3: Controllers

Section 4: Dependency Injection (DI)

Lesson: Overview

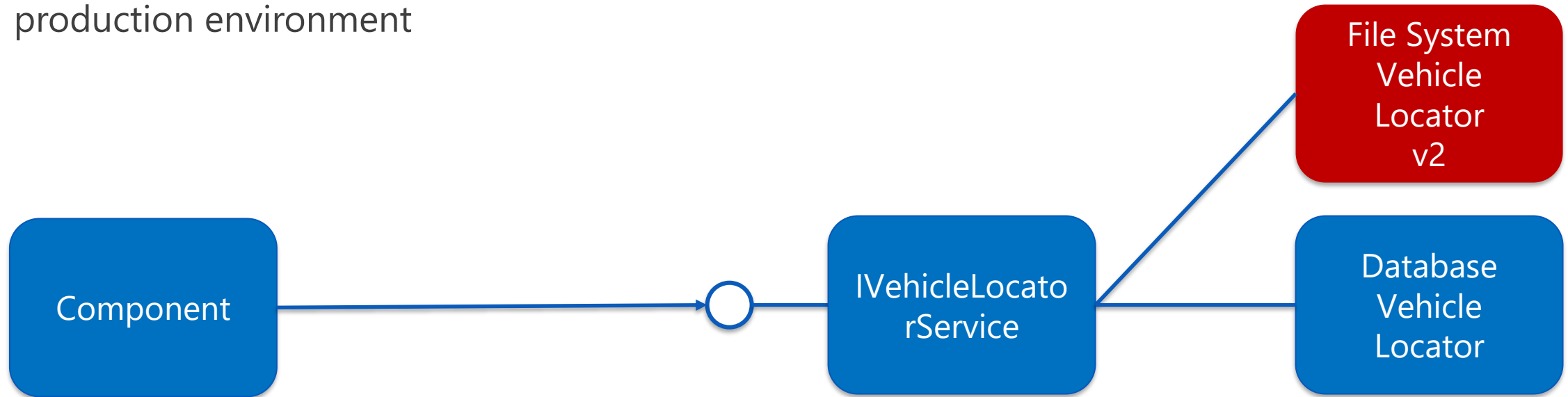
Without Dependency Injection (DI)

Tightly Coupled - Component determines which implementation to use



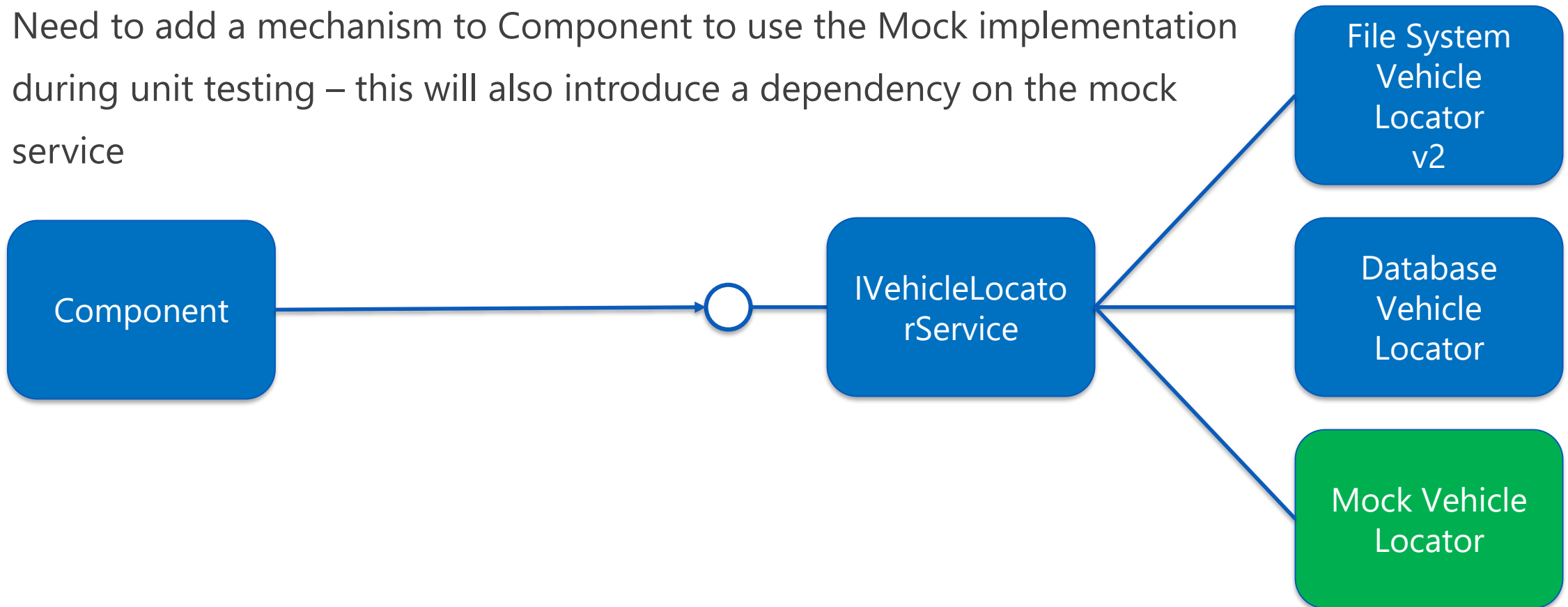
What's the Problem? (Part 1)

- Redeploy of new service that is tightly coupled means Component also needs to be rebuilt, retested, and redeployed
- Both components will be shipped/deployed, whereas only one may be required for the production environment



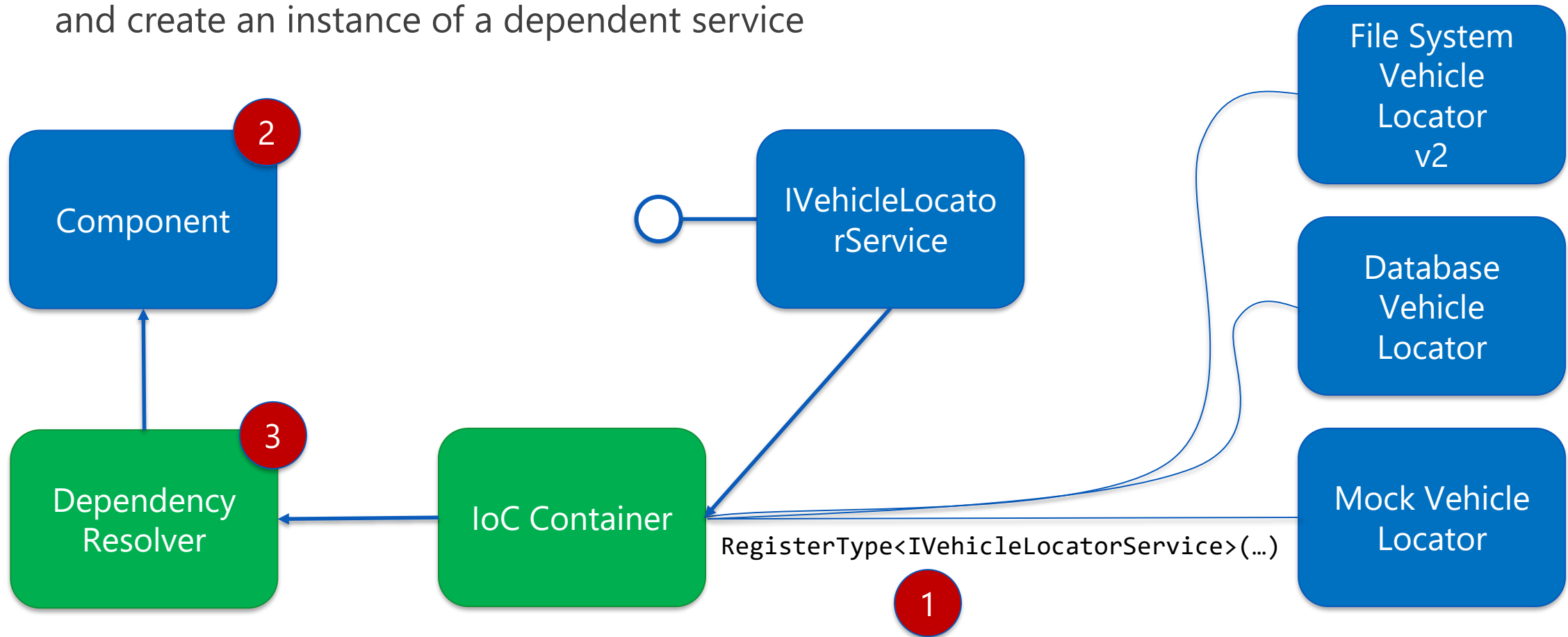
What's the Problem? (Part 2)

- Unit testing should be proving Component in isolation, and not rely on external resources such as databases or file systems
- Need to add a mechanism to Component to use the Mock implementation during unit testing – this will also introduce a dependency on the mock service



Inversion of Control

- DI is a design pattern which removes the responsibility of a component to know how to locate and create an instance of a dependent service



Framework – Provided DI in ASP.NET Core

```
1  public void ConfigureServices(IServiceCollection services)
2  {
3      // Add framework services.
4      services.AddEntityFramework()
5          .AddSqlServer()
6          .AddDbContext<ApplicationDbContext>(options =>
7              options.UseSqlServer(Configuration["Data:DefaultConnection:ConnectionString"]));
8
9      services.AddIdentity<ApplicationUser, IdentityRole>()
10         .AddEntityFrameworkStores<ApplicationDbContext>()
11         .AddDefaultTokenProviders();
12
13     services.AddMvc();
14
15     // Add application services.
16     services.AddTransient<IEmailSender, AuthMessageSender>();
17     services.AddTransient<ISmsSender, AuthMessageSender>();
18 }
```

DI is provided by ASP.NET Core, and is used to resolve Controllers as well as services

Framework services are set up in the ConfigureServices function in ASP.NET Core

Your own services can also be registered here – specifying Interface, Class, and Lifetime

Service Lifetimes and Registration Options

Transient

- Transient lifetime services are created each time they are requested. This lifetime works best for lightweight, stateless service

Scoped

- Scoped lifetime services are created once per request

Singleton

- Singleton lifetime services are created the first time they are requested, and then every subsequent request will use the same instance. If your application requires singleton behaviour, allowing the services container to manage the service's lifetime is recommended instead of implementing the singleton design pattern and managing your object's lifetime in the class yourself

Instance

- Similar to Singleton, but the instance is created when added to the services container and reused throughout, whereas with a Singleton the instance is first created only when first requested for use

Types of DI

- Constructor Injection
- Setter Injection

```
public class CalcController : Controller
{
    //Inject via Setter
    [FromServices]
    0 references
    public ISmsSender SmsSender { get; set; }

    private readonly ILogger _logger;

    //Inject via Constructor
    0 references
    public CalcController(ILogger logger)
    {
        _logger = logger;
    }
}
```

Demo: Dependency Injection

Module 3: Controllers

Section 5: Caching

Lesson: Overview

Caching

- In-memory Caching
 - Stored in memory of a single server
- Distributed Caching
 - Shared by multiple servers
 - Used by server-farm hosted apps
- Response Caching
- Output Caching

In-Memory Caching

```
public Task Invoke(HttpContext httpContext)
{
    string cacheKey = "GreetingMiddleware-Invoke";
    string greeting;

    // try to get the cached item; null if not found
    // greeting = _memoryCache.Get(cacheKey) as string;

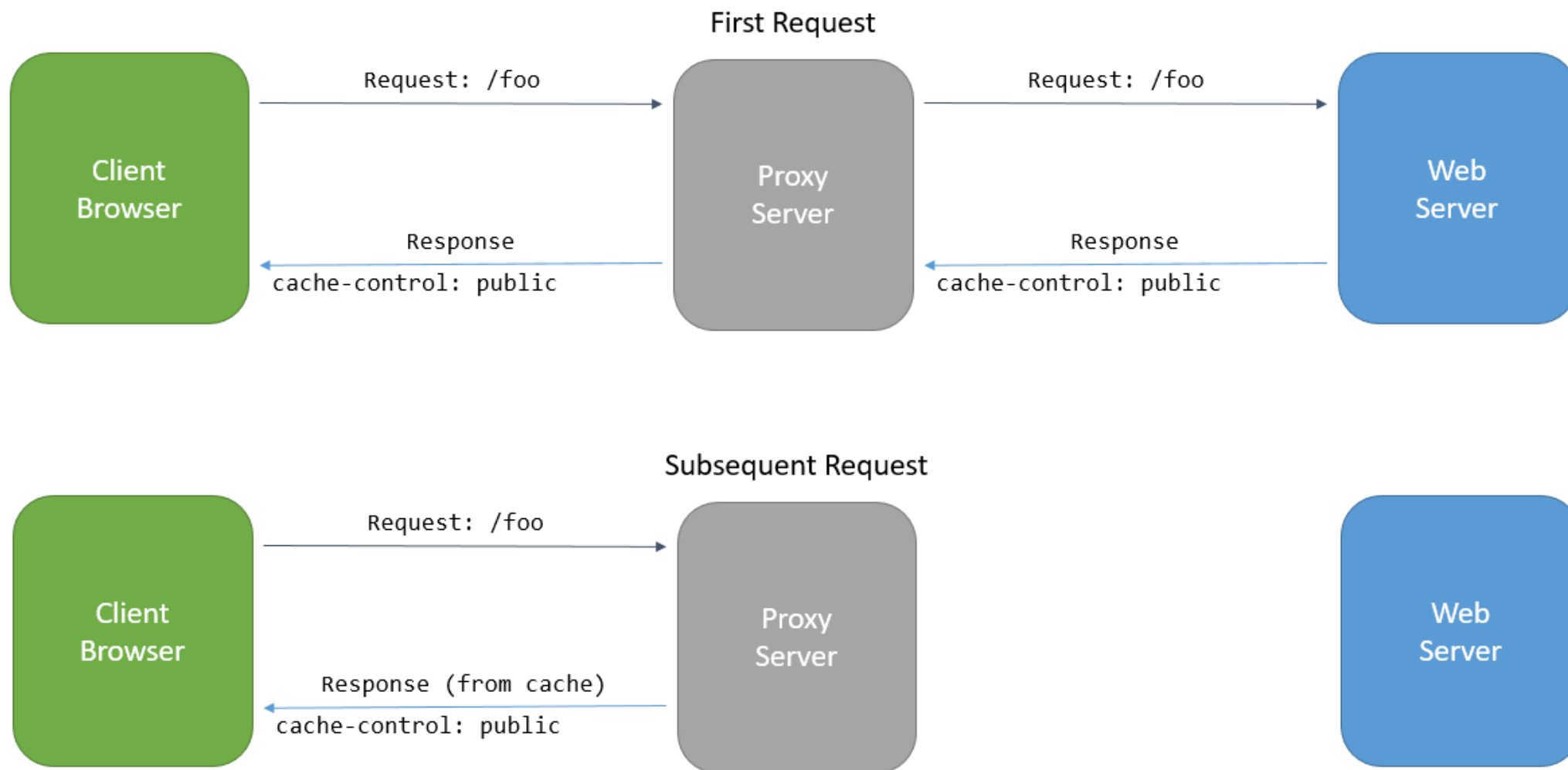
    // alternately, TryGet returns true if the cache entry was found
    if(!_memoryCache.TryGetValue(cacheKey, out greeting))
    {
        // fetch the value from the source
        greeting = _greetingService.Greet("world");

        // store in the cache
        _memoryCache.Set(cacheKey, greeting,
            new MemoryCacheEntryOptions()
                .SetAbsoluteExpiration(TimeSpan.FromMinutes(1)));
        _logger.LogInformation($"{cacheKey} updated from source.");
    }
    else
    {
        _logger.LogInformation($"{cacheKey} retrieved from cache.");
    }

    return httpContext.Response.WriteAsync(greeting);
}
```

Example of
Caching used by
Middleware

Response Caching



Demo: In-Memory Caching

Controller Best Practices

- Keep controllers lightweight
 - Use filters
- High Cohesion
 - Make sure all actions are closely related
- Low Coupling
 - Controllers should know as little about the rest of the system as possible
 - Simplifies testing and changes
 - Repository pattern
 - Wrap data context calls into another object

Demo: URL Configuration

Module Summary

- Controller and its role in MVC pattern
- Controller development
- Action methods
- Action filters
- Asynchronous controller actions
- Error-Logging Module
- Dependency Injection
- Caching
- Controller best practices



Lab: Controllers



