**1. Describe the architecture of your last project. What technologies and tools did you use?**

**Answer:** My last project followed a layered architecture including Presentation Layer (Angular), Business Logic Layer (.NET Core Web API), and Data Access Layer (EF Core). We hosted services on Azure App Services, used Azure SQL Database, and integrated Azure Blob Storage for files. DevOps pipelines were implemented using Azure DevOps for CI/CD.

**2. How do you configure a DbContext in EF Core?**

**Answer:** We create a class inheriting from DbContext, define DbSet<TEntity> properties, and override OnModelCreating() for custom configurations. It is registered in Startup.cs using services.AddDbContext<>() method with connection string from appsettings.json.

**3. What are the key advantages of .NET Core?**

**Answer:**

* Cross-platform support
* Improved performance
* Lightweight and modular
* Built-in dependency injection
* Side-by-side versioning

**4. What is middleware in .NET Core?**

**Answer:** Middleware is software assembled into an application pipeline to handle requests and responses. Examples include authentication, logging, CORS, and exception handling.

**5. How do you create a custom middleware in ASP.NET Core?**

**Answer:**

1. Create a class with a constructor accepting RequestDelegate and an Invoke method.
2. Write your logic in Invoke.
3. Register the middleware in Startup.cs using app.UseMiddleware<>().

**6. What is the purpose of MVC Filters?**

**Answer:** Filters are used to run code before or after specific stages in the request processing pipeline. Examples include AuthorizationFilter, ActionFilter, ResultFilter, and ExceptionFilter.

**7. Difference between Filters and Middleware?**

**Answer:**

* Middleware is global and runs for all requests; filters are scoped to controllers/actions.
* Middleware has access to both request and response early; filters operate closer to MVC execution.

**8. What are SOLID Principles?**

**Answer:**

* S: Single Responsibility Principle
* O: Open/Closed Principle
* L: Liskov Substitution Principle
* I: Interface Segregation Principle
* D: Dependency Inversion Principle

**9. SQL: What are cursors?**

**Answer:** Cursors allow row-by-row processing of query results. They are slower than set-based operations and are used in scenarios requiring iterative logic.

**10. SQL: What are triggers?**

**Answer:** Triggers are special stored procedures executed automatically in response to events like INSERT, UPDATE, or DELETE on a table.

**11. SQL: How do you optimize stored procedures?**

**Answer:**

* Use proper indexes
* Avoid SELECT \*
* Use EXISTS instead of IN
* Avoid unnecessary cursors
* Use SET NOCOUNT ON

**12. SQL: Select returns too many rows, causing latency. How do you resolve?**

**Answer:**

* Apply proper filtering (WHERE clause)
* Use pagination
* Optimize indexes
* Review execution plan for bottlenecks

**13. What are Microservices? How are they different from Monolithic apps?**

**Answer:** Microservices are independent services communicating over HTTP. Monolithic apps are single units. Microservices allow independent scaling, deployment, and fault isolation.

**14. What triggers have you used in Azure Functions?**

**Answer:**

* HTTP Trigger
* Timer Trigger
* Queue Trigger
* Blob Trigger
* Service Bus Trigger

**15. Advantages of Azure Functions over Web API?**

**Answer:**

* Serverless (no infrastructure management)
* Pay-per-execution pricing
* Built-in integration with Azure services
* Auto-scaling

**16. How do you check logs in Azure Portal?**

**Answer:**

* Application Insights
* Azure Monitor
* Log Analytics Query in Log Workspace

**17. What authentication methods have you used?**

**Answer:**

* JWT (Bearer Token)
* OAuth 2.0
* Azure AD Authentication (MSAL)

**18. Code Challenge: FizzBuzz**

**Answer:**

for (int i = 1; i <= n; i++)

{

if (i % 15 == 0) Console.WriteLine("FizzBuzz");

else if (i % 3 == 0) Console.WriteLine("Fizz");

else if (i % 5 == 0) Console.WriteLine("Buzz");

else Console.WriteLine(i);

}