Manish Jaiswal

Aync Programing Series

APPLYING

Async and Await in Real-World Projects

swipe

Manish Jaiswal

Aync Programing Series

01

Async & Await: A Game Changer

Async and await have revolutionized asynchronous programming in C#. They make code more **readable**, **maintainable**, and easier to reason about.

But how do you apply them effectively in real-world projects? Let's dive in!

swipe

Manish Jaiswal

Aync Programing Series

02

Why Async & Await?

async and await are essential for non-blocking operations in .NET 8. They allow your applications to perform tasks like I/O operations, network calls, and database queries without freezing the UI or blocking threads.

```
1  public async Task<string> GetDataAsync()
2  {
3      HttpClient client = new HttpClient();
4      string data = await client
5                    .GetStringAsync("https://api.example.com/data");
6      return data;
7  }
8
```

swipe

Manish Jaiswal

Aync Programing Series

03

Practical Example: Async File Reading

Read large files without blocking the **main thread**. Ideal for applications needing to handle extensive data processing.

```
1  public async Task<string> ReadFileAsync(string filePath)
2  {
3      using StreamReader reader = new StreamReader(filePath);
4      string content = await reader.ReadToEndAsync();
5      return content;
6  }
```

swipe

Manish Jaiswal

Aync Programing Series

04

Best Practices: Avoiding Deadlocks

Deadlocks can occur if async methods are not used correctly. Avoid calling **.Result** or **.Wait()** on tasks, and always use await for asynchronous operations.

```
// Avoid this
var result = GetDataAsync().Result;

// Correct approach
var result = await GetDataAsync();
```

swipe

Manish Jaiswal

Aync Programing Series

05

Exception Handling

Handle exceptions in async methods using **try-catch**. Unhandled exceptions can cause the entire application to crash.

```
1  public async Task<string> SafeGetDataAsync()
2  {
3      try
4      {
5          HttpClient client = new HttpClient();
6          return await client.GetStringAsync("https://api.example.com/data");
7      }
8      catch (HttpRequestException ex)
9      {
10         // Handle the exception
11         return $"Error: {ex.Message}";
12     }
13 }
```

swipe

Manish Jaiswal

Aync Programing Series

06

Optimize for Performance

Use **ConfigureAwait(false)** to reduce unnecessary context switching, especially in library code where UI context is not required.

```
1  public async Task<string> FetchDataAsync()
2  {
3      HttpClient client = new HttpClient();
4      return await client
5                  .GetStringAsync("https://api.example.com/data")
6                  .ConfigureAwait(false);
7  }
```

swipe

Manish Jaiswal

Aync Programing Series

07

Task.WhenAll for Parallelism

Leverage **Task.WhenAll** to run multiple tasks in parallel, improving the overall performance of your application.

```
1  public async Task ProcessMultipleTasksAsync()
2  {
3      var task1 = GetDataAsync("https://api.example.com/data1");
4      var task2 = GetDataAsync("https://api.example.com/data2");
5
6      await Task.WhenAll(task1, task2);
7
8      var result1 = await task1;
9      var result2 = await task2;
10 }
```

swipe

Manish Jaiswal

Aync Programing Series

Interview Questions

1. What is the purpose of the async and await keywords in C#?
2. How does the async keyword work in C#?
3. What is the difference between synchronous and asynchronous programming?
4. What is the role of Task in asynchronous programming?
5. Explain the concept of "context switching" in the context of async programming. How does ConfigureAwait(false) help?
6. How does the Task.WhenAll method work, and when would you use it?
7. What are the performance considerations when using async and await?
8. How do you prevent deadlocks when using async and await?

swipe