1. **Why do we use ActionResult? Support your answer with scenarios or problems**

- We use ActionResult in ASP.NET MVC because it provides flexibility in returning different types of responses from a controller action. Instead of being limited to one specific return type (like ViewResult), ActionResult can return various results such as ViewResult, JsonResult, RedirectResult, or FileResult.
- Scenario*:*
  If you are building an API endpoint that sometimes needs to return a JSON response and sometimes a file, using ActionResult allows the controller to return the appropriate response type without changing the method signature.

2. What does the HttpContext request and response message consist of?

- **Request:** Contains information sent by the client to the server, including the URL, HTTP method (GET, POST, etc.), headers, cookies, query strings, and body data.
- **Response:** Contains information sent from the server back to the client, including status code (200, 404, 500), headers, cookies, content type, and the body of the response (HTML, JSON, file, etc.).

3. What is the difference between HTTP and HTTPS?

- **HTTP (Hypertext Transfer Protocol):** Data is transferred in plain text and can be intercepted by attackers.
- **HTTPS (Hypertext Transfer Protocol Secure):** Uses SSL/TLS encryption to secure the communication between client and server, protecting data integrity and confidentiality.

4. What are segments and fragments in a URL? Give a real URL example

- **Segments:** Parts of the URL path separated by slashes `/`. Example: In `https://example.com/products/electronics/phones`, the segments are `products`, `electronics`, and `phones`.
- **Fragments:** The portion of a URL after the `#` symbol, usually used to navigate to a section within the page. Example: `https://example.com/products#reviews` → the fragment is `reviews`.

5. What is Builder and Dependency Injection with a real-life example? Clarify it

- **Builder Pattern:** Used to construct complex objects step by step.
- **Dependency Injection (DI):** A design pattern where dependencies (services, objects) are provided to a class instead of being created inside the class.

Real-life example:
In an e-commerce application, a `PaymentService` may depend on `IPaymentGateway`. Instead of hardcoding the payment gateway inside the service, DI injects the implementation (e.g., PayPal, Stripe) at runtime. This improves flexibility and testability.

6. What is the difference between Web Pages (Razor) and MVC? State two business cases and compare them

- **Web Pages (Razor):** A lightweight framework for building simple, single-page websites using C# and Razor syntax directly in `.cshtml` files.
- **MVC (Model-View-Controller):** A structured framework for building large, scalable applications with clear separation of concerns.

Business Cases:

1. **Small personal blog:** Razor Pages are better because they are simple and fast to set up.
2. **Enterprise e-commerce system:** MVC is better because it separates logic, UI, and data, making the system maintainable and scalable.

7. What is the Content-Type in the Response Message, where do we use it, and why?
The `Content-Type` header tells the client how to interpret the data returned from the server. Examples include:

- `text/html` → for HTML documents
- `application/json` → for JSON APIs
- `image/png` → for PNG images

It is used to ensure that the browser or client application can correctly process and display the response.

8. What is Minification, Web Bundle, WebPack, and Lazy Loading of the client side, and what is their role in increasing performance across the network?

- **Minification:** Reduces file size by removing unnecessary characters (spaces, comments) from CSS, JS, and HTML.
- **Web Bundle:** Combines multiple files (JS or CSS) into a single bundle to reduce HTTP requests.
- **WebPack:** A module bundler that optimizes and bundles assets like JavaScript, CSS, and images.
- **Lazy Loading:** Delays loading of non-critical resources (like images or modules) until they are needed.

Role in performance: These techniques reduce page load time, decrease bandwidth usage, and improve user experience by making websites faster and more efficient.