# ASP.NET MVC Interview Questions by Karthik M

Q1. What is ASP.NET MVC? How is it different from ASP.NET Web Forms?

Answer:

ASP.NET MVC stands for Model-View-Controller, a design pattern that separates an application into three components:

a. Model: Handles application data and business logic.
b. View: Represents the UI part of the application.
c. Controller: Responds to user inputs and interacts with the Model and View.

Differences from Web Forms:

a. No ViewState in MVC, so pages are lightweight.
b. MVC gives full control over HTML, while Web Forms use server controls.
c. MVC follows the Separation of Concerns principle.
d. MVC applications are more easily testable than Web Forms.

Q2. Explain the ASP.NET MVC Request Life Cycle.

Answer:
The ASP.NET MVC Request Life Cycle involves these key steps:

Step - 1
Routing: URL is matched to a route defined in RouteConfig.

Step - 2
Controller Initialization: Appropriate Controller is instantiated.

Step - 3
Action Execution: The action method is invoked.

Step - 4
Result Execution: ActionResult (like ViewResult) is returned.

Step - 5
View Rendering: The View is rendered and returned to the browser.

Q3. What is a Controller in MVC?

Answer:
A Controller handles incoming HTTP requests and returns responses. Controllers inherit from the Controller base class.

Example:

```
public class ProductController : Controller
{
    public ActionResult Details(int id)
    {
        // Fetch product by id
        return View();
    }
}
```

Q4. What is an Action Method in MVC?

Answer:
Action Methods are public methods inside a Controller class that handle HTTP requests.

Example:
```
public ActionResult About()
{
    return View();
}
```

An Action Method must be:
a. Public
b. Non-static
c. Not decorated with [NonAction] attribute

Q5. What is Routing in ASP.NET MVC?

Answer:
Routing defines how URLs map to controller actions. It is configured in RouteConfig.cs.

Example:
```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

Thus, the URL /Product/Details/5 calls ProductController's Details method with id = 5.

Q6. How can you pass data from Controller to View in MVC?

Answer:
Three common ways to pass data:

a. ViewBag (dynamic): ViewBag.Message = "Hello World";
b. ViewData (dictionary): ViewData["Message"] = "Hello World";
c. Model (strongly typed): return View(productList);

In the View, you can access:

a. @ViewBag.Message
b. @ViewData["Message"]
c. @Model.PropertyName

Q7. What is the difference between ViewBag, ViewData, and TempData?

Answer:

a. ViewBag: Dynamic wrapper around ViewData, lives for the current request.
b. ViewData: Dictionary object, also lives only for the current request.
c. TempData: Uses Session to persist data for one more request (useful after redirects).

Example using TempData:
```
TempData["SuccessMessage"] = "Record Saved Successfully!";
```

Q8. What is a Strongly Typed View?

Answer:
A Strongly Typed View is a View that is bound to a specific Model class.

At the top of the Razor View, you declare the model:
```
@model MyApp.Models.Product
```

Then you can access properties with IntelliSense:
@Model.ProductName
@Model.Price

This makes development strongly typed, compile-time checked, and more efficient.

Q9. What is Razor View Engine in ASP.NET MVC?

Answer:
Razor is the View Engine that allows mixing C# code with HTML easily.
Syntax uses @ to indicate C# code.

Example:
```
@foreach (var item in Model.Products)
{
    <p>@item.Name</p>
}
```

Advantages:

a. Cleaner syntax
b. Faster rendering
c. No need for <% %> syntax like older Web Forms

Q10. What is the difference between Html.Partial() and Html.RenderPartial()?

Answer:

Html.Partial() returns a string that represents HTML content.

Html.RenderPartial() writes HTML directly to the response stream — slightly faster.

Example using Html.Partial():
```
@Html.Partial("_ProductPartial", productList)
```

Example using Html.RenderPartial():
```
@{ Html.RenderPartial("_ProductPartial", productList); }
```

Use Html.Partial() when you need the output returned. Use Html.RenderPartial() for slightly better performance when you don't need the returned HTML.

Q11. What are different types of Action Results in MVC?

Answer:
Action Results represent different types of responses that a controller action can return.

Common types:

a. ViewResult — returns HTML and markup (using Views).
b. PartialViewResult — returns a partial View.
c. RedirectResult — redirects to another URL.
d. RedirectToRouteResult — redirects to another route.
e. JsonResult — returns JSON-formatted data.
f. FileResult — returns a file to download.
g. ContentResult — returns plain text.

Example returning a JSON:
```
return Json(new { id = 1, name = "Product1" }, JsonRequestBehavior.AllowGet);
```

Q12. How can you restrict an action method to be invoked only by POST request?

Answer:
Use the [HttpPost] attribute above the Action Method.

Example:
```
[HttpPost]
public ActionResult Save(Product model)
{
    // Save logic
    return RedirectToAction("Index");
}
```

Similarly, use [HttpGet] for GET requests.

Q13. What is the difference between RedirectToAction() and Redirect()?

Answer:

RedirectToAction("Index") redirects to a specific action inside a controller.

Redirect("http://example.com") redirects to a specified URL.

Example using RedirectToAction:
```
return RedirectToAction("Index", "Home");
```

Example using Redirect:
```
return Redirect("https://www.google.com");
```

RedirectToAction is route-based; Redirect is URL-based.

Q14. What is the use of the [NonAction] attribute in MVC?

Answer:
The [NonAction] attribute is used to mark a public method in a controller that should not be treated as an Action Method.

Example:
```
public class ProductController : Controller
{
    [NonAction]
    public void HelperMethod()
    {
        // Not an action
    }
}
```

Q15. What are Filters in MVC?

Answer:
Filters are attributes that add extra behavior to action methods or controllers.

Types of filters:

a. Authorization filters (e.g., [Authorize])
b. Action filters (e.g., [OnActionExecuting])
c. Result filters (e.g., [OnResultExecuting])
d. Exception filters (e.g., [HandleError])

Example:
```
[Authorize]
```

```
public ActionResult Dashboard()
{
    return View();
}
```

Q16. What is the difference between TempData, ViewData, and ViewBag?

Answer:

| Aspect | ViewData | ViewBag | TempData |
|---|---|---|---|
| Type | Dictionary | Dynamic | Dictionary |
| Life Span | Current Request | Current | Request Persists until next request |
| Usage | ViewData["key"] | ViewBag.key | TempData["key"] |
| Use Case | Passing data to View | Passing data to View | Passing data across Redirects |

Q17. What is Model Binding in MVC?

Answer:
Model Binding maps incoming request data (Form, Query String, Route Data) to action method parameters.

Example:

```
public ActionResult Save(Product model)
{
    // Model Binding automatically fills model properties
    return View();
}
```

If the posted form has input fields named ProductName, Price, they are automatically mapped to the Product object.

Q18. How can you validate a model in ASP.NET MVC?

Answer:
Use Data Annotations on the model properties and call ModelState.IsValid in the controller.

Example:

```
public class Product
{
    [Required]
    public string ProductName { get; set; }

    [Range(1, 10000)]
    public decimal Price { get; set; }
}
```

Controller code:

```
if (ModelState.IsValid)
{
    // Save Product
}
```

Q19. What is the difference between View() and PartialView() in MVC?

Answer:

View() renders the full View including Layout.

PartialView() renders only a section of a page, without Layout.

Example:
return View("Index");

return PartialView("_ProductList", products);

Partial Views are useful for AJAX updates and reusable UI parts.

Q20. How do you implement Custom Error Handling in MVC?

Answer:
You can use:

[HandleError] attribute on controllers/actions

customErrors section in web.config

Example with [HandleError]:
```
[HandleError(View = "Error")]
public class ProductController : Controller
{
    public ActionResult Create()
    {
        throw new Exception("Something went wrong!");
    }
}
```

Also, configure in web.config:
```
<customErrors mode="On" defaultRedirect="~/Home/Error" />
```

Q21. What is a View in MVC?

Answer:
A View is the user interface (UI) component in MVC architecture. It presents data to the user and accepts input.

Example View (Index.cshtml):
```
@model MyApp.Models.Product

<h2>Product Details</h2> <p>Name: @Model.ProductName</p> <p>Price: @Model.Price</p>
```
The View uses the data passed from the Controller and renders it as HTML.

Q22. What is Razor Syntax in ASP.NET MVC?

Answer:
Razor is the syntax used to embed server-side C# code into HTML markup easily.

Syntax Examples:

Inline expressions: @Model.ProductName

Loop example:
```
@foreach (var item in Model.Products)
```

```
{
    <p>@item.Name</p>
}

Rules:

a. Code statements start with @
b. No need for <% %> like older ASP.NET Web Forms.


Q23. How do you create a strongly typed View?

Answer:
At the top of the View, declare the model type using @model:

@model MyApp.Models.Employee

Then you can access the Model properties with IntelliSense:

<p>@Model.EmployeeName</p> <p>@Model.Salary</p>
Strongly typed Views ensure type safety and better productivity.




Q24. What are HTML Helpers in MVC?

Answer:
HTML Helpers are methods that simplify rendering standard HTML elements.

Common examples:

@Html.TextBoxFor(m => m.ProductName)

@Html.LabelFor(m => m.ProductName)

@Html.DropDownListFor(m => m.CategoryId, categoryList)

They help generate clean and consistent HTML controls in Views.




Q25. What is the difference between Html.TextBox() and Html.TextBoxFor()?

Answer:

Html.TextBox() is loosely typed (string-based), no IntelliSense.
Example: @Html.TextBox("ProductName")

Html.TextBoxFor() is strongly typed (model-based), compile-time checked.
Example: @Html.TextBoxFor(m => m.ProductName)

Prefer TextBoxFor() for strong typing and error checking.




Q26. How do you create a DropDownList using HTML Helpers?

Answer:
You can create a DropDownList using Html.DropDownListFor().

Example:
@Html.DropDownListFor(m => m.CategoryId, new SelectList(Model.Categories, "CategoryId",
"CategoryName"))

Here:

```

m => m.CategoryId is the model property bound.

Model.Categories is the source list.

Q27. What is ViewStart file in MVC?

Answer:
The _ViewStart.cshtml file contains code that should run before every View is rendered.

Example content of _ViewStart.cshtml:

```
@{ Layout = "~/Views/Shared/_Layout.cshtml"; }
```

It sets a default Layout for all Views, so you don’t have to set Layout individually in every View.

Q28. What is the purpose of Layout pages in MVC?

Answer:
Layout pages are like Master Pages in Web Forms.
They allow you to define a common structure (e.g., header, footer, navigation) and reuse it across multiple Views.

Example Layout (_Layout.cshtml):

```
<!DOCTYPE html>
<html>
<head>
<title>@ViewBag.Title</title>
</head>
<body> @RenderBody() </body>
</html>
```

Each View is injected into @RenderBody() placeholder at runtime.

Q29. How do you render a Partial View manually inside a View?

Answer:
Use either:

```
@Html.Partial("_ProductPartial") (returns HTML string)

@{ Html.RenderPartial("_ProductPartial"); } (writes directly to response)
```

Both render _ProductPartial.cshtml inside the parent View.

Use Partial Views when you want to reuse pieces of UI.

Q30. How do you perform form submissions in MVC?

Answer:
You create a form using Html.BeginForm().

Example:

```
@using (Html.BeginForm("Save", "Product", FormMethod.Post))
{
    @Html.LabelFor(m => m.ProductName)
```

```
    @Html.TextBoxFor(m => m.ProductName)

    <input type="submit" value="Save" />
}
```

When the user submits the form, it posts the data to Save action in ProductController.

Q31. What is ModelState in ASP.NET MVC?

Answer:
ModelState is a property of the Controller that holds the state of model binding and validation.

Typically, you check if the model is valid before proceeding:

```
if (ModelState.IsValid)
{
    // Save to database
}
else
{
    // Return validation errors to View
}
```

It automatically tracks validation errors based on Data Annotations applied to Model properties.

Q32. What is Data Annotation in MVC?

Answer:
Data Annotations are attributes applied to Model properties to enforce validation rules.

Example:

```
public class Product
{
    [Required(ErrorMessage = "Product Name is required")]
    public string ProductName { get; set; }

    [Range(1, 10000)]
    public decimal Price { get; set; }
}
```

When you post data, MVC will automatically validate these rules.

Q33. How do you create custom validation attributes in MVC?

Answer:
You create a class by inheriting from ValidationAttribute and override IsValid().

Example:

```
public class NoZeroPriceAttribute : ValidationAttribute
{
    public override bool IsValid(object value)
    {
        decimal price = (decimal)value;
        return price != 0;
    }
}
```

Apply it to a property:

```
[NoZeroPrice(ErrorMessage = "Price cannot be zero")]
public decimal Price { get; set; }
```

Q34. What is Unobtrusive JavaScript validation in MVC?

Answer:
Unobtrusive Validation separates JavaScript behavior from HTML markup, resulting in cleaner code.

To enable it:

Include jQuery and jQuery Validation libraries.

Set these keys in web.config:

```
<appSettings>
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
</appSettings>
```

It automatically enables client-side validation based on Data Annotations.

Q35. How do you perform server-side validation in MVC?

Answer:
Server-side validation is done by checking ModelState.IsValid in Controller Actions.

Example:

```
[HttpPost]
public ActionResult Save(Product model)
{
    if (ModelState.IsValid)
    {
        // Save data
    }
    return View(model);
}
```

Even if client-side validation is disabled, server-side validation ensures data integrity.

Q36. What are strongly typed HTML helpers?

Answer:
Strongly typed helpers are bound to a specific Model property.

Example:

```
@Html.TextBoxFor(m => m.ProductName)
@Html.EditorFor(m => m.Price)
@Html.DisplayFor(m => m.ProductName)
```

Advantages:

a. Compile-time checking
b. IntelliSense support

c. Less prone to typos

Q37. How do you return JSON from an MVC Controller?

Answer:
You can use the JsonResult return type.

Example:

```
public JsonResult GetProduct(int id)
{
    var product = new Product { Id = id, Name = "Pen", Price = 100 };
    return Json(product, JsonRequestBehavior.AllowGet);
}
```

This will serialize the product object to JSON and return it to the client.

Q38. What is Dependency Injection (DI) in MVC?

Answer:
Dependency Injection is a design pattern to inject dependencies into classes rather than having them instantiate the dependencies themselves.

Benefits:

Loose coupling
Easier testing
Better maintainability

In ASP.NET Core MVC, built-in DI is configured using:

```
services.AddScoped<IProductService, ProductService>();
```

In traditional MVC, you can use libraries like Unity, Ninject, Autofac for DI.

Q39. How do you implement custom Dependency Injection in ASP.NET MVC?

Answer:
You create your own dependency resolver by implementing IDependencyResolver.

Example:

```
public class CustomResolver : IDependencyResolver
{
    public object GetService(Type serviceType)
    {
        // Instantiate services here
        return new ProductService();
    }

    public IEnumerable<object> GetServices(Type serviceType)
    {
        return new List<object>();
    }
}
```

And register it in Global.asax:

```
DependencyResolver.SetResolver(new CustomResolver());
```

Q40. What is Bundling and Minification in MVC?

Answer:
Bundling and Minification are performance techniques to:

Bundle multiple CSS or JS files into one file.

Minify (remove whitespaces and comments) the file content.

Example in BundleConfig.cs:

```
bundles.Add(new ScriptBundle("/bundles/jquery").Include(
    "/Scripts/jquery-{version}.js"
));

bundles.Add(new StyleBundle("/Content/css").Include(
    "/Content/bootstrap.css",
    "~/Content/site.css"
));
```

It reduces the number of requests and improves page load time.

Q41. How do you enable Bundling and Minification in MVC?

Answer:
By default, Bundling and Minification happen only in Release mode.

You can force them manually in BundleConfig.cs by setting:
`BundleTable.EnableOptimizations = true;`

And then register bundles inside Application_Start() in Global.asax:
`BundleConfig.RegisterBundles(BundleTable.Bundles);`

This combines and compresses CSS and JS for faster page loads.

Q42. What is the use of AntiForgeryToken in MVC?

Answer:
Anti-Forgery Tokens prevent Cross-Site Request Forgery (CSRF) attacks.

In the View:

```
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    // Form fields
}
```

In the Controller:

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Save(Product model)
{
    // Save logic
}
```

The server checks that the token submitted matches the one generated, ensuring the request is genuine.

Q43. What is the [AllowAnonymous] attribute in MVC?

Answer:
[AllowAnonymous] allows unauthenticated users to access specific actions or controllers even if the entire application requires authentication.

Example:

```
[Authorize]
public class DashboardController : Controller
{
    public ActionResult Index() { return View(); }

    [AllowAnonymous]
    public ActionResult Help() { return View(); }
}
```

Here, Dashboard/Help can be accessed without logging in.

Q44. How do you handle 404 errors in MVC?

Answer:
You can handle 404 errors globally by updating web.config:

```
<system.web>
    <customErrors mode="On" defaultRedirect="~/Error/General">
        <error statusCode="404" redirect="~/Error/NotFound"/>
    </customErrors>
</system.web>
```

You can also create an ErrorController with NotFound and General actions to handle these gracefully.

Q45. What is Areas in ASP.NET MVC?

Answer:
Areas allow you to divide a large MVC application into smaller functional sections, each with its own Controllers, Views, and Models.

For example:

Admin Area
Customer Area
Billing Area

Each Area has its own AreaRegistration.cs file where its routes are registered.

Example:

```
context.MapRoute(
    "Admin_default",
    "Admin/{controller}/{action}/{id}",
    new { action = "Index", id = UrlParameter.Optional }
);
```

Q46. What is Output Caching in MVC?

Answer:
Output Caching stores the response of a Controller action and reuses it for subsequent

requests, boosting performance.

Example:

```
[OutputCache(Duration = 60)]
public ActionResult Index()
{
    // This View will be cached for 60 seconds
    return View();
}
```

Benefits:

a. Reduces server processing
b. Improves load time

Q47. What is Child Action Only in MVC?

Answer:
[ChildActionOnly] attribute restricts an action method to be called only from another View, not directly via URL.

Example:

```
[ChildActionOnly]
public ActionResult Menu()
{
    return PartialView("_Menu");
}
```

You render it inside another View like:

```
@Html.Action("Menu")
```

If a user tries to directly access /Home/Menu, they will get an error.

Q48. How do you use Authorize attribute with Roles?

Answer:
You can specify roles allowed to access a controller or action.

Example:

```
[Authorize(Roles = "Admin,Manager")]
public ActionResult ManageUsers()
{
    return View();
}
```

Only users in the "Admin" or "Manager" roles can access ManageUsers.

Q49. How can you send JSON data from JavaScript to an MVC controller?

Answer:
Using jQuery $.ajax() method:

```
$.ajax({
    url: '/Product/Save',
```

```
    type: 'POST',
    data: JSON.stringify({ Id: 1, Name: "Pen" }),
    contentType: 'application/json; charset=utf-8',
    success: function(response) {
        alert(response);
    }
});
```

Controller:

```
[HttpPost]
public JsonResult Save(Product product)
{
    // Save product
    return Json("Success");
}
```

Q50. What is TempData Keep() and Peek() in MVC?

Answer:

TempData.Keep() preserves data for another request manually.
TempData.Peek() reads data without marking it for deletion.

Example:

```
string message = TempData.Peek("Message") as string;
TempData.Keep("Message");
```

Normally, TempData gets cleared after one read, but Keep() and Peek() allow retaining it.

Q51. How do you make an Ajax call in MVC to update part of a page?

Answer:
Use jQuery $.ajax() to call an action and update a div without refreshing the full page.

Example:

```
$.ajax({
    url: '/Product/GetProductList',
    type: 'GET',
    success: function (data) {
        $('#productDiv').html(data);
    }
});
```

In your View:

```
<div id="productDiv"></div>
```

Controller:

```
public ActionResult GetProductList()
{
    return PartialView("_ProductListPartial", products);
}
```

Q52. What is the difference between ViewBag and TempData in MVC?

Answer:

| Aspect | ViewBag | TempData |
|---|---|---|
| Life Span | Current HTTP Request | Current + Next Request |
| Usage | Dynamic property (ViewBag.Name) | Dictionary-based (TempData["Name"]) |
| Purpose | Pass data from Controller to View | Pass data between Controllers |

Q53. What are the different types of Filters in MVC?

Answer:

a. Authorization Filters — Implement IAuthorizationFilter.
b. Action Filters — Implement IActionFilter.
c. Result Filters — Implement IResultFilter.
d. Exception Filters — Implement IExceptionFilter.

You can create custom filters by inheriting ActionFilterAttribute and overriding methods like OnActionExecuting() and OnResultExecuting().

Example:

```
public class CustomActionFilter : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        // Your custom logic
    }
}
```

Q54. What is Web API and how is it different from MVC?

Answer:

| Aspect | MVC | Web API |
|---|---|---|
| Purpose | Return Views (HTML) | Return Data (JSON, XML) |
| Return Type | ActionResult/ViewResult | HttpResponseMessage/IHttpActionResult |
| Use Case | Web Applications | RESTful Services |

MVC is ideal for UI-rendered apps.
Web API is used for services consumed by clients like mobile apps, SPA (Single Page Applications), etc.

Q55. How can you secure Web API endpoints?

Answer:
Options include:

a. Basic Authentication
b. Token-based Authentication (JWT)
c. OAuth
d. API Key validation
e. Custom Authorization Filters

Example Basic Authentication setup:

```
[Authorize]
public class ProductsController : ApiController
{
```

```
    // Secure API
}
```

You also configure authentication handlers in Startup.cs for ASP.NET Core.

Q56. How to prevent Cross-Site Scripting (XSS) in MVC?

Answer:
MVC automatically encodes output. Use:

```
@Html.Encode(Model.ProductName)
```

Or directly in Razor:

```
<p>@Model.ProductName</p>
```

To disable encoding (if absolutely required):

```
@Html.Raw(Model.Description)
```

Always validate and sanitize inputs to prevent XSS attacks.

Q57. How can you call a Web API from MVC?

Answer:
You can use HttpClient inside a controller.

Example:

```
HttpClient client = new HttpClient();
client.BaseAddress = new Uri("https://api.example.com/");
HttpResponseMessage response = await client.GetAsync("api/products");
if (response.IsSuccessStatusCode)
{
    var products = await response.Content.ReadAsAsync<IEnumerable<Product>>();
}
```

This helps MVC applications consume external APIs.

Q58. What is the use of Route Constraints in MVC?

Answer:
Route Constraints restrict which URLs match a route based on pattern matching.

Example:

```
routes.MapRoute(
    "Default",
    "{controller}/{action}/{id}",
    new { controller = "Home", action = "Index", id = UrlParameter.Optional },
    new { id = @"\d+" } // id must be numeric
);
```

This ensures that id must be digits only (regular expression constraint).

Q59. How do you handle Global Exception Handling in MVC?

Answer:

You can create a custom filter:

```
public class GlobalExceptionFilter : FilterAttribute, IExceptionFilter
{
    public void OnException(ExceptionContext filterContext)
    {
        filterContext.ExceptionHandled = true;
        filterContext.Result = new ViewResult { ViewName = "Error" };
    }
}
```

Register it globally:

```
GlobalFilters.Filters.Add(new GlobalExceptionFilter());
```

Or configure in web.config using customErrors.

Q60. How can you create a Custom Route Handler in MVC?

Answer:
Create a class that implements IRouteHandler:

```
public class CustomRouteHandler : IRouteHandler
{
    public IHttpHandler GetHttpHandler(RequestContext requestContext)
    {
        return new MyHttpHandler();
    }
}
```

Then map it in RouteConfig.cs:

```
routes.Add(new Route("customroute/{id}", new CustomRouteHandler()));
```

This gives you complete control over how specific requests are processed.

Q61. How do you implement Dependency Injection in MVC without using third-party libraries?

Answer:
You can manually inject dependencies using constructor injection.

Example Controller:

```
public class ProductController : Controller
{
    private readonly IProductService _productService;

    public ProductController(IProductService productService)
    {
        _productService = productService;
    }

    public ActionResult Index()
    {
        var products = _productService.GetProducts();
        return View(products);
    }
}
```

You instantiate dependencies in a custom IDependencyResolver implementation.

Q62. What is Unit Testing in MVC and how is it achieved?

Answer:
Unit Testing in MVC involves testing Controllers independently of Views and external systems.

Tools commonly used:
NUnit, MSTest, or xUnit for writing tests
Moq for mocking dependencies

Example Test:

```
[TestMethod]
public void Index_Returns_View_With_Products()
{
    // Arrange
    var mockService = new Mock<IProductService>();
    mockService.Setup(s => s.GetProducts()).Returns(new List<Product>());
    var controller = new ProductController(mockService.Object);

    // Act
    var result = controller.Index() as ViewResult;

    // Assert
    Assert.IsNotNull(result);
}
```

Q63. How to restrict access to certain controllers only to Admins in MVC?

Answer:
Use the [Authorize(Roles="Admin")] attribute on the controller or action.

Example:

```
[Authorize(Roles = "Admin")]
public class AdminDashboardController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

Only users in the Admin role can access this Controller.

Q64. What are TempData, ViewData, and ViewBag — and when do you use each?

Answer:

| Feature | ViewData | ViewBag | TempData |
|---|---|---|---|
| Type | Dictionary (string, object) | Dynamic Property | Dictionary (string, object) |
| Lifetime | Current Request | Current Request | Current + Next Request |
| Purpose | Pass data to View | Pass data to View | Pass data across Redirects |
| Syntax | ViewData["Name"] | ViewBag.Name | TempData["Name"] |

Use ViewData/ViewBag for passing data to Views, and TempData to persist data across redirects.

Q65. What is the difference between RedirectToAction and Redirect in MVC?

Answer:

RedirectToAction() redirects to another action within MVC (using routing).
Redirect() sends an HTTP redirect to a URL.

Example RedirectToAction:
```
return RedirectToAction("Index", "Home");
```

Example Redirect:
```
return Redirect("https://www.example.com");
```

Note: RedirectToAction is preferred inside MVC because it respects route mappings.

Q66. What is the difference between Session and TempData?

Answer:

| Aspect | Session | TempData |
|---|---|---|
| Lifetime | Until Browser Closes/Timeout | Only Current + Next Request |
| Purpose | Store large user data (e.g., Profile, Cart) | Store small messages (e.g., success/error) |
| Accessibility | Controller, View, other layers | Mainly for Controller/View |

Session is bigger and longer-lived. TempData is short-lived and light-weight.

Q67. How do you use OutputCache with parameters in MVC?

Answer:
You can cache based on parameters using the VaryByParam property.

Example:

```
[OutputCache(Duration = 60, VaryByParam = "id")]
public ActionResult Details(int id)
{
    return View();
}
```

Here, caching is done separately for each id value passed.

Q68. What is ActionName Attribute in MVC?

Answer:
[ActionName] allows you to rename the URL path of an action without changing the method name.

Example:

```
[ActionName("Details")]
public ActionResult ShowDetails(int id)
{
    return View();
}
```

The URL will map to /ControllerName/Details even though the method is called ShowDetails.

Q69. What is the use of ValidateInput(false) in MVC?

Answer:
ValidateInput(false) disables the ASP.NET MVC request validation for the current action, allowing potentially unsafe input (like HTML).

Example:

```
[HttpPost]
[ValidateInput(false)]
public ActionResult Save(string content)
{
    // Save HTML content
    return View();
}
```

Warning: Use this carefully. Always sanitize user input to prevent XSS.

Q70. What are NonAction Methods in MVC?

Answer:
Methods marked with [NonAction] are NOT considered Controller Actions and cannot be invoked via URLs.

Example:

```
public class ProductController : Controller
{
    [NonAction]
    public string HelperMethod()
    {
        return "Helper Logic";
    }
}
```

HelperMethod() cannot be accessed by routing — it is only used internally within the controller.

Q71. What is a Partial View in MVC?

Answer:
A Partial View is like a reusable sub-View that can be embedded inside a View.

Creating a Partial View:
_Views/Shared/_ProductSummary.cshtml

Inside a normal View:

```
@Html.Partial("_ProductSummary", product)
```

Or, if you want it to run controller logic:

```
@Html.Action("Summary", "Product")
```

Note: Partial Views are ideal for modular UI components like headers, footers, product cards, etc.

Q72. What are Tag Helpers in ASP.NET Core MVC?

Answer:
Tag Helpers allow you to use server-side code directly inside HTML tags for better readability.

Example:

```
<input asp-for="Email" class="form-control" />
```

This will automatically bind to the Email property of the Model.

You no longer need to use Html.TextBoxFor() or Html.EditorFor().
Tag Helpers feel more like HTML-native syntax and are easier to maintain.

Q73. How do you post a file to MVC Controller?

Answer:
Form setup:

```
<form action="/Upload/Submit" method="post" enctype="multipart/form-data">
    <input type="file" name="file" /> <button type="submit">Upload</button>
</form>
```

Controller:

```
[HttpPost]
public ActionResult Submit(HttpPostedFileBase file)
{
    if (file != null && file.ContentLength > 0)
    {
        var path = Path.Combine(Server.MapPath("~/Uploads"),
Path.GetFileName(file.FileName));
        file.SaveAs(path);
    }
    return RedirectToAction("Index");
}
```

You must set enctype="multipart/form-data" in the form for file uploads to work.

Q74. What is Authorization Filter and Authentication Filter in MVC?

Answer:
Authentication Filter: Ensures the user is authenticated (e.g., logged in).
Authorization Filter: Ensures the user is authorized (e.g., has correct roles or permissions).

Example Authorization filter:

```
[Authorize(Roles = "Admin")]
public class AdminController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

Note: In ASP.NET MVC 5, Authentication Filters were introduced separately for better separation of concerns.

Q75. How to implement Forms Authentication in MVC?

Answer:
Set <authentication mode="Forms"> in web.config.

web.config:

```
<authentication mode="Forms">
    <forms loginUrl="~/Account/Login" timeout="30" />
</authentication>
```

Login Controller Action:

```
[HttpPost]
public ActionResult Login(string username, string password)
{
    if (IsValidUser(username, password))
    {
        FormsAuthentication.SetAuthCookie(username, false);
        return RedirectToAction("Index", "Home");
    }
    return View();
}
```

This enables cookie-based login without needing ASP.NET Identity.

Q76. What is RoutePrefix Attribute in MVC?

Answer:
[RoutePrefix] allows you to define a common route prefix for all actions in a controller.

Example:

```
[RoutePrefix("admin/products")]
public class ProductAdminController : Controller
{
    [Route("list")]
    public ActionResult ListProducts()
    {
        return View();
    }
}
```

This action would be accessible via /admin/products/list.

It helps organize and simplify your route configurations.

Q77. How do you handle concurrent updates in MVC applications?

Answer:
Use Optimistic Concurrency:

a. Include a hidden RowVersion field in the View.
b. Compare RowVersion in the database before updating.
c. If mismatch occurs, prompt user to reload data.

Example View:

```
@Html.HiddenFor(m => m.RowVersion)
```

Example Update Action:

```
if (dbEntry.RowVersion != model.RowVersion)
{
    // Conflict detected
}
```

Q78. What is AntiXSS and how is it handled in MVC?

Answer:
AntiXSS libraries help prevent Cross-Site Scripting (XSS) attacks by encoding data properly.
MVC already uses the AntiXSS encoder internally (in newer versions) to sanitize all outputs.

You should:
a. Always use @Model.Property in Views.
b. Avoid @Html.Raw() unless absolutely needed.
c. Validate user input manually if raw HTML must be allowed.

Q79. What is ContentResult in MVC?

Answer:
ContentResult returns plain text, HTML, XML, or JSON as response instead of a View.

Example:

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

You can also set content type:

```
return Content("<h1>Hello</h1>", "text/html");
```

Note: This is useful for returning small dynamic responses without rendering a full View.

Q81. How to implement custom Model Binder in MVC?

Answer:
Create a class that implements IModelBinder:

```
public class CustomDateBinder : IModelBinder
{
    public object BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext)
    {
        var value = bindingContext.ValueProvider.GetValue(bindingContext.ModelName);
        DateTime date;
        if (DateTime.TryParse(value.AttemptedValue, out date))
            return date;
        return null;
    }
}
```

Register it in Global.asax:

```
ModelBinders.Binders.Add(typeof(DateTime), new CustomDateBinder());
```

Now, MVC will use your binder whenever it encounters a DateTime binding.

Q82. What is Cross-Site Request Forgery (CSRF) and how does MVC prevent it?

Answer:
CSRF is an attack where unauthorized commands are transmitted from a user that the website trusts.

MVC prevents it using AntiForgeryToken:

```
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    // Form fields
}
```

Controller must have:

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Save(DataModel model)
{
    // Secure Save Logic
}
```

This ensures form submissions are genuine.

Q83. How to implement Attribute Routing in MVC?

Answer:
Enable it in RouteConfig.cs:

```
routes.MapMvcAttributeRoutes();
```

Then decorate actions or controllers:

```
[Route("products/all")]
public ActionResult ListProducts()
{
    return View();
}
```

You can also add parameters:

```
[Route("products/{id:int}")]
```

Note: Attribute routing gives you more control and cleaner URLs compared to convention-based routing.

Q84. What is Html.RenderAction and Html.Action difference?

Answer:

| Feature | Html.Action | Html.RenderAction |
|---|---|---|
| Return Type | Returns MvcHtmlString (can use in Razor) | Directly writes to Response stream |
| Usage | @Html.Action("ChildAction") | @{ Html.RenderAction("ChildAction"); } |
| Performance | Slightly slower | Slightly faster |

Note: Use RenderAction if you don't need to manipulate the output in Razor.

Q85. How do you maintain session in ASP.NET MVC?

Answer:
Session is maintained using Session object:

```
Session["Username"] = "Karthik";
```

Retrieve it later:

```
string username = Session["Username"] as string;
```

Important:

a. Sessions can be InProc (in memory), StateServer, or SQL Server-based.
b. Use Session carefully to avoid heavy memory usage.

Q86. Difference between Server.Transfer and Response.Redirect?

Answer:

| Feature | Server.Transfer | Response.Redirect |
|---|---|---|
| Action | Transfers request internally | Redirects to a new URL |
| Browser | URL Remains the same | Changes to new URL |
| Performance | Faster (no extra round trip) | Slower (new HTTP request) |

Note: Server.Transfer is rarely used in MVC. Response.Redirect is more common when you need to move to a different URL.

Q87. How do you create strongly typed views in MVC?

Answer:
At the top of the View, declare the Model type:

```
@model MyProject.Models.Product
```

Then use:

```
@Html.DisplayFor(m => m.Name)
@Html.EditorFor(m => m.Price)
```

Strongly typed views provide:

a. IntelliSense support
b. Compile-time checking
c. Better maintainability

Q88. How do you validate a Model using Data Annotations?

Answer:
In Model class:

```
public class Product
{
    [Required]
    public string Name { get; set; }

    [Range(1, 1000)]
    public decimal Price { get; set; }
}
```

In Controller:

```
if (ModelState.IsValid)
{
    // Save to database
}
```

MVC will automatically validate based on the annotations.

Q89. What is Razor View Engine?

Answer:
Razor is a markup syntax that lets you embed server-side code into web pages using @.

Example:

```
<p>Hello, @Model.Name</p>
```

Benefits:

a. Lightweight and clean
b. Easy to read
c. Intellisense support
d. Faster than old WebForms syntax

Note: Razor engine compiles Views into C# classes behind the scenes.

Q90. How do you pass data from Controller to View in MVC?

Answer:
You can use:
a. ViewBag: ViewBag.Message = "Hello";
b. ViewData: ViewData["Message"] = "Hello";
c. TempData (for redirect scenarios)
d. Model (preferred for large structured data)

Example:

```
return View(model);
```

And in View:
```
@Model.PropertyName
```

Note: Passing strongly typed Models is considered the best practice.

Q91. What are View Components in ASP.NET Core MVC?

Answer:
View Components are similar to Partial Views but with more power:

They have their own logic (like mini-controllers).

They don’t participate in routing.

Create a View Component:
```
public class CartSummaryViewComponent : ViewComponent
{
    public IViewComponentResult Invoke()
    {
        var cartCount = 5; // Fetch from DB or Session
        return View(cartCount);
```

```
    }
}

Invoke it in a View:
@await Component.InvokeAsync("CartSummary")

Note: Perfect for building dynamic widgets like shopping carts, side menus, etc.
```

```
Q92. What is Middleware in ASP.NET Core MVC?

Answer:
Middleware is software that sits in the request pipeline to handle HTTP requests and
responses.

Example custom middleware:

public class RequestLoggerMiddleware
{
    private readonly RequestDelegate _next;

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

    public async Task Invoke(HttpContext context)
    {
        Console.WriteLine("Request: " + context.Request.Path);
        await _next(context);
    }
}

Configure in Startup.cs:
app.UseMiddleware<RequestLoggerMiddleware>();

Middlewares are key for Cross-Cutting Concerns (logging, authentication, error handling).
```

```
Q93. What is the difference between AddMvc() and AddControllersWithViews()?

Answer:


Aspect                  AddMvc()                                AddControllersWithViews()
Services Added          Controller + View + API + Razor Pages   Controller + View Only
Razor Pages Support     Yes                                     No
Use When                Building MVC + Razor Pages app          Building pure MVC app


Note: In newer ASP.NET Core versions, prefer AddControllersWithViews() if you are only
doing MVC (no Razor Pages).
```

```
Q94. How can you validate complex nested Models in MVC?

Answer:
Use IValidatableObject in the parent Model:

public class Order : IValidatableObject
{
    public Customer CustomerInfo { get; set; }
    public List<Product> Products { get; set; }

    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
```

```
    {
        if (Products.Count == 0)
            yield return new ValidationResult("At least one product is required.");
    }
}
```

This allows you to define cross-field validation inside your Model itself.

Q95. What is the Repository Pattern in MVC?

Answer:
Repository Pattern abstracts data access logic from business logic.

Example interface:

```
public interface IProductRepository
{
    IEnumerable<Product> GetAll();
    Product GetById(int id);
}
```

Concrete implementation:

```
public class ProductRepository : IProductRepository
{
    public IEnumerable<Product> GetAll()
    {
        // Fetch from DB
    }
}
```

Controller uses Repository:

```
private readonly IProductRepository _repo;
public ProductController(IProductRepository repo)
{
    _repo = repo;
}
```

Advantages:
a. Loose coupling
b. Easier testing
c. Code reuse

Q96. How do you handle multiple Submit buttons in MVC forms?

Answer:
Use name and value attributes and detect in Controller:

```
<form method="post">
    <button name="action" value="Save">Save</button>
    <button name="action" value="Cancel">Cancel</button>
</form>
```

Controller:

```
[HttpPost]
public ActionResult Submit(string action)
{
    if (action == "Save")
        // Save logic
    else if (action == "Cancel")
        // Cancel logic
    return View();
}
```

Simple and effective way to differentiate buttons.

Q97. What is Kestrel in ASP.NET Core?

Answer:
Kestrel is the cross-platform web server used internally by ASP.NET Core.

a. Lightweight
b. High-performance
c. Handles HTTP traffic
d. Can be used directly or behind IIS, Nginx, Apache, etc.

You configure it in Program.cs:
builder.WebHost.UseKestrel();

Q98. What is API Versioning in Web API?

Answer:
API Versioning allows you to maintain multiple versions of your Web APIs without breaking existing clients.

Install package:
Microsoft.AspNetCore.Mvc.Versioning

Configure in Startup.cs:
```
services.AddApiVersioning(options =>
{
    options.AssumeDefaultVersionWhenUnspecified = true;
    options.DefaultApiVersion = new ApiVersion(1, 0);
});
```

Decorate Controllers:
```
[ApiVersion("1.0")]
[Route("api/v{version:apiVersion}/products")]
public class ProductsV1Controller : ControllerBase
```

Now you can support /api/v1/products.

Q99. What are Razor Pages in ASP.NET Core MVC?

Answer:
Razor Pages are a new way of building Web UIs in ASP.NET Core:

a. Page-focused
b. File-based (.cshtml + .cshtml.cs)
c. No need for separate Controller classes

Example page:

/Pages/Products.cshtml

```
@page
@model ProductsModel

<h2>Products</h2>
```

PageModel (Products.cshtml.cs):

```
public class ProductsModel : PageModel
{
    public void OnGet()
    {
        // Load products
```

```
    }
}

Note: Perfect for simple CRUD apps!
```

Q100. What are the best practices you should follow in ASP.NET MVC?

Answer:

a. Always validate user input both on client and server side.
b. Use strongly-typed views, avoid using ViewBag/ViewData unless necessary.
c. Avoid putting business logic in Controllers; use Services.
d. Secure your application with AntiForgeryTokens.
e. Use Dependency Injection for better testability.
f. Apply appropriate Filters (Authorization, Exception Handling).
g. Optimize performance with Output Caching and Bundling.
h. Keep Controllers thin, Services thick (Fat Model - Thin Controller principle).
i. Use Repository Pattern and Unit of Work for data access abstraction.
j. Implement proper error handling and logging (Exception Filters, Global.asax).