একজন **Lead Software Engineer (.NET)** এর ইন্টারভিউয়ের জন্য গুরুত্বপূর্ণ কিছু প্রশ্ন এবং সেগুলোর বাংলা উত্তর নিচে দেওয়া হলো। এগুলো টেকনিক্যাল, আর্কিটেকচারাল, লিডারশিপ এবং বাস্তব অভিজ্ঞতার ভিত্তিতে সাজানো।

**🔹 ১. আপনি .NET Framework ও .NET Core এর পার্থক্য ব্যাখ্যা করুন।**

**উত্তর:**  
.NET Framework মূলত Windows-ভিত্তিক অ্যাপ্লিকেশনের জন্য। এটি Windows OS-এ কাজ করে এবং বেশিরভাগ ডেস্কটপ ও WebForms অ্যাপ তৈরি করতে ব্যবহৃত হয়।  
.NET Core হলো cross-platform, open-source এবং lightweight। এটি Windows, Linux, ও macOS - তিনটি প্ল্যাটফর্মেই চলে।  
বর্তমানে .NET 5/6/7/8 থেকে শুরু করে এটি "One .NET" হিসেবে পরিচিত — মানে একটি ইউনিফায়েড প্ল্যাটফর্ম।

**🔹 ২. SOLID principles কী এবং আপনি কীভাবে এগুলো প্রয়োগ করেন আপনার প্রজেক্টে?**

**উত্তর:**  
SOLID মানে ৫টি ডিজাইন প্রিন্সিপল:

1. **S - Single Responsibility Principle:** একেক ক্লাসের একেক দায়িত্ব।
2. **O - Open/Closed Principle:** কোড এক্সটেনশনের জন্য ওপেন, কিন্তু মোডিফিকেশনের জন্য ক্লোজড।
3. **L - Liskov Substitution Principle:** ডেরাইভড ক্লাস প্যারেন্ট ক্লাস হিসেবে ব্যবহারযোগ্য হওয়া উচিত।
4. **I - Interface Segregation Principle:** ছোট ছোট ইন্টারফেস তৈরি করা উচিত।
5. **D - Dependency Inversion Principle:** উচ্চ লেভেলের ক্লাস যেন নিম্ন লেভেলের ক্লাসের উপর নির্ভর না করে — ইন্টারফেস ব্যবহার করে ডিপেন্ডেন্সি ইনভার্ট করা উচিত।

আমার প্রজেক্টে আমি DI container ব্যবহার করি (যেমন Microsoft.Extensions.DependencyInjection), ক্লাস গুলোর দায়িত্ব ছোট রাখি এবং Interface-based coding করি।

**🔹 ৩. Clean Architecture এবং তার মূল স্তরগুলো কী?**

**উত্তর:**  
**Clean Architecture** অ্যাপ্লিকেশনকে বিভিন্ন স্তরে ভাগ করে রাখে, যেমন:

1. **Entities (Core Business Rules)**
2. **Use Cases (Application Logic)**
3. **Interface Adapters (Controllers, Presenters, Gateways)**
4. **Frameworks & Drivers (UI, DB, External Services)**

এর মূল উদ্দেশ্য হলো:

* Dependency গুলো ভেতরের দিকে যাবে
* Core Logic কখনো DB বা UI-র উপর নির্ভর করবে না
* টেস্টযোগ্য, স্কেলযোগ্য এবং মডিউলার অ্যাপ তৈরি করা

**🔹 ৪. Entity Framework Core এ Lazy Loading এবং Eager Loading-এর পার্থক্য কী?**

**উত্তর:**

* **Lazy Loading:** যখন আপনি নেভিগেশন প্রপার্টি অ্যাক্সেস করেন তখনি ডাটা ফেচ হয়।
* **Eager Loading:** .Include() ব্যবহার করে আগেই প্রয়োজনীয় রিলেটেড ডাটা লোড করে নেয়।

**Lazy Loading** পারফরম্যান্সের জন্য খারাপ হতে পারে যদি না বুঝে ব্যবহার করা হয়। বড় প্রজেক্টে আমি সাধারণত **Eager Loading** ব্যবহার করি যাতে ডাটা ফেচ কনট্রোল করা যায়।

**🔹 ৫. আপনি কীভাবে টিম ম্যানেজ করেন? কিভাবে কোড রিভিউ পরিচালনা করেন?**

**উত্তর:**  
আমি প্রতিটি ডেভেলপারের জন্য স্পষ্ট টাস্ক ডিফাইন করি (Jira বা Azure DevOps-এ)।

* আমরা Code Review টুল (যেমন GitHub PR, GitLab MR) ব্যবহার করি
* Code standards (naming, structure, SOLID) মেনে চলি
* PR approve করার আগে আমি লজিক, পারফরম্যান্স, এবং সিকিউরিটি দেখছি
* পাশাপাশি junior dev দের কোড বুঝিয়ে দেওয়া, mentorship করা — এটাও আমার দায়িত্ব

**🔹 ৬. Microservices আর Monolith এর মধ্যে পার্থক্য কী? এবং আপনি কখন Microservices ব্যবহার করবেন?**

**উত্তর:**

* **Monolith:** পুরো অ্যাপ একটি কোডবেজে — সব কম্পোনেন্ট tightly coupled
* **Microservices:** ছোট ছোট সার্ভিস, একেকটা সার্ভিস আলাদা responsibility নিয়ে কাজ করে, loosely coupled

**Microservices ব্যবহার করব যখন:**

* Team বড়
* ডিপ্লয়মেন্ট আলাদা আলাদা দরকার
* Scalable architecture চাই (কিছু পার্ট বেশি লোড নিবে)
* Tech stack ভিন্ন হতে পারে (একটা সার্ভিস .NET, অন্যটা Node.js)

**🔹 ৭. কোনো প্রজেক্টে আপনি কীভাবে পারফরম্যান্স বটলনেক চিহ্নিত ও সমাধান করেছেন?**

**উত্তর:**  
একবার একটা প্রজেক্টে রিপোর্ট পেজ লোড হতে অনেক সময় নিচ্ছিল।

* আমি **SQL Profiler** দিয়ে কোয়েরি ট্র্যাক করি
* দেখতে পাই কিছু N+1 কোয়েরি সমস্যা ছিল
* .Include() ব্যবহার করে optimize করি
* Indexing ও paging যুক্ত করি
* এরপর পারফরম্যান্স 70% বেড়ে যায়

**🔹 ৮. CI/CD কীভাবে ইমপ্লিমেন্ট করেন?**

**উত্তর:**  
আমি Azure DevOps বা GitHub Actions ব্যবহার করি।

**স্টেপগুলো:**

1. Build →
2. Unit Test →
3. Code Analysis (SonarQube / linting) →
4. Deploy to Test →
5. Manual/Auto approval →
6. Deploy to Production

আমি YAML pipeline ব্যবহার করে automation করি।

**🔹 ৯. আপনি junior developer-দের mentorship কীভাবে করেন?**

**উত্তর:**

* শুরুতে তাদের বুঝিয়ে দেই design pattern, coding best practices
* Code review এর মাধ্যমে ফিডব্যাক দেই
* Pair programming করি
* ছোট ছোট task দিয়ে grow করতে সাহায্য করি
* Motivation ও communication শিখাই

**🔹 ১০. আপনার সবচেয়ে বড় চ্যালেঞ্জ কী ছিল টিম লিড হিসেবে?**

**উত্তর:**  
একবার আমাদের টিমে ডেডলাইন মিস হওয়ার ঝুঁকি ছিল কারণ রিকোয়ারমেন্ট সময়মতো আসেনি।

* আমি ক্লায়েন্টের সাথে সোজা কথা বলি
* পুরনো ফিচার refactor করে নতুন ফিচার accommodate করি
* কিছু টাস্ক pair করে parallel করি
* শেষে সময়মতো ডেলিভারি হয় এবং ক্লায়েন্ট সন্তুষ্ট ছিল

অবশ্যই! নিচে আরও কিছু **Lead Software Engineer (C#)** ইন্টারভিউ প্রশ্ন ও উত্তর দেওয়া হলো — **সম্পূর্ণ বাংলায়**, যেন আপনি সহজেই বুঝতে ও প্রস্তুতি নিতে পারেন।

**🔶 ১১. var, dynamic, আর object এর মধ্যে পার্থক্য কী?**

**উত্তর:**

| **কী** | **var** | **dynamic** | **object** |
| --- | --- | --- | --- |
| টাইপ নির্ধারণ | কম্পাইল টাইমে | রানটাইমে | কম্পাইল টাইমে |
| পারফরম্যান্স | বেশি | কম | মাঝারি |
| ব্যবহার | সাধারণ টাইপিং হেল্প | জটিল টাইপ বা COM object | যেকোনো টাইপ ধরে রাখতে |

**উদাহরণ:**

var name = "Rahim"; // string টাইপ

dynamic x = 5; x = "Hello"; // টাইপ রানটাইমে ঠিক হবে

object y = "World"; // casting করতে হবে

**🔶 ১২. C# এ try-catch-finally কীভাবে কাজ করে?**

**উত্তর:**

try-catch-finally ব্লক ব্যবহার করে আপনি error (exception) ধরতে ও handle করতে পারেন।

**উদাহরণ:**

try {

int a = 5;

int b = 0;

int result = a / b;

}

catch (DivideByZeroException ex) {

Console.WriteLine("ভাগ করা যাবে না শূন্য দিয়ে!");

}

finally {

Console.WriteLine("এই কোড সবসময়ই চলবে।");

}

**🔶 ১৩. C# এ abstract class আর interface এর পার্থক্য কী?**

**উত্তর:**

| **বিষয়** | **abstract class** | **interface** |
| --- | --- | --- |
| কন্টেন্ট | ফিল্ড, প্রপার্টি, কন্সট্রাক্টর থাকতে পারে | শুধু declaration |
| Multiple inheritance | নয় | হ্যাঁ |
| Access modifier | থাকে | থাকে না (C# 8+ এ কিছু support এসেছে) |

আপনি যখন common behavior চান এবং কিছু implement করতে চান, তখন abstract class।  
শুধু চুক্তি/চেহারা নির্ধারণ করতে চাইলে interface।

**🔶 ১৪. C# এর lock কী এবং এটি কেন ব্যবহার করা হয়?**

**উত্তর:**

lock একটি **thread-safety মেকানিজম**, যা এক সময় একটিমাত্র থ্রেডকে নির্দিষ্ট অংশ এক্সেস করতে দেয়।

**উদাহরণ:**

private readonly object \_lockObj = new object();

public void UpdateBalance() {

lock(\_lockObj) {

// এখানে একসাথে একটাই থ্রেড ঢুকতে পারবে

balance += 100;

}

}

এটি ডেটা কনসিস্টেন্সি রক্ষা করতে ব্যবহৃত হয় multithreading পরিস্থিতিতে।

**🔶 ১৫. StringBuilder আর string এর মধ্যে পার্থক্য কী?**

**উত্তর:**

* string immutable → প্রতিবার পরিবর্তনে নতুন instance তৈরি হয়
* StringBuilder mutable → একই object-এ পরিবর্তন হয়

যখন আপনি অনেক string কনক্যাট করছেন (যেমন: লুপে), তখন StringBuilder ব্যবহার করলে পারফরম্যান্স অনেক ভালো হয়।

**🔶 ১৬. C# এ LINQ এর Select, Where, GroupBy, OrderBy এর ব্যবহার দেখান।**

**উত্তর:**

ধরি List<Employee> আছে।

var result = employees

.Where(e => e.Salary > 20000) // filter

.OrderBy(e => e.Name) // sort

.Select(e => e.Name); // শুধু নাম

var grouped = employees

.GroupBy(e => e.Department)

.Select(g => new {

Department = g.Key,

Count = g.Count()

});

LINQ কোডকে SQL এর মতো করে লিখতে সাহায্য করে।

**🔶 ১৭. ref আর out কী এবং কবে ব্যবহার করবেন?**

**উত্তর:**

* ref: আগেই ভ্যালু থাকতে হবে
* out: মেথডের ভিতরে অবশ্যই মান দিতে হবে

**উদাহরণ:**

void Add(ref int x) {

x += 5;

}

void GetData(out int x) {

x = 10;

}

ref ও out মূলত **call by reference** এর জন্য ব্যবহৃত হয়।

**🔶 ১৮. C# এ sealed কী করে?**

**উত্তর:**

sealed ক্লাস বা মেথডকে further inheritance থেকে রক্ষা করে।

sealed class FinalClass {}

class A {

public virtual void Display() {}

}

class B : A {

public sealed override void Display() {}

}

sealed ব্যবহার করলে পারফরম্যান্সও বাড়তে পারে কারণ runtime inheritance check কম হয়।

**🔶 ১৯. C# এ Extension Method কী?**

**উত্তর:**

Extension Method দিয়ে আপনি বিদ্যমান class এর উপর নতুন method যোগ করতে পারেন, মূল class না পরিবর্তন করে।

**উদাহরণ:**

public static class StringExtensions {

public static bool IsEmail(this string input) {

return input.Contains("@");

}

}

// ব্যবহার:

string email = "abc@test.com";

bool result = email.IsEmail();

**🔶 ২০. একটি সিনিয়র লিড হিসেবে আপনি কিভাবে কোড স্ট্যান্ডার্ড ও ক্লিন কোড নিশ্চিত করেন?**

**উত্তর:**

* টিমে কোড গাইডলাইন তৈরি করি (naming, formatting, commenting)
* প্রতিটি Pull Request/Code Review বাধ্যতামূলক
* StyleCop, SonarQube, FxCop ব্যবহার করি
* ক্লিন কোড বই (Robert C. Martin) অনুসরণ করি
* Refactoring নিয়মিত করি ও logic-based review করি