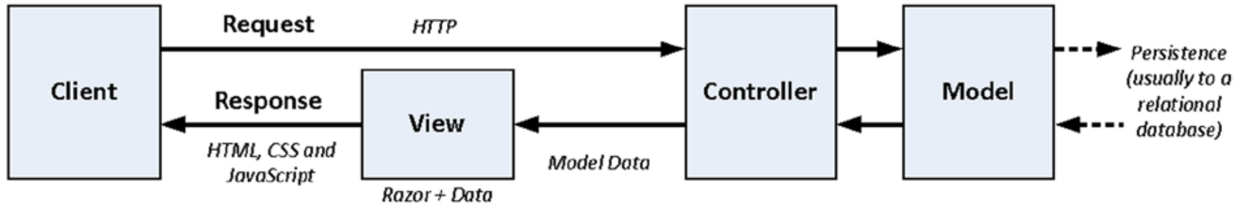
AspNetCore MVC Controllers

https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-controller?view=aspnetcore-5.0&tabs=visual-studio

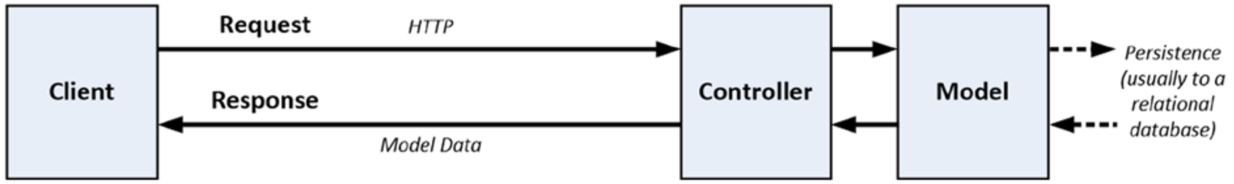
CHAPTER 17

Architecture of ASP.NET Web Applications

- MVC application



- Web API application



Entry point for ASP.NET Core apps

```
public class Program
{
    public static void Main(string[] args) {
        BuildWebHost(args).Run();
    }

    public static IWebHost BuildWebHost(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>()
            .Build();

}
```

- `WebHost` -- provides convenience methods for creating instances of

  AspNetCore.Hosting.IWebHost and **AspNetCore.Hosting.IWebHostBuilder** with pre-configured defaults
- **`CreateDefaultBuilder --`** initializes a new instance of **`WebHostBuilder`**
- **`UseStartup<Startup> --`** identifies class **`Startup`** that will provide application-specific configuration
- **`Build`** -- processes all configuration settings and creates object that implements **`IWebHost`**
- **`Run`** start handling HTTP requests

Creating Controllers

- MVC recognizes any controller class and sends requests, without any limitation on how the request is processed and responded, to:
  - **controller** – any **public** class whose name ends with **Controller**
  - **action** – any **public** method in controller's class
- Pure POCO (plain old CLR object) controllers, which have no dependencies on the **AspNetCore namespaces**, are not especially useful because they don't have access to the features that MVC provides for processing requests
- An easier way to create controllers is to derive classes from the **AspNetCore.Mvc.Controller** class, which defines methods and properties that provide access to MVC features in a more concise and useful manner

Key Features of `Controller`

- **Action methods --** A controller's behavior is partitioned into multiple methods :
  - each action method is exposed on a **different URL**
  - is invoked with parameters extracted from the incoming request
- **Action results --** object describing the result of an action :
  - rendering a view
  - redirecting to a different URL or action method
- **Filters --** encapsulate reusable behaviors (for example, authentication), and then tag each behavior onto one or more controllers or action methods by putting an attribute in your source code

Controller

```
//     A base class for an MVC controller with view support.
public abstract class Controller : ControllerBase, IActionFilter,
                     IFilterMetadata, IAsyncActionFilter, IDisposable
{
    protected Controller();
    public dynamic ViewBag { get; }
    [ViewDataDictionary]
    public ViewDataDictionary ViewData { get; set; }
    public ITempDataDictionary TempData { get; set; }
    [NonAction]
    public virtual JsonResult Json(object data,
                         JsonSerializerSettings serializerSettings); //+1
    public virtual void OnActionExecuting(ActionExecutingContext context); //+2
    public virtual PartialViewResult PartialView(string viewName, object model); //+3
    public virtual ViewResult View(string viewName, object model); //+3
}
```

ControllerBase

```
public abstract class ControllerBase{
//base class for MVC controller without view support
    protected ControllerBase();
     public ClaimsPrincipal User { get; }
     public HttpContext HttpContext { get; }
     public HttpRequest Request { get; }
     public HttpResponse Response { get; }
     public RouteData RouteData { get; }
     public ModelStateDictionary ModelState { get; }
     public IModelMetadataProvider MetadataProvider { get; set; }
     public IModelBinderFactory ModelBinderFactory { get; set; }
     [ControllerContext]
     public ControllerContext ControllerContext { get; set; }
     public IObjectModelValidator ObjectValidator { get; set; }
     public IUrlHelper Url { get; set; }
     public virtual RedirectResult Redirect(string url); //+42 !!!
     public virtual RedirectToActionResult RedirectToAction(string actionName,
               string controllerName, object routeValues, string fragment); //+5
     ........
}
```

Context data

- Controllers rarely exist in isolation and usually need to access data from the incoming request, collectively known as **context data**:
    - query string values
    - form values
    - parameters parsed from the URL by the routing system

- There are three main ways to access context data:
    - extract it from a set of **context objects**
    - receive the **data as a parameter** to an action method
    - explicitly invoke the framework's **model binding** feature

Getting Data from Context Objects

Property	**Description**
HttpContext	Encapsulates HTTP-specific information about an individual HTTP request
HttpContext. **Request**	gets **Http.HttpRequest** object for this request
HttpContext. **Response**	gets **Http.HttpResponse** object for this request
HttpContext.**Session**	State store for the visitor’s session
HttpContext.**User**	gets or sets the user for this request
Request	returns an **HttpRequest** object that describes request received from the client
Request.QueryString	GET variables sent with this request
Request.Form	POST variables sent with this request
Request.HttpMethod	HTTP method (verb, such as GET or POST)
Request.Host	IP address of the user making this request
RouteData	**RouteData** produced by the routing system when it matched the request
RouteData.Routes	**RouteTable.Routes** entry for this request
RouteData.Values	Active route parameters
TempData	Temporary data items stored for the current user

Actions

- Action methods help us to pass models to views, file streams, and also redirect to another action method
- Typically an action method returns an instance of an `ActionResult`
- An action could have a **void** or a **string** return types
- Action methods can define parameters that are used by MVC to pass context data to a controller, including details of the HTTP request

Actions Requirements

- It must be public
- It can’t be static
- It can’t be an extension method
- It can’t be a constructor, getter, or setter
- It can’t have open generic types
- It can’t be a method of the `Controller` or `ControllerBase` base classes
- It can’t contain **ref** or **out** parameters
- It can’t be decorated with the **NonAction** action selector

ActionResult

- `ActionResult` – return type of controller method, that describes what the response from controller will be:
  - rendering by a view
  - redirecting to another URL or action method.
- Derived `ActionResult` types :
  - `ViewResult` – renders a specified view to the response stream;
  - `PartialViewResult` – renders a partial view;
  - `RedirectResult` – is used to perform an HTTP redirect to a given URL;
  - `RedirectToRouteResult` – is used to redirect by using the specified route values dictionary;
  - `ContentResult` – is used to return to an action as plain text;
  - `JsonResult` – serializes the specified object to JSON format; ...

ActionResult Classes

```
public interface IActionResult{
    Task ExecuteResultAsync( ActionContext context);
}
public abstract class ActionResult : IActionResult{
    public virtual void ExecuteResult(ActionContext context);
    public virtual Task ExecuteResultAsync(ActionContext context);
}
```

```
public class ViewResult : ActionResult {
    public ViewResult();
    public int? StatusCode { get; set; }
    public string ViewName { get; set; }
    public object Model { get; }
    public ViewDataDictionary ViewData { get; set; }
    public ITempDataDictionary TempData { get; set; }
    public IViewEngine ViewEngine { get; set; }
    public string ContentType { get; set; }
    [AsyncStateMachine(typeof(< ExecuteResultAsync > d__26))]
    public override Task ExecuteResultAsync(ActionContext context);
}
```

Deriving from Controller

```
public class DerivedController : Controller
{
    public ViewResult Index() =>
        View("Result", $"This is a derived controller");
}
```

Result.cshtml

```
@model string
@{ Layout = null; }
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Controllers and Actions</title>
    <link rel="stylesheet" asp-href-include="lib/bootstrap/dist/css/*.min.css" />
</head>
<body class="m-1 p-1">
     Model Data: @Model
</body>
</html>
```

ControllersAndActions
1) derived
2) home

RedirectToActionResult

```
[HttpPost]
public RedirectToActionResult ReceiveForm(string name, string city)
{
    TempData["name"] = name;
    TempData["city"] = city;
    return RedirectToAction(nameof(Data));
}

public ViewResult Data()
{
    string name = TempData["name"] as string;
    string city = TempData["city"] as string;
    return View("Result", $"{name} lives in {city}");
}
```

Invoking of Action Methods and Route Data

- MVC invokes different controller classes (and different action methods within them) depending on the incoming URL
- Action methods can define parameters that are used by MVC to pass context data to a controller, including details of the HTTP request
- In MVC applications it's more typical to pass in parameters as route data than passing them as query strings
- Default URL routing logic used by MVC uses a format like this to determine what code to invoke:

/[Controller]/[ActionName]/[Parameters]

Method with Parameters: Matching the Query Strings

```
        public IActionResult Welcome(string name, int numTimes = 1)
         {
             ViewBag.Message = "Hello " + name;
             ViewBag.NumTimes = numTimes;
             return View();
         }
```

http://localhost:52808/example/welcome?name=Anatoliy&numTimes=4

- URL segment (Parameters) is not used
- **name** and **numTimes** parameters are passed as query strings
- ? (question mark) in URL is a separator, and the query strings follow
- & character separates query strings

ControllersAndActions
1) example

ViewResult

- There are a number of overridden versions of **Controller.View()** , that correspond to setting different properties on the **ViewResult** object that is created
- common return syntax : `return View();`
- it returns **ViewResult**
- **Strongly typed view** -- view includes details of the type of the **view model** object
- object in the view can be accessed using the Razor **Model** keyword

Providing a View Model Object

```
public ViewResult Index()
{
      DateTime date = DateTime.Now;
      return View(date);
}
```

Index.cshtml

```
@model DateTime
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
The day is: @Model.DayOfWeek
```

Looking in for the `View`

- /`Views`/ <ControllerName> / <ViewName> `.cshtml`
- /`Views/Shared`/ <ViewName> `.cshtml`