**Basic Questions**

**1. What is ASP.NET Core? How is it different from ASP.NET?**

* **Answer:**
  + ASP.NET Core is a cross-platform, open-source framework for building modern, cloud-based, and internet-connected applications like web apps, IoT apps, and APIs.
* **Differences from ASP.NET:**
  + **Cross-platform**: Runs on Windows, macOS, and Linux.
  + **Performance**: Faster and optimized for modern workloads.
  + **Modular Framework**: Uses NuGet packages, allowing modular deployments.
  + **Unified Framework**: Combines MVC and Web API into a single framework.
* **Explanation**: ASP.NET Core was introduced to overcome the limitations of ASP.NET by improving scalability, performance, and deployment flexibility.

**2. What is middleware in ASP.NET Core?**

* **Answer:**
  + Middleware is software that sits between the server and the application, handling requests and responses.
  + Middleware components are executed in a pipeline, where each one can:
    - Process the incoming request.
    - Pass the request to the next middleware in the pipeline.
    - Modify the response before sending it back to the client.
* **Example**:

csharp

Copy code

app.Use(async (context, next) =>

{

// Custom middleware logic

await next.Invoke(); // Pass to the next middleware

});

* **Explanation**: Middleware allows you to control how requests and responses are handled, enabling custom logic like logging, authentication, etc.

**3. What are Dependency Injection and its benefits in ASP.NET Core?**

* **Answer:**
  + Dependency Injection (DI) is a design pattern that allows classes to receive their dependencies from an external source rather than creating them internally.
* **Benefits:**
  + **Loose coupling**: Makes code easier to maintain and test.
  + **Improved testability**: Dependencies can be mocked or replaced in tests.
  + **Centralized configuration**: Services are registered in Startup.cs.
* **Example**:

csharp

Copy code

services.AddTransient<IMyService, MyService>();

* **Explanation**: DI is built into ASP.NET Core, making it easy to manage dependencies and follow best practices.

**4. What is the purpose of the Startup.cs file?**

* **Answer**: The Startup.cs file is the entry point for configuring the application. It contains:
  1. ConfigureServices method: Used to configure services (e.g., DI, authentication, etc.).
  2. Configure method: Used to define the middleware pipeline.
* **Example**:

csharp

Copy code

public void ConfigureServices(IServiceCollection services)

{

services.AddControllersWithViews();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapDefaultControllerRoute();

});

}

* **Explanation**: This file sets up everything your application needs, such as routing, services, and middleware.

**Intermediate Questions**

**5. Explain the difference between IActionResult and ActionResult<T>.**

* **Answer:**
  + **IActionResult**: A general interface allowing different types of results (e.g., ViewResult, JsonResult, etc.).
  + **ActionResult<T>**: A generic type introduced in ASP.NET Core 2.1 that provides strong typing for API responses, useful for returning data (e.g., objects) with a status code.
* **Example**:

csharp

Copy code

// IActionResult

public IActionResult Get()

{

return View();

}

// ActionResult<T>

public ActionResult<string> Get()

{

return "Hello, World!";

}

* **Explanation**: Use ActionResult<T> for APIs to enforce type safety, while IActionResult is suitable for flexible result handling.

**6. What are Tag Helpers in ASP.NET Core?**

* **Answer:**
  + Tag Helpers are server-side components that enable adding server-side logic to HTML elements.
  + They make views more readable and cleaner by embedding logic directly into HTML.
* **Example**:

html

Copy code

<form asp-controller="Home" asp-action="Submit">

<input type="text" asp-for="Name" />

</form>

* **Explanation**: Tag Helpers simplify the process of creating dynamic views without using traditional Razor syntax (e.g., @Html.ActionLink).

**7. What is Kestrel in ASP.NET Core?**

* **Answer:**
  + Kestrel is the default, lightweight, and cross-platform web server in ASP.NET Core.
  + It is designed for high performance and supports HTTP/2 and HTTPS.
* **Explanation**: Kestrel acts as the in-process server for handling web requests and can be paired with IIS or used standalone.

**8. What is the purpose of the appsettings.json file?**

* **Answer**: The appsettings.json file is used to store configuration data like connection strings, logging settings, and application-specific settings.
* **Example**:

json

Copy code

{

"ConnectionStrings": {

"DefaultConnection": "Server=myServer;Database=myDb;User=myUser;Password=myPass;"

},

"Logging": {

"LogLevel": {

"Default": "Warning"

}

}

}

* **Explanation**: It centralizes configuration, and settings can be accessed using the IConfiguration interface.

**Advanced Questions**

**9. How do you implement authentication in ASP.NET Core?**

* **Answer**: ASP.NET Core provides built-in support for authentication via middleware and identity services.
* **Steps**:
  1. **Add authentication services in Startup.cs:**

csharp

Copy code

services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)

.AddCookie();

* 1. **Configure middleware:**

csharp

Copy code

app.UseAuthentication();

app.UseAuthorization();

* 1. **Use authentication attributes:**

csharp

Copy code

[Authorize]

public IActionResult SecureAction()

{

return View();

}

* **Explanation**: Authentication is modular and can be integrated with JWT, OAuth, or external providers like Google.

**10. What are Filters in ASP.NET Core?**

* **Answer**:
  + Filters are used to execute logic at different stages of the request processing pipeline (e.g., before/after an action, during exceptions).
* **Types**:
  + Authorization filters
  + Action filters
  + Exception filters
  + Resource filters
* **Example**:

csharp

Copy code

public class CustomActionFilter : IActionFilter

{

public void OnActionExecuting(ActionExecutingContext context) { /\* Logic \*/ }

public void OnActionExecuted(ActionExecutedContext context) { /\* Logic \*/ }

}

* **Explanation**: Filters allow you to encapsulate reusable logic like logging or error handling.