

1 ASP.NET MVC Interview Questions by Karthik M
2 =====

3
4 Q1. What is ASP.NET MVC? How is it different from ASP.NET Web Forms?

5
6 Answer:

7
8 ASP.NET MVC stands for Model-View-Controller, a design pattern that separates an application into three components:

- 9
10 a. Model: Handles application data and business logic.
11 b. View: Represents the UI part of the application.
12 c. Controller: Responds to user inputs and interacts with the Model and View.

13
14 Differences from Web Forms:

- 15 -----
16 a. No ViewState in MVC, so pages are lightweight.
17 b. MVC gives full control over HTML, while Web Forms use server controls.
18 c. MVC follows the Separation of Concerns principle.
19 d. MVC applications are more easily testable than Web Forms.

20
21
22
23
24 Q2. Explain the ASP.NET MVC Request Life Cycle.

25
26 Answer:

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

28
29 Step - 1

30 Routing: URL is matched to a route defined in RouteConfig.

31
32 Step - 2

33 Controller Initialization: Appropriate Controller is instantiated.

34
35 Step - 3

36 Action Execution: The action method is invoked.

37
38 Step - 4

39 Result Execution: ActionResult (like ViewResult) is returned.

40
41 Step - 5

42 View Rendering: The View is rendered and returned to the browser.

43
44
45
46 Q3. What is a Controller in MVC?

47
48 Answer:

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

50
51 Example:

```
52 public class ProductController : Controller
53 {
54     public ActionResult Details(int id)
55     {
56         // Fetch product by id
57         return View();
58     }
59 }
```

60
61
62
63
64 Q4. What is an Action Method in MVC?

65
66 Answer:

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

```
68
69 Example:
70 public ActionResult About()
71 {
72     return View();
73 }
74
```

75 An Action Method must be:

- 76 a. Public
- 77 b. Non-static
- 78 c. Not decorated with [NonAction] attribute

79

80

81

82 Q5. What is Routing in ASP.NET MVC?

83

84 Answer:

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

86

87 Example:

```
88 routes.MapRoute(
89     name: "Default",
90     url: "{controller}/{action}/{id}",
91     defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
92 );
93
```

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

95

96

97

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

99

100 Answer:

101 Three common ways to pass data:

102

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

106

107 In the View, you can access:

108

- 109 a. @ViewBag.Message
- 110 b. @ViewData["Message"]
- 111 c. @Model.PropertyName

112

113

114

115

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

117

118 Answer:

119

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

123

124 Example using TempData:

```
125 TempData["SuccessMessage"] = "Record Saved Successfully!";
126
127
128
```

129 Q8. What is a Strongly Typed View?

130

131 Answer:

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

133

134 At the top of the Razor View, you declare the model:

```
135 @model MyApp.Models.Product
136
```

137 Then you can access properties with IntelliSense:

138 @Model.ProductName

139 @Model.Price

140

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

142

143

144

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

146

147 Answer:

148 Razor is the View Engine that allows mixing C# code with HTML easily.

149 Syntax uses @ to indicate C# code.

150

151 Example:

152 @foreach (var item in Model.Products)

153 {

154 <p>@item.Name</p>

155 }

156

157 Advantages:

158

159 a. Cleaner syntax

160 b. Faster rendering

161 c. No need for <% %> syntax like older Web Forms

162

163

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

165

166 Answer:

167

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

169

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

171

172 Example using Html.Partial():

173 @Html.Partial("_ProductPartial", productList)

174

175 Example using Html.RenderPartial():

176 @{ Html.RenderPartial("_ProductPartial", productList); }

177

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

179

180

181

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

183

184 Answer:

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

186

187 Common types:

188

189 a. ViewResult – returns HTML and markup (using Views).

190 b. PartialViewResult – returns a partial View.

191 c. RedirectResult – redirects to another URL.

192 d. RedirectToRouteResult – redirects to another route.

193 e. JsonResult – returns JSON-formatted data.

194 f. FileResult – returns a file to download.

195 g. ContentResult – returns plain text.

196

197

198 Example returning a JSON:

199 return Json(new { id = 1, name = "Product1" }, JsonRequestBehavior.AllowGet);

200

201

202

203

204
205 Q12. How can you restrict an action method to be invoked only by POST request?
206

207 Answer:

208 Use the [HttpPost] attribute above the Action Method.

209 Example:

```
210 [HttpPost]
211 public ActionResult Save(Product model)
212 {
213     // Save logic
214     return RedirectToAction("Index");
215 }
216
```

217 Similarly, use [HttpGet] for GET requests.

218
219
220
221
222 Q13. What is the difference between RedirectToAction() and Redirect()?

223 Answer:

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

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

226 Example using RedirectToAction:

```
227 return RedirectToAction("Index", "Home");
228
```

229 Example using Redirect:

```
230 return Redirect("https://www.google.com");
231
```

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

233
234 Q14. What is the use of the [NonAction] attribute in MVC?

235 Answer:

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

237 Example:

```
238 public class ProductController : Controller
239 {
240     [NonAction]
241     public void HelperMethod()
242     {
243         // Not an action
244     }
245 }
246
```

247
248 Q15. What are Filters in MVC?

249 Answer:

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

251 Types of filters:

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

256 Example:

```
257 [Authorize]
```

```

272 public ActionResult Dashboard()
273 {
274     return View();
275 }
276
277
278
279

```

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:

```

300 public ActionResult Save(Product model)
301 {
302     // Model Binding automatically fills model properties
303     return View();
304 }
305

```

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:

```

317 public class Product
318 {
319     [Required]
320     public string ProductName { get; set; }
321
322     [Range(1, 10000)]
323     public decimal Price { get; set; }
324 }
325

```

Controller code:

```

327 if (ModelState.IsValid)
328 {
329     // Save Product
330 }
331
332
333
334
335
336

```

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

338

339 Answer:

340

341 View() renders the full View including Layout.

342

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

344

345 Example:

346 return View("Index");

347

348 return PartialView("_ProductList", products);

349

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

351

352

353

354

355

356

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

358

359 Answer:

360 You can use:

361

362 [HandleError] attribute on controllers/actions

363

364 customErrors section in web.config

365

366 Example with [HandleError]:

367 [HandleError(View = "Error")]

368 public class ProductController : Controller

369 {

370 public ActionResult Create()

371 {

372 throw new Exception("Something went wrong!");

373 }

374 }

375

376 Also, configure in web.config:

377 <customErrors mode="On" defaultRedirect="~/Home/Error" />

378

379

380

381 Q21. What is a View in MVC?

382

383 Answer:

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

385

386 Example View (Index.cshtml):

387 @model MyApp.Models.Product

388

389 <h2>Product Details</h2> <p>Name: @Model.ProductName</p> <p>Price: @Model.Price</p>

390 The View uses the data passed from the Controller and renders it as HTML.

391

392

393

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

395

396 Answer:

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

398

399 Syntax Examples:

400

401 Inline expressions: @Model.ProductName

402

403 Loop example:

404 @foreach (var item in Model.Products)

```
405 {
406     <p>@item.Name</p>
407 }
408
409 Rules:
410
411 a. Code statements start with @
412 b. No need for <% %> like older ASP.NET Web Forms.
413
414
415 Q23. How do you create a strongly typed View?
416
417 Answer:
418 At the top of the View, declare the model type using @model:
419
420 @model MyApp.Models.Employee
421
422 Then you can access the Model properties with IntelliSense:
423
424 <p>@Model.EmployeeName</p> <p>@Model.Salary</p>
425 Strongly typed Views ensure type safety and better productivity.
426
427
428
429
430 Q24. What are HTML Helpers in MVC?
431
432 Answer:
433 HTML Helpers are methods that simplify rendering standard HTML elements.
434
435 Common examples:
436
437 @Html.TextBoxFor(m => m.ProductName)
438
439 @Html.LabelFor(m => m.ProductName)
440
441 @Html.DropDownListFor(m => m.CategoryId, categoryList)
442
443 They help generate clean and consistent HTML controls in Views.
444
445
446
447
448 Q25. What is the difference between Html.TextBox() and Html.TextBoxFor()?
449
450 Answer:
451
452 Html.TextBox() is loosely typed (string-based), no IntelliSense.
453 Example: @Html.TextBox("ProductName")
454
455 Html.TextBoxFor() is strongly typed (model-based), compile-time checked.
456 Example: @Html.TextBoxFor(m => m.ProductName)
457
458 Prefer TextBoxFor() for strong typing and error checking.
459
460
461
462
463 Q26. How do you create a DropDownList using HTML Helpers?
464
465 Answer:
466 You can create a DropDownList using Html.DropDownListFor().
467
468 Example:
469 @Html.DropDownListFor(m => m.CategoryId, new SelectList(Model.Categories, "CategoryId",
"CategoryName"))
470
471 Here:
472
```

473 m => m.CategoryId is the model property bound.
474
475 Model.Categories is the source list.
476
477
478
479 Q27. What is ViewStart file in MVC?
480
481 Answer:
482 The _ViewStart.cshtml file contains code that should run before every View is rendered.
483
484 Example content of _ViewStart.cshtml:
485
486 @{ Layout = "~/Views/Shared/_Layout.cshtml"; }
487
488 It sets a default Layout for all Views, so you don't have to set Layout individually in every View.
489
490
491
492
493 Q28. What is the purpose of Layout pages in MVC?
494
495 Answer:
496 Layout pages are like Master Pages in Web Forms.
497 They allow you to define a common structure (e.g., header, footer, navigation) and reuse it across multiple Views.
498
499 Example Layout (_Layout.cshtml):
500
501 <!DOCTYPE html>
502 <html>
503 <head>
504 <title>@ViewBag.Title</title>
505 </head>
506 <body> @RenderBody() </body>
507 </html>
508
509 Each View is injected into @RenderBody() placeholder at runtime.
510
511
512
513
514 Q29. How do you render a Partial View manually inside a View?
515
516 Answer:
517 Use either:
518
519 @Html.Partial("_ProductPartial") (returns HTML string)
520
521 @{ Html.RenderPartial("_ProductPartial"); } (writes directly to response)
522
523 Both render _ProductPartial.cshtml inside the parent View.
524
525 Use Partial Views when you want to reuse pieces of UI.
526
527
528
529
530 Q30. How do you perform form submissions in MVC?
531
532 Answer:
533 You create a form using Html.BeginForm().
534
535 Example:
536
537 @using (Html.BeginForm("Save", "Product", FormMethod.Post))
538 {
539 @Html.LabelFor(m => m.ProductName)


```
540         @Html.TextBoxFor(m => m.ProductName)
541
542         <input type="submit" value="Save" />
543     }
544
```

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

546
547
548

549 Q31. What is ModelState in ASP.NET MVC?

550

551 Answer:

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

553

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

555

```
556 if (ModelState.IsValid)
557 {
558     // Save to database
559 }
560 else
561 {
562     // Return validation errors to View
563 }
564
```

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

566
567
568
569

570 Q32. What is Data Annotation in MVC?

571

572 Answer:

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

574

575 Example:

576

```
577 public class Product
578 {
579     [Required(ErrorMessage = "Product Name is required")]
580     public string ProductName { get; set; }
581
582     [Range(1, 10000)]
583     public decimal Price { get; set; }
584 }
585
```

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

587
588
589
590
591

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

593

594 Answer:

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

596

597 Example:

598

```
599 public class NoZeroPriceAttribute : ValidationAttribute
600 {
601     public override bool IsValid(object value)
602     {
603         decimal price = (decimal)value;
604         return price != 0;
605     }
606 }
```

```
607
608 Apply it to a property:
609
610 [NoZeroPrice(ErrorMessage = "Price cannot be zero")]
611 public decimal Price { get; set; }
612
613
614
615
616
617 Q34. What is Unobtrusive JavaScript validation in MVC?
618
619 Answer:
620 Unobtrusive Validation separates JavaScript behavior from HTML markup, resulting in
621 cleaner code.
622
623 To enable it:
624
625 Include jQuery and jQuery Validation libraries.
626
627 Set these keys in web.config:
628
629 <appSettings>
630     <add key="ClientValidationEnabled" value="true" />
631     <add key="UnobtrusiveJavaScriptEnabled" value="true" />
632 </appSettings>
633
634 It automatically enables client-side validation based on Data Annotations.
635
636
637
638 Q35. How do you perform server-side validation in MVC?
639
640 Answer:
641 Server-side validation is done by checking ModelState.IsValid in Controller Actions.
642
643 Example:
644
645 [HttpPost]
646 public ActionResult Save(Product model)
647 {
648     if (ModelState.IsValid)
649     {
650         // Save data
651     }
652     return View(model);
653 }
654
655 Even if client-side validation is disabled, server-side validation ensures data
656 integrity.
657
658
659 Q36. What are strongly typed HTML helpers?
660
661 Answer:
662 Strongly typed helpers are bound to a specific Model property.
663
664 Example:
665
666 @Html.TextBoxFor(m => m.ProductName)
667 @Html.EditorFor(m => m.Price)
668 @Html.DisplayFor(m => m.ProductName)
669
670 Advantages:
671
672 a. Compile-time checking
673 b. IntelliSense support
```

674 c. Less prone to typos

675

676

677

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

679

680 Answer:

681 You can use the JsonResult return type.

682

683 Example:

684

685 public JsonResult GetProduct(int id)

686 {

687 var product = new Product { Id = id, Name = "Pen", Price = 100 };

688 return Json(product, JsonRequestBehavior.AllowGet);

689 }

690

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

692

693

694

695

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

697

698 Answer:

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

700

701 Benefits:

702

703 Loose coupling

704 Easier testing

705 Better maintainability

706

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

708

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

710

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

712

713

714

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

716

717 Answer:

718 You create your own dependency resolver by implementing IDependencyResolver.

719

720 Example:

721

722 public class CustomResolver : IDependencyResolver

723 {

724 public object GetService(Type serviceType)

725 {

726 // Instantiate services here

727 return new ProductService();

728 }

729

730 public IEnumerable<object> GetServices(Type serviceType)

731 {

732 return new List<object>();

733 }

734 }

735

736 And register it in Global.asax:

737

738 DependencyResolver.SetResolver(new CustomResolver());

739

740

741

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.

810 Q43. What is the [AllowAnonymous] attribute in MVC?
811
812 Answer:
813 [AllowAnonymous] allows unauthenticated users to access specific actions or controllers
even if the entire application requires authentication.

814
815 Example:
816
817 [Authorize]
818 public class DashboardController : Controller
819 {
820 public ActionResult Index() { return View(); }
821
822 [AllowAnonymous]
823 public ActionResult Help() { return View(); }
824 }
825

826 Here, Dashboard/Help can be accessed without logging in.
827
828
829
830
831

832 Q44. How do you handle 404 errors in MVC?
833
834 Answer:
835 You can handle 404 errors globally by updating web.config:
836
837 <system.web>
838 <customErrors mode="On" defaultRedirect="~/Error/General">
839 <error statusCode="404" redirect="~/Error/NotFound"/>
840 </customErrors>
841 </system.web>
842

843 You can also create an ErrorController with NotFound and General actions to handle these
gracefully.
844
845
846
847

848 Q45. What is Areas in ASP.NET MVC?
849
850 Answer:
851 Areas allow you to divide a large MVC application into smaller functional sections, each
with its own Controllers, Views, and Models.
852

853 For example:
854
855 Admin Area
856 Customer Area
857 Billing Area
858

859 Each Area has its own AreaRegistration.cs file where its routes are registered.
860

861 Example:
862
863 context.MapRoute(
864 "Admin_default",
865 "Admin/{controller}/{action}/{id}",
866 new { action = "Index", id = UrlParameter.Optional }
867);
868
869
870
871

872 Q46. What is Output Caching in MVC?
873
874 Answer:
875 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',
```

```

943     type: 'POST',
944     data: JSON.stringify({ Id: 1, Name: "Pen" }),
945     contentType: 'application/json; charset=utf-8',
946     success: function(response) {
947         alert(response);
948     }
949 });

```

950 Controller:

```

951 [HttpPost]
952 public JsonResult Save(Product product)
953 {
954     // Save product
955     return Json("Success");
956 }

```

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

958 Answer:

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

961 Example:

```

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

```

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

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

966 Answer:

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

968 Example:

```

969 $.ajax({
970     url: '/Product/GetProductList',
971     type: 'GET',
972     success: function (data) {
973         $('#productDiv').html(data);
974     }
975 });

```

976 In your View:

```

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

```

978 Controller:

```

979 public ActionResult GetProductList()
980 {
981     return PartialView("_ProductListPartial", products);
982 }

```

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

984 Answer:

1012			
1013	Aspect	ViewBag	TempData
1014	=====		
1015	Life Span	Current HTTP Request	Current + Next Request
1016	Usage	Dynamic property (ViewBag.Name)	Dictionary-based (TempData["Name"])
1017	Purpose	Pass data from Controller to View	Pass data between Controllers
1018			
1019			

1020 Q53. What are the different types of Filters in MVC?

1021
1022 Answer:

- 1023
- 1024 a. Authorization Filters – Implement IAuthorizationFilter.
 - 1025 b. Action Filters – Implement IActionFilter.
 - 1026 c. Result Filters – Implement IResultFilter.
 - 1027 d. Exception Filters – Implement IExceptionHandler.

1028
1029 You can create custom filters by inheriting ActionFilterAttribute and overriding methods like OnActionExecuting() and OnResultExecuting().

1030
1031 Example:

```
1032
1033 public class CustomActionFilter : ActionFilterAttribute
1034 {
1035     public override void OnActionExecuting(ActionExecutingContext filterContext)
1036     {
1037         // Your custom logic
1038     }
1039 }
```

1040
1041
1042
1043
1044 Q54. What is Web API and how is it different from MVC?

1045
1046 Answer:

1047			
1048			
1049	Aspect	MVC	Web API
1050	=====		
1051	Purpose	Return Views (HTML)	Return Data (JSON, XML)
1052	Return Type	ActionResult/ViewResult	HttpResponseMessage/IHttpActionResult
1053	Use Case	Web Applications	RESTful Services

1054
1055
1056 MVC is ideal for UI-rendered apps.

1057 Web API is used for services consumed by clients like mobile apps, SPA (Single Page Applications), etc.

1058
1059
1060
1061
1062
1063 Q55. How can you secure Web API endpoints?

1064
1065 Answer:

1066 Options include:

- 1067
- 1068 a. Basic Authentication
 - 1069 b. Token-based Authentication (JWT)
 - 1070 c. OAuth
 - 1071 d. API Key validation
 - 1072 e. Custom Authorization Filters

1073
1074 Example Basic Authentication setup:

```
1075
1076 [Authorize]
1077 public class ProductsController : ApiController
1078 {
```



```
1079         // Secure API
1080     }
1081
1082     You also configure authentication handlers in Startup.cs for ASP.NET Core.
1083
1084
1085
1086
1087     Q56. How to prevent Cross-Site Scripting (XSS) in MVC?
1088
1089     Answer:
1090     MVC automatically encodes output. Use:
1091
1092     @Html.Encode(Model.ProductName)
1093
1094     Or directly in Razor:
1095
1096     <p>@Model.ProductName</p>
1097     To disable encoding (if absolutely required):
1098
1099     @Html.Raw(Model.Description)
1100
1101     Always validate and sanitize inputs to prevent XSS attacks.
1102
1103
1104
1105
1106     Q57. How can you call a Web API from MVC?
1107
1108     Answer:
1109     You can use HttpClient inside a controller.
1110
1111     Example:
1112
1113     HttpClient client = new HttpClient();
1114     client.BaseAddress = new Uri("https://api.example.com/");
1115     HttpResponseMessage response = await client.GetAsync("api/products");
1116     if (response.IsSuccessStatusCode)
1117     {
1118         var products = await response.Content.ReadAsAsync<IEnumerable<Product>>();
1119     }
1120
1121     This helps MVC applications consume external APIs.
1122
1123
1124
1125
1126     Q58. What is the use of Route Constraints in MVC?
1127
1128     Answer:
1129     Route Constraints restrict which URLs match a route based on pattern matching.
1130
1131     Example:
1132
1133     routes.MapRoute(
1134         "Default",
1135         "{controller}/{action}/{id}",
1136         new { controller = "Home", action = "Index", id = UrlParameter.Optional },
1137         new { id = @"\d+" } // id must be numeric
1138     );
1139
1140     This ensures that id must be digits only (regular expression constraint).
1141
1142
1143
1144
1145     Q59. How do you handle Global Exception Handling in MVC?
1146
1147     Answer:
```

```
1148 You can create a custom filter:
1149
1150 public class GlobalExceptionHandler : FilterAttribute, IExceptionHandler
1151 {
1152     public void OnException(ExceptionContext filterContext)
1153     {
1154         filterContext.ExceptionHandled = true;
1155         filterContext.Result = new ViewResult { ViewName = "Error" };
1156     }
1157 }
1158
1159 Register it globally:
1160
1161 GlobalFilters.Filters.Add(new GlobalExceptionHandler());
1162 Or configure in web.config using customErrors.
1163
1164
1165
1166
1167 Q60. How can you create a Custom Route Handler in MVC?
1168
1169 Answer:
1170 Create a class that implements IRouteHandler:
1171
1172 public class CustomRouteHandler : IRouteHandler
1173 {
1174     public IHttpHandler GetHttpHandler(RequestContext requestContext)
1175     {
1176         return new MyHttpHandler();
1177     }
1178 }
1179
1180 Then map it in RouteConfig.cs:
1181
1182 routes.Add(new Route("customroute/{id}", new CustomRouteHandler()));
1183
1184 This gives you complete control over how specific requests are processed.
1185
1186
1187
1188
1189 Q61. How do you implement Dependency Injection in MVC without using third-party
libraries?
1190
1191 Answer:
1192 You can manually inject dependencies using constructor injection.
1193
1194 Example Controller:
1195
1196 public class ProductController : Controller
1197 {
1198     private readonly IProductService _productService;
1199
1200     public ProductController(IProductService productService)
1201     {
1202         _productService = productService;
1203     }
1204
1205     public ActionResult Index()
1206     {
1207         var products = _productService.GetProducts();
1208         return View(products);
1209     }
1210 }
1211
1212 You instantiate dependencies in a custom IDependencyResolver implementation.
1213
1214
1215
```

1216
1217 Q62. What is Unit Testing in MVC and how is it achieved?
1218
1219 Answer:
1220 Unit Testing in MVC involves testing Controllers independently of Views and external systems.

1221
1222 Tools commonly used:
1223 NUnit, MSTest, or xUnit for writing tests
1224 Moq for mocking dependencies
1225

1226 Example Test:
1227
1228 [TestMethod]
1229 public void Index_Returns_View_With_Products()
1230 {
1231 // Arrange
1232 var mockService = new Mock<IProductService>();
1233 mockService.Setup(s => s.GetProducts()).Returns(new List<Product>());
1234 var controller = new ProductController(mockService.Object);
1235
1236 // Act
1237 var result = controller.Index() as ViewResult;
1238
1239 // Assert
1240 Assert.IsNotNull(result);
1241 }
1242
1243
1244
1245

1246 Q63. How to restrict access to certain controllers only to Admins in MVC?
1247
1248 Answer:
1249 Use the [Authorize(Roles="Admin")] attribute on the controller or action.
1250

1251 Example:
1252

1253 [Authorize(Roles = "Admin")]
1254 public class AdminDashboardController : Controller
1255 {
1256 public ActionResult Index()
1257 {
1258 return View();
1259 }
1260 }
1261

1262 Only users in the Admin role can access this Controller.
1263
1264
1265

1266
1267
1268 Q64. What are TempData, ViewData, and ViewBag – and when do you use each?
1269
1270 Answer:
1271
1272

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"]

1278
1279
1280

```

1281 Use ViewData/ViewBag for passing data to Views, and TempData to persist data across
1282 redirects.
1283
1284
1285
1286 Q65. What is the difference between RedirectToAction and Redirect in MVC?
1287
1288 Answer:
1289
1290 RedirectToAction() redirects to another action within MVC (using routing).
1291 Redirect() sends an HTTP redirect to a URL.
1292
1293 Example RedirectToAction:
1294 return RedirectToAction("Index", "Home");
1295
1296 Example Redirect:
1297 return Redirect("https://www.example.com");
1298
1299 Note: RedirectToAction is preferred inside MVC because it respects route mappings.
1300
1301
1302
1303
1304
1305 Q66. What is the difference between Session and TempData?
1306
1307 Answer:
1308
1309 Aspect          Session          TempData
1310 =====
1311
1312 Lifetime          Until Browser Closes/Timeout          Only Current + Next
1313 Request
1314 Purpose          Store large user data (e.g., Profile, Cart)          Store small messages
1315 (e.g., success/error)
1316 Accessibility    Controller, View, other layers          Mainly for
1317 Controller/View
1318
1319 Session is bigger and longer-lived. TempData is short-lived and light-weight.
1320
1321
1322
1323 Q67. How do you use OutputCache with parameters in MVC?
1324
1325 Answer:
1326 You can cache based on parameters using the VaryByParam property.
1327
1328 Example:
1329
1330 [OutputCache(Duration = 60, VaryByParam = "id")]
1331 public ActionResult Details(int id)
1332 {
1333     return View();
1334 }
1335
1336 Here, caching is done separately for each id value passed.
1337
1338
1339
1340 Q68. What is ActionName Attribute in MVC?
1341
1342 Answer:
1343 [ActionName] allows you to rename the URL path of an action without changing the method
1344 name.
1345
1346 Example:

```

```
1344
1345     [ActionName("Details")]
1346     public ActionResult ShowDetails(int id)
1347     {
1348         return View();
1349     }
1350
1351     The URL will map to /ControllerName/Details even though the method is called ShowDetails.
1352
1353
1354
1355
1356     Q69. What is the use of ValidateInput(false) in MVC?
1357
1358     Answer:
1359     ValidateInput(false) disables the ASP.NET MVC request validation for the current action,
1360     allowing potentially unsafe input (like HTML).
1361
1362     Example:
1363
1364     [HttpPost]
1365     [ValidateInput(false)]
1366     public ActionResult Save(string content)
1367     {
1368         // Save HTML content
1369         return View();
1370     }
1371
1372     Warning: Use this carefully. Always sanitize user input to prevent XSS.
1373
1374
1375
1376     Q70. What are NonAction Methods in MVC?
1377
1378     Answer:
1379     Methods marked with [NonAction] are NOT considered Controller Actions and cannot be
1380     invoked via URLs.
1381
1382     Example:
1383
1384     public class ProductController : Controller
1385     {
1386         [NonAction]
1387         public string HelperMethod()
1388         {
1389             return "Helper Logic";
1390         }
1391     }
1392
1393     HelperMethod() cannot be accessed by routing – it is only used internally within the
1394     controller.
1395
1396
1397
1398     Q71. What is a Partial View in MVC?
1399
1400     Answer:
1401     A Partial View is like a reusable sub-View that can be embedded inside a View.
1402
1403     Creating a Partial View:
1404     _Views/Shared/_ProductSummary.cshtml
1405
1406     Inside a normal View:
1407     @Html.Partial("_ProductSummary", product)
1408
1409     Or, if you want it to run controller logic:
1410     @Html.Action("Summary", "Product")
```

1410 Note: Partial Views are ideal for modular UI components like headers, footers, product
1411 cards, etc.

1412
1413

1414 Q72. What are Tag Helpers in ASP.NET Core MVC?

1415

1416 Answer:

1417 Tag Helpers allow you to use server-side code directly inside HTML tags for better
1418 readability.

1418

1419 Example:

1420

1421 <input asp-for="Email" class="form-control" />

1422 This will automatically bind to the Email property of the Model.

1423

1424 You no longer need to use Html.TextBoxFor() or Html.EditorFor().

1425 Tag Helpers feel more like HTML-native syntax and are easier to maintain.

1426

1427

1428

1429 Q73. How do you post a file to MVC Controller?

1430

1431 Answer:

1432 Form setup:

1433

1434 <form action="/Upload/Submit" method="post" enctype="multipart/form-data">

1435 <input type="file" name="file" /> <button type="submit">Upload</button>

1436 </form>

1437

1438 Controller:

1439

1440 [HttpPost]

1441 public ActionResult Submit(HttpPostedFileBase file)

1442 {

1443 if (file != null && file.ContentLength > 0)

1444 {

1445 var path = Path.Combine(Server.MapPath("~/Uploads"),

1446 Path.GetFileName(file.FileName));

1447 file.SaveAs(path);

1448 }

1449 return RedirectToAction("Index");

1450 }

1451

1452 You must set enctype="multipart/form-data" in the form for file uploads to work.

1453

1454

1455

1456

1457 Q74. What is Authorization Filter and Authentication Filter in MVC?

1458

1459 Answer:

1460 Authentication Filter: Ensures the user is authenticated (e.g., logged in).

1461 Authorization Filter: Ensures the user is authorized (e.g., has correct roles or
1462 permissions).

1463

1464 Example Authorization filter:

1465

1466 [Authorize(Roles = "Admin")]

1467 public class AdminController : Controller

1468 {

1469 public ActionResult Index()

1470 {

1471 return View();

1472 }

1473 }

1474

1475 Note: In ASP.NET MVC 5, Authentication Filters were introduced separately for better
1476 separation of concerns.

1474
1475
1476
1477 Q75. How to implement Forms Authentication in MVC?
1478
1479 Answer:
1480 Set <authentication mode="Forms"> in web.config.
1481
1482 web.config:
1483
1484 <authentication mode="Forms">
1485 <forms loginUrl="~/Account/Login" timeout="30" />
1486 </authentication>
1487
1488 Login Controller Action:
1489
1490 [HttpPost]
1491 public ActionResult Login(string username, string password)
1492 {
1493 if (IsValidUser(username, password))
1494 {
1495 FormsAuthentication.SetAuthCookie(username, false);
1496 return RedirectToAction("Index", "Home");
1497 }
1498 return View();
1499 }
1500
1501 This enables cookie-based login without needing ASP.NET Identity.
1502
1503
1504
1505

1506 Q76. What is RoutePrefix Attribute in MVC?
1507
1508 Answer:
1509 [RoutePrefix] allows you to define a common route prefix for all actions in a controller.
1510
1511 Example:
1512
1513 [RoutePrefix("admin/products")]
1514 public class ProductAdminController : Controller
1515 {
1516 [Route("list")]
1517 public ActionResult ListProducts()
1518 {
1519 return View();
1520 }
1521 }
1522
1523 This action would be accessible via /admin/products/list.
1524
1525 It helps organize and simplify your route configurations.
1526
1527
1528
1529

1530 Q77. How do you handle concurrent updates in MVC applications?
1531
1532 Answer:
1533 Use Optimistic Concurrency:
1534
1535 a. Include a hidden RowVersion field in the View.
1536 b. Compare RowVersion in the database before updating.
1537 c. If mismatch occurs, prompt user to reload data.
1538
1539 Example View:
1540
1541 @Html.HiddenFor(m => m.RowVersion)
1542

```
1543 Example Update Action:
1544
1545 if (dbEntry.RowVersion != model.RowVersion)
1546 {
1547     // Conflict detected
1548 }
1549
1550
1551
1552
1553 Q78. What is AntiXSS and how is it handled in MVC?
1554
1555 Answer:
1556 AntiXSS libraries help prevent Cross-Site Scripting (XSS) attacks by encoding data
1557 properly.
1558 MVC already uses the AntiXSS encoder internally (in newer versions) to sanitize all
1559 outputs.
1560
1561 You should:
1562 a. Always use @Model.Property in Views.
1563 b. Avoid @Html.Raw() unless absolutely needed.
1564 c. Validate user input manually if raw HTML must be allowed.
1565
1566
1567 Q79. What is ContentResult in MVC?
1568
1569 Answer:
1570 ContentResult returns plain text, HTML, XML, or JSON as response instead of a View.
1571
1572 Example:
1573
1574 public ContentResult HelloWorld()
1575 {
1576     return Content("Hello, World!");
1577 }
1578
1579 You can also set content type:
1580 return Content("<h1>Hello</h1>", "text/html");
1581
1582 Note: This is useful for returning small dynamic responses without rendering a full View.
1583
1584
1585
1586
1587 Q81. How to implement custom Model Binder in MVC?
1588
1589 Answer:
1590 Create a class that implements IModelBinder:
1591
1592 public class CustomDateBinder : IModelBinder
1593 {
1594     public object BindModel(ControllerContext controllerContext, ModelBindingContext
1595     bindingContext)
1596     {
1597         var value = bindingContext.ValueProvider.GetValue(bindingContext.ModelName);
1598         DateTime date;
1599         if (DateTime.TryParse(value.AttemptedValue, out date))
1600             return date;
1601         return null;
1602     }
1603 }
1604
1605 Register it in Global.asax:
1606 ModelBinders.Binders.Add(typeof(DateTime), new CustomDateBinder());
1607
1608 Now, MVC will use your binder whenever it encounters a DateTime binding.
```



```

1609
1610
1611 Q82. What is Cross-Site Request Forgery (CSRF) and how does MVC prevent it?
1612
1613 Answer:
1614 CSRF is an attack where unauthorized commands are transmitted from a user that the
1615 website trusts.
1616
1617 MVC prevents it using AntiForgeryToken:
1618
1619 @using (Html.BeginForm())
1620 {
1621     @Html.AntiForgeryToken()
1622     // Form fields
1623 }
1624
1625 Controller must have:
1626
1627 [HttpPost]
1628 [ValidateAntiForgeryToken]
1629 public ActionResult Save(DataModel model)
1630 {
1631     // Secure Save Logic
1632 }
1633
1634 This ensures form submissions are genuine.
1635
1636
1637 Q83. How to implement Attribute Routing in MVC?
1638
1639 Answer:
1640 Enable it in RouteConfig.cs:
1641 routes.MapMvcAttributeRoutes();
1642
1643 Then decorate actions or controllers:
1644 [Route("products/all")]
1645 public ActionResult ListProducts()
1646 {
1647     return View();
1648 }
1649
1650 You can also add parameters:
1651 [Route("products/{id:int}")]
1652
1653 Note: Attribute routing gives you more control and cleaner URLs compared to
1654 convention-based routing.
1655
1656
1657 Q84. What is Html.RenderAction and Html.Action difference?
1658
1659 Answer:
1660
1661 Feature                Html.Action                Html.RenderAction
1662 =====
1663
1664 Return Type            Returns MvcHtmlString (can use in Razor)    Directly writes to Response
1665 stream
1666 Usage                  @Html.Action("ChildAction")                @{
1667 Html.RenderAction("ChildAction"); }
1668 Performance            Slightly slower                            Slightly faster
1669
1670 Note: Use RenderAction if you don't need to manipulate the output in Razor.
1671
1672 Q85. How do you maintain session in ASP.NET MVC?

```

```

1673
1674 Answer:
1675 Session is maintained using Session object:
1676
1677 Session["Username"] = "Karthik";
1678
1679 Retrieve it later:
1680 string username = Session["Username"] as string;
1681
1682 Important:
1683
1684 a. Sessions can be InProc (in memory), StateServer, or SQL Server-based.
1685 b. Use Session carefully to avoid heavy memory usage.
1686
1687
1688
1689
1690 Q86. Difference between Server.Transfer and Response.Redirect?
1691
1692 Answer:
1693
1694
1695 Feature          Server.Transfer          Response.Redirect
1696 Action           Transfers request internally    Redirects to a new URL
1697 Browser          URL Remains the same           Changes to new URL
1698 Performance      Faster (no extra round trip)    Slower (new HTTP request)
1699
1700 Note: Server.Transfer is rarely used in MVC. Response.Redirect is more common when you
1701 need to move to a different URL.
1702
1703
1704
1705 Q87. How do you create strongly typed views in MVC?
1706
1707 Answer:
1708 At the top of the View, declare the Model type:
1709
1710 @model MyProject.Models.Product
1711
1712 Then use:
1713
1714 @Html.DisplayFor(m => m.Name)
1715 @Html.EditorFor(m => m.Price)
1716
1717 Strongly typed views provide:
1718
1719 a. IntelliSense support
1720 b. Compile-time checking
1721 c. Better maintainability
1722
1723
1724
1725
1726 Q88. How do you validate a Model using Data Annotations?
1727
1728 Answer:
1729 In Model class:
1730
1731 public class Product
1732 {
1733     [Required]
1734     public string Name { get; set; }
1735
1736     [Range(1, 1000)]
1737     public decimal Price { get; set; }
1738 }
1739
1740 In Controller:

```

```
1741
1742 if (ModelState.IsValid)
1743 {
1744     // Save to database
1745 }
1746
1747 MVC will automatically validate based on the annotations.
1748
1749
1750
1751
1752 Q89. What is Razor View Engine?
1753
1754 Answer:
1755 Razor is a markup syntax that lets you embed server-side code into web pages using @.
1756
1757 Example:
1758
1759 <p>Hello, @Model.Name</p>
1760
1761 Benefits:
1762
1763 a. Lightweight and clean
1764 b. Easy to read
1765 c. Intellisense support
1766 d. Faster than old WebForms syntax
1767
1768 Note: Razor engine compiles Views into C# classes behind the scenes.
1769
1770
1771
1772
1773 Q90. How do you pass data from Controller to View in MVC?
1774
1775 Answer:
1776 You can use:
1777 a. ViewBag: ViewBag.Message = "Hello";
1778 b. ViewData: ViewData["Message"] = "Hello";
1779 c. TempData (for redirect scenarios)
1780 d. Model (preferred for large structured data)
1781
1782 Example:
1783
1784 return View(model);
1785
1786 And in View:
1787 @Model.PropertyName
1788
1789 Note: Passing strongly typed Models is considered the best practice.
1790
1791
1792
1793
1794 Q91. What are View Components in ASP.NET Core MVC?
1795
1796 Answer:
1797 View Components are similar to Partial Views but with more power:
1798
1799 They have their own logic (like mini-controllers).
1800
1801 They don't participate in routing.
1802
1803 Create a View Component:
1804 public class CartSummaryViewComponent : ViewComponent
1805 {
1806     public IViewComponentResult Invoke()
1807     {
1808         var cartCount = 5; // Fetch from DB or Session
1809         return View(cartCount);
1810     }
1811 }
```

```

1810     }
1811 }
1812
1813 Invoke it in a View:
1814 @await Component.InvokeAsync("CartSummary")
1815
1816 Note: Perfect for building dynamic widgets like shopping carts, side menus, etc.
1817
1818
1819
1820
1821 Q92. What is Middleware in ASP.NET Core MVC?
1822
1823 Answer:
1824 Middleware is software that sits in the request pipeline to handle HTTP requests and
    responses.
1825
1826 Example custom middleware:
1827
1828 public class RequestLoggerMiddleware
1829 {
1830     private readonly RequestDelegate _next;
1831
1832     public RequestLoggerMiddleware(RequestDelegate next)
1833     {
1834         _next = next;
1835     }
1836
1837     public async Task Invoke(HttpContext context)
1838     {
1839         Console.WriteLine("Request: " + context.Request.Path);
1840         await _next(context);
1841     }
1842 }
1843
1844 Configure in Startup.cs:
1845 app.UseMiddleware<RequestLoggerMiddleware>();
1846
1847 Middlewares are key for Cross-Cutting Concerns (logging, authentication, error handling).
1848
1849
1850
1851 Q93. What is the difference between AddMvc() and AddControllersWithViews()?
1852
1853 Answer:
1854
1855
1856 Aspect                AddMvc()                AddControllersWithViews()
1857 Services Added         Controller + View + API + Razor Pages  Controller + View Only
1858 Razor Pages Support    Yes                            No
1859 Use When               Building MVC + Razor Pages app        Building pure MVC app
1860
1861
1862 Note: In newer ASP.NET Core versions, prefer AddControllersWithViews() if you are only
    doing MVC (no Razor Pages).
1863
1864
1865
1866 Q94. How can you validate complex nested Models in MVC?
1867
1868 Answer:
1869 Use IValidatableObject in the parent Model:
1870
1871 public class Order : IValidatableObject
1872 {
1873     public Customer CustomerInfo { get; set; }
1874     public List<Product> Products { get; set; }
1875
1876     public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)

```

```
1877     {
1878         if (Products.Count == 0)
1879             yield return new ValidationResult("At least one product is required.");
1880     }
1881 }
1882
```

1883 This allows you to define cross-field validation inside your Model itself.

1884

1885

1886

1887 Q95. What is the Repository Pattern in MVC?

1888

1889 Answer:

1890 Repository Pattern abstracts data access logic from business logic.

1891

1892 Example interface:

```
1893
1894 public interface IProductRepository
1895 {
1896     IEnumerable<Product> GetAll();
1897     Product GetById(int id);
1898 }
1899
```

1900 Concrete implementation:

```
1901
1902 public class ProductRepository : IProductRepository
1903 {
1904     public IEnumerable<Product> GetAll()
1905     {
1906         // Fetch from DB
1907     }
1908 }
1909
```

1910 Controller uses Repository:

```
1911
1912 private readonly IProductRepository _repo;
1913 public ProductController(IProductRepository repo)
1914 {
1915     _repo = repo;
1916 }
1917
```

1918 Advantages:

- 1919 a. Loose coupling
- 1920 b. Easier testing
- 1921 c. Code reuse

1922

1923 Q96. How do you handle multiple Submit buttons in MVC forms?

1924

1925 Answer:

1926 Use name and value attributes and detect in Controller:

```
1927
1928 <form method="post">
1929     <button name="action" value="Save">Save</button>
1930     <button name="action" value="Cancel">Cancel</button>
1931 </form>
1932
```

1933 Controller:

```
1934
1935 [HttpPost]
1936 public ActionResult Submit(string action)
1937 {
1938     if (action == "Save")
1939         // Save logic
1940     else if (action == "Cancel")
1941         // Cancel logic
1942     return View();
1943 }
1944
```

1945 Simple and effective way to differentiate buttons.

1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

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
```

2014 }
2015 }

2016
2017 Note: Perfect for simple CRUD apps!

2018
2019
2020
2021
2022 Q100. What are the best practices you should follow in ASP.NET MVC?

2023
2024 Answer:

- 2025
2026 a. Always validate user input both on client and server side.
2027 b. Use strongly-typed views, avoid using ViewBag/ViewData unless necessary.
2028 c. Avoid putting business logic in Controllers; use Services.
2029 d. Secure your application with AntiForgeryTokens.
2030 e. Use Dependency Injection for better testability.
2031 f. Apply appropriate Filters (Authorization, Exception Handling).
2032 g. Optimize performance with Output Caching and Bundling.
2033 h. Keep Controllers thin, Services thick (Fat Model - Thin Controller principle).
2034 i. Use Repository Pattern and Unit of Work for data access abstraction.
2035 j. Implement proper error handling and logging (Exception Filters, Global.asax).
2036