Web Presentation Layer

Introduction to ASP.NET

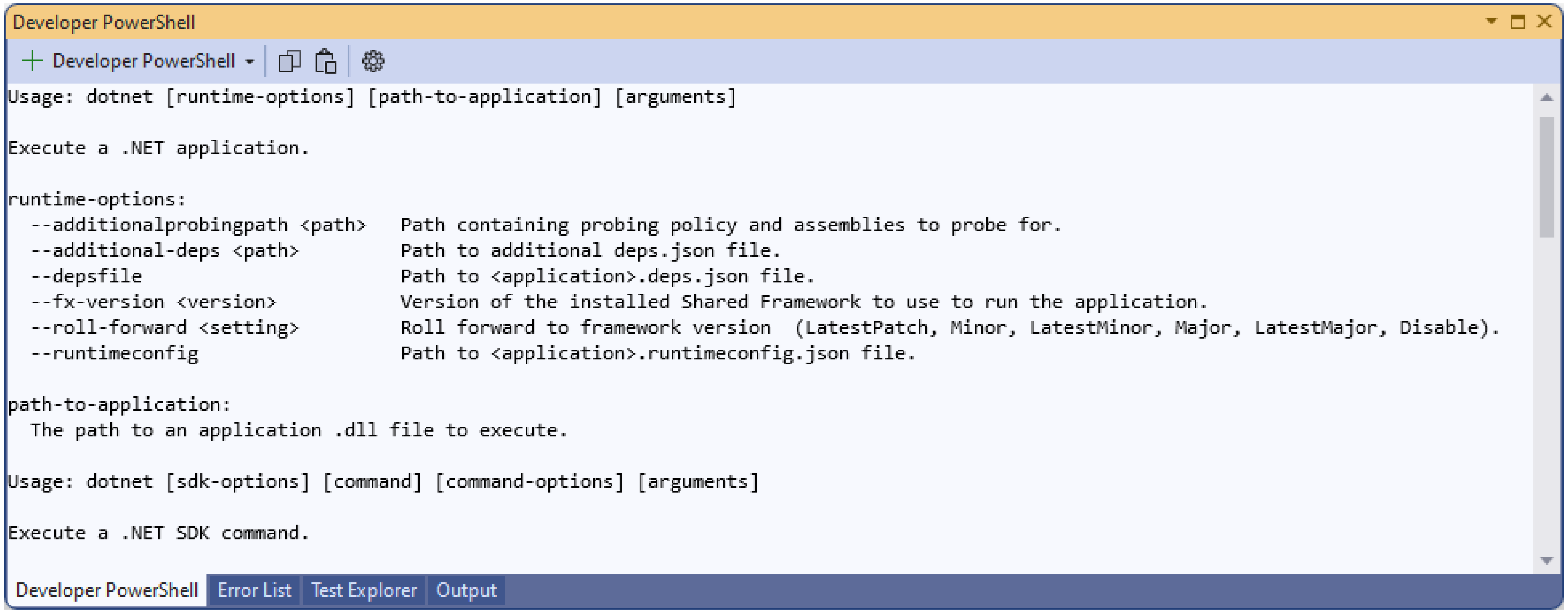
- This section gives an overview of ASP.NET:
  - ASP.NET Core
  - DotNet Command Line
  - ASP.NET Model View Controller
  - Model View Controller
  - Create ASP.NET Core Project
  - ASP.NET Core MVC Project

ASP.NET Core

- ASP.NET Core introduced a modular version of Web Applications for .NET
  - This was a work in progress for a long time!
  - Works on Windows, Linux and Mac
  - Configure application for specific .NET components
- There is a set of Command Line tools for working with .NET Core on Windows, Linux and Mac
  - IDE tools are evolving to support .NET Core

DotNet Command Line

- Install .NET 8.0 and then view help:



```
Usage: dotnet [runtime-options] [path-to-application] [arguments]

Execute a .NET application.

runtime-options:
  --additionalprobingpath <path>   Path containing probing policy and assemblies to probe for.
  --additional-deps <path>         Path to additional deps.json file.
  --depsfile                       Path to <application>.deps.json file.
  --fx-version <version>           Version of the installed Shared Framework to use to run the application.
  --roll-forward <setting>         Roll forward to framework version  (LatestPatch, Minor, LatestMinor, Major, LatestMajor, Disable).
  --runtimeconfig                  Path to <application>.runtimeconfig.json file.

path-to-application:
  The path to an application .dll file to execute.

Usage: dotnet [sdk-options] [command] [command-options] [arguments]

Execute a .NET SDK command.
```

- Create project using:

```
>dotnet new console
```

Creates Console Project

```
>dotnet new mvc –auth Individual
```

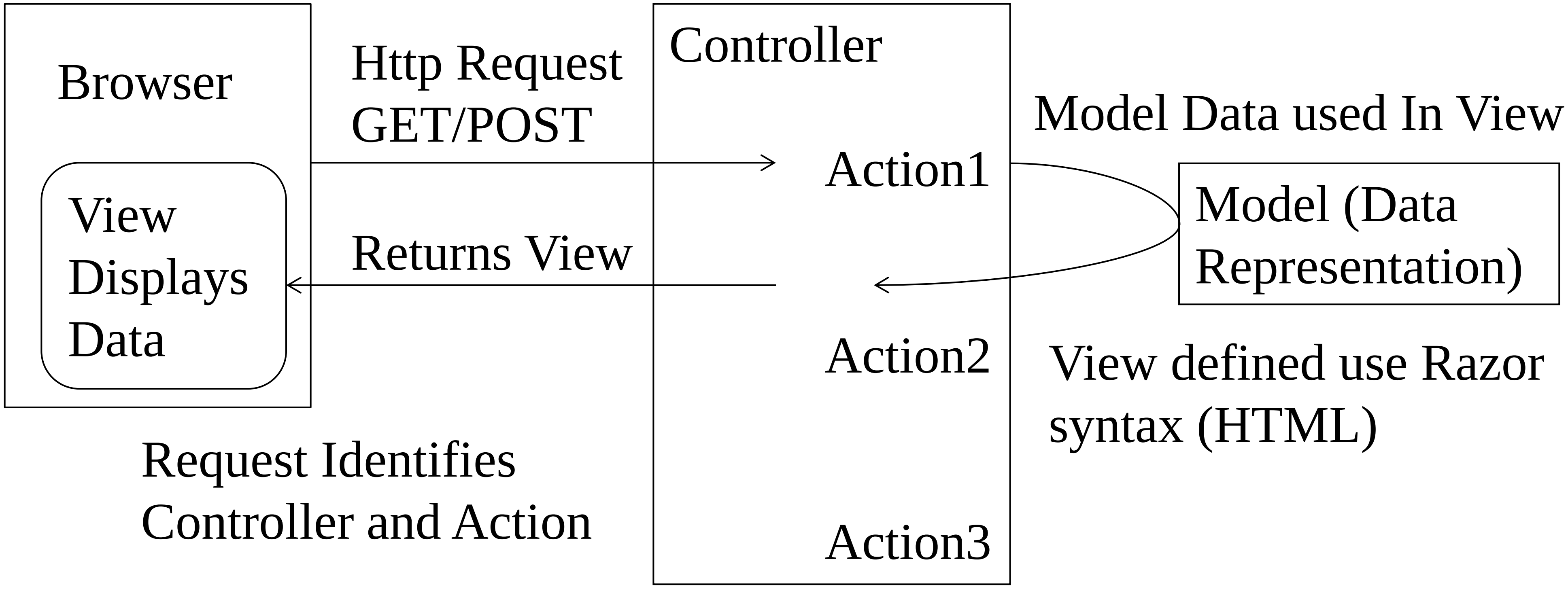
Creates MVC Project with Authentication

ASP.NET Model View Controller

- The .NET Framework and tools support various versions of ASP.NET MVC.
  - The models can be:
    - A POCO type (Entity Data Model)
    - .NET 5/6/7/8 supports C#9 records
  - Views are defined as pages using the Razor syntax
  - HTML helpers used for Display or Validation
    - Many HTML Tag Helpers (ASPNET Core)
  - Controllers define Actions for GET and POST
    - Determine what is displayed within View
  - Standard look provided by Layout

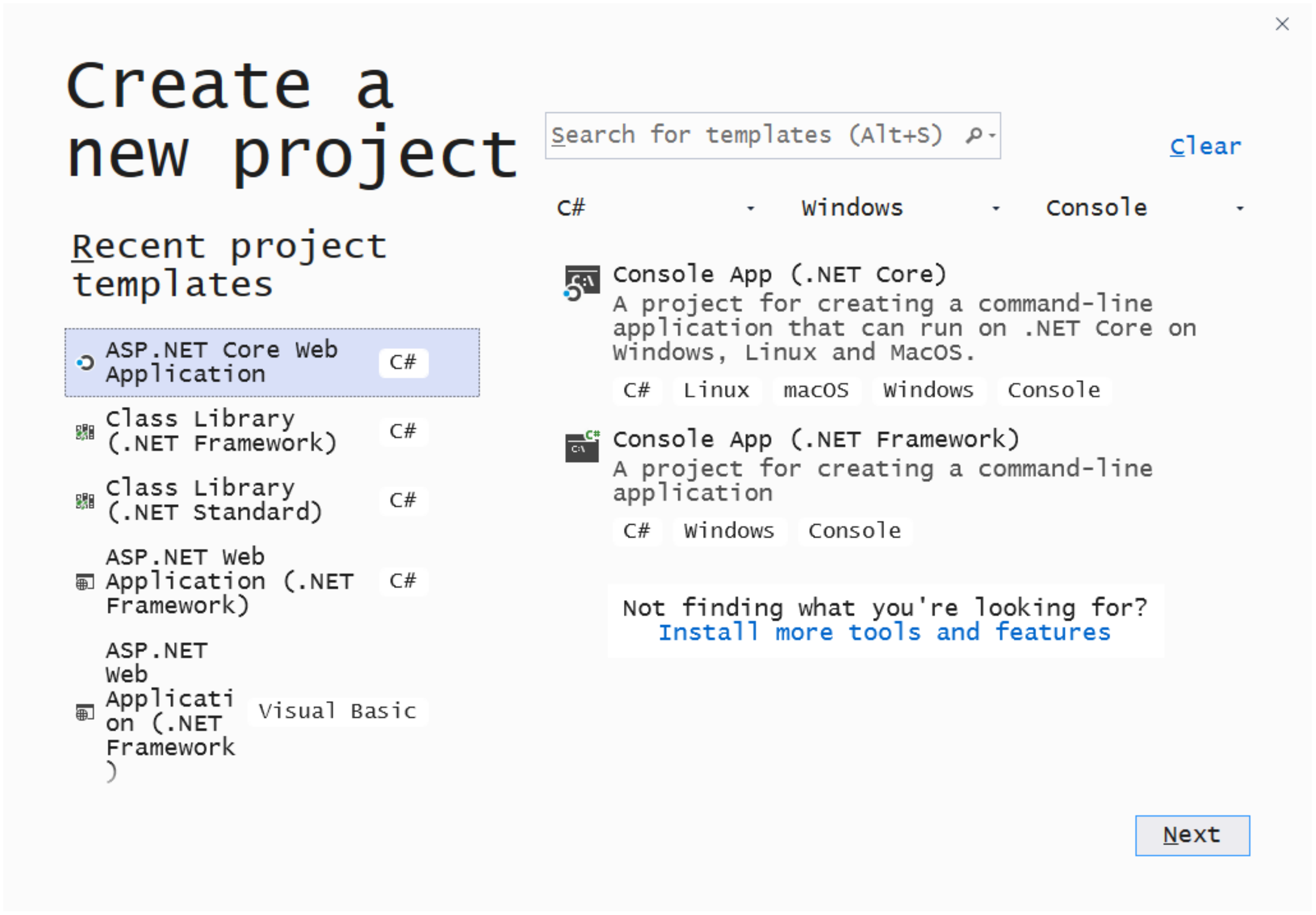
Model View Controller

- MVC Usage:



Create ASP.NET Core Project

- ASP.NET Web Application:



Create a new project

Recent project templates

ASP.NET Core Web Application	C#
Class Library (.NET Framework)	C#
Class Library (.NET Standard)	C#
ASP.NET Web Application (.NET Framework)	C#
ASP.NET Web Application (.NET Framework)	Visual Basic

Search for templates (Alt+S)

Clear

C# Windows Console

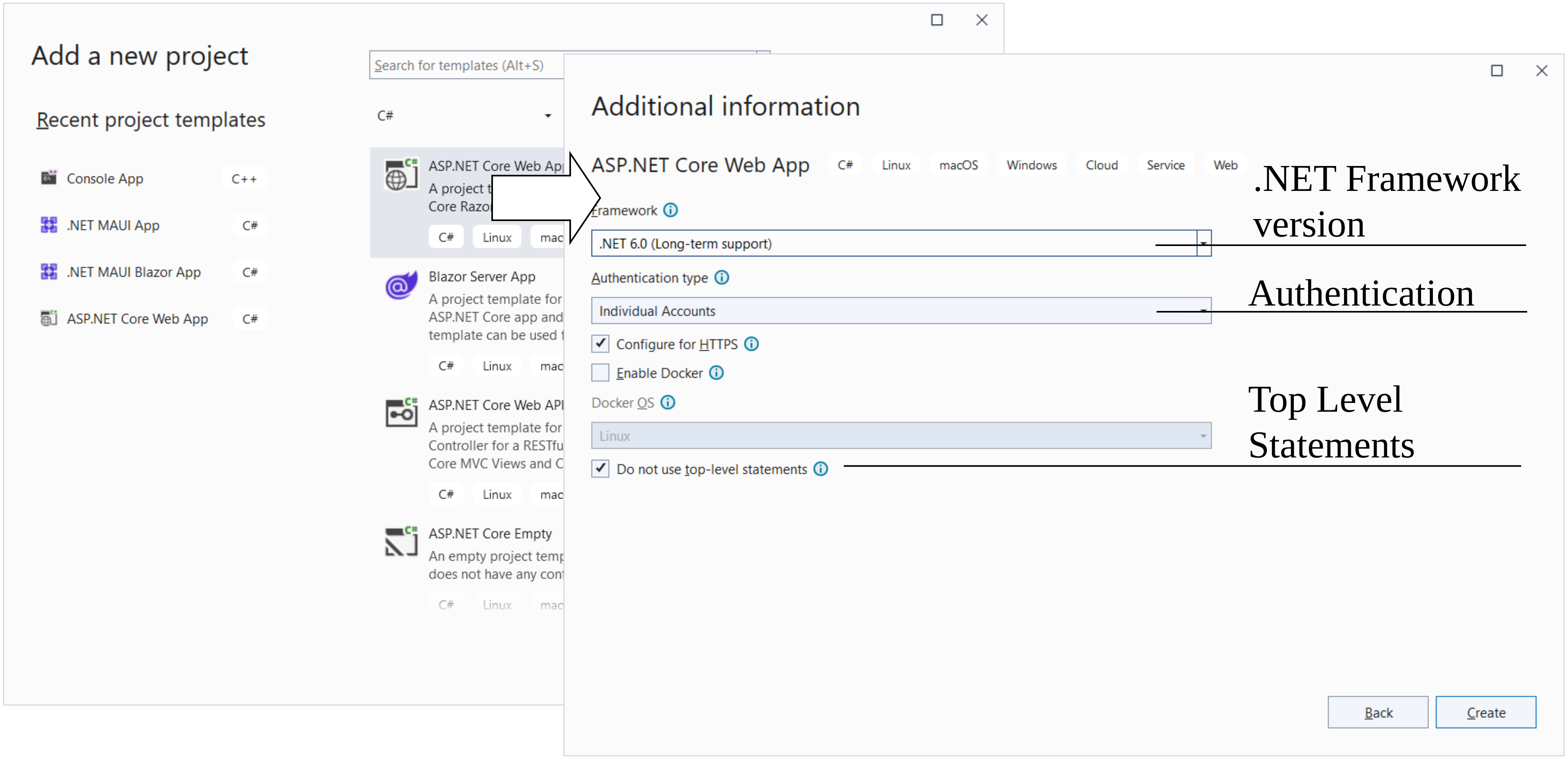
Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Console

Console App (.NET Framework)
A project for creating a command-line application
C# Windows Console

Not finding what you're looking for?
Install more tools and features

New Project Dialog ASPNET Core

- Range of projects available:



Add a new project

Search for templates (Alt+S)

Recent project templates

Console App	C++
.NET MAUI App	C#
.NET MAUI Blazor App	C#
ASP.NET Core Web App	C#

C#

ASP.NET Core Web Ap
A project t
Core Razo
C# Linux mac

Blazor Server App
A project template for
ASP.NET Core app and
template can be used f
C# Linux mac

ASP.NET Core Web API
A project template for
Controller for a RESTfu
Core MVC Views and C
C# Linux mac

ASP.NET Core Empty
An empty project temp
does not have any con
C# Linux mac

Additional information

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Framework: .NET 6.0 (Long-term support)

Authentication type: Individual Accounts

☑ Configure for HTTPS

☐ Enable Docker

Docker OS: Linux

☑ Do not use top-level statements

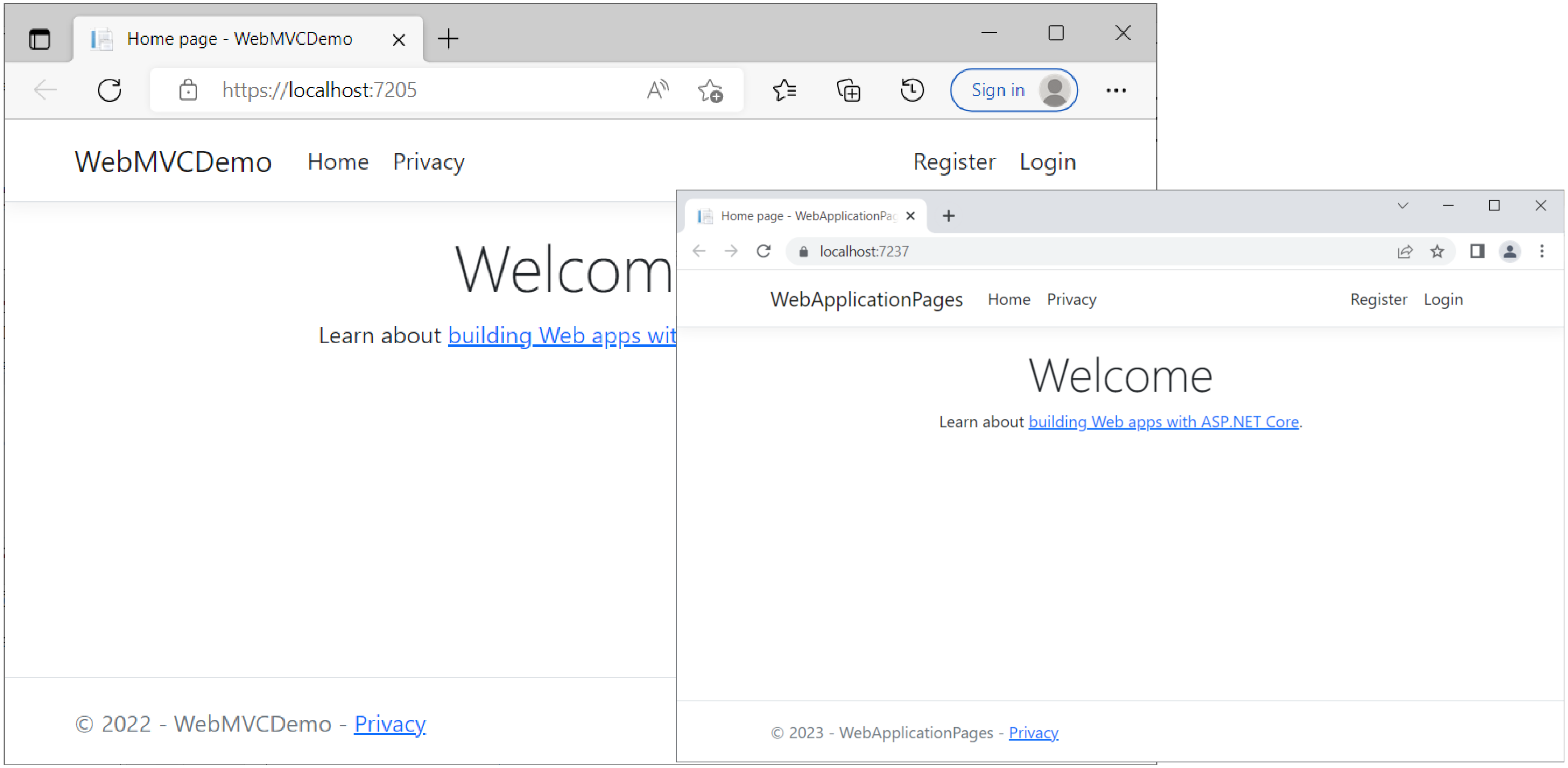
Back Create

ASPNET Core MVC Project

- The Standard MVC Project contains:
  - Controllers
    - Responses provided to Actions
  - Views
    - Views Display Model Data
    - Shared Folder contains Layout
  - Model folder for custom data definition
  - Services folder for Application Services

Project Home Page ASP.NET Core

- The project template provides many features:



Welcom

Learn about building Web apps wit

© 2022 - WebMVCDemo - Privacy

Home page - WebApplicationPa
localhost:7237
WebApplicationPages Home Privacy
Register Login

Welcome

Learn about building Web apps with ASP.NET Core.

© 2023 - WebApplicationPages - Privacy

Introduction to ASP.NET - Summary

- This section gave an overview of ASP.NET:
  - ASP.NET Core
  - DotNet Command Line
  - ASP.NET Model View Controller
  - Model View Controller
  - Create ASP.NET Core Project
  - ASP.NET Core MVC Project

Web Application Essentials

- This section covers many core application features:
  - Configuration Introduction
  - Configuration
  - Trivial Web Application
  - MVC and Routing
  - Html Helpers vs Tag Helpers
  - View Imports
  - ActionLink

Configuration Introduction

- The layout of Configuration for .NET applications has changed considerably over the years
  - Early .NET Core applications separated configuration into a Startup class
  - .NET 8 now gives two options for this configuration (Program class):
    - Top-level statements, or
    - Statements within a ‘main’ method

Configuration

- The approach taken since .NET Core is to use a pluggable stack:
  - Earlier versions of ASP.NET presumed a stack based on Web Server - IIS
  - ASP.NET Core allows pluggable and lightweight stack
  - Add only required features for web application
    - Some items require both adding and then to say that it is to be used!

Trivial Web Application

- Creation of trivial Web Applications:

Using top-level statements

```
ŵẳś čụîľđêś   ŴêčAřřľîçẳʧîộŋ CśêẳʧêBụîľđêś ẳśĝș

ŵẳś ẳřř   čụîľđêś Bụîľđ

ẳřř ŇẳřĞêʧ               Ȟêľľộ Ŵộśľđ

ẳřř Ŗụŋ
```

Build WebApplication

Static Response

Run Web Application

- Previously some configuration was seen explicitly like ‘UseRouting()’ and ‘UseEndpoints()’
  – These are called as part of the build

MVC and Routing

- If an MVC application is required the Application Builder needs to enable controllers
- Routing can be defined using:

Core 3.1

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

Core 6.0

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

app.MapRazorPages();
```

Html Helpers vs Tag Helpers

- Html helpers are .NET methods used to generate html (preceded by @ for razor syntax), e.g.

```
@Html.TextBoxFor(model => model.Name, new { @class = "prominent" })
```

- Tag helpers appear as attributes within html, therefore looks more familiar to client side developers, e.g.

```
<input asp-for="Name" class="prominent" />
```

- Both generate:

```
<input class="prominent" type="text" id="Name" name="Name" value="Albert" />
```

  – Although attributes may be in different order!
  – Some tag helpers may not be used as self closing elements

View Imports

- The _viewimports.cshtml file contains using directives used by the views:

```
@using WebAppCoreNetCore
@using WebAppCoreNetCore.Models
@using WebAppCoreNetCore.Models.AccountViewModels
@using WebAppCoreNetCore.Models.ManageViewModels
@using Microsoft.AspNetCore.Identity
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper *,WebAppCoreNetCore
```

Additional Tag Helpers can also be made available to Views

Assembly (current assembly)!

ActionLink (Html Helpers)

- ActionLink defines a link to another view via an Action:

```
Html.ActionLink("Data", "Index", "Data")
```

Link Text | Action Name | Controller Name

- Many scaffolding views provide links to another action/view
  - Appropriate data needs to be passed to Action

```
@Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
```

Uncomment and Edit entries as appropriate

Link (ASP.NET Core)

- Use <a> to define a link to another view via an Action:

```
<a asp-controller="Home" asp-action="Index">Home</a>
```

Controller Name

Action Name

Link Text

- Where links are required to pass additional information Tag helpers can be used:

```
<a asp-action="Edit" asp-route-id="@item.Id">Edit</a>
```

Tag Helper

Razor evaluation for id to be passed as parameter

Web Application Essentials - Summary

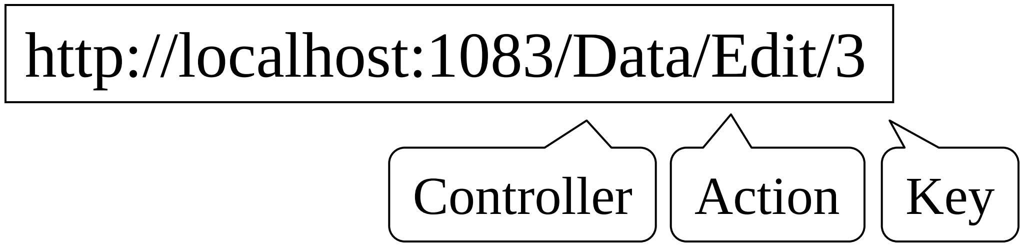
- This section covered many core application features:
  - Configuration Introduction
  - Configuration
  - Trivial Web Application
  - MVC and Routing
  - Html Helpers vs Tag Helpers
  - View Imports
  - ActionLink

Model View Controller

- This section introduces the MVC approach:
  - Routing
  - Controller
  - Add Controller
  - Add View
  - View Models

Routing

- ASP.NET WebForms mapped URLs to ASPX pages (addressing files) or Handlers
- MVC maps URLs to Controller and Actions
  - Actions are executed on the Controller and an appropriate View displayed


http://localhost:1083/Data/Edit/3

Controller

- Earlier versions of MVC technology required Controller to inherit from a special base class
- Core Controllers can be a simple class:

```
public class SimpleController
{
        public IActionResult SomeText()
        {
            return new ContentResult() { Content = "The answer is 42" };
        }
}
```

Explicit creation of Result

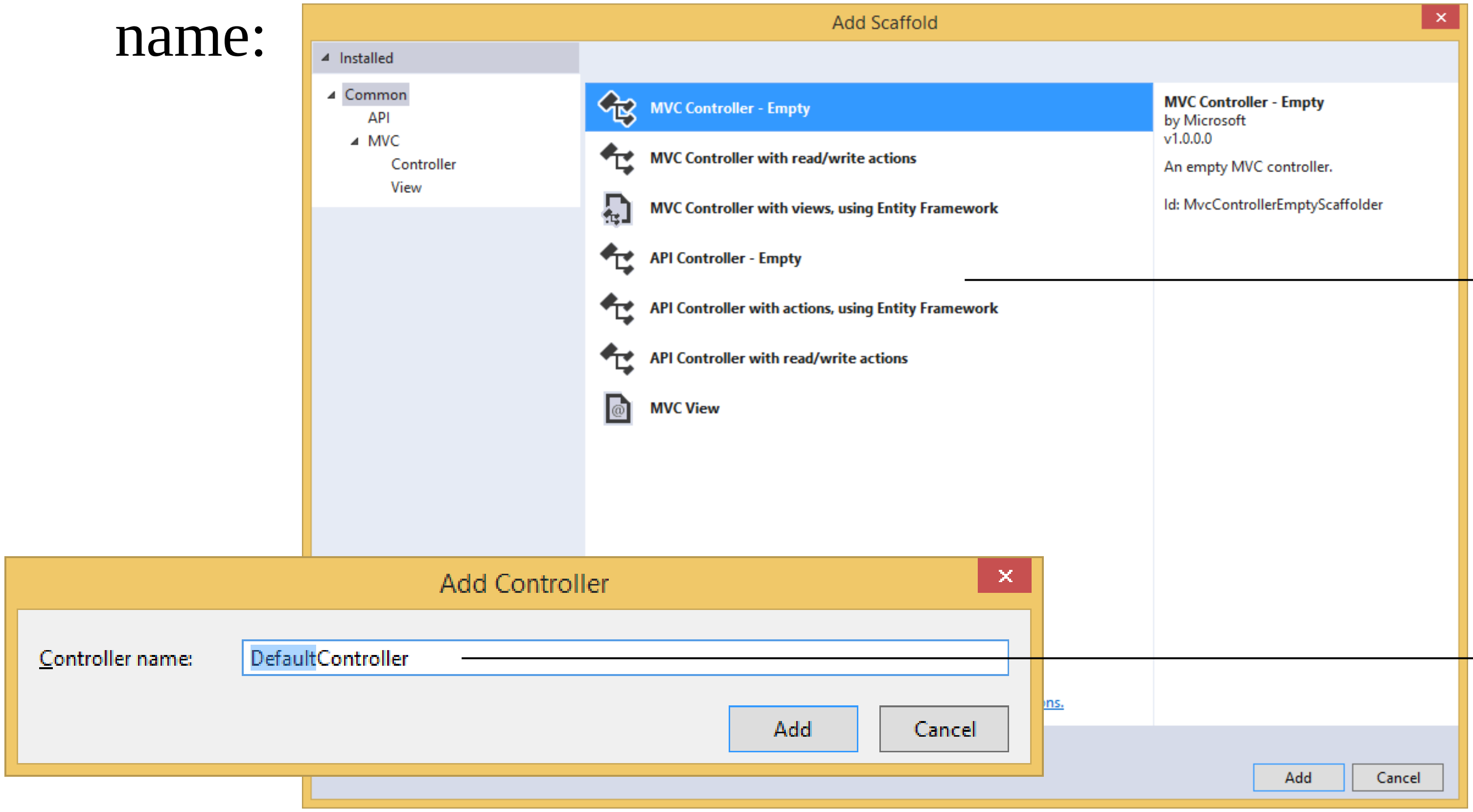
- This would respond to:

Controller Action

```
http://localhost:31637/Simple/sometext
```

Add Controller

- New Controller can be added by right clicking on ‘Controllers’
- Name the controller, but leave ‘Controller’ at the end of the name:


Add Scaffold

Installed
- Common
  - API
  - MVC
    - Controller
    - View

MVC Controller - Empty
MVC Controller with read/write actions
MVC Controller with views, using Entity Framework
API Controller - Empty
API Controller with actions, using Entity Framework
API Controller with read/write actions
MVC View

MVC Controller - Empty
by Microsoft
v1.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder

Add Controller

Controller name: DefaultController

Choose Scaffolding Option

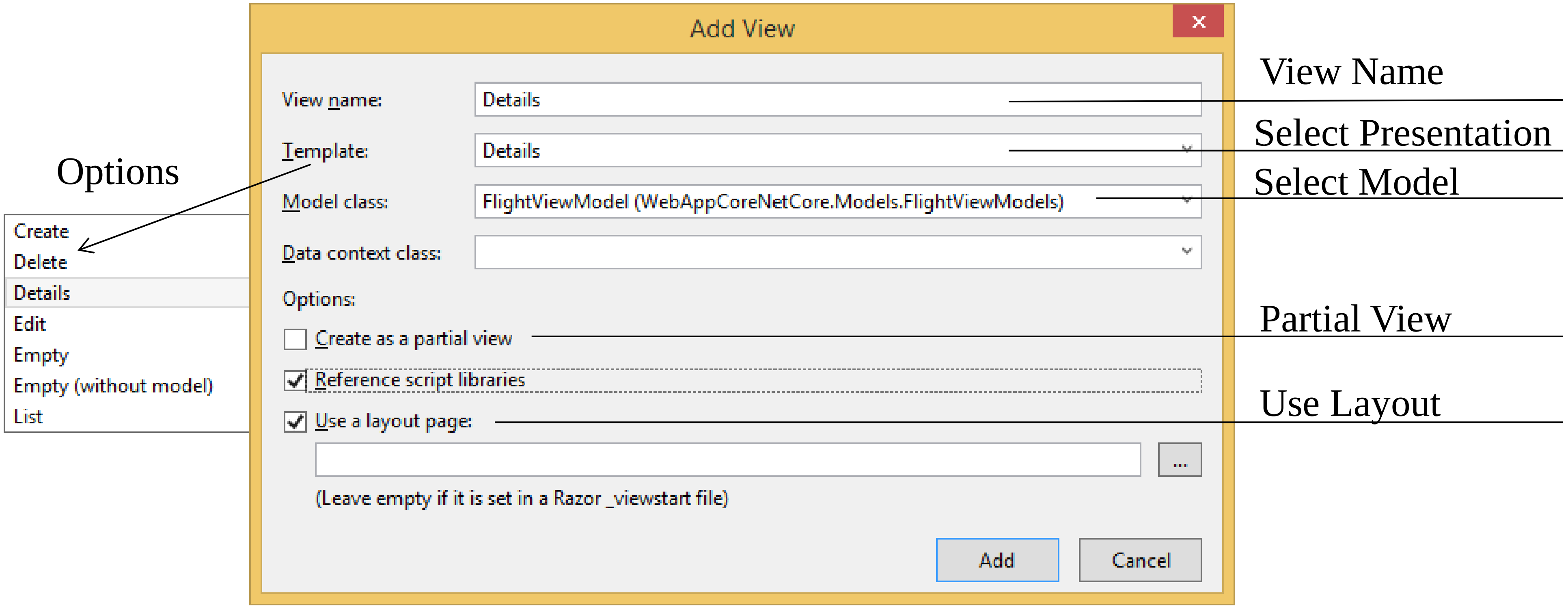
Name Controller

Adding View

- One of the conventions used within MVC is for a View for an action to have the same name as the Action and be within a folder named after the controller.
  - Alternatively the view can be within a Shared folder
- The View can have an alternative name, however it needs to be identified
- A View can be added as a simple View (Add New Item) or a strongly typed view

Add View

- A new View can be added by right clicking on the Action
  - View can be complete Page or Partial View for a Control



View name: Details

Template: Details

Model class: FlightViewModel (WebAppCoreNetCore.Models.FlightViewModels)

Data context class:

Options:

☐ Create as a partial view

☑ Reference script libraries

☑ Use a layout page:

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

MVC Controller Actions and Views

- Actions are typically paired with Views
  - Action methods within the Controller respond to GET and POST
    - The IActionResult defines the data to be returned
    - Additional information can be passed to a view
      - ViewData or ViewBag (from MVC 3)
  - Views are provided as Razor Views (cshtml files)
    - Views are defined using HTML, Tag Helpers and HTML helpers

MVC Controller

```
public class DataController : Controller
{
     // GET: /Data/
     public IActionResult Index() { … }
     // GET: /Data/Details/5
     public IActionResult Details(int id){ …}
     // GET: /Data/Create
     public IActionResult Create(){ …}
     // POST: /Data/Create
     [HttpPost]
     public IActionResult Create(PassengerDetail pd){ … }
     // GET: /Data/Edit/5
     public IActionResult Edit(int id){ … }
     // POST: /Data/Edit/5
     [HttpPost]
     public IActionResult Edit(int id, PassengerDetail pd){ … }
}
```

Default HTTP Action GET

Overloaded Methods for GET and POST

MVC Action POST

- When data is posted back the action should check ModelState:

```
[HttpPost]
public IActionResult Edit(int id, PassengerDetail pd)
{
   if (ModelState.IsValid)
   {
      try
      {
         …
         return RedirectToAction("Index");
      }
      catch
      {   return View(pd); }
   }else
   {   return View(pd); }
}
```

Checks Post Data

Redirect on Success

Return to Page on Failure

Return to Page on Failure

Model Binding

- Model Binding within MVC can provide automatic population of objects
  - Data Posted back could be obtained from Request
  - Parameters on Postback Actions can be used
  - Parameters as objects will be populated
    - From previous slides:

Fields populate Properties

```
public ActionResult Edit(int id, PassengerDetail pd)
```

View Models

- The use of model binding attributes allows restriction of posted data bound to objects
  - Whilst this works it is potentially error prone
- Another common approach is to use a Pattern using View Models
  - A ViewModel is a class which contains the properties to be mapped into a view and back from an edit view
  - Prevents over-posting!

Model View Controller - Summary

- This section introduced the MVC approach:
  - Routing
  - Controller
  - Add Controller
  - Add View
  - View Models

Razor Syntax

- This section gives an overview of Razor
  - Razor View Start
  - Razor Syntax
  - Razor Syntax Usage

Razor View Start

- The start page for the Razor Engine is:

```
_ViewStart.cshtml
```

  - This file contains (acts like ASP.NET master page):

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

- The content for the page (view) is displayed using:

```
@RenderBody()
```

Main body of View

```
@RenderSection("scripts", required: false)
```

Razor Syntax

- Razor makes use of ‘@’ symbol extensively within views:
  - Within the html @ precedes server side code
  - @* … *@ denotes comments
  - Use of @ also provides html encoding
  - @ can be included by ‘escaping’ i.e. @@
  - Razor engine recognises email addresses
- Whilst Razor allows adding server side code to the View, this should only be used for presentation purposes

Razor Syntax Usage (C#) continued

- Server side code within view:

Code Type	**Razor**	**Comment**

Block	@{
	        int val;
	}
		Code within block is treated as server side code. Variable can be used later within page.

Expression (Implicit)	<p>@item.Name</p>	Html <p> will enclose value resulting from evaluating the expression after @. Use of symbols will end an implicit expression (generics cannot be used)
Expression (Explicitly)	<p>@(val*val)</p>	Html <p> will enclose value resulting from evaluating the expression within the (). Symbols for operators may be used
Looping	@for, @foreach, @while, @do while	Use of the looping constructs are provided within the razor syntax
Flow control	@if, @select	The usual flow control statements

Razor Syntax Usage (C#) continued

- Server side code within view:

Code Type	**Razor**	**Comment**

Text and Markup	@while(ok)
	{
	   <tr><td>@item.Name</td></tr>
	}
		Loops are treated as server side ‘while’ but markup is rendered with expression evaluated to obtain Name property

Functions	@functions{
	    static int Square(int n)
	    { return n * n;
	    }
	}
		Functions and properties can be defined for use within the view

Razor Syntax - Summary

- This section gave an overview of Razor
  - Razor View Start
  - Razor Syntax
  - Razor View Code

State Management

- This Section gives an introduction to State Management:
  - State Management Introduction
  - State Options
  - Configuring Session
  - Using Session

State Management Introduction

- ASP.NET Core provides mechanisms for many aspects of state with different scopes
- ASP.NET Core requires configuring the features required
  - Thus if not configured it is not provides
  - Some features only via the Context (HttpContext)
    - Context object created for each request
    - Available either on Controller (HttpContext)
- Ideally work in a stateless manner!

State Options

- Some state options available on Controller

State (Controller)	**Description**
ViewData (ViewBag)	Data visible within action and view (current request)
TempData	Data visible to action and view redirected to

- Some state options available on Context

State (HttpContext)	**Description**
Session	User Session (duration of browser interaction)

Configuring Session

```
…
builder.Services.AddControllersWithViews();

builder.Services.AddSession( options =>
        {
            options.IdleTimeout = TimeSpan.FromMinutes(20);
        });
…
```

Add Session Capability

Configure Session Timout

```
var app = builder.Build();
…
app.UseSession();
…
```

Enable Session

Using Session

- The **Session** property of HttpContext may be used as a collection:

```
HttpContext.Session.SetString("Data", "Hello");

string data = HttpContext.Session.GetString("Data");
```

- The Session uses caching 'IDistributedCache'.
- AddMemoryCache adds in-memory caching primarily for use during development and testing.
- An scalable distributed cache is the Redis cache
  - Install using Nuget
    - Microsoft.Extensions.Caching.Redis.Core
  - Redis 64 can be installed locally to try out this caching

State Management - Summary

- This Section gives an introduction to State Management:
  - State Management Introduction
  - State Options
  - Configuring Session
  - Using Session

Views and Partial Views

- This section gives an overview of Views:
  - Partial Views
  - Partial Tag Helper
  - View and Partial View
  - RenderAction
  - ViewComponent
  - Display and Editor Templates

Partial Views

- Partial Views allow creation of 'controls' which avoids duplication of code
- Creating Partial Views is very similar to creating Views
- To display a Partial View within a View use a Html helper
  - Specify action (current views Model passed by default):

    ```
    @Html.Partial("SomeData") or @await Html.PartialAsync("SomeData")
    ```
  - Can refine data to be passed from model (select from collection!):

    ```
    @Html.Partial("Create", Model.First(p => p.Name.Equals("Fred7")))
    ```
  - By default the view model is passed to the Partial View
- Alternatively use Html.RenderPartialAsync
  - Writes directly to response stream
  - Html.Partial returns a string wrapper, which could be stored and reused

Partial Tag Helper

- The partial tag helper allows inclusion of partial views with a view
- Alternative to Html Helpers:

```
<partial name="_SomeData" for="Data" />
```

Attribute	**Description**
name	Name of partial view (required)
fallback-name	Alternative name for a partial view if ‘name’ cannot be found
for	Model expression (passed as partial view model)
model	Model passed to partial view (cannot be used with for)
optional	Will result in ‘no-op’ if partial view is not found

Views and Partial Views - Summary

- This section gave an overview of Views:
  - Partial Views
  - Partial Tag Helper
  - View and Partial View
  - RenderAction
  - ViewComponent
  - Display and Editor Templates

Validation

- This Section introduces validation:
  - Validation Introduction
  - Data Annotations and Validation
  - DataAnnotations
  - Displaying Validation Messages

Validation Introduction

- Validation can be provided both client side and server side
  - Client side validation can reduce the number of round trips to the server
  - Server side validation is required as client side validation cannot be guaranteed
- Server side validation can be checked by use of ‘ModelState.IsValid’, presuming parameter model binding
- Alternatively use:
  - TryUpdateModelAsync and TryValidateModel

Data Annotations and Validation

- Client side validation messages are provided by Html helpers or Tag helpers
- Property validation and messages to be displayed can be defined using Data Annotations
- Data Annotations can be applied directly to properties of the data typed being displayed
- Alternatively:
  - Create a type with Data Annotations
  - Use the ‘ModelMetadataType’ attribute to associate with data type

DataAnnotations (C#)

- Validation of Properties defined using DataAnnotations:

```
[ModelMetadataType(typeof(PassengerDetailsMetadata))]
partial class PassengerDetails
{
    public class PassengerDetailsMetadata
    {
        public int Id { get; set; }
        [Required(ErrorMessage="{0} required!")]
        [StringLength(50,ErrorMessage="{0} length less than or equal {1}!")]
        public string Name { get; set; }

        [Required(ErrorMessage = "{0} required!")]
        [Range(0,30,ErrorMessage = "{0} between {1} and {2} inclusive!")]
        public int Weight { get; set; }
    }
}
```

Displaying Validation Messages

- MVC can provide validation on Client side:

```
<form asp-action="Edit" >
    <div class="form-horizontal">
        <h4>Flight</h4>
        <hr />
        <div asp-validation-summary="ValidationSummary.ModelOnly" class="text-danger"></div>
        <input type="hidden" asp-for="Id" />
        <div class="form-group">
            <label asp-for="Destination" class="col-md-2 control-label"></label>
            <div class="col-md-10">
                <input asp-for="Destination" class="form-control" />
                <span asp-validation-for="Destination" class="text-danger" />
            </div>
        </div>
```

Display Summary of Messages

Validation Message

Validation - Summary

- This Section introduced validation:
  - Validation Introduction
  - Data Annotations and Validation
  - DataAnnotations
  - Displaying Validation Messages

Web API and REST

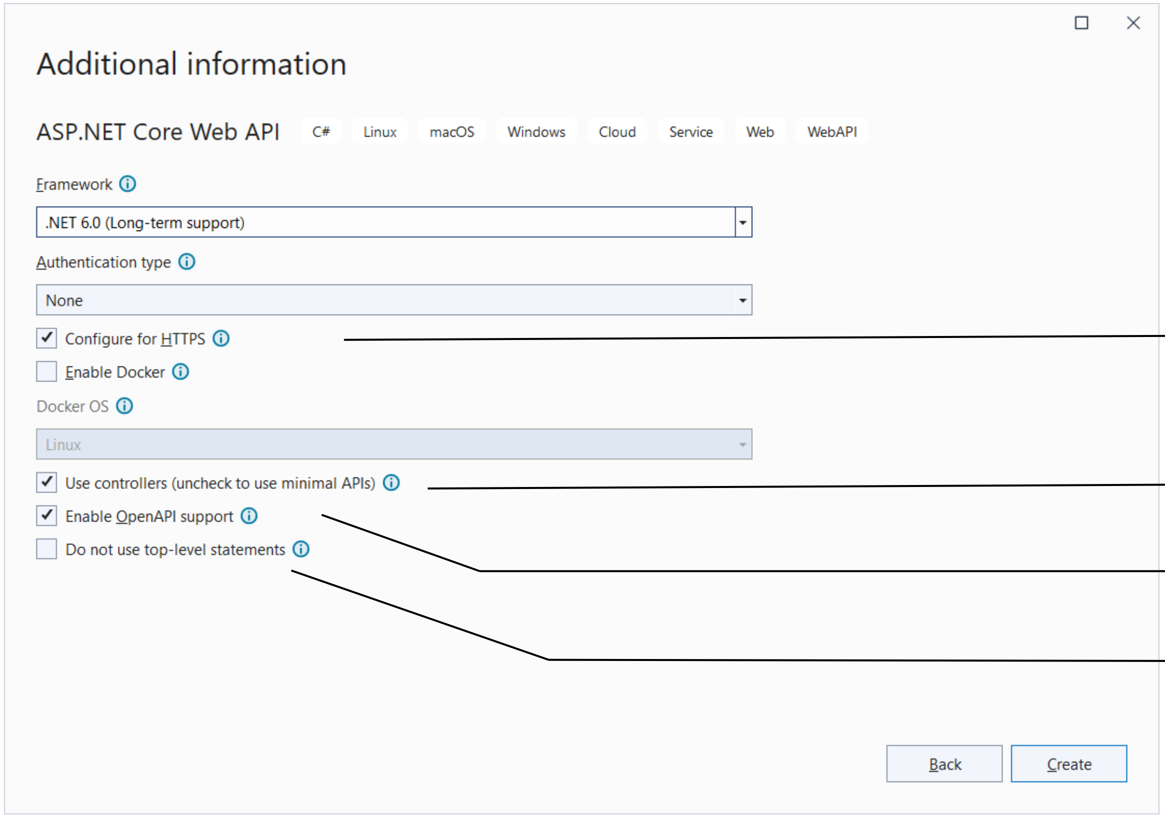
- This section introduces the use of Web API and REST:
  - Web API
  - Web API Controller
  - REST Verbs
  - Resources
  - Default Controller
  - Customising Controller
  - Attributes

Web API

- Web API was originally added for ASP.NET MVC 4/5 and now ASP.NET Core
  - Supports ‘Representational State Transfer’ (REST)
  - HTTP provides access from wide range of clients
    - Desktop/Server
    - Phones/Tables
  - Supports common data format, i.e. XML, JSON
  - Allows defining of uri through routes
  - Supports wide range of HTTP verbs
  - Allows support for OpenAPI
  - .NET 6 introduces ‘minimal APIs’

Creating Web API Application

- When creating a project the following dialog is displayed:



Additional information

ASP.NET Core Web API

Framework: .NET 6.0 (Long-term support)

Authentication type: None

[✓] Configure for HTTPS

[ ] Enable Docker

Docker OS: Linux

[✓] Use controllers (uncheck to use minimal APIs)

[✓] Enable OpenAPI support

[ ] Do not use top-level statements

Support for HTTPS

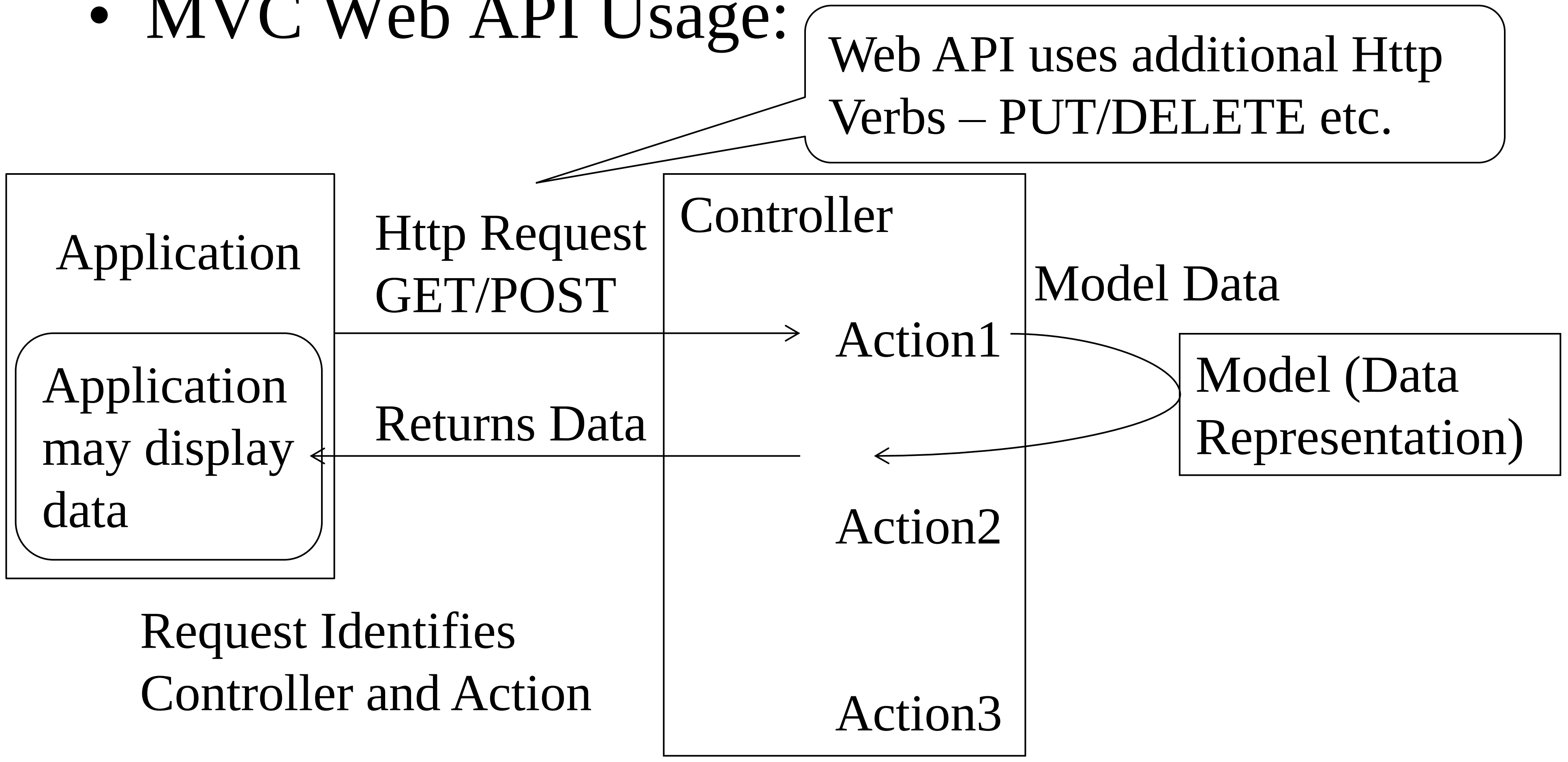
Support for minimal API

Enable Open API

Top-level Statements?

Web API Controller

- MVC Web API Usage:


Web API uses additional Http Verbs – PUT/DELETE etc.
Application
Application may display data
Http Request GET/POST
Returns Data
Controller
Action1
Action2
Action3
Model Data
Model (Data Representation)
Request Identifies Controller and Action

REST Verbs

- Representational State Transfer (REST) is an architectural approach to providing
  - Resource based rather than remote methods
  - General usage:

Verb	**Description (action on resource)**
GET	read
POST	insert
PUT	replace
DELETE	remove
Custom verb	Define custom meaning!

Default Controller (ASP.NET Core)

- Adding the template creates an example controller of the form:

```
[ApiController]
[Route("api/[controller]")]
public class ValuesController : ControllerBase
{
    [HttpGet]
    public IActionResult Get()  // GET: api/values
    {
        return Ok(new string[] { "value1", "value2" });
    }
    [HttpGet("{id}", Name = "Get")]
    public IActionResult Get(int id)  // GET api/values/5
    {
        return Ok("value“);
    }     …
```

Indicates API

Route including Controller Name

Uri for Request

Alternatively return ActionResult<IEnumerable<string>>

Attributes indicated HTTP Method

Route Name

Routing Information can now be within Attribute Parameter

Value in Uri passed as parameter

Customizing Controller

- Defining a Passenger Controller:

```
[ApiController][Route("api/[controller]")]
public class PassengersAPIController : ControllerBase
{
    [HttpGet]
    public IActionResult Get()  // GET: api/passengers
    {
        return Ok(new PassengerDetails[] {
                              new PassengerDetails(){ Name="Fred1",Weight=17} });
    }
    [HttpGet("{id}")]
    public IActionResult Get(string id)  // GET api/passengersapi/5
    {
        return Ok(new PassengerDetails() { Name = "Fred1", Weight = 17 });
    }       …
```

Get returns Sequence of PassengerDetails

Single PassengerDetails identified by id

Action POST

- When data is posted back the action should check ModelState (ApiController introduced in 2.1):

```
// POST api/values
[HttpPost]
public IActionResult Post([FromBody]PassengerDetails value)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    else
    { return Created(value); }
}
```

Adding [ApiController] on Controller means this check can now be omitted!

Checks Post Data

Return 400

Return 201 for success

Overload of Created can be used to return URI for location of resource (using named 'GET' route)!

Attributes

- Many attributes influence routing:

Attribute	**Description**
Area	Applied to Controller to specify area
Route	Applied to Controller to give detailed route to resource including parameters ‘[controller]’ is placeholder for controller
HttpGet, HttpPost, HttpPut, HttpHead, etc.	Attributes applied to Action to determine Method
AcceptVerbs	Applied to Action to specify none standard Method

Web API and REST - Summary

- This section introduced the use of Web API and REST:
  - Web API
  - Web API Controller
  - REST Verbs
  - Resources
  - Default Controller
  - Customising Controller
  - Attributes