Full Stack Developer Interview Questions and Answers

Dot Net Full Stack Developer

Interview Questions and Answers

Sai Reddy 7/14/25 Interview Tips

Dot Net Full Stack Developer Interview Q&A

Duration 30 min

1. What are the differences between .NET Core and .NET Framework? Why is .NET Core preferred in modern development?

.NET Core is **cross-platform**, **lightweight**, and optimized for **modern web/cloud** apps.

Feature	.NET Core	.NET Framework
Platform	Cross-platform (Windows, Linux, macOS)	Windows only
Performance	High performance, lightweight	Comparatively heavier
App Type Support	Console, Web, Microservices	Desktop, Web
Hosting	Kestrel + IIS or self-hosted	IIS only

2. What are the Action Filters in MVC ? Explain them?

In ASP.NET Core (and earlier ASP.NET MVC), **Action Filters** are a type of **filter** that allow you to run code **before or after** certain stages in the **execution pipeline** of an action method. They are particularly useful for **cross-cutting concerns** such as logging, caching, authorization, input validation, exception handling, and more.

Types of Filters in ASP.NET Core

ASP.NET Core defines several types of filters. These include:

Filter Type	Interface	When It Executes
Authorization	IAuthorizationFilter	Before everything; determines if the user is allowed
Resource	IResourceFilter	Before model binding and action selection
Action	IActionFilter	Before and after the action method executes
Exception	IExceptionFilter	On unhandled exceptions during pipeline execution
Result	IResultFilter	Before and after the result (e.g., View or JSON)

Focus: Action Filters

Action filters let you execute code:

- **Before** the action method runs (OnActionExecuting)
- **After** the action method runs (OnActionExecuted)

Interfaces:

- IActionFilter – Synchronous
- IAsyncActionFilter – Asynchronous

How to Create a Custom Action Filter

```
public class LogActionFilter : IActionFilter
{
    public void OnActionExecuting(ActionExecutingContext context)
    {
        // Code before action executes
        Console.WriteLine("Action Starting: " + context.ActionDescriptor.DisplayName);
    }

    public void OnActionExecuted(ActionExecutedContext context)
    {
        // Code after action executes
        Console.WriteLine("Action Finished: " + context.ActionDescriptor.DisplayName);
    }
}
```

Usage in Controller:

```
[ServiceFilter(typeof(LogActionFilter))]
public class MyController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

Importance & Benefits of Action Filters

Action filters are important for handling **cross-cutting concerns** without duplicating code in multiple action methods.

Registering Filters Globally

You can register filters globally in Startup.cs or Program.cs:

```
services.AddControllersWithViews(options =>
{
   options.Filters.Add<LogActionFilter>();
});
```

3. Explain the Middleware pipeline in ASP.NET Core. How do you register and use custom middleware?

Middleware components handle requests/responses in a pipeline using app.Use, app.Run, or app.Map.

Custom Middleware Example:

```
public class LoggingMiddleware
{
   private readonly RequestDelegate _next;
```

```
    public LoggingMiddleware(RequestDelegate next) => _next = next;

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

// Register in Program.cs or Startup.cs
app.UseMiddleware<LoggingMiddleware>();
```

4. What is Dependency Injection in .NET Core and how is it configured?

DI allows you to inject required services automatically.

Service Configuration:

```
services.AddScoped<IMyService, MyService>(); // in Program.cs
```

Using in Controller:

```
public class MyController : Controller
{
    private readonly IMyService _service;
    public MyController(IMyService service) => _service = service;
}
```

Use **Scoped** for per-request, **Singleton** for application-wide, **Transient** for each usage.

5. How do you create and consume Web APIs in .NET Core? Explain routing and versioning.

Use [ApiController], attribute routing, and versioning techniques.

API Example:

```
[ApiController]
[Route("api/v1/[controller]")]
public class ProductsController : ControllerBase
{
    [HttpGet("{id}")]
    public IActionResult GetProduct(int id) => Ok(new { Id = id, Name = "Laptop" });
}
```

Versioning via Header:

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

6. How do you secure a .NET Core API using JWT tokens?

Authenticate users and authorize API using JWT.

Startup.cs Configuration:

```
services.AddAuthentication("Bearer")
.AddJwtBearer("Bearer", options => {
   options.TokenValidationParameters = new TokenValidationParameters {
      ValidateIssuer = true,
      ValidateAudience = true,
      ValidateLifetime = true,
      ValidateIssuerSigningKey = true,
      ValidIssuer = "yourdomain.com",
      ValidAudience = "yourdomain.com",
      IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("secret-key"))
   };
});

app.UseAuthentication();
app.UseAuthorization();
```

Use in Controller:

```
[Authorize]
[HttpGet("secure-data")]
public IActionResult GetSecureData() => Ok("JWT protected data");
```

7. What is async and await in C#? Difference Between Asynchronous and Synchronous Code?

async and await are **keywords used to write asynchronous code** in a non-blocking way.

Asynchronous code **allows your application to continue executing while waiting for a time-consuming task**, like database access or an API call, without freezing the app.

Feature	Synchronous	Asynchronous
Execution	Tasks run one after another	Tasks can run in the background
Blocking	Blocks the thread until task completes	Doesn't block the thread
Performance	Slower with I/O operations	Faster, more scalable
Use Case	Small, quick tasks	Long-running, I/O-bound tasks (DB, HTTP)

Exg:

Synchronous Code (Blocks Thread)

```
public string GetData()
{
   var client = new HttpClient();
   var result = client.GetStringAsync("https://api.example.com/data").Result; // BLOCKING
   return result;
}
```

Asynchronous Code (Non-Blocking)

```
public async Task<string> GetDataAsync()
{
   var client = new HttpClient();
   var result = await client.GetStringAsync("https://api.example.com/data"); // NON-BLOCKING
```

```
return result;  }
```

8. Write a SQL query to find the second highest salary from an Employee table.

```
SELECT MAX(Salary) AS SecondHighestSalary
FROM Employees
WHERE Salary < (SELECT MAX(Salary) FROM Employees);
```

Alternate using ROW_NUMBER():

```
SELECT Salary
FROM (
   SELECT Salary, ROW_NUMBER() OVER (ORDER BY Salary DESC) AS rn
   FROM Employees
) AS result
WHERE rn = 2;
```

9. What are the differences between clustered and non-clustered indexes?

Feature	Clustered Index	Non-Clustered Index
Data Storage	Sorts and stores actual data rows	Stores pointers to actual rows
Number per Table	Only one (primary key by default)	Multiple
Performance	Faster for range queries	Faster for lookups on specific cols

Clustered is good for **range-based filtering**, non-clustered for **quick search**.

10. How do you optimize a slow-running SQL query?

Optimization Techniques:

- Create **indexes** on frequently filtered columns.
- Use EXISTS instead of IN.
- Avoid SELECT *; choose required columns.

```
SELECT Name FROM Employees
```

- Use SQL Profiler or **execution plans**.

]

11. Explain normalization and denormalization with examples.

Normalization:

- 1NF: Eliminate repeating groups (Atomic values).
- 2NF: No partial dependencies.
- 3NF: No transitive dependencies.

Example:

- Employee table split into:
    - Employees (EmpId, Name, DeptId)
    - Departments (DeptId, DeptName)

Denormalization is the opposite—adding redundant data for performance.

12. What is Dapper and how is it different from Entity Framework?

Dapper is a **micro ORM**:

- Uses raw SQL (high performance)
- No change tracking
- Lightweight

Example:

```
using (var con = new SqlConnection(connectionString))
{
   var result = con.Query<Employee>("SELECT * FROM Employees").ToList();
}
```

Use Dapper when you need control over SQL and better performance.

13. How do you perform parameterized queries using Dapper to prevent SQL Injection?

Example:

```
var sql = "SELECT * FROM Employees WHERE Id = @Id";
var employee = connection.QueryFirstOrDefault<Employee>(sql, new { Id = 1 });
```

Automatically prevents SQL injection using anonymous objects.

14. Explain the difference between Template-driven and Reactive forms in Angular.

Feature	Template-Driven	Reactive
Code Style	HTML driven	TypeScript driven
Validation	HTML based (required, etc.)	Programmatic (Validators.required)
Flexibility	Less	More

Reactive Example:

```
form = this.fb.group({
  name: ['', Validators.required],
  age: [null, Validators.min(18)]
});
```

15. How do you communicate between components in Angular (parent to child and vice versa)?

Parent to Child:

```
@Input() userName: string;
```

Child to Parent:

```
@Output() notify = new EventEmitter<string>();
this.notify.emit("Data from child");
```

Service for unrelated components:

```
@Injectable({ providedIn: 'root' })
export class DataService {
  subject = new BehaviorSubject<any>(null);
}
```

16. What are Observables and how are they used with HTTPClient in Angular?

Observables handle **async streams** like HTTP, events.

Example:

```
this.http.get<Product[]>('/api/products')
.pipe(
  map(data => data.filter(p => p.price > 100)),
  catchError(err => of([]))
)
.subscribe(result => this.products = result);
```

17. What are App Services and how do you deploy a .NET Core API to Azure App Service?

Azure App Service is a **PaaS** to host APIs/web apps.

Deployment Steps:

1. In Visual Studio, right-click project → Publish → Azure → App Service.
2. Choose or create Resource Group + App Service Plan.
3. Publish and access via Azure-provided URL.

Or

1. In Visual Studio, right-click project → Publish
2. Click on import profile – get deployment profile from you Azure team.
3. Click on publish.