নিচে Lead Software Engineer পদের জন্য ASP.NET Core ভিত্তিক কিছু গুরুত্বপূর্ণ ইন্টারভিউ প্রশ্ন এবং তাদের বাংলা উত্তর দেওয়া হলো:

**✅ প্রশ্ন ১: ASP.NET Core কী? এটি ASP.NET থেকে কীভাবে আলাদা?**

**উত্তর:**  
ASP.NET Core হলো একটি ওপেন সোর্স, ক্রস-প্ল্যাটফর্ম ফ্রেমওয়ার্ক যা মডার্ন ওয়েব অ্যাপ্লিকেশন তৈরি করার জন্য ব্যবহৃত হয়। এটি ASP.NET এর নতুন সংস্করণ, যা .NET Core এর উপর ভিত্তি করে তৈরি।

**ASP.NET থেকে পার্থক্য:**

* ASP.NET শুধুমাত্র Windows-এ চলে, কিন্তু ASP.NET Core চলে Windows, Linux, ও macOS-এ।
* ASP.NET Core হালকা, দ্রুত এবং মডিউলার আর্কিটেকচারে তৈরি।
* Dependency Injection (DI) Built-in।
* CLI (Command Line Interface) সাপোর্ট করে।

**✅ প্রশ্ন ২: Middleware কী? ASP.NET Core এ এটি কীভাবে কাজ করে?**

**উত্তর:**  
Middleware হলো এমন এক ধরনের কম্পোনেন্ট যা HTTP রিকোয়েস্ট ও রেসপন্স পাইপলাইনে কাজ করে। প্রতিটি middleware রিকোয়েস্ট প্রসেস করে এবং পরবর্তী middleware-এ পাঠায়।

**উদাহরণ:**

app.UseMiddleware<CustomMiddleware>();

ASP.NET Core এ Startup.cs ফাইলে Configure মেথডের মধ্যে middleware গুলো চেইন আকারে ডিফাইন করা হয়।

**✅ প্রশ্ন ৩: Dependency Injection কী এবং ASP.NET Core এ এটি কীভাবে কাজ করে?**

**উত্তর:**  
Dependency Injection (DI) হলো একটি ডিজাইন প্যাটার্ন, যা ক্লাসের নির্ভরতাগুলো (dependencies) বাহ্যিকভাবে ইনজেক্ট করে।

ASP.NET Core এ DI built-in। আপনি Startup.cs-এর ConfigureServices মেথডে সার্ভিস রেজিস্টার করেন:

services.AddScoped<IMyService, MyService>();

তারপর Constructor এর মাধ্যমে ইনজেক্ট করে ব্যবহার করেন।

**✅ প্রশ্ন ৪: IActionResult এবং ActionResult এর মধ্যে পার্থক্য কী?**

**উত্তর:**

* IActionResult একটি ইন্টারফেস, যা বিভিন্ন ধরণের রেসপন্স রিটার্ন করতে দেয়।
* ActionResult একটি ক্লাস যা IActionResult কে ইমপ্লিমেন্ট করে এবং এটিতে টাইপ সেফটি থাকে।

**উদাহরণ:**

public IActionResult GetData() => Ok("Data");

public ActionResult<string> GetData() => "Data";

**✅ প্রশ্ন ৫: আপনি কীভাবে Custom Exception Handling করবেন ASP.NET Core এ?**

**উত্তর:**  
Custom Exception Handling করার জন্য middleware তৈরি করা হয় বা UseExceptionHandler ব্যবহার করা হয়।

**উদাহরণ:**

app.UseExceptionHandler("/Home/Error");

অথবা Custom Middleware:

public class ExceptionMiddleware { ... }

**✅ প্রশ্ন ৬: আপনি Clean Architecture কীভাবে বাস্তবায়ন করেন ASP.NET Core এ?**

**উত্তর:**  
Clean Architecture ৪টি লেয়ারে ভাগ করা হয়:

1. **Domain Layer** – Business logic এবং Entities
2. **Application Layer** – Use cases
3. **Infrastructure Layer** – External services (DB, API)
4. **Presentation Layer** – UI বা API controllers

**ASP.NET Core এ:**

* Project structure আলাদা থাকে।
* DI এর মাধ্যমে Infrastructure ক্লাসগুলো inject করা হয়।
* Domain layer এ কোনো ফ্রেমওয়ার্কের রেফারেন্স থাকে না।

**✅ প্রশ্ন ৭: আপনি কীভাবে Authentication ও Authorization ইমপ্লিমেন্ট করেন?**

**উত্তর:**

* Authentication: User কে যাচাই করে।
* Authorization: কে কী করতে পারবে সেটা নিয়ন্ত্রণ করে।

**ASP.NET Core এ:**

services.AddAuthentication(...);

services.AddAuthorization(...);

Controller এ:

[Authorize(Roles = "Admin")]

public IActionResult AdminAction() { ... }

JWT, Cookie বা OAuth ব্যবহার করে Authentication করা যায়।

**✅ প্রশ্ন ৮: Entity Framework Core এ Migration কীভাবে কাজ করে?**

**উত্তর:**  
Migration ব্যবহার করে আপনি আপনার ডাটাবেসের স্কিমা কন্ট্রোল করতে পারেন কোডের মাধ্যমে।

**ধাপগুলো:**

1. Add-Migration InitialCreate
2. Update-Database

EF Core আপনার Model থেকে ডাটাবেসে টেবিল ও কলাম তৈরি করে।

**✅ প্রশ্ন ৯: Performance optimization করার জন্য আপনি কী কী পদ্ধতি নেন?**

**উত্তর:**

* Caching (Memory/Distributed)
* Asynchronous Programming (async/await)
* Reduce DB calls
* Pagination ব্যবহার করা
* Compiled Queries in EF Core
* Response Compression Middleware

**✅ প্রশ্ন ১০: CQRS ও Mediator Pattern কীভাবে ব্যবহার করেন?**

**উত্তর:**  
CQRS (Command Query Responsibility Segregation) দ্বারা আপনি Query ও Command আলাদা করেন।

Mediator Pattern ব্যবহারে Mediator লাইব্রেরি (যেমন MediatR) ব্যবহার করে আপনি Request/Response প্যাটার্নে কাজ করেন।

**CQRS উদাহরণ:**

* Command: CreateOrderCommand
* Query: GetOrdersQuery

অবশ্যই, নিচে ASP.NET Core ভিত্তিক Lead Software Engineer পদের জন্য আরও বিস্তারিতভাবে বাংলা ভাষায় ইন্টারভিউ প্রশ্ন এবং উত্তর ব্যাখ্যা করা হলো:

**✅ প্রশ্ন ১: ASP.NET Core কী এবং এটি ASP.NET থেকে কীভাবে আলাদা?**

**🔍 বিস্তারিত ব্যাখ্যা:**

ASP.NET Core হলো Microsoft-এর তৈরি একটি ওপেন সোর্স, ক্রস-প্ল্যাটফর্ম ফ্রেমওয়ার্ক যা দিয়ে আপনি ওয়েব অ্যাপ্লিকেশন, API, ও ক্লাউড-ভিত্তিক সার্ভিস তৈরি করতে পারেন।

**মূল পার্থক্য ASP.NET ও ASP.NET Core এর মধ্যে:**

| **ASP.NET** | **ASP.NET Core** |
| --- | --- |
| শুধুমাত্র Windows-এ চলে | Windows, Linux, Mac সব জায়গায় চলে |
| IIS নির্ভর | Kestrel নামে লাইটওয়েট সার্ভার আছে |
| Web Forms, MVC, Web API আলাদা | সব একত্রে (unified programming model) |
| Dependency Injection নেই | Built-in Dependency Injection সাপোর্ট |

👉 ASP.NET Core মূলত performance, modularity, ও scalability এর জন্য ডিজাইন করা হয়েছে।

**✅ প্রশ্ন ২: Middleware কী এবং ASP.NET Core এ এটি কিভাবে কাজ করে?**

**🔍 বিস্তারিত ব্যাখ্যা:**

Middleware হচ্ছে এমন কিছু ক্লাস যা HTTP রিকোয়েস্ট আসা থেকে শুরু করে রেসপন্স পাঠানো পর্যন্ত প্রত্যেক ধাপে কাজ করে।

প্রত্যেকটি middleware:

* রিকোয়েস্ট প্রসেস করে
* পরবর্তী middleware কে কল করে
* চাইলে রিকোয়েস্ট আটকে রাখতে পারে

**Middleware তৈরি করার উদাহরণ:**

public class LoggingMiddleware

{

private readonly RequestDelegate \_next;

public LoggingMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task Invoke(HttpContext context)

{

Console.WriteLine("Request আসছে: " + context.Request.Path);

await \_next(context); // পরবর্তী middleware কল করা

Console.WriteLine("Response যাচ্ছে");

}

}

**Register করতে হবে Startup.cs এ:**

app.UseMiddleware<LoggingMiddleware>();

**✅ প্রশ্ন ৩: Dependency Injection (DI) কী এবং কিভাবে ব্যবহার হয়?**

**🔍 বিস্তারিত ব্যাখ্যা:**

Dependency Injection এমন একটি প্যাটার্ন যা কোনো ক্লাসের উপর নির্ভরশীল অবজেক্টগুলো বাইরে থেকে সরবরাহ করে দেয়।

ASP.NET Core এ DI built-in রয়েছে এবং আপনি সাধারণত ৩ ধরণের লাইফটাইম ব্যবহার করেন:

* Transient – প্রতি বার নতুন অবজেক্ট তৈরি হয়
* Scoped – প্রতি HTTP রিকোয়েস্টে একটি অবজেক্ট তৈরি হয়
* Singleton – পুরো অ্যাপ্লিকেশন লাইফটাইমে একটি অবজেক্ট তৈরি হয়

**উদাহরণ:**

services.AddScoped<ILogService, LogService>();

**ব্যবহার:**

public class HomeController : Controller

{

private readonly ILogService \_logger;

public HomeController(ILogService logger)

{

\_logger = logger;

}

}

**✅ প্রশ্ন ৪: IActionResult vs ActionResult – পার্থক্য কী?**

**🔍 বিস্তারিত ব্যাখ্যা:**

| **IActionResult** | **ActionResult** |
| --- | --- |
| শুধুমাত্র HTTP রেসপন্স টাইপ | HTTP রেসপন্স + টাইপ সেফ ডাটা |
| Flexible কিন্তু টাইপ চেকিং নেই | Strongly typed রিটার্ন ভ্যালু |

**IActionResult উদাহরণ:**

public IActionResult Get()

{

return Ok("Hello");

}

**ActionResult উদাহরণ:**

public ActionResult<string> Get()

{

return "Hello";

}

**✅ প্রশ্ন ৫: Custom Exception Handling কিভাবে করবেন?**

**🔍 বিস্তারিত ব্যাখ্যা:**

অ্যাপ্লিকেশনে এক্সেপশন হলে ব্যবহারকারীকে সুন্দরভাবে জানানো এবং লগ করে রাখা দরকার।

**Built-in Middleware ব্যবহার:**

app.UseExceptionHandler("/Home/Error");

**Custom Middleware উদাহরণ:**

public class GlobalExceptionMiddleware

{

private readonly RequestDelegate \_next;

public async Task Invoke(HttpContext context)

{

try

{

await \_next(context);

}

catch (Exception ex)

{

// Error handle, log and custom response

context.Response.StatusCode = 500;

await context.Response.WriteAsync("Internal Server Error");

}

}

}

**✅ প্রশ্ন ৬: Clean Architecture কীভাবে প্রয়োগ করবেন?**

**🔍 বিস্তারিত ব্যাখ্যা:**

Clean Architecture এর মূল নীতি হলো: **Dependency rule** — যত গুরুত্বপূর্ণ, তত ভিতরের লেয়ারে থাকবে।

**লেয়ারের বিবরণ:**

1. **Domain Layer:** Entity, Value Object, Business Logic (কোনো external dependency নয়)
2. **Application Layer:** Use Cases বা Services
3. **Infrastructure Layer:** Database, File System, API etc.
4. **Web/API Layer:** ASP.NET Controller, UI ইত্যাদি

**নির্দেশনা:**

* কোনো লেয়ার অন্য লেয়ারের উপরে নির্ভরশীল নয়
* Dependency inversion ব্যবহার করে নিচের লেয়ার উপরের ইন্টারফেসকে implement করে

**✅ প্রশ্ন ৭: Authentication ও Authorization কীভাবে বাস্তবায়ন করবেন?**

**🔍 বিস্তারিত ব্যাখ্যা:**

**Authentication** – ইউজার কে তা যাচাই করা  
**Authorization** – ইউজার কী করতে পারবে তা নির্ধারণ করা

**JWT Authentication উদাহরণ:**

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options => {

options.TokenValidationParameters = new TokenValidationParameters {

ValidateIssuer = true,

ValidateAudience = true,

...

};

});

**Controller:**

[Authorize(Roles = "Admin")]

public IActionResult AdminOnly() { ... }

**✅ প্রশ্ন ৮: Entity Framework Core এ Migration কীভাবে কাজ করে?**

**🔍 বিস্তারিত ব্যাখ্যা:**

**Migration** হলো Database-এর পরিবর্তনগুলো version control এর মাধ্যমে পরিচালনা করার পদ্ধতি।

**প্রসেস:**

1. Add-Migration InitialCreate
2. Update-Database

**Model ক্লাস:**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

}

এই পরিবর্তনকে ডাটাবেজে পাঠানো হয় Migration এর মাধ্যমে।

**✅ প্রশ্ন ৯: ASP.NET Core অ্যাপ্লিকেশনের পারফরম্যান্স উন্নত করতে কী করবেন?**

**🔍 বিস্তারিত ব্যাখ্যা:**

* **Caching ব্যবহার করবেন**: MemoryCache, Redis
* **Asynchronous Programming**: async/await দিয়ে IO-bound কাজ দ্রুত হবে
* **Database Optimization**: Lazy loading এড়িয়ে, include দিয়ে eager loading
* **Pagination**: পুরো ডাটা না এনে পেজ করে আনা
* **Minimize HTTP calls**
* **Response Compression Middleware**

services.AddResponseCompression();

app.UseResponseCompression();

**✅ প্রশ্ন ১০: CQRS ও Mediator Pattern কী এবং আপনি কিভাবে ব্যবহার করবেন?**

**🔍 বিস্তারিত ব্যাখ্যা:**

CQRS মানে Command ও Query আলাদা করা।

* **Command:** ডেটা পরিবর্তন করে (write)
* **Query:** ডেটা পড়ে (read)

**Mediator Pattern (MediatR লাইব্রেরি দিয়ে):**

* সব Command বা Query এর জন্য handler থাকে
* Controller সরাসরি সার্ভিস কল না করে Mediator-কে বলে

**CQRS উদাহরণ:**

public class CreateProductCommand : IRequest<bool>

{

public string Name { get; set; }

}

public class CreateProductHandler : IRequestHandler<CreateProductCommand, bool>

{

public async Task<bool> Handle(CreateProductCommand request, CancellationToken ct)

{

// Database logic here

return true;

}

}