1. Introduction to ASP.NET MVC

MVC Architecture

- **Question:** What is ASP.NET MVC, and how does it differ from ASP.NET Web Forms?
  Answer: ASP.NET MVC is a framework for building web applications using the Model-View-Controller architecture. It differs from Web Forms by providing better control over HTML, no ViewState, and support for Test-Driven Development (TDD). Web Forms is event-driven, while MVC follows a request-response model.

Components of MVC

- **Question:** What are the three main components of the MVC architecture, and what are their roles?
  Answer:
  - **Model:** Represents the data and business logic of the application.
  - **View:** Displays the data (UI) and sends user input to the controller.
  - **Controller:** Handles user input, interacts with the model, and returns the appropriate view.

---

2. Controllers in ASP.NET MVC

Controller Basics

- **Question:** What is the role of a controller in ASP.NET MVC?
  Answer: A controller in MVC handles incoming HTTP requests, interacts with the model to process the data, and returns the appropriate view to the user.

Action Methods

- **Question:** What are action methods in ASP.NET MVC?
  Answer: Action methods are methods in a controller class that handle requests and return responses. Each action method typically corresponds to a unique URL or route.
- **Question:** How do you specify an action method for different HTTP verbs (GET, POST, PUT, DELETE)?
  Answer: You use attributes like `[HttpGet]`, `[HttpPost]`, `[HttpPut]`, and `[HttpDelete]` to specify which HTTP verb an action method responds to.

```csharp
[HttpPost] public ActionResult SubmitForm() { // Handle form submission }
```

---

3. Routing in ASP.NET MVC

Default Routing

- **Question:** How does routing work in ASP.NET MVC?
  Answer: Routing maps incoming URLs to controller actions. The default route is defined in the `RouteConfig` file, mapping URLs to the `{controller}/{action}/{id}` pattern:

```csharp
routes.MapRoute( name: "Default", url: "{controller}/{action}/{id}", defaults: new
```

```
{ controller = "Home", action = "Index", id = UrlParameter.Optional } );
```

Attribute Routing

- **Question:** What is attribute routing, and how is it used in ASP.NET MVC?
  Answer: Attribute routing allows you to define routes using attributes directly on action methods and controllers, providing more control and clarity.

```csharp
[Route("products/{id}")] public ActionResult Details(int id) { // Show product details }
```

---

4. Models in ASP.NET MVC

Model Binding

- **Question:** What is model binding in ASP.NET MVC?
  Answer: Model binding automatically maps data from HTTP requests (form fields, query strings, etc.) to action method parameters or model properties. MVC handles type conversion and validation automatically.

Strongly Typed Models

- **Question:** What is a strongly-typed view in ASP.NET MVC?
  Answer: A strongly-typed view is a view that is bound to a specific model type. The model is passed from the controller to the view, enabling IntelliSense and type safety when accessing model properties.

ViewModels

- **Question:** What is a ViewModel, and how is it used in ASP.NET MVC?
  Answer: A ViewModel is a class that contains data specifically for displaying in a view. It can combine properties from multiple models or contain additional properties needed for the view. It helps decouple the view from the domain model.

---

5. Views in ASP.NET MVC

Razor View Engine

- **Question:** What is the Razor view engine, and how does it differ from the traditional ASPX view engine?
  Answer: The Razor view engine is a lightweight markup engine introduced in MVC 3. It uses the `@` symbol to embed C# code into HTML. Razor syntax is more concise and readable compared to the traditional ASPX engine, which uses `<%= %>` for code rendering.

Partial Views

- **Question:** What is a partial view in ASP.NET MVC, and how is it used?
  Answer: A partial view is a reusable view that renders a portion of the page. It can be included in other views using the `@Html.Partial()` or `@Html.RenderPartial()` methods. It's useful for rendering common UI components like navigation bars or forms.

Layouts

- **Question:** What is a layout page in ASP.NET MVC?
  Answer: A layout page defines a common structure (e.g., header, footer) for multiple views. It's similar to a master page in Web Forms. Views can inherit a layout to ensure a consistent structure across pages:

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

---

6. Data Annotations and Validation

Model Validation

- **Question:** How does model validation work in ASP.NET MVC?
  Answer: Model validation is achieved using data annotations. Attributes like `[Required]`, `[StringLength]`, `[Range]`, and `[EmailAddress]` are applied to model properties, and MVC automatically checks these validations when binding data in a form.

Custom Validation

- **Question:** How do you create custom validation in ASP.NET MVC?
  Answer: You can create custom validation by implementing the `IValidatableObject` interface or creating a custom attribute by inheriting from `ValidationAttribute` and overriding the `IsValid` method.

```csharp
public class CustomAgeAttribute : ValidationAttribute { protected override ValidationResult IsValid(object value, ValidationContext validationContext) { int age = (int)value; if (age < 18) { return new ValidationResult("Age must be at least 18."); } return ValidationResult.Success; } }
```

---

7. Action Results and View Results

ActionResult

- **Question:** What is `ActionResult` in ASP.NET MVC?
  Answer: `ActionResult` is the base class for all action method return types. It can represent different kinds of responses, such as HTML views (`ViewResult`), JSON data (`JsonResult`), file downloads (`FileResult`), or redirects (`RedirectResult`).

ViewResult

- **Question:** How does `ViewResult` work in ASP.NET MVC?
  Answer: `ViewResult` is a type of `ActionResult` that renders a view. It is the most common return type in an MVC controller, typically used to display HTML:

```csharp
return View("Index");
```

JSON and File Results

- **Question:** How do you return JSON data from an action method in ASP.NET MVC?
  Answer: You can return JSON data using the `Json()` method. This is useful for AJAX requests:

```csharp
return Json(new { Name = "John", Age = 25 });
```

- **Question:** How do you return a file in ASP.NET MVC?
  Answer: You can return a file using the `FileResult` class, which allows users to download a file from the server:

```csharp
return File(filePath, "application/pdf", "document.pdf");
```

---

8. Filters in ASP.NET MVC

Action Filters

- **Question:** What are action filters in ASP.NET MVC?
  Answer: Action filters allow you to run code before or after an action method executes. Common filters include `AuthorizeAttribute`, `HandleErrorAttribute`, and `OutputCacheAttribute`. Custom filters can be created by inheriting from `ActionFilterAttribute`.

Types of Filters

- **Question:** What are the different types of filters in ASP.NET MVC?
  Answer:
  - **Authorization Filters:** Handle user authentication and authorization.
  - **Action Filters:** Execute logic before or after action methods.
  - **Result Filters:** Execute code before or after a view result is processed.
  - **Exception Filters:** Handle exceptions raised during action execution.

Creating Custom Filters

- **Question:** How do you create a custom filter in ASP.NET MVC?
  Answer: Custom filters are created by inheriting from `ActionFilterAttribute` or `IActionFilter` and overriding the `OnActionExecuting` or `OnActionExecuted` methods:

```csharp
public class CustomLogAttribute : ActionFilterAttribute { public override void OnActionExecuting(ActionExecutingContext filterContext) { // Custom logic before the action executes } }
```

---

9. Authentication and Authorization in ASP.NET MVC

Forms Authentication

- **Question:** How does forms authentication work in ASP.NET MVC?
  Answer: Forms authentication redirects unauthorized users to a login page. After login, an authentication ticket (cookie) is created to identify the user. This is configured in the `Web.config`:

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

Role-Based Authorization

- **Question:** How is role-based authorization implemented in ASP.NET MVC?
  Answer: Role-based authorization allows you to restrict access to controller actions based on the user's role. You use the `[Authorize]` attribute to define roles that are allowed access:

```
[Authorize(Roles = "Admin")] public ActionResult AdminDashboard() { // Only users in the "Admin" role can access this action return View(); }
```

Custom Authentication

- **Question:** How do you implement custom authentication in ASP.NET MVC?
  Answer: Custom authentication can be implemented by creating your own logic to validate users and set the authentication ticket manually. This is done using `FormsAuthentication.SetAuthCookie` or `FormsAuthenticationTicket` to create an authentication cookie after verifying user credentials.

---

10. **State Management in ASP.NET MVC**

TempData, ViewData, and ViewBag

- **Question:** What is the difference between `TempData`, `ViewData`, and `ViewBag` in ASP.NET MVC?
  Answer:
  - **ViewData:** A dictionary object used to pass data from the controller to the view. Data is available only for the current request.
  - **ViewBag:** A dynamic wrapper around `ViewData` that allows data to be accessed with properties instead of keys. Data is also available only for the current request.
  - **TempData:** Used to store data temporarily between multiple requests (e.g., after a redirect). It is based on `Session` and persists until read.

Session Management

- **Question:** How does session management work in ASP.NET MVC?
  Answer: Session management in MVC works similarly to Web Forms. You can store and retrieve

data across multiple requests using the `Session` object:

```
Session["UserName"] = "John"; string name = Session["UserName"].ToString();
```

---

11. **AJAX and jQuery in ASP.NET MVC**

AJAX Helpers

- **Question:** What are AJAX helpers in ASP.NET MVC, and how do they work?
  Answer: AJAX helpers are used to perform asynchronous requests in an MVC application without refreshing the page. Common helpers include `Ajax.BeginForm` and `Ajax.ActionLink` for making asynchronous form submissions and links:

```
@using (Ajax.BeginForm("SubmitForm", new AjaxOptions { UpdateTargetId = "result"
})) { <input type="submit" value="Submit" /> }
```

jQuery Integration

- **Question:** How is jQuery used in ASP.NET MVC?
  Answer: jQuery can be used to perform client-side tasks such as DOM manipulation, AJAX calls, and event handling in ASP.NET MVC. You can make an AJAX request using jQuery like this:

```
$.ajax({ url: '/Home/GetData', type: 'GET', success: function(data) {
$('#result').html(data); } });
```

---

12. **Dependency Injection (DI) in ASP.NET MVC**

DI in MVC

- **Question:** How does Dependency Injection work in ASP.NET MVC?
  Answer: Dependency Injection (DI) is implemented in ASP.NET MVC using Inversion of Control (IoC) containers. Popular containers include Unity, Ninject, and Autofac. These frameworks manage the lifetime of dependencies and inject them into controllers or services.

```
public class HomeController : Controller { private readonly IMyService _myService;
public HomeController(IMyService myService) { _myService = myService; } }
```

Setting up DI

- **Question:** How do you set up dependency injection in ASP.NET MVC?
  Answer: You register your dependencies in the IoC container during application startup in `Global.asax`. For example, using Unity:

```
var container = new UnityContainer(); container.RegisterType<IMyService, MyService>
```

```
(); DependencyResolver.SetResolver(new UnityDependencyResolver(container));
```

---

13. **Unit Testing in ASP.NET MVC**

Unit Testing Controllers

- **Question:** How do you unit test controllers in ASP.NET MVC?
  Answer: Unit testing controllers in ASP.NET MVC involves testing the action methods independently of other components. You can mock dependencies and test the return type and data of the action method. For example:

```
[TestMethod] public void Index_ReturnsViewResult() { var controller = new HomeController(); var result = controller.Index() as ViewResult; Assert.IsNotNull(result); }
```

Mocking Dependencies

- **Question:** How do you mock dependencies in ASP.NET MVC for unit testing?
  Answer: Dependencies like services or repositories can be mocked using frameworks like Moq or NSubstitute. You inject these mocked objects into your controllers to isolate the unit of work being tested.

```
var mockService = new Mock<IMyService>(); var controller = new HomeController(mockService.Object);
```

---

14. **Performance Optimization in ASP.NET MVC**

Output Caching

- **Question:** How do you implement output caching in ASP.NET MVC?
  Answer: Output caching stores the result of a controller action and serves it from the cache for subsequent requests. This can be implemented using the `[OutputCache]` attribute:

```
[OutputCache(Duration = 60, VaryByParam = "none")] public ActionResult Index() { return View(); }
```

Bundling and Minification

- **Question:** What is bundling and minification in ASP.NET MVC, and how does it improve performance?
  Answer: Bundling combines multiple JavaScript and CSS files into a single file, and minification reduces file size by removing unnecessary characters like whitespace. These techniques reduce the number of HTTP requests and file sizes, improving page load times.

```
BundleConfig.RegisterBundles(BundleTable.Bundles);
```

---

15. **Handling Exceptions in ASP.NET MVC**

Global Exception Handling

- **Question:** How do you handle global exceptions in ASP.NET MVC?
  Answer: Global exceptions can be handled by overriding the `OnException` method in a controller or by using the `HandleError` attribute. You can also use global filters to catch and log exceptions:

```
protected override void OnException(ExceptionContext filterContext) { // Log the error filterContext.ExceptionHandled = true; filterContext.Result = RedirectToAction("Error"); }
```

Custom Error Pages

- **Question:** How do you configure custom error pages in ASP.NET MVC?
  Answer: Custom error pages can be configured using the `HandleError` attribute or in the `Web.config` file:

```
<customErrors mode="On"> <error statusCode="404" redirect="~/Error/NotFound" /> <error statusCode="500" redirect="~/Error/ServerError" /> </customErrors>
```

---

16. **Entity Framework Integration in ASP.NET MVC**

Using Entity Framework

- **Question:** How do you integrate Entity Framework in an ASP.NET MVC application?
  Answer: Entity Framework is used as an ORM to interact with the database in an MVC application. You can inject a `DbContext` into a repository or directly into a controller for data access.

```
public class ApplicationDbContext : DbContext { public DbSet<Product> Products { get; set; } }
```

Lazy Loading and Eager Loading

- **Question:** What is the difference between lazy loading and eager loading in Entity Framework?
  Answer:
  - **Lazy Loading:** Related data is loaded when it is accessed, reducing the initial query load but potentially causing additional database queries.
  - **Eager Loading:** Related data is loaded along with the main entity using `Include()`, reducing the number of queries but increasing the size of the initial query.

---

17. **Advanced Concepts in ASP.NET MVC**

Areas in ASP.NET MVC

- **Question:** What are areas in ASP.NET MVC, and why are they used?
  Answer: Areas in ASP.NET MVC help organize large applications by dividing them into smaller

functional sections, each with its own controllers, views, and models. This allows for better separation of concerns and code organization.

Asynchronous Controllers

- **Question:** How do asynchronous controllers work in ASP.NET MVC?
  Answer: Asynchronous controllers handle long-running tasks without blocking the request thread, improving application scalability. You can define an asynchronous action method using the `async` and `await` keywords:

```
public async Task<ActionResult> GetData() { var data = await
service.GetDataAsync(); return View(data); }
```
