

# .NET MICROSERVICES

4. forduló



A kategória támogatója: DXC Technology

Ismertető a feladatlaphoz

Közeleg az 5. forduló, figyelj az időpontokra!

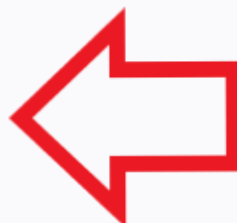
Használd a naptárat:

## KATEGÓRIÁIM

Összesen 10 kategóriára jelentkezted



**Versenynaptár letöltése**



Vagy figyeld kategóriánként az időpontokat (íme egy MINTA, hol találod):



## ● 3. FORDULÓ

A lezárt fordulókban eddig megszerzett pontok:

0/100 pont

**BOSCH**  
Invented for life

## Fordulók

Forduló	Pontok, időtartam	Feladat megoldható	Státusz
7. forduló	23 pont 25:00	 2023.11.28. 20:00-tól 2023.11.28. 20:35-ig	Feladatlap
6. forduló	23 pont 30:00	 2023.11.21. 20:00-tól 2023.11.21. 20:40-ig	Feladatlap
5. forduló	28 pont 25:00	 2023.11.14. 20:00-tól 2023.11.14. 20:35-ig	Feladatlap

Amennyiben olyan kategóriában játszol, ahol van csatolmány, de hibába ütközel a letöltésnél, ott valószínűleg a vírusirtó korlátoz, annak ideiglenes kikapcsolása megoldhatja a problémát. (Körülbelül minden 3000. letöltésnél fordul ez elő.)

Jó versenyzést kívánunk!

## 1. feladat 5 pont

Microservice architektúra esetén jellemzően sok kicsit containerrel szoktuk dolgozni. Ezeknek a containereknek törekszünk az erőforrás igényeiket mérsékelni, gyakran 1 CPU magon több container is osztozik. Ebben az esetben érdemes lehet az async Task-ok hívásánál `.ConfigureAwait(false)`-t használni?

Vizsgáld meg a válaszokat és válaszd ki a helyes állításokat!

## Válaszok

- ☐ Library kódban mindenképpen.
- ☐ Nagy terhelésnél ez a beállítás javíthat a performancián, mivel az async kezelt requestek bármelyik szálon folytatódhatnak, így rövidebb ideig állnak sorban, hogy visszakapják a contextusukat.
- ☐ ASP.NET Core esetében kis valószínűséggel javíthat a performancián.
- ☐ A bounded model synchronization context miatt hibát okozhat.

☐ A fentiek közül egyik sem.

## 2. feladat 2 pont

Mi a Repository Pattern fő célja az alkalmazás fejlesztése során?

### Válasz

- ☐ A Repository Pattern segít az alkalmazás fő struktúrájának és működési logikájának meghatározásában.
- ☐ A Repository Pattern olyan adatbázis-kezelési technikát nyújt, amely hatékonyan kezeli a komplex lekérdezéseket.
- ☐ A Repository Pattern segít a műveletek osztályokon keresztüli szétválasztásában és az adatbázishoz való hozzáférés elszigetelésében.
- ☐ A Repository Pattern célja a felhasználói felületek tervezése és felépítése a gyorsabb fejlesztés érdekében.

## 3. feladat 1 pont

Mi a microservices architektúra fő kihívása a szoftverfejlesztésben?

### Válasz

- ☐ A microservices architektúra megnöveli az alkalmazás teljesítményét azáltal, hogy közvetlenül összekapcsolja az összes komponenst.
- ☐ A microservices architektúra bonyolultabbá teszi a fejlesztési folyamatot a sok kis szolgáltatás kezelése miatt.
- ☐ A microservices architektúra lehetővé teszi a monolitikus alkalmazások egyszerű átalakítását anélkül, hogy változtatni kellene a kódstruktúrán.
- ☐ A microservices architektúra teljes mértékben megszünteti az alkalmazások közötti kommunikációt.

## 4. feladat 2 pont

Mi a RESTful API (Representational State Transfer) tervezés egyik alapelve az erőforrások kezelésében?

## Válaszok

- ☐ Az erőforrásokat csak POST metódussal lehet létrehozni, és csak DELETE metódussal lehet törölni.
- ☐ Az erőforrásokat különböző URL-ekkel kell azonosítani, amiket a kliens kéréseiben használhat.
- ☐ Az erőforrások kezeléséhez minden kérésnek tartalmaznia kell az összes rendelkezésre álló adatot.
- ☐ Az erőforrásokhoz való hozzáférés és módosítás az előzetesen meghatározott HTTP metódusokkal történik.
- ☐ A fentiek közül egyik sem.

## 5. feladat 3 pont

Egy webalkalmazáson dolgozol, ahol a Dependency Injection (DI) konténerben többféle élettartam-értékhatározást (lifetime scope) alkalmazol a függőségek kezelésére.

Az alábbi kód példát mutat egy egyszerű DI konfigurációra:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<IMailService, MailService>();
    services.AddTransient<IUserService, UserService>();
    services.AddSingleton<ICacheService, CacheService>();
}
```

Melyik függőségi élettartam-értékhatározás lenne legmegfelelőbb, ha egy olyan függőséget használunk, amely minden egyes request során megosztott az azonos kérési számban, de más requestek során külön példányban van szükség?

## Válasz

- ☐ `services.AddScoped<ILogger, Logger>();`
- ☐ `services.AddSingleton<ILogger, Logger>();`
- ☐ `services.AddTransient<ILogger, Logger>();`
- ☐ Más válasz

## 6. feladat 5 pont

Adott következő két osztály:

```
public class BaseClass
{
    public BaseClass(string message)
    {
        Initialize(message);
    }

    public virtual void Initialize(string message)
    {
        Console.WriteLine("Ezt az üzenetet kaptam:" + message);
    }
}

public class DerivedClass : BaseClass
{
    public DerivedClass(string message) : base(message)
    {
        Console.WriteLine($"Az üzenet a következő: {message}");
    }

    public override void Initialize(string message)
    {
        Console.WriteLine("Ez egy inicializáló metódus!");
    }
}
```

Mi lesz az alábbi kódrészletnek a kimenetele?

```
var derived = new DerivedClass("Szia uram!");
derived.Initialize("Szia főnök!");
```

## Válasz

- ☐ Szia uram! Ezt az üzenetet kaptam: Szia főnök!
- ☐ Ezt az üzenetet kaptam: Szia uram! Ez egy inicializáló metódus!
- ☐ Ez egy inicializáló metódus! Ezt az üzenetet kaptam: Szia uram!

☐ Ez egy inicializáló metódus! Az üzenet a következő: Szia uram! Ez egy inicializáló metódus!

## 7. feladat 4 pont

Lehet-e a command-nak visszatérési értéke CQS illetve CQRS tervezési minta esetén?

### Válasz

- ☐ Nem, a definíció szerint a parancsoknak nincs visszatérési értékük.
- ☐ CQRS esetében a command-nak nem lehet visszatérési értéke de a CQS bizonyos feltételek mellett megengedi.
- ☐ CQS esetében a command-nak nem lehet visszatérési értéke de a CQRS bizonyos feltételek mellett megengedi.
- ☐ Mindkét esetben definiálható visszatérési érték a végrehajtás eredményéről.

Megoldások beküldése